made with doc-o-matic

RAD Studio VCL Reference

Copyright(C) 2009 Embarcadero Technologies, Inc. All Rights Reserved.

Table of Contents

SocktCnst Namespace 1

Constants **2**

SocktCnst.ckInterceptGUID Constant 2
SocktCnst.ckPort Constant 3
SocktCnst.ckRegistered Constant 3
SocktCnst.ckShowHost Constant 3
SocktCnst.ckThreadCacheSize Constant 4
SocktCnst.ckTimeout Constant 4
SocktCnst.csSettings Constant 4
SocktCnst.KEY_IE Constant 5
SocktCnst.KEY_SOCKETSERVER Constant 5
SocktCnst.SAlreadyRunning Constant 5
SocktCnst.SApplicationName Constant 6
SocktCnst.SErrChangeSettings Constant 6
SocktCnst.SErrClose Constant 6
SocktCnst.SHostUnknown Constant 7
SocktCnst.SNotShown Constant 7
SocktCnst.SNotUntilRestart Constant 7
SocktCnst.SNoWinSock2 Constant 8
SocktCnst.SOpenError Constant 8
SocktCnst.SQueryDisconnect Constant 8
SocktCnst.SServiceName Constant 9
SocktCnst.SServiceOnly Constant 9
SocktCnst.SStatusline Constant 9

SocktComp Namespace 11

Classes **13**

ESocketError Class 13
TAbstractSocket Class 14
TAbstractSocket Members 15
TAbstractSocket Methods 15
TAbstractSocket Properties 21
TClientSocket Class 24
TClientSocket Members 25

TClientSocket.Create Constructor 26
TClientSocket.Destroy Destructor 26
TClientSocket Methods 27
TClientSocket Properties 28
TClientSocket Events 32
TClientWinSocket Class 35
TClientWinSocket Members 36
TClientWinSocket Methods 37
TClientWinSocket Properties 38
TCustomServerSocket Class 39
TCustomServerSocket Members 40
TCustomServerSocket.Destroy Destructor 41
TCustomServerSocket Fields 42
TCustomServerSocket Methods 42
TCustomServerSocket Properties 49
TCustomServerSocket Events 50
TCustomSocket Class 56
TCustomSocket Members 56
TCustomSocket Methods 57
TCustomSocket Events 59
TCustomWinSocket Class 64
TCustomWinSocket Members 64
TCustomWinSocket.Create Constructor 66
TCustomWinSocket.Destroy Destructor 67
TCustomWinSocket Methods 67
TCustomWinSocket Properties 82
TCustomWinSocket Events 89
TServerAcceptThread Class 90
TServerAcceptThread Members 91
TServerAcceptThread.Create Constructor 91
TServerAcceptThread Methods 91
TServerAcceptThread Properties 95
TServerClientThread Class 95
TServerClientThread Members 96
TServerClientThread.Create Constructor 97
TServerClientThread.Destroy Destructor 98
TServerClientThread Methods 98
TServerClientThread Properties 102
TServerClientWinSocket Class 104

TServerClientWinSocket Members 105
TServerClientWinSocket.Create Constructor 106
TServerClientWinSocket.Destroy Destructor 107
TServerClientWinSocket Properties 107
TServerSocket Class 108
TServerSocket Members 108
TServerSocket.Create Constructor 110
TServerSocket Properties 110
TServerSocket Events 114
TServerWinSocket Class 120
TServerWinSocket Members 121
TServerWinSocket.Create Constructor 123
TServerWinSocket.Destroy Destructor 123
TServerWinSocket Methods 124
TServerWinSocket Properties 129
TServerWinSocket Events 132
TWinSocketStream Class 138
TWinSocketStream Members 139
TWinSocketStream.Create Constructor 140
TWinSocketStream.Destroy Destructor 140
TWinSocketStream Methods 141
TWinSocketStream Properties 143

Functions **144**
ScktComp.SetErrorProc Function 144

Structs, Records, Enums **145**
ScktComp.TAsyncStyle Enumeration 145
ScktComp.TClientType Enumeration 146
ScktComp.TCMLookupComplete Record 146
ScktComp.TCMSocketMessage Record 147
ScktComp.TErrorEvent Enumeration 147
ScktComp.TLookupState Enumeration 148
ScktComp.TServerType Enumeration 148
ScktComp.TSocketEvent Enumeration 148

Types **150**
ScktComp.TAsyncStyles Type 150
ScktComp.TGetSocketEvent Type 150
ScktComp.TGetThreadEvent Type 151
ScktComp.TSocketErrorEvent Type 151

ScktComp.TSocketErrorProc Type 152
ScktComp.TSocketEventEvent Type 152
ScktComp.TSocketNotifyEvent Type 152
ScktComp.TThreadNotifyEvent Type 153

Constants 154
ScktComp.CM_DEFERFREE Constant 154
ScktComp.CM_LOOKUPCOMPLETE Constant 154
ScktComp.CM_SOCKETMESSAGE Constant 154

ScktMain Namespace 157

Classes 158
TSocketForm Class 158
TSocketForm Members 158
TSocketForm Fields 160
TSocketForm Methods 173
TSocketForm Properties 181
TSocketService Class 182
TSocketService Members 182
TSocketService.CreateNew Constructor 183
TSocketService Methods 184

Types 186
ScktMain.TSocketProc Type 186

Variables 187
ScktMain.SocketForm Variable 187
ScktMain.SocketService Variable 187

Constants 188
ScktMain.UI_INITIALIZE Constant 188
ScktMain.WM_MIDASICON Constant 188

SConnect Namespace 189

Classes 191
EInterpreterError Class 191
ESocketConnectionError Class 192
TCustomDataBlockInterpreter Class 192
TCustomDataBlockInterpreter Members 193
TCustomDataBlockInterpreter Methods 193

TDataBlock Class 197
TDataBlock Members 197
TDataBlock.Create Constructor 199
TDataBlock.Destroy Destructor 199
TDataBlock Methods 200
TDataBlock Properties 204
TDataBlockInterpreter Class 207
TDataBlockInterpreter Members 207
TDataBlockInterpreter.Create Constructor 208
TDataBlockInterpreter.Destroy Destructor 208
TDataBlockInterpreter Methods 209
TDataDispatch Class 215
TDataDispatch Members 215
TDataDispatch.Create Constructor 216
TDataDispatch.Destroy Destructor 217
TDataDispatch Methods 217
TDataDispatch Properties 219
TPacketInterceptFactory Class 219
TPacketInterceptFactory Members 220
TPacketInterceptFactory Methods 220
TSocketConnection Class 221
TSocketConnection Members 222
TSocketConnection.Create Constructor 223
TSocketConnection Methods 223
TSocketConnection Properties 225
TSocketTransport Class 229
TSocketTransport Members 230
TSocketTransport.Create Constructor 231
TSocketTransport.Destroy Destructor 231
TSocketTransport Methods 232
TSocketTransport Properties 234
TStreamedConnection Class 236
TStreamedConnection Members 236
TStreamedConnection.Create Constructor 237
TStreamedConnection.Destroy Destructor 238
TStreamedConnection Methods 238
TStreamedConnection Properties 245
TTransportThread Class 248
TTransportThread Members 249

TTransportThread.Create Constructor 249
TTransportThread.Destroy Destructor 249
TTransportThread Methods 250
TTransportThread Properties 254
TWebConnection Class 254
TWebConnection Members 255
TWebConnection.Create Constructor 256
TWebConnection.Destroy Destructor 257
TWebConnection Methods 257
TWebConnection Properties 261

Interfaces 266
IDataBlock Interface 266
IDataBlock Members 267
IDataBlock Methods 267
IDataBlock Properties 272
IDataIntercept Interface 274
IDataIntercept Members 275
IDataIntercept Methods 275
ISendDataBlock Interface 276
ISendDataBlock Members 277
ISendDataBlock Methods 277
ITransport Interface 278
ITransport Members 278
ITransport Methods 279
ITransport Properties 281

Functions 282
SConnect.GetPacketInterceptorList Function 282
SConnect.LoadWinSock2 Function 282

Structs, Records, Enums 283
SConnect.TVarFlag Enumeration 283

Types 284
SConnect.PIntArray Type 284
SConnect.PVariantArray Type 284
SConnect.TIntArray Type 285
SConnect.TVarFlags Type 285
SConnect.TVariantArray Type 285

Variables 286

SConnect.WSACloseEvent Variable 286
SConnect.WSACreateEvent Variable 286
SConnect.WSAEventSelect Variable 287
SConnect.WSAResetEvent Variable 287

Constants 288
SConnect.asCreateObject Constant 288
SConnect.asError Constant 288
SConnect.asFreeObject Constant 289
SConnect.asGetAppServers Constant 289
SConnect.asGetGUID Constant 289
SConnect.asGetID Constant 290
SConnect.asGetServers Constant 290
SConnect.asInvoke Constant 290
SConnect.asMask Constant 291
SConnect.asSoapCommand Constant 291
SConnect.CallSig Constant 291
SConnect.ResultSig Constant 292
SConnect.THREAD_EXCEPTION Constant 292
SConnect.THREAD_RECEIVEDSTREAM Constant 292
SConnect.THREAD_REPLACETRANSPORT Constant 293
SConnect.THREAD_SENDNOTIFY Constant 293
SConnect.THREAD_SENDSTREAM Constant 293

ScreenTips Namespace 295

Classes 297
EAllocatedActionsException Class 297
TAllocatedActions Class 297
TAllocatedActions Members 298
TAllocatedActions.Create Constructor 298
TAllocatedActions.Destroy Destructor 298
TAllocatedActions Methods 299
TAllocatedActions Properties 300
TAssociateItem Class 301
TAssociateItem Members 301
TAssociateItem.Create Constructor 301
TAssociateItem.Destroy Destructor 301
TAssociateItem Properties 302
TAssociateItemList Class 302

TAssociateItemList Members 303
TAssociateItemList.Create Constructor 303
TAssociateItemList.Destroy Destructor 303
TAssociateItemList Methods 303
TCustomScreenTipsManager Class 305
TCustomScreenTipsManager Members 305
TCustomScreenTipsManager.Create Constructor 306
TCustomScreenTipsManager.Destroy Destructor 307
TCustomScreenTipsManager Methods 307
TCustomScreenTipsManager Properties 312
TCustomScreenTipsManager Events 315
TCustomScreenTipsPopup Class 316
TCustomScreenTipsPopup Members 317
TCustomScreenTipsPopup.Create Constructor 317
TCustomScreenTipsPopup.Destroy Destructor 318
TCustomScreenTipsPopup Methods 318
TCustomScreenTipsPopup Properties 321
TDisplayOffset Class 324
TDisplayOffset Members 324
TDisplayOffset.Create Constructor 324
TDisplayOffset Properties 325
TScreenTipBorderStyle Class 326
TScreenTipBorderStyle Members 326
TScreenTipBorderStyle.Create Constructor 326
TScreenTipBorderStyle Properties 327
TScreenTipItem Class 328
TScreenTipItem Members 328
TScreenTipItem.Create Constructor 329
TScreenTipItem.Destroy Destructor 329
TScreenTipItem Methods 329
TScreenTipItem Properties 331
TScreenTipsCollection Class 334
TScreenTipsCollection Members 334
TScreenTipsCollection.Create Constructor 335
TScreenTipsCollection Methods 335
TScreenTipsCollection Properties 337
TScreenTipsManager Class 337
TScreenTipsManager Members 338
TScreenTipsManager Properties 339

TScreenTipsManager Events 341
TScreenTipsPopup Class 343
TScreenTipsPopup Members 343
TScreenTipsPopup Properties 344
TScreenTipsWindow Class 351
TScreenTipsWindow Members 351
TScreenTipsWindow.Create Constructor 352
TScreenTipsWindow.Destroy Destructor 352
TScreenTipsWindow Methods 353
TScreenTipsWindow Properties 357

Interfaces 361
IScreenTipItem Interface 361
IScreenTipItem Members 361
IScreenTipItem Methods 361

Structs, Records, Enums 363
ScreenTips.TDescriptionType Enumeration 363
ScreenTips.TImageAlign Enumeration 363
ScreenTips.TItemImageAlign Enumeration 364
ScreenTips.TScreenTipOption Enumeration 364
ScreenTips.TScreenTipPopupType Enumeration 365
ScreenTips.TTipBorderStyle Enumeration 365

Types 366
ScreenTips.TCreateScreenTipEvent Type 366
ScreenTips.TDuplicateCreateErrorEvent Type 366
ScreenTips.TScreenTipOptions Type 367
ScreenTips.TScreenTipsWindowClass Type 367
ScreenTips.TScreenTipWinClass Type 367
ScreenTips.TShowScreenTipEvent Type 368

Constants 369
ScreenTips.cDefaultBorderColor Constant 369
ScreenTips.cDefaultEndColor Constant 369
ScreenTips.cDefaultStartColor Constant 370
ScreenTips.cScreenTipHeaderSpacing Constant 370
ScreenTips.cScreenTipMinimumHeight Constant 370
ScreenTips.cScreenTipTextAndImageWidth Constant 371
ScreenTips.cScreenTipTextOnlyWidth Constant 371

SessColn Namespace 373

Classes **374**

ESessionsException Class 374

TAbstractSessionIDGen Class 374

TAbstractSessionIDGen Members 375

TAbstractSessionIDGen Methods 375

TAbstractSessionStatusChange Class 375

TAbstractSessionStatusChange Members 376

TAbstractSessionStatusChange Methods 376

TSessionIDGen Class 377

TSessionIDGen Members 377

TSessionIDGen.Create Constructor 377

TSessionIDGen Methods 378

TSessionItem Class 378

TSessionItem Members 378

TSessionItem.Create Constructor 379

TSessionItem.Destroy Destructor 379

TSessionItem Methods 380

TSessionItem Properties 380

TSessions Class 384

TSessions Members 384

TSessions.Create Constructor 385

TSessions.Destroy Destructor 385

TSessions Methods 386

TSessions Properties 394

Structs, Records, Enums **396**

SessColn.TSessionStatusFlag Enumeration 396

Types **397**

SessColn.TSessionStatusFlags Type 397

Variables **398**

SessColn.Sessions Variable 398

ShadowWnd Namespace 399

Classes **400**

TShadowWindow Class 400

TShadowWindow Members 400

TShadowWindow.Create Constructor 401
TShadowWindow.CreateShadow Constructor 401
TShadowWindow.Destroy Destructor 401
TShadowWindow Methods 402
TShadowWindow Properties 405

Structs, Records, Enums 406
ShadowWnd.TControlSide Enumeration 406

SHDocVw Namespace 407

Classes 413
CoCScriptErrorList Class 413
CoCScriptErrorList Members 413
CoCScriptErrorList Methods 414
CoInternetExplorer Class 414
CoInternetExplorer Members 415
CoInternetExplorer Methods 415
CoSearchAssistantOC Class 415
CoSearchAssistantOC Members 416
CoSearchAssistantOC Methods 416
CoShellBrowserWindow Class 417
CoShellBrowserWindow Members 417
CoShellBrowserWindow Methods 417
CoShellNameSpace Class 418
CoShellNameSpace Members 418
CoShellNameSpace Methods 418
CoShellUIHelper Class 419
CoShellUIHelper Members 419
CoShellUIHelper Methods 420
CoShellWindows Class 420
CoShellWindows Members 421
CoShellWindows Methods 421
TInternetExplorer Class 421
TInternetExplorer Members 422
TInternetExplorer.Create Constructor 425
TInternetExplorer.Destroy Destructor 425
TInternetExplorer Methods 426
TInternetExplorer Properties 448
TInternetExplorer Events 458

TShellNameSpace Class 466
TShellNameSpace Members 466
TShellNameSpace.Create Constructor 467
TShellNameSpace.Destroy Destructor 468
TShellNameSpace Methods 469
TShellNameSpace Properties 479
TShellNameSpace Events 482
TShellUIHelper Class 483
TShellUIHelper Members 484
TShellUIHelper.Create Constructor 484
TShellUIHelper.Destroy Destructor 485
TShellUIHelper Methods 486
TShellUIHelper Properties 494
TShellWindows Class 494
TShellWindows Members 495
TShellWindows.Create Constructor 495
TShellWindows.Destroy Destructor 496
TShellWindows Methods 497
TShellWindows Properties 502
TShellWindows Events 503
TWebBrowser Class 504
TWebBrowser Members 504
TWebBrowser Methods 507
TWebBrowser Properties 519
TWebBrowser Events 544
TWebBrowser_V1 Class 569
TWebBrowser_V1 Members 570
TWebBrowser_V1 Methods 571
TWebBrowser_V1 Properties 577
TWebBrowser_V1 Events 597

Interfaces 616
_SearchAssistantEvents Interface 616
_SearchAssistantEvents Members 617
_SearchAssistantEvents Methods 617
DShellNameSpaceEvents Interface 618
DShellNameSpaceEvents Members 618
DShellNameSpaceEvents Methods 618
DShellWindowsEvents Interface 619
DShellWindowsEvents Members 620

DShellWindowsEvents Methods 620
DWebBrowserEvents Interface 621
DWebBrowserEvents Members 621
DWebBrowserEvents Methods 621
DWebBrowserEvents2 Interface 626
DWebBrowserEvents2 Members 626
DWebBrowserEvents2 Methods 627
IScriptErrorList Interface 635
IScriptErrorList Members 635
IScriptErrorList Methods 636
IScriptErrorListDisp Interface 639
IScriptErrorListDisp Members 640
IScriptErrorListDisp Methods 640
ISearch Interface 644
ISearch Members 644
ISearch Methods 644
ISearch Properties 645
ISearchAssistantOC Interface 646
ISearchAssistantOC Members 646
ISearchAssistantOC Methods 647
ISearchAssistantOC Properties 654
ISearchAssistantOC2 Interface 656
ISearchAssistantOC2 Members 657
ISearchAssistantOC2 Methods 658
ISearchAssistantOC2 Properties 658
ISearchAssistantOC2Disp Interface 658
ISearchAssistantOC2Disp Members 659
ISearchAssistantOC2Disp Methods 659
ISearchAssistantOC2Disp Properties 664
ISearchAssistantOC3 Interface 666
ISearchAssistantOC3 Members 666
ISearchAssistantOC3 Methods 668
ISearchAssistantOC3 Properties 668
ISearchAssistantOC3Disp Interface 669
ISearchAssistantOC3Disp Members 669
ISearchAssistantOC3Disp Methods 670
ISearchAssistantOC3Disp Properties 674
ISearchAssistantOCDisp Interface 677
ISearchAssistantOCDisp Members 678

ISearchAssistantOCDisp Methods 678
ISearchAssistantOCDisp Properties 683
ISearchDisp Interface 685
ISearchDisp Members 685
ISearchDisp Properties 685
ISearches Interface 686
ISearches Members 686
ISearches Methods 687
ISearches Properties 688
ISearchesDisp Interface 689
ISearchesDisp Members 689
ISearchesDisp Methods 689
ISearchesDisp Properties 690
IShellFavoritesNameSpace Interface 691
IShellFavoritesNameSpace Members 691
IShellFavoritesNameSpace Methods 692
IShellFavoritesNameSpace Properties 695
IShellFavoritesNameSpaceDisp Interface 695
IShellFavoritesNameSpaceDisp Members 696
IShellFavoritesNameSpaceDisp Methods 696
IShellFavoritesNameSpaceDisp Properties 699
IShellNameSpace Interface 700
IShellNameSpace Members 700
IShellNameSpace Methods 701
IShellNameSpace Properties 706
IShellNameSpaceDisp Interface 709
IShellNameSpaceDisp Members 709
IShellNameSpaceDisp Methods 710
IShellNameSpaceDisp Properties 714
IShellUIHelper Interface 716
IShellUIHelper Members 717
IShellUIHelper Methods 717
IShellUIHelperDisp Interface 721
IShellUIHelperDisp Members 721
IShellUIHelperDisp Methods 721
IShellWindows Interface 725
IShellWindows Members 725
IShellWindows Methods 726
IShellWindows Properties 728

IShellWindowsDisp Interface 729
IShellWindowsDisp Members 729
IShellWindowsDisp Methods 730
IShellWindowsDisp Properties 732
IWebBrowser Interface 733
IWebBrowser Members 733
IWebBrowser Methods 734
IWebBrowser Properties 744
IWebBrowser2 Interface 749
IWebBrowser2 Members 750
IWebBrowser2 Methods 752
IWebBrowser2 Properties 759
IWebBrowser2Disp Interface 763
IWebBrowser2Disp Members 763
IWebBrowser2Disp Methods 764
IWebBrowser2Disp Properties 768
IWebBrowserApp Interface 776
IWebBrowserApp Members 776
IWebBrowserApp Methods 778
IWebBrowserApp Properties 782
IWebBrowserAppDisp Interface 785
IWebBrowserAppDisp Members 785
IWebBrowserAppDisp Methods 786
IWebBrowserAppDisp Properties 789
IWebBrowserDisp Interface 795
IWebBrowserDisp Members 795
IWebBrowserDisp Methods 796
IWebBrowserDisp Properties 798

Types **802**

SHDocVw.BrowserNavConstants Type 804
SHDocVw.CommandStateChangeConstants Type 804
SHDocVw.CScriptErrorList Type 804
SHDocVw.InternetExplorer Type 805
SHDocVw.OLECMDEXECOPT Type 805
SHDocVw.OLECMDF Type 806
SHDocVw.OLECMDID Type 806
SHDocVw.POleVariant1 Type 807
SHDocVw.RefreshConstants Type 807
SHDocVw.SearchAssistantOC Type 808

SHDocVw.SecureLockIconConstants Type 808
SHDocVw.ShellBrowserWindow Type 808
SHDocVw.ShellNameSpace Type 809
SHDocVw.ShellUIHelper Type 809
SHDocVw.ShellWindowFindWindowOptions Type 809
SHDocVw.ShellWindows Type 810
SHDocVw.ShellWindowTypeConstants Type 810
SHDocVw.tagREADYSTATE Type 810
SHDocVw.TInternetExplorerBeforeNavigate2 Type 811
SHDocVw.TInternetExplorerClientToHostWindow Type 811
SHDocVw.TInternetExplorerCommandStateChange Type 811
SHDocVw.TInternetExplorerDocumentComplete Type 812
SHDocVw.TInternetExplorerFileDownload Type 812
SHDocVw.TInternetExplorerNavigateComplete2 Type 812
SHDocVw.TInternetExplorerNavigateError Type 813
SHDocVw.TInternetExplorerNewWindow2 Type 813
SHDocVw.TInternetExplorerOnFullScreen Type 813
SHDocVw.TInternetExplorerOnMenuBar Type 814
SHDocVw.TInternetExplorerOnStatusBar Type 814
SHDocVw.TInternetExplorerOnTheaterMode Type 814
SHDocVw.TInternetExplorerOnToolBar Type 815
SHDocVw.TInternetExplorerOnVisible Type 815
SHDocVw.TInternetExplorerPrintTemplateInstantiation Type 815
SHDocVw.TInternetExplorerPrintTemplateTeardown Type 816
SHDocVw.TInternetExplorerPrivacyImpactedStateChange Type 816
SHDocVw.TInternetExplorerProgressChange Type 816
SHDocVw.TInternetExplorerPropertyChange Type 817
SHDocVw.TInternetExplorerSetSecureLockIcon Type 817
SHDocVw.TInternetExplorerStatusTextChange Type 817
SHDocVw.TInternetExplorerTitleChange Type 818
SHDocVw.TInternetExplorerUpdatePageStatus Type 818
SHDocVw.TInternetExplorerWindowClosing Type 818
SHDocVw.TInternetExplorerWindowSetHeight Type 819
SHDocVw.TInternetExplorerWindowSetLeft Type 819
SHDocVw.TInternetExplorerWindowSetResizable Type 819
SHDocVw.TInternetExplorerWindowSetTop Type 820
SHDocVw.TInternetExplorerWindowSetWidth Type 820
SHDocVw.TShellNameSpaceFavoritesSelectionChange Type 820
SHDocVw.TShellWindowsWindowRegistered Type 821

SHDocVw.TShellWindowsWindowRevoked Type 821
SHDocVw.TWebBrowser_V1BeforeNavigate Type 821
SHDocVw.TWebBrowser_V1CommandStateChange Type 822
SHDocVw.TWebBrowser_V1FrameBeforeNavigate Type 822
SHDocVw.TWebBrowser_V1FrameNavigateComplete Type 823
SHDocVw.TWebBrowser_V1FrameNewWindow Type 823
SHDocVw.TWebBrowser_V1NavigateComplete Type 823
SHDocVw.TWebBrowser_V1NewWindow Type 824
SHDocVw.TWebBrowser_V1ProgressChange Type 824
SHDocVw.TWebBrowser_V1PropertyChange Type 824
SHDocVw.TWebBrowser_V1Quit Type 825
SHDocVw.TWebBrowser_V1StatusTextChange Type 825
SHDocVw.TWebBrowser_V1TitleChange Type 825
SHDocVw.TWebBrowserBeforeNavigate2 Type 826
SHDocVw.TWebBrowserClientToHostWindow Type 826
SHDocVw.TWebBrowserCommandStateChange Type 827
SHDocVw.TWebBrowserDocumentComplete Type 827
SHDocVw.TWebBrowserFileDownload Type 828
SHDocVw.TWebBrowserNavigateComplete2 Type 828
SHDocVw.TWebBrowserNavigateError Type 829
SHDocVw.TWebBrowserNewWindow2 Type 829
SHDocVw.TWebBrowserOnFullScreen Type 830
SHDocVw.TWebBrowserOnMenuBar Type 830
SHDocVw.TWebBrowserOnStatusBar Type 830
SHDocVw.TWebBrowserOnTheaterMode Type 831
SHDocVw.TWebBrowserOnToolBar Type 831
SHDocVw.TWebBrowserOnVisible Type 831
SHDocVw.TWebBrowserPrintTemplateInstantiation Type 832
SHDocVw.TWebBrowserPrintTemplateTeardown Type 832
SHDocVw.TWebBrowserPrivacyImpactedStateChange Type 832
SHDocVw.TWebBrowserProgressChange Type 833
SHDocVw.TWebBrowserPropertyChange Type 833
SHDocVw.TWebBrowserSetSecureLockIcon Type 834
SHDocVw.TWebBrowserStatusTextChange Type 834
SHDocVw.TWebBrowserTitleChange Type 834
SHDocVw.TWebBrowserUpdatePageStatus Type 835
SHDocVw.TWebBrowserWindowClosing Type 835
SHDocVw.TWebBrowserWindowSetHeight Type 835
SHDocVw.TWebBrowserWindowSetLeft Type 836

SHDocVw.TWebBrowserWindowSetResizable Type 836
SHDocVw.TWebBrowserWindowSetTop Type 836
SHDocVw.TWebBrowserWindowSetWidth Type 837
SHDocVw.WebBrowser Type 837
SHDocVw.WebBrowser_V1 Type 837

Constants **838**

SHDocVw.CGID_IWebBrowser Constant 840
SHDocVw.CLASS_CScriptErrorList Constant 841
SHDocVw.CLASS_InternetExplorer Constant 841
SHDocVw.CLASS_SearchAssistantOC Constant 841
SHDocVw.CLASS_ShellBrowserWindow Constant 842
SHDocVw.CLASS_ShellNameSpace Constant 842
SHDocVw.CLASS_ShellUIHelper Constant 842
SHDocVw.CLASS_ShellWindows Constant 843
SHDocVw.CLASS_WebBrowser Constant 843
SHDocVw.CLASS_WebBrowser_V1 Constant 843
SHDocVw.CSC_NAVIGATEBACK Constant 844
SHDocVw.CSC_NAVIGATEFORWARD Constant 844
SHDocVw.CSC_UPDATECOMMANDS Constant 844
SHDocVw.DIID__SearchAssistantEvents Constant 845
SHDocVw.DIID_DShellNameSpaceEvents Constant 845
SHDocVw.DIID_DShellWindowsEvents Constant 845
SHDocVw.DIID_DWebBrowserEvents Constant 846
SHDocVw.DIID_DWebBrowserEvents2 Constant 846
SHDocVw.dtlOcxPage Constant 846
SHDocVw.dtlServerPage Constant 847
SHDocVw.HTMLID_FIND Constant 847
SHDocVw.HTMLID_OPTIONS Constant 847
SHDocVw.HTMLID_VIEWSOURCE Constant 848
SHDocVw.IID_IScriptErrorList Constant 848
SHDocVw.IID_ISearch Constant 848
SHDocVw.IID_ISearchAssistantOC Constant 849
SHDocVw.IID_ISearchAssistantOC2 Constant 849
SHDocVw.IID_ISearchAssistantOC3 Constant 849
SHDocVw.IID_ISearches Constant 850
SHDocVw.IID_IShellFavoritesNameSpace Constant 850
SHDocVw.IID_IShellNameSpace Constant 850
SHDocVw.IID_IShellUIHelper Constant 851
SHDocVw.IID_IShellWindows Constant 851

SHDocVw.IID_IWebBrowser Constant 851
SHDocVw.IID_IWebBrowser2 Constant 852
SHDocVw.IID_IWebBrowserApp Constant 852
SHDocVw.LIBID_SHDocVw Constant 852
SHDocVw.navAllowAutosearch Constant 853
SHDocVw.navBrowserBar Constant 853
SHDocVw.navNoHistory Constant 853
SHDocVw.navNoReadFromCache Constant 854
SHDocVw.navNoWriteToCache Constant 854
SHDocVw.navOpenInNewWindow Constant 854
SHDocVw.OLECMDEXECOPT_DODEFAULT Constant 855
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER Constant 855
SHDocVw.OLECMDEXECOPT_PROMPTUSER Constant 855
SHDocVw.OLECMDEXECOPT_SHOWHELP Constant 856
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU Constant 856
SHDocVw.OLECMDF_ENABLED Constant 856
SHDocVw.OLECMDF_INVISIBLE Constant 857
SHDocVw.OLECMDF_LATCHED Constant 857
SHDocVw.OLECMDF_NINCHED Constant 857
SHDocVw.OLECMDF_SUPPORTED Constant 858
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS Constant 858
SHDocVw.OLECMDID_CLEARSELECTION Constant 858
SHDocVw.OLECMDID_CLOSE Constant 859
SHDocVw.OLECMDID_COPY Constant 859
SHDocVw.OLECMDID_CUT Constant 859
SHDocVw.OLECMDID_DELETE Constant 860
SHDocVw.OLECMDID_DONTDOWNLOADCSS Constant 860
SHDocVw.OLECMDID_ENABLE_INTERACTION Constant 860
SHDocVw.OLECMDID_FIND Constant 861
SHDocVw.OLECMDID_GETPRINTTEMPLATE Constant 861
SHDocVw.OLECMDID_GETZOOMRANGE Constant 861
SHDocVw.OLECMDID_HIDETOOLBARS Constant 862
SHDocVw.OLECMDID_HTTPEQUIV Constant 862
SHDocVw.OLECMDID_HTTPEQUIV_DONE Constant 862
SHDocVw.OLECMDID_NEW Constant 863
SHDocVw.OLECMDID_ONTOOLBARACTIVATED Constant 863
SHDocVw.OLECMDID_ONUNLOAD Constant 863
SHDocVw.OLECMDID_OPEN Constant 864
SHDocVw.OLECMDID_PAGESETUP Constant 864

SHDocVw.OLECMDID_PASTE Constant 864
SHDocVw.OLECMDID_PASTESPECIAL Constant 865
SHDocVw.OLECMDID_PREREFRESH Constant 865
SHDocVw.OLECMDID_PRINT Constant 865
SHDocVw.OLECMDID_PRINT2 Constant 866
SHDocVw.OLECMDID_PRINTPREVIEW Constant 866
SHDocVw.OLECMDID_PRINTPREVIEW2 Constant 866
SHDocVw.OLECMDID_PROPERTIES Constant 867
SHDocVw.OLECMDID_PROPERTYBAG2 Constant 867
SHDocVw.OLECMDID_REDO Constant 867
SHDocVw.OLECMDID_REFRESH Constant 868
SHDocVw.OLECMDID_SAVE Constant 868
SHDocVw.OLECMDID_SAVEAS Constant 868
SHDocVw.OLECMDID_SAVECOPYAS Constant 869
SHDocVw.OLECMDID_SELECTALL Constant 869
SHDocVw.OLECMDID_SETDOWNLOADSTATE Constant 869
SHDocVw.OLECMDID_SETPRINTTEMPLATE Constant 870
SHDocVw.OLECMDID_SETPROGRESSMAX Constant 870
SHDocVw.OLECMDID_SETPROGRESSPOS Constant 870
SHDocVw.OLECMDID_SETPROGRESSTEXT Constant 871
SHDocVw.OLECMDID_SETTITLE Constant 871
SHDocVw.OLECMDID_SHOWFIND Constant 871
SHDocVw.OLECMDID_SHOWMESSAGE Constant 872
SHDocVw.OLECMDID_SHOWPAGESETUP Constant 872
SHDocVw.OLECMDID_SHOWPRINT Constant 872
SHDocVw.OLECMDID_SHOWSCRIPTERROR Constant 873
SHDocVw.OLECMDID_SPELL Constant 873
SHDocVw.OLECMDID_STOP Constant 873
SHDocVw.OLECMDID_STOPDOWNLOAD Constant 874
SHDocVw.OLECMDID_UNDO Constant 874
SHDocVw.OLECMDID_UPDATECOMMANDS Constant 874
SHDocVw.OLECMDID_UPDATEPAGESTATUS Constant 875
SHDocVw.OLECMDID_ZOOM Constant 875
SHDocVw.READYSTATE_COMPLETE Constant 875
SHDocVw.READYSTATE_INTERACTIVE Constant 876
SHDocVw.READYSTATE_LOADED Constant 876
SHDocVw.READYSTATE_LOADING Constant 876
SHDocVw.READYSTATE_UNINITIALIZED Constant 877
SHDocVw.REFRESH_COMPLETELY Constant 877

SHDocVw.REFRESH_CONTINUE Constant 877
SHDocVw.REFRESH_IFEXPIRED Constant 878
SHDocVw.REFRESH_NORMAL Constant 878
SHDocVw.secureLockIconMixed Constant 878
SHDocVw.secureLockIconSecure128Bit Constant 879
SHDocVw.secureLockIconSecure40Bit Constant 879
SHDocVw.secureLockIconSecure56Bit Constant 879
SHDocVw.secureLockIconSecureFortezza Constant 880
SHDocVw.secureLockIconSecureUnknownBits Constant 880
SHDocVw.secureLockIconUnsecure Constant 880
SHDocVw.SHDocVwMajorVersion Constant 881
SHDocVw.SHDocVwMinorVersion Constant 881
SHDocVw.SWC_3RDPARTY Constant 881
SHDocVw.SWC_BROWSER Constant 882
SHDocVw.SWC_CALLBACK Constant 882
SHDocVw.SWC_EXPLORER Constant 882
SHDocVw.SWFO_COOKIEPASSED Constant 883
SHDocVw.SWFO_INCLUDEPENDING Constant 883
SHDocVw.SWFO_NEEDDISPATCH Constant 883

ShellAnimations Namespace 885

Classes **886**
TShellResources Class 886

SimpleDS Namespace 887

Classes **888**
TInternalSQLDataSet Class 888
TInternalSQLDataSet Members 888
TInternalSQLDataSet Properties 891
TSimpleDataSet Class 896
TSimpleDataSet Members 897
TSimpleDataSet.Create Constructor 899
TSimpleDataSet.Destroy Destructor 900
TSimpleDataSet Methods 900
TSimpleDataSet Properties 903
TSimpleDataSet Events 930

SiteComp Namespace 959

Classes **963**
TAbstractDesigntimeWarnings Class 963
TAbstractDesigntimeWarnings Members 963
TAbstractDesigntimeWarnings Methods 964
TAbstractInetFileType Class 965
TAbstractInetFileType Members 965
TAbstractInetFileType Methods 966
TAbstractInetFileType Properties 967
TNotifyList Class 968
TNotifyList Members 968
TNotifyList.Create Constructor 969
TNotifyList.Destroy Destructor 970
TNotifyList Methods 970
TNotifyList Properties 972
TScriptComponent Class 973
TScriptComponent Members 973
TScriptComponent.Create Constructor 975
TScriptComponent Methods 975
TScriptComponent Properties 975
TScriptObject Class 976

Interfaces **977**
IActionImageProducer Interface 978
IActionImageProducer Members 979
IActionImageProducer Methods 979
IAdapterAccess Interface 981
IAdapterAccess Members 981
IAdapterAccess Methods 981
IAdapterActionAccess Interface 982
IAdapterActionAccess Members 983
IAdapterActionAccess Methods 983
IAdapterActionAttributes Interface 983
IAdapterActionAttributes Members 984
IAdapterActionAttributes Methods 984
IAdapterActionAttributes Properties 984
IAdapterDispatcher Interface 985
IAdapterDispatcher Members 985

IAdapterDispatcher Methods 986
IAdapterDispatchParams Interface 987
IAdapterDispatchParams Members 988
IAdapterDispatchParams Methods 988
IAdapterDispatchParams Properties 990
IAdapterError Interface 992
IAdapterError Members 992
IAdapterError Methods 992
IAdapterError Properties 993
IAdapterErrors Interface 995
IAdapterErrors Members 995
IAdapterErrors Methods 995
IAdapterErrors Properties 997
IAdapterFieldAccess Interface 998
IAdapterFieldAccess Members 999
IAdapterFieldAccess Methods 999
IAdapterFieldAttributes Interface 999
IAdapterFieldAttributes Members 1000
IAdapterFieldAttributes Methods 1000
IAdapterFieldAttributes Properties 1001
IAdapterHiddenField Interface 1001
IAdapterHiddenField Members 1002
IAdapterHiddenField Methods 1002
IAdapterHiddenField Properties 1003
IAdapterHiddenFields Interface 1003
IAdapterHiddenFields Members 1004
IAdapterHiddenFields Methods 1004
IAdapterHiddenFields Properties 1004
IAdapterItemRequestParams Interface 1005
IAdapterItemRequestParams Members 1006
IAdapterItemRequestParams Methods 1006
IAdapterItemRequestParams Properties 1008
IAdapterMode Interface 1009
IAdapterMode Members 1010
IAdapterMode Methods 1010
IAdapterMode Properties 1011
IAdapterRequestHandler Interface 1012
IAdapterRequestHandler Members 1012
IAdapterRequestHandler Methods 1012

IAsFormatted Interface 1013
IAsFormatted Members 1014
IAsFormatted Methods 1014
IDefaultPageFileName Interface 1014
IDefaultPageFileName Members 1015
IDefaultPageFileName Methods 1015
IFieldValuesAdapter Interface 1015
IFieldValuesAdapter Members 1016
IFieldValuesAdapter Methods 1016
IFindIncludeFile Interface 1016
IFindIncludeFile Members 1017
IFindIncludeFile Methods 1017
IGetAdapterActions Interface 1017
IGetAdapterActions Members 1018
IGetAdapterActions Methods 1018
IGetAdapterDisplayCharacteristics Interface 1018
IGetAdapterDisplayCharacteristics Members 1019
IGetAdapterDisplayCharacteristics Methods 1019
IGetAdapterErrors Interface 1019
IGetAdapterErrors Members 1020
IGetAdapterErrors Methods 1020
IGetAdapterFields Interface 1020
IGetAdapterFields Members 1021
IGetAdapterFields Methods 1021
IGetAdapterHiddenFields Interface 1021
IGetAdapterHiddenFields Members 1021
IGetAdapterHiddenFields Methods 1022
IGetAdapterImage Interface 1022
IGetAdapterImage Members 1023
IGetAdapterImage Methods 1023
IGetAdapterItemRequestParams Interface 1023
IGetAdapterItemRequestParams Members 1024
IGetAdapterItemRequestParams Methods 1024
IGetAdapterRequestDefaultResponse Interface 1024
IGetAdapterRequestDefaultResponse Members 1025
IGetAdapterRequestDefaultResponse Methods 1025
IGetAdapterRequestDefaultResponse Properties 1026
IGetAdapterValuesList Interface 1026
IGetAdapterValuesList Members 1027

IGetAdapterValuesList Methods 1027
IGetDefaultAction Interface 1027
IGetDefaultAction Members 1027
IGetDefaultAction Methods 1028
IGetDefaultAction Properties 1028
IGetDesigntimeWarnings Interface 1029
IGetDesigntimeWarnings Members 1029
IGetDesigntimeWarnings Methods 1029
IGetFieldValuesAdapter Interface 1030
IGetFieldValuesAdapter Members 1030
IGetFieldValuesAdapter Methods 1030
IGetHTMLStyle Interface 1031
IGetHTMLStyle Members 1031
IGetHTMLStyle Methods 1032
IGetProducerComponent Interface 1034
IGetProducerComponent Members 1034
IGetProducerComponent Methods 1034
IGetScriptObject Interface 1035
IGetScriptObject Members 1035
IGetScriptObject Methods 1035
IGetScriptObjectClassKey Interface 1036
IGetScriptObjectClassKey Members 1036
IGetScriptObjectClassKey Methods 1036
IGetWebAppComponents Interface 1037
IGetWebAppComponents Members 1037
IGetWebAppComponents Methods 1038
IGetWebAppComponents Properties 1039
IIDEFileManager Interface 1040
IIDEFileManager Members 1040
IIDEFileManager Methods 1040
IImageProducer Interface 1041
IImageProducer Members 1042
IImageProducer Methods 1042
IInternetEnvOptions Interface 1042
IInternetEnvOptions Members 1043
IInternetEnvOptions Methods 1043
IInternetEnvOptions Properties 1044
IInternetExecuteScript Interface 1045
IInternetExecuteScript Members 1046

IInternetExecuteScript Methods 1046
IInternetFileTypes Interface 1047
IInternetFileTypes Members 1047
IInternetFileTypes Methods 1047
IInternetScriptDebugger Interface 1048
IInternetScriptDebugger Members 1048
IInternetScriptDebugger Methods 1048
IIsAdapterActionList Interface 1049
IIsAdapterActionList Members 1049
IIsAdapterActionList Methods 1049
IIsAdapterActionList Properties 1050
IIterateIntfSupport Interface 1050
IIterateIntfSupport Members 1051
IIterateIntfSupport Methods 1051
IIterateObjectSupport Interface 1051
IIterateObjectSupport Members 1052
IIterateObjectSupport Methods 1052
IIteratorIndex Interface 1052
IIteratorIndex Members 1053
IIteratorIndex Methods 1053
IIteratorIndex Properties 1054
IIteratorSupport Interface 1054
IIteratorSupport Members 1055
IIteratorSupport Methods 1055
INotifyAdapterChange Interface 1057
INotifyAdapterChange Members 1057
INotifyAdapterChange Methods 1058
INotifyList Interface 1058
INotifyList Members 1059
INotifyList Methods 1059
IPageDispatcher Interface 1060
IPageDispatcher Members 1060
IPageDispatcher Methods 1061
IPageDispatcher Properties 1064
IPageResult Interface 1064
IPageResult Members 1065
IPageResult Methods 1065
ISessionsService Interface 1066
ISessionsService Members 1067

ISessionsService Methods 1067
ISessionsServiceConnectID Interface 1068
ISessionsServiceConnectID Members 1069
ISessionsServiceConnectID Methods 1069
ISessionsServicePersist Interface 1070
ISessionsServicePersist Members 1070
ISessionsServicePersist Methods 1070
ISessionsServiceStoreID Interface 1071
ISessionsServiceStoreID Members 1072
ISessionsServiceStoreID Methods 1072
IValuesListAdapter Interface 1072
IValuesListAdapter Members 1073
IValuesListAdapter Methods 1074
IWebActionsList Interface 1076
IWebActionsList Members 1076
IWebActionsList Methods 1077
IWebActionsList Properties 1078
IWebApplicationInfo Interface 1078
IWebApplicationInfo Members 1079
IWebApplicationInfo Methods 1079
IWebApplicationInfo Properties 1080
IWebContainerSubComponent Interface 1081
IWebDataFields Interface 1081
IWebDataFields Members 1082
IWebDataFields Methods 1082
IWebDataFields Properties 1083
IWebEnabled Interface 1083
IWebEnabled Members 1084
IWebEnabled Methods 1084
IWebFormComponentContainer Interface 1084
IWebFormComponentContainer Members 1085
IWebFormComponentContainer Methods 1085
IWebFormComponentContainer Properties 1086
IWebGetActionName Interface 1086
IWebGetActionName Members 1087
IWebGetActionName Methods 1087
IWebGetActionName Properties 1087
IWebGetDisplayLabel Interface 1088
IWebGetDisplayLabel Members 1088

IWebGetDisplayLabel Methods 1088
IWebGetDisplayWidth Interface 1089
IWebGetDisplayWidth Members 1089
IWebGetDisplayWidth Methods 1089
IWebGetFieldName Interface 1090
IWebGetFieldName Members 1090
IWebGetFieldName Methods 1090
IWebGetFieldName Properties 1091
IWebGetFieldValue Interface 1091
IWebGetFieldValue Members 1091
IWebGetFieldValue Methods 1092
IWebGetFieldValue Properties 1092
IWebGetFieldValues Interface 1092
IWebGetFieldValues Members 1093
IWebGetFieldValues Methods 1093
IWebGetMaxLength Interface 1095
IWebGetMaxLength Members 1095
IWebGetMaxLength Methods 1096
IWebGetSessionIDs Interface 1096
IWebGetSessionIDs Members 1096
IWebGetSessionIDs Methods 1097
IWebImageHREF Interface 1097
IWebImageHREF Members 1098
IWebImageHREF Methods 1098
IWebInputName Interface 1098
IWebInputName Members 1099
IWebInputName Methods 1099
IWebInputName Properties 1099
IWebIsDefaultAction Interface 1100
IWebIsDefaultAction Members 1100
IWebIsDefaultAction Methods 1100
IWebIsDefaultField Interface 1101
IWebIsDefaultField Members 1101
IWebIsDefaultField Methods 1101
IWebSession Interface 1102
IWebSession Members 1102
IWebSession Methods 1103
IWebSessionAttributes Interface 1103
IWebSessionAttributes Members 1104

IWebSessionAttributes Methods 1104
IWebSessionAttributes Properties 1106
IWebSessionID Interface 1108
IWebSessionID Members 1108
IWebSessionID Methods 1108
IWebSessionID Properties 1109
IWebSessionIDs Interface 1109
IWebSessionIDs Members 1110
IWebSessionIDs Methods 1110
IWebSessionIDs Properties 1111
IWebSessionValues Interface 1112
IWebSessionValues Members 1112
IWebSessionValues Methods 1112
IWebSessionValues Properties 1113
IWebSetActionName Interface 1114
IWebSetActionName Members 1114
IWebSetActionName Methods 1114
IWebSetFieldName Interface 1115
IWebSetFieldName Members 1115
IWebSetFieldName Methods 1115
IWebUserList Interface 1116
IWebUserList Members 1116
IWebUserList Methods 1117
IWebValueIsEqual Interface 1118
IWebValueIsEqual Members 1118
IWebValueIsEqual Methods 1118

Functions **1120**

SiteComp.FullyQualifiedFieldName Function 1120
SiteComp.FullyQualifiedName Function 1120

Structs, Records, Enums **1121**

SiteComp.TAdapterActionHTMLElementType Enumeration 1121
SiteComp.TAdapterDisplayCharacteristic Enumeration 1122
SiteComp.TAdapterDisplayHTMLElementType Enumeration 1123
SiteComp.TAdapterDisplayViewModeType Enumeration 1123
SiteComp.TAdapterInputHTMLElementType Enumeration 1124
SiteComp.TAdapterItemRequestParamOption Enumeration 1125
SiteComp.TDispatchPageFlag Enumeration 1126
SiteComp.TQualifyOption Enumeration 1126

Types **1128**
SiteComp.TAdapterDisplayCharacteristics Type 1128
SiteComp.TAdapterItemRequestParamOptions Type 1128
SiteComp.TDispatchPageFlags Type 1129
SiteComp.TSessionID Type 1129

Variables **1130**
SiteComp.ExecuteScript Variable 1130
SiteComp.GetHTMLSampleImage Variable 1130
SiteComp.IDEFileManager Variable 1131
SiteComp.InetFileTypes Variable 1131
SiteComp.InternetEnvOptions Variable 1131
SiteComp.ScriptDebugger Variable 1132

Constants **1133**
SiteComp.AdapterActionHTMLElementTypeNames Constant 1133
SiteComp.AdapterDisplayHTMLElementTypeNames Constant 1133
SiteComp.AdapterDisplayViewModeTypeNames Constant 1134
SiteComp.AdapterInputHTMLElementTypeNames Constant 1134

SiteProd Namespace 1135

Classes **1136**
TAdapterPageProducer Class 1136
TAdapterPageProducer Members 1137
TAdapterPageProducer Properties 1138
TAdapterPageProducer Events 1142
TBaseAdapterPageProducer Class 1143
TBaseAdapterPageProducer Members 1144
TBaseAdapterPageProducer.Create Constructor 1145
TBaseAdapterPageProducer.Destroy Destructor 1145
TBaseAdapterPageProducer Methods 1145
TBaseAdapterPageProducer Properties 1156
TBaseAdapterPageProducer Events 1158
TCustomAdapterPageProducer Class 1159
TCustomAdapterPageProducer Members 1160
TCustomAdapterPageProducer.Create Constructor 1161
TCustomAdapterPageProducer.Destroy Destructor 1162
TCustomAdapterPageProducer Methods 1162
TCustomAdapterPageProducer Properties 1165

TSitePageElements Class 1167
TSitePageElements Members 1167
TSitePageElements Fields 1168

SOAPAttachIntf Namespace 1171

Classes **1172**
TSoapDataList Class 1172
TSoapDataList Members 1172
TSoapDataList Methods 1173

Interfaces **1176**
IMimeAttachmentHandler Interface 1176
IMimeAttachmentHandler Members 1176
IMimeAttachmentHandler Methods 1177
IMimeAttachmentHandler Properties 1182
IMimeAttachmentHandler Events 1183

Types **1185**
SOAPAttachIntf.TOnGetAttachmentEvent Type 1185
SOAPAttachIntf.TOnSendAttachmentEvent Type 1185

SOAPConn Namespace 1187

Classes **1188**
TSoapConnection Class 1188
TSoapConnection Members 1189
TSoapConnection.Create Constructor 1190
TSoapConnection.Destroy Destructor 1191
TSoapConnection Methods 1191
TSoapConnection Properties 1195
TSoapConnection Events 1201

SOAPDm Namespace 1205

Classes **1206**
TSoapDataModule Class 1206
TSoapDataModule Members 1207
TSoapDataModule.Create Constructor 1208
TSoapDataModule.Destroy Destructor 1208
TSoapDataModule Methods 1209

TSoapDataModule Properties 1230

SOAPDomConv Namespace 1233

Classes **1234**

TSOAPDOMProcessor Class 1234

TSOAPDOMProcessor Members 1234

TSOAPDOMProcessor.Destroy Destructor 1235

TSOAPDOMProcessor Fields 1235

TSOAPDOMProcessor Methods 1236

Interfaces **1241**

IDOMHeaderProcessor Interface 1241

IDOMHeaderProcessor Members 1242

IDOMHeaderProcessor Methods 1242

Structs, Records, Enums **1244**

SOAPDomConv.TDOMHeaderProcessorEntry Record 1244

Types **1245**

SOAPDomConv.TDOMHeaderProcessorArray Type 1245

SOAPHTTPClient Namespace 1247

Classes **1248**

THTTPRIO Class 1248

THTTPRIO Members 1249

THTTPRIO.Create Constructor 1249

THTTPRIO.Destroy Destructor 1250

THTTPRIO Methods 1250

THTTPRIO Properties 1252

SOAPHTTPDisp Namespace 1257

Classes **1258**

THTTPSoapDispatchNode Class 1258

THTTPSoapDispatchNode Members 1258

THTTPSoapDispatchNode Fields 1259

THTTPSoapDispatchNode Methods 1259

THTTPSoapDispatchNode Properties 1260

Interfaces **1262**

IHTTPSoapDispatch Interface 1262

IHTTPSoapDispatch Members 1262
IHTTPSoapDispatch Methods 1263

SOAPHTTPPasInv Namespace 1265

Classes 1266
THTTPSoapCppInvoker Class 1266
THTTPSoapPascalInvoker Class 1267
THTTPSoapPascalInvoker Members 1268
THTTPSoapPascalInvoker Methods 1269

SOAPHTTPTrans Namespace 1271

Classes 1272
ESOAPHTTPException Class 1272
ESOAPHTTPException Members 1272
ESOAPHTTPException.Create Constructor 1274
ESOAPHTTPException.CreateFmt Constructor 1274
ESOAPHTTPException Properties 1275
THTTPReqResp Class 1276
THTTPReqResp Members 1277
THTTPReqResp.Create Constructor 1278
THTTPReqResp.Destroy Destructor 1278
THTTPReqResp Methods 1279
THTTPReqResp Properties 1287
THTTPReqResp Events 1294

Interfaces 1296
IHTTPReqResp Interface 1296
IHTTPReqResp Members 1296
IHTTPReqResp Methods 1296

Structs, Records, Enums 1298
SOAPHTTPTrans.SOAPInvokeOptions Enumeration 1298

Types 1299
SOAPHTTPTrans.TBeforePostEvent Type 1299
SOAPHTTPTrans.TPostingDataEvent Type 1299
SOAPHTTPTrans.TReceivingDataEvent Type 1300
SOAPHTTPTrans.TSOAPInvokeOptions Type 1300
SOAPHTTPTrans.TWinInetErrorEvent Type 1301

SOAPLinked Namespace 1303

Classes **1305**
ECustomException Class 1305
ECustomException Members 1305
ECustomException.Create Constructor 1307
ECustomException.Destroy Destructor 1309
ECustomException Methods 1310
TFTCompT Class 1311
TFTCompT Members 1311
TFTCompT Methods 1311
TFTCompT Events 1312
TFTCompTI Class 1313
TFTCompTI Members 1313
TFTCompTI Methods 1315
TLinkedRIO Class 1316
TLinkedRIO Members 1317
TLinkedRIO.Create Constructor 1317
TLinkedRIO.CreateFile Constructor 1318
TLinkedRIO.Destroy Destructor 1318
TLinkedRIO Methods 1319
TLinkedRIO Properties 1320
TLinkedWebNode Class 1321
TLinkedWebNode Members 1321
TLinkedWebNode.Create Constructor 1322
TLinkedWebNode.Destroy Destructor 1322
TLinkedWebNode Methods 1322
TLinkedWebNode Properties 1325
TLogLinkedWebNode Class 1326
TLogLinkedWebNode Members 1327
TLogLinkedWebNode.Create Constructor 1327
TLogLinkedWebNode.Destroy Destructor 1328
TLogLinkedWebNode Methods 1328
TLogLinkedWebNode Properties 1329
TWinHandleUser Class 1329
TWinHandleUser Members 1330
TWinHandleUser Methods 1331

Interfaces **1332**

IIntegerTest Interface 1332
IIntegerTest Members 1332
IIntegerTest Methods 1332
IObjectGetter Interface 1333
IObjectGetter Members 1333
IObjectGetter Methods 1334
IStringTest Interface 1334
IStringTest Members 1334
IStringTest Methods 1335

Structs, Records, Enums 1336
SOAPLinked.EventKind Enumeration 1336

Types 1337
SOAPLinked.TCStringEvent Type 1337
SOAPLinked.TCustExCallback Type 1337
SOAPLinked.TStringEvent Type 1338
SOAPLinked.TVStringEvent Type 1338

SOAPMidas Namespace 1339

Interfaces 1340
IAppServerSOAP Interface 1340
IAppServerSOAP Members 1340
IAppServerSOAP Methods 1341

Functions 1347
SOAPMidas.RegDefIAppServerInvClass Function 1347

SOAPPasInv Namespace 1349

Classes 1350
TSoapPascalInvoker Class 1350
TSoapPascalInvoker Members 1350
TSoapPascalInvoker.Create Constructor 1351
TSoapPascalInvoker.Destroy Destructor 1352
TSoapPascalInvoker Fields 1352
TSoapPascalInvoker Methods 1353
TSoapPascalInvoker Properties 1355
TSoapPascalInvoker Events 1356

Interfaces 1359

IAdapterConverter Interface 1359
IAdapterConverter Members 1359
IAdapterConverter Methods 1359
IInvokerConverter Interface 1360
IInvokerConverter Members 1360
IInvokerConverter Methods 1360

Functions **1362**
SOAPPasInv.GetSOAPHeaders Function 1362

Types **1363**
SOAPPasInv.TAfterDispatchEvent Type 1363
SOAPPasInv.TBeforeDispatchEvent Type 1363
SOAPPasInv.TBeforeDispatchEvent2 Type 1364
SOAPPasInv.TOnExceptionEvent Type 1364

Sockets Namespace 1367

Classes **1369**
ESocketError Class 1369
TBaseSocket Class 1370
TBaseSocket Members 1370
TBaseSocket.Create Constructor 1371
TBaseSocket.Destroy Destructor 1372
TBaseSocket Methods 1372
TBaseSocket Properties 1383
TBaseSocket Events 1388
TClientSocketThread Class 1396
TClientSocketThread Members 1396
TClientSocketThread.Create Constructor 1397
TClientSocketThread.Destroy Destructor 1397
TClientSocketThread Methods 1397
TClientSocketThread Properties 1399
TCustomIpClient Class 1400
TCustomIpClient Members 1400
TCustomIpClient.Create Constructor 1402
TCustomIpClient Methods 1402
TCustomIpClient Properties 1405
TCustomIpClient Events 1405
TCustomTcpServer Class 1406
TCustomTcpServer Members 1407

TCustomTcpServer.Create Constructor 1408
TCustomTcpServer.Destroy Destructor 1408
TCustomTcpServer Methods 1409
TCustomTcpServer Properties 1411
TCustomTcpServer Events 1412
TIpSocket Class 1416
TIpSocket Members 1417
TIpSocket.Create Constructor 1418
TIpSocket Methods 1418
TIpSocket Properties 1423
TRawSocket Class 1431
TRawSocket Members 1431
TRawSocket.Create Constructor 1432
TRawSocket Properties 1433
TServerSocketThread Class 1433
TServerSocketThread Members 1434
TServerSocketThread.Create Constructor 1434
TServerSocketThread.Destroy Destructor 1435
TServerSocketThread Methods 1435
TServerSocketThread Properties 1437
TServerSocketThread Events 1439
TTcpClient Class 1439
TTcpClient Members 1441
TTcpClient Properties 1443
TTcpClient Events 1450
TTcpServer Class 1459
TTcpServer Members 1461
TTcpServer Properties 1463
TTcpServer Events 1469
TUdpSocket Class 1477
TUdpSocket Members 1478
TUdpSocket.Create Constructor 1479
TUdpSocket Properties 1480
TUdpSocket Events 1490

Structs, Records, Enums 1499

Sockets.TIPHeader Record 1499
Sockets.TServerSocketBlockMode Enumeration 1500
Sockets.TSocketDomain Enumeration 1500
Sockets.TSocketType Enumeration 1502

Types **1504**
Sockets.TGetThreadEvent Type 1504
Sockets.TSocketAcceptEvent Type 1504
Sockets.TSocketBlockMode Type 1505
Sockets.TSocketDataEvent Type 1505
Sockets.TSocketErrorEvent Type 1505
Sockets.TSocketHost Type 1506
Sockets.TSocketNotifyEvent Type 1506
Sockets.TSocketPort Type 1506
Sockets.TSocketProtocol Type 1507

Constants **1508**
Sockets.CRLF Constant 1508

SqlExpr Namespace 1509

Classes **1512**
SPParamDesc Class 1512
SPParamDesc Members 1513
SPParamDesc Fields 1513
TConnectionData Class 1515
TConnectionData Members 1515
Create Constructor 1516
TConnectionData.Destroy Destructor 1517
TConnectionData Methods 1517
TConnectionData Properties 1519
TCustomSQLDataSet Class 1520
TCustomSQLDataSet Members 1521
Create Constructor 1524
TCustomSQLDataSet.Destroy Destructor 1525
TCustomSQLDataSet Methods 1525
TCustomSQLDataSet Properties 1551
TCustomSQLDataSet Events 1567
TFLDDesc Class 1570
TFLDDesc Members 1571
TFLDDesc Fields 1571
TFLDDesc Properties 1574
TSQLBlobStream Class 1577
TSQLBlobStream Members 1578
TSQLBlobStream.Create Constructor 1578

TSQLBlobStream.Destroy Destructor 1579
TSQLBlobStream Methods 1579
TSQLConnection Class 1580
TSQLConnection Members 1581
TSQLConnection.Create Constructor 1583
TSQLConnection.Destroy Destructor 1584
TSQLConnection Methods 1584
TSQLConnection Properties 1607
TSQLConnection Events 1625
TSQLDataLink Class 1628
TSQLDataLink Members 1629
TSQLDataLink.Create Constructor 1629
TSQLDataLink Methods 1630
TSQLDataSet Class 1631
TSQLDataSet Members 1633
TSQLDataSet.Create Constructor 1635
TSQLDataSet Methods 1636
TSQLDataSet Properties 1637
TSQLMonitor Class 1643
TSQLMonitor Members 1644
TSQLMonitor.Create Constructor 1645
TSQLMonitor.Destroy Destructor 1645
TSQLMonitor Methods 1645
TSQLMonitor Properties 1649
TSQLMonitor Events 1652
TSQLQuery Class 1654
TSQLQuery Members 1654
TSQLQuery.Create Constructor 1657
TSQLQuery.Destroy Destructor 1658
TSQLQuery Methods 1658
TSQLQuery Properties 1659
TSqlServerMethod Class 1665
TSqlServerMethod Members 1666
TSqlServerMethod.Create Constructor 1668
TSqlServerMethod Methods 1669
TSqlServerMethod Properties 1669
TSQLStoredProc Class 1671
TSQLStoredProc Members 1671
TSQLStoredProc.Create Constructor 1674

TSQLStoredProc Methods 1675
TSQLStoredProc Properties 1676
TSQLTable Class 1680
TSQLTable Members 1681
TSQLTable.Create Constructor 1684
TSQLTable.Destroy Destructor 1685
TSQLTable Methods 1685
TSQLTable Properties 1687
TTransactionItem Class 1694
TTransactionItem Members 1694
TTransactionItem Fields 1694

Functions **1696**
SqlExpr.FreeProcParams Function 1696
SqlExpr.GetConnectionNames Function 1696
SqlExpr.GetConnectionRegistryFile Function 1697
SqlExpr.GetDelegateDriverNames Function 1697
SqlExpr.GetDriverNames Function 1697
SqlExpr.GetDriverRegistryFile Function 1698
SqlExpr.LoadParamListItems Function 1698

Structs, Records, Enums **1699**
SqlExpr.EConnectFlag Enumeration 1699
SqlExpr.TConnectionState Enumeration 1700
SqlExpr.TConnectionUserType Enumeration 1700
SqlExpr.TSchemaType Enumeration 1701
SqlExpr.TSQLExceptionType Enumeration 1702
SqlExpr.TSQLSchemaInfo Record 1703
SqlExpr.TTableScope Enumeration 1703
SqlExpr.TTransactionDesc Record 1704
SqlExpr.TTransIsolationLevel Enumeration 1704

Types **1706**
SqlExpr.SQLResult Type 1706
SqlExpr.TFieldDescList Type 1707
SqlExpr.TFieldList Type 1707
SqlExpr.TGetDriverFunc Type 1708
SqlExpr.TLocale Type 1708
SqlExpr.TLocaleCode Type 1708
SqlExpr.TParseInsertSqlEvent Type 1709
SqlExpr.TParseSqlEvent Type 1709

SqlExpr.TSQLConnectionLoginEvent Type 1709
SqlExpr.TTableScopes Type 1710
SqlExpr.TTraceEvent Type 1710
SqlExpr.TTraceLogEvent Type 1710

Constants 1711

SqlExpr.BlobTypeMap Constant 1711
SqlExpr.DataTypeMap Constant 1712
SqlExpr.DefaultCursor Constant 1712
SqlExpr.DefaultMaxBlobSize Constant 1713
SqlExpr.DefaultRowsetSize Constant 1713
SqlExpr.eSQLAutoIncr Constant 1713
SqlExpr.eSQLDefault Constant 1714
SqlExpr.eSQLFunction Constant 1714
SqlExpr.eSQLLocal Constant 1714
SqlExpr.eSQLNonUnique Constant 1715
SqlExpr.eSQLPackage Constant 1715
SqlExpr.eSQLPrimaryKey Constant 1715
SqlExpr.eSQLProcedure Constant 1716
SqlExpr.eSQLRowId Constant 1716
SqlExpr.eSQLRowVersion Constant 1716
SqlExpr.eSQLSynonym Constant 1717
SqlExpr.eSQLSysProcedure Constant 1717
SqlExpr.eSQLSystemTable Constant 1717
SqlExpr.eSQLTable Constant 1718
SqlExpr.eSQLTempTable Constant 1718
SqlExpr.eSQLUnique Constant 1718
SqlExpr.eSQLView Constant 1719
SqlExpr.FldSubTypeMap Constant 1719
SqlExpr.FldTypeMap Constant 1719
SqlExpr.HourGlassCursor Constant 1720
SqlExpr.SAnd Constant 1721
SqlExpr.SOrderBy Constant 1721
SqlExpr.SParam Constant 1721
SqlExpr.SSelect Constant 1722
SqlExpr.SSelectSpaces Constant 1722
SqlExpr.SSelectStar Constant 1722
SqlExpr.SSelectStarFrom Constant 1723
SqlExpr.SUB_TYPE_MEMO Constant 1723
SqlExpr.SWhere Constant 1723

SqlExpr.TErrorMessageSize Constant 1724

SqlTimSt Namespace 1725

Classes **1727**

TTimeZone Record 1727

TTimeZone Members 1727

TTimeZone Fields 1727

TTimeZone Methods 1728

Functions **1729**

SqlTimSt.CheckSqlTimeStamp Function 1729

SqlTimSt.DateTimeToSQLTimeStamp Function 1730

SqlTimSt.LocalToUTC Function 1730

SqlTimSt.SQLDayOfWeek Function 1731

SqlTimSt.SQLTimeStampToDateTime Function 1731

SqlTimSt.SQLTimeStampToStr Function 1732

SqlTimSt.StrToSQLTimeStamp Function 1732

SqlTimSt.TryStrToSQLTimeStamp Function 1733

SqlTimSt.UTCToLocal Function 1733

SqlTimSt.VarIsSQLTimeStamp Function 1734

SqlTimSt.VarSQLTimeStamp Function 1734

SqlTimSt.VarSQLTimeStampCreate Function 1735

SqlTimSt.VarToSQLTimeStamp Function 1736

Structs, Records, Enums **1737**

SqlTimSt.TSQLTimeStamp Record 1737

Types **1739**

SqlTimSt.PSQLTimeStamp Type 1739

Constants **1740**

SqlTimSt.NullSQLTimeStamp Constant 1740

StdActnMenus Namespace 1741

Classes **1742**

TStandardAddRemoveItem Class 1742

TStandardAddRemoveItem Members 1742

TStandardAddRemoveItem Methods 1743

TStandardButtonControl Class 1744

TStandardButtonControl Members 1744

TStandardButtonControl Methods 1744
TStandardCustomizePopup Class 1745
TStandardCustomizePopup Members 1745
TStandardCustomizePopup Methods 1746
TStandardDropDownButton Class 1747
TStandardDropDownButton Members 1747
TStandardDropDownButton Methods 1747
TStandardMenuButton Class 1748
TStandardMenuButton Members 1749
TStandardMenuButton Methods 1749
TStandardMenuExpandBtn Class 1750
TStandardMenuExpandBtn Members 1750
TStandardMenuExpandBtn Methods 1751
TStandardMenuItem Class 1751
TStandardMenuItem Members 1752
TStandardMenuItem Methods 1752
TStandardMenuPopup Class 1756
TStandardMenuPopup Members 1756
TStandardMenuPopup Methods 1756
TStandardToolScrollBtn Class 1758
TStandardToolScrollBtn Members 1758
TStandardToolScrollBtn Methods 1758

Functions **1760**
StdActnMenus.RegisterStandardMenus Function 1760

StdActns Namespace 1761

Classes **1763**
TBrowseForFolder Class 1764
TBrowseForFolder Members 1764
TBrowseForFolder Methods 1765
TBrowseForFolder Properties 1767
TBrowseForFolder Events 1772
TColorSelect Class 1774
TColorSelect Members 1775
TColorSelect Methods 1776
TColorSelect Properties 1776
TColorSelect Events 1781
TCommonDialogAction Class 1784

TCommonDialogAction Members 1785
TCommonDialogAction.Create Constructor 1785
TCommonDialogAction.Destroy Destructor 1786
TCommonDialogAction Fields 1786
TCommonDialogAction Methods 1786
TCommonDialogAction Properties 1789
TCommonDialogAction Events 1790
TEditAction Class 1791
TEditAction Members 1792
TEditAction.Destroy Destructor 1792
TEditAction Methods 1793
TEditAction Properties 1794
TEditCopy Class 1795
TEditCopy Members 1796
TEditCopy Methods 1796
TEditCut Class 1797
TEditCut Members 1798
TEditCut Methods 1798
TEditDelete Class 1799
TEditDelete Members 1799
TEditDelete Methods 1800
TEditPaste Class 1801
TEditPaste Members 1802
TEditPaste Methods 1802
TEditSelectAll Class 1803
TEditSelectAll Members 1804
TEditSelectAll Methods 1804
TEditUndo Class 1805
TEditUndo Members 1806
TEditUndo Methods 1806
TFileAction Class 1807
TFileAction Members 1808
TFileAction Methods 1809
TFileAction Properties 1809
TFileExit Class 1810
TFileExit Members 1810
TFileExit Methods 1811
TFileExit Properties 1812
TFileExit Events 1816

TFileOpen Class 1817
TFileOpen Members 1818
TFileOpen.Create Constructor 1819
TFileOpen Methods 1820
TFileOpen Properties 1821
TFileOpen Events 1826
TFileOpenWith Class 1829
TFileOpenWith Members 1830
TFileOpenWith Methods 1831
TFileOpenWith Properties 1832
TFileOpenWith Events 1834
TFilePageSetup Class 1836
TFilePageSetup Members 1836
TFilePageSetup Methods 1837
TFilePageSetup Properties 1838
TFilePageSetup Events 1842
TFilePrintSetup Class 1845
TFilePrintSetup Members 1846
TFilePrintSetup Methods 1847
TFilePrintSetup Properties 1847
TFilePrintSetup Events 1852
TFileSaveAs Class 1855
TFileSaveAs Members 1855
TFileSaveAs.Create Constructor 1856
TFileSaveAs Methods 1857
TFileSaveAs Properties 1857
TFileSaveAs Events 1861
TFontEdit Class 1864
TFontEdit Members 1865
TFontEdit Methods 1866
TFontEdit Properties 1866
TFontEdit Events 1871
THelpAction Class 1874
THelpAction Members 1874
THelpAction.Create Constructor 1875
THelpAction Methods 1875
THelpContents Class 1876
THelpContents Members 1876
THelpContents Methods 1877

THelpContextAction Class 1877
THelpContextAction Members 1878
THelpContextAction Methods 1878
THelpOnHelp Class 1879
THelpOnHelp Members 1880
THelpOnHelp Methods 1880
THelpTopicSearch Class 1881
THelpTopicSearch Members 1881
THelpTopicSearch Methods 1881
THintAction Class 1882
THintAction Members 1883
THintAction.Create Constructor 1883
THintAction Properties 1884
TPrintDlg Class 1884
TPrintDlg Members 1885
TPrintDlg Methods 1886
TPrintDlg Properties 1886
TPrintDlg Events 1891
TSearchAction Class 1894
TSearchAction Members 1895
TSearchAction.Create Constructor 1895
TSearchAction.Destroy Destructor 1896
TSearchAction Fields 1896
TSearchAction Methods 1897
TSearchFind Class 1899
TSearchFind Members 1900
TSearchFind Methods 1901
TSearchFind Properties 1902
TSearchFind Events 1906
TSearchFindFirst Class 1909
TSearchFindFirst Members 1910
TSearchFindFirst.Create Constructor 1911
TSearchFindNext Class 1912
TSearchFindNext Members 1912
TSearchFindNext.Create Constructor 1913
TSearchFindNext Methods 1913
TSearchFindNext Properties 1915
TSearchFindNext Events 1920
TSearchReplace Class 1921

TSearchReplace Members 1922
TSearchReplace Methods 1923
TSearchReplace Properties 1924
TSearchReplace Events 1929
TWindowAction Class 1932
TWindowAction Members 1933
TWindowAction.Create Constructor 1933
TWindowAction Methods 1933
TWindowAction Properties 1935
TWindowArrange Class 1936
TWindowArrange Members 1936
TWindowArrange Methods 1937
TWindowCascade Class 1937
TWindowCascade Members 1938
TWindowCascade Methods 1938
TWindowClose Class 1939
TWindowClose Members 1940
TWindowClose Methods 1940
TWindowMinimizeAll Class 1941
TWindowMinimizeAll Members 1942
TWindowMinimizeAll Methods 1942
TWindowTileHorizontal Class 1943
TWindowTileHorizontal Members 1943
TWindowTileHorizontal Methods 1944
TWindowTileVertical Class 1944
TWindowTileVertical Members 1945
TWindowTileVertical Methods 1945

Structs, Records, Enums **1947**
StdActns.TBrowseForFolderOption Enumeration 1947

Types **1949**
StdActns.TBrowseForFolderCallbackEvent Type 1949
StdActns.TBrowseOptions Type 1949
StdActns.TCommonDialogClass Type 1950

StdConvs Namespace 1953

Functions **1958**
StdConvs.CelsiusToFahrenheit Function 1958
StdConvs.FahrenheitToCelsius Function 1958

StdConvs.InitStdConvs Function 1959

Variables **1960**

StdConvs.auAcres Variable 1962
StdConvs.auAres Variable 1963
StdConvs.auCentares Variable 1963
StdConvs.auHectares Variable 1963
StdConvs.auSquareCentimeters Variable 1964
StdConvs.auSquareDecameters Variable 1964
StdConvs.auSquareDecimeters Variable 1964
StdConvs.auSquareFeet Variable 1965
StdConvs.auSquareHectometers Variable 1965
StdConvs.auSquareInches Variable 1965
StdConvs.auSquareKilometers Variable 1966
StdConvs.auSquareMeters Variable 1966
StdConvs.auSquareMiles Variable 1966
StdConvs.auSquareMillimeters Variable 1967
StdConvs.auSquareRods Variable 1967
StdConvs.auSquareYards Variable 1967
StdConvs.cbArea Variable 1968
StdConvs.cbDistance Variable 1968
StdConvs.cbMass Variable 1969
StdConvs.cbTemperature Variable 1969
StdConvs.cbTime Variable 1969
StdConvs.cbVolume Variable 1970
StdConvs.duAngstroms Variable 1971
StdConvs.duAstronomicalUnits Variable 1971
StdConvs.duCentimeters Variable 1972
StdConvs.duChains Variable 1972
StdConvs.duCubits Variable 1972
StdConvs.duDecameters Variable 1973
StdConvs.duDecimeters Variable 1973
StdConvs.duFathoms Variable 1973
StdConvs.duFeet Variable 1974
StdConvs.duFurlongs Variable 1974
StdConvs.duGigameters Variable 1974
StdConvs.duHands Variable 1975
StdConvs.duHectometers Variable 1975
StdConvs.duInches Variable 1975
StdConvs.duKilometers Variable 1976

StdConvs.duLightYears Variable 1976
StdConvs.duLinks Variable 1976
StdConvs.duMegameters Variable 1977
StdConvs.duMeters Variable 1977
StdConvs.duMicromicrons Variable 1977
StdConvs.duMicrons Variable 1978
StdConvs.duMiles Variable 1978
StdConvs.duMillimeters Variable 1978
StdConvs.duMillimicrons Variable 1979
StdConvs.duNauticalMiles Variable 1979
StdConvs.duPaces Variable 1979
StdConvs.duParsecs Variable 1980
StdConvs.duPicas Variable 1980
StdConvs.duPoints Variable 1980
StdConvs.duRods Variable 1981
StdConvs.duYards Variable 1981
StdConvs.muCentigrams Variable 1981
StdConvs.muDecagrams Variable 1982
StdConvs.muDecigrams Variable 1982
StdConvs.muDrams Variable 1982
StdConvs.muGrains Variable 1983
StdConvs.muGrams Variable 1983
StdConvs.muHectograms Variable 1983
StdConvs.muKilograms Variable 1984
StdConvs.muLongTons Variable 1984
StdConvs.muMetricTons Variable 1984
StdConvs.muMicrograms Variable 1985
StdConvs.muMilligrams Variable 1985
StdConvs.muNanograms Variable 1985
StdConvs.muOunces Variable 1986
StdConvs.muPounds Variable 1986
StdConvs.muStones Variable 1986
StdConvs.muTons Variable 1987
StdConvs.tuCelsius Variable 1987
StdConvs.tuCenturies Variable 1987
StdConvs.tuDateTime Variable 1988
StdConvs.tuDays Variable 1988
StdConvs.tuDecades Variable 1988
StdConvs.tuFahrenheit Variable 1989

StdConvs.tuFortnights Variable 1989
StdConvs.tuHours Variable 1989
StdConvs.tuJulianDate Variable 1990
StdConvs.tuKelvin Variable 1990
StdConvs.tuMillennia Variable 1990
StdConvs.tuMilliSeconds Variable 1991
StdConvs.tuMinutes Variable 1991
StdConvs.tuModifiedJulianDate Variable 1991
StdConvs.tuMonths Variable 1992
StdConvs.tuRankine Variable 1992
StdConvs.tuReaumur Variable 1992
StdConvs.tuSeconds Variable 1993
StdConvs.tuWeeks Variable 1993
StdConvs.tuYears Variable 1993
StdConvs.vuAcreFeet Variable 1994
StdConvs.vuAcreInches Variable 1994
StdConvs.vuCentiLiters Variable 1994
StdConvs.vuCordFeet Variable 1995
StdConvs.vuCords Variable 1995
StdConvs.vuCubicCentimeters Variable 1995
StdConvs.vuCubicDecameters Variable 1996
StdConvs.vuCubicDecimeters Variable 1996
StdConvs.vuCubicFeet Variable 1996
StdConvs.vuCubicHectometers Variable 1997
StdConvs.vuCubicInches Variable 1997
StdConvs.vuCubicKilometers Variable 1997
StdConvs.vuCubicMeters Variable 1998
StdConvs.vuCubicMiles Variable 1998
StdConvs.vuCubicMillimeters Variable 1998
StdConvs.vuCubicYards Variable 1999
StdConvs.vuDecaLiters Variable 1999
StdConvs.vuDecasteres Variable 1999
StdConvs.vuDeciLiters Variable 2000
StdConvs.vuDecisteres Variable 2000
StdConvs.vuDryBuckets Variable 2000
StdConvs.vuDryBushels Variable 2001
StdConvs.vuDryGallons Variable 2001
StdConvs.vuDryPecks Variable 2001
StdConvs.vuDryPints Variable 2002

StdConvs.vuDryQuarts Variable 2002
StdConvs.vuFluidCups Variable 2002
StdConvs.vuFluidGallons Variable 2003
StdConvs.vuFluidGills Variable 2003
StdConvs.vuFluidOunces Variable 2003
StdConvs.vuFluidPints Variable 2004
StdConvs.vuFluidQuarts Variable 2004
StdConvs.vuFluidTablespoons Variable 2004
StdConvs.vuFluidTeaspoons Variable 2005
StdConvs.vuHectoLiters Variable 2005
StdConvs.vuKiloLiters Variable 2005
StdConvs.vuLiters Variable 2006
StdConvs.vuMilliLiters Variable 2006
StdConvs.vuSteres Variable 2006
StdConvs.vuUKBuckets Variable 2007
StdConvs.vuUKBushels Variable 2007
StdConvs.vuUKGallons Variable 2007
StdConvs.vuUKGills Variable 2008
StdConvs.vuUKOunces Variable 2008
StdConvs.vuUKPecks Variable 2008
StdConvs.vuUKPints Variable 2009
StdConvs.vuUKPottles Variable 2009
StdConvs.vuUKQuarts Variable 2009

Constants **2010**

StdConvs.CubicMetersPerAcreFoot Constant 2011
StdConvs.CubicMetersPerAcreInch Constant 2011
StdConvs.CubicMetersPerCord Constant 2012
StdConvs.CubicMetersPerCordFoot Constant 2012
StdConvs.CubicMetersPerCubicFoot Constant 2012
StdConvs.CubicMetersPerCubicInch Constant 2013
StdConvs.CubicMetersPerCubicMile Constant 2013
StdConvs.CubicMetersPerCubicYard Constant 2013
StdConvs.CubicMetersPerUKBucket Constant 2014
StdConvs.CubicMetersPerUKBushel Constant 2014
StdConvs.CubicMetersPerUKGallon Constant 2014
StdConvs.CubicMetersPerUKGill Constant 2015
StdConvs.CubicMetersPerUKOunce Constant 2015
StdConvs.CubicMetersPerUKPeck Constant 2015
StdConvs.CubicMetersPerUKPint Constant 2016

StdConvs.CubicMetersPerUKPottle Constant 2016
StdConvs.CubicMetersPerUKQuart Constant 2016
StdConvs.CubicMetersPerUSDryBucket Constant 2017
StdConvs.CubicMetersPerUSDryBushel Constant 2017
StdConvs.CubicMetersPerUSDryGallon Constant 2017
StdConvs.CubicMetersPerUSDryPeck Constant 2018
StdConvs.CubicMetersPerUSDryPint Constant 2018
StdConvs.CubicMetersPerUSDryQuart Constant 2018
StdConvs.CubicMetersPerUSFluidCup Constant 2019
StdConvs.CubicMetersPerUSFluidGallon Constant 2019
StdConvs.CubicMetersPerUSFluidGill Constant 2019
StdConvs.CubicMetersPerUSFluidOunce Constant 2020
StdConvs.CubicMetersPerUSFluidPint Constant 2020
StdConvs.CubicMetersPerUSFluidQuart Constant 2020
StdConvs.CubicMetersPerUSFluidTablespoon Constant 2021
StdConvs.CubicMetersPerUSFluidTeaspoon Constant 2021
StdConvs.GramsPerDrams Constant 2021
StdConvs.GramsPerGrains Constant 2022
StdConvs.GramsPerLongTons Constant 2022
StdConvs.GramsPerOunces Constant 2022
StdConvs.GramsPerPound Constant 2023
StdConvs.GramsPerStones Constant 2023
StdConvs.GramsPerTons Constant 2023
StdConvs.MetersPerAstronomicalUnit Constant 2024
StdConvs.MetersPerChain Constant 2024
StdConvs.MetersPerCubit Constant 2024
StdConvs.MetersPerFathom Constant 2025
StdConvs.MetersPerFoot Constant 2025
StdConvs.MetersPerFurlong Constant 2025
StdConvs.MetersPerHand Constant 2026
StdConvs.MetersPerInch Constant 2026
StdConvs.MetersPerLightSecond Constant 2026
StdConvs.MetersPerLightYear Constant 2027
StdConvs.MetersPerLink Constant 2027
StdConvs.MetersPerMile Constant 2027
StdConvs.MetersPerNauticalMiles Constant 2028
StdConvs.MetersPerPace Constant 2028
StdConvs.MetersPerParsec Constant 2028
StdConvs.MetersPerPica Constant 2029

StdConvs.MetersPerPoint Constant 2029
StdConvs.MetersPerRod Constant 2029
StdConvs.MetersPerYard Constant 2030
StdConvs.SquareMetersPerAcre Constant 2030
StdConvs.SquareMetersPerSquareFoot Constant 2030
StdConvs.SquareMetersPerSquareInch Constant 2031
StdConvs.SquareMetersPerSquareMile Constant 2031
StdConvs.SquareMetersPerSquareRod Constant 2031
StdConvs.SquareMetersPerSquareYard Constant 2032

StdCtrls Namespace 2033

Classes 2035
TButton Class 2035
TButton Members 2036
TButton Properties 2038
TButton Events 2072
TButtonActionLink Class 2106
TButtonActionLink Members 2107
TButtonActionLink Fields 2107
TButtonActionLink Methods 2107
TButtonControl Class 2109
TButtonControl Members 2109
TButtonControl.Create Constructor 2110
TButtonControl Methods 2110
TButtonControl Properties 2112
TCheckBox Class 2116
TCheckBox Members 2116
TCheckBox Properties 2118
TCheckBox Events 2151
TComboBox Class 2184
TComboBox Members 2185
TComboBox Properties 2189
TComboBox Events 2227
TCustomButton Class 2259
TCustomButton Members 2259
TCustomButton.Create Constructor 2260
TCustomButton.Destroy Destructor 2261
TCustomButton Enumerations 2261

TCustomButton Methods 2262
TCustomButton Properties 2267
TCustomButton Events 2275
TCustomCheckBox Class 2276
TCustomCheckBox Members 2276
TCustomCheckBox.Create Constructor 2277
TCustomCheckBox Methods 2277
TCustomCheckBox Properties 2280
TCustomCombo Class 2283
TCustomCombo Members 2284
TCustomCombo.Create Constructor 2285
TCustomCombo.Destroy Destructor 2286
TCustomCombo Fields 2286
TCustomCombo Methods 2290
TCustomCombo Properties 2304
TCustomCombo Events 2318
TCustomComboBox Class 2322
TCustomComboBox Members 2323
TCustomComboBox.Create Constructor 2325
TCustomComboBox.Destroy Destructor 2326
TCustomComboBox Methods 2326
TCustomComboBox Properties 2331
TCustomComboBox Events 2337
TCustomComboBoxStrings Class 2339
TCustomComboBoxStrings Members 2339
TCustomComboBoxStrings.Create Constructor 2340
TCustomComboBoxStrings.Destroy Destructor 2340
TCustomComboBoxStrings Methods 2340
TCustomComboBoxStrings Properties 2344
TCustomEdit Class 2345
TCustomEdit Members 2345
TCustomEdit.Create Constructor 2347
TCustomEdit Methods 2347
TCustomEdit Properties 2364
TCustomEdit Events 2380
TCustomGroupBox Class 2381
TCustomGroupBox Members 2382
TCustomGroupBox.Create Constructor 2382
TCustomGroupBox Methods 2382

TCustomLabel Class 2384
TCustomLabel Members 2384
TCustomLabel.Create Constructor 2385
TCustomLabel Methods 2385
TCustomLabel Properties 2388
TCustomListBox Class 2398
TCustomListBox Members 2399
TCustomListBox.Create Constructor 2400
TCustomListBox.Destroy Destructor 2401
TCustomListBox Fields 2401
TCustomListBox Methods 2402
TCustomListBox Properties 2422
TCustomListBox Events 2438
TCustomMemo Class 2443
TCustomMemo Members 2444
TCustomMemo.Create Constructor 2446
TCustomMemo.Destroy Destructor 2446
TCustomMemo Methods 2446
TCustomMemo Properties 2450
TCustomStaticText Class 2457
TCustomStaticText Members 2458
TCustomStaticText.Create Constructor 2458
TCustomStaticText Methods 2459
TCustomStaticText Properties 2461
TEdit Class 2463
TEdit Members 2464
TEdit Properties 2466
TEdit Events 2501
TGroupBox Class 2535
TGroupBox Members 2535
TGroupBox Properties 2537
TGroupBox Events 2565
TImageMargins Class 2602
TImageMargins Members 2603
TImageMargins Methods 2603
TImageMargins Properties 2605
TImageMargins Events 2607
TLabel Class 2607
TLabel Members 2608

TLabel Properties 2609
TLabel Events 2637
TListBox Class 2657
TListBox Members 2658
TListBox Properties 2661
TListBox Events 2698
TMemo Class 2737
TMemo Members 2737
TMemo Properties 2740
TMemo Events 2777
TPushButtonActionLink Class 2811
TPushButtonActionLink Members 2811
TPushButtonActionLink Methods 2811
TRadioButton Class 2813
TRadioButton Members 2813
TRadioButton.Create Constructor 2815
TRadioButton Methods 2815
TRadioButton Properties 2818
TRadioButton Events 2847
TScrollBar Class 2881
TScrollBar Members 2881
TScrollBar.Create Constructor 2883
TScrollBar Methods 2883
TScrollBar Properties 2886
TScrollBar Events 2920
TStaticText Class 2945
TStaticText Members 2946
TStaticText Properties 2947
TStaticText Events 2977

Structs, Records, Enums 2999

StdCtrls.TCheckBoxState Enumeration 2999
StdCtrls.TComboBoxStyle Enumeration 3000
StdCtrls.TEditCharCase Enumeration 3000
StdCtrls.TEllipsisPosition Enumeration 3001
StdCtrls.TImageAlignment Enumeration 3002
StdCtrls.TListBoxStyle Enumeration 3003
StdCtrls.TScrollCode Enumeration 3004
StdCtrls.TScrollStyle Enumeration 3005
StdCtrls.TSelection Record 3006

StdCtrls.TStaticBorderStyle Enumeration 3006
StdCtrls.TTextLayout Enumeration 3007

Types **3008**
StdCtrls.TButtonActionLinkClass Type 3008
StdCtrls.TCustomComboBoxStringsClass Type 3008
StdCtrls.TDrawItemEvent Type 3009
StdCtrls.TFNDrawText Type 3010
StdCtrls.TLBFindDataEvent Type 3010
StdCtrls.TLBGetDataEvent Type 3011
StdCtrls.TLBGetDataObjectEvent Type 3011
StdCtrls.TMeasureItemEvent Type 3012
StdCtrls.TOwnerDrawState Type 3012
StdCtrls.TPushButtonActionLinkClass Type 3013
StdCtrls.TScrollEvent Type 3013

StdStyleActnCtrls Namespace 3015

Classes **3016**
TStandardStyleActionBars Class 3016
TStandardStyleActionBars Members 3016
TStandardStyleActionBars Methods 3016

Variables **3019**
StdStyleActnCtrls.StandardStyle Variable 3019

StrUtils Namespace 3021

Functions **3024**
StrUtils.AnsiContainsStr Function 3025
StrUtils.AnsiContainsText Function 3026
StrUtils.AnsiEndsStr Function 3026
StrUtils.AnsiEndsText Function 3027
StrUtils.AnsiIndexStr Function 3028
StrUtils.AnsiIndexText Function 3028
StrUtils.AnsiLeftStr Function 3029
StrUtils.AnsiMatchStr Function 3029
StrUtils.AnsiMatchText Function 3030
StrUtils.AnsiMidStr Function 3030
StrUtils.AnsiReplaceStr Function 3031
StrUtils.AnsiReplaceText Function 3032

StrUtils.AnsiResemblesText Function 3032
StrUtils.AnsiReverseString Function 3033
StrUtils.AnsiRightStr Function 3033
StrUtils.AnsiStartsStr Function 3034
StrUtils.AnsiStartsText Function 3034
StrUtils.ContainsStr Function 3035
StrUtils.ContainsText Function 3035
StrUtils.DecodeSoundexInt Function 3036
StrUtils.DecodeSoundexWord Function 3036
StrUtils.DupeString Function 3037
StrUtils.EndsStr Function 3037
StrUtils.EndsText Function 3038
StrUtils.IfThen Function 3038
StrUtils.IndexStr Function 3039
StrUtils.IndexText Function 3039
StrUtils.LeftBStr Function 3040
StrUtils.LeftStr Function 3040
StrUtils.MatchStr Function 3041
StrUtils.MatchText Function 3041
StrUtils.MidBStr Function 3042
StrUtils.MidStr Function 3043
StrUtils.PosEx Function 3043
StrUtils.RandomFrom Function 3044
StrUtils.ReplaceStr Function 3044
StrUtils.ReplaceText Function 3045
StrUtils.ResemblesText Function 3046
StrUtils.ReverseString Function 3046
StrUtils.RightBStr Function 3046
StrUtils.RightStr Function 3047
StrUtils.SearchBuf Function 3048
StrUtils.Soundex Function 3048
StrUtils.SoundexCompare Function 3049
StrUtils.SoundexInt Function 3050
StrUtils.SoundexProc Function 3050
StrUtils.SoundexSimilar Function 3051
StrUtils.SoundexWord Function 3052
StrUtils.StartsStr Function 3052
StrUtils.StartsText Function 3053
StrUtils.StuffString Function 3053

Structs, Records, Enums **3055**
StrUtils.TStringSeachOption Enumeration 3055

Types **3056**
StrUtils.TCompareTextProc Type 3056
StrUtils.TSoundexIntLength Type 3056
StrUtils.TSoundexLength Type 3057
StrUtils.TStringSearchOptions Type 3057

Variables **3058**
StrUtils.AnsiResemblesProc Variable 3058
StrUtils.ResemblesProc Variable 3058

Constants **3060**
StrUtils.WordDelimiters Constant 3060

SvcMgr Namespace 3061

Classes **3063**
TDependencies Class 3063
TDependencies Members 3063
TDependencies.Create Constructor 3064
TDependencies Methods 3064
TDependencies Properties 3065
TDependency Class 3066
TDependency Members 3066
TDependency Methods 3067
TDependency Properties 3067
TEventLogger Class 3068
TEventLogger Members 3069
TEventLogger.Create Constructor 3069
TEventLogger.Destroy Destructor 3070
TEventLogger Methods 3070
TService Class 3071
TService Members 3071
TService.CreateNew Constructor 3073
TService.Destroy Destructor 3073
TService Methods 3073
TService Properties 3077
TService Events 3085
TServiceApplication Class 3089

TServiceApplication Members 3090
TServiceApplication.Create Constructor 3091
TServiceApplication.Destroy Destructor 3091
TServiceApplication Methods 3091
TServiceApplication Properties 3094
TServiceThread Class 3095
TServiceThread Members 3096
TServiceThread.Create Constructor 3096
TServiceThread Methods 3096

Structs, Records, Enums 3102
SvcMgr.TCurrentStatus Enumeration 3102
SvcMgr.TErrorSeverity Enumeration 3103
SvcMgr.TServiceType Enumeration 3103
SvcMgr.TStartType Enumeration 3104

Types 3106
SvcMgr.TContinueEvent Type 3106
SvcMgr.TPauseEvent Type 3106
SvcMgr.TServiceController Type 3107
SvcMgr.TServiceEvent Type 3107
SvcMgr.TStartEvent Type 3108
SvcMgr.TStopEvent Type 3108

Variables 3109
SvcMgr.Application Variable 3109

Constants 3110
SvcMgr.CM_SERVICE_CONTROL_CODE Constant 3110

SyncObjs Namespace 3111

Classes 3112
TConditionVariableCS Class 3112
TConditionVariableCS Members 3113
TConditionVariableCS.Create Constructor 3114
TConditionVariableCS Fields 3114
TConditionVariableCS Methods 3115
TConditionVariableHelper Class Helper 3117
TConditionVariableHelper Members 3118
TConditionVariableHelper Methods 3118
TConditionVariableMutex Class 3120

TConditionVariableMutex Members 3121
TConditionVariableMutex.Create Constructor 3122
TConditionVariableMutex.Destroy Destructor 3123
TConditionVariableMutex Methods 3123
TCriticalSection Class 3125
TCriticalSection Members 3126
TCriticalSection.Create Constructor 3127
TCriticalSection.Destroy Destructor 3127
TCriticalSection Fields 3127
TCriticalSection Methods 3128
TCriticalSectionHelper Class Helper 3130
TCriticalSectionHelper Members 3131
TCriticalSectionHelper Methods 3131
TEvent Class 3133
TEvent Members 3134
Create Constructor 3135
TEvent Methods 3136
THandleObject Class 3137
THandleObject Members 3138
THandleObject.Create Constructor 3139
THandleObject.Destroy Destructor 3139
THandleObject Fields 3140
THandleObject Methods 3140
THandleObject Properties 3142
TMutex Class 3143
TMutex Members 3143
Create Constructor 3145
TMutex Methods 3146
TSemaphore Class 3147
TSemaphore Members 3148
Create Constructor 3149
TSemaphore Methods 3150
TSimpleEvent Class 3152
TSynchroObject Class 3152
TSynchroObject Members 3153
TSynchroObject Methods 3154

Structs, Records, Enums 3156

SyncObjs.TWaitResult Enumeration 3156

Types **3157**
SyncObjs.THandleObjectArray Type 3157

System Namespace 3159

Classes **3174**
AnsiStringBase Class 3174
AnsiStringBase Members 3176
AnsiStringBase::~AnsiStringBase Destructor 3178
AnsiStringBase Constructor 3178
AnsiStringBase Structures 3179
AnsiStringBase Enumerations 3180
AnsiStringBase Data Members 3182
AnsiStringBase Methods 3182
AnsiStringBase Operators 3204
AnsiStringT Template 3207
AnsiStringT Members 3209
AnsiStringT::~AnsiStringT Destructor 3211
AnsiStringT Constructor 3211
AnsiStringT Methods 3213
AnsiStringT Operators 3231
Currency Class 3235
Currency Members 3235
Currency Constructor 3237
Currency Operators 3238
Currency Friends 3253
CurrencyBase Class 3255
CurrencyBase Members 3255
CurrencyBase Data Members 3256
DelphiInterface Template 3256
DelphiInterface Members 3257
DelphiInterface::~DelphiInterface Destructor 3257
DelphiInterface Constructor 3257
DelphiInterface Methods 3258
DelphiInterface Operators 3259
DynamicArray Template 3261
DynamicArray Members 3263
DynamicArray::~DynamicArray Destructor 3263
DynamicArray Constructor 3264

DynamicArray Methods 3264
DynamicArray Operators 3268
DynamicArray Properties 3270
DynArrayException Class 3271
DynArrayNullData Class 3272
DynArrayOutOfRange Class 3272
DynArrayOutOfRange Members 3272
DynArrayOutOfRange::DynArrayOutOfRange Constructor 3273
DynArrayOutOfRange Data Members 3273
OleVariant Class 3274
OleVariant Members 3275
OleVariant::~OleVariant Destructor 3279
OleVariant Constructor 3279
OleVariant Data Members 3282
OleVariant Operators 3301
SmallString Template 3307
SmallString Members 3307
SmallString Constructor 3308
SmallString Operators 3309
SmallStringBase Template 3310
SmallStringBase Members 3310
SmallStringBase Data Members 3310
sPointer Template 3311
sPointer Members 3311
sPointer::~sPointer Destructor 3312
sPointer Constructor 3312
sPointer Operators 3313
TAggregatedObject Class 3313
TAggregatedObject Members 3314
TAggregatedObject::Create Constructor 3315
TAggregatedObject Methods 3315
TAggregatedObject Properties 3317
TClassHelperBase Class 3318
TClassHelperBase Members 3318
TClassHelperBase._Create Constructor 3319
TClassHelperBase Fields 3319
TContainedObject Class 3320
TContainedObject Members 3321
TContainedObject Methods 3322

TDateTime Class 3323
TDateTime Members 3324
TDateTime Constructor 3325
TDateTime Enumerations 3327
TDateTime Methods 3327
TDateTime Operators 3333
TDateTimeBase Class 3341
TDateTimeBase Members 3341
TDateTimeBase Data Members 3342
TInterfacedObject Class 3342
TInterfacedObject Members 3343
TInterfacedObject Data Members 3344
TInterfacedObject Methods 3344
TInterfacedObject Properties 3347
TMetaClass Class 3348
TMonitor Record 3349
TMonitor Members 3350
TMonitor Methods 3350
TObject Class 3354
TObject Members 3355
TObject::Create Constructor 3355
TObject::Destroy Destructor 3356
TObject Methods 3356
UnicodeString Class 3381
UnicodeString Members 3382
UnicodeString::~UnicodeString Destructor 3384
UnicodeString Constructor 3384
UnicodeString Structures 3387
UnicodeString Enumerations 3387
UnicodeString Methods 3389
UnicodeString Operators 3413
UnicodeString Friends 3420
Variant Class 3421
Variant Members 3422
Variant::~Variant Destructor 3424
Variant Constructor 3424
Variant Methods 3428
Variant Operators 3437
WideString Class 3467

WideString Members 3467
WideString::~WideString Destructor 3469
WideString Constructor 3469
WideString Enumerations 3471
WideString Methods 3472
WideString Operators 3487

Interfaces 3493
IComparable Interface 3493
IComparable Members 3493
IComparable Methods 3494
IDispatch Interface 3494
IDispatch Members 3495
IDispatch Methods 3495
IEnumerable Interface 3497
IEnumerable Members 3498
IEnumerable Methods 3498
IEnumerator Interface 3498
IEnumerator Members 3499
IEnumerator Methods 3499
IEnumerator Properties 3500
IEquatable Interface 3500
IEquatable Members 3501
IEquatable Methods 3501
IInterface Interface 3502
IInterface Members 3502
IInterface Methods 3503
IInvokable Interface 3504

Functions 3506
System.Abs Function 3511
System.AcquireExceptionObject Function 3512
System.AddModuleUnloadProc Function 3514
System.Addr Function 3514
System.AllocMem Function 3515
System.AnsiToUtf8 Function 3517
System.Append Function 3520
System.ArcTan Function 3521
System.Assert Function 3522
System.Assigned Function 3524

System.AssignFile Function 3525
System.AttemptToUseSharedMemoryManager Function 3527
System.BeginThread Function 3527
System.BlockRead Function 3528
System.BlockWrite Function 3530
System.Break Function 3531
System.ChDir Function 3534
System::CheckSafecallResult Function 3536
System.Chr Function 3536
System.Close Function 3538
System.CloseFile Function 3539
System.CompToCurrency Function 3542
System.CompToDouble Function 3543
System.Concat Function 3545
System.Continue Function 3546
System.Copy Function 3547
System.Cos Function 3549
System.CurrencyToComp Function 3550
System.Dec Function 3550
System.Delete Function 3552
System.Dispose Function 3553
System.DoubleToComp Function 3555
System.DynArrayClear Function 3555
System.DynArrayDim Function 3555
System.DynArraySetLength Function 3556
System.EndThread Function 3556
System.EnumModules Function 3556
System.EnumResourceModules Function 3557
System.Eof Function 3558
System.Eoln Function 3559
System.Erase Function 3560
System.Error Function 3560
System.ExceptAddr Function 3562
System.ExceptObject Function 3562
System.Exclude Function 3564
System.Exit Function 3565
System.Exp Function 3567
System.FilePos Function 3567
System.FileSize Function 3569

System.FillChar Function 3570
System.Finalize Function 3572
System.FindClassHInstance Function 3573
System::FindDynaInst Function 3573
System.FindHInstance Function 3574
System.FindResourceHInstance Function 3574
System.Flush Function 3575
System.FPower10 Function 3576
System.Frac Function 3576
System.FreeMem Function 3577
System.FreeMemory Function 3579
System.Get8087CW Function 3580
System.GetDir Function 3580
System.GetHeapStatus Function 3581
System.GetLastError Function 3582
System.GetMem Function 3584
System.GetMemory Function 3585
System.GetMemoryManager Function 3587
System.GetMemoryManagerState Function 3591
System.GetMemoryMap Function 3592
System.GetMinimumBlockAlignment Function 3592
System.GetModuleFileName Function 3593
System.GetVariantManager Function 3594
System.Halt Function 3594
System.Hi Function 3595
System.High Function 3596
System.Inc Function 3597
System.Include Function 3598
System.Initialize Function 3599
System.Insert Function 3601
System.Int Function 3602
System.IOResult Function 3602
System.IsMemoryManagerSet Function 3604
System.IsVariantManagerSet Function 3606
System.Length Function 3607
System.Ln Function 3608
System.Lo Function 3609
System.LoadResourceModule Function 3609
System::LoadResourceString Function 3610

System.LoadResString Function 3610
System.Low Function 3610
System.Mark Function 3612
System.MemoryBarrier Function 3612
System.MkDir Function 3612
System.MonitorEnter Function 3614
System.MonitorExit Function 3614
System.MonitorPulse Function 3615
System.MonitorPulseAll Function 3615
System.MonitorTryEnter Function 3616
System.MonitorWait Function 3616
System.Move Function 3617
System.MoveChars Function 3618
System.New Function 3619
System.Odd Function 3621
System.OleStrToString Function 3622
System.OleStrToStrVar Function 3622
System.Ord Function 3623
System.ParamCount Function 3623
System.ParamStr Function 3625
System.Pi Function 3626
System.Pos Function 3627
System.Pred Function 3628
System.Ptr Function 3630
System.PUCS4Chars Function 3632
System.RaiseList Function 3632
System.Random Function 3632
System.Randomize Function 3633
System.Read Function 3634
System.ReadLn Function 3636
System.ReallocMem Function 3637
System.ReallocMemory Function 3638
System.RegisterExpectedMemoryLeak Function 3640
System.RegisterModule Function 3640
System.Release Function 3641
System.ReleaseExceptionObject Function 3641
System.RemoveModuleUnloadProc Function 3641
System.Rename Function 3642
System.Reset Function 3643

System.Rewrite Function 3645
System.RmDir Function 3647
System.Round Function 3648
System.RunError Function 3649
System.Seek Function 3650
System.SeekEof Function 3651
System.SeekEoln Function 3652
System.Set8087CW Function 3653
System.SetCodePage Function 3655
System.SetInOutRes Function 3655
System.SetLastError Function 3656
System.SetLength Function 3657
System.SetLineBreakStyle Function 3659
System.SetMemoryManager Function 3660
System.SetMinimumBlockAlignment Function 3664
System.SetMultiByteConversionCodePage Function 3665
System.SetRaiseList Function 3665
System.SetString Function 3665
System.SetTextBuf Function 3667
System.SetVariantManager Function 3668
System.ShareMemoryManager Function 3668
System.Sin Function 3669
System.SizeOf Function 3670
System.Slice Function 3671
System.Sqr Function 3674
System.Sqrt Function 3675
System.Str Function 3676
System.StringCodePage Function 3676
System.StringElementSize Function 3677
System.StringOfChar Function 3677
System.StringRefCount Function 3678
System.StringToOleStr Function 3678
System.StringToWideChar Function 3679
System.Succ Function 3680
System.Swap Function 3680
System.SysAllocMem Function 3681
System.SysFreeMem Function 3681
System.SysGetMem Function 3682
System.SysReallocMem Function 3683

System.SysRegisterExpectedMemoryLeak Function 3683
System.SysUnregisterExpectedMemoryLeak Function 3684
System.TextStart Function 3684
System.Trunc Function 3684
System.Truncate Function 3685
System.UCS4StringToUnicodeString Function 3686
System.UCS4StringToWideString Function 3687
System.UnicodeStringToUCS4String Function 3689
System.UnicodeToUtf8 Function 3690
System.UniqueString Function 3693
System.UnregisterExpectedMemoryLeak Function 3694
System.UnregisterModule Function 3694
System.UpCase Function 3695
System.UTF8Decode Function 3696
System.UTF8Encode Function 3696
System.UTF8EncodeToShortString Function 3697
System.Utf8ToAnsi Function 3698
System.UTF8ToString Function 3698
System.Utf8ToUnicode Function 3699
System.UTF8ToUnicodeString Function 3700
System.UTF8ToWideString Function 3700
System.Val Function 3701
System.VarArrayRedim Function 3702
System.VarClear Function 3704
System.WideCharLenToString Function 3705
System.WideCharLenToStrVar Function 3706
System.WideCharToString Function 3706
System.WideCharToStrVar Function 3707
System.WideStringToUCS4String Function 3707
System.Write Function 3707
System.WriteLn Function 3710
System.YieldProcessor Function 3711

Structs, Records, Enums **3712**

System.PackageInfoTable Record 3713
System.PackageUnitEntry Record 3713
System.TCallDesc Record 3714
System.TChunkStatus Enumeration 3714
System.TCVModInfo Record 3715
System.TDispatchMessage Record 3716

System.TDispDesc Record 3716
System.TDynArrayTypeInfo Record 3716
System.TFileRec Record 3717
System.TGUID Record 3718
System::THeapBlock Structure 3720
System::THeapStatus Record 3720
System.TInitContext Record 3721
System.TInterfaceEntry Record 3722
System.TInterfaceTable Record 3723
System::TLibModule Record 3723
System::TMemoryManager Record 3724
System::TMemoryManagerEx Record 3725
System::TMemoryManagerState Record 3730
System.TMethod Record 3731
System.TMinimumBlockAlignment Enumeration 3732
System::TModuleUnloadRec Record 3732
System.TMonitorSupport Record 3733
System::TResStringRec Record 3733
System.TRuntimeError Enumeration 3734
System::TSmallBlockTypeState Record 3736
System.TTextLineBreakStyle Enumeration 3737
System.TTextRec Record 3738
System::TVarArray Record 3739
System::TVarArrayBound Record 3740
System.TVarData Record 3740
System::TVariantManager Record 3745
System::TVarRec Record 3747
System::TCallDesc Structure 3749
System::TDispDesc Structure 3749

Types **3750**

System.CppLongInt Type 3752
System.CppULongInt Type 3753
System.GetPackageInfoTable Type 3753
System.HGLOBAL Type 3753
System.HINST Type 3754
System.HMODULE Type 3754
System.HRESULT Type 3754
System.HRSRC Type 3755
System.Int16 Type 3755

System.Int32 Type 3755
System.Int8 Type 3756
System.IntegerArray Type 3756
System.IUnknown Type 3756
System.PackageInfo Type 3757
System.PAnsiString Type 3757
System.PBoolean Type 3758
System.PByte Type 3758
System::PCallDesc Type 3758
System.PCardinal Type 3759
System.PComp Type 3759
System.PCurrency Type 3759
System.PCVModInfo Type 3760
System.PDate Type 3760
System.PDateTime Type 3761
System.PDispatch Type 3761
System::PDispDesc Type 3761
System.PDouble Type 3762
System.PDynArrayTypeInfo Type 3762
System.PError Type 3762
System.PExtended Type 3763
System.PGUID Type 3763
System.PInitContext Type 3765
System.PInt64 Type 3765
System.PInteger Type 3765
System.PIntegerArray Type 3766
System.PInterfaceEntry Type 3766
System.PInterfaceTable Type 3766
System.PLibModule Type 3767
System.PLongint Type 3767
System.PLongWord Type 3768
System::PMemoryManager Type 3768
System::PMemoryManagerEx Type 3768
System::PModuleUnloadRec Type 3769
System.PMonitor Type 3769
System.PMonitorSupport Type 3769
System.PointerArray Type 3770
System.POleVariant Type 3770
System.PPAnsiChar Type 3770

System.PPChar Type 3771
System.PPCharArray Type 3771
System.PPDispatch Type 3771
System.PPMonitor Type 3772
System.PPointer Type 3772
System.PPointerArray Type 3772
System.PPUnknown Type 3773
System.PPWideChar Type 3773
System.PRawByteString Type 3773
System::PResStringRec Type 3774
System.PShortInt Type 3774
System.PShortString Type 3774
System.PSingle Type 3775
System::PSmallBlockTypeState Type 3775
System.PSmallInt Type 3775
System.PString Type 3776
System.PTextBuf Type 3776
System.PUCS2Char Type 3776
System.PUCS4Char Type 3777
System.PUCS4CharArray Type 3777
System.PUInt64 Type 3777
System.PUnicodeString Type 3778
System.PUnitEntryTable Type 3778
System.PUnknown Type 3778
System.PUTF8String Type 3779
System.PVarArray Type 3779
System.PVarArrayBoundArray Type 3779
System.PVarArrayCoorArray Type 3780
System.PVarData Type 3780
System.PVariant Type 3780
System::PVariantManager Type 3781
System::PVarRec Type 3781
System.PWideString Type 3782
System.PWord Type 3782
System.PWordBool Type 3782
System.RawByteString Type 3783
System::ResourceString Type 3783
System.TAssertErrorProc Type 3783
System::TBooleanDynArray Type 3784

System.TBoundArray Type 3784
System::TByteDynArray Type 3784
System::TCardinalDynArray Type 3785
System.TClass Type 3785
System.TClassHelperBaseClass Type 3786
System.TDate Type 3786
System.TDateTime Type 3787
System.TDLLProc Type 3788
System.TDLLProcEx Type 3788
System::TDoubleDynArray Type 3788
System.TEnumModuleFunc Type 3789
System.TEnumModuleFuncLW Type 3789
System.TFileIOFunc Type 3789
System.THandle Type 3790
System::TInt64DynArray Type 3790
System::TIntegerDynArray Type 3790
System.TInterfacedClass Type 3791
System::TLongWordDynArray Type 3791
System.TMemoryMap Type 3791
System.TModuleUnloadProc Type 3792
System.TModuleUnloadProcLW Type 3793
System.TPCharArray Type 3793
System.TResourceHandle Type 3793
System.TSafeCallErrorProc Type 3794
System::TShortIntDynArray Type 3794
System::TSingleDynArray Type 3794
System::TSmallBlockTypeStates Type 3795
System::TSmallIntDynArray Type 3795
System::TStringDynArray Type 3796
System.TSystemThreadEndProc Type 3796
System.TSystemThreadFuncProc Type 3796
System.TTextBuf Type 3797
System.TTextIOFunc Type 3797
System::TThreadFunc Type 3797
System.TTime Type 3798
System.TUCS4CharArray Type 3798
System.TVarArrayBoundArray Type 3798
System.TVarArrayCoorArray Type 3799
System.TVarOp Type 3799

System.TVarType Type 3801
System::TWideStringDynArray Type 3801
System::TWordDynArray Type 3801
System.UCS2Char Type 3802
System.UCS4Char Type 3802
System.UCS4String Type 3802
System.UInt16 Type 3803
System.UInt32 Type 3803
System.UInt8 Type 3804
System.UnitEntryTable Type 3804
System.UTF8String Type 3804

Variables **3806**

System.AbstractErrorProc Variable 3807
System.AllocMemCount Variable 3807
System.AllocMemSize Variable 3809
System.AssertErrorProc Variable 3810
System.CmdLine Variable 3812
System.CmdShow Variable 3812
System.CPUCount Variable 3813
System.DebugHook Variable 3813
System.Default8087CW Variable 3813
System.DefaultSystemCodePage Variable 3815
System.DefaultTextLineBreakStyle Variable 3815
System.DefaultUnicodeCodePage Variable 3816
System.DispCallByIDProc Variable 3816
System.DLLProc Variable 3816
System.envp Variable 3817
System.ErrorAddr Variable 3817
System.ErrorProc Variable 3818
System.ErrOutput Variable 3820
System.ExceptClsProc Variable 3820
System.ExceptionAcquired Variable 3821
System.ExceptionClass Variable 3821
System.ExceptObjProc Variable 3821
System.ExceptProc Variable 3822
System.ExitCode Variable 3823
System.ExitProc Variable 3823
System.ExitProcessProc Variable 3824
System.FileMode Variable 3824

System.HeapAllocFlags Variable 3825
System.HPrevInst Variable 3826
System.InitProc Variable 3826
System.Input Variable 3827
System.IsConsole Variable 3828
System.IsLibrary Variable 3828
System.IsMultiThread Variable 3829
System.JITEnable Variable 3829
System.LibModuleList Variable 3830
System.MainInstance Variable 3830
System.MainThreadID Variable 3831
System.ModuleUnloadList Variable 3831
System.MonitorSupport Variable 3831
System.NeverSleepOnMMThreadContention Variable 3832
System.NoErrMsg Variable 3832
System.Output Variable 3833
System.RaiseExceptionProc Variable 3833
System.RaiseExceptObjProc Variable 3834
System.RandSeed Variable 3834
System.ReportMemoryLeaksOnShutdown Variable 3835
System.RTLUnwindProc Variable 3835
System.SafeCallErrorProc Variable 3836
System.SystemThreadEndProc Variable 3836
System.SystemThreadFuncProc Variable 3837
System.Test8086 Variable 3837
System.Test8087 Variable 3837
System.TestFDIV Variable 3838
System.UTF8CompareLocale Variable 3838
System.VarAddRefProc Variable 3838
System.VarClearProc Variable 3839
System.VarCopyProc Variable 3839
System.VarToLStrProc Variable 3840
System.VarToWStrProc Variable 3840

Constants **3842**

System.CompilerVersion Constant 3844
System.CPUi386 Constant 3844
System.CPUi486 Constant 3845
System.CPUPentium Constant 3845
System.E_NOINTERFACE Constant 3845

System.E_NOTIMPL Constant 3846
System.E_UNEXPECTED Constant 3846
System.fmClosed Constant 3846
System.fmInOut Constant 3847
System.fmInput Constant 3847
System.fmOutput Constant 3847
System.hfFieldSize Constant 3848
System.hfMonitorOffset Constant 3848
System.INFINITE Constant 3848
System.MaxInt Constant 3849
System.MaxLongint Constant 3849
System.NumSmallBlockTypes Constant 3849
System.opAdd Constant 3850
System.opAnd Constant 3850
System.opCmpEQ Constant 3851
System.opCmpGE Constant 3851
System.opCmpGT Constant 3852
System.opCmpLE Constant 3852
System.opCmpLT Constant 3853
System.opCmpNE Constant 3853
System.opCompare Constant 3854
System.opDivide Constant 3854
System.opIntDivide Constant 3855
System.opModulus Constant 3855
System.opMultiply Constant 3855
System.opNegate Constant 3856
System.opNot Constant 3856
System.opOr Constant 3857
System.opShiftLeft Constant 3857
System.opShiftRight Constant 3858
System.opSubtract Constant 3858
System.opXor Constant 3859
System.RTLVersion Constant 3859
System.S_FALSE Constant 3859
System.S_OK Constant 3860
System.sLineBreak Constant 3860
System.tfCRLF Constant 3861
System.varAny Constant 3861
System.varArray Constant 3861

System.varBoolean Constant 3862
System.varByRef Constant 3862
System.varByte Constant 3863
System.varCurrency Constant 3863
System.varDate Constant 3864
System.varDispatch Constant 3864
System.varDouble Constant 3865
System.varEmpty Constant 3865
System.varError Constant 3866
System.varInt64 Constant 3867
System.varInteger Constant 3867
System.varLongWord Constant 3868
System.varNull Constant 3868
System.varOleStr Constant 3869
System.varShortInt Constant 3869
System.varSingle Constant 3870
System.varSmallint Constant 3870
System.varStrArg Constant 3871
System.varString Constant 3871
System.varTypeMask Constant 3872
System.varUInt64 Constant 3872
System.varUnknown Constant 3873
System.varUString Constant 3873
System.varVariant Constant 3873
System.varWord Constant 3874
System.vmtAddRef Constant 3874
System.vmtAfterConstruction Constant 3875
System.vmtAutoTable Constant 3875
System.vmtBeforeDestruction Constant 3875
System.vmtClassName Constant 3876
System.vmtCreateObject Constant 3876
System.vmtDefaultHandler Constant 3876
System.vmtDestroy Constant 3877
System.vmtDispatch Constant 3877
System.vmtDynamicTable Constant 3877
System.vmtEquals Constant 3878
System.vmtFieldTable Constant 3878
System.vmtFreeInstance Constant 3878
System.vmtGetHashCode Constant 3879

System.vmtInitTable Constant 3879
System.vmtInstanceSize Constant 3879
System.vmtIntfTable Constant 3880
System.vmtMethodTable Constant 3880
System.vmtNewInstance Constant 3880
System.vmtParent Constant 3881
System.vmtQueryInterface Constant 3881
System.vmtRelease Constant 3881
System.vmtSafeCallException Constant 3882
System.vmtSelfPtr Constant 3882
System.vmtToString Constant 3882
System.vmtTypeInfo Constant 3883
System.vtAnsiString Constant 3883
System.vtBoolean Constant 3883
System.vtChar Constant 3884
System.vtClass Constant 3884
System.vtCurrency Constant 3885
System.vtExtended Constant 3885
System.vtInt64 Constant 3886
System.vtInteger Constant 3886
System.vtInterface Constant 3887
System.vtObject Constant 3887
System.vtPChar Constant 3888
System.vtPointer Constant 3888
System.vtPWideChar Constant 3889
System.vtString Constant 3889
System.vtUnicodeString Constant 3890
System.vtVariant Constant 3890
System.vtWideChar Constant 3890
System.vtWideString Constant 3891

Macros **3892**

System::USE Macro 3892
System::USEADDITIONALFILES Macro 3892
System::USEASM Macro 3893
System::USEDATAMODULE Macro 3893
System::USEDATAMODULENS Macro 3893
System::USEDEF Macro 3894
System::USEFILE Macro 3894
System::USEFORM Macro 3895

System::USEFORMNS Macro 3895
System::USEFORMRES Macro 3895
System::USEIDL Macro 3896
System::USELIB Macro 3896
System::USEOBJ Macro 3896
System::USEPACKAGE Macro 3897
System::USERC Macro 3897
System::USERES Macro 3897
System::USETLB Macro 3898
System::USEUNIT Macro 3898

Sysutils Namespace 3899

Functions 3900
Sysutils::IntToStr Function 3900
Sysutils::IntToStr Function 3900

SysUtils Namespace 3901

Classes 3915
EAbort Class 3916
EAbstractError Class 3916
EAccessViolation Class 3917
EArgumentException Class 3918
EArgumentOutOfRangeException Class 3918
EAssertionFailed Class 3918
EControlC Class 3919
EConvertError Class 3919
EDivByZero Class 3921
EEncodingError Class 3922
EExternal Class 3922
EExternal Members 3923
EExternal Fields 3925
EExternalException Class 3925
EHeapException Class 3926
EHeapException Members 3927
EHeapException Methods 3928
EInOutError Class 3929
EInOutError Members 3931
EInOutError Fields 3932

EIntError Class 3933
EIntfCastError Class 3934
EIntOverflow Class 3934
EInvalidCast Class 3935
EInvalidContainer Class 3935
EInvalidInsert Class 3936
EInvalidOp Class 3936
EInvalidPointer Class 3937
EMathError Class 3937
EMonitor Class 3938
EMonitorLockException Class 3938
ENoConstructException Class 3939
ENoMonitorSupportException Class 3939
EOSError Class 3939
EOSError Members 3940
EOSError Fields 3941
EOutOfMemory Class 3942
EOverflow Class 3942
EPackageError Class 3943
EPrivilege Class 3943
EProgrammerNotFound Class 3944
EPropReadOnly Class 3944
EPropWriteOnly Class 3945
ERangeError Class 3945
ESafecallException Class 3946
EStackOverflow Class 3947
EUnderflow Class 3947
EVariantError Class 3948
EWin32Error Class 3949
Exception Class 3949
Exception Members 3950
Exception.Create Constructor 3952
Exception.CreateFmt Constructor 3954
Exception.CreateFmtHelp Constructor 3955
Exception.CreateHelp Constructor 3955
CreateRes Constructor 3956
CreateResFmt Constructor 3957
CreateResFmtHelp Constructor 3958
CreateResHelp Constructor 3959

Exception.Destroy Destructor 3960
Exception Fields 3960
Exception Methods 3961
Exception Properties 3964
EZeroDivide Class 3966
TBigEndianUnicodeEncoding Class 3967
TBigEndianUnicodeEncoding Members 3967
TBigEndianUnicodeEncoding Methods 3968
TEncoding Class 3969
TEncoding Members 3972
TEncoding Fields 3973
TEncoding Methods 3974
TEncoding Properties 3984
TLanguages Class 3987
TLanguages Members 3987
TLanguages.Create Constructor 3988
TLanguages Methods 3988
TLanguages Properties 3989
TMBCSEncoding Class 3992
TMBCSEncoding Members 3992
Create Constructor 3993
TMBCSEncoding Methods 3994
TMultiReadExclusiveWriteSynchronizer Class 3997
TMultiReadExclusiveWriteSynchronizer Members 3998
TMultiReadExclusiveWriteSynchronizer.Create Constructor 3999
TMultiReadExclusiveWriteSynchronizer.Destroy Destructor 3999
TMultiReadExclusiveWriteSynchronizer Methods 4000
TMultiReadExclusiveWriteSynchronizer Properties 4002
TSimpleRWSync Class 4002
TSimpleRWSync Members 4003
TSimpleRWSync.Create Constructor 4004
TSimpleRWSync.Destroy Destructor 4004
TSimpleRWSync Methods 4005
TStringBuilder Class 4006
TStringBuilder Members 4010
Create Constructor 4010
TStringBuilder Fields 4011
TStringBuilder Methods 4012
TStringBuilder Properties 4021

TThreadLocalCounter Class 4023
TThreadLocalCounter Members 4023
TThreadLocalCounter.Destroy Destructor 4023
TThreadLocalCounter Methods 4024
TUnicodeEncoding Class 4024
TUnicodeEncoding Members 4025
TUnicodeEncoding.Create Constructor 4026
TUnicodeEncoding Methods 4026
TUTF7Encoding Class 4029
TUTF7Encoding Members 4029
TUTF7Encoding.Create Constructor 4030
TUTF7Encoding Methods 4031
TUTF8Encoding Class 4034
TUTF8Encoding Members 4034
TUTF8Encoding.Create Constructor 4035
TUTF8Encoding Methods 4035

Interfaces 4037
IReadWriteSync Interface 4037
IReadWriteSync Members 4037
IReadWriteSync Methods 4037

Functions 4039
SysUtils.Abort Function 4047
SysUtils.AddExitProc Function 4048
SysUtils.AddTerminateProc Function 4048
SysUtils.AdjustLineBreaks Function 4049
SysUtils.AnsiCompareFileName Function 4049
SysUtils.AnsiCompareStr Function 4050
SysUtils.AnsiCompareText Function 4051
SysUtils.AnsiDequotedStr Function 4051
SysUtils.AnsiExtractQuotedStr Function 4052
SysUtils.AnsiLastChar Function 4052
SysUtils.AnsiLowerCase Function 4053
SysUtils.AnsiLowerCaseFileName Function 4053
SysUtils.AnsiPos Function 4054
SysUtils.AnsiQuotedStr Function 4054
SysUtils.AnsiSameStr Function 4055
SysUtils.AnsiSameText Function 4055
SysUtils.AnsiStrAlloc Function 4056

SysUtils.AnsiStrComp Function 4056
SysUtils.AnsiStrIComp Function 4057
SysUtils.AnsiStrLastChar Function 4057
SysUtils.AnsiStrLComp Function 4058
SysUtils.AnsiStrLIComp Function 4058
SysUtils.AnsiStrLower Function 4059
SysUtils.AnsiStrPos Function 4060
SysUtils.AnsiStrRScan Function 4060
SysUtils.AnsiStrScan Function 4061
SysUtils.AnsiStrUpper Function 4061
SysUtils.AnsiUpperCase Function 4062
SysUtils.AnsiUpperCaseFileName Function 4062
SysUtils.AppendStr Function 4063
SysUtils.AssignStr Function 4063
SysUtils.Beep Function 4063
SysUtils.BoolToStr Function 4066
SysUtils.ByteLength Function 4068
SysUtils.BytesOf Function 4068
SysUtils.ByteToCharIndex Function 4069
SysUtils.ByteToCharLen Function 4070
SysUtils.ByteType Function 4070
SysUtils.CallTerminateProcs Function 4071
SysUtils.ChangeFileExt Function 4071
SysUtils.ChangeFilePath Function 4073
SysUtils.CharInSet Function 4073
SysUtils.CharLength Function 4074
SysUtils.CharToByteIndex Function 4075
SysUtils.CharToByteLen Function 4076
SysUtils.CharToElementIndex Function 4076
SysUtils.CharToElementIndex Function 4076
SysUtils.CharToElementLen Function 4077
SysUtils.CharToElementLen Function 4077
SysUtils.CheckWin32Version Function 4077
SysUtils.CompareMem Function 4078
SysUtils.CompareStr Function 4078
SysUtils.CompareText Function 4079
SysUtils.CreateDir Function 4083
SysUtils.CreateGUID Function 4084
SysUtils.CurrentYear Function 4086

SysUtils.CurrToStr Function 4086
SysUtils.CurrToStrF Function 4086
SysUtils.Date Function 4087
SysUtils.DateTimeToFileDate Function 4088
SysUtils.DateTimeToStr Function 4089
SysUtils.DateTimeToString Function 4089
SysUtils.DateTimeToSystemTime Function 4091
SysUtils.DateTimeToTimeStamp Function 4092
SysUtils.DateToStr Function 4092
SysUtils.DayOfWeek Function 4094
SysUtils.DecodeDate Function 4095
SysUtils.DecodeDateFully Function 4096
SysUtils.DecodeTime Function 4097
SysUtils.DelegatesEqual Function 4098
SysUtils.DeleteFile Function 4098
SysUtils.DirectoryExists Function 4100
SysUtils.DiskFree Function 4101
SysUtils.DiskSize Function 4102
SysUtils.DisposeStr Function 4103
SysUtils.ElementToCharIndex Function 4104
SysUtils.ElementToCharIndex Function 4104
SysUtils.ElementToCharLen Function 4104
SysUtils.ElementToCharLen Function 4105
SysUtils.EncodeDate Function 4105
SysUtils.EncodeTime Function 4106
SysUtils.ExceptionErrorMessage Function 4107
SysUtils.ExcludeTrailingBackslash Function 4108
SysUtils.ExcludeTrailingPathDelimiter Function 4109
SysUtils.ExpandFileName Function 4109
SysUtils.ExpandFileNameCase Function 4110
SysUtils.ExpandUNCFileName Function 4111
SysUtils.ExtractFileDir Function 4112
SysUtils.ExtractFileDrive Function 4112
SysUtils.ExtractFileExt Function 4113
SysUtils.ExtractFileName Function 4113
SysUtils.ExtractFilePath Function 4115
SysUtils.ExtractRelativePath Function 4117
SysUtils.ExtractShortPathName Function 4118
SysUtils.FileAge Function 4119

SysUtils.FileClose Function 4121
SysUtils.FileCreate Function 4128
SysUtils.FileDateToDateTime Function 4130
SysUtils.FileExists Function 4133
SysUtils.FileGetAttr Function 4139
SysUtils.FileGetDate Function 4142
SysUtils.FileIsReadOnly Function 4142
SysUtils.FileOpen Function 4143
SysUtils.FileRead Function 4150
SysUtils.FileSearch Function 4156
SysUtils.FileSeek Function 4157
SysUtils.FileSetAttr Function 4162
SysUtils.FileSetDate Function 4164
SysUtils.FileSetReadOnly Function 4165
SysUtils.FileWrite Function 4166
SysUtils.FinalizePackage Function 4172
SysUtils.FindClose Function 4172
SysUtils.FindCmdLineSwitch Function 4174
SysUtils.FindFirst Function 4175
SysUtils.FindNext Function 4177
SysUtils.FloatToCurr Function 4179
SysUtils.FloatToDateTime Function 4179
SysUtils.FloatToDecimal Function 4180
SysUtils.FloatToStr Function 4180
SysUtils.FloatToStrF Function 4182
SysUtils.FloatToText Function 4183
SysUtils.FloatToTextFmt Function 4184
SysUtils.FmtLoadStr Function 4185
SysUtils.FmtStr Function 4185
SysUtils.ForceDirectories Function 4186
SysUtils.Format Function 4188
SysUtils.FormatBuf Function 4193
SysUtils.FormatCurr Function 4194
SysUtils.FormatDateTime Function 4195
SysUtils.FormatFloat Function 4197
SysUtils.FreeAndNil Function 4201
SysUtils.GDAL Function 4202
SysUtils.GetCurrentDir Function 4202
SysUtils.GetEnvironmentVariable Function 4204

SysUtils.GetFileVersion Function 4205
SysUtils.GetFormatSettings Function 4206
SysUtils.GetLocaleChar Function 4206
SysUtils.GetLocaleFormatSettings Function 4206
SysUtils.GetLocaleStr Function 4207
SysUtils.GetModuleName Function 4207
SysUtils.GetPackageDescription Function 4208
SysUtils.GetPackageInfo Function 4209
SysUtils.GetTime Function 4209
SysUtils.GUIDToString Function 4210
SysUtils.HashName Function 4212
SysUtils.IncAMonth Function 4212
SysUtils.IncludeTrailingBackslash Function 4213
SysUtils.IncludeTrailingPathDelimiter Function 4213
SysUtils.IncMonth Function 4214
SysUtils.InitializePackage Function 4215
SysUtils.IntToHex Function 4215
SysUtils.IntToStr Function 4216
SysUtils.IsAssembly Function 4217
SysUtils.IsDelimiter Function 4218
SysUtils.IsEqualGUID Function 4218
SysUtils.IsLeadChar Function 4220
SysUtils.IsLeadChar Function 4220
SysUtils.IsLeapYear Function 4220
SysUtils.IsPathDelimiter Function 4222
SysUtils.IsValidIdent Function 4222
SysUtils.Languages Function 4223
SysUtils.LastDelimiter Function 4223
SysUtils.LoadPackage Function 4224
SysUtils.LoadStr Function 4224
SysUtils.LowerCase Function 4225
SysUtils.MSecsToTimeStamp Function 4226
SysUtils.NewStr Function 4227
SysUtils.NextCharIndex Function 4227
SysUtils.Now Function 4227
SysUtils.OutOfMemoryError Function 4228
SysUtils.PlatformBytesOf Function 4229
SysUtils.PlatformStringOf Function 4229
SysUtils.QuotedStr Function 4229

SysUtils.RaiseLastOSError Function 4230
SysUtils.RaiseLastWin32Error Function 4231
SysUtils.RCS Function 4232
SysUtils.RemoveDir Function 4232
SysUtils.RenameFile Function 4233
SysUtils.ReplaceDate Function 4236
SysUtils.ReplaceTime Function 4236
SysUtils.RPR Function 4237
SysUtils.SafeLoadLibrary Function 4238
SysUtils.SameFileName Function 4238
SysUtils.SameStr Function 4239
SysUtils.SameText Function 4239
SysUtils.SetCurrentDir Function 4240
SysUtils.ShowException Function 4240
SysUtils.Sleep Function 4241
SysUtils.StrAlloc Function 4241
SysUtils.StrBufSize Function 4242
SysUtils.StrByteType Function 4242
SysUtils.StrCat Function 4243
SysUtils.StrCharLength Function 4244
SysUtils.StrComp Function 4244
SysUtils.StrCopy Function 4246
SysUtils.StrDispose Function 4247
SysUtils.StrECopy Function 4248
SysUtils.StrEnd Function 4249
SysUtils.StrFmt Function 4250
SysUtils.StrIComp Function 4251
SysUtils.StringOf Function 4253
SysUtils.StringReplace Function 4253
SysUtils.StringToGUID Function 4254
SysUtils.StrLCat Function 4255
SysUtils.StrLComp Function 4257
SysUtils.StrLCopy Function 4258
SysUtils.StrLen Function 4260
SysUtils.StrLFmt Function 4261
SysUtils.StrLIComp Function 4262
SysUtils.StrLower Function 4264
SysUtils.StrMove Function 4265
SysUtils.StrNew Function 4266

SysUtils.StrNextChar Function 4268
SysUtils.StrPas Function 4268
SysUtils.StrPas Function 4269
SysUtils.StrPCopy Function 4271
SysUtils.StrPLCopy Function 4272
SysUtils.StrPos Function 4272
SysUtils.StrPos Function 4273
SysUtils.StrRScan Function 4274
SysUtils.StrScan Function 4275
SysUtils.StrToBool Function 4276
SysUtils.StrToBoolDef Function 4277
SysUtils.StrToCurr Function 4278
SysUtils.StrToCurrDef Function 4278
SysUtils.StrToDate Function 4279
SysUtils.StrToDateDef Function 4280
SysUtils.StrToDateTime Function 4281
SysUtils.StrToDateTimeDef Function 4282
SysUtils.StrToFloat Function 4282
SysUtils.StrToFloatDef Function 4286
SysUtils.StrToInt Function 4286
SysUtils.StrToInt64 Function 4287
SysUtils.StrToInt64Def Function 4288
SysUtils.StrToIntDef Function 4288
SysUtils.StrToTime Function 4291
SysUtils.StrToTimeDef Function 4292
SysUtils.StrUpper Function 4292
SysUtils.Supports Function 4294
SysUtils.SysErrorMessage Function 4294
SysUtils.SystemTimeToDateTime Function 4295
SysUtils.TextPos Function 4296
SysUtils.TextPos Function 4296
SysUtils.TextToFloat Function 4296
SysUtils.Time Function 4297
SysUtils.TimeStampToDateTime Function 4298
SysUtils.TimeStampToMSecs Function 4299
SysUtils.TimeToStr Function 4299
SysUtils.Trim Function 4301
SysUtils.TrimLeft Function 4301
SysUtils.TrimRight Function 4302

SysUtils.TryEncodeDate Function 4302
SysUtils.TryEncodeTime Function 4303
SysUtils.TryFloatToCurr Function 4304
SysUtils.TryFloatToDateTime Function 4304
SysUtils.TryStrToBool Function 4305
SysUtils.TryStrToCurr Function 4305
SysUtils.TryStrToDate Function 4306
SysUtils.TryStrToDateTime Function 4307
SysUtils.TryStrToFloat Function 4307
SysUtils.TryStrToInt Function 4308
SysUtils.TryStrToInt64 Function 4309
SysUtils.TryStrToTime Function 4309
SysUtils.UIntToStr Function 4310
SysUtils.UIntToStr Function 4310
SysUtils.UnloadPackage Function 4310
SysUtils.UpperCase Function 4311
SysUtils.WideBytesOf Function 4312
SysUtils.WideCompareStr Function 4313
SysUtils.WideCompareText Function 4313
SysUtils.WideFmtStr Function 4314
SysUtils.WideFormat Function 4314
SysUtils.WideFormatBuf Function 4315
SysUtils.WideLowerCase Function 4316
SysUtils.WideSameStr Function 4316
SysUtils.WideSameText Function 4316
SysUtils.WideStrAlloc Function 4317
SysUtils.WideStringOf Function 4317
SysUtils.WideUpperCase Function 4318
SysUtils.Win32Check Function 4318
SysUtils.WrapText Function 4318

Structs, Records, Enums **4320**

SysUtils.Int64Rec Record 4320
SysUtils.LongRec Record 4321
SysUtils.TCharSearch Record 4321
SysUtils.TExceptionRecord Record 4322
SysUtils.TFilenameCaseMatch Enumeration 4323
SysUtils.TFloatFormat Enumeration 4323
SysUtils.TFloatRec Record 4325
SysUtils.TFloatValue Enumeration 4325

SysUtils.TFormatSettings Record 4326
SysUtils.TLangRec Record 4327
SysUtils.TLocaleOptions Enumeration 4328
SysUtils.TMbcsByteType Enumeration 4329
SysUtils.TNameType Enumeration 4329
SysUtils.TSearchRec Record 4330
SysUtils.TSysLocale Record 4333
SysUtils.TThreadInfo Record 4334
SysUtils.TTimeStamp Record 4334
SysUtils.WordRec Record 4335

Types **4336**

SysUtils.ExceptClass Type 4336
SysUtils.PByteArray Type 4337
SysUtils.PDayTable Type 4337
SysUtils.PExceptionRecord Type 4338
SysUtils.PThreadInfo Type 4338
SysUtils.PWordArray Type 4338
SysUtils.TByteArray Type 4339
SysUtils.TBytes Type 4339
SysUtils.TCharArray Type 4340
SysUtils.TDayTable Type 4340
SysUtils.TFileName Type 4340
SysUtils.TFunc Type 4341
SysUtils.TIntegerSet Type 4341
SysUtils.TMREWSync Type 4341
SysUtils.TPackageInfoProc Type 4342
SysUtils.TPredicate Type 4342
SysUtils.TProc Type 4343
SysUtils.TProcedure Type 4343
SysUtils.TReplaceFlags Type 4343
SysUtils.TSysCharSet Type 4344
SysUtils.TTerminateProc Type 4344
SysUtils.TValidatePackageProc Type 4345
SysUtils.TWordArray Type 4345

Variables **4347**

SysUtils.CurrencyDecimals Variable 4348
SysUtils.CurrencyFormat Variable 4348
SysUtils.CurrencyString Variable 4349

SysUtils.DateSeparator Variable 4349
SysUtils.DecimalSeparator Variable 4350
SysUtils.EmptyAnsiStr Variable 4351
SysUtils.EmptyStr Variable 4351
SysUtils.EmptyWideStr Variable 4351
SysUtils.EraNames Variable 4352
SysUtils.EraYearOffsets Variable 4352
SysUtils.FalseBoolStrs Variable 4352
SysUtils.GetDiskFreeSpaceEx Variable 4353
SysUtils.HexDisplayPrefix Variable 4353
SysUtils.LeadBytes Variable 4354
SysUtils.ListSeparator Variable 4354
SysUtils.LongDateFormat Variable 4354
SysUtils.LongDayNames Variable 4355
SysUtils.LongMonthNames Variable 4355
SysUtils.LongTimeFormat Variable 4356
SysUtils.NegCurrFormat Variable 4356
SysUtils.NullAnsiStr Variable 4357
SysUtils.NullStr Variable 4357
SysUtils.NullWideStr Variable 4358
SysUtils.ShortDateFormat Variable 4358
SysUtils.ShortDayNames Variable 4359
SysUtils.ShortMonthNames Variable 4359
SysUtils.ShortTimeFormat Variable 4360
SysUtils.SysLocale Variable 4360
SysUtils.ThousandSeparator Variable 4362
SysUtils.TimeAMString Variable 4362
SysUtils.TimePMString Variable 4362
SysUtils.TimeSeparator Variable 4363
SysUtils.TrueBoolStrs Variable 4363
SysUtils.TwoDigitYearCenturyWindow Variable 4364
SysUtils.Win32BuildNumber Variable 4364
SysUtils.Win32CSDVersion Variable 4365
SysUtils.Win32MajorVersion Variable 4365
SysUtils.Win32MinorVersion Variable 4366
SysUtils.Win32Platform Variable 4366

Constants **4368**

SysUtils.DateDelta Constant 4369
SysUtils.DefaultFalseBoolStr Constant 4370

SysUtils.DefaultTrueBoolStr Constant 4370
SysUtils.DriveDelim Constant 4371
SysUtils.faAnyFile Constant 4371
SysUtils.faArchive Constant 4372
SysUtils.faDirectory Constant 4372
SysUtils.faHidden Constant 4373
SysUtils.faNormal Constant 4374
SysUtils.faReadOnly Constant 4374
SysUtils.faSymLink Constant 4375
SysUtils.faSysFile Constant 4375
SysUtils.faTemporary Constant 4376
SysUtils.faVolumeID Constant 4376
SysUtils.fmOpenRead Constant 4377
SysUtils.fmOpenReadWrite Constant 4378
SysUtils.fmOpenWrite Constant 4378
SysUtils.fmShareCompat Constant 4379
SysUtils.fmShareDenyNone Constant 4380
SysUtils.fmShareDenyRead Constant 4380
SysUtils.fmShareDenyWrite Constant 4381
SysUtils.fmShareExclusive Constant 4382
SysUtils.HoursPerDay Constant 4382
SysUtils.MaxCurrency Constant 4383
SysUtils.MaxDateTime Constant 4383
SysUtils.MaxEraCount Constant 4384
SysUtils.MinCurrency Constant 4384
SysUtils.MinDateTime Constant 4384
SysUtils.MinsPerDay Constant 4385
SysUtils.MinsPerHour Constant 4386
SysUtils.MonthDays Constant 4386
SysUtils.MSecsPerDay Constant 4387
SysUtils.MSecsPerSec Constant 4387
SysUtils.PathDelim Constant 4388
SysUtils.PathSep Constant 4388
SysUtils.pfBCB4Produced Constant 4389
SysUtils.pfConsumerBCB Constant 4389
SysUtils.pfConsumerCompat Constant 4390
SysUtils.pfConsumerDelphi Constant 4390
SysUtils.pfConsumerMask Constant 4391
SysUtils.pfDelphi4Produced Constant 4391

SysUtils.pfDesignOnly Constant 4392
SysUtils.pfExeModule Constant 4392
SysUtils.pfIgnoreDupUnits Constant 4392
SysUtils.pfLibraryModule Constant 4393
SysUtils.pfModuleTypeMask Constant 4393
SysUtils.pfNeverBuild Constant 4394
SysUtils.pfPackageModule Constant 4394
SysUtils.pfProducerMask Constant 4395
SysUtils.pfProducerUndefined Constant 4395
SysUtils.pfRunOnly Constant 4396
SysUtils.pfV3Produced Constant 4396
SysUtils.SecsPerDay Constant 4397
SysUtils.SecsPerHour Constant 4397
SysUtils.SecsPerMin Constant 4398
SysUtils.SwitchChars Constant 4398
SysUtils.ufImplicitUnit Constant 4399
SysUtils.ufMainUnit Constant 4399
SysUtils.ufOrgWeakUnit Constant 4400
SysUtils.ufPackageUnit Constant 4400
SysUtils.ufWeakPackageUnit Constant 4400
SysUtils.ufWeakUnit Constant 4401
SysUtils.UnixDateDelta Constant 4401

TabNotBk Namespace 4403

Classes 4404
TTabbedNotebook Class 4404
TTabbedNotebook Members 4404
TTabbedNotebook.Create Constructor 4405
TTabbedNotebook.Destroy Destructor 4406
TTabbedNotebook Methods 4406
TTabbedNotebook Properties 4413
TTabbedNotebook Events 4435
TTabPage Class 4443
TTabPage Members 4444
TTabPage.Create Constructor 4444
TTabPage Methods 4445
TTabPage Properties 4445

Types 4454

TabNotBk.TPageChangeEvent Type 4454

Constants **4455**

TabNotBk.CM_TABFONTCHANGED Constant 4455

Tabs Namespace 4457

Classes **4458**

TScroller Class 4458

TScroller Members 4458

TScroller.Create Constructor 4459

TScroller.Destroy Destructor 4460

TScroller Methods 4460

TScroller Properties 4460

TScroller Events 4463

TTabList Class 4463

TTabList Members 4464

TTabList.Create Constructor 4464

TTabList Methods 4464

TTabSet Class 4470

TTabSet Members 4471

TTabSet.Create Constructor 4473

TTabSet.Destroy Destructor 4473

TTabSet Methods 4474

TTabSet Properties 4480

TTabSet Events 4509

Structs, Records, Enums **4538**

Tabs.TEdgePart Enumeration 4538

Tabs.TEdgeType Enumeration 4538

Tabs.TScrollBtn Enumeration 4539

Tabs.TScrollOrientation Enumeration 4539

Tabs.TTabSetTabStyle Enumeration 4540

Types **4541**

Tabs.TDrawTabEvent Type 4541

Tabs.TMeasureTabEvent Type 4541

Tabs.TTabChangeEvent Type 4542

Tabs.TTabSetDrawTabEvent Type 4542

Tabs.TTabStyle Type 4543

TConnect Namespace 4545

Classes **4546**

TLocalConnection Class 4546

TLocalConnection Members 4547

TLocalConnection.Create Constructor 4547

TLocalConnection.Destroy Destructor 4548

TLocalConnection Methods 4548

TLocalConnection Properties 4553

ThemedActnCtrls Namespace 4555

Classes **4556**

TThemedAddRemoveItem Class 4556

TThemedAddRemoveItem Members 4556

TThemedAddRemoveItem Methods 4557

TThemedButtonControl Class 4559

TThemedButtonControl Members 4559

TThemedButtonControl Methods 4560

TThemedCustomizePopup Class 4561

TThemedCustomizePopup Members 4562

TThemedCustomizePopup.Create Constructor 4562

TThemedCustomizePopup Methods 4562

TThemedDropDownButton Class 4563

TThemedDropDownButton Members 4563

TThemedDropDownButton Methods 4564

TThemedExpandBtn Class 4566

TThemedExpandBtn Members 4566

TThemedExpandBtn Methods 4566

TThemedMenuButton Class 4567

TThemedMenuButton Members 4567

TThemedMenuButton Methods 4568

TThemedMenuItem Class 4569

TThemedMenuItem Members 4569

TThemedMenuItem Methods 4570

TThemedPopupMenu Class 4573

TThemedPopupMenu Members 4573

TThemedPopupMenu.Create Constructor 4573

TThemedPopupMenu Methods 4574

TThemedToolScrollBtn Class 4575
TThemedToolScrollBtn Members 4575
TThemedToolScrollBtn Methods 4576

Themes Namespace 4577

Classes **4579**
TThemeServices Class 4579
TThemeServices Members 4579
TThemeServices.Create Constructor 4581
TThemeServices.Destroy Destructor 4581
TThemeServices Methods 4581
TThemeServices Properties 4591
TThemeServices Events 4592

Functions **4593**
Themes.ThemeControl Function 4593
Themes.ThemeServices Function 4593
Themes.UnthemedDesigner Function 4594

Structs, Records, Enums **4595**
Themes.TThemedButton Enumeration 4595
Themes.TThemedClock Enumeration 4598
Themes.TThemedComboBox Enumeration 4598
Themes.TThemedEdit Enumeration 4599
Themes.TThemedElement Enumeration 4600
Themes.TThemedElementDetails Record 4602
Themes.TThemedExplorerBar Enumeration 4603
Themes.TThemedHeader Enumeration 4604
Themes.TThemedListview Enumeration 4605
Themes.TThemedMenu Enumeration 4606
Themes.TThemedPage Enumeration 4607
Themes.TThemedProgress Enumeration 4608
Themes.TThemedRebar Enumeration 4609
Themes.TThemedScrollBar Enumeration 4610
Themes.TThemedSpin Enumeration 4612
Themes.TThemedStartPanel Enumeration 4613
Themes.TThemedStatus Enumeration 4614
Themes.TThemedTab Enumeration 4615
Themes.TThemedTaskBand Enumeration 4616
Themes.TThemedTaskBar Enumeration 4617

Themes.TThemedToolBar Enumeration 4618
Themes.TThemedToolTip Enumeration 4620
Themes.TThemedTrackBar Enumeration 4621
Themes.TThemedTrayNotify Enumeration 4622
Themes.TThemedTreeview Enumeration 4623
Themes.TThemedWindow Enumeration 4624

Types **4629**
Themes.PThemedElementDetails Type 4629
Themes.TThemeData Type 4629
Themes.TThemeServicesClass Type 4630

Variables **4631**
Themes.ThemeServicesClass Variable 4631

ToolWin Namespace 4633

Classes **4634**
TToolDockForm Class 4634
TToolDockForm Members 4634
TToolDockForm.Create Constructor 4635
TToolDockForm Methods 4635
TToolDockObject Class 4637
TToolDockObject Members 4638
TToolDockObject.Create Constructor 4638
TToolDockObject Methods 4638
TToolWindow Class 4639
TToolWindow Members 4640
TToolWindow.Create Constructor 4640
TToolWindow Methods 4641
TToolWindow Properties 4642

Structs, Records, Enums **4644**
ToolWin.TEdgeBorder Enumeration 4644
ToolWin.TEdgeStyle Enumeration 4645
ToolWin.TSizingOrientation Enumeration 4645

Types **4647**
ToolWin.TEdgeBorders Type 4647

Types Namespace 4649

Functions **4651**
Types.Bounds Function 4651
Types.CenteredRect Function 4652
Types.CenterPoint Function 4652
Types.EqualRect Function 4652
Types.IntersectRect Function 4653
Types.IsRectEmpty Function 4653
Types.OffsetRect Function 4654
Types.Point Function 4654
Types.PtInRect Function 4655
Types.Rect Function 4655
Types.RectHeight Function 4656
Types.RectWidth Function 4656
Types.SmallPoint Function 4656
Types.SplitRect Function 4657
Types.SplitRect Function 4657
Types.UnionRect Function 4658

Structs, Records, Enums **4659**
Types.tagSIZE Record 4659
Types.TPoint Record 4660
Types.TRect Record 4660
Types.TSmallPoint Record 4661
Types.TSplitRectType Enumeration 4662

Types **4663**
Types.DWORD Type 4663
Types.PByte Type 4664
Types.PDouble Type 4664
Types.PInteger Type 4664
Types.PLongint Type 4665
Types.PPoint Type 4665
Types.PRect Type 4666
Types.PSize Type 4666
Types.PSmallInt Type 4666
Types.PSmallPoint Type 4667
Types.SIZE Type 4667
Types.tagPOINT Type 4667
Types.TBooleanDynArray Type 4668
Types.TByteDynArray Type 4668

Types.TCardinalDynArray Type 4668
Types.TDoubleDynArray Type 4669
Types.TInt64DynArray Type 4669
Types.TIntegerDynArray Type 4670
Types.TLongWordDynArray Type 4670
Types.TShortIntDynArray Type 4670
Types.TSingleDynArray Type 4671
Types.TSize Type 4671
Types.TSmallIntDynArray Type 4672
Types.TStringDynArray Type 4672
Types.TValueRelationship Type 4673
Types.TWideStringDynArray Type 4673
Types.TWordDynArray Type 4674

Constants **4675**
Types.EqualsValue Constant 4675
Types.GreaterThanValue Constant 4675
Types.LessThanValue Constant 4676
Types.RT_RCDATA Constant 4676

TypeTrans Namespace 4679

Classes **4680**
ETypeTransException Class 4680
TTypeTranslator Class 4680
TTypeTranslator Members 4681
TTypeTranslator.Create Constructor 4681
TTypeTranslator.Destroy Destructor 4681
TTypeTranslator Methods 4681

Functions **4684**
TypeTrans.FloatToStrEx Function 4684
TypeTrans.GetEnumValueEx Function 4684
TypeTrans.GetEnumValueExW Function 4685
TypeTrans.SetEnumPropEx Function 4685
TypeTrans.SetEnumPropExW Function 4685
TypeTrans.StrToFloatEx Function 4686

Variables **4687**
TypeTrans.TypeTranslator Variable 4687

TypInfo Namespace 4689

Classes **4693**
EPropertyConvertError Class 4693
EPropertyError Class 4693
TPublishableVariantType Class 4694
TPublishableVariantType Members 4695
TPublishableVariantType Methods 4698

Functions **4701**
TypInfo.FreeAndNilProperties Function 4703
TypInfo.GetAnsiStrProp Function 4703
TypInfo.GetDynArrayProp Function 4704
TypInfo.GetEnumName Function 4705
TypInfo.GetEnumProp Function 4708
TypInfo.GetEnumValue Function 4710
TypInfo.GetFloatProp Function 4711
TypInfo.GetInt64Prop Function 4712
TypInfo.GetInterfaceProp Function 4712
TypInfo.GetMethodProp Function 4713
TypInfo.GetObjectProp Function 4714
TypInfo.GetObjectPropClass Function 4715
TypInfo.GetOrdProp Function 4716
TypInfo.GetPropInfo Function 4717
TypInfo.GetPropInfos Function 4717
TypInfo.GetPropList Function 4718
TypInfo.GetPropName Function 4719
TypInfo.GetPropValue Function 4719
TypInfo.GetSetElementName Function 4720
TypInfo.GetSetElementValue Function 4721
TypInfo.GetSetProp Function 4722
TypInfo.GetStrProp Function 4722
TypInfo.GetTypeData Function 4723
TypInfo.GetTypeName Function 4724
TypInfo.GetUnicodeStrProp Function 4724
TypInfo.GetVariantProp Function 4725
TypInfo.GetWideStrProp Function 4726
TypInfo.IsPublishedProp Function 4727
TypInfo.IsStoredProp Function 4727

TypInfo.PropIsType Function 4728
TypInfo.PropType Function 4729
TypInfo.SamePropTypeName Function 4729
TypInfo.SetAnsiStrProp Function 4730
TypInfo.SetDynArrayProp Function 4731
TypInfo.SetEnumProp Function 4732
TypInfo.SetFloatProp Function 4734
TypInfo.SetInt64Prop Function 4735
TypInfo.SetInterfaceProp Function 4736
TypInfo.SetMethodProp Function 4737
TypInfo.SetObjectProp Function 4737
TypInfo.SetOrdProp Function 4738
TypInfo.SetPropValue Function 4739
TypInfo.SetSetProp Function 4740
TypInfo.SetStrProp Function 4741
TypInfo.SetToString Function 4742
TypInfo.SetUnicodeStrProp Function 4742
TypInfo.SetVariantProp Function 4743
TypInfo.SetWideStrProp Function 4744
TypInfo.SortPropList Function 4745
TypInfo.StringToSet Function 4746

Structs, Records, Enums **4747**
TypInfo.TFloatType Enumeration 4747
TypInfo.TIntfFlag Enumeration 4751
TypInfo.TManagedField Record 4751
TypInfo.TMethodKind Enumeration 4751
TypInfo.TOrdType Enumeration 4753
TypInfo.TParamFlag Enumeration 4756
TypInfo.TPropData Record 4757
TypInfo.TPropInfo Record 4757
TypInfo.TTypeData Record 4758
TypInfo.TTypeInfo Record 4759
TypInfo.TTypeKind Enumeration 4760

Types **4762**
TypInfo.PPropInfo Type 4762
TypInfo.PPropList Type 4763
TypInfo.PPTypeInfo Type 4763
TypInfo.PTypeData Type 4764

TypInfo.PTypeInfo Type 4767
TypInfo.ShortStringBase Type 4770
TypInfo.TIntfFlags Type 4770
TypInfo.TIntfFlagsBase Type 4771
TypInfo.TParamFlags Type 4772
TypInfo.TParamFlagsBase Type 4772
TypInfo.TPropInfoProc Type 4772
TypInfo.TPropList Type 4773
TypInfo.TTypeKinds Type 4773

Variables **4774**
TypInfo.BooleanIdents Variable 4774
TypInfo.DotSep Variable 4774

Constants **4775**
TypInfo.tkAny Constant 4775
TypInfo.tkMethods Constant 4775
TypInfo.tkProperties Constant 4776

ValEdit Namespace 4777

Classes **4778**
TItemProp Class 4778
TItemProp Members 4778
TItemProp.Create Constructor 4779
TItemProp.Destroy Destructor 4779
TItemProp Methods 4780
TItemProp Properties 4781
TValueListEditor Class 4784
TValueListEditor Members 4785
TValueListEditor.Create Constructor 4788
TValueListEditor.Destroy Destructor 4788
TValueListEditor Methods 4788
TValueListEditor Properties 4800
TValueListEditor Events 4838
TValueListStrings Class 4882
TValueListStrings Members 4883
TValueListStrings.Create Constructor 4883
TValueListStrings.Destroy Destructor 4884
TValueListStrings Methods 4884
TValueListStrings Properties 4890

Structs, Records, Enums **4892**
ValEdit.TDisplayOption Enumeration 4892
ValEdit.TKeyOption Enumeration 4893

Types **4894**
ValEdit.TDisplayOptions Type 4894
ValEdit.TGetPickListEvent Type 4895
ValEdit.TItemProps Type 4895
ValEdit.TKeyOptions Type 4895
ValEdit.TOnValidateEvent Type 4896

VarCmplx Namespace 4899

Functions **4901**
VarCmplx.VarAsComplex Function 4902
VarCmplx.VarComplex Function 4903
VarCmplx.VarComplexAbs Function 4903
VarCmplx.VarComplexAbsSqr Function 4904
VarCmplx.VarComplexAngle Function 4905
VarCmplx.VarComplexArcCos Function 4905
VarCmplx.VarComplexArcCosH Function 4906
VarCmplx.VarComplexArcCot Function 4907
VarCmplx.VarComplexArcCotH Function 4908
VarCmplx.VarComplexArcCsc Function 4909
VarCmplx.VarComplexArcCscH Function 4911
VarCmplx.VarComplexArcSec Function 4912
VarCmplx.VarComplexArcSecH Function 4913
VarCmplx.VarComplexArcSin Function 4914
VarCmplx.VarComplexArcSinH Function 4915
VarCmplx.VarComplexArcTan Function 4916
VarCmplx.VarComplexArcTanH Function 4917
VarCmplx.VarComplexConjugate Function 4918
VarCmplx.VarComplexCos Function 4919
VarCmplx.VarComplexCosH Function 4920
VarCmplx.VarComplexCot Function 4921
VarCmplx.VarComplexCotH Function 4922
VarCmplx.VarComplexCreate Function 4923
VarCmplx.VarComplexCsc Function 4924
VarCmplx.VarComplexCscH Function 4925
VarCmplx.VarComplexExp Function 4926

VarCmplx.VarComplexFromPolar Function 4927
VarCmplx.VarComplexInverse Function 4927
VarCmplx.VarComplexLn Function 4928
VarCmplx.VarComplexLog10 Function 4929
VarCmplx.VarComplexLog2 Function 4929
VarCmplx.VarComplexLogN Function 4930
VarCmplx.VarComplexPower Function 4930
VarCmplx.VarComplexSec Function 4931
VarCmplx.VarComplexSecH Function 4932
VarCmplx.VarComplexSign Function 4933
VarCmplx.VarComplexSimplify Function 4934
VarCmplx.VarComplexSin Function 4934
VarCmplx.VarComplexSinH Function 4935
VarCmplx.VarComplexSqr Function 4936
VarCmplx.VarComplexSqrt Function 4937
VarCmplx.VarComplexTan Function 4938
VarCmplx.VarComplexTanH Function 4939
VarCmplx.VarComplexTimesImaginary Function 4940
VarCmplx.VarComplexTimesNegI Function 4940
VarCmplx.VarComplexTimesPosI Function 4941
VarCmplx.VarComplexTimesReal Function 4941
VarCmplx.VarComplexToPolar Function 4942
VarCmplx.VarIsComplex Function 4942

Variables **4944**

VarCmplx.ComplexNumberDefuzzAtZero Variable 4944
VarCmplx.ComplexNumberSymbol Variable 4944
VarCmplx.ComplexNumberSymbolBeforeImaginary Variable 4945

VarConv Namespace 4947

Functions **4948**

VarConv.VarAsConvert Function 4948
VarConv.VarConvert Function 4949
VarConv.VarConvertCreate Function 4950
VarConv.VarIsConvert Function 4951

VarHlpr Namespace 4953

Functions **4955**

VarHlpr.VariantAdd Function 4955
VarHlpr.VariantAdd2 Function 4956
VarHlpr.VariantAnd Function 4956
VarHlpr.VariantAnd2 Function 4957
VarHlpr.VariantArrayRedim Function 4957
VarHlpr.VariantCast Function 4958
VarHlpr.VariantClear Function 4958
VarHlpr.VariantCpy Function 4959
VarHlpr.VariantDiv Function 4959
VarHlpr.VariantDiv2 Function 4960
VarHlpr.VariantFromUnicodeString Function 4960
VarHlpr.VariantGetElement Function 4961
VarHlpr.VariantMod Function 4962
VarHlpr.VariantMod2 Function 4962
VarHlpr.VariantMul Function 4963
VarHlpr.VariantMul2 Function 4963
VarHlpr.VariantNeg Function 4964
VarHlpr.VariantNot Function 4964
VarHlpr.VariantOr Function 4964
VarHlpr.VariantOr2 Function 4965
VarHlpr.VariantPutElement Function 4966
VarHlpr.VariantShl Function 4966
VarHlpr.VariantShl2 Function 4967
VarHlpr.VariantShr Function 4967
VarHlpr.VariantShr2 Function 4968
VarHlpr.VariantSub Function 4968
VarHlpr.VariantSub2 Function 4969
VarHlpr.VariantToUnicodeString Function 4969
VarHlpr.VariantXor Function 4970
VarHlpr.VariantXor2 Function 4970

Variants Namespace 4973

Classes **4978**

EVariantArrayCreateError Class 4978
EVariantArrayLockedError Class 4979
EVariantBadIndexError Class 4980
EVariantBadVarTypeError Class 4980
EVariantDispatchError Class 4981

EVariantInvalidArgError Class 4981
EVariantInvalidNullOpError Class 4982
EVariantInvalidOpError Class 4983
EVariantNotAnArrayError Class 4983
EVariantNotImplError Class 4984
EVariantOutOfMemoryError Class 4984
EVariantOverflowError Class 4985
EVariantRangeCheckError Class 4986
EVariantTypeCastError Class 4986
EVariantUnexpectedError Class 4987
TCustomVariantType Class 4987
TCustomVariantType Members 4988
Create Constructor 4990
TCustomVariantType.Destroy Destructor 4991
TCustomVariantType Methods 4992
TCustomVariantType Properties 5013
TInvokeableVariantType Class 5014
TInvokeableVariantType Members 5015
TInvokeableVariantType Methods 5017

Interfaces 5022

IVarInstanceReference Interface 5022
IVarInstanceReference Members 5022
IVarInstanceReference Methods 5023
IVarInvokeable Interface 5023
IVarInvokeable Members 5024
IVarInvokeable Methods 5024

Functions 5027

Variants._DispInvoke Function 5029
Variants._OleVarFromInt Function 5030
Variants._OleVarFromInt64 Function 5030
Variants._OleVarFromLStr Function 5031
Variants._OleVarFromPStr Function 5031
Variants._OleVarFromUInt64 Function 5031
Variants._OleVarFromUStr Function 5032
Variants._OleVarFromVar Function 5032
Variants._VarAdd Function 5032
Variants._VarAddRef Function 5033
Variants._VarAnd Function 5033

Variants._VarArrayGet Function 5034
Variants._VarArrayPut Function 5034
Variants._VarArrayRedim Function 5034
Variants._VarCast Function 5035
Variants._VarCastOle Function 5035
Variants._VarClear Function 5035
Variants._VarClr Function 5036
Variants._VarCmpEQ Function 5036
Variants._VarCmpGE Function 5037
Variants._VarCmpGT Function 5037
Variants._VarCmpLE Function 5037
Variants._VarCmpLT Function 5038
Variants._VarCmpNE Function 5038
Variants._VarCopy Function 5038
Variants._VarCopyNoInd Function 5039
Variants._VarDiv Function 5039
Variants._VarFromBool Function 5040
Variants._VarFromCurr Function 5040
Variants._VarFromDisp Function 5040
Variants._VarFromDynArray Function 5041
Variants._VarFromInt Function 5041
Variants._VarFromInt64 Function 5041
Variants._VarFromIntf Function 5042
Variants._VarFromLStr Function 5042
Variants._VarFromPStr Function 5043
Variants._VarFromReal Function 5043
Variants._VarFromTDateTime Function 5043
Variants._VarFromUInt64 Function 5044
Variants._VarFromUStr Function 5044
Variants._VarFromWChar Function 5044
Variants._VarFromWStr Function 5045
Variants._VarMod Function 5045
Variants._VarMul Function 5046
Variants._VarNeg Function 5046
Variants._VarNot Function 5046
Variants._VarOr Function 5047
Variants._VarRDiv Function 5047
Variants._VarShl Function 5047
Variants._VarShr Function 5048

Variants._VarSub Function 5048
Variants._VarToBool Function 5049
Variants._VarToCurrency Function 5049
Variants._VarToDisp Function 5049
Variants._VarToDynArray Function 5050
Variants._VarToInt64 Function 5050
Variants._VarToInteger Function 5050
Variants._VarToIntf Function 5051
Variants._VarToLStr Function 5051
Variants._VarToPStr Function 5052
Variants._VarToReal Function 5052
Variants._VarToUInt64 Function 5052
Variants._VarToUStr Function 5053
Variants._VarToWStr Function 5053
Variants._VarXor Function 5053
Variants.DynArrayFromVariant Function 5054
Variants.DynArrayToVariant Function 5054
Variants.FindCustomVariantType Function 5055
Variants.FindVarData Function 5055
Variants.HandleConversionException Function 5056
Variants.Null Function 5056
Variants.Unassigned Function 5058
Variants.VarArrayAsPSafeArray Function 5059
Variants.VarArrayCreate Function 5059
Variants.VarArrayCreateError Function 5061
Variants.VarArrayDimCount Function 5061
Variants.VarArrayGet Function 5063
Variants.VarArrayHighBound Function 5065
Variants.VarArrayLock Function 5067
Variants.VarArrayLowBound Function 5067
Variants.VarArrayOf Function 5069
Variants.VarArrayPut Function 5069
Variants.VarArrayRef Function 5071
Variants.VarArrayUnlock Function 5072
Variants.VarAsError Function 5072
Variants.VarAsType Function 5073
Variants.VarCastError Function 5073
Variants.VarCheckEmpty Function 5074
Variants.VarCompareValue Function 5074

Variants.VarCopyNoInd Function 5075
Variants.VarEnsureRange Function 5076
Variants.VarFromDateTime Function 5076
Variants.VarInRange Function 5077
Variants.VarInvalidNullOp Function 5077
Variants.VarInvalidOp Function 5078
Variants.VarIsArray Function 5078
Variants.VarIsByRef Function 5079
Variants.VarIsClear Function 5079
Variants.VarIsCustom Function 5080
Variants.VarIsEmpty Function 5080
Variants.VarIsEmptyParam Function 5081
Variants.VarIsError Function 5082
Variants.VarIsFloat Function 5082
Variants.VarIsNull Function 5083
Variants.VarIsNumeric Function 5083
Variants.VarIsOrdinal Function 5084
Variants.VarIsStr Function 5084
Variants.VarIsType Function 5085
Variants.VarOverflowError Function 5085
Variants.VarRangeCheckError Function 5086
Variants.VarResultCheck Function 5086
Variants.VarSameValue Function 5087
Variants.VarSupports Function 5087
Variants.VarToDateTime Function 5088
Variants.VarToStr Function 5088
Variants.VarToStrDef Function 5089
Variants.VarToWideStr Function 5089
Variants.VarToWideStrDef Function 5090
Variants.VarType Function 5090
Variants.VarTypeAsText Function 5092
Variants.VarTypeIsValidArrayType Function 5092
Variants.VarTypeIsValidElementType Function 5093

Structs, Records, Enums **5094**

Variants.TBooleanToStringRule Enumeration 5094
Variants.TNullCompareRule Enumeration 5094
Variants.TVarCompareResult Enumeration 5095
Variants.TVariantRelationship Enumeration 5096

Types **5097**
Variants.TAnyProc Type 5097
Variants.TCustomVariantTypeClass Type 5097
Variants.TVarDataArray Type 5098
Variants.TVarDispProc Type 5098

Variables **5100**
Variants.BooleanToStringRule Variable 5100
Variants.ChangeAnyProc Variable 5101
Variants.ClearAnyProc Variable 5101
Variants.EmptyParam Variable 5102
Variants.NullAsStringValue Variable 5102
Variants.NullEqualityRule Variable 5103
Variants.NullMagnitudeRule Variable 5103
Variants.NullStrictConvert Variable 5104
Variants.OleVariantInt64AsDouble Variable 5105
Variants.PackVarCreation Variable 5105
Variants.RefAnyProc Variable 5106
Variants.VarDispProc Variable 5106

VarUtils Namespace 5107

Classes **5110**
ESafeArrayBoundsError Class 5110
ESafeArrayCreateError Class 5110
ESafeArrayError Class 5111
ESafeArrayError Members 5112
ESafeArrayError.CreateHResult Constructor 5113
ESafeArrayError Properties 5113
ESafeArrayLockedError Class 5114

Functions **5115**
VarUtils.SafeArrayAccessData Function 5115
VarUtils.SafeArrayAllocData Function 5116
VarUtils.SafeArrayAllocDescriptor Function 5116
VarUtils.SafeArrayCheck Function 5117
VarUtils.SafeArrayCopy Function 5117
VarUtils.SafeArrayCopyData Function 5118
VarUtils.SafeArrayCreate Function 5118
VarUtils.SafeArrayDestroy Function 5119

VarUtils.SafeArrayDestroyData Function 5120
VarUtils.SafeArrayDestroyDescriptor Function 5120
VarUtils.SafeArrayError Function 5120
VarUtils.SafeArrayGetDim Function 5121
VarUtils.SafeArrayGetElement Function 5121
VarUtils.SafeArrayGetElemSize Function 5122
VarUtils.SafeArrayGetLBound Function 5122
VarUtils.SafeArrayGetUBound Function 5123
VarUtils.SafeArrayLock Function 5123
VarUtils.SafeArrayPtrOfIndex Function 5124
VarUtils.SafeArrayPutElement Function 5125
VarUtils.SafeArrayRedim Function 5125
VarUtils.SafeArrayUnaccessData Function 5126
VarUtils.SafeArrayUnlock Function 5127
VarUtils.VarExceptionToResult Function 5127
VarUtils.VariantChangeType Function 5128
VarUtils.VariantClear Function 5128
VarUtils.VariantCopy Function 5129
VarUtils.VariantCopyInd Function 5130
VarUtils.VariantInit Function 5130

Structs, Records, Enums **5132**

VarUtils.TVarTypeToElementInfo Record 5132

Variables **5133**

VarUtils.VarAdd Variable 5133
VarUtils.VarAnd Variable 5134
VarUtils.VarBoolFromStr Variable 5134
VarUtils.VarBstrFromBool Variable 5135
VarUtils.VarBstrFromCy Variable 5136
VarUtils.VarBstrFromDate Variable 5137
VarUtils.VarCmp Variable 5138
VarUtils.VarCyFromStr Variable 5139
VarUtils.VarDateFromStr Variable 5139
VarUtils.VarDiv Variable 5140
VarUtils.VarI4FromStr Variable 5141
VarUtils.VariantChangeTypeEx Variable 5142
VarUtils.VarIDiv Variable 5142
VarUtils.VarMod Variable 5143
VarUtils.VarMul Variable 5143

VarUtils.VarNeg Variable 5144
VarUtils.VarNot Variable 5144
VarUtils.VarOr Variable 5145
VarUtils.VarR4FromStr Variable 5145
VarUtils.VarR8FromStr Variable 5146
VarUtils.VarSub Variable 5147
VarUtils.VarXor Variable 5147

Constants **5149**

VarUtils.ARR_DISPATCH Constant 5150
VarUtils.ARR_FIXEDSIZE Constant 5150
VarUtils.ARR_NONE Constant 5151
VarUtils.ARR_OLESTR Constant 5151
VarUtils.ARR_UNKNOWN Constant 5152
VarUtils.ARR_VARIANT Constant 5152
VarUtils.CMaxArrayVarType Constant 5153
VarUtils.CMinArrayVarType Constant 5153
VarUtils.CVarTypeToElementInfo Constant 5154
VarUtils.GenericOperations Constant 5155
VarUtils.GenericSafeArrays Constant 5155
VarUtils.GenericSafeArrayUsesLibC Constant 5156
VarUtils.GenericVariants Constant 5156
VarUtils.GenericVarUtils Constant 5157
VarUtils.VAR_ARRAYISLOCKED Constant 5157
VarUtils.VAR_BADINDEX Constant 5158
VarUtils.VAR_BADVARTYPE Constant 5158
VarUtils.VAR_CMP_EQ Constant 5158
VarUtils.VAR_CMP_GT Constant 5159
VarUtils.VAR_CMP_LT Constant 5159
VarUtils.VAR_CMP_NULL Constant 5160
VarUtils.VAR_EXCEPTION Constant 5160
VarUtils.VAR_INVALIDARG Constant 5161
VarUtils.VAR_LOCALE_USER_DEFAULT Constant 5161
VarUtils.VAR_NOTIMPL Constant 5161
VarUtils.VAR_OK Constant 5162
VarUtils.VAR_OUTOFMEMORY Constant 5162
VarUtils.VAR_OVERFLOW Constant 5163
VarUtils.VAR_PARAMNOTFOUND Constant 5163
VarUtils.VAR_TYPEMISMATCH Constant 5163
VarUtils.VAR_UNEXPECTED Constant 5164

VCLCom Namespace 5165

Classes **5166**

TComponentFactory Class 5166

TComponentFactory Members 5166

TComponentFactory.Create Constructor 5167

TComponentFactory Methods 5167

WebAdapt Namespace 5169

Classes **5175**

EAdapterException Class 5177

EAdapterException Members 5177

EAdapterException.CreateID Constructor 5178

EAdapterException Properties 5178

EAdapterFieldException Class 5178

EAdapterFieldException Members 5179

EAdapterFieldException.Create Constructor 5179

EAdapterFieldException Properties 5180

TAbstractFieldValuesAdapter Class 5180

TAbstractFieldValuesAdapter Members 5182

TAbstractFieldValuesAdapter Methods 5185

TAdapter Class 5190

TAdapter Members 5192

TAdapter Properties 5195

TAdapter Events 5197

TAdapterAction Class 5201

TAdapterAction Members 5202

TAdapterAction Properties 5205

TAdapterAction Events 5206

TAdapterActions Class 5209

TAdapterActions Members 5210

TAdapterActions Methods 5212

TAdapterActions Properties 5214

TAdapterApplicationField Class 5214

TAdapterApplicationField Members 5215

TAdapterApplicationField Properties 5218

TAdapterApplicationTitleField Class 5218

TAdapterApplicationTitleField Members 5219

TAdapterApplicationTitleField Methods 5222
TAdapterBooleanField Class 5223
TAdapterBooleanField Members 5224
TAdapterBooleanField Properties 5227
TAdapterBooleanField Events 5230
TAdapterDefaultActions Class 5231
TAdapterDefaultActions Members 5232
TAdapterDefaultActions.Destroy Destructor 5233
TAdapterDefaultActions Methods 5234
TAdapterDefaultActions Properties 5237
TAdapterDefaultFields Class 5237
TAdapterDefaultFields Members 5238
TAdapterDefaultFields.Destroy Destructor 5240
TAdapterDefaultFields Methods 5240
TAdapterDefaultFields Properties 5243
TAdapterEndUserDisplayNameField Class 5244
TAdapterEndUserDisplayNameField Members 5244
TAdapterEndUserDisplayNameField Methods 5247
TAdapterEndUserField Class 5248
TAdapterEndUserField Members 5249
TAdapterEndUserField Properties 5251
TAdapterEndUserLoggedInField Class 5252
TAdapterEndUserLoggedInField Members 5252
TAdapterEndUserLoggedInField Methods 5255
TAdapterError Class 5256
TAdapterError Members 5256
TAdapterError.Create Constructor 5258
TAdapterError Methods 5258
TAdapterErrorsList Class 5259
TAdapterErrorsList Members 5260
TAdapterErrorsList.Create Constructor 5262
TAdapterErrorsList.Destroy Destructor 5262
TAdapterErrorsList Methods 5263
TAdapterErrorsList Properties 5265
TAdapterField Class 5267
TAdapterField Members 5268
TAdapterField Properties 5271
TAdapterField Events 5275
TAdapterFields Class 5276

TAdapterFields Members 5277
TAdapterFields Methods 5278
TAdapterFields Properties 5280
TAdapterFileField Class 5281
TAdapterFileField Members 5283
TAdapterFileField Properties 5287
TAdapterFileField Events 5289
TAdapterGotoPageAction Class 5291
TAdapterGotoPageAction Members 5292
TAdapterGotoPageAction Properties 5295
TAdapterGotoPageAction Events 5296
TAdapterHiddenField Class 5299
TAdapterHiddenField Members 5299
TAdapterHiddenField.Create Constructor 5301
TAdapterHiddenField Methods 5301
TAdapterHiddenFieldsList Class 5301
TAdapterHiddenFieldsList Members 5302
TAdapterHiddenFieldsList.Create Constructor 5304
TAdapterHiddenFieldsList.Destroy Destructor 5304
TAdapterHiddenFieldsList Methods 5304
TAdapterHiddenFieldsList Properties 5307
TAdapterImageField Class 5308
TAdapterImageField Members 5310
TAdapterImageField Properties 5314
TAdapterImageField Events 5317
TAdapterLoginField Class 5320
TAdapterLoginField Members 5321
TAdapterLoginField Properties 5323
TAdapterMemoField Class 5323
TAdapterMemoField Members 5325
TAdapterMemoField Properties 5328
TAdapterMemoField Events 5331
TAdapterMultiValueField Class 5332
TAdapterMultiValueField Members 5334
TAdapterMultiValueField Properties 5338
TAdapterMultiValueField Events 5341
TAdapterNamedDisplayField Class 5343
TAdapterNamedDisplayField Members 5344
TAdapterNamedDisplayField.Create Constructor 5346

TAdapterNamedDisplayField Methods 5346
TAdapterNamedDisplayField Properties 5350
TAdapterNamedField Class 5351
TAdapterNamedField Members 5352
TAdapterNamedField Methods 5354
TAdapterNamedField Properties 5367
TAdapterNextPageAction Class 5370
TAdapterNextPageAction Members 5371
TAdapterNextPageAction Methods 5374
TAdapterNextPageAction Properties 5376
TAdapterNextPageAction Events 5377
TAdapterNextPageField Class 5379
TAdapterNextPageField Members 5381
TAdapterNextPageField.Destroy Destructor 5383
TAdapterNextPageField Methods 5383
TAdapterNextPageField Properties 5386
TAdapterObjectErrorsList Class 5387
TAdapterObjectErrorsList Members 5388
TAdapterObjectErrorsList.Create Constructor 5389
TAdapterObjectErrorsList Methods 5390
TAdapterPasswordField Class 5391
TAdapterPasswordField Members 5392
TAdapterPasswordField Methods 5394
TAdapterPasswordField Properties 5395
TAdapterPrevPageAction Class 5396
TAdapterPrevPageAction Members 5397
TAdapterPrevPageAction Methods 5401
TAdapterPrevPageAction Properties 5402
TAdapterPrevPageAction Events 5403
TAdapterSubComponent Class 5406
TAdapterSubComponent Members 5407
TAdapterSubComponent Methods 5408
TAdapterSubComponent Properties 5418
TAdapterUpdateField Class 5419
TAdapterUpdateField Members 5421
TAdapterUpdateField.Create Constructor 5423
TAdapterUpdateField Methods 5424
TAdapterUpdateField Properties 5432
TAdapterUpdateField Events 5433

TAdapterUserNameField Class 5433
TAdapterUserNameField Members 5434
TAdapterUserNameField Methods 5437
TAdapterUserNameField Properties 5437
TApplicationAdapter Class 5439
TApplicationAdapter Members 5440
TApplicationAdapter Properties 5444
TApplicationAdapter Events 5445
TBaseAdapterAction Class 5447
TBaseAdapterAction Members 5448
TBaseAdapterAction Methods 5450
TBaseAdapterAction Properties 5473
TBaseAdapterAction Events 5477
TBaseAdapterErrorsList Class 5479
TBaseAdapterErrorsList Members 5479
TBaseAdapterErrorsList Methods 5481
TBaseAdapterErrorsList Properties 5487
TBaseAdapterMultiValueField Class 5488
TBaseAdapterMultiValueField Members 5489
TBaseAdapterMultiValueField.Create Constructor 5492
TBaseAdapterMultiValueField.Destroy Destructor 5492
TBaseAdapterMultiValueField Methods 5493
TBaseEndUserAdapter Class 5497
TBaseEndUserAdapter Members 5499
TBaseEndUserAdapter Methods 5503
TBaseEndUserAdapter Properties 5515
TBaseEndUserAdapter Events 5516
TBaseValuesListAdapter Class 5517
TBaseValuesListAdapter Members 5518
TBaseValuesListAdapter Methods 5522
TBaseValuesListField Class 5527
TBaseValuesListField Members 5528
TBaseValuesListField Methods 5530
TBaseValuesListField Properties 5530
TCustomAdapter Class 5531
TCustomAdapter Members 5533
TCustomAdapter.Create Constructor 5536
TCustomAdapter.Destroy Destructor 5536
TCustomAdapter Methods 5536

TCustomAdapter Properties 5571
TCustomAdapter Events 5575
TCustomAdapterAction Class 5579
TCustomAdapterAction Members 5580
TCustomAdapterAction Methods 5583
TCustomAdapterAction Properties 5586
TCustomAdapterAction Events 5587
TCustomAdapterBooleanField Class 5589
TCustomAdapterBooleanField Members 5591
TCustomAdapterBooleanField Methods 5594
TCustomAdapterBooleanField Events 5597
TCustomAdapterField Class 5597
TCustomAdapterField Members 5598
TCustomAdapterField Methods 5602
TCustomAdapterField Events 5603
TCustomAdapterFileField Class 5604
TCustomAdapterFileField Members 5605
TCustomAdapterFileField Methods 5608
TCustomAdapterFileField Properties 5612
TCustomAdapterFileField Events 5613
TCustomAdapterGotoPageAction Class 5614
TCustomAdapterGotoPageAction Members 5615
TCustomAdapterGotoPageAction Methods 5619
TCustomAdapterGotoPageAction Events 5622
TCustomAdapterImageField Class 5623
TCustomAdapterImageField Members 5624
TCustomAdapterImageField Methods 5627
TCustomAdapterImageField Properties 5634
TCustomAdapterImageField Events 5635
TCustomAdapterMemoField Class 5638
TCustomAdapterMemoField Members 5639
TCustomAdapterMemoField Methods 5642
TCustomAdapterMemoField Events 5644
TCustomAdapterMultiValueField Class 5646
TCustomAdapterMultiValueField Members 5647
TCustomAdapterMultiValueField Methods 5650
TCustomAdapterMultiValueField Properties 5653
TCustomAdapterMultiValueField Events 5654
TCustomAdapterPageAction Class 5656

TCustomAdapterPageAction Members 5657
TCustomAdapterPageAction Methods 5661
TCustomApplicationAdapter Class 5663
TCustomApplicationAdapter Members 5664
TCustomApplicationAdapter Methods 5668
TCustomApplicationAdapter Properties 5671
TCustomApplicationAdapterAction Class 5671
TCustomApplicationAdapterAction Members 5672
TCustomApplicationAdapterAction Properties 5676
TCustomEndUserAdapter Class 5676
TCustomEndUserAdapter Members 5678
TCustomEndUserAdapter Methods 5682
TCustomEndUserAdapter Events 5685
TCustomEndUserAdapterAction Class 5687
TCustomEndUserAdapterAction Members 5688
TCustomEndUserAdapterAction Properties 5691
TCustomEndUserSessionAdapter Class 5692
TCustomEndUserSessionAdapter Members 5694
TCustomEndUserSessionAdapter Methods 5698
TCustomEndUserSessionAdapter Properties 5700
TCustomLoginFormAdapter Class 5701
TCustomLoginFormAdapter Members 5702
TCustomLoginFormAdapter Methods 5706
TCustomLoginFormAdapter Properties 5713
TCustomLoginFormAdapter Events 5715
TCustomLoginFormAdapterAction Class 5717
TCustomLoginFormAdapterAction Members 5718
TCustomLoginFormAdapterAction Properties 5721
TCustomPagedAdapter Class 5721
TCustomPagedAdapterAction Class 5722
TCustomPagedAdapterAction Members 5723
TCustomPagedAdapterAction Methods 5727
TCustomPagedAdapterAction Properties 5727
TCustomStringsValuesList Class 5728
TCustomStringsValuesList Members 5729
TCustomStringsValuesList.Create Constructor 5733
TCustomStringsValuesList.Destroy Destructor 5733
TCustomStringsValuesList Methods 5733
TCustomStringsValuesList Properties 5737

TCustomStringsValuesList Events 5738
TDefaultFieldsAdapter Class 5738
TDefaultFieldsAdapter Members 5739
TDefaultFieldsAdapter Methods 5743
TDefaultFieldsAdapter Properties 5746
TDefaultFieldsPagedAdapter Class 5747
TDefaultFieldsPagedAdapter Members 5749
TDefaultFieldsPagedAdapter Methods 5753
TDefaultFieldsPagedAdapter Properties 5760
TDefaultFieldsPagedAdapter Events 5762
TEndUserAdapter Class 5764
TEndUserAdapter Members 5765
TEndUserAdapter Properties 5770
TEndUserAdapter Events 5771
TEndUserAdapterLoginFormAction Class 5776
TEndUserAdapterLoginFormAction Members 5777
TEndUserAdapterLoginFormAction Methods 5780
TEndUserAdapterLoginFormAction Properties 5781
TEndUserAdapterLoginFormAction Events 5782
TEndUserAdapterLogoutAction Class 5784
TEndUserAdapterLogoutAction Members 5785
TEndUserAdapterLogoutAction Methods 5788
TEndUserAdapterLogoutAction Properties 5790
TEndUserAdapterLogoutAction Events 5790
TEndUserSessionAdapter Class 5792
TEndUserSessionAdapter Members 5794
TEndUserSessionAdapter Properties 5799
TEndUserSessionAdapter Events 5800
TFieldValuesAdapter Class 5803
TFieldValuesAdapter Members 5805
TFieldValuesAdapter.Create Constructor 5809
TFieldValuesAdapter Methods 5809
TFieldValuesAdapter Properties 5812
TFieldValuesValueField Class 5813
TFieldValuesValueField Members 5814
TFieldValuesValueField Methods 5816
TFieldValuesValueField Properties 5817
TImplementedAdapterAction Class 5818
TLogicalPageList Class 5819

TLogicalPageList Members 5819
TLogicalPageList.Create Constructor 5824
TLoginFormAdapter Class 5824
TLoginFormAdapter Members 5826
TLoginFormAdapter Properties 5830
TLoginFormAdapter Events 5832
TLoginFormAdapterLoginAction Class 5835
TLoginFormAdapterLoginAction Members 5836
TLoginFormAdapterLoginAction Methods 5840
TLoginFormAdapterLoginAction Properties 5842
TLoginFormAdapterLoginAction Events 5842
TPagedAdapter Class 5844
TPagedAdapter Members 5845
TPagedAdapter Properties 5850
TPagedAdapter Events 5852
TStringsValuesList Class 5857
TStringsValuesList Members 5858
TStringsValuesList Properties 5862
TStringsValuesList Events 5863
TUpdateFileList Class 5864
TUpdateFileList Members 5864
TUpdateFileList.Create Constructor 5865
TUpdateFileList Properties 5865
TValuesListNameField Class 5866
TValuesListNameField Members 5867
TValuesListNameField Methods 5869
TValuesListValueField Class 5869
TValuesListValueField Members 5870
TValuesListValueField Methods 5872

Interfaces 5874
IAdapterEditor Interface 5874
IAdapterEditor Members 5875
IAdapterEditor Methods 5875
IAdapterNotifyAdapterChange Interface 5875
IAdapterNotifyAdapterChange Members 5876
IAdapterNotifyAdapterChange Methods 5876
ICheckOrigValue Interface 5876
ICheckOrigValue Members 5877
ICheckOrigValue Methods 5877

ICheckValueChange Interface 5877
ICheckValueChange Members 5878
ICheckValueChange Methods 5878
IClearAdapterValues Interface 5878
IClearAdapterValues Members 5879
IClearAdapterValues Methods 5879
ICreateActionRequestContext Interface 5879
ICreateActionRequestContext Members 5879
ICreateActionRequestContext Methods 5880
IEchoActionFieldValue Interface 5880
IEchoActionFieldValue Members 5880
IEchoActionFieldValue Methods 5881
IEchoActionFieldValue Properties 5881
IEchoAdapterFieldValues Interface 5882
IEchoAdapterFieldValues Members 5882
IEchoAdapterFieldValues Methods 5882
IEchoAdapterFieldValues Properties 5883
IExecuteValue Interface 5883
IExecuteValue Members 5884
IExecuteValue Methods 5884
IFullyQualifyInputNames Interface 5885
IFullyQualifyInputNames Members 5885
IFullyQualifyInputNames Methods 5885
IGetAdapterErrorsList Interface 5886
IGetAdapterErrorsList Members 5886
IGetAdapterErrorsList Methods 5886
IGetAdapterModeNames Interface 5887
IGetAdapterModeNames Members 5887
IGetAdapterModeNames Methods 5887
IGetRecordKeys Interface 5888
IGetRecordKeys Members 5888
IGetRecordKeys Methods 5888
IIdentifyAdapter Interface 5889
ISetSubComponents Interface 5889
ISetSubComponents Members 5889
ISetSubComponents Methods 5890
ISetWebFieldValue Interface 5890
ISetWebFieldValue Members 5891
ISetWebFieldValue Methods 5891

IUpdateValue Interface 5891
IUpdateValue Members 5891
IUpdateValue Methods 5892
IValidateValue Interface 5892
IValidateValue Members 5892
IValidateValue Methods 5893
IWebActionsSubcomponent Interface 5893
IWebFieldsSubcomponent Interface 5893
IWebGetFieldPrevValue Interface 5894
IWebGetFieldPrevValue Members 5894
IWebGetFieldPrevValue Methods 5894
IWebGetFieldPrevValue Properties 5895

Functions **5897**
WebAdapt.AddDefaultActions Function 5897
WebAdapt.AddDefaultFields Function 5897
WebAdapt.FindAdapterAction Function 5898
WebAdapt.FindAdapterField Function 5898
WebAdapt.FindAdapterInParent Function 5898
WebAdapt.MakeAdapterRequestIdentifier Function 5899
WebAdapt.MakeValidAdapterFieldName Function 5899
WebAdapt.RaiseFileUploadNotSupported Function 5899
WebAdapt.RaiseMultipleFilesException Function 5900
WebAdapt.RaiseMultipleValuesException Function 5900

Structs, Records, Enums **5901**
WebAdapt.TAdapterHiddenFieldOption Enumeration 5901
WebAdapt.TIteratorMethod Enumeration 5902

Types **5903**
WebAdapt.IActionFieldValue Type 5904
WebAdapt.IActionRequest Type 5905
WebAdapt.TAbstractWebRequestFile Type 5905
WebAdapt.TActionExecuteEvent Type 5905
WebAdapt.TActionExecuteHandledEvent Type 5906
WebAdapt.TActionGetEnabledEvent Type 5906
WebAdapt.TActionGetParamsEvent Type 5907
WebAdapt.TActionRequestFieldsEvent Type 5907
WebAdapt.TActionRequestFieldsHandledEvent Type 5908
WebAdapt.TAdapterActionExecuteEvent Type 5909
WebAdapt.TAdapterActionExecuteHandledEvent Type 5909

WebAdapt.TAdapterActionGetParamsEvent Type 5910
WebAdapt.TAdapterActionsClass Type 5911
WebAdapt.TAdapterFieldBooleanValueEvent Type 5911
WebAdapt.TAdapterFieldFileEvent Type 5912
WebAdapt.TAdapterFieldFileListEvent Type 5912
WebAdapt.TAdapterFieldGetBooleanEvent Type 5913
WebAdapt.TAdapterFieldGetMultiValueEvent Type 5913
WebAdapt.TAdapterFieldGetMultiValueStringEvent Type 5914
WebAdapt.TAdapterFieldGetStringEvent Type 5914
WebAdapt.TAdapterFieldGetValueCountEvent Type 5915
WebAdapt.TAdapterFieldGetValueEvent Type 5915
WebAdapt.TAdapterFieldHasValueEvent Type 5916
WebAdapt.TAdapterFieldsClass Type 5916
WebAdapt.TAdapterFieldStringEvent Type 5917
WebAdapt.TAdapterFieldStringsEvent Type 5917
WebAdapt.TAdapterFieldValidateValueEvent Type 5918
WebAdapt.TAdapterFieldValueEvent Type 5918
WebAdapt.TAdapterHiddenFieldOptions Type 5919
WebAdapt.TEndUserGetNameEvent Type 5919
WebAdapt.TEndUserGetUserIDEvent Type 5920
WebAdapt.TEndUserHandledEvent Type 5920
WebAdapt.TEndUserIsLoggedInEvent Type 5921
WebAdapt.TEndUserRightsEvent Type 5921
WebAdapt.TEndUserUserIDEvent Type 5922
WebAdapt.TGetEOFEvent Type 5922
WebAdapt.TGetRecordCountEvent Type 5923
WebAdapt.TGetRecordEvent Type 5923
WebAdapt.TGetRecordIndexEvent Type 5924
WebAdapt.THandledEvent Type 5924
WebAdapt.TImageFieldGetHREFEvent Type 5925
WebAdapt.TImageFieldGetImageEvent Type 5925
WebAdapt.TImageFieldGetParamsEvent Type 5926
WebAdapt.TIterateRecordsEvent Type 5926
WebAdapt.TLoginUserIDEvent Type 5927
WebAdapt.TObjectList Type 5928
WebAdapt.TValidateAdapterFieldEvent Type 5928
WebAdapt.TWebFormGetPageNameEvent Type 5928

Variables **5930**

WebAdapt.NotifyPossibleHTMLChange Variable 5930

Constants **5931**
WebAdapt.sActionRequestLong Constant 5931
WebAdapt.sCurrentPage Constant 5931
WebAdapt.sGotoPageAction Constant 5932
WebAdapt.sImageRequest Constant 5932
WebAdapt.sLoginPassword Constant 5932
WebAdapt.sLoginUserName Constant 5933
WebAdapt.sNextPageAction Constant 5933
WebAdapt.sPrevPageAction Constant 5933

WebBroker Namespace 5935

Classes **5936**
TWebApplication Class 5936
TWebApplication Members 5937
TWebApplication.Create Constructor 5937
TWebApplication.Destroy Destructor 5938
TWebApplication Methods 5938
TWebApplication Properties 5939

Types **5941**
WebBroker.THandleShutdownException Type 5941
WebBroker.TServerExceptionEvent Type 5941

Variables **5942**
WebBroker.Application Variable 5942
WebBroker.HandleShutdownException Variable 5942

WebBrokerSOAP Namespace 5945

Classes **5946**
THTTPSoapDispatcher Class 5946
THTTPSoapDispatcher Members 5947
THTTPSoapDispatcher.Create Constructor 5947
THTTPSoapDispatcher.Destroy Destructor 5948
THTTPSoapDispatcher Methods 5948
THTTPSoapDispatcher Properties 5950

Functions **5951**
WebBrokerSOAP.GetSOAPWebModule Function 5951

WebCntxt Namespace 5953

Classes **5955**
TAbstractAdapterRequest Class 5955
TAbstractAdapterRequest Members 5956
TAbstractAdapterRequest.Create Constructor 5957
TAbstractAdapterRequest Properties 5957
TAbstractAdapterResponse Class 5958
TAbstractAdapterResponse Members 5958
TAbstractAdapterResponse.Create Constructor 5959
TAbstractAdapterResponse Properties 5960
TAbstractWebContext Class 5960
TAbstractWebContext Members 5961
TAbstractWebContext.Create Constructor 5962
TAbstractWebContext.Destroy Destructor 5963
TAbstractWebContext Methods 5963
TAbstractWebContext Properties 5971
TWebContext Class 5976
TWebContext Members 5976
TWebContext.Create Constructor 5978
TWebContext.Destroy Destructor 5979
TWebContext Methods 5979

Interfaces **5988**
IWebEndUser Interface 5988
IWebEndUser Members 5989
IWebEndUser Methods 5989
IWebEndUser Properties 5994
IWebPageInfo Interface 5996
IWebPageInfo Members 5997
IWebPageInfo Methods 5997
IWebPageInfo Properties 5999

Functions **6003**
WebCntxt.WebContext Function 6003

Structs, Records, Enums **6004**
WebCntxt.TFindPageOption Enumeration 6004

Types **6005**
WebCntxt.TFindPageOptions Type 6005

WebCntxt.TGetWebContextProc Type 6005
WebCntxt.TGetWebEndUserProc Type 6006
WebCntxt.TSetWebContextProc Type 6006

Variables **6007**
WebCntxt.GetModuleFileNameProc Variable 6007
WebCntxt.GetWebContextProc Variable 6007
WebCntxt.GetWebEndUserProc Variable 6008
WebCntxt.SetWebContextProc Variable 6008

WebComp Namespace 6009

Classes **6011**
TLayoutAttributes Class 6011
TLayoutAttributes Members 6012
TLayoutAttributes.Create Constructor 6013
TLayoutAttributes Fields 6013
TLayoutWebContent Class 6014
TLayoutWebContent Members 6015
TLayoutWebContent.Create Constructor 6016
TLayoutWebContent Methods 6016
TLayoutWebContent Properties 6022
TWebComponentClassDescription Class 6023
TWebComponentClassDescription Members 6023
TWebComponentClassDescription Fields 6024
TWebComponentList Class 6025
TWebComponentList Members 6026
TWebComponentList.Create Constructor 6027
TWebComponentList.Destroy Destructor 6027
TWebComponentList Methods 6028
TWebComponentList Properties 6033
TWebComponentList Events 6035
TWebComponentsEditorHelper Class 6035
TWebComponentsEditorHelper Members 6036
TWebComponentsEditorHelper Methods 6037
TWebContainedComponent Class 6038
TWebContainedComponent Members 6039
TWebContainedComponent.Create Constructor 6039
TWebContainedComponent.Destroy Destructor 6040
TWebContainedComponent Methods 6040

TWebContainedComponent Properties 6044
TWebContainedComponentHelper Class 6045
TWebContainedComponentHelper Members 6045
TWebContainedComponentHelper.Create Constructor 6047
TWebContainedComponentHelper.Destroy Destructor 6047
TWebContainedComponentHelper Methods 6047
TWebContainedComponentHelper Properties 6051
TWebContainedComponentHelper Events 6052
TWebContainedContainerComponent Class 6052
TWebContainedContainerComponent Members 6053
TWebContainedContainerComponent.Create Constructor 6054
TWebContainedContainerComponent.Destroy Destructor 6054
TWebContainedContainerComponent Methods 6055
TWebContainedContainerComponent Properties 6061
TWebContainerComponent Class 6063
TWebContainerComponent Members 6063
TWebContainerComponent.Create Constructor 6064
TWebContainerComponent.Destroy Destructor 6064
TWebContainerComponent Methods 6064
TWebContainerComponent Properties 6067
TWebContainerComponentHelper Class 6068
TWebContainerComponentHelper Members 6068
TWebContainerComponentHelper.Create Constructor 6069
TWebContainerComponentHelper.Destroy Destructor 6070
TWebContainerComponentHelper Methods 6070
TWebContainerComponentHelper Properties 6073
TWebContainerComponentHelper Events 6074
TWebContentOptions Class 6076
TWebContentOptions Members 6076
TWebContentOptions.Create Constructor 6077
TWebContentOptions Properties 6077

Interfaces 6079
IAddFieldsEditor Interface 6079
IAddFieldsEditor Members 6079
IAddFieldsEditor Methods 6080
IGetWebComponentList Interface 6081
IGetWebComponentList Members 6081
IGetWebComponentList Methods 6081
IGetWebComponentNodeName Interface 6082

IGetWebComponentNodeName Members 6082
IGetWebComponentNodeName Methods 6083
IGetXMLStream Interface 6083
IGetXMLStream Members 6083
IGetXMLStream Methods 6084
ILayoutWebContent Interface 6084
ILayoutWebContent Members 6085
ILayoutWebContent Methods 6085
INotifyActivateModule Interface 6087
INotifyActivateModule Members 6087
INotifyActivateModule Methods 6087
INotifyWebActivate Interface 6088
INotifyWebActivate Members 6088
INotifyWebActivate Methods 6088
IProducerEditorViewSupport Interface 6089
IProducerEditorViewSupport Members 6089
IProducerEditorViewSupport Methods 6090
ISetWebContentOptions Interface 6091
ISetWebContentOptions Members 6092
ISetWebContentOptions Methods 6092
ITopLevelWebComponent Interface 6092
IWebComponent Interface 6093
IWebComponent Members 6093
IWebComponent Methods 6094
IWebComponent Properties 6096
IWebComponentCommands Interface 6097
IWebComponentCommands Members 6097
IWebComponentCommands Methods 6097
IWebComponentContainer Interface 6098
IWebComponentContainer Members 6099
IWebComponentContainer Methods 6099
IWebComponentContainer Properties 6100
IWebComponentEditor Interface 6100
IWebComponentEditor Members 6101
IWebComponentEditor Methods 6101
IWebComponentEditorHelper Interface 6102
IWebComponentEditorHelper Members 6102
IWebComponentEditorHelper Methods 6102
IWebContent Interface 6103

IWebContent Members 6103
IWebContent Methods 6103
IWebVariablesContainerName Interface 6104
IWebVariablesContainerName Members 6104
IWebVariablesContainerName Methods 6105
IWebVariablesContainerName Properties 6105

Functions 6106
WebComp.DesigningComponent Function 6106
WebComp.EnumRegisteredWebComponents Function 6106
WebComp.FindWebComponentEditorHelper Function 6107
WebComp.QualifyFileName Function 6107
WebComp.RegisterWebComponents Function 6107
WebComp.UnRegisterWebComponents Function 6108
WebComp.ValidateWebParent Function 6108

Structs, Records, Enums 6109
WebComp.TLabelPosition Enumeration 6109
WebComp.TWebContentFlag Enumeration 6109

Types 6111
WebComp.TEnumWebComponentsProc Type 6111
WebComp.TGetComponentListEvent Type 6111
WebComp.TLayout Type 6112
WebComp.TSetComponentListEvent Type 6112
WebComp.TWebComponentContainer Type 6113
WebComp.TWebContentFlags Type 6113

Constants 6114
WebComp.EnumRegisteredWebComponentsProc Constant 6114
WebComp.FindWebComponentEditorHelperProc Constant 6114
WebComp.RegisterWebComponentsProc Constant 6115
WebComp.UnRegisterWebComponentsProc Constant 6115

WebContnrs Namespace 6117

Classes 6118
ENamedVariantsError Class 6118
TAbstractNamedVariants Class 6118
TAbstractNamedVariants Members 6119
TAbstractNamedVariants.Destroy Destructor 6120
TAbstractNamedVariants Methods 6120

TAbstractNamedVariants Properties 6131
TNamedVariantsList Class 6134
TNamedVariantsList Members 6134
TNamedVariantsList.Destroy Destructor 6135
TNamedVariantsList Methods 6136

Structs, Records, Enums **6140**
WebContnrs.TNamedVariantItem Record 6140

Types **6141**
WebContnrs.PNamedVariantItem Type 6141
WebContnrs.PNamedVariantList Type 6141
WebContnrs.TNamedVariantList Type 6141

WebDisp Namespace 6143

Classes **6146**
EAdapterDispatcherException Class 6146
EPageDispatcherException Class 6147
ESessionException Class 6147
TAdapterDispatcher Class 6148
TAdapterDispatcher Members 6149
TAdapterDispatcher Events 6150
TAdapterDispatchParams Class 6153
TAdapterDispatchParams Members 6154
TAdapterDispatchParams.Create Constructor 6156
TAdapterDispatchParams.Destroy Destructor 6156
TAdapterDispatchParams Methods 6157
TAdapterDispatchParams Properties 6159
TAdapterItemRequestParams Class 6160
TAdapterItemRequestParams Members 6161
TAdapterItemRequestParams.Create Constructor 6162
TAdapterItemRequestParams.Destroy Destructor 6162
TAdapterItemRequestParams Methods 6163
TAdapterItemRequestParams Properties 6165
TBaseRequestHandler Class 6166
TBaseRequestHandler Members 6166
TBaseRequestHandler Methods 6167
TBaseRequestHandler Properties 6168
TCustomAdapterDispatcher Class 6168
TCustomAdapterDispatcher Members 6169

TCustomAdapterDispatcher Methods 6170
TCustomAdapterDispatcher Events 6177
TCustomLocateFileService Class 6181
TCustomLocateFileService Members 6181
TCustomLocateFileService Methods 6182
TCustomLocateFileService Events 6186
TCustomPageDispatcher Class 6188
TCustomPageDispatcher Members 6188
TCustomPageDispatcher Methods 6189
TCustomPageDispatcher Properties 6195
TCustomPageDispatcher Events 6196
TCustomWebAppComponents Class 6201
TCustomWebAppComponents Members 6202
TCustomWebAppComponents Methods 6204
TCustomWebAppComponents Properties 6211
TCustomWebAppComponents Events 6215
TInvalidWebSession Class 6216
TInvalidWebSession Members 6216
TInvalidWebSession.Create Constructor 6217
TInvalidWebSession Methods 6217
TLocateFileService Class 6220
TLocateFileService Members 6221
TLocateFileService Events 6221
TMultiModuleWebAppServices Class 6223
TMultiModuleWebAppServices Members 6223
TMultiModuleWebAppServices Methods 6224
TMultiModuleWebAppServices Properties 6233
TMultiModuleWebAppServices Events 6235
TPageDispatcher Class 6236
TPageDispatcher Members 6236
TPageDispatcher Properties 6238
TPageDispatcher Events 6238
TWebAppComponents Class 6243
TWebAppComponents Members 6244
TWebAppComponents Properties 6245
TWebAppComponents Events 6249
TWebSession Class 6251
TWebSession Members 6251
Create Constructor 6252

TWebSession Methods 6252

Interfaces 6257

IWebSessionAccess Interface 6257

IWebSessionAccess Members 6257

IWebSessionAccess Methods 6257

Functions 6259

WebDisp.AdapterRequestParamsAsStrings Function 6259

WebDisp.DispatchPageName Function 6259

WebDisp.EncodeParamNameValues Function 6260

WebDisp.EncodeParamsAsHTTPQuery Function 6260

WebDisp.ExpandScriptException Function 6260

WebDisp.ExtractParamNameValues Function 6261

WebDisp.FindDefaultAppPageName Function 6261

WebDisp.PageNameOfRequest Function 6261

WebDisp.RaiseDispatchBlankPageNameException Function 6262

WebDisp.RaisePageDoesNotProvideContent Function 6262

WebDisp.RaisePageNotFound Function 6262

WebDisp.RedirectToPageName Function 6263

WebDisp.RegisterAdapterRequestIdentifier Function 6263

WebDisp.SafeRedirect Function 6263

Structs, Records, Enums 6265

WebDisp.TPageAccessDenied Enumeration 6265

Types 6266

WebDisp.TActivateWebModuleEvent Type 6266

WebDisp.TCanViewPageEvent Type 6267

WebDisp.TDispatchActionEvent Type 6267

WebDisp.TDispatchActionHandledEvent Type 6268

WebDisp.TDispatchImageEvent Type 6269

WebDisp.TDispatchImageHandledEvent Type 6269

WebDisp.TDispatchPageEvent Type 6270

WebDisp.TDispatchPageHandledEvent Type 6270

WebDisp.TDispatchPageParamsHandledEvent Type 6271

WebDisp.TGetAdapterRequestParamsEvent Type 6272

WebDisp.TLocateFileServiceFindFileEvent Type 6273

WebDisp.TLocateFileServiceFindStreamEvent Type 6273

WebDisp.TLoginPagedHandledEvent Type 6274

WebDisp.TLoginRequiredEvent Type 6274

WebDisp.TPageAccessDeniedEvent Type 6275
WebDisp.TWebRequest Type 6276
WebDisp.TWebResponse Type 6276

Constants 6277

WebDisp.sFailurePageParamName Constant 6277
WebDisp.sPageParamName Constant 6277
WebDisp.sRequestIDParamName Constant 6278
WebDisp.sSessionCookieName Constant 6278
WebDisp.sSuccessPageParamName Constant 6278

WebFact Namespace 6279

Classes 6280

TBaseWebModuleFactory Class 6280
TBaseWebModuleFactory Members 6281
TBaseWebModuleFactory.Create Constructor 6281
TBaseWebModuleFactory Methods 6282
TBaseWebPageInfo Class 6285
TBaseWebPageInfo Members 6285
TBaseWebPageInfo.Create Constructor 6286
TBaseWebPageInfo Methods 6286
TBaseWebPageModuleFactory Class 6290
TBaseWebPageModuleFactory Members 6291
TBaseWebPageModuleFactory.Create Constructor 6291
TBaseWebPageModuleFactory Methods 6292
TWebAppDataModuleFactory Class 6295
TWebAppDataModuleFactory Members 6295
TWebAppDataModuleFactory.Create Constructor 6296
TWebAppDataModuleFactory Methods 6296
TWebAppPageModuleFactory Class 6297
TWebAppPageModuleFactory Members 6298
TWebAppPageModuleFactory.Create Constructor 6298
TWebAppPageModuleFactory Methods 6298
TWebDataModuleFactory Class 6299
TWebDataModuleFactory Members 6300
TWebDataModuleFactory.Create Constructor 6300
TWebPageInfo Class 6301
TWebPageInfo Members 6301
TWebPageInfo.Create Constructor 6302

TWebPageModuleFactory Class 6303
TWebPageModuleFactory Members 6303
TWebPageModuleFactory.Create Constructor 6304

Structs, Records, Enums **6305**
WebFact.TWebPageAccessFlags Enumeration 6305

Types **6306**
WebFact.TWebPageAccess Type 6306

WebForm Namespace 6307

Classes **6310**
TAbstractControlRequirements Class 6311
TAbstractControlRequirements Members 6311
TAbstractControlRequirements Methods 6312
TAbstractFormRequirements Class 6314
TAbstractFormRequirements Members 6314
TAbstractFormRequirements Methods 6315
TAdapterActionButton Class 6316
TAdapterActionButton Members 6317
TAdapterActionButton Properties 6318
TAdapterActionsListHelper Class 6324
TAdapterActionsListHelper Members 6325
TAdapterActionsListHelper.Create Constructor 6326
TAdapterActionsListHelper Methods 6327
TAdapterActionsListHelper Properties 6330
TAdapterCommandColumn Class 6331
TAdapterCommandColumn Members 6332
TAdapterCommandColumn Properties 6333
TAdapterCommandGroup Class 6337
TAdapterCommandGroup Members 6338
TAdapterCommandGroup Properties 6339
TAdapterControlGroup Class 6341
TAdapterControlGroup Members 6342
TAdapterControlGroup.Create Constructor 6343
TAdapterControlGroup.Destroy Destructor 6344
TAdapterControlGroup Methods 6344
TAdapterControlGroup Properties 6352
TAdapterDataDisplay Class 6354
TAdapterDataDisplay Members 6355

TAdapterDataDisplay.Create Constructor 6356
TAdapterDataDisplay Methods 6356
TAdapterDataDisplay Properties 6362
TAdapterDisplayColumn Class 6364
TAdapterDisplayColumn Members 6364
TAdapterDisplayColumn Properties 6366
TAdapterDisplayField Class 6371
TAdapterDisplayField Members 6372
TAdapterDisplayField Properties 6373
TAdapterDisplayReferenceGroup Class 6381
TAdapterDisplayReferenceGroup Members 6382
TAdapterDisplayReferenceGroup.Create Constructor 6382
TAdapterDisplayReferenceGroup.Destroy Destructor 6382
TAdapterDisplayReferenceGroup Methods 6383
TAdapterDisplayReferenceGroup Properties 6386
TAdapterEditColumn Class 6387
TAdapterEditColumn Members 6387
TAdapterEditColumn Properties 6389
TAdapterErrorList Class 6397
TAdapterErrorList Members 6397
TAdapterErrorList Properties 6398
TAdapterFieldGroup Class 6399
TAdapterFieldGroup Members 6400
TAdapterFieldGroup Properties 6401
TAdapterForm Class 6404
TAdapterForm Members 6405
TAdapterForm Properties 6406
TAdapterFormLayout Class 6408
TAdapterFormLayout Members 6408
TAdapterFormLayout.Create Constructor 6410
TAdapterFormLayout Methods 6410
TAdapterFormLayout Properties 6416
TAdapterGrid Class 6419
TAdapterGrid Members 6421
TAdapterGrid Properties 6422
TControlRequirements Class 6425
TControlRequirements Members 6426
TControlRequirements.Create Constructor 6426
TControlRequirements.Destroy Destructor 6427

TControlRequirements Methods 6427
TControlRequirements Properties 6429
TCustomAdapterActionButton Class 6430
TCustomAdapterActionButton Members 6431
TCustomAdapterActionButton.Create Constructor 6431
TCustomAdapterActionButton.Destroy Destructor 6432
TCustomAdapterActionButton Methods 6432
TCustomAdapterActionButton Properties 6437
TCustomAdapterCommandColumn Class 6442
TCustomAdapterCommandColumn Members 6443
TCustomAdapterCommandColumn.Create Constructor 6444
TCustomAdapterCommandColumn.Destroy Destructor 6445
TCustomAdapterCommandColumn Methods 6445
TCustomAdapterCommandColumn Properties 6449
TCustomAdapterCommandGroup Class 6454
TCustomAdapterCommandGroup Members 6454
TCustomAdapterCommandGroup.Create Constructor 6456
TCustomAdapterCommandGroup.Destroy Destructor 6456
TCustomAdapterCommandGroup Methods 6456
TCustomAdapterCommandGroup Properties 6458
TCustomAdapterDisplayColumn Class 6461
TCustomAdapterDisplayColumn Members 6461
TCustomAdapterDisplayColumn.Create Constructor 6463
TCustomAdapterDisplayColumn Methods 6463
TCustomAdapterDisplayColumn Properties 6466
TCustomAdapterDisplayField Class 6467
TCustomAdapterDisplayField Members 6468
TCustomAdapterDisplayField.Create Constructor 6469
TCustomAdapterDisplayField Methods 6469
TCustomAdapterDisplayField Properties 6472
TCustomAdapterEditColumn Class 6475
TCustomAdapterEditColumn Members 6476
TCustomAdapterEditColumn Methods 6477
TCustomAdapterErrorList Class 6478
TCustomAdapterErrorList Members 6479
TCustomAdapterErrorList.Create Constructor 6480
TCustomAdapterErrorList.Destroy Destructor 6480
TCustomAdapterErrorList Methods 6480
TCustomAdapterErrorList Properties 6482

TCustomAdapterFieldGroup Class 6483
TCustomAdapterFieldGroup Members 6484
TCustomAdapterFieldGroup.Create Constructor 6485
TCustomAdapterFieldGroup.Destroy Destructor 6485
TCustomAdapterFieldGroup Methods 6486
TCustomAdapterFieldGroup Properties 6488
TCustomAdapterForm Class 6490
TCustomAdapterForm Members 6491
TCustomAdapterForm.Create Constructor 6491
TCustomAdapterForm.Destroy Destructor 6491
TCustomAdapterForm Methods 6492
TCustomAdapterForm Properties 6494
TCustomAdapterGrid Class 6496
TCustomAdapterGrid Members 6498
TCustomAdapterGrid.Create Constructor 6499
TCustomAdapterGrid.Destroy Destructor 6499
TCustomAdapterGrid Methods 6500
TCustomAdapterGrid Properties 6502
TDesigntimeWarnings Class 6504
TDesigntimeWarnings Members 6504
TDesigntimeWarnings.Create Constructor 6504
TDesigntimeWarnings.Destroy Destructor 6504
TDesigntimeWarnings Methods 6505
TDesigntimeWarnings Properties 6506
TGeneratedFunctions Class 6506
TGeneratedFunctions Members 6506
TGeneratedFunctions.Create Constructor 6507
TGeneratedFunctions.Destroy Destructor 6507
TGeneratedFunctions Methods 6507
TGeneratedFunctions Properties 6508

Interfaces 6510

IAdapterDisplay Interface 6510
IGetAdapterFormLayout Interface 6510
IGetAdapterFormLayout Members 6511
IGetAdapterFormLayout Methods 6511
IGetAdapterModeOfDisplay Interface 6511
IGetAdapterModeOfDisplay Members 6512
IGetAdapterModeOfDisplay Methods 6512
IGetAdapterOfDisplay Interface 6512

IGetAdapterOfDisplay Members 6513
IGetAdapterOfDisplay Methods 6513
IGetAdapterVariableName Interface 6513
IGetAdapterVariableName Members 6514
IGetAdapterVariableName Methods 6514
IGetAdapterVariableName Properties 6514
IGetControlRequirements Interface 6515
IGetControlRequirements Members 6515
IGetControlRequirements Methods 6515
IGetFormRequirements Interface 6516
IGetFormRequirements Members 6516
IGetFormRequirements Methods 6516
IGetVariablesContainerOfDisplay Interface 6517
IGetVariablesContainerOfDisplay Members 6517
IGetVariablesContainerOfDisplay Methods 6517

Functions **6519**
WebForm.CalcDisplayFieldHTMLElementType Function 6519
WebForm.CalcDisplayFieldViewMode Function 6519
WebForm.FindVariablesContainer Function 6520
WebForm.FindVariablesContainerInParent Function 6520
WebForm.GetAdapterModeOfAdapterInParent Function 6520
WebForm.MakeAdapterHTMLName Function 6521
WebForm.MakeAdapterVariableName Function 6521

Structs, Records, Enums **6522**
WebForm.TActionButtonHideOption Enumeration 6522
WebForm.TColumnHTMLElementType Enumeration 6523
WebForm.TCommandHTMLElementType Enumeration 6523
WebForm.TDisplayFieldHideOption Enumeration 6524
WebForm.TDisplayFieldHTMLElementType Enumeration 6525
WebForm.TDisplayFieldViewMode Enumeration 6525
WebForm.TInputFieldHTMLElementType Enumeration 6526
WebForm.TLayoutState Enumeration 6527

Types **6528**
WebForm.TActionButtonHideOptions Type 6528
WebForm.TAdapterDataDisplayClass Type 6528
WebForm.TDisplayFieldHideOptions Type 6528

Constants **6530**

WebForm.sDisplayText Constant 6530
WebForm.sFieldText Constant 6530
WebForm.sFormGet Constant 6531
WebForm.sFormPost Constant 6531
WebForm.sMultipartForm Constant 6531

WebModu Namespace 6533

Classes 6535

EWebModuleException Class 6535
TAdapterActionName Class 6536
TAdapterActionName Members 6536
TAdapterActionName.Create Constructor 6537
TAdapterActionName Methods 6537
TAdapterActionName Properties 6538
TCustomWebAppDataModule Class 6539
TCustomWebAppDataModule Members 6540
TCustomWebAppDataModule.Create Constructor 6541
TCustomWebAppDataModule.CreateNew Constructor 6541
TCustomWebAppDataModule.Destroy Destructor 6542
TCustomWebAppDataModule Methods 6542
TCustomWebAppDataModule Properties 6543
TCustomWebAppPageModule Class 6544
TCustomWebAppPageModule Members 6545
TCustomWebAppPageModule.Create Constructor 6546
TCustomWebAppPageModule.CreateNew Constructor 6547
TCustomWebAppPageModule.Destroy Destructor 6547
TCustomWebAppPageModule Methods 6547
TCustomWebAppPageModule Properties 6549
TCustomWebDataModule Class 6550
TCustomWebDataModule Members 6551
TCustomWebDataModule.Create Constructor 6551
TCustomWebDataModule.CreateNew Constructor 6552
TCustomWebDataModule.Destroy Destructor 6552
TCustomWebDataModule Methods 6553
TCustomWebDataModule Properties 6554
TCustomWebDataModule Events 6555
TCustomWebPageModule Class 6556
TCustomWebPageModule Members 6557

TCustomWebPageModule.Create Constructor 6559
TCustomWebPageModule.CreateNew Constructor 6559
TCustomWebPageModule.Destroy Destructor 6560
TCustomWebPageModule Methods 6560
TCustomWebPageModule Properties 6561
TCustomWebPageModule Events 6563
TSiteModuleHelper Class 6565
TSiteModuleHelper Members 6566
TSiteModuleHelper.Create Constructor 6567
TSiteModuleHelper.Destroy Destructor 6567
TSiteModuleHelper Methods 6568
TSiteModuleHelper Properties 6571
TSiteModuleHelper Events 6573
TSitePageModuleHelper Class 6574
TSitePageModuleHelper Members 6574
TSitePageModuleHelper.Create Constructor 6576
TSitePageModuleHelper.Destroy Destructor 6577
TSitePageModuleHelper Methods 6577
TSitePageModuleHelper Properties 6582
TSitePageModuleHelper Events 6583
TWebAppDataModule Class 6585
TWebAppDataModule Members 6585
TWebAppDataModule Properties 6586
TWebAppDataModule Events 6587
TWebAppPageModule Class 6588
TWebAppPageModule Members 6589
TWebAppPageModule Properties 6591
TWebAppPageModule Events 6592
TWebAppPageModuleHelper Class 6594
TWebAppPageModuleHelper Members 6594
TWebAppPageModuleHelper Methods 6596
TWebDataModule Class 6597
TWebDataModule Members 6597
TWebDataModule Events 6598
TWebNotifyContainer Class 6599
TWebNotifyContainer Members 6600
TWebNotifyContainer.Create Constructor 6601
TWebNotifyContainer.Destroy Destructor 6601
TWebNotifyContainer Fields 6602

TWebNotifyContainer Methods 6602
TWebPageModule Class 6603
TWebPageModule Members 6604
TWebPageModule Properties 6606
TWebPageModule Events 6607
TWebPageModuleHelper Class 6609
TWebPageModuleHelper Members 6609
TWebPageModuleHelper Methods 6611

Functions **6612**
WebModu.FindNamedModuleVariable Function 6612
WebModu.GetModuleVariable Function 6612
WebModu.GetModuleVariableCount Function 6612

Types **6614**
WebModu.TGetParamsEvent Type 6614

WebNode Namespace 6615

Interfaces **6616**
IWebNode Interface 6616
IWebNode Members 6617
IWebNode Methods 6617
IWebNode Properties 6619

WebReq Namespace 6621

Classes **6622**
TWebModuleFactoryList Class 6622
TWebModuleFactoryList Members 6622
TWebModuleFactoryList.Create Constructor 6623
TWebModuleFactoryList.Destroy Destructor 6624
TWebModuleFactoryList Methods 6624
TWebModuleFactoryList Properties 6625
TWebModuleList Class 6626
TWebModuleList Members 6627
TWebModuleList.Create Constructor 6627
TWebModuleList.Destroy Destructor 6628
TWebModuleList Methods 6628
TWebModuleList Properties 6633
TWebRequestHandler Class 6633

TWebRequestHandler Members 6634
TWebRequestHandler.Create Constructor 6635
TWebRequestHandler.Destroy Destructor 6635
TWebRequestHandler Methods 6635
TWebRequestHandler Properties 6638

Functions **6642**
WebReq.WebRequestHandler Function 6642

Structs, Records, Enums **6643**
WebReq.TRequestNotification Enumeration 6643

Variables **6644**
WebReq.WebRequestHandlerProc Variable 6644

WebScript Namespace 6645

Classes **6647**
ERecursiveIncludeFile Class 6647
EScriptEvalError Class 6648
TAbstractScriptEngine Class 6648
TAbstractScriptEngine Members 6648
TAbstractScriptEngine.Create Constructor 6649
TAbstractScriptEngine.Destroy Destructor 6649
TAbstractScriptEngine Methods 6650
TAbstractScriptEngine Properties 6651
TAbstractScriptObjectFactories Class 6652
TAbstractScriptObjectFactories Members 6652
TAbstractScriptObjectFactories Methods 6653
TAbstractScriptObjectFactories Properties 6654
TAbstractScriptObjectFactory Class 6655
TAbstractScriptObjectFactory Members 6656
TAbstractScriptObjectFactory Methods 6656
TCurrentWebPageInfo Class 6657
TCurrentWebPageInfo Members 6658
TCurrentWebPageInfo.Create Constructor 6659
TCurrentWebPageInfo Methods 6660
THTMLItem Class 6662
THTMLItem Members 6663
THTMLItem.Create Constructor 6664
THTMLItem Methods 6664

THTMLItem Properties 6664
THTMLItems Class 6665
THTMLItems Members 6666
THTMLItems.Create Constructor 6666
THTMLItems.Destroy Destructor 6666
THTMLItems Methods 6667
THTMLItems Properties 6667
TLineNumberMap Class 6668
TLineNumberMap Members 6669
TLineNumberMap Methods 6670
TLineNumberMap Properties 6670
TScriptCodeItem Class 6671
TScriptCodeItem Members 6671
TScriptCodeItem Methods 6672
TScriptEnginesList Class 6673
TScriptEnginesList Members 6673
TScriptEnginesList.Create Constructor 6674
TScriptEnginesList.Destroy Destructor 6674
TScriptEnginesList Methods 6675
TScriptEnginesList Properties 6676
TScriptError Class 6677
TScriptError Members 6677
TScriptError.Create Constructor 6678
TScriptError Fields 6678
TScriptError Methods 6679
TScriptErrors Class 6681
TScriptErrors Members 6681
TScriptErrors.Create Constructor 6681
TScriptErrors.Destroy Destructor 6682
TScriptErrors Methods 6682
TScriptFile Class 6683
TScriptFile Members 6684
TScriptFile.Create Constructor 6684
TScriptFile.Destroy Destructor 6684
TScriptFile Fields 6685
TScriptFile Methods 6685
TScriptFile Properties 6686
TScriptGlobalObjects Class 6687
TScriptGlobalObjects Members 6687

TScriptGlobalObjects.Create Constructor 6688
TScriptGlobalObjects.Destroy Destructor 6689
TScriptGlobalObjects Methods 6689
TScriptGlobalObjects Properties 6691
TScriptIncludeFile Class 6692
TScriptIncludeFile Members 6692
TScriptIncludeFile.Create Constructor 6693
TScriptIncludeFile Methods 6693
TScriptIncludeFiles Class 6693
TScriptIncludeFiles Members 6694
TScriptIncludeFiles.Create Constructor 6694
TScriptIncludeFiles Methods 6694
TScriptIncludeFiles Properties 6695
TScriptIncludeItem Class 6696
TScriptIncludeItem Members 6696
TScriptIncludeItem.Create Constructor 6697
TScriptIncludeItem Methods 6697
TScriptItem Class 6698
TScriptItem Members 6698
TScriptItem.Create Constructor 6699
TScriptItem Methods 6699
TScriptItem Properties 6699
TScriptItems Class 6700
TScriptItems Members 6700
TScriptItems.Create Constructor 6701
TScriptItems Methods 6701
TScriptItems Properties 6701
TScriptObjectFactories Class 6702
TScriptObjectFactories Members 6703
TScriptObjectFactories.Create Constructor 6704
TScriptObjectFactories.Destroy Destructor 6704
TScriptObjectFactories Methods 6705
TScriptObjectFactories Properties 6706
TScriptProducer Class 6707
TScriptProducer Members 6707
TScriptProducer.Create Constructor 6708
TScriptProducer.Destroy Destructor 6708
TScriptProducer Fields 6709
TScriptProducer Methods 6710

TScriptProducer Properties 6714
TVariableLookup Class 6716
TVariableLookup Members 6716
TVariableLookup.Create Constructor 6717
TVariableLookup Methods 6717
TVariableLookup Properties 6718

Structs, Records, Enums **6720**
WebScript.TGlobalScriptVariables Enumeration 6720
WebScript.TIncludeType Enumeration 6721
WebScript.TLineNumbers Record 6721

Types **6722**
WebScript.CreateComponentStubFactory Type 6722
WebScript.PLineNumbers Type 6722

Constants **6723**
WebScript.GlobalScriptVariableNames Constant 6723

WebServExp Namespace 6725

Classes **6727**
TWebServExp Class 6727
TWebServExp Members 6727
TWebServExp.Create Constructor 6729
TWebServExp.Destroy Destructor 6729
TWebServExp Methods 6730
TWebServExp Properties 6731
TWebServExp Events 6733

Interfaces **6735**
IWebServExp Interface 6735
IWebServExp Members 6735
IWebServExp Methods 6736
IWebServExpAccess Interface 6737
IWebServExpAccess Members 6737
IWebServExpAccess Methods 6737

Functions **6738**
WebServExp.GetAliasBaseTypeInfo Function 6738
WebServExp.GetBindingType Function 6738
WebServExp.IsBaseClassTypeInfo Function 6739

Structs, Records, Enums **6740**
WebServExp.ArgumentType Enumeration 6740
WebServExp.MessageType Enumeration 6740
WebServExp.TSchemaType Record 6741

Types **6742**
WebServExp.TAfterPublishingWSDLEvent Type 6742
WebServExp.TBeforePublishingTypesEvent Type 6742
WebServExp.TPublishingTypeEvent Type 6743
WebServExp.TSchemaTypeArray Type 6744

WebSess Namespace 6745

Classes **6746**
TCustomSessionsService Class 6746
TCustomSessionsService Members 6747
TCustomSessionsService.Create Constructor 6747
TCustomSessionsService Methods 6748
TCustomSessionsService Properties 6754
TCustomSessionsService Events 6755
TSessionsService Class 6756
TSessionsService Members 6757
TSessionsService Properties 6757
TSessionsService Events 6759
TWebSessionIDsImpl Class 6759
TWebSessionIDsImpl Members 6760
TWebSessionIDsImpl.Create Constructor 6761
TWebSessionIDsImpl.Destroy Destructor 6761
TWebSessionIDsImpl Methods 6762
TWebSessionImpl Class 6763
TWebSessionImpl Members 6763
TWebSessionImpl.Create Constructor 6765
TWebSessionImpl.Destroy Destructor 6765
TWebSessionImpl Methods 6766
TWebSessionImpl Properties 6770

Structs, Records, Enums **6773**
WebSess.TEndSessionReason Enumeration 6773

Types **6774**
WebSess.TEndSessionEvent Type 6774

WebSess.TStartSessionEvent Type 6774

WebUsers Namespace 6777

Classes **6779**

EUserIDNotFoundException Class 6779
EValidateUserException Class 6779
EWebUsersListException Class 6780
TCustomWebUserList Class 6781
TCustomWebUserList Members 6781
TCustomWebUserList.Create Constructor 6782
TCustomWebUserList.Destroy Destructor 6782
TCustomWebUserList Methods 6782
TCustomWebUserList Properties 6784
TCustomWebUserList Events 6785
TWebUserItem Class 6787
TWebUserItem Members 6788
TWebUserItem.Create Constructor 6788
TWebUserItem.Destroy Destructor 6789
TWebUserItem Methods 6789
TWebUserItem Properties 6790
TWebUserItems Class 6792
TWebUserItems Members 6792
TWebUserItems.Create Constructor 6793
TWebUserItems Methods 6793
TWebUserItems Properties 6796
TWebUserList Class 6796
TWebUserList Members 6797
TWebUserList Properties 6797
TWebUserList Events 6798

Structs, Records, Enums **6801**

WebUsers.TValidateUserError Enumeration 6801

Types **6802**

WebUsers.TCheckAccessRightsEvent Type 6802
WebUsers.TCheckAccessRightsHandledEvent Type 6802
WebUsers.TValidateUserErrorEvent Type 6803
WebUsers.TValidateUserEvent Type 6804
WebUsers.TValidateUserHandledEvent Type 6804

WideStrings Namespace 6807

Classes **6808**

TWideStringList Class 6808

TWideStringList Members 6809

Create Constructor 6810

TWideStringList.Destroy Destructor 6811

TWideStringList Methods 6812

TWideStringList Properties 6822

TWideStringList Events 6824

TWideStrings Class 6825

TWideStrings Members 6826

TWideStrings.Destroy Destructor 6828

TWideStrings Methods 6828

TWideStrings Properties 6847

TWideStringsEnumerator Class 6856

TWideStringsEnumerator Members 6856

TWideStringsEnumerator.Create Constructor 6857

TWideStringsEnumerator Methods 6857

TWideStringsEnumerator Properties 6858

Interfaces **6859**

IWideStringsAdapter Interface 6859

IWideStringsAdapter Members 6859

IWideStringsAdapter Methods 6859

Structs, Records, Enums **6861**

WideStrings.TWideStringItem Record 6861

Types **6862**

WideStrings.PWideStringItem Type 6862

WideStrings.PWideStringItemList Type 6862

WideStrings.TWideStringItemList Type 6863

WideStrings.TWideStringListSortCompare Type 6863

WideStrUtils Namespace 6865

Functions **6867**

WideStrUtils.AnsiToUtf8Ex Function 6868

WideStrUtils.ConvertStreamFromAnsiToUTF8 Function 6868

WideStrUtils.ConvertStreamFromUTF8ToAnsi Function 6869

WideStrUtils.DetectUTF8Encoding Function 6869
WideStrUtils.HasExtendCharacter Function 6870
WideStrUtils.HasUTF8BOM Function 6870
WideStrUtils.inOpArray Function 6871
WideStrUtils.inOpSet Function 6871
WideStrUtils.IsUTF8LeadByte Function 6871
WideStrUtils.IsUTF8String Function 6872
WideStrUtils.IsUTF8TrailByte Function 6872
WideStrUtils.LoadResWideString Function 6873
WideStrUtils.LoadWideStr Function 6873
WideStrUtils.UTF8CharLength Function 6874
WideStrUtils.UTF8CharSize Function 6874
WideStrUtils.UTF8LowerCase Function 6875
WideStrUtils.Utf8ToAnsiEx Function 6875
WideStrUtils.UTF8UpperCase Function 6876
WideStrUtils.WideAdjustLineBreaks Function 6876
WideStrUtils.WideDequotedStr Function 6877
WideStrUtils.WideExtractQuotedStr Function 6877
WideStrUtils.WideLastChar Function 6878
WideStrUtils.WideQuotedStr Function 6878
WideStrUtils.WideReplaceStr Function 6879
WideStrUtils.WideReplaceText Function 6880
WideStrUtils.WideStringReplace Function 6880
WideStrUtils.WStrAlloc Function 6881
WideStrUtils.WStrBufSize Function 6881
WideStrUtils.WStrCat Function 6882
WideStrUtils.WStrComp Function 6882
WideStrUtils.WStrCopy Function 6883
WideStrUtils.WStrDispose Function 6883
WideStrUtils.WStrEnd Function 6884
WideStrUtils.WStrLCopy Function 6884
WideStrUtils.WStrLen Function 6885
WideStrUtils.WStrMove Function 6885
WideStrUtils.WStrNew Function 6886
WideStrUtils.WStrPCopy Function 6886
WideStrUtils.WStrPLCopy Function 6887
WideStrUtils.WStrPos Function 6887
WideStrUtils.WStrScan Function 6888

Structs, Records, Enums **6889**

WideStrUtils.TEncodeType Enumeration 6889

Types **6890**

WideStrUtils.CharSet Type 6890

Constants **6891**

WideStrUtils.sUTF8BOMString Constant 6891

WinHelpViewer Namespace 6893

Interfaces **6894**

IWinHelpTester Interface 6894

IWinHelpTester Members 6894

IWinHelpTester Methods 6894

Variables **6897**

WinHelpViewer.ViewerName Variable 6897

WinHelpViewer.WinHelpTester Variable 6897

WSDLBind Namespace 6899

Classes **6902**

TBinding Class 6902

TBinding Members 6903

TBinding Methods 6907

TBindingOperation Class 6909

TBindingOperation Members 6910

TBindingOperation Methods 6915

TBindingOperations Class 6917

TBindingOperations Members 6917

TBindingOperations Methods 6922

TBindings Class 6923

TBindings Members 6924

TBindings Methods 6929

TDefinition Class 6930

TDefinition Members 6930

TDefinition Methods 6935

TDocumentation Class 6938

TDocumentationCollection Class 6939

TDocumentationCollection Members 6939

TDocumentationCollection Methods 6944

TDocumented Class 6945

TDocumented Members 6946
TDocumented Methods 6950
TFault Class 6951
TFaults Class 6952
TFaults Members 6952
TFaults Methods 6957
TImport Class 6958
TImport Members 6959
TImport Methods 6963
TImports Class 6965
TImports Members 6965
TImports Methods 6970
TMessage Class 6971
TMessage Members 6972
TMessage Methods 6976
TMessages Class 6978
TMessages Members 6978
TMessages Methods 6983
TOperation Class 6984
TOperation Members 6985
TOperation Methods 6989
TOperations Class 6991
TOperations Members 6992
TOperations Methods 6997
TParam Class 6998
TParam Members 6999
TParam Methods 7003
TPart Class 7005
TPart Members 7005
TPart Methods 7010
TParts Class 7012
TParts Members 7012
TParts Methods 7017
TPort Class 7018
TPort Members 7019
TPort Methods 7023
TPorts Class 7025
TPorts Members 7025
TPorts Methods 7030

TPortType Class 7031
TPortType Members 7032
TPortType Methods 7036
TPortTypes Class 7038
TPortTypes Members 7038
TPortTypes Methods 7043
TService Class 7044
TService Members 7045
TService Methods 7049
TServices Class 7051
TServices Members 7051
TServices Methods 7056
TTypes Class 7057
TTypes Members 7058
TTypes Methods 7062
TWSDLDocument Class 7063
TWSDLDocument Members 7064
TWSDLDocument Methods 7068
TWSDLDocument Properties 7070
TXMLSchemaDefs Class 7071
TXMLSchemaDefs Members 7071
TXMLSchemaDefs Methods 7076

Interfaces 7078
IBinding Interface 7078
IBinding Members 7079
IBinding Methods 7080
IBinding Properties 7081
IBindingInput Interface 7082
IBindingOperation Interface 7083
IBindingOperation Members 7083
IBindingOperation Methods 7085
IBindingOperation Properties 7086
IBindingOperations Interface 7087
IBindingOperations Members 7087
IBindingOperations Methods 7089
IBindingOperations Properties 7090
IBindingOutput Interface 7090
IBindings Interface 7091
IBindings Members 7091

IBindings Methods 7092
IBindings Properties 7093
IDefinition Interface 7094
IDefinition Members 7094
IDefinition Methods 7096
IDefinition Properties 7098
IDocumentation Interface 7100
IDocumentationCollection Interface 7101
IDocumentationCollection Members 7101
IDocumentationCollection Methods 7103
IDocumentationCollection Properties 7103
IDocumented Interface 7104
IDocumented Members 7104
IDocumented Methods 7105
IDocumented Properties 7106
IFault Interface 7106
IFaults Interface 7107
IFaults Members 7107
IFaults Methods 7109
IFaults Properties 7109
IImport Interface 7110
IImport Members 7110
IImport Methods 7112
IImport Properties 7113
IImports Interface 7113
IImports Members 7114
IImports Methods 7115
IImports Properties 7116
IMessage Interface 7116
IMessage Members 7117
IMessage Methods 7118
IMessage Properties 7119
IMessages Interface 7120
IMessages Members 7120
IMessages Methods 7122
IMessages Properties 7122
IOperation Interface 7123
IOperation Members 7123
IOperation Methods 7125

IOperation Properties 7126
IOperations Interface 7127
IOperations Members 7127
IOperations Methods 7129
IOperations Properties 7130
IParam Interface 7131
IParam Members 7131
IParam Methods 7132
IParam Properties 7134
IPart Interface 7134
IPart Members 7135
IPart Methods 7136
IPart Properties 7138
IParts Interface 7139
IParts Members 7139
IParts Methods 7140
IParts Properties 7141
IPort Interface 7142
IPort Members 7142
IPort Methods 7144
IPort Properties 7145
IPorts Interface 7145
IPorts Members 7146
IPorts Methods 7147
IPorts Properties 7148
IPortType Interface 7148
IPortType Members 7149
IPortType Methods 7150
IPortType Properties 7151
IPortTypes Interface 7152
IPortTypes Members 7152
IPortTypes Methods 7154
IPortTypes Properties 7154
IService Interface 7155
IService Members 7155
IService Methods 7157
IService Properties 7157
IServices Interface 7158
IServices Members 7158

IServices Methods 7160
IServices Properties 7161
ITypes Interface 7161
ITypes Members 7162
ITypes Methods 7163
ITypes Properties 7163
IWSDLDocument Interface 7164
IWSDLDocument Members 7164
IWSDLDocument Methods 7166
IWSDLDocument Properties 7167
IXMLSchemaDefs Interface 7168
IXMLSchemaDefs Members 7168
IXMLSchemaDefs Methods 7169
IXMLSchemaDefs Properties 7170

Functions 7171
WSDLBind.GetDefinition Function 7171
WSDLBind.LoadWSDLDoc Function 7171
WSDLBind.NewWSDLDoc Function 7171

Constants 7173
WSDLBind.SBinding Constant 7173
WSDLBind.SDefinitions Constant 7173
WSDLBind.SFault Constant 7174
WSDLBind.SImport Constant 7174
WSDLBind.SInput Constant 7174
WSDLBind.SLocation Constant 7175
WSDLBind.SMessage Constant 7175
WSDLBind.SNamespace Constant 7175
WSDLBind.SOperation Constant 7176
WSDLBind.SOutput Constant 7176
WSDLBind.SPart Constant 7176
WSDLBind.SPort Constant 7177
WSDLBind.SPortType Constant 7177
WSDLBind.SService Constant 7177
WSDLBind.SType Constant 7178
WSDLBind.STypes Constant 7178

WSDLIntf Namespace 7179

Classes 7181

TDOMStrings Class 7181
TDOMStrings Members 7181
TDOMStrings.Create Constructor 7182
TDOMStrings.Destroy Destructor 7182
TDOMStrings Methods 7182
TDOMStrings Properties 7184

Structs, Records, Enums 7185
WSDLIntf.TWebServiceBindingType Enumeration 7185
WSDLIntf.TWSDLElement Enumeration 7186
WSDLIntf.TWString Record 7186

Types 7187
WSDLIntf.TWSDLElements Type 7187

Constants 7188
WSDLIntf.Directives Constant 7189
WSDLIntf.DynArrayTypes Constant 7189
WSDLIntf.Httpns Constant 7190
WSDLIntf.Operators Constant 7190
WSDLIntf.OtherReservedWords Constant 7190
WSDLIntf.ReservedWords Constant 7191
WSDLIntf.SAddress Constant 7191
WSDLIntf.SAnySimpleType Constant 7192
WSDLIntf.SAnyType Constant 7192
WSDLIntf.SArray Constant 7192
WSDLIntf.SArrayOf Constant 7193
WSDLIntf.SArrayType Constant 7193
WSDLIntf.SBinding Constant 7193
WSDLIntf.SBody Constant 7194
WSDLIntf.ScalarArrayTypes Constant 7194
WSDLIntf.SComplexType Constant 7194
WSDLIntf.SDefinitions Constant 7195
WSDLIntf.SDynArray Constant 7195
WSDLIntf.SElement Constant 7195
WSDLIntf.SEncodingStyle Constant 7196
WSDLIntf.SFault Constant 7196
WSDLIntf.SHeader Constant 7196
WSDLIntf.SImport Constant 7197
WSDLIntf.SInOut Constant 7197
WSDLIntf.SInput Constant 7197

WSDLIntf.SLocation Constant 7198
WSDLIntf.SMessage Constant 7198
WSDLIntf.SName Constant 7198
WSDLIntf.SNameSpace Constant 7199
WSDLIntf.SNsPrefix Constant 7199
WSDLIntf.Soapns Constant 7199
WSDLIntf.SOperation Constant 7200
WSDLIntf.SOutput Constant 7200
WSDLIntf.SPart Constant 7200
WSDLIntf.SParts Constant 7201
WSDLIntf.SPort Constant 7201
WSDLIntf.SPortType Constant 7201
WSDLIntf.SRequest Constant 7202
WSDLIntf.SResponse Constant 7202
WSDLIntf.SReturn Constant 7202
WSDLIntf.SSchema Constant 7203
WSDLIntf.SService Constant 7203
WSDLIntf.SSoapAction Constant 7203
WSDLIntf.SStyle Constant 7204
WSDLIntf.STns Constant 7204
WSDLIntf.STransport Constant 7204
WSDLIntf.SType Constant 7205
WSDLIntf.STypes Constant 7205
WSDLIntf.SUnknown Constant 7205
WSDLIntf.SUse Constant 7206
WSDLIntf.SVerb Constant 7206
WSDLIntf.tns Constant 7206
WSDLIntf.Wsdlns Constant 7207
WSDLIntf.xsdns Constant 7207

WSDLItems Namespace 7209

Classes **7211**

TWSDLItems Class 7211
TWSDLItems Members 7212
Create Constructor 7217
TWSDLItems Methods 7218
TWSDLItems Properties 7245

Interfaces **7247**

IHeaderInfo Interface 7247
IHeaderInfo Members 7247
IHeaderInfo Methods 7248
IHeaderInfo Properties 7251
IQualifiedName Interface 7252
IQualifiedName Members 7253
IQualifiedName Methods 7253
IQualifiedName Properties 7255
IWSDLItems Interface 7256
IWSDLItems Members 7256
IWSDLItems Methods 7256

Functions 7259
WSDLItems.HasDefinition Function 7259
WSDLItems.HasDefinition Function 7259
WSDLItems.NewQualifiedName Function 7259

Structs, Records, Enums 7261
WSDLItems.IterateImportOptions Enumeration 7261

Types 7262
WSDLItems.IHeaderInfoArray Type 7262
WSDLItems.IQualifiedNameArray Type 7262
WSDLItems.TWSDLIterateProc Type 7263

WSDLNode Namespace 7265

Classes 7266
EWSDLLoadException Class 7266
TWSDLClientNode Class 7266
TWSDLClientNode Members 7267
TWSDLClientNode Methods 7267
TWSDLClientNode Properties 7270
TWSDLView Class 7271
TWSDLView Members 7271
TWSDLView Fields 7272
TWSDLView Methods 7272
TWSDLView Properties 7273

Functions 7277
WSDLNode.ActivateWSDL Function 7277

WSDLPub Namespace 7279

Classes **7280**

TWSDLHTMLPublish Class 7280

TWSDLHTMLPublish Members 7281

TWSDLHTMLPublish.Create Constructor 7281

TWSDLHTMLPublish.Destroy Destructor 7282

TWSDLHTMLPublish Methods 7282

TWSDLHTMLPublish Properties 7287

TWSDLHTMLPublish Events 7289

TWSDLPublish Class 7291

TWSDLPublish Members 7291

TWSDLPublish Methods 7292

TWSDLPublish Properties 7293

TWSDLPublish Events 7293

Interfaces **7295**

IWSDLPublish Interface 7295

IWSDLPublish Members 7295

IWSDLPublish Methods 7295

Functions **7297**

WSDLPub.WSDLPubFactory Function 7297

Structs, Records, Enums **7298**

WSDLPub.SOAPPublishOptions Enumeration 7298

Types **7299**

WSDLPub.TBeforePublishingWSDLEvent Type 7299

WSDLPub.TSOAPPublishOptions Type 7299

XMLBrokr Namespace 7301

Classes **7303**

TXMLBroker Class 7303

TXMLBroker Members 7304

TXMLBroker.Create Constructor 7305

TXMLBroker.Destroy Destructor 7306

TXMLBroker Methods 7306

TXMLBroker Properties 7321

TXMLBroker Events 7329

TXMLParams Class 7333
TXMLParams Members 7334
TXMLParams Methods 7334

Interfaces 7337
INotifyConnectionChange Interface 7337
INotifyConnectionChange Members 7337
INotifyConnectionChange Methods 7337

Functions 7339
XMLBrokr.FormatXML Function 7339

Structs, Records, Enums 7340
XMLBrokr.TXMLOption Enumeration 7340

Types 7341
XMLBrokr.TGetErrorResponseEvent Type 7341
XMLBrokr.TGetResponseEvent Type 7341
XMLBrokr.TRequestRecordsEvent Type 7342
XMLBrokr.TRequestUpdateEvent Type 7342
XMLBrokr.TXMLOptions Type 7342

Constants 7343
XMLBrokr.sPostDelta Constant 7343
XMLBrokr.sProducer Constant 7343
XMLBrokr.sRedirect Constant 7343

XMLDoc Namespace 7345

Classes 7347
TXMLDocument Class 7347
TXMLDocument Members 7348
Create Constructor 7351
TXMLDocument.Destroy Destructor 7352
TXMLDocument Methods 7352
TXMLDocument Properties 7380
TXMLDocument Events 7392
TXMLNode Class 7396
TXMLNode Members 7397
TXMLNode.Create Constructor 7400
TXMLNode.CreateHosted Constructor 7401
TXMLNode.Destroy Destructor 7401

TXMLNode Methods 7402
TXMLNode Properties 7437
TXMLNode Events 7443
TXMLNodeCollection Class 7444
TXMLNodeCollection Members 7445
TXMLNodeCollection Methods 7450
TXMLNodeCollection Properties 7457
TXMLNodeList Class 7460
TXMLNodeList Members 7461
TXMLNodeList.Create Constructor 7462
TXMLNodeList.Destroy Destructor 7463
TXMLNodeList Methods 7464
TXMLNodeList Properties 7474
TXMLNodeList Events 7476

Interfaces 7478
IXMLDocumentAccess Interface 7478
IXMLDocumentAccess Members 7478
IXMLDocumentAccess Methods 7478
IXMLDocumentAccess Properties 7479
IXMLNodeAccess Interface 7480
IXMLNodeAccess Members 7480
IXMLNodeAccess Methods 7482
IXMLNodeAccess Properties 7487

Functions 7488
XMLDoc.CheckEncoding Function 7488
XMLDoc.CreateDOMNode Function 7488
XMLDoc.DetectCharEncoding Function 7489
XMLDoc.FormatXMLData Function 7490
XMLDoc.LoadXMLData Function 7490
XMLDoc.LoadXMLDocument Function 7491
XMLDoc.NewXMLDocument Function 7491
XMLDoc.XMLDocError Function 7492
XMLDoc.XMLStringToUnicodeString Function 7492
XMLDoc.XMLStringToWideString Function 7492

Structs, Records, Enums 7494
XMLDoc.TNodeChange Enumeration 7494
XMLDoc.TNodeClassInfo Record 7495
XMLDoc.TNodeListOperation Enumeration 7495

XMLDoc.TXMLDocumentSource Enumeration 7496
XMLDoc.TXMLPrologItem Enumeration 7497

Types **7498**
XMLDoc.TNodeChangeEvent Type 7498
XMLDoc.TNodeClassArray Type 7499
XMLDoc.TNodeListNotification Type 7499
XMLDoc.TXMLNodeArray Type 7500
XMLDoc.TXMLNodeClass Type 7501
XMLDoc.TXMLNodeCollectionClass Type 7501

Variables **7502**
XMLDoc.UseXSDBooleanStrings Variable 7502

xmldom Namespace 7503

Classes **7506**
DOMException Class 7506
DOMException Members 7506
DOMException Fields 7508
EDOMParseError Class 7508
EDOMParseError Members 7509
EDOMParseError.Create Constructor 7510
EDOMParseError Properties 7511
TDOMVendor Class 7514
TDOMVendor Members 7514
TDOMVendor Methods 7514
TDOMVendorList Class 7516
TDOMVendorList Members 7516
TDOMVendorList Methods 7517
TDOMVendorList Properties 7519

Interfaces **7521**
IDOMAttr Interface 7521
IDOMAttr Members 7522
IDOMAttr Methods 7523
IDOMAttr Properties 7524
IDOMCDATASection Interface 7525
IDOMCharacterData Interface 7526
IDOMCharacterData Members 7526
IDOMCharacterData Methods 7527

IDOMCharacterData Properties 7529
IDOMComment Interface 7530
IDOMDocument Interface 7530
IDOMDocument Members 7530
IDOMDocument Methods 7532
IDOMDocument Properties 7536
IDOMDocumentFragment Interface 7537
IDOMDocumentType Interface 7538
IDOMDocumentType Members 7538
IDOMDocumentType Methods 7539
IDOMDocumentType Properties 7541
IDOMElement Interface 7542
IDOMElement Members 7543
IDOMElement Methods 7544
IDOMElement Properties 7548
IDOMEntity Interface 7549
IDOMEntity Members 7549
IDOMEntity Methods 7550
IDOMEntity Properties 7551
IDOMEntityReference Interface 7552
IDOMImplementation Interface 7552
IDOMImplementation Members 7552
IDOMImplementation Methods 7553
IDOMNamedNodeMap Interface 7554
IDOMNamedNodeMap Members 7554
IDOMNamedNodeMap Methods 7554
IDOMNamedNodeMap Properties 7556
IDOMNode Interface 7557
IDOMNode Members 7558
IDOMNode Methods 7559
IDOMNode Properties 7567
IDOMNodeEx Interface 7577
IDOMNodeEx Members 7577
IDOMNodeEx Methods 7578
IDOMNodeEx Properties 7580
IDOMNodeList Interface 7581
IDOMNodeList Members 7581
IDOMNodeList Methods 7581
IDOMNodeList Properties 7582

IDOMNodeSelect Interface 7583
IDOMNodeSelect Members 7583
IDOMNodeSelect Methods 7583
IDOMNotation Interface 7584
IDOMNotation Members 7584
IDOMNotation Methods 7585
IDOMNotation Properties 7586
IDOMParseError Interface 7586
IDOMParseError Members 7587
IDOMParseError Methods 7587
IDOMParseError Properties 7589
IDOMParseOptions Interface 7591
IDOMParseOptions Members 7591
IDOMParseOptions Methods 7592
IDOMParseOptions Properties 7594
IDOMPersist Interface 7595
IDOMPersist Members 7595
IDOMPersist Methods 7596
IDOMPersist Properties 7599
IDOMProcessingInstruction Interface 7599
IDOMProcessingInstruction Members 7599
IDOMProcessingInstruction Methods 7600
IDOMProcessingInstruction Properties 7601
IDOMText Interface 7602
IDOMText Members 7602
IDOMText Methods 7603
IDOMXMLProlog Interface 7604
IDOMXMLProlog Members 7604
IDOMXMLProlog Methods 7605
IDOMXMLProlog Properties 7606
IDOMXSLProcessor Interface 7607
IDOMXSLProcessor Members 7607
IDOMXSLProcessor Methods 7608
IDOMXSLProcessor Properties 7610

Functions 7612

xmldom.DOMVendorNotSupported Function 7612
xmldom.ExtractLocalName Function 7612
xmldom.ExtractPrefix Function 7613
xmldom.GetDOM Function 7614

xmldom.GetDOMNodeEx Function 7614
xmldom.GetDOMVendor Function 7615
xmldom.IsPrefixed Function 7615
xmldom.MakeNodeName Function 7616
xmldom.NodeMatches Function 7616
xmldom.RegisterDOMVendor Function 7617
xmldom.SameNamespace Function 7617
xmldom.UnRegisterDOMVendor Function 7618

Types **7619**
xmldom.DOMNodeType Type 7619
xmldom.DOMString Type 7619
xmldom.DOMTimeStamp Type 7620
xmldom.TAsyncEventHandler Type 7620
xmldom.TDOMVendorArray Type 7621

Variables **7622**
xmldom.DefaultDOMVendor Variable 7622
xmldom.DOMVendors Variable 7622

Constants **7624**
xmldom.ATTRIBUTE_NODE Constant 7624
xmldom.CDATA_SECTION_NODE Constant 7625
xmldom.COMMENT_NODE Constant 7625
xmldom.DOCUMENT_FRAGMENT_NODE Constant 7625
xmldom.DOCUMENT_NODE Constant 7626
xmldom.DOCUMENT_TYPE_NODE Constant 7626
xmldom.DOMSTRING_SIZE_ERR Constant 7626
xmldom.DOMWrapperVersion Constant 7627
xmldom.ELEMENT_NODE Constant 7627
xmldom.ENTITY_NODE Constant 7627
xmldom.ENTITY_REFERENCE_NODE Constant 7628
xmldom.HIERARCHY_REQUEST_ERR Constant 7628
xmldom.INDEX_SIZE_ERR Constant 7628
xmldom.INUSE_ATTRIBUTE_ERR Constant 7629
xmldom.INVALID_ACCESS_ERR Constant 7629
xmldom.INVALID_CHARACTER_ERR Constant 7629
xmldom.INVALID_MODIFICATION_ERR Constant 7630
xmldom.INVALID_STATE_ERR Constant 7630
xmldom.NAMESPACE_ERR Constant 7630
xmldom.NO_DATA_ALLOWED_ERR Constant 7631

xmldom.NO_MODIFICATION_ALLOWED_ERR Constant 7631
xmldom.NOT_FOUND_ERR Constant 7631
xmldom.NOT_SUPPORTED_ERR Constant 7632
xmldom.NOTATION_NODE Constant 7632
xmldom.NSDelim Constant 7632
xmldom.PROCESSING_INSTRUCTION_NODE Constant 7633
xmldom.SEncoding Constant 7633
xmldom.SHttp Constant 7633
xmldom.SStandalone Constant 7634
xmldom.SVersion Constant 7634
xmldom.SXML Constant 7634
xmldom.SXMLNamespaceURI Constant 7635
xmldom.SXMLNS Constant 7635
xmldom.SXMLPrefixNamespaceURI Constant 7635
xmldom.SYNTAX_ERR Constant 7636
xmldom.TEXT_NODE Constant 7636
xmldom.WRONG_DOCUMENT_ERR Constant 7636

XMLIntf Namespace 7637

Classes 7639
EXMLDocError Class 7639

Interfaces 7640
IXMLDocument Interface 7640
IXMLDocument Members 7641
IXMLDocument Methods 7642
IXMLDocument Properties 7661
IXMLNode Interface 7669
IXMLNode Members 7670
IXMLNode Methods 7671
IXMLNode Properties 7687
IXMLNodeCollection Interface 7698
IXMLNodeCollection Members 7698
IXMLNodeCollection Methods 7700
IXMLNodeCollection Properties 7702
IXMLNodeList Interface 7703
IXMLNodeList Members 7704
IXMLNodeList Methods 7705
IXMLNodeList Properties 7714

Structs, Records, Enums **7716**
XMLIntf.TNodeType Enumeration 7716
XMLIntf.TParseOption Enumeration 7718
XMLIntf.TXMLDocOption Enumeration 7718
XMLIntf.TXMLEncodingType Enumeration 7719

Types **7721**
XMLIntf.TParseOptions Type 7721
XMLIntf.TXMLDocOptions Type 7722

Constants **7723**
XMLIntf.NodeTypeNames Constant 7723

Xmlxform Namespace 7725

Classes **7726**
TranslateException Class 7726
TXMLTransform Class 7727
TXMLTransform Members 7727
TXMLTransform Methods 7728
TXMLTransform Properties 7731
TXMLTransform Events 7737
TXMLTransformClient Class 7739
TXMLTransformClient Members 7740
TXMLTransformClient.Create Constructor 7741
TXMLTransformClient.Destroy Destructor 7741
TXMLTransformClient Methods 7741
TXMLTransformClient Properties 7744
TXMLTransformProvider Class 7747
TXMLTransformProvider Members 7748
TXMLTransformProvider.Create Constructor 7749
TXMLTransformProvider.Destroy Destructor 7749
TXMLTransformProvider Methods 7750
TXMLTransformProvider Properties 7751
TXMLTransformProvider Events 7754

Types **7761**
Xmlxform.TRowEvent Type 7761
Xmlxform.TTranslateEvent Type 7762

XPActnCtrls Namespace 7763

Classes **7764**
TXPStyleAddRemoveItem Class 7764
TXPStyleAddRemoveItem Members 7764
TXPStyleAddRemoveItem Methods 7765
TXPStyleButton Class 7767
TXPStyleButton Members 7767
TXPStyleButton Methods 7767
TXPStyleCustomizePopup Class 7769
TXPStyleCustomizePopup Members 7769
TXPStyleCustomizePopup.Create Constructor 7769
TXPStyleCustomizePopup Methods 7770
TXPStyleDropDownBtn Class 7770
TXPStyleDropDownBtn Members 7771
TXPStyleDropDownBtn Methods 7771
TXPStyleExpandBtn Class 7772
TXPStyleExpandBtn Members 7773
TXPStyleExpandBtn Methods 7773
TXPStyleMenuButton Class 7774
TXPStyleMenuButton Members 7774
TXPStyleMenuButton Methods 7774
TXPStyleMenuItem Class 7775
TXPStyleMenuItem Members 7775
TXPStyleMenuItem Methods 7776
TXPStylePopupMenu Class 7778
TXPStylePopupMenu Members 7779
TXPStylePopupMenu.Create Constructor 7779
TXPStylePopupMenu.Destroy Destructor 7779
TXPStylePopupMenu Methods 7780
TXPStyleToolScrollBtn Class 7782
TXPStyleToolScrollBtn Members 7782
TXPStyleToolScrollBtn Methods 7782

XPMan Namespace 7785

Classes **7786**
TXPManifest Class 7786

XPStyleActnCtrls Namespace 7787

Classes **7788**
TXPStyleActionBars Class 7788
TXPStyleActionBars Members 7788
TXPStyleActionBars Methods 7788
Variables **7791**
XPStyleActnCtrls.XPStyle Variable 7791

XSBuiltIns Namespace 7793

Classes **7795**
EXSDateTimeException Class 7795
EXSDecimalException Class 7796
EXSHexBinaryException Class 7796
TXMLData Class 7797
TXMLData Members 7797
TXMLData.Create Constructor 7798
TXMLData.Destroy Destructor 7798
TXMLData Methods 7798
TXMLData Properties 7801
TXSBoolean Class 7801
TXSBoolean Members 7801
TXSBoolean Methods 7802
TXSBoolean Properties 7803
TXSCustomDateTime Class 7803
TXSCustomDateTime Members 7804
TXSCustomDateTime.Create Constructor 7805
TXSCustomDateTime.Destroy Destructor 7805
TXSCustomDateTime Fields 7806
TXSCustomDateTime Methods 7806
TXSCustomDateTime Properties 7815
TXSDate Class 7822
TXSDate Members 7822
TXSDate.Create Constructor 7823
TXSDate Methods 7823
TXSDate Properties 7826
TXSDateTime Class 7828
TXSDateTime Members 7829
TXSDateTime Methods 7830
TXSDateTime Properties 7832

TXSDecimal Class 7832
TXSDecimal Members 7833
TXSDecimal Methods 7833
TXSDecimal Properties 7834
TXSDuration Class 7836
TXSDuration Members 7836
TXSDuration.Create Constructor 7838
TXSDuration Methods 7838
TXSDuration Properties 7839
TXSHexBinary Class 7840
TXSHexBinary Members 7841
TXSHexBinary Methods 7841
TXSHexBinary Properties 7842
TXSInteger Class 7843
TXSInteger Members 7843
TXSInteger Methods 7844
TXSInteger Properties 7845
TXSLong Class 7845
TXSLong Members 7845
TXSLong Methods 7846
TXSLong Properties 7847
TXSString Class 7847
TXSString Members 7847
TXSString Methods 7848
TXSTime Class 7849
TXSTime Members 7849
TXSTime Methods 7850
TXSTime Properties 7856

Functions **7861**

XSBuiltIns.DateTimeToXMLTime Function 7861
XSBuiltIns.DateTimeToXSDateTime Function 7861
XSBuiltIns.GetDataFromFile Function 7862
XSBuiltIns.InitXSTypes Function 7862
XSBuiltIns.SoapFloatToStr Function 7863
XSBuiltIns.SoapStrToFloat Function 7863
XSBuiltIns.XMLTimeToDateTime Function 7863

Types **7865**

XSBuiltIns.TXSTimeInstant Type 7865

Constants **7866**
XSBuiltIns.IncrementAmount Constant 7866
XSBuiltIns.MaxDay Constant 7867
XSBuiltIns.MaxMonth Constant 7867
XSBuiltIns.MinDay Constant 7867
XSBuiltIns.MinMonth Constant 7868
XSBuiltIns.SDecimal Constant 7868
XSBuiltIns.SHexMarker Constant 7868
XSBuiltIns.SLocalTimeMarker Constant 7869
XSBuiltIns.SNAN Constant 7869
XSBuiltIns.SNegative Constant 7869
XSBuiltIns.SoapDecimalSeparator Constant 7870
XSBuiltIns.SoapTimePrefix Constant 7870
XSBuiltIns.SPlus Constant 7870
XSBuiltIns.SSciNotationMarker Constant 7871
XSBuiltIns.XMLDateSeparator Constant 7871
XSBuiltIns.XMLDayPos Constant 7871
XSBuiltIns.XMLDefaultYearDigits Constant 7872
XSBuiltIns.XMLDurationDay Constant 7872
XSBuiltIns.XMLDurationHour Constant 7872
XSBuiltIns.XMLDurationMinute Constant 7873
XSBuiltIns.XMLDurationMonth Constant 7873
XSBuiltIns.XMLDurationSecond Constant 7873
XSBuiltIns.XMLDurationStart Constant 7874
XSBuiltIns.XMLDurationYear Constant 7874
XSBuiltIns.XMLHourOffsetMinusMarker Constant 7874
XSBuiltIns.XMLHourOffsetPlusMarker Constant 7875
XSBuiltIns.XMLMilSecPos Constant 7875
XSBuiltIns.XMLMonthPos Constant 7875
XSBuiltIns.XMLTimeSeparator Constant 7876
XSBuiltIns.XMLYearPos Constant 7876

XSLProd Namespace 7877

Classes **7878**
EXSLPageProducerError Class 7878
TCustomXSLPageProducer Class 7878
TCustomXSLPageProducer Members 7879
TCustomXSLPageProducer Methods 7884

TCustomXSLPageProducer Properties 7891
TXSLPageProducer Class 7893
TXSLPageProducer Members 7894
TXSLPageProducer Properties 7899

ZLib Namespace 7901

Classes 7904
EZCompressionError Class 7904
EZDecompressionError Class 7904
EZLibError Class 7905
TCustomZStream Class 7905
TCustomZStream Members 7906
TCustomZStream.Create Constructor 7906
TCustomZStream Methods 7906
TCustomZStream Events 7907
TZCompressionStream Class 7907
TZCompressionStream Members 7908
Create Constructor 7908
TZCompressionStream.Destroy Destructor 7909
TZCompressionStream Methods 7909
TZCompressionStream Properties 7911
TZCompressionStream Events 7911
TZDecompressionStream Class 7912
TZDecompressionStream Members 7913
TZDecompressionStream.Create Constructor 7913
TZDecompressionStream.Destroy Destructor 7913
TZDecompressionStream Methods 7914
TZDecompressionStream Events 7915

Functions 7917
ZLib.adler32 Function 7917
ZLib.deflate Function 7918
ZLib.deflateEnd Function 7918
ZLib.deflateInit_ Function 7918
ZLib.DeflateInit2_ Function 7919
ZLib.inflate Function 7919
ZLib.inflateEnd Function 7919
ZLib.inflateInit_ Function 7920
ZLib.inflateInit2_ Function 7920

ZLib.inflateReset Function 7920
ZLib.MoveI32 Function 7921
ZLib.ZCompress Function 7921
ZLib.ZCompress Function 7921
ZLib.ZCompressStr Function 7922
ZLib.ZCompressStr Function 7922
ZLib.ZCompressStream Function 7922
ZLib.ZCompressStream Function 7923
ZLib.ZDecompress Function 7923
ZLib.ZDecompressStr Function 7923
ZLib.ZDecompressStream Function 7924
ZLib.zlibAllocMem Function 7924
ZLib.zlibFreeMem Function 7924
ZLib.ZSendToBrowser Function 7925

Structs, Records, Enums **7926**
ZLib.TCompressionLevel Enumeration 7926
ZLib.TZCompressionLevel Enumeration 7927
ZLib.TZStreamRec Record 7927

Types **7929**
ZLib.TCompressionStream Type 7929
ZLib.TCustomZLibStream Type 7929
ZLib.TDecompressionStream Type 7930
ZLib.TZAlloc Type 7930
ZLib.TZFree Type 7930

Constants **7932**
ZLib._z_errmsg Constant 7932
ZLib.SZInvalid Constant 7933
ZLib.Z_ASCII Constant 7933
ZLib.Z_BEST_COMPRESSION Constant 7933
ZLib.Z_BEST_SPEED Constant 7934
ZLib.Z_BINARY Constant 7934
ZLib.Z_BUF_ERROR Constant 7934
ZLib.Z_DATA_ERROR Constant 7935
ZLib.Z_DEFAULT_COMPRESSION Constant 7935
ZLib.Z_DEFAULT_STRATEGY Constant 7935
ZLib.Z_DEFLATED Constant 7936
ZLib.Z_ERRNO Constant 7936
ZLib.Z_FILTERED Constant 7936

ZLib.Z_FINISH Constant 7937
ZLib.Z_FULL_FLUSH Constant 7937
ZLib.Z_HUFFMAN_ONLY Constant 7937
ZLib.Z_MEM_ERROR Constant 7938
ZLib.Z_NEED_DICT Constant 7938
ZLib.Z_NO_COMPRESSION Constant 7938
ZLib.Z_NO_FLUSH Constant 7939
ZLib.Z_OK Constant 7939
ZLib.Z_PARTIAL_FLUSH Constant 7939
ZLib.Z_STREAM_END Constant 7940
ZLib.Z_STREAM_ERROR Constant 7940
ZLib.Z_SYNC_FLUSH Constant 7940
ZLib.Z_UNKNOWN Constant 7941
ZLib.Z_VERSION_ERROR Constant 7941
ZLib.ZLevels Constant 7941
ZLib.ZLIB_VERSION Constant 7942

Index a

1 SocktCnst Namespace

This is namespace SocktCnst.

Topics

Name	Description
Constants (see page 2)	The following table lists constants in this documentation.

Constants

Name	Description
ckInterceptGUID (see page 2)	This is constant SocktCnst.ckInterceptGUID.
ckPort (see page 3)	This is constant SocktCnst.ckPort.
ckRegistered (see page 3)	This is constant SocktCnst.ckRegistered.
ckShowHost (see page 3)	This is constant SocktCnst.ckShowHost.
ckThreadCacheSize (see page 4)	This is constant SocktCnst.ckThreadCacheSize.
ckTimeout (see page 4)	This is constant SocktCnst.ckTimeout.
csSettings (see page 4)	This is constant SocktCnst.csSettings.
KEY_IE (see page 5)	This is constant SocktCnst.KEY_IE.
KEY_SOCKETSERVER (see page 5)	This is constant SocktCnst.KEY_SOCKETSERVER.
SAlreadyRunning (see page 5)	This is constant SocktCnst.SAlreadyRunning.
SApplicationName (see page 6)	This is constant SocktCnst.SApplicationName.
SErrChangeSettings (see page 6)	This is constant SocktCnst.SErrChangeSettings.
SErrClose (see page 6)	This is constant SocktCnst.SErrClose.
SHostUnknown (see page 7)	This is constant SocktCnst.SHostUnknown.
SNotShown (see page 7)	This is constant SocktCnst.SNotShown.
SNotUntilRestart (see page 7)	This is constant SocktCnst.SNotUntilRestart.
SNoWinSock2 (see page 8)	This is constant SocktCnst.SNoWinSock2.
SOpenError (see page 8)	This is constant SocktCnst.SOpenError.
SQueryDisconnect (see page 8)	This is constant SocktCnst.SQueryDisconnect.
SServiceName (see page 9)	This is constant SocktCnst.SServiceName.
SServiceOnly (see page 9)	This is constant SocktCnst.SServiceOnly.
SStatusline (see page 9)	This is constant SocktCnst.SStatusline.

1.1 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ckInterceptGUID (see page 2)	This is constant SocktCnst.ckInterceptGUID.
ckPort (see page 3)	This is constant SocktCnst.ckPort.
ckRegistered (see page 3)	This is constant SocktCnst.ckRegistered.
ckShowHost (see page 3)	This is constant SocktCnst.ckShowHost.
ckThreadCacheSize (see page 4)	This is constant SocktCnst.ckThreadCacheSize.
ckTimeout (see page 4)	This is constant SocktCnst.ckTimeout.
csSettings (see page 4)	This is constant SocktCnst.csSettings.
KEY_IE (see page 5)	This is constant SocktCnst.KEY_IE.
KEY_SOCKETSERVER (see page 5)	This is constant SocktCnst.KEY_SOCKETSERVER.
SAlreadyRunning (see page 5)	This is constant SocktCnst.SAlreadyRunning.
SApplicationName (see page 6)	This is constant SocktCnst.SApplicationName.
SErrChangeSettings (see page 6)	This is constant SocktCnst.SErrChangeSettings.
SErrClose (see page 6)	This is constant SocktCnst.SErrClose.
SHostUnknown (see page 7)	This is constant SocktCnst.SHostUnknown.
SNotShown (see page 7)	This is constant SocktCnst.SNotShown.
SNotUntilRestart (see page 7)	This is constant SocktCnst.SNotUntilRestart.
SNoWinSock2 (see page 8)	This is constant SocktCnst.SNoWinSock2.
SOpenError (see page 8)	This is constant SocktCnst.SOpenError.
SQueryDisconnect (see page 8)	This is constant SocktCnst.SQueryDisconnect.
SServiceName (see page 9)	This is constant SocktCnst.SServiceName.
SServiceOnly (see page 9)	This is constant SocktCnst.SServiceOnly.
SStatusline (see page 9)	This is constant SocktCnst.SStatusline.

1.1.1 SocktCnst.ckInterceptGUID Constant

Pascal

```
ckInterceptGUID = 'InterceptGUID';
```

C++

```
ckInterceptGUID = 'InterceptGUID';
```

File

SocktCnst

Description

This is constant SocktCnst.ckInterceptGUID.

1.1.2 SocktCnst.ckPort Constant

Pascal

```
ckPort = 'Port';
```

C++

```
ckPort = 'Port';
```

File

SocktCnst

Description

This is constant SocktCnst.ckPort.

1.1.3 SocktCnst.ckRegistered Constant

Pascal

```
ckRegistered = 'RegisteredOnly';
```

C++

```
ckRegistered = 'RegisteredOnly';
```

File

SocktCnst

Description

This is constant SocktCnst.ckRegistered.

1.1.4 SocktCnst.ckShowHost Constant

Pascal

```
ckShowHost = 'ShowHost';
```

C++

```
ckShowHost = 'ShowHost';
```

File

SocktCnst

Description

This is constant SocktCnst.ckShowHost.

1.1.5 SocktCnst.ckThreadCacheSize Constant

Pascal

```
ckThreadCacheSize = 'ThreadCacheSize';
```

C++

```
ckThreadCacheSize = 'ThreadCacheSize';
```

File

SocktCnst

Description

This is constant SocktCnst.ckThreadCacheSize.

1.1.6 SocktCnst.ckTimeout Constant

Pascal

```
ckTimeout = 'Timeout';
```

C++

```
ckTimeout = 'Timeout';
```

File

SocktCnst

Description

This is constant SocktCnst.ckTimeout.

1.1.7 SocktCnst.csSettings Constant

Pascal

```
csSettings = 'Settings';
```

C++

```
csSettings = 'Settings';
```

File

SocktCnst

Description

This is constant SocktCnst.csSettings.

1.1.8 SocktCnst.KEY_IE Constant

Pascal

```
KEY_IE = 'SOFTWARE\Microsoft\Internet Explorer';
```

C++

```
KEY_IE = 'SOFTWARE\Microsoft\Internet Explorer';
```

File

SocktCnst

Description

This is constant SocktCnst.KEY_IE.

1.1.9 SocktCnst.KEY_SOCKETSERVER Constant

Pascal

```
KEY_SOCKETSERVER = '\SOFTWARE\Borland\Socket Server';
```

C++

```
KEY_SOCKETSERVER = '\SOFTWARE\Borland\Socket Server';
```

File

SocktCnst

Description

This is constant SocktCnst.KEY_SOCKETSERVER.

1.1.10 SocktCnst.SAlreadyRunning Constant

Pascal

```
SAlreadyRunning = 'The Socket Server is already running';
```

C++

```
SAlreadyRunning = 'The Socket Server is already running';
```

File

SocktCnst

Description

This is constant SocktCnst.SAlreadyRunning.

1.1.11 ScktCnst.SApplicationName Constant

Pascal

```
SApplicationName = 'Borland Socket Server';
```

C++

```
SApplicationName = 'Borland Socket Server';
```

File

ScktCnst

Description

This is constant ScktCnst.SApplicationName.

1.1.12 ScktCnst.SErrChangeSettings Constant

Pascal

```
SErrChangeSettings = 'Cannot change settings when there are active connections. Kill
connections?';
```

C++

```
SErrChangeSettings = 'Cannot change settings when there are active connections. Kill
connections?';
```

File

ScktCnst

Description

This is constant ScktCnst.SErrChangeSettings.

1.1.13 ScktCnst.SErrClose Constant

Pascal

```
SErrClose = 'Cannot exit when there are active connections. Kill connections?';
```

C++

```
SErrClose = 'Cannot exit when there are active connections. Kill connections?';
```

File

ScktCnst

Description

This is constant ScktCnst.SErrClose.

1.1.14 SocktCnst.SHostUnknown Constant

Pascal

```
SHostUnknown = '(Unknown)';
```

C++

```
SHostUnknown = '(Unknown)';
```

File

SocktCnst

Description

This is constant SocktCnst.SHostUnknown.

1.1.15 SocktCnst.SNotShown Constant

Pascal

```
SNotShown = '(Not Shown)';
```

C++

```
SNotShown = '(Not Shown)';
```

File

SocktCnst

Description

This is constant SocktCnst.SNotShown.

1.1.16 SocktCnst.SNotUntilRestart Constant

Pascal

```
SNotUntilRestart = 'This change will not take affect until the Socket Server is restarted';
```

C++

```
SNotUntilRestart = 'This change will not take affect until the Socket Server is restarted';
```

File

SocktCnst

Description

This is constant SocktCnst.SNotUntilRestart.

1.1.17 ScktCnst.SNoWinSock2 Constant

Pascal

```
SNoWinSock2 = 'WinSock 2 must be installed to use the socket connection';
```

C++

```
SNoWinSock2 = 'WinSock 2 must be installed to use the socket connection';
```

File

ScktCnst

Description

This is constant ScktCnst.SNoWinSock2.

1.1.18 ScktCnst.SOpenError Constant

Pascal

```
SOpenError = 'Error opening port %d with error: %s';
```

C++

```
SOpenError = 'Error opening port %d with error: %s';
```

File

ScktCnst

Description

This is constant ScktCnst.SOpenError.

1.1.19 ScktCnst.SQueryDisconnect Constant

Pascal

```
SQueryDisconnect = 'Disconnecting clients can cause application errors. Continue?';
```

C++

```
SQueryDisconnect = 'Disconnecting clients can cause application errors. Continue?';
```

File

ScktCnst

Description

This is constant ScktCnst.SQueryDisconnect.

1.1.20 SocktCnst.SServiceName Constant

Pascal

```
SServiceName = 'SocketServer';
```

C++

```
SServiceName = 'SocketServer';
```

File

SocktCnst

Description

This is constant SocktCnst.SServiceName.

1.1.21 SocktCnst.SServiceOnly Constant

Pascal

```
SServiceOnly = 'The Socket Server can only be run as a service on NT 3.51 and prior';
```

C++

```
SServiceOnly = 'The Socket Server can only be run as a service on NT 3.51 and prior';
```

File

SocktCnst

Description

This is constant SocktCnst.SServiceOnly.

1.1.22 SocktCnst.SStatusline Constant

Pascal

```
SStatusline = '%d current connections';
```

C++

```
SStatusline = '%d current connections';
```

File

SocktCnst

Description

This is constant SocktCnst.SStatusline.

2 ScktComp Namespace

This is namespace ScktComp.

Topics

Name	Description
Classes (see page 13)	The following table lists classes in this documentation.
Functions (see page 144)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 145)	The following table lists structs, records, enums in this documentation.
Types (see page 150)	The following table lists types in this documentation.
Constants (see page 154)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESocketError (see page 13)	ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TAbstractSocket (see page 14)	TAbstractSocket is the abstract base class for all socket components.
	TClientSocket (see page 24)	TClientSocket manages socket connections for a TCP/IP client.
	TClientWinSocket (see page 35)	TClientWinSocket describes the client endpoint of a Windows socket connection.
	TCustomServerSocket (see page 39)	TCustomServerSocket is the base class for all server socket components.
	TCustomSocket (see page 56)	TCustomSocket is the base class for socket components.
	TCustomWinSocket (see page 64)	TCustomWinSocket is the base class for all Windows socket objects.
	TServerAcceptThread (see page 90)	This is class ScktComp.TServerAcceptThread.
	TServerClientThread (see page 95)	TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.
	TServerClientWinSocket (see page 104)	TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.
	TServerSocket (see page 108)	TServerSocket manages server socket connections for a TCP/IP server.
	TServerWinSocket (see page 120)	TServerWinSocket describes the endpoint of a listening server socket connection.
	TWinSocketStream (see page 138)	TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

Constants

Name	Description
CM_DEFERFREE (see page 154)	This is constant ScktComp.CM_DEFERFREE.
CM_LOOKUPCOMPLETE (see page 154)	This is constant ScktComp.CM_LOOKUPCOMPLETE.
CM_SOCKETMESSAGE (see page 154)	This is constant ScktComp.CM_SOCKETMESSAGE.

Functions

	Name	Description
	SetErrorProc (see page 144)	Replaces the exception handler for error messages that are received from a Windows socket connection.

Structs, Records, Enums

	Name	Description
	TAsyncStyle (see page 145)	This is record ScktComp.TAsyncStyle.
	TClientType (see page 146)	This is record ScktComp.TClientType.
	TCMLookupComplete (see page 146)	This is record ScktComp.TCMLookupComplete.
	TCMSocketMessage (see page 147)	This is record ScktComp.TCMSocketMessage.
	TErrorEvent (see page 147)	This is record ScktComp.TErrorEvent.
	TLookupState (see page 148)	This is record ScktComp.TLookupState.
	TServerType (see page 148)	This is record ScktComp.TServerType.
	TSocketEvent (see page 148)	TSocketEvent identifies the operation that a socket is performing.

Types

Name	Description
TAsyncStyles (see page 150)	This is type ScktComp.TAsyncStyles.
TGetSocketEvent (see page 150)	This is type ScktComp.TGetSocketEvent.
TGetThreadEvent (see page 151)	This is type ScktComp.TGetThreadEvent.
TSocketErrorEvent (see page 151)	TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.
TSocketErrorProc (see page 152)	This is type ScktComp.TSocketErrorProc.
TSocketEventEvent (see page 152)	This is type ScktComp.TSocketEventEvent.
TSocketNotifyEvent (see page 152)	The type for event handlers which respond to events during the expected course of forming or using a socket connection.
TThreadNotifyEvent (see page 153)	This is type ScktComp.TThreadNotifyEvent.

2.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESocketError (see page 13)	ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TAbstractSocket (see page 14)	TAbstractSocket is the abstract base class for all socket components.
	TClientSocket (see page 24)	TClientSocket manages socket connections for a TCP/IP client.
	TClientWinSocket (see page 35)	TClientWinSocket describes the client endpoint of a Windows socket connection.
	TCustomServerSocket (see page 39)	TCustomServerSocket is the base class for all server socket components.
	TCustomSocket (see page 56)	TCustomSocket is the base class for socket components.
	TCustomWinSocket (see page 64)	TCustomWinSocket is the base class for all Windows socket objects.
	TServerAcceptThread (see page 90)	This is class ScktComp.TServerAcceptThread.
	TServerClientThread (see page 95)	TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.
	TServerClientWinSocket (see page 104)	TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.
	TServerSocket (see page 108)	TServerSocket manages server socket connections for a TCP/IP server.
	TServerWinSocket (see page 120)	TServerWinSocket describes the endpoint of a listening server socket connection.
	TWinSocketStream (see page 138)	TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

2.1.1 ESocketError Class

ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.

Class Hierarchy

Pascal

```
ESocketError = class(Exception);
```

C++

```
class ESocketError : public Exception;
```

File

ScktComp

Description

ESocketError is raised if

A socket cannot be created.

A socket cannot be initialized.

A socket cannot be opened.

An application tries to change the properties of an already open socket.

A failure occurs when reading from or writing to a socket.

A socket cannot be properly closed.

There are two ESocketError classes, one in the ScktComp (see page 11) unit and one in the Sockets (see page 1367) unit. The two exception classes are identically defined and serve the same function for different categories of socket objects. The two definitions occur simply to ensure that ESocketError is defined for all socket objects.

See Also

TCustomWinSocket (see page 64)

2.1.2 TAbstractSocket Class

TAbstractSocket is the abstract base class for all socket components.

Class Hierarchy

Pascal

```
TAbstractSocket = class(TComponent);
```

C++

```
class TAbstractSocket : public TComponent;
```

File

ScktComp

Description

TAbstractSocket introduces properties and methods to enable an application to work with sockets. A socket encapsulates a set of communication protocols to allow the application to connect to other machines for reading or writing information. Sockets (see page 1367) provide connections based on the TCP/IP protocol. They also allow connections that use the Xerox Network System (see page 3159) (XNS), Digital's DECnet protocol, or Novell's IPX/SPX family. Sockets (see page 1367) allow an application to form connections to other machines without being concerned with the details of the actual networking software.

The properties of TAbstractSocket describe the IP address of the socket and service it provides or seeks. Not all descendants of TAbstractSocket use all of these properties. For example, server sockets do not surface the IP address because it is read implicitly from the system running the application.

Do not create instances of TAbstractSocket. Use a descendant of TAbstractSocket to add the ability to communicate with other machines to an application. To create a socket that initiates the connection with another machine, use TClientSocket (see page 24). To create a socket that responds to requests for connections from other machines, use TServerSocket (see page 108).

See Also

TClientSocket (see page 24)

TCustomWinSocket (see page 64)

TServerSocket (see page 108)

TCustomSocket (see page 56)

2.1.2.1 TAbstractSocket Members

The following tables list the members exposed by TAbstractSocket.

Protected Methods

	Name	Description
	DoActivate (see page 16)	Provides the prototype of a method to open or close the socket connection.
	Error (see page 16)	Provides the prototype of a method that responds to error conditions.
	Event (see page 17)	Provides the prototype of a method that responds when specific socket events occur.
	InitSocket (see page 18)	Tells a Windows socket object to call the Error (see page 16) and Event (see page 17) methods.
	Loaded (see page 18)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 21) property.
	SetActive (see page 19)	Sets the Active (see page 21) property.
	SetAddress (see page 20)	Sets the Address (see page 22) property.
	SetHost (see page 20)	Sets the Host (see page 22) property.
	SetPort (see page 20)	Sets the Port (see page 23) property.
	SetService (see page 21)	Sets the Service (see page 23) property.

Protected Properties

	Name	Description
	Active (see page 21)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 22)	Specifies the IP address of the server system.
	Host (see page 22)	Specifies an alias for the IP address of the server system.
	Port (see page 23)	Specifies the ID number that identifies the server socket connection.
	Service (see page 23)	Specifies the name of the service for which the socket connection is used.

Public Methods

	Name	Description
	Close (see page 16)	Shuts down the socket connection.
	Open (see page 19)	Opens the socket connection.

2.1.2.2 TAbstractSocket Methods

The methods of the TAbstractSocket class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 16)	Provides the prototype of a method to open or close the socket connection.
	Error (see page 16)	Provides the prototype of a method that responds to error conditions.
	Event (see page 17)	Provides the prototype of a method that responds when specific socket events occur.
	InitSocket (see page 18)	Tells a Windows socket object to call the Error (see page 16) and Event (see page 17) methods.
	Loaded (see page 18)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 21) property.
	SetActive (see page 19)	Sets the Active (see page 21) property.
	SetAddress (see page 20)	Sets the Address (see page 22) property.
	SetHost (see page 20)	Sets the Host (see page 22) property.
	SetPort (see page 20)	Sets the Port (see page 23) property.
	SetService (see page 21)	Sets the Service (see page 23) property.

Public Methods

	Name	Description
	Close (see page 16)	Shuts down the socket connection.
	Open (see page 19)	Opens the socket connection.

2.1.2.2.1 TAbstractSocket.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 21) property to false.

For client sockets, Close terminates the connection to the server. For server sockets, Close shuts down the connection so that the server socket is no longer listening for client requests.

See Also

Active (see page 21)

2.1.2.2.2 TAbstractSocket.DoActivate Method

Provides the prototype of a method to open or close the socket connection.

Pascal

```
procedure DoActivate(Value: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall DoActivate(Boolean Value) = 0;
```

Description

Descendants of TAbstractSocket (see page 14) override DoActivate to provide a method that initiates or terminates the socket connection. The steps that need to be taken to open or close a connection depend on whether the socket is a client socket or a server socket.

See Also

Active (see page 21)

Open (see page 19)

Close (see page 16)

2.1.2.2.3 TAbstractSocket.Error Method

Provides the prototype of a method that responds to error conditions.

Pascal

```
procedure Error(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer);
virtual; abstract;
```

C++

```
virtual __fastcall Error(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode) = 0;
```

Description

Descendants of TAbstractSocket (see page 14) must override the abstract or, in C++ terminology, pure virtual Error method to provide a response when socket errors occur. Applications cannot call Error. The Windows socket object that encounters the error condition calls Error automatically.

The Socket parameter indicates the Windows socket object that encounters the error condition.

The ErrorEvent parameter indicates what the socket was trying to do when the error occurred. It has one of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	For client sockets, this indicates that the client socket can't locate the server, or that a problem on the server prevents the opening of a connection. For server sockets, this indicates that a client connection request that has already been accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.
eeAccept	For server sockets only, this indicates that a problem occurred when trying to accept a client connection request.

The ErrorCode parameter is the error code received by the Windows socket object. Setting this value to 0 in the Error method prevents the socket from raising an exception. For information on possible error codes, see the Microsoft documentation on Windows sockets.

See Also

OnErrorEvent (see page 89)

InitSocket (see page 18)

2.1.2.2.4 TAbstractSocket.Event Method

Provides the prototype of a method that responds when specific socket events occur.

Pascal

```
procedure Event(Socket: TCustomWinSocket; SocketEvent: TSocketEvent); virtual; abstract;
```

C++

```
virtual __fastcall Event(TCustomWinSocket Socket, TSocketEvent SocketEvent) = 0;
```

Description

Descendants of TAbstractSocket (see page 14) must override the abstract or, in C++ terminology, pure virtual Event method to provide a response to calls that arise during the normal functioning of the socket. Applications cannot call Event. The Windows socket object calls Event at appropriate times.

The Socket parameter indicates the Windows socket object that calls Event to indicate when specific events occur.

The SocketEvent parameter indicates the event that occurred. It has one of the following values:

Value	Meaning
seLookup	(client sockets only) The Windows socket object is about to locate the server socket. Changing the properties of the client socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will affect the attempted connection.
seConnecting	(client sockets only) The server socket has been located but the connection isn't completed. This is the first opportunity to obtain the actual server port and IP address used for the connection, which may differ from those of the listening socket contacted initially.
seConnect	The socket connection has just been established. This is the point when a socket starts reading or writing over the connection.
seListen	(server sockets only) The Windows socket object has been initialized and is about to open a listening connection. Changing the properties of the server socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will the listening connection.
seAccept	(server sockets only) A client connection request has just been accepted. This is the first opportunity to obtain the port and IP address of the individual client connection.
seWrite	The socket is ready for information to be written.
seRead	The socket is receiving information from a socket on the other end of the connection.

See Also

OnSocketEvent (see page 90)

InitSocket (see page 18)

2.1.2.2.5 TAbstractSocket.InitSocket Method

Tells a Windows socket object to call the Error (see page 16) and Event (see page 17) methods.

Pascal

```
procedure InitSocket(Socket: TCustomWinSocket);
```

C++

```
__fastcall InitSocket(TCustomWinSocket Socket);
```

Description

Applications cannot call this protected method. InitSocket is called automatically by the socket's constructor, immediately after the Windows socket object is created. It allows the socket component to respond to events on the Windows socket object by assigning an OnErrorEvent handler that calls Error (see page 16) and an OnSocketEvent handler that calls Event (see page 17).

The Socket parameter specifies the Windows socket object that passes events on to the socket component.

See Also

Error (see page 16)

TAbstractSocket (see page 14)

OnErrorEvent (see page 89)

OnSocketEvent (see page 90)

2.1.2.2.6 TAbstractSocket.Loaded Method

Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see

page 21) property.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Applications cannot call this protected method. Loaded is called automatically after the socket component has been loaded from a stream. Loaded overrides the default method to implement the value of the Active (see page 21) property that was saved with the socket component.

See Also

Active (see page 21)

2.1.2.2.7 TAbstractSocket.Open Method

Opens the socket connection.

Pascal

```
procedure Open;
```

C++

```
__fastcall Open();
```

Description

Call Open to initiate the socket connection. Open sets the Active (see page 21) property to true.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

Active (see page 21)

2.1.2.2.8 TAbstractSocket.SetActive Method

Sets the Active (see page 21) property.

Pascal

```
procedure SetActive(Value: Boolean);
```

C++

```
__fastcall SetActive(Boolean Value);
```

Description

SetActive is the protected implementation of the Active (see page 21) property. SetActive calls the DoActivate (see page 16) method to open or close the socket connection.

See Also

Active (see page 21)

DoActivate (see page 16)

2.1.2.2.9 TAbstractSocket.SetAddress Method

Sets the Address (see page 22) property.

Pascal

```
procedure SetAddress(Value: string);
```

C++

```
__fastcall SetAddress(UnicodeString Value);
```

Description

SetAddress is the protected implementation of the Address (see page 22) property. SetAddress raises an ESocketError (see page 13) exception if an attempt is made to set the Address (see page 22) property when the socket connection is open.

See Also

Active (see page 21)

Address (see page 22)

ESocketError (see page 13)

2.1.2.2.10 TAbstractSocket.SetHost Method

Sets the Host (see page 22) property.

Pascal

```
procedure SetHost(Value: string);
```

C++

```
__fastcall SetHost(UnicodeString Value);
```

Description

SetHost is the protected implementation of the Host (see page 22) property. SetHost raises an ESocketError (see page 13) exception if an attempt is made to set the Host (see page 22) property when the socket connection is open.

See Also

Active (see page 21)

ESocketError (see page 13)

Host (see page 22)

2.1.2.2.11 TAbstractSocket.SetPort Method

Sets the Port (see page 23) property.

Pascal

```
procedure SetPort(Value: Integer);
```

C++

```
__fastcall SetPort(int Value);
```

Description

SetPort is the protected implementation of the Port (see page 23) property. SetPort raises an ESocketError (see page 13) exception if an attempt is made to set the Port (see page 23) property when the socket connection is open.

See Also

Active (see page 21)

ESocketError (see page 13)

Port (see page 23)

2.1.2.2.12 TAbstractSocket.SetService Method

Sets the Service (see page 23) property.

Pascal

```
procedure SetService(Value: string);
```

C++

```
__fastcall SetService(UnicodeString Value);
```

Description

SetService is the protected implementation of the Service (see page 23) property. SetService raises an ESocketError (see page 13) exception if an attempt is made to set the Service (see page 23) property when the socket connection is open.

See Also

Active (see page 21)

ESocketError (see page 13)

Service (see page 23)

2.1.2.3 TAbstractSocket Properties

The properties of the TAbstractSocket class are listed here.

Protected Properties

	Name	Description
	Active (see page 21)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 22)	Specifies the IP address of the server system.
	Host (see page 22)	Specifies an alias for the IP address of the server system.
	Port (see page 23)	Specifies the ID number that identifies the server socket connection.
	Service (see page 23)	Specifies the name of the service for which the socket connection is used.

2.1.2.3.1 TAbstractSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting

Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to true to make the socket open a connection when the application starts running. At runtime, use the Open (see page 19) or Close (see page 16) method to open or close the connection.

See Also

Close (see page 16)

Open (see page 19)

SetActive (see page 19)

2.1.2.3.2 TAbstractSocket.Address Property

Specifies the IP address of the server system.

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

Address is a string of four numeric (byte) values in the standard Internet dot notation, such as

```
123.197.1.2
```

For client sockets, set Address to the IP address of the server to which the socket should connect. When the connection is opened, the value of Address is bound to the connection.

If the socket specifies the Host (see page 22) property, the address for the connection is taken from the IP address associated with Host (see page 22), rather than the value of Address. Specifying the server system by giving the IP address is faster, because the socket doesn't need to search for the IP address associated with the host name before it can locate the server system.

A single system may support more than one IP address.

Note: Trying to change Address when the connection is open will raise an ESocketError (see page 13) exception.

See Also

ESocketError (see page 13)

Host (see page 22)

SetAddress (see page 20)

2.1.2.3.3 TAbstractSocket.Host Property

Specifies an alias for the IP address of the server system.

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

Host is a string containing the domain name and service of a particular system, such as

```
http://www.wSite.Com
```

For client sockets, set Host to the system with which the client socket should form a connection. When the socket opens a connection, it looks up the IP address for the server socket using the value of Host.

Some servers change the system or IP address that is associated with a particular host name. Using a host name allows the client socket to find the abstract site represented by the host name, even when it has moved to a new IP address.

If Host is set, it takes precedence over the Address (see page 22) property when looking up the address of the server.

Note: Trying to change Host when the connection is open will raise an ESocketError (see page 13) exception.

See Also

Address (see page 22)

ESocketError (see page 13)

SetHost (see page 20)

2.1.2.3.4 TAbstractSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Port numbers allow a single system, identified by the Host (see page 22) or Address (see page 22) property, to host multiple connections simultaneously. Many values of Port are associated by convention with a particular service such as ftp or http.

For server sockets, Port is the ID of the connection on which the server socket listens for client requests. Server sockets generally set Port to a predefined value which clients can use to initiate connections.

For client sockets, Port is the ID of the desired server connection. The value of Port is usually associated with the service the client wishes to use on the server application.

Note: Trying to change Port when the connection is open will raise an ESocketError (see page 13) exception.

See Also

Address (see page 22)

ESocketError (see page 13)

Host (see page 22)

Service (see page 23)

SetPort (see page 20)

2.1.2.3.5 TAbstractSocket.Service Property

Specifies the name of the service for which the socket connection is used.

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

Use Service to identify the use of the connection. Windows provides There are a number of standard service names such as ftp, http, finger, and time. Servers can specify additional services and their associated ports in a SERVICES file. For more information, see the Microsoft documentation for Windows sockets.

Certain port numbers are reserved for specific values of service. Thus, Service provides a more meaningful way to specify the server Port (see page 23) to use for the socket connection. For server sockets, using Service rather than Port (see page 23) ensures that the server will listen for TCP/IP requests on the appropriate port.

Note: Trying to change Service when the connection is open will raise an ESocketError (see page 13) exception.

See Also

ESocketError (see page 13)

Port (see page 23)

SetService (see page 21)

LookupService (see page 75)

2.1.3 TClientSocket Class

TClientSocket manages socket connections for a TCP/IP client.

Class Hierarchy

Pascal

```
TClientSocket = class(TCustomSocket);
```

C++

```
class TClientSocket : public TCustomSocket;
```

File

ScktComp

Description

Add a TClientSocket object to a form or data module to turn an application into a TCP/IP client. TClientSocket specifies a desired connection to a TCP/IP server, manages the connection when it is open, and terminates the connection when the application is through.

Note: See the CHAT demo for an example of how to use TClientSocket.

See Also

TClientWinSocket (see page 35)

TServerSocket (see page 108)

2.1.3.1 TClientSocket Members

The following tables list the members exposed by TClientSocket.

Protected Events

	Name	Description
	OnAccept (see page 59)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnListen (see page 61)	Occurs just before a server socket is opened for listening.

Protected Methods

	Name	Description
V	DoActivate (see page 27)	Provides the prototype of a method to open or close the socket connection.
A	Error (see page 16)	Provides the prototype of a method that responds to error conditions.
V	Error (see page 58)	Generates an OnError (see page 61) event.
A	Event (see page 17)	Provides the prototype of a method that responds when specific socket events occur.
V	Event (see page 58)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetClientType (see page 27)	This is GetClientType, a member of class TClientSocket.
	InitSocket (see page 18)	Tells a Windows socket object to call the Error (see page 16) and Event (see page 17) methods.
V	Loaded (see page 18)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 21) property.
	SetActive (see page 19)	Sets the Active (see page 21) property.
	SetAddress (see page 20)	Sets the Address (see page 22) property.
	SetClientType (see page 27)	This is SetClientType, a member of class TClientSocket.
	SetHost (see page 20)	Sets the Host (see page 22) property.
	SetPort (see page 20)	Sets the Port (see page 23) property.
	SetService (see page 21)	Sets the Service (see page 23) property.

Public Methods

	Name	Description
	Close (see page 16)	Shuts down the socket connection.
V	Create (see page 26)	Creates an instance of TClientSocket (see page 24).
V	Destroy (see page 26)	Destroys an instance of TClientSocket (see page 24).
	Open (see page 19)	Opens the socket connection.

Public Properties

	Name	Description
R	Socket (see page 31)	Describes the client endpoint of the Windows client socket connection.

Published Events

	Name	Description
	OnConnect (see page 32)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 32)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 33)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 33)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 34)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 34)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 35)	Occurs when a client socket should write information to the socket connection.

Published Properties

	Name	Description
	Active (see page 28)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 28)	Specifies the IP address of the server system.
	ClientType (see page 29)	Specifies whether the client socket reads and writes information asynchronously over the socket connection.
	Host (see page 30)	Specifies an alias for the IP address of the server system.
	Port (see page 30)	Specifies the ID number that identifies the server socket connection.
	Service (see page 31)	Specifies the name of the service for which the socket connection is used.

2.1.3.2 TClientSocket.Create Constructor

Creates an instance of TClientSocket (see page 24).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TClientSocket(TComponent * AOwner);
```

Description

Call Create to create and initialize a client socket component at runtime. TClientSocket (see page 24) objects placed at design time are created automatically.

After calling the inherited constructor, Create

Allocates the TClientWinSocket (see page 35) object that describes the client endpoint of the Windows socket connection.

I initializes the event and error dispatchers for the client socket.

See Also

TClientWinSocket (see page 35)

Create

2.1.3.3 TClientSocket.Destroy Destructor

Destroys an instance of TClientSocket (see page 24).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClientSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the client socket is not nil, and only then calls Destroy.

Destroy frees the TClientWinSocket (see page 35) object that describes the client endpoint of the Windows socket connection.

See Also

System::TObject::Free (see page 3372)

TClientWinSocket (see page 35)

2.1.3.4 TClientSocket Methods

The methods of the TClientSocket class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 27)	Provides the prototype of a method to open or close the socket connection.
	GetClientType (see page 27)	This is GetClientType, a member of class TClientSocket.
	SetClientType (see page 27)	This is SetClientType, a member of class TClientSocket.

2.1.3.4.1 TClientSocket.DoActivate Method

Provides the prototype of a method to open or close the socket connection.

Pascal

```
procedure DoActivate(Value: Boolean); override;
```

C++

```
virtual __fastcall DoActivate(Boolean Value);
```

Description

Descendants of TAbstractSocket override DoActivate to provide a method that initiates or terminates the socket connection. The steps that need to be taken to open or close a connection depend on whether the socket is a client socket or a server socket.

See Also

Active (see page 21)

Open (see page 19)

Close (see page 16)

2.1.3.4.2 TClientSocket.GetClientType Method

Pascal

```
function GetClientType: TClientType;
```

C++

```
__fastcall TClientType GetClientType();
```

Description

This is GetClientType, a member of class TClientSocket.

2.1.3.4.3 TClientSocket.SetClientType Method

Pascal

```
procedure SetClientType(Value: TClientType);
```

C++

```
__fastcall SetClientType(TClientType Value);
```

Description

This is SetClientType, a member of class TClientSocket.

2.1.3.5 TClientSocket Properties

The properties of the TClientSocket class are listed here.

Public Properties

	Name	Description
	Socket (see page 31)	Describes the client endpoint of the Windows client socket connection.

Published Properties

	Name	Description
	Active (see page 28)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 28)	Specifies the IP address of the server system.
	ClientType (see page 29)	Specifies whether the client socket reads and writes information asynchronously over the socket connection.
	Host (see page 30)	Specifies an alias for the IP address of the server system.
	Port (see page 30)	Specifies the ID number that identifies the server socket connection.
	Service (see page 31)	Specifies the name of the service for which the socket connection is used.

2.1.3.5.1 TClientSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to true to make the socket open a connection when the application starts running. At runtime, use the Open or Close method to open or close the connection.

See Also

Close (see page 16)

Open (see page 19)

SetActive (see page 19)

2.1.3.5.2 TClientSocket.Address Property

Specifies the IP address of the server system.

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

Address is a string of four numeric (byte) values in the standard Internet dot notation, such as

`123.197.1.2`

For client sockets, set Address to the IP address of the server to which the socket should connect. When the connection is opened, the value of Address is bound to the connection.

If the socket specifies the Host property, the address for the connection is taken from the IP address associated with Host, rather than the value of Address. Specifying the server system by giving the IP address is faster, because the socket doesn't need to search for the IP address associated with the host name before it can locate the server system.

A single system may support more than one IP address.

Note: Trying to change Address when the connection is open will raise an ESocketError exception.

See Also

ESocketError (see page 13)

Host (see page 22)

SetAddress (see page 20)

2.1.3.5.3 TClientSocket.ClientType Property

Specifies whether the client socket reads and writes information asynchronously over the socket connection.

Pascal

```
property ClientType: TClientType;
```

C++

```
__property TClientType ClientType;
```

Description

Set ClientType to ctNonBlocking to enable the client socket to respond to asynchronous reading and writing events. When ClientType is ctNonBlocking, execution is not blocked by reading and writing over the socket connection. OnRead (see page 34) or OnWrite (see page 35) events occur when the socket needs to read or write over the connection.

Set ClientType to ctBlocking to force all reading and writing to occur synchronously. It is a good idea to include the client socket object in a thread if the ClientType is ctBlocking, so that I/O does not block all execution within the client application.

When ClientType is ctBlocking, use a TWinSocketStream (see page 138) object for reading and writing. TWinSocketStream (see page 138) prevents the application from hanging indefinitely if a problem occurs while reading or writing. It also can wait for the socket connection to indicate its readiness for reading.

Use a non-blocking socket when the socket needs to synchronize reading and writing with server sockets.

See Also

OnRead (see page 63)

OnWrite (see page 63)

TThread

TWinSocketStream (see page 138)

2.1.3.5.4 TClientSocket.Host Property

Specifies an alias for the IP address of the server system.

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

Host is a string containing the domain name and service of a particular system, such as

`http://www.wSite.Com`

For client sockets, set Host to the system with which the client socket should form a connection. When the socket opens a connection, it looks up the IP address for the server socket using the value of Host.

Some servers change the system or IP address that is associated with a particular host name. Using a host name allows the client socket to find the abstract site represented by the host name, even when it has moved to a new IP address.

If Host is set, it takes precedence over the Address property when looking up the address of the server.

Note: Trying to change Host when the connection is open will raise an ESocketError exception.

See Also

Address (see page 22)

ESocketError (see page 13)

SetHost (see page 20)

2.1.3.5.5 TClientSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Port numbers allow a single system, identified by the Host or Address property, to host multiple connections simultaneously. Many values of Port are associated by convention with a particular service such as ftp or http.

For server sockets, Port is the ID of the connection on which the server socket listens for client requests. Server sockets generally set Port to a predefined value which clients can use to initiate connections.

For client sockets, Port is the ID of the desired server connection. The value of Port is usually associated with the service the client wishes to use on the server application.

Note: Trying to change Port when the connection is open will raise an ESocketError exception.

See Also

Address (see page 22)

ESocketError (see page 13)

Host (see page 22)

Service (see page 23)

SetPort (see page 20)

2.1.3.5.6 TClientSocket.Service Property

Specifies the name of the service for which the socket connection is used.

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

Use Service to identify the use of the connection. Windows provides There are a number of standard service names such as ftp, http, finger, and time. Servers can specify additional services and their associated ports in a SERVICES file. For more information, see the Microsoft documentation for Windows sockets.

Certain port numbers are reserved for specific values of service. Thus, Service provides a more meaningful way to specify the server Port to use for the socket connection. For server sockets, using Service rather than Port ensures that the server will listen for TCP/IP requests on the appropriate port.

Note: Trying to change Service when the connection is open will raise an ESocketError exception.

See Also

ESocketError (see page 13)

Port (see page 23)

SetService (see page 21)

LookupService (see page 75)

2.1.3.5.7 TClientSocket.Socket Property

Describes the client endpoint of the Windows client socket connection.

Pascal

```
property Socket: TClientWinSocket;
```

C++

```
__property TClientWinSocket Socket;
```

Description

Use Socket to

Determine the address and port of both the client and server sockets that are bound to the connection.

Read (see page 3634) or write information through the socket connection.

Determine what server notifications the client will respond to.

Obtain the Windows socket handle for making Windows socket API calls.

2.1.3.6 TClientSocket Events

The events of the TClientSocket class are listed here.

Published Events

	Name	Description
	OnConnect (see page 32)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 32)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 33)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 33)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 34)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 34)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 35)	Occurs when a client socket should write information to the socket connection.

2.1.3.6.1 TClientSocket.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

See Also

OnAccept (see page 59)

OnConnecting (see page 60)

OnLookup (see page 62)

2.1.3.6.2 TClientSocket.OnConnecting Event

Occurs for a client socket after the server socket has been located, but before the connection is established.

Pascal

```
property OnConnecting: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnecting;
```

Description

Write an OnConnecting event handler for a client socket to take specific action just before the connection to a server socket is established. This is the first opportunity to obtain the actual port and IP address of the server endpoint of the connection that is about to form.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

See Also

OnConnect (see page 60)

OnLookup (see page 62)

2.1.3.6.3 TClientSocket.OnDisconnect Event

Occurs just before a client socket closes the connection to a server socket.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler to take specific action when the connection to a server socket is about to be terminated. OnDisconnect occurs after the value of the Active property is set to false, but before the actual connection is closed.

See Also

Active (see page 21)

Close (see page 16)

OnConnect (see page 60)

2.1.3.6.4 TClientSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the ErrorCode parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError from being raised.

See Also

ESocketError (see page 13)

2.1.3.6.5 TClientSocket.OnLookup Event

Occurs when a client socket is about to look up the server socket with which it wants to connect.

Pascal

```
property OnLookup: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnLookup;
```

Description

Write an OnLookup event handler for a client socket to take specific action just before attempting to locate a server socket. This is the first opportunity to make Windows API calls that affect the client properties of the socket, such as specifying a particular port number. Use the SocketHandle property of the Socket parameter for Windows API calls.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

Note: Changing the Address, Host, Port, or Service properties of socket component in an OnLookup event handler has no effect on the address or port used to locate a server socket. These properties must be correct before the Open method is called.

See Also

Address (see page 22)

Host (see page 22)

OnConnect (see page 60)

OnConnecting (see page 60)

Open (see page 19)

Port (see page 23)

Service (see page 23)

2.1.3.6.6 TClientSocket.OnRead Event

Occurs when a client socket should read information from the socket connection.

Pascal

```
property OnRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnRead;
```

Description

Write an OnRead event handler to read from the socket connection. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual reading.

Note: Non-blocking sockets do not always receive an OnRead event for the last bit of data passed over the connection. When using a non-blocking socket, check for any unread data in the OnDisconnect event to make sure that everything is handled.

See Also

ReceiveBuf (see page 77)

ReceiveText (see page 78)

TWinSocketStream (see page 138)

OnDisconnect (see page 61)

2.1.3.6.7 TClientSocket.OnWrite Event

Occurs when a client socket should write information to the socket connection.

Pascal

```
property OnWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnWrite;
```

Description

Write an OnWrite to write from the socket connection. If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual writing.

See Also

SendBuf (see page 78)

SendStream (see page 79)

SendStreamThenDrop (see page 79)

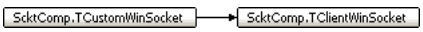
SendText (see page 80)

TWinSocketStream (see page 138)

2.1.4 TClientWinSocket Class

TClientWinSocket describes the client endpoint of a Windows socket connection.

Class Hierarchy



Pascal

```
TClientWinSocket = class(TCustomWinSocket);
```

C++

```
class TClientWinSocket : public TCustomWinSocket;
```

File

ScktComp

Description

Client socket components use TClientWinSocket to manage the Windows socket API calls for a TCP/IP client application. TClientWinSocket differs from its immediate ancestor TCustomWinSocket (see page 64) only in that it introduces a ClientType (see page 38) property to distinguish blocking from non-blocking client sockets.

See Also

TClientSocket (see page 24)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

2.1.4.1 TClientWinSocket Members

The following tables list the members exposed by TClientWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 68)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 70)	Opens a listening connection.
	DoOpen (see page 71)	Opens a connection to a remote socket.
	Error (see page 71)	Generates an OnErrorEvent (see page 89) event.
	Event (see page 72)	Triggers an OnSocketEvent (see page 90) event.
	InitSocket (see page 72)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 81)	Sets the ASyncStyles (see page 83) property.
	SetClientType (see page 38)	This is SetClientType, a member of class TClientWinSocket.

Public Events

	Name	Description
	OnErrorEvent (see page 89)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 90)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
	Accept (see page 68)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 69)	Shuts down the socket connection if it is open.
	Connect (see page 38)	Responds to connection notifications.
	Create (see page 66)	Creates an instance of TCustomWinSocket (see page 64) from a Windows socket handle.
	DefaultHandler (see page 70)	Processes message records.
	Destroy (see page 67)	Destroys an instance of TCustomWinSocket (see page 64).
	Disconnect (see page 70)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 73)	Opens a listening connection for a server socket.
	Lock (see page 74)	Starts a critical section, blocking all other execution threads until the Unlock (see page 81) method is called.
	LookupName (see page 74)	Generates the portion of a Windows Internet socket address structure that describes the host.

	LookupService (see page 75)	Returns the port number associated with a specific service.
	Open (see page 75)	Opens a connection to a remote socket.
V	Read (see page 76)	Generates an OnSocketEvent (see page 90) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 77)	Reads up to Count bytes from the socket connection into the Buf parameter.
	ReceiveLength (see page 77)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 78)	Reads a string from the socket connection.
	SendBuf (see page 78)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection.
	SendStreamThenDrop (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 80)	Writes the string S to the socket connection.
	Unlock (see page 81)	Reenables execution in other threads after a critical section begun by the Lock (see page 74) method.
V	Write (see page 81)	Generates an OnSocketEvent (see page 90) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	Addr (see page 82)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 83)	Determines which asynchronous events the socket can receive.
	ClientType (see page 38)	Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.
R	Connected (see page 83)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 84)	Points to application-specific data associated with the socket.
R	Handle (see page 84)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 85)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 85)	Specifies the name of the local system.
R	LocalPort (see page 86)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 86)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 87)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 87)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 88)	Specifies the name of the remote system.
R	RemotePort (see page 88)	Specifies the ID number of the port used by the RemoteHost (see page 88) to form the socket connection.
R	SocketHandle (see page 89)	Specifies the Windows handle for the socket.

2.1.4.2 TClientWinSocket Methods

The methods of the TClientWinSocket class are listed here.

Protected Methods

	Name	Description
	SetClientType (see page 38)	This is SetClientType, a member of class TClientWinSocket.

Public Methods

	Name	Description
V	Connect (see page 38)	Responds to connection notifications.

2.1.4.2.1 TClientWinSocket.Connect Method

Responds to connection notifications.

Pascal

```
procedure Connect(Socket: TSocket); override;
```

C++

```
virtual __fastcall Connect(TSocket Socket);
```

Description

Applications can not call this protected method. Connect is called in response to notifications that the client socket has successfully completed a connection to a server socket. The Socket parameter is the Windows socket handle for the socket that receives the notification.

Connect generates an OnSocketEvent (see page 90) event with the SocketEvent parameter set to seConnect. Socket components assign an OnSocketEvent (see page 90) event handler that responds to this event by generating an OnConnect event on the socket component.

Note: Connect is only called if the ASyncStyles (see page 83) property includes asConnect.

See Also

ASyncStyles (see page 83)

OnSocketEvent (see page 90)

OnConnect (see page 60)

2.1.4.2.2 TClientWinSocket.SetClientType Method

Pascal

```
procedure SetClientType(Value: TClientType);
```

C++

```
__fastcall SetClientType(TClientType Value);
```

Description

This is SetClientType, a member of class TClientWinSocket.

2.1.4.3 TClientWinSocket Properties

The properties of the TClientWinSocket class are listed here.

Public Properties

	Name	Description
	ClientType (see page 38)	Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.

2.1.4.3.1 TClientWinSocket.ClientType Property

Specifies whether the client Windows socket reads and writes information asynchronously over the socket connection.

Pascal

```
property ClientType: TClientType;
```

C++

```
__property TClientType ClientType;
```

Description

Client socket components set ClientType to reflect their own ClientType property. When ClientType is ctNonBlocking, the client Windows socket responds to asynchronous reading and writing events, and execution is not blocked by reading and writing over the connection. OnSocketEvent (see page 90) events occur when the socket needs to read or write over the connection.

When ClientType is ctBlocking, all reading and writing occur synchronously. It is a good idea to perform all reading and writing in a separate thread if ClientType is ctBlocking. Reading and writing in a separate thread ensures that these operations do not block all execution within the client application.

When ClientType is ctBlocking, use a TWinSocketStream (see page 138) object for reading and writing. TWinSocketStream (see page 138) times out so that the application does not hang when a problem occurs while reading or writing. TWinSocketStream (see page 138) also provides a method to wait, before reading, until the socket connection is ready to send information.

Use a non-blocking socket when the socket needs to synchronize reading and writing with server sockets.

See Also

OnSocketEvent (see page 90)

TThread

TWinSocketStream (see page 138)

2.1.5 TCustomServerSocket Class

TCustomServerSocket is the base class for all server socket components.

Class Hierarchy

Pascal

```
TCustomServerSocket = class(TCustomSocket);
```

C++

```
class TCustomServerSocket : public TCustomSocket;
```

File

ScktComp

Description

TCustomServerSocket introduces new properties and events specific to the functions of a TCP/IP server.

Do not create instances of TCustomServerSocket. Use TCustomServerSocket as a base class when declaring server socket objects that listen for connection requests from other machines.

See Also

TServerWinSocket (see page 120)

TServerSocket (see page 108)

2.1.5.1 TCustomServerSocket Members

The following tables list the members exposed by TCustomServerSocket.

Protected Events

	Name	Description
	OnAccept (see page 59)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 51)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 52)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 52)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 53)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 53)	Occurs when the server socket should write information to a client socket.
	OnConnect (see page 60)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 60)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 61)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 61)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetSocket (see page 54)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 104) object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 54)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 61)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 62)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 63)	Occurs when a client socket should read information from the socket connection.
	OnThreadEnd (see page 55)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 55)	Occurs when the execution thread for a connection to a client socket starts up.
	OnWrite (see page 63)	Occurs when a client socket should write information to the socket connection.

Protected Fields

	Name	Description
	FServerSocket (see page 42)	This is FServerSocket, a member of class TCustomServerSocket.

Protected Methods

	Name	Description
V	DoActivate (see page 42)	Opens or Closes the socket connection.
A	Error (see page 16)	Provides the prototype of a method that responds to error conditions.
V	Error (see page 58)	Generates an OnError (see page 61) event.
A	Event (see page 17)	Provides the prototype of a method that responds when specific socket events occur.
V	Event (see page 58)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetGetSocketEvent (see page 43)	Returns the OnGetSocket (see page 54) event handler.
	GetGetThreadEvent (see page 43)	Returns the OnGetThread (see page 54) event handler.
	GetOnClientError (see page 44)	Returns the OnClientError (see page 52) event handler.
	GetOnClientEvent (see page 44)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 44)	Returns the OnThreadEnd (see page 55) event handler.
	GetOnThreadStart (see page 45)	Returns the OnThreadStart (see page 55) event handler.
	GetServerType (see page 45)	Returns the ServerType (see page 49) property.
	GetThreadCacheSize (see page 45)	Returns the value of the ThreadCacheSize (see page 50) property.

	InitSocket (see page 18)	Tells a Windows socket object to call the Error (see page 16) and Event (see page 17) methods.
V	Loaded (see page 18)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 21) property.
	SetActive (see page 19)	Sets the Active (see page 21) property.
	SetAddress (see page 20)	Sets the Address (see page 22) property.
	SetGetSocketEvent (see page 46)	Sets the OnGetSocket (see page 54) event handler.
	SetGetThreadEvent (see page 46)	Sets the OnGetThread (see page 54) event handler.
	SetHost (see page 20)	Sets the Host (see page 22) property.
	SetOnClientError (see page 47)	Sets the OnClientError (see page 52) event handler.
	SetOnClientEvent (see page 47)	Sets the client event handlers of the associated TServerWinSocket (see page 120) object.
	SetOnThreadEnd (see page 47)	Sets the OnThreadEnd (see page 55) event handler.
	SetOnThreadStart (see page 48)	Sets the OnThreadStart (see page 55) event handler.
	SetPort (see page 20)	Sets the Port (see page 23) property.
	SetServerType (see page 48)	Sets the ServerType (see page 49) property.
	SetService (see page 21)	Sets the Service (see page 23) property.
	SetThreadCacheSize (see page 49)	Sets the ThreadCacheSize (see page 50) property.

Protected Properties

	Name	Description
	Active (see page 21)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 22)	Specifies the IP address of the server system.
	Host (see page 22)	Specifies an alias for the IP address of the server system.
	Port (see page 23)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 49)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 23)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 50)	Specifies the maximum number of threads that can be reused for new client connections.

Public Methods

	Name	Description
	Close (see page 16)	Shuts down the socket connection.
V	Destroy (see page 41)	Destroys an instance of TCustomServerSocket (see page 39).
	Open (see page 19)	Opens the socket connection.

2.1.5.2 TCustomServerSocket.Destroy Destructor

Destroys an instance of TCustomServerSocket (see page 39).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomServerSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the server socket is not nil, and only then calls Destroy.

Destroy frees the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

System::TObject::Free (see page 3372)

TServerWinSocket (see page 120)

2.1.5.3 TCustomServerSocket Fields

The fields of the TCustomServerSocket class are listed here.

Protected Fields

	Name	Description
	FServerSocket (see page 42)	This is FServerSocket, a member of class TCustomServerSocket.

2.1.5.3.1 TCustomServerSocket.FServerSocket Field

Pascal

```
FServerSocket: TServerWinSocket;
```

C++

```
TServerWinSocket FServerSocket;
```

Description

This is FServerSocket, a member of class TCustomServerSocket.

2.1.5.4 TCustomServerSocket Methods

The methods of the TCustomServerSocket class are listed here.

Protected Methods

	Name	Description
	DoActivate (see page 42)	Opens or Closes the socket connection.
	GetGetSocketEvent (see page 43)	Returns the OnGetSocket (see page 54) event handler.
	GetGetThreadEvent (see page 43)	Returns the OnGetThread (see page 54) event handler.
	GetOnClientError (see page 44)	Returns the OnClientError (see page 52) event handler.
	GetOnClientEvent (see page 44)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 44)	Returns the OnThreadEnd (see page 55) event handler.
	GetOnThreadStart (see page 45)	Returns the OnThreadStart (see page 55) event handler.
	GetServerType (see page 45)	Returns the ServerType (see page 49) property.
	GetThreadCacheSize (see page 45)	Returns the value of the ThreadCacheSize (see page 50) property.
	SetGetSocketEvent (see page 46)	Sets the OnGetSocket (see page 54) event handler.
	SetGetThreadEvent (see page 46)	Sets the OnGetThread (see page 54) event handler.
	SetOnClientError (see page 47)	Sets the OnClientError (see page 52) event handler.
	SetOnClientEvent (see page 47)	Sets the client event handlers of the associated TServerWinSocket (see page 120) object.
	SetOnThreadEnd (see page 47)	Sets the OnThreadEnd (see page 55) event handler.
	SetOnThreadStart (see page 48)	Sets the OnThreadStart (see page 55) event handler.
	SetServerType (see page 48)	Sets the ServerType (see page 49) property.
	SetThreadCacheSize (see page 49)	Sets the ThreadCacheSize (see page 50) property.

2.1.5.4.1 TCustomServerSocket.DoActivate Method

Opens or Closes the socket connection.

Pascal

```
procedure DoActivate(Value: Boolean); override;
```

C++

```
virtual __fastcall DoActivate(Boolean Value);
```

Description

DoActivate provides the underlying implementation of the Open (see page 19) and Close (see page 16) methods. When Value is true, DoActivate opens a listening connection to accept connection requests. When Value is false, DoActivate terminates the current listening connection and any connections to client sockets.

See Also

Active (see page 21)

Open (see page 19)

Close (see page 16)

2.1.5.4.2 TCustomServerSocket.GetGetSocketEvent Method

Returns the OnGetSocket (see page 54) event handler.

Pascal

```
function GetGetSocketEvent: TGetSocketEvent;
```

C++

```
__fastcall TGetSocketEvent GetGetSocketEvent();
```

Description

Applications cannot call this protected method. GetGetSocketEvent returns the OnGetSocket (see page 54) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

OnGetSocket (see page 54)

OnGetSocket (see page 137)

2.1.5.4.3 TCustomServerSocket.GetGetThreadEvent Method

Returns the OnGetThread (see page 54) event handler.

Pascal

```
function GetGetThreadEvent: TGetThreadEvent;
```

C++

```
__fastcall TGetThreadEvent GetGetThreadEvent();
```

Description

Applications cannot call this protected method. GetGetThreadEvent returns the OnGetThread (see page 54) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

OnGetThread (see page 54)

OnGetThread (see page 137)

2.1.5.4.4 TCustomServerSocket.GetOnClientError Method

Returns the OnClientError (see page 52) event handler.

Pascal

```
function GetOnClientError: TSocketErrorEvent;
```

C++

```
__fastcall TSocketErrorEvent GetOnClientError();
```

Description

Applications cannot call this protected method. GetOnClientError returns the OnClientError (see page 52) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

OnClientError (see page 52)

OnClientError (see page 135)

2.1.5.4.5 TCustomServerSocket.GetOnClientEvent Method

Provides indexed access to the various event handlers that respond to client events.

Pascal

```
function GetOnClientEvent(Index: Integer): TSocketNotifyEvent;
```

C++

```
__fastcall TSocketNotifyEvent GetOnClientEvent(int Index);
```

Description

Applications can't call this protected method. It is used internally to implement the event handlers that respond to client events.

See Also

OnClientRead (see page 53)

OnClientWrite (see page 53)

OnClientConnect (see page 51)

OnClientDisconnect (see page 52)

OnClientRead (see page 135)

OnClientWrite (see page 136)

OnClientConnect (see page 133)

OnClientDisconnect (see page 134)

2.1.5.4.6 TCustomServerSocket.GetOnThreadEnd Method

Returns the OnThreadEnd (see page 55) event handler.

Pascal

```
function GetOnThreadEnd: TThreadNotifyEvent;
```

C++

```
__fastcall TThreadNotifyEvent GetOnThreadEnd();
```

Description

Applications cannot call this protected method. GetOnThreadEnd returns the OnThreadEnd (see page 55) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

OnThreadEnd (see page 55)

OnThreadEnd (see page 138)

2.1.5.4.7 TCustomServerSocket.GetOnThreadStart Method

Returns the OnThreadStart (see page 55) event handler.

Pascal

```
function GetOnThreadStart: TThreadNotifyEvent;
```

C++

```
__fastcall TThreadNotifyEvent GetOnThreadStart();
```

Description

Applications cannot call this protected method. GetOnThreadStart returns the OnThreadStart (see page 55) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

OnThreadStart (see page 55)

OnThreadStart (see page 138)

2.1.5.4.8 TCustomServerSocket.GetServerType Method

Returns the ServerType (see page 49) property.

Pascal

```
function GetServerType: TServerType;
```

C++

```
__fastcall TServerType GetServerType();
```

Description

Applications cannot call this protected method. GetServerType returns the value of the ServerType (see page 49) property of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

ServerType (see page 49)

ServerType (see page 131)

2.1.5.4.9 TCustomServerSocket.GetThreadCacheSize Method

Returns the value of the ThreadCacheSize (see page 50) property.

Pascal

```
function GetThreadCacheSize: Integer;
```

C++

```
__fastcall int GetThreadCacheSize();
```

Description

Applications cannot call this protected method. GetThreadCacheSize returns the value of the ThreadCacheSize (see page 50) property of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

ThreadCacheSize (see page 50)

ThreadCacheSize (see page 132)

2.1.5.4.10 TCustomServerSocket.SetGetSocketEvent Method

Sets the OnGetSocket (see page 54) event handler.

Pascal

```
procedure SetGetSocketEvent(Value: TGetSocketEvent);
```

C++

```
__fastcall SetGetSocketEvent(TGetSocketEvent Value);
```

Description

Applications cannot call this protected method. SetGetSocketEvent sets the OnGetSocket (see page 54) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 120) to implement their OnGetSocket (see page 54) event.

See Also

OnGetSocket (see page 54)

OnGetSocket (see page 137)

2.1.5.4.11 TCustomServerSocket.SetGetThreadEvent Method

Sets the OnGetThread (see page 54) event handler.

Pascal

```
procedure SetGetThreadEvent(Value: TGetThreadEvent);
```

C++

```
__fastcall SetGetThreadEvent(TGetThreadEvent Value);
```

Description

Applications cannot call this protected method. SetGetThreadEvent sets the OnGetThread (see page 54) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 120) to implement their OnGetThread (see page 54) event.

See Also

OnGetThread (see page 54)

OnGetThread (see page 137)

2.1.5.4.12 TCustomServerSocket.SetOnClientError Method

Sets the OnClientError (see page 52) event handler.

Pascal

```
procedure SetOnClientError(Value: TSocketErrorEvent);
```

C++

```
__fastcall SetOnClientError(TSocketErrorEvent Value);
```

Description

Applications cannot call this protected method. SetOnClientError sets the OnClientError (see page 52) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 120) to implement their OnClientError (see page 52) event.

See Also

OnClientError (see page 52)

OnClientError (see page 135)

2.1.5.4.13 TCustomServerSocket.SetOnClientEvent Method

Sets the client event handlers of the associated TServerWinSocket (see page 120) object.

Pascal

```
procedure SetOnClientEvent(Index: Integer; Value: TSocketNotifyEvent);
```

C++

```
__fastcall SetOnClientEvent(int Index, TSocketNotifyEvent Value);
```

Description

Applications cannot call this protected method. SetOnClientEvent sets the various event handlers that respond to events from the server endpoint of a client connection. Server socket components use TServerWinSocket (see page 120) to implement these events.

See Also

OnClientRead (see page 53)

OnClientWrite (see page 53)

OnClientConnect (see page 51)

OnClientDisconnect (see page 52)

OnClientRead (see page 135)

OnClientWrite (see page 136)

OnClientConnect (see page 133)

OnClientDisconnect (see page 134)

2.1.5.4.14 TCustomServerSocket.SetOnThreadEnd Method

Sets the OnThreadEnd (see page 55) event handler.

Pascal

```
procedure SetOnThreadEnd(Value: TThreadNotifyEvent);
```

C++

```
__fastcall SetOnThreadEnd(TThreadNotifyEvent Value);
```

Description

Applications cannot call this protected method. SetOnThreadEnd sets the OnThreadEnd (see page 55) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 120) to implement their OnThreadEnd (see page 55) event.

See Also

OnThreadEnd (see page 55)

OnThreadEnd (see page 138)

2.1.5.4.15 TCustomServerSocket.SetOnThreadStart Method

Sets the OnThreadStart (see page 55) event handler.

Pascal

```
procedure SetOnThreadStart(Value: TThreadNotifyEvent);
```

C++

```
__fastcall SetOnThreadStart(TThreadNotifyEvent Value);
```

Description

Applications cannot call this protected method. SetOnThreadStart sets the OnThreadEnd (see page 55) event handler of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 120) to implement their OnThreadEnd (see page 55) event.

See Also

OnThreadStart (see page 55)

OnThreadEnd (see page 138)

2.1.5.4.16 TCustomServerSocket.SetServerType Method

Sets the ServerType (see page 49) property.

Pascal

```
procedure SetServerType(Value: TServerType);
```

C++

```
__fastcall SetServerType(TServerType Value);
```

Description

Applications cannot call this protected method. SetServerType sets the ServerType (see page 49) property of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 120) to implement their ServerType (see page 49) property.

See Also

ServerType (see page 49)

ServerType (see page 131)

2.1.5.4.17 TCustomServerSocket.SetThreadCacheSize Method

Sets the ThreadCacheSize (see page 50) property.

Pascal

```
procedure SetThreadCacheSize(Value: Integer);
```

C++

```
__fastcall SetThreadCacheSize(int Value);
```

Description

Applications cannot call this protected method. SetThreadCacheSize sets the ThreadCacheSize (see page 50) property of the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. Server socket objects use TServerWinSocket (see page 120) to implement their ThreadCacheSize (see page 50) property.

See Also

ThreadCacheSize (see page 50)

ThreadCacheSize (see page 132)

2.1.5.5 TCustomServerSocket Properties

The properties of the TCustomServerSocket class are listed here.

Protected Properties

	Name	Description
	ServerType (see page 49)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	ThreadCacheSize (see page 50)	Specifies the maximum number of threads that can be reused for new client connections.

2.1.5.5.1 TCustomServerSocket.ServerType Property

Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the server socket. When ServerType is stThreadBlocking, the execution of the connection thread is suspended while reading or writing until all information has been transferred over the connection. The thread for each connection generates OnClientRead (see page 53) or OnClientWrite (see page 53) events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over the socket connections asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default. OnClientRead (see page 53) or OnClientWrite (see page 53) events occur when the client socket on the other end of one of the connections tries to send or receive information over the connection.

Use a non-blocking socket when the socket needs to synchronize reading and writing with client sockets.

Note: When ServerType is stThreadBlocking, it is important that the OnClient... event handlers contain thread-safe code.

See Also

OnClientRead (see page 53)

OnClientWrite (see page 53)

TServerClientThread (see page 95)

2.1.5.5.2 TCustomServerSocket.ThreadCacheSize Property

Specifies the maximum number of threads that can be reused for new client connections.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType (see page 49) is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New (see page 3619) connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

See Also

ServerType (see page 49)

TServerClientThread (see page 95)

ActiveThreads (see page 130)

IdleThreads (see page 131)

2.1.5.6 TCustomServerSocket Events

The events of the TCustomServerSocket class are listed here.

Protected Events

	Name	Description
	OnClientConnect (see page 51)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 52)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 52)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 53)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 53)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 54)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 104) object to represent the server endpoint of a connection to a client socket.

	OnGetThread (see page 54)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnThreadEnd (see page 55)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 55)	Occurs when the execution thread for a connection to a client socket starts up.

2.1.5.6.1 TCustomServerSocket.OnClientConnect Event

Occurs when a client socket completes a connection accepted by the server socket.

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write (see page 3707) an OnClientConnect event handler to take specific action when a client socket completes the socket connection to the server socket. For example, the socket may start reading or writing over the connection in an OnClientConnect event handler.

The order of server socket events leading up to OnClientConnect is as follows:

1An OnListen (see page 61) event occurs just before the server socket is opened for listening.

2The server socket receives client requests in a listening queue. The server socket accepts one of those requests, and receives a Windows socket handle for the new socket connection.

3The server socket generates an OnGetSocket (see page 54) event, passing in the Windows socket handle. If a TServerClientWinSocket (see page 104) object for the server endpoint of the new connection is not created in an OnGetSocket (see page 54) event handler, the server socket creates one. The TServerWinSocket (see page 120) object continues to listen for other clients.

4An OnAccept (see page 59) event occurs, using the new TServerClientWinSocket (see page 104) object.

5If ServerType (see page 49) is stThreadBlocking and no thread is available in the cache, an OnGetThread (see page 54) event occurs. If the OnGetThread (see page 54) event handler does not create a thread, the server socket creates a TServerClientThread (see page 95).

6If ServerType (see page 49) is stThreadBlocking, an OnThreadStart (see page 55) event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket (see page 104) object and an OnClientConnect event occurs.

Note: If ServerType (see page 49) is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientConnect event handler for TServerSocket (see page 108) is also set when setting the OnClientConnect event handler of the associated TServerWinSocket (see page 120).

See Also

OnAccept (see page 59)

OnGetSocket (see page 54)

OnGetThread (see page 54)

OnListen (see page 61)

OnThreadStart (see page 55)

ServerType (see page 49)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

GetClientThread (see page 127)

2.1.5.6.2 TCustomServerSocket.OnClientDisconnect Event

Occurs when one of the connections to a client socket is closed.

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write (see page 3707) an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client connection does not close the server socket. The server socket remains open and listening for client requests on its listening connection.

The TServerClientWinSocket (see page 104) that describes the server endpoint of the client connection is freed after OnClientDisconnect. If ServerType (see page 49) is stThreadBlocking, OnThreadEnd (see page 55) occurs after OnClientDisconnect.

Note: If ServerType (see page 49) is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientDisconnect event handler for TServerSocket (see page 108) is also set when setting the OnClientDisconnect event handler of the associated TServerWinSocket (see page 120).

See Also

OnThreadEnd (see page 55)

ServerType (see page 49)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

GetClientThread (see page 127)

2.1.5.6.3 TCustomServerSocket.OnClientError Event

Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write (see page 3707) an OnClientError event handler to respond to errors that arise with a connection to an individual client. Set the ErrorCode parameter to 0 if the OnClientError event handler successfully deals with the error condition to prevent an ESocketError (see page 13) from being raised.

Note: If ServerType (see page 49) is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientError event handler for TServerSocket (see page 108) is also set when setting the OnClientError event handler of the associated TServerWinSocket (see page 120).

See Also

ESocketError (see page 13)

ServerType (see page 49)

TServerClientWinSocket (see page 104)

GetClientThread (see page 127)

2.1.5.6.4 TCustomServerSocket.OnClientRead Event

Occurs when the server socket should read information from a client socket.

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write (see page 3707) an OnClientRead event handler to read from the socket connection. If the ServerType (see page 49) property is stThreadBlocking, use a TWinSocketStream (see page 138) object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, use the methods of the Socket parameter to perform the actual reading.

Note: If ServerType (see page 49) is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 49)

ReceiveBuf (see page 77)

ReceiveText (see page 78)

TServerClientWinSocket (see page 104)

GetClientThread (see page 127)

TWinSocketStream (see page 138)

2.1.5.6.5 TCustomServerSocket.OnClientWrite Event

Occurs when the server socket should write information to a client socket.

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write (see page 3707) an OnClientWrite event handler to write to the socket connection. If the ServerType (see page 49) property is stThreadBlocking, use a TWinSocketStream (see page 138) object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, use the methods of the Socket parameter to perform the actual writing.

Note: If ServerType (see page 49) is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 49)

SendBuf (see page 78)

SendStream (see page 79)

SendText (see page 80)

GetClientThread (see page 127)

TWinSocketStream (see page 138)

2.1.5.6.6 TCustomServerSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket (see page 104) object to represent the server endpoint of a connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write (see page 3707) an OnGetSocket event handler to create a specialized descendant of TServerClientWinSocket (see page 104) for the server socket component to use. Return the new TServerClientWinSocket (see page 104) object in the ClientSocket parameter. The Sender parameter is the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection.

See Also

TServerClientWinSocket (see page 104)

Create (see page 106)

TServerWinSocket (see page 120)

2.1.5.6.7 TCustomServerSocket.OnGetThread Event

Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write (see page 3707) an OnGetThread event handler to create a specialized descendant of TServerClientThread (see page 95) for the connection to the client socket. Return the new TServerClientThread (see page 95) object in the SocketThread parameter.

Most applications that use thread-blocking server components will want to provide an OnGetThread event handler and implement a TServerClientThread (see page 95) that handles its own reading and writing in a thread-safe manner rather than relying on the default TServerClientThread (see page 95) which triggers OnClientRead (see page 53) and OnClientWrite (see page 53) events. This is because the OnClientRead (see page 53) and OnClientWrite (see page 53) event handlers reside on the server socket, which is in global memory.

The Sender parameter is the TServerWinSocket (see page 120) object that describes the server endpoint of the listening connection. The ClientSocket parameter is the TServerClientWinSocket (see page 104) object that describes the server endpoint of the connection to the client socket that is about to be formed.

See Also

ServerType (see page 49)

TServerClientThread (see page 95)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

2.1.5.6.8 TCustomServerSocket.OnThreadEnd Event

Occurs when a client socket connection is terminated and the associated thread finishes execution.

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write (see page 3707) an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect (see page 52) event.

The Sender parameter is the TServerWinSocket (see page 120) that describes the server endpoint of the listening connection. The Thread parameter is the thread that is finishing up.

See Also

OnClientDisconnect (see page 52)

TServerWinSocket (see page 120)

2.1.5.6.9 TCustomServerSocket.OnThreadStart Event

Occurs when the execution thread for a connection to a client socket starts up.

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write (see page 3707) an OnThreadStart to initialize the TServerClientThread (see page 95) for the connection to a single client socket. The Sender parameter is the TServerWinSocket (see page 120) that describes the server endpoint of the listening connection. The Thread parameter is the thread that is about to start execution.

See Also

TServerClientThread (see page 95)

TServerWinSocket (see page 120)

2.1.6 TCustomSocket Class

TCustomSocket is the base class for socket components.

Class Hierarchy

Pascal

```
TCustomSocket = class(TAbstractSocket);
```

C++

```
class TCustomSocket : public TAbstractSocket;
```

File

ScktComp

Description

Override TCustomSocket to create a socket component that can be assigned event handlers for the events and error conditions that occur. In addition to the properties and methods introduced by TAbstractSocket (see page 14) that enable an application to work with Windows sockets, TCustomSocket introduces several events that can be used by applications.

Do not create instances of TCustomSocket. Use a descendant of TCustomSocket to add the ability to communicate with other machines using TCP/IP. To create a socket that initiates the connection with another machine, use TClientSocket (see page 24). To create a socket that responds to requests for connections from other machines, use TServerSocket (see page 108).

See Also

TClientSocket (see page 24)

TCustomWinSocket (see page 64)

TServerSocket (see page 108)

2.1.6.1 TCustomSocket Members

The following tables list the members exposed by TCustomSocket.

Protected Events

	Name	Description
	OnAccept (see page 59)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnConnect (see page 60)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 60)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 61)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 61)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnListen (see page 61)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 62)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 63)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 63)	Occurs when a client socket should write information to the socket connection.

Protected Methods

	Name	Description
	DoActivate (see page 16)	Provides the prototype of a method to open or close the socket connection.
	Error (see page 58)	Generates an OnError (see page 61) event.
	Event (see page 58)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	InitSocket (see page 18)	Tells a Windows socket object to call the Error (see page 16) and Event (see page 17) methods.
	Loaded (see page 18)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 21) property.
	SetActive (see page 19)	Sets the Active (see page 21) property.
	SetAddress (see page 20)	Sets the Address (see page 22) property.
	SetHost (see page 20)	Sets the Host (see page 22) property.
	SetPort (see page 20)	Sets the Port (see page 23) property.
	SetService (see page 21)	Sets the Service (see page 23) property.

Protected Properties

	Name	Description
	Active (see page 21)	Indicates whether the socket connection is open and available for communication with other machines.
	Address (see page 22)	Specifies the IP address of the server system.
	Host (see page 22)	Specifies an alias for the IP address of the server system.
	Port (see page 23)	Specifies the ID number that identifies the server socket connection.
	Service (see page 23)	Specifies the name of the service for which the socket connection is used.

Public Methods

	Name	Description
	Close (see page 16)	Shuts down the socket connection.
	Open (see page 19)	Opens the socket connection.

2.1.6.2 TCustomSocket Methods

The methods of the TCustomSocket class are listed here.

Protected Methods

	Name	Description
	Error (see page 58)	Generates an OnError (see page 61) event.
	Event (see page 58)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.

2.1.6.2.1 TCustomSocket.Error Method

Generates an OnError (see page 61) event.

Pascal

```
procedure Error(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer);
override;
```

C++

```
virtual __fastcall Error(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode);
```

Description

Applications cannot call Error. It responds to error conditions by calling the OnError (see page 61) event handler if it is assigned. Override Error to add class-specific error processing or to block the OnError (see page 61) event.

The Socket parameter indicates the Windows socket object which encounters the error condition.

The ErrorEvent parameter indicates what the socket was trying to do when the error occurred. It has one of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	For client sockets, this indicates that the client socket can't locate the server, or that a problem on the server prevents the opening of a connection. For server sockets, this indicates that a client connection request that has already been accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.
eeAccept	For server sockets only, this indicates that a problem occurred when trying to accept a client connection request.

The ErrorCode parameter is the error code received by the Windows socket object. Setting this value to 0 in the Error method prevents the socket from raising an exception. For information on possible error codes, see the Microsoft documentation on Windows sockets.

See Also

OnError (see page 61)

2.1.6.2.2 TCustomSocket.Event Method

Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.

Pascal

```
procedure Event(Socket: TCustomWinSocket; SocketEvent: TSocketEvent); override;
```

C++

```
virtual __fastcall Event(TCustomWinSocket Socket, TSocketEvent SocketEvent);
```

Description

Applications cannot call Event. It is called automatically when various events occur. Override Event to introduce class-specific responses when certain events occur or to prevent the socket from calling the associated event handler.

The Socket parameter indicates the Windows socket object that calls Event to indicate when specific events occur.

The SocketEvent parameter indicates the event that occurred.

See Also

OnAccept (see page 59)

OnConnect (see page 60)

OnConnecting (see page 60)

OnDisconnect (see page 61)

OnListen (see page 61)

OnLookup (see page 62)

OnRead (see page 63)

OnWrite (see page 63)

2.1.6.3 TCustomSocket Events

The events of the TCustomSocket class are listed here.

Protected Events

	Name	Description
	OnAccept (see page 59)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnConnect (see page 60)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 60)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 61)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 61)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnListen (see page 61)	Occurs just before a server socket is opened for listening.
	OnLookup (see page 62)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 63)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 63)	Occurs when a client socket should write information to the socket connection.

2.1.6.3.1 TCustomSocket.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnAccept;
```

Description

Write (see page 3707) an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that will be used in the server endpoint of the connection. The port number, and possibly the IP address, will differ from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect (see page 60) event occurs for the client socket.

See Also

OnConnect (see page 60)

OnListen (see page 61)

2.1.6.3.2 TCustomSocket.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write (see page 3707) an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

When a client socket opens a connection, the following events occur:

1An OnLookup (see page 62) event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting (see page 60) event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect event occurs after the connection is established.

See Also

OnAccept (see page 59)

OnConnecting (see page 60)

OnLookup (see page 62)

2.1.6.3.3 TCustomSocket.OnConnecting Event

Occurs for a client socket after the server socket has been located, but before the connection is established.

Pascal

```
property OnConnecting: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnecting;
```

Description

Write (see page 3707) an OnConnecting event handler for a client socket to take specific action just before the connection to a server socket is established. This is the first opportunity to obtain the actual port and IP address of the server endpoint of the connection that is about to form.

When a client socket opens a connection, the following events occur:

1An OnLookup (see page 62) event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect (see page 60) event occurs after the connection is established.

See Also

OnConnect (see page 60)

OnLookup (see page 62)

2.1.6.3.4 TCustomSocket.OnDisconnect Event

Occurs just before a client socket closes the connection to a server socket.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write (see page 3707) an OnDisconnect event handler to take specific action when the connection to a server socket is about to be terminated. OnDisconnect occurs after the value of the Active (see page 21) property is set to false, but before the actual connection is closed.

See Also

Active (see page 21)

Close (see page 16)

OnConnect (see page 60)

2.1.6.3.5 TCustomSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write (see page 3707) an OnError event handler to respond to errors that arise with the socket connection. Set the ErrorCode parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError (see page 13) from being raised.

See Also

ESocketError (see page 13)

2.1.6.3.6 TCustomSocket.OnListen Event

Occurs just before a server socket is opened for listening.

Pascal

```
property OnListen: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnListen;
```

Description

Write (see page 3707) an OnListen event handler for a server socket to take specific action just before the socket is opened for listening. OnListen occurs after Address (see page 22) and Port (see page 23) have been bound to the socket connection, but before it is opened. This is the last opportunity to make changes to the socket endpoint before it is opened for listening. Use the SocketHandle property of the Socket parameter for Windows API calls that change the socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. The connection to a client socket is completed when a request from the queue is accepted.

See Also

Accept (see page 68)

Address (see page 22)

Port (see page 23)

2.1.6.3.7 TCustomSocket.OnLookup Event

Occurs when a client socket is about to look up the server socket with which it wants to connect.

Pascal

```
property OnLookup: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnLookup;
```

Description

Write (see page 3707) an OnLookup event handler for a client socket to take specific action just before attempting to locate a server socket. This is the first opportunity to make Windows API calls that affect the client properties of the socket, such as specifying a particular port number. Use the SocketHandle property of the Socket parameter for Windows API calls.

When a client socket opens a connection, the following events occur:

1An OnLookup event occurs prior to an attempt to locate the server socket.

2The Windows socket is set up, and initialized for event notification.

3An OnConnecting (see page 60) event occurs after the server socket is located.

4The connection request is accepted by the server and completed by the client socket.

5An OnConnect (see page 60) event occurs after the connection is established.

Note: Changing the Address (see page 22), Host (see page 22), Port (see page 23), or Service (see page 23) properties of socket component in an OnLookup event handler has no effect on the address or port used to locate a server socket. These properties must be correct before the Open (see page 19) method is called.

See Also

Address (see page 22)

Host (see page 22)

OnConnect (see page 60)

OnConnecting (see page 60)

Open (see page 19)

Port (see page 23)

Service (see page 23)

2.1.6.3.8 TCustomSocket.OnRead Event

Occurs when a client socket should read information from the socket connection.

Pascal

```
property OnRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnRead;
```

Description

Write (see page 3707) an OnRead event handler to read from the socket connection. If the socket is a blocking socket, use a TWinSocketStream (see page 138) object to read from the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual reading.

Note: Non-blocking sockets do not always receive an OnRead event for the last bit of data passed over the connection. When using a non-blocking socket, check for any unread data in the OnDisconnect (see page 61) event to make sure that everything is handled.

See Also

ReceiveBuf (see page 77)

ReceiveText (see page 78)

TWinSocketStream (see page 138)

OnDisconnect (see page 61)

2.1.6.3.9 TCustomSocket.OnWrite Event

Occurs when a client socket should write information to the socket connection.

Pascal

```
property OnWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnWrite;
```

Description

Write (see page 3707) an OnWrite to write from the socket connection. If the socket is a blocking socket, use a TWinSocketStream (see page 138) object to write to the connection. Otherwise, u Use the methods of the Socket parameter to perform the actual writing.

See Also

SendBuf (see page 78)

SendStream (see page 79)

SendStreamThenDrop (see page 79)

SendText (see page 80)

TWinSocketStream (see page 138)

2.1.7 TCustomWinSocket Class

TCustomWinSocket is the base class for all Windows socket objects.

Class Hierarchy

Pascal

```
TCustomWinSocket = class;
```

C++

```
class TCustomWinSocket;
```

File

ScktComp

Description

TCustomWinSocket introduces properties, events, and methods to describe an endpoint in a Windows socket connection. Descendants of TCustomWinSocket are used by socket components to manage the Windows socket API calls and to store information about a socket communication link.

A Windows socket encapsulates a set of communication protocols to allow the application to connect to other machines for reading or writing information. Windows sockets provide connections based on the TCP/IP protocol. They also allow connections that use the Xerox Network System (see page 3159) (XNS), Digital's DECnet protocol, or Novell's IPX/SPX family. Sockets (see page 1367) allow an application to form connections to other machines without being concerned with the details of the actual networking software.

Descendants of TCustomWinSocket represent different endpoints in socket connections. TClientWinSocket (see page 35) represents the client endpoint of a socket communication link to a server socket. TServerWinSocket (see page 120) represents the server endpoint of a listening connection. TServerClientWinSocket (see page 104) represents the server endpoint of a socket communication link to a client socket.

See Also

TAbstractSocket (see page 14)

TClientWinSocket (see page 35)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

2.1.7.1 TCustomWinSocket Members

The following tables list the members exposed by TCustomWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 68)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 70)	Opens a listening connection.
	DoOpen (see page 71)	Opens a connection to a remote socket.
	Error (see page 71)	Generates an OnErrorEvent (see page 89) event.
	Event (see page 72)	Triggers an OnSocketEvent (see page 90) event.
	InitSocket (see page 72)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 81)	Sets the ASyncStyles (see page 83) property.

Public Events

	Name	Description
	OnErrorEvent (see page 89)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 90)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
V	Accept (see page 68)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 69)	Shuts down the socket connection if it is open.
V	Connect (see page 69)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 66)	Creates an instance of TCustomWinSocket (see page 64) from a Windows socket handle.
V	DefaultHandler (see page 70)	Processes message records.
V	Destroy (see page 67)	Destroys an instance of TCustomWinSocket (see page 64).
V	Disconnect (see page 70)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 73)	Opens a listening connection for a server socket.
	Lock (see page 74)	Starts a critical section, blocking all other execution threads until the Unlock (see page 81) method is called.
	LookupName (see page 74)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 75)	Returns the port number associated with a specific service.
	Open (see page 75)	Opens a connection to a remote socket.
V	Read (see page 76)	Generates an OnSocketEvent (see page 90) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 77)	Reads up to Count bytes from the socket connection into the Buf parameter.
	ReceiveLength (see page 77)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 78)	Reads a string from the socket connection.
	SendBuf (see page 78)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection.
	SendStreamThenDrop (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 80)	Writes the string S to the socket connection.
	Unlock (see page 81)	Reenables execution in other threads after a critical section begun by the Lock (see page 74) method.
V	Write (see page 81)	Generates an OnSocketEvent (see page 90) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	Addr (see page 82)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 83)	Determines which asynchronous events the socket can receive.
R	Connected (see page 83)	Indicates whether the socket connection is open and available for communication with other machines.

	Data (see page 84)	Points to application-specific data associated with the socket.
R	Handle (see page 84)	Specifies the handle of the window that receives notifications from the Windows socket.
R	LocalAddress (see page 85)	Specifies the IP address of the local system that was bound to the socket connection.
R	LocalHost (see page 85)	Specifies the name of the local system.
R	LocalPort (see page 86)	Specifies the ID number of the port used by the socket connection.
R	LookupState (see page 86)	Indicates any asynchronous activities occurring while opening a connection.
R	RemoteAddr (see page 87)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
R	RemoteAddress (see page 87)	Specifies the IP address of the remote system that was bound to the socket connection.
R	RemoteHost (see page 88)	Specifies the name of the remote system.
R	RemotePort (see page 88)	Specifies the ID number of the port used by the RemoteHost (see page 88) to form the socket connection.
R	SocketHandle (see page 89)	Specifies the Windows handle for the socket.

2.1.7.2 TCustomWinSocket.Create Constructor

Creates an instance of TCustomWinSocket (see page 64) from a Windows socket handle.

Pascal

```
constructor Create(ASocket: TSocket);
```

C++

```
__fastcall TCustomWinSocket(TSocket ASocket);
```

Description

Call Create to create a Windows socket object. Most Windows socket objects are created automatically by the socket component that uses them. Applications may create their own Windows socket objects in an OnGetSocket event handler for a server socket component.

After calling the inherited constructor, Create

Makes sure that Winsock.dll is initialized.

Allocates helper objects to manage multithreading issues.

Initializes ASyncStyles (see page 83) to [asRead, asWrite, asConnect, asClose]

Initializes the SocketHandle (see page 89) property to the ASocket parameter and sets the Connected (see page 83) property to reflect whether ASocket is the handle of a valid open Windows socket.

Initializes the Addr (see page 82) property.

See Also

Addr (see page 82)

ASyncStyles (see page 83)

Connected (see page 83)

SocketHandle (see page 89)

OnGetSocket (see page 54)

2.1.7.3 TCustomWinSocket.Destroy Destructor

Destroys an instance of TCustomWinSocket (see page 64).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWinSocket();
```

Description

Most applications do not need to explicitly free Windows socket objects. They are freed by the socket components that use them.

See Also

System::TObject::Free (see page 3372)

TCustomSocket (see page 56)

2.1.7.4 TCustomWinSocket Methods

The methods of the TCustomWinSocket class are listed here.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 68)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 70)	Opens a listening connection.
	DoOpen (see page 71)	Opens a connection to a remote socket.
	Error (see page 71)	Generates an OnErrorEvent (see page 89) event.
	Event (see page 72)	Triggers an OnSocketEvent (see page 90) event.
	InitSocket (see page 72)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 81)	Sets the ASyncStyles (see page 83) property.

Public Methods

	Name	Description
	Accept (see page 68)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 69)	Shuts down the socket connection if it is open.
	Connect (see page 69)	Provides the interface for a method that responds to connection notifications from client sockets.
	DefaultHandler (see page 70)	Processes message records.
	Disconnect (see page 70)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 73)	Opens a listening connection for a server socket.
	Lock (see page 74)	Starts a critical section, blocking all other execution threads until the Unlock (see page 81) method is called.
	LookupName (see page 74)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 75)	Returns the port number associated with a specific service.
	Open (see page 75)	Opens a connection to a remote socket.
	Read (see page 76)	Generates an OnSocketEvent (see page 90) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 77)	Reads up to Count bytes from the socket connection into the Buf parameter.
	ReceiveLength (see page 77)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 78)	Reads a string from the socket connection.
	SendBuf (see page 78)	Writes Count bytes to the socket connection from the Buf parameter.

	SendStream (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection.
	SendStreamThenDrop (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 80)	Writes the string S to the socket connection.
	Unlock (see page 81)	Reenables execution in other threads after a critical section begun by the Lock (see page 74) method.
	Write (see page 81)	Generates an OnSocketEvent (see page 90) of type seWrite in response to notifications that the socket connection is ready.

2.1.7.4.1 TCustomWinSocket.Accept Method

Provides the interface for a method that accepts a connection to a client socket.

Pascal

```
procedure Accept(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Accept(TSocket Socket);
```

Description

Accept is called automatically in response to notifications that a client socket is requesting a connection. The Socket parameter is the Windows socket handle for the socket that receives the notification.

The Accept method for TCustomWinSocket (see page 64) does nothing with the notification. Descendants of TCustomWinSocket (see page 64) that represent listening server sockets override this method to accept the connection to the client socket.

Accept is only called if the ASyncStyles (see page 83) property includes asAccept.

See Also

ASyncStyles (see page 83)

2.1.7.4.2 TCustomWinSocket.AsyncInitSocket Method

Looks up information or forms socket connections asynchronously.

Pascal

```
procedure AsyncInitSocket(const Name: string; const Address: string; const Service: string;
Port: Word; QueueSize: Integer; Client: Boolean);
```

C++

```
__fastcall AsyncInitSocket(const UnicodeString Name, const UnicodeString Address, const
UnicodeString Service, Word Port, int QueueSize, Boolean Client);
```

Description

AsyncInitSocket is used internally to open a socket connection in non-blocking sockets. It is called by the Open (see page 75) or Listen (see page 73) method when that method's Block parameter is false.

The Name parameter is the host name for the socket. The Address parameter is the IP address. The Service parameter is the service for which the socket is to used. The Port parameter is the port number for the socket. The QueueSize parameter indicates the number of client requests a listening connection can hold while waiting to form connections. The Client parameter indicates whether the socket address structure is to be used by a client socket to describe the destination of a connection or by a server socket to describe the listening connection.

See Also

ASyncStyles (see page 83)

ServerType (see page 131)

ClientType (see page 38)

Addr (see page 82)

Listen (see page 73)

LookupName (see page 74)

LookupService (see page 75)

Open (see page 75)

RemoteAddr (see page 87)

Address (see page 22)

Host (see page 22)

Port (see page 23)

Service (see page 23)

2.1.7.4.3 TCustomWinSocket.Close Method

Shuts down the socket connection if it is open.

Pascal

```
procedure Close;
```

C++

```
__fastcall Close();
```

Description

Call Close to terminate an open socket connection. Close triggers an OnSocketEvent (see page 90) of type seDisconnect before the connection is shut down. If the Windows socket is a listening server socket, Close shuts down all open connections to client sockets before shutting down the listening connection.

See Also

Connected (see page 83)

OnSocketEvent (see page 90)

2.1.7.4.4 TCustomWinSocket.Connect Method

Provides the interface for a method that responds to connection notifications from client sockets.

Pascal

```
procedure Connect(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Connect(TSocket Socket);
```

Description

Connect is called automatically in response to notifications that the client socket has successfully completed a connection to a

server socket. The Socket parameter is the Windows socket handle for the socket that receives the notification.

The Connect method for TCustomWinSocket (see page 64) does nothing with the notification. Descendants of TCustomWinSocket (see page 64) that represent client sockets override this method to respond to connection notifications.

Connect is only called if the ASyncStyles (see page 83) property includes asConnect.

See Also

ASyncStyles (see page 83)

2.1.7.4.5 TCustomWinSocket.DefaultHandler Method

Processes message records.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

DefaultHandler overrides the inherited method in order to process all messages that are sent to the Windows socket object. DefaultHandler passes the messages to the Windows CallWindowProc function.

See Also

TMessage

2.1.7.4.6 TCustomWinSocket.Disconnect Method

Shuts down the socket connection represented by a Windows socket handle.

Pascal

```
procedure Disconnect(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Disconnect(TSocket Socket);
```

Description

Disconnect is called by the Close (see page 69) method to terminate an individual socket connection.

Disconnect calls the Lock (see page 74) method to make sure the method is thread-safe. Then, it generates an OnSocketEvent (see page 90) of type seDisconnect. After the OnSocketEvent (see page 90), Disconnect closes the connection. Finally, it calls Unlock (see page 81) to end the critical section started by Lock (see page 74).

See Also

Close (see page 69)

Lock (see page 74)

OnSocketEvent (see page 90)

Unlock (see page 81)

2.1.7.4.7 TCustomWinSocket.DoListen Method

Opens a listening connection.

Pascal

```
procedure DoListen(QueueSize: Integer);
```

C++

```
__fastcall DoListen(int QueueSize);
```

Description

The socket uses DoListen internally to open a listening connection with a queue that can hold QueueSize client connection requests. DoListen handles those tasks that are common to opening a listening connection synchronously or asynchronously.

Do not call DoListen to open a listening connection. Instead, call the public Listen (see page 73) method, which orchestrates the opening of a connection either synchronously or asynchronously, depending on the socket type.

See Also

Listen (see page 73)

2.1.7.4.8 TCustomWinSocket.DoOpen Method

Opens a connection to a remote socket.

Pascal

```
procedure DoOpen;
```

C++

```
__fastcall DoOpen();
```

Description

The socket uses DoOpen internally to open a connection to a remote socket. DoOpen handles those tasks that are common to establishing a connection synchronously or asynchronously.

Do not call DoOpen to open a connection. Instead, call the public Open (see page 75) method, which orchestrates the opening of a connection either synchronously or asynchronously, depending on the socket type.

See Also

Open (see page 75)

2.1.7.4.9 TCustomWinSocket.Error Method

Generates an OnErrorEvent (see page 89) event.

Pascal

```
procedure Error(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer);
dynamic;
```

C++

```
__fastcall Error(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode);
```

Description

Error is called automatically when the Windows socket object receives error notifications. It generates the OnErrorEvent (see page 89) event. When overriding this method, be sure to call the inherited method, because socket components rely on the OnErrorEvent (see page 89) event for their own error handling.

The Socket parameter indicates the Windows socket object that encounters the error condition.

The ErrorEvent parameter indicates what the socket was trying to do when the error occurred. It has one of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	For client sockets, this indicates that the client socket can't locate the server, or that a problem on the server prevents the opening of a connection. For server sockets, this indicates that a client connection request that has already been accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.
eeAccept	For server sockets only, this indicates that a problem occurred when trying to accept a client connection request.

The ErrorCode parameter is the error code received by the Windows socket object. Changing this value to 0 in the Error method prevents the socket from raising an exception. For information on possible error codes, see the Microsoft documentation on Windows sockets.

See Also

OnErrorEvent (see page 89)

2.1.7.4.10 TCustomWinSocket.Event Method

Triggers an OnSocketEvent (see page 90) event.

Pascal

```
procedure Event(Socket: TCustomWinSocket; SocketEvent: TSocketEvent); dynamic;
```

C++

```
__fastcall Event(TCustomWinSocket Socket, TSocketEvent SocketEvent);
```

Description

Event is called automatically when the Windows socket object receives notifications or when it performs certain actions. It generates the OnSocketEvent (see page 90) event. When overriding this method, be sure to call the inherited method, because socket components rely on the OnSocketEvent (see page 90) event to generate many of their own events. The `Socket` parameter indicates the Windows socket that generates or receives notification of the event. The `SocketEvent` parameter indicates the event that occurred.

See Also

Connected (see page 83)

OnSocketEvent (see page 90)

2.1.7.4.11 TCustomWinSocket.InitSocket Method

Generates a windows Internet socket address structure from a description of the desired socket port.

Pascal

```
function InitSocket(const Name: string; const Address: string; const Service: string; Port:
Word; Client: Boolean): TSockAddrIn;
```

C++

```
__fastcall TSockAddrIn InitSocket(const UnicodeString Name, const UnicodeString Address, const
UnicodeString Service, Word Port, Boolean Client);
```

Description

Applications can not call InitSocket. It is called by the Open (see page 75) or Listen (see page 73) method to obtain a Windows Internet socket address structure for the parameters obtained from a socket component.

The Name parameter is the host name for the socket.The Address parameter is the IP address. The Service parameter is the service for which the socket is to used. The Port parameter is the port number for the socket. The Client parameter indicates whether the socket address structure is to be used by a client socket to describe the destination of a connection or by a server socket to describe the listening connection.

The host portion of the Windows Internet socket address is obtained from the host name if it is given. Otherwise the Address parameter is used. If Client is true, either the Name or the Address parameter must be specified. If Client is false, they may both be left blank to specify a server that listens on multiple addresses.

If port number for the Windows Internet socket address is obtained from the Service parameter if it is given. Otherwise, the Port parameter is used. One of these two parameters must be specified.

See Also

Addr (see page 82)

Listen (see page 73)

LookupName (see page 74)

LookupService (see page 75)

Open (see page 75)

RemoteAddr (see page 87)

Address (see page 22)

Host (see page 22)

Port (see page 23)

Service (see page 23)

AsyncInitSocket (see page 68)

2.1.7.4.12 TCustomWinSocket.Listen Method

Opens a listening connection for a server socket.

Pascal

```
procedure Listen(const Name: string; const Address: string; const Service: string; Port: Word;
QueueSize: Integer; Block: Boolean = True);
```

C++

```
__fastcall Listen(const UnicodeString Name, const UnicodeString Address, const UnicodeString
Service, Word Port, int QueueSize, Boolean Block = True);
```

Description

Server socket components call Listen to open a listening connection.

Listen creates the Windows socket that will make the listening connection, and binds the Windows Internet socket address derived from the parameters to the socket. Listen then generates an OnSocketEvent (see page 90) of type seListen. Finally, Listen opens the socket as a listening connection with a queue that can hold QueueSize client requests, setting the Connected (see page 83) property to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the server socket component.

QueueSize indicates how many client requests the listening connection can hold. TServerSocket (see page 108) sets the QueueSize to 5.

Block indicates whether the socket can look up the information it needs to establish a connection asynchronously. When Block is true, the socket blocks execution for each lookup instead of retrieving information about the connection using asynchronous notifications.

See Also

Connected (see page 83)

InitSocket (see page 72)

OnSocketEvent (see page 90)

Address (see page 22)

Host (see page 22)

Port (see page 23)

Service (see page 23)

TServerSocket (see page 108)

2.1.7.4.13 TCustomWinSocket.Lock Method

Starts a critical section, blocking all other execution threads until the Unlock (see page 81) method is called.

Pascal

```
procedure Lock;
```

C++

```
__fastcall Lock();
```

Description

Call Lock before beginning a section of code that is not thread-safe. Call Unlock (see page 81) when the critical section of code is completed. Unless absolutely necessary, do not use Lock to lock lengthy operations such as reading and writing over a blocking connection. Locking lengthy operations can severely impact performance, because all execution threads in the application that use the Windows socket object must wait until Unlock (see page 81) is called.

See Also

TThread

Unlock (see page 81)

2.1.7.4.14 TCustomWinSocket.LookupName Method

Generates the portion of a Windows Internet socket address structure that describes the host.

Pascal

```
function LookupName(const name: string): TInAddr;
```

C++

```
__fastcall TInAddr LookupName(const UnicodeString name);
```

Description

Call LookupName to obtain the Internet address associated with the system identified by the name parameter. LookupName is used by InitSocket (see page 72) to generate the portion of the Windows Internet socket address structure that identifies the host.

See Also

InitSocket (see page 72)

Host (see page 22)

2.1.7.4.15 TCustomWinSocket.LookupService Method

Returns the port number associated with a specific service.

Pascal

```
function LookupService(const service: string): Integer;
```

C++

```
__fastcall int LookupService(const UnicodeString service);
```

Description

Call LookupService to obtain the standard port number for the service named by the service parameter. Windows provides a number of standard service names such as ftp, http, finger, or time. Each standard service is associated, by convention, with a specific port number. These standard port numbers allow client sockets to indicate the desired service by requesting a specific port number. They allow server sockets to listen on an expected port number for the service offered. Servers can specify additional services and their associated ports in a SERVICES file. For more information on the SERVICES file, see the Microsoft documentation for Windows sockets.

InitSocket (see page 72) calls LookupService to obtain the port number for the Service specified by a socket component.

See Also

InitSocket (see page 72)

Service (see page 23)

2.1.7.4.16 TCustomWinSocket.Open Method

Opens a connection to a remote socket.

Pascal

```
procedure Open(const Name: string; const Address: string; const Service: string; Port: Word;
Block: Boolean = True);
```

C++

```
__fastcall Open(const UnicodeString Name, const UnicodeString Address, const UnicodeString
Service, Word Port, Boolean Block = True);
```

Description

Client socket components call Open to open a connection to a server socket.

Open creates the Windows socket that will connect to the server socket, and generates an OnSocketEvent (see page 90) of type seLookup. After the OnSocketEvent (see page 90), Open binds the Windows Internet socket address derived from the parameters to the socket, locating the server socket. Open then generates an OnSocketEvent (see page 90) of type seConnecting. When the server socket accepts the connection, Open completes the connection to the server socket. Then it

generates an OnSocketEvent (see page 90) of type seConnect. After the OnSocketEvent (see page 90), the Connected (see page 83) property is set to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the client socket component.

Block indicates whether the socket can look up the information it needs to establish a connection asynchronously. When Block is true, the socket blocks execution for each lookup instead of retrieving information about the connection using asynchronous notifications.

See Also

Connected (see page 83)

InitSocket (see page 72)

OnSocketEvent (see page 90)

Address (see page 22)

Host (see page 22)

Port (see page 23)

Service (see page 23)

TClientSocket (see page 24)

2.1.7.4.17 TCustomWinSocket.Read Method

Generates an OnSocketEvent (see page 90) of type seRead in response to notifications that the socket connection is ready.

Pascal

```
procedure Read(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Read(TSocket Socket);
```

Description

Read is called in response to asynchronous notifications that the socket connection is ready for reading. The Windows socket can then read from the connection in the OnSocketEvent (see page 90) event handler or the socket component can read from the connection in an OnRead or OnClientRead event handler. The event handler can use the ReceiveBuf (see page 77) method or the ReceiveText (see page 78) method to read the information from the connection.

Note: Read is only called for asynchronous read notifications in a non-blocking socket. Blocking sockets must use a TWinSocketStream (see page 138) object for reading. The TWinSocketStream (see page 138) object waits for the remote socket to be ready before transferring information.

See Also

ASyncStyles (see page 83)

OnSocketEvent (see page 90)

ReceiveBuf (see page 77)

ReceiveText (see page 78)

OnRead (see page 63)

OnClientRead (see page 53)

TWinSocketStream (see page 138)

2.1.7.4.18 TCustomWinSocket.ReceiveBuf Method

Reads up to Count bytes from the socket connection into the Buf parameter.

Pascal

```
function ReceiveBuf(var Buf; Count: Integer): Integer;
```

C++

```
__fastcall int ReceiveBuf( Buf, int Count);
```

Description

Use ReceiveBuf to read from the socket connection in the OnSocketEvent (see page 90) event handler of a Windows socket object or in the OnRead or OnClientRead event handler of a socket component. ReceiveBuf returns the number of bytes actually read. If no bytes are read, ReceiveBuf returns –1.

ReceiveBuf only works in response to a read notification to a non-blocking windows socket. Blocking sockets must use a TWinSocketStream (see page 138) for reading. The TWinSocketStream (see page 138) object waits for the remote socket to be ready before transferring information.

Note: While the ReceiveLength (see page 77) method can return an estimate of the size of buffer required to retrieve information from the socket, the number of bytes it returns is not necessarily accurate.

See Also

ASyncStyles (see page 83)

OnSocketEvent (see page 90)

ReceiveLength (see page 77)

ReceiveText (see page 78)

OnRead (see page 63)

OnClientRead (see page 53)

TWinSocketStream (see page 138)

2.1.7.4.19 TCustomWinSocket.ReceiveLength Method

Returns the number of bytes ready to be sent over the socket connection.

Pascal

```
function ReceiveLength: Integer;
```

C++

```
__fastcall int ReceiveLength();
```

Description

Call ReceiveLength to determine the amount of information to read over the socket connection in response to an asynchronous read notification.

Note: ReceiveLength is not guaranteed to be accurate for streaming socket connections.

See Also

ASyncStyles (see page 83)

ReceiveBuf (see page 77)

2.1.7.4.20 TCustomWinSocket.ReceiveText Method

Reads a string from the socket connection.

Pascal

```
function ReceiveText: string;
```

C++

```
__fastcall UnicodeString ReceiveText();
```

Description

Use ReceiveText to read a string from the socket connection in the OnSocketEvent (see page 90) event handler of a Windows socket object or in the OnRead or OnClientRead event handler of a socket component. ReceiveText returns the string that was read.

ReceiveText only works in response to a read notification to a non-blocking windows socket. Blocking sockets must use a TWinSocketStream (see page 138) for reading. The TWinSocketStream (see page 138) object waits for the remote socket to be ready before transferring information.

See Also

ASyncStyles (see page 83)

OnSocketEvent (see page 90)

ReceiveBuf (see page 77)

OnRead (see page 63)

OnClientRead (see page 53)

TWinSocketStream (see page 138)

2.1.7.4.21 TCustomWinSocket.SendBuf Method

Writes Count bytes to the socket connection from the Buf parameter.

Pascal

```
function SendBuf(var Buf; Count: Integer): Integer;
```

C++

```
__fastcall int SendBuf( Buf, int Count);
```

Description

Use SendBuf to write to the socket connection. Call this method from the OnSocketEvent (see page 90) event handler of a Windows socket object or in the OnWrite or OnClientWrite event handler of a socket component. Alternately, Use SendBuf to write when a connection is first formed when the socket does not expect notification that the socket on the other end of the connection is expecting to read.

For non-blocking sockets, the data is sent to the WinSock DLL which has it's own internal buffers. If the WinSock can accept additional data, SendBuf returns immediately with the number of bytes queued. If the WinSock internal buffer space is not able to accept the buffer being sent, SendBuf returns -1 and no data is queued at all. In this case, wait a bit for the WinSock to have a chance to send out already-queued data; then try again.

For blocking sockets, SendBuf returns the number of bytes actually written.

If an error occurs while writing to the connection, SendBuf terminates the connection and raises an ESocketError (see page 13) exception.

See Also

ESocketError (see page 13)

OnSocketEvent (see page 90)

SendStream (see page 79)

SendText (see page 80)

OnWrite (see page 63)

OnClientWrite (see page 53)

TWinSocketStream (see page 138)

2.1.7.4.22 TCustomWinSocket.SendStream Method

Writes all the information that can be read from the AStream parameter to the socket connection.

Pascal

```
function SendStream(AStream: TStream): Boolean;
```

C++

```
__fastcall Boolean SendStream(TStream AStream);
```

Description

Use SendStream to write to the socket connection. The writing may occur in the OnSocketEvent (see page 90) event handler of a Windows socket object or in the OnWrite or OnClientWrite event handler of a socket component. Alternately, SendStream may write from a socket that is expected to write to the connection without a notification to signal the connection's readiness to read. SendStream reads information from the stream indicated by AStream and writes it to the socket connection. The value returned by SendStream indicates whether any information was successfully written to the connection.

Note: The Stream passed as a parameter to SendStream becomes “owned” by the windows socket object. The Windows socket object frees the stream when it is finished with it. Do not attempt to free the stream after it has been passed as a parameter.

See Also

OnSocketEvent (see page 90)

SendBuf (see page 78)

SendText (see page 80)

OnWrite (see page 63)

OnClientWrite (see page 53)

TStream

TWinSocketStream (see page 138)

2.1.7.4.23 TCustomWinSocket.SendStreamThenDrop Method

Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.

Pascal

```
function SendStreamThenDrop(AStream: TStream): Boolean;
```

C++

```
__fastcall Boolean SendStreamThenDrop(TStream AStream);
```

Description

Use SendStreamThenDrop to write to a socket connection when the connection is no longer needed after the writing is complete. SendStream (see page 79) reads information from the stream indicated by AStream and writes it to the socket connection, shutting down the connection when it is done. The value returned by SendStream (see page 79) indicates whether any information was successfully written to the connection.

Note: The Stream passed as a parameter to SendStream (see page 79) becomes "owned" by the windows socket object. The Windows socket object frees the stream when it is finished with it. Do not attempt to free the stream after it has been passed as a parameter.

See Also

Disconnect (see page 70)

SendBuf (see page 78)

SendStream (see page 79)

SendText (see page 80)

TStream

TWinSocketStream (see page 138)

2.1.7.4.24 TCustomWinSocket.SendText Method

Writes the string S to the socket connection.

Pascal

```
function SendText(const S: AnsiString): Integer;
```

C++

```
__fastcall int SendText(const AnsiString S);
```

Description

Use SendText to write a string to the socket connection. The writing may occur in the OnSocketEvent (see page 90) event handler of a Windows socket object or in the OnWrite or OnClientWrite event handler of a socket component. Alternately, SendText may write from a socket that is expected to write to the connection without a notification to signal the connection's readiness to read. If an error occurs while writing to the connection, SendText terminates the connection and raises an ESocketError (see page 13) exception.

SendText returns the number of bytes sent. Note that this may be less than the length of the string S if the socket is nonblocking.

See Also

ESocketError (see page 13)

OnSocketEvent (see page 90)

SendBuf (see page 78)

SendStream (see page 79)

OnWrite (see page 63)

OnClientWrite (see page 53)

TWinSocketStream (see page 138)

2.1.7.4.25 TCustomWinSocket.SetAsyncStyles Method

Sets the ASyncStyles (see page 83) property.

Pascal

```
procedure SetAsyncStyles(Value: TASyncStyles);
```

C++

```
__fastcall SetAsyncStyles(TASyncStyles Value);
```

Description

SetAsyncStyles implements the ASyncStyles (see page 83) property. Override SetAsyncStyles to limit the value of ASyncStyles (see page 83) to notifications that are appropriate to a particular type of Windows socket.

See Also

ASyncStyles (see page 83)

2.1.7.4.26 TCustomWinSocket.Unlock Method

Reenables execution in other threads after a critical section begun by the Lock (see page 74) method.

Pascal

```
procedure Unlock;
```

C++

```
__fastcall Unlock();
```

Description

Call Unlock at the end of a section of code that is not thread-safe. Every call to Unlock should match a call to the Lock (see page 74) method at the beginning of the critical section of code.

See Also

Lock (see page 74)

TThread

2.1.7.4.27 TCustomWinSocket.Write Method

Generates an OnSocketEvent (see page 90) of type seWrite in response to notifications that the socket connection is ready.

Pascal

```
procedure Write(Socket: TSocket); virtual;
```

C++

```
virtual __fastcall Write(TSocket Socket);
```

Description

Write is called in response to asynchronous notifications that the socket connection is ready for the socket to send information. The Windows socket can write to the connection in the OnSocketEvent (see page 90) event handler or the socket component can write to the connection in an OnWrite or OnClientWrite event handler. The event handler can use the SendBuf (see page

78) method, the SendStream (see page 79) method, or the SendText (see page 80) method to write the information to the connection.

Note: Write is only called for asynchronous write notifications in a non-blocking socket. Blocking sockets must use their own mechanisms for determining when the connection is ready for the socket to write.

See Also

ASyncStyles (see page 83)

OnSocketEvent (see page 90)

SendBuf (see page 78)

SendStream (see page 79)

SendText (see page 80)

OnWrite (see page 63)

OnClientWrite (see page 53)

TWinSocketStream (see page 138)

2.1.7.5 TCustomWinSocket Properties

The properties of the TCustomWinSocket class are listed here.

Public Properties

	Name	Description
	Addr (see page 82)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 83)	Determines which asynchronous events the socket can receive.
	Connected (see page 83)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 84)	Points to application-specific data associated with the socket.
	Handle (see page 84)	Specifies the handle of the window that receives notifications from the Windows socket.
	LocalAddress (see page 85)	Specifies the IP address of the local system that was bound to the socket connection.
	LocalHost (see page 85)	Specifies the name of the local system.
	LocalPort (see page 86)	Specifies the ID number of the port used by the socket connection.
	LookupState (see page 86)	Indicates any asynchronous activities occurring while opening a connection.
	RemoteAddr (see page 87)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
	RemoteAddress (see page 87)	Specifies the IP address of the remote system that was bound to the socket connection.
	RemoteHost (see page 88)	Specifies the name of the remote system.
	RemotePort (see page 88)	Specifies the ID number of the port used by the RemoteHost (see page 88) to form the socket connection.
	SocketHandle (see page 89)	Specifies the Windows handle for the socket.

2.1.7.5.1 TCustomWinSocket.Addr Property

Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.

Pascal

```
property Addr: TSockAddrIn;
```

C++

```
__property TSockAddrIn Addr;
```

Description

Use Addr to represent the port used by the socket when working directly with Windows socket API calls. For client sockets, the value of Addr represents the target remote port. For server sockets, Addr is determined by the values of the LocalAddress (see page 85) and LocalPort (see page 86) properties.

See Also

LocalAddress (see page 85)

LocalPort (see page 86)

RemoteAddr (see page 87)

2.1.7.5.2 TCustomWinSocket.ASyncStyles Property

Determines which asynchronous events the socket can receive.

Pascal

```
property ASyncStyles: TAsyncStyles;
```

C++

```
__property TAsyncStyles ASyncStyles;
```

Description

Use ASyncStyles to determine what sorts of notifications the socket needs to respond to. Set ASyncStyles to change the notifications that the socket receives from the socket connection. ASyncStyles is a set drawn from the following values:

Value	Meaning
asRead	The socket receives notification that the connection is ready for reading.
asWrite	The socket receives notification that the connection is ready for writing.
asOOB	The socket receives notification when out-of-band data arrives.
asAccept	The socket receives notification when another socket requests a connection.
asConnect	The socket receives notification when a communication link to another socket is opened.
asClose	The socket receives notification when a communication link to another socket is terminated.

Any notification specified by ASyncStyles arrives as a window message to the Handle (see page 84) property.

Note: If the socket is a non-blocking socket, then ASyncStyles should include asRead and asWrite so that the socket will be informed of asynchronous reading and writing events.

See Also

Handle (see page 84)

ClientType (see page 38)

ServerType (see page 131)

2.1.7.5.3 TCustomWinSocket.Connected Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read (see page 76) Connected before trying to use or change the Windows socket connection. Connected indicates whether the socket connection has been established. When Connected is true, the Windows socket is open and available for use. When Connected is false, the Windows socket is closed and can be changed.

To open a client Windows socket, use the Open (see page 75) method. To open a server Windows socket, use the Listen (see page 73) method. The Windows socket that represents the server endpoint of a connection to a client socket is opened in its constructor.

To close a Windows socket, use the Close (see page 69) method.

See Also

Open (see page 75)

Close (see page 69)

Listen (see page 73)

Active (see page 21)

2.1.7.5.4 TCustomWinSocket.Data Property

Points to application-specific data associated with the socket.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate information specific to the socket connection with the Windows socket object. For example, Data can be used to store access and authentication information used by a server socket to evaluate client connection requests.

Data can store information in a thread-safe manner that would otherwise require the use of global variables. This is useful given the multi-threaded nature of many client-server applications.

2.1.7.5.5 TCustomWinSocket.Handle Property

Specifies the handle of the window that receives notifications from the Windows socket.

Pascal

```
property Handle: HWnd;
```

C++

```
__property HWnd Handle;
```

Description

Use Handle to send messages to the Windows socket or to subclass the Window procedure that receives messages of asynchronous events from the socket connection.

See Also

ASyncStyles (see page 83)

2.1.7.5.6 TCustomWinSocket.LocalAddress Property

Specifies the IP address of the local system that was bound to the socket connection.

Pascal

```
property LocalAddress: string;
```

C++

```
__property UnicodeString LocalAddress;
```

Description

LocalAddress is a string of four numeric (byte) values in the standard Internet dot notation, such as

`123.197.1.2`

Each address identifies the system and indicates the types of interfaces supported by the socket. A single system (machine) can support multiple addresses. LocalAddress is the address from among the possible addresses supported by the local system that is bound to the socket connection.

A single LocalAddress can be used simultaneously by multiple socket connections. Each of those connections must have a unique LocalPort (see page 86) number to distinguish it from other connections to the same LocalAddress.

See Also

LocalHost (see page 85)

LocalPort (see page 86)

RemoteAddress (see page 87)

Address (see page 22)

2.1.7.5.7 TCustomWinSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: string;
```

C++

```
__property UnicodeString LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the Windows socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as

`http://www.wSite.Com`

LocalHost provides an alias for the local IP address given in the LocalAddress (see page 85) property. Most Intranets provide host names for the IP addresses of systems on the net. On Windows 95 and NT machines, if a host name is not available, create one for the local IP address by entering the name into the HOSTS file. See the Microsoft documentation on Windows sockets for more information on the HOSTS file.

See Also

LocalAddress (see page 85)

RemoteHost (see page 88)

Host (see page 22)

2.1.7.5.8 TCustomWinSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: Integer;
```

C++

```
__property int LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Port numbers allow a single system, identified by the LocalAddress (see page 85) property, to host multiple connections simultaneously. Each combination of LocalPort and LocalAddress (see page 85) can only be bound to a single socket connection. This combination is represented by the Addr (see page 82) property.

Many values of LocalPort are associated by convention with a particular service such as ftp or http. To determine whether the value of LocalPort is associated with a specific service, use the LookupService (see page 75) method.

See Also

Addr (see page 82)

LocalAddress (see page 85)

LookupService (see page 75)

RemotePort (see page 88)

2.1.7.5.9 TCustomWinSocket.LookupState Property

Indicates any asynchronous activities occurring while opening a connection.

Pascal

```
property LookupState: TLookupState;
```

C++

```
__property TLookupState LookupState;
```

Description

LookupState is used internally to coordinate asynchronous activities while opening a connection. The AsyncInitSocket (see page 68) method sets the value of LookupState to indicate when it is looking up various types of information. The following table lists the possible values:

Value	Meaning
lsIdle	The socket is not currently looking up any information asynchronously.
lsLookupAddress	The socket is looking up an IP address asynchronously and has not yet received the result.
lsLookupService	The socket is looking up a Service asynchronously and has not yet received the result.

See Also

AsyncInitSocket (see page 68)

Open (see page 75)

Listen (see page 73)

2.1.7.5.10 TCustomWinSocket.RemoteAddr Property

Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.

Pascal

```
property RemoteAddr: TSockAddrIn;
```

C++

```
__property TSockAddrIn RemoteAddr;
```

Description

When working directly with Windows socket API calls, use RemoteAddr to represent the port used by the socket on the other end of the connection. The value of RemoteAddr is determined by the values of the RemoteAddress (see page 87) and RemotePort (see page 88) properties.

See Also

Addr (see page 82)

RemoteAddress (see page 87)

RemotePort (see page 88)

2.1.7.5.11 TCustomWinSocket.RemoteAddress Property

Specifies the IP address of the remote system that was bound to the socket connection.

Pascal

```
property RemoteAddress: string;
```

C++

```
__property UnicodeString RemoteAddress;
```

Description

RemoteAddress is a string of four numeric (byte) values in the standard Internet dot notation, such as

```
123.197.1.2
```

Each address identifies the system on the other end of the socket connection and indicates the types of interfaces supported by the remote socket. A single system can support multiple addresses. RemoteAddress is the address from among the possible addresses supported by the remote system that is bound to the socket connection.

A single RemoteAddress may be participating in multiple socket connections. Each of those connections has a unique RemotePort (see page 88) number to distinguish it from other connections to the same RemoteAddress. The RemoteAddr (see page 87) property uniquely identifies the RemoteAddress and RemotePort (see page 88) combination used by the socket connection.

See Also

LocalAddress (see page 85)

RemoteAddr (see page 87)

RemoteHost (see page 88)

RemotePort (see page 88)

Address (see page 22)

2.1.7.5.12 TCustomWinSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: string;
```

C++

```
__property UnicodeString RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as

`http://www.wSite.Com`

LocalHost (see page 85) provides an alias for the local IP address given in the RemoteAddress (see page 87) property.

See Also

LocalHost (see page 85)

RemoteAddress (see page 87)

Address (see page 22)

Host (see page 22)

2.1.7.5.13 TCustomWinSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost (see page 88) to form the socket connection.

Pascal

```
property RemotePort: Integer;
```

C++

```
__property int RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteAddress (see page 87) or RemoteHost (see page 88) property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteAddress (see page 87) can only be bound to a single socket connection. This combination is represented by the RemoteAddr (see page 87) property.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupService (see page 75) method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

See Also

LookupState (see page 86)

RemoteAddr (see page 87)

RemoteAddress (see page 87)

RemoteHost (see page 88)

Port (see page 23)

2.1.7.5.14 TCustomWinSocket.SocketHandle Property

Specifies the Windows handle for the socket.

Pascal

```
property SocketHandle: TSocket;
```

C++

```
__property TSocket SocketHandle;
```

Description

Use SocketHandle for Windows socket API calls that require the handle to the socket. When the Connected (see page 83) property is false, SocketHandle is INVALID_SOCKET.

SocketHandle is set to a valid Windows socket handle by the Listen (see page 73) or Open (see page 75) method.

See Also

Connected (see page 83)

Create (see page 66)

Listen (see page 73)

Open (see page 75)

2.1.7.6 TCustomWinSocket Events

The events of the TCustomWinSocket class are listed here.

Public Events

	Name	Description
	OnErrorEvent (see page 89)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 90)	Occurs when the Windows socket receives notification of asynchronous events.

2.1.7.6.1 TCustomWinSocket.OnErrorEvent Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnErrorEvent: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnErrorEvent;
```

Description

Socket components provide an OnErrorEvent event handler that converts error notifications into error events on the socket component. Most applications will therefore not provide an event handler for OnErrorEvent.

See Also

ESocketError (see page 13)

InitSocket (see page 18)

ASyncStyles (see page 83)

2.1.7.6.2 TCustomWinSocket.OnSocketEvent Event

Occurs when the Windows socket receives notification of asynchronous events.

Pascal

```
property OnSocketEvent: TSocketEventEvent;
```

C++

```
__property TSocketEventEvent OnSocketEvent;
```

Description

Socket components provide an OnSocketEvent event handler that converts Windows socket notifications into events on the socket component. Most applications will therefore not provide an event handler for OnSocketEvent. Replacing this event handler can prevent some expected events from occurring.

The Sender parameter is the Windows socket object that owns the event handler. Usually, this is the same as the Socket parameter.

The Socket parameter indicates the Windows socket that generates or receives notification of the event.

The SocketEvent parameter indicates the event that occurred.

See Also

ASyncStyles (see page 83)

InitSocket (see page 18)

2.1.8 TServerAcceptThread Class

Class Hierarchy

Pascal

```
TServerAcceptThread = class(TThread);
```

C++

```
class TServerAcceptThread : public TThread;
```

File

ScktComp

Description

This is class ScktComp.TServerAcceptThread.

2.1.8.1 TServerAcceptThread Members

The following tables list the members exposed by TServerAcceptThread.

Public Methods

	Name	Description
	Create (see page 91)	This is Create, a member of class TServerAcceptThread.
V	Execute (see page 91)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Properties

	Name	Description
R	ServerSocket (see page 95)	This is ServerSocket, a member of class TServerAcceptThread.

2.1.8.2 TServerAcceptThread.Create Constructor

Pascal

```
constructor Create(CreateSuspended: Boolean; ASocket: TServerWinSocket);
```

C++

```
__fastcall TServerAcceptThread(Boolean CreateSuspended, TServerWinSocket ASocket);
```

Description

This is Create, a member of class TServerAcceptThread.

2.1.8.3 TServerAcceptThread Methods

The methods of the TServerAcceptThread class are listed here.

Public Methods

	Name	Description
V	Execute (see page 91)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

2.1.8.3.1 TServerAcceptThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
```

```
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
```

```
      YouProcess.Priority := tpLower;  // set the priority to lower than normal
      YouProcess.Resume; { now run the thread }
    end
    else
      MessageDlg('This thread is still running.  You are going to hurt yourself!',
        mtInformation, [mbOk], 0);
  end;

  procedure TForm1.Button5Click(Sender: TObject);
  begin
    if youthreadRunning then YouProcess.Terminate();
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    mythreadRunning:= FALSE;
    youthreadRunning:= FALSE;
  end;
```

See Also

Create

Terminated

Synchronize

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

2.1.8.4 TServerAcceptThread Properties

The properties of the TServerAcceptThread class are listed here.

Public Properties

	Name	Description
	ServerSocket (see page 95)	This is ServerSocket, a member of class TServerAcceptThread.

2.1.8.4.1 TServerAcceptThread.ServerSocket Property

Pascal

```
property ServerSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerSocket;
```

Description

This is ServerSocket, a member of class TServerAcceptThread.

2.1.9 TServerClientThread Class

TServerClientThread is an execution thread used for a single connection to a client socket accepted by a Windows server socket.

Class Hierarchy

Pascal

```
TServerClientThread = class(TThread);
```

C++

```
class TServerClientThread : public TThread;
```

File

ScktComp

Description

TServerWinSocket (see page 120) objects spawn TServerClientThread objects to handle separate client connections in separate execution threads. This allows server sockets to perform slow reading and writing operations with client sockets without adversely affecting the performance on other connections.

TServerClientThread introduces new properties to

Access the TServerWinSocket (see page 120) object that spawned the thread and the TServerClientWinSocket (see page 104) that is connected to the client socket.

Store and access thread-specific data.

Determine if the thread is stored in a thread cache maintained by the TServerWinSocket (see page 120).

Applications that receive frequent client requests will want to derive a descendant class from TServerClientThread to handle reading and writing to separate clients. This is because TServerClientThread objects use the OnClientRead and OnClientWrite events of the associated server socket. These events are not thread-local.

See Also

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

2.1.9.1 TServerClientThread Members

The following tables list the members exposed by TServerClientThread.

Protected Methods

	Name	Description
	ClientExecute (see page 98)	Simulates OnClientRead and OnClientWrite events for the associated server socket.
	DoTerminate (see page 99)	Removes the thread from the list of active threads maintained by the associated server socket.
	EndConnect (see page 99)	Indicates whether the thread should execute again for another client.
	Error (see page 100)	Generates an OnClientError event for the associated server socket.
	Event (see page 100)	Generates an OnSocketEvent event for the associated server socket.
	Execute (see page 101)	Manages the caching of server client threads and calls the ClientExecute (see page 98) method to run the thread for individual client connections.
	HandleException (see page 101)	Displays an error message using the main VCL thread.
	ReActivate (see page 101)	Initializes the thread for a particular client connection.
	StartConnect (see page 102)	Indicates when the thread is ready to execute.

Public Methods

	Name	Description
	Create (see page 97)	Creates an instance of TServerClientThread (see page 95).
V	Destroy (see page 98)	Destroys the server client thread object and releases the memory allocated to it.

Public Properties

	Name	Description
R	ClientSocket (see page 102)	Specifies the TServerClientWinSocket (see page 104) object that uses the thread for all reading and writing over the socket connection.
	Data (see page 103)	Points to application-specific data associated with the thread.
	KeepInCache (see page 103)	Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 104).
R	ServerSocket (see page 104)	Specifies the TServerWinSocket (see page 120) object that manages all the TServerClientThread (see page 95) objects for its client connections.

2.1.9.2 TServerClientThread.Create Constructor

Creates an instance of TServerClientThread (see page 95).

Pascal

```
constructor Create(CreateSuspended: Boolean; ASocket: TServerClientWinSocket);
```

C++

```
__fastcall TServerClientThread(Boolean CreateSuspended, TServerClientWinSocket ASocket);
```

Description

Use Create to create an instance of TServerClientThread (see page 95) for a connection to a client socket, and add the thread to the cache maintained by the associated listening socket.

Initializes FreeOnTerminate to true.

Initializes Priority to tpHigher.

Initializes ClientSocket (see page 102) to the ASocket parameter and ServerSocket (see page 104) to ASocket.ServerWinSocket.

Starts the thread executing.

Adds the thread to the cache maintained by ServerSocket (see page 104).

Sets the OnErrorEvent and OnSocketEvent event handlers of ClientSocket (see page 102) so that events are sent to ServerSocket (see page 104).

Signals when initialization is complete and the thread is ready for reading from or writing to the client connection.

See Also

ClientSocket (see page 102)

FreeOnTerminate

Priority

ServerSocket (see page 104)

OnErrorEvent (see page 89)

OnSocketEvent (see page 90)

2.1.9.3 TServerClientThread.Destroy Destructor

Destroys the server client thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerClientThread();
```

Description

Do not call Destroy directly in an application. ServerClientThread objects are freed by the TServerWinSocket (see page 120) object that manages them.

Destroy frees the TServerClientWinSocket (see page 104) object that encapsulates the connection to the client socket before calling the inherited destructor.

See Also

ClientSocket (see page 102)

ServerSocket (see page 104)

2.1.9.4 TServerClientThread Methods

The methods of the TServerClientThread class are listed here.

Protected Methods

	Name	Description
	ClientExecute (see page 98)	Simulates OnClientRead and OnClientWrite events for the associated server socket.
	DoTerminate (see page 99)	Removes the thread from the list of active threads maintained by the associated server socket.
	EndConnect (see page 99)	Indicates whether the thread should execute again for another client.
	Error (see page 100)	Generates an OnClientError event for the associated server socket.
	Event (see page 100)	Generates an OnSocketEvent event for the associated server socket.
	Execute (see page 101)	Manages the caching of server client threads and calls the ClientExecute (see page 98) method to run the thread for individual client connections.
	HandleException (see page 101)	Displays an error message using the main VCL thread.
	ReActivate (see page 101)	Initializes the thread for a particular client connection.
	StartConnect (see page 102)	Indicates when the thread is ready to execute.

2.1.9.4.1 TServerClientThread.ClientExecute Method

Simulates OnClientRead and OnClientWrite events for the associated server socket.

Pascal

```
procedure ClientExecute; virtual;
```

C++

```
virtual __fastcall ClientExecute();
```

Description

TServerClientThread (see page 95) calls ClientExecute from its Execute (see page 101) method. ClientExecute checks

whether there is any information to read from the associated socket connection, and if so, simulates an OnClientRead event on the associated server socket. If there is no information sent from the client, ClientExecute simulates an OnClientWrite event.

Because the server socket that receives these events is not local to the thread, the events occur within the main VCL thread using the Synchronize method. This method can be too slow if the server socket has many client connections, because each thread must wait for any other thread to finish reading or writing. When a server socket receives many client connection requests, override ClientExecute to handle the reading or writing using a thread-local instance of TWinSocketStream (see page 138). In the ClientExecute method, be sure to check the Terminated property periodically.

Note: Do not use the properties and methods of global objects directly in the ClientExecute method. Instead, separate the use of objects that are not thread-local into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Execute (see page 101)

Synchronize

TWinSocketStream (see page 138)

Terminated

DoTerminate (see page 99)

2.1.9.4.2 TServerClientThread.DoTerminate Method

Removes the thread from the list of active threads maintained by the associated server socket.

Pascal

```
procedure DoTerminate; override;
```

C++

```
virtual __fastcall DoTerminate();
```

Description

DoTerminate overrides the inherited method to suppress the OnTerminate event. Instead, it removes the thread from the Connections property of the associated TServerWinSocket (see page 120). The thread is then either added to the server socket's thread cache or destroyed.

See Also

ClientExecute (see page 98)

Terminated

Connections (see page 130)

2.1.9.4.3 TServerClientThread.EndConnect Method

Indicates whether the thread should execute again for another client.

Pascal

```
function EndConnect: Boolean;
```

C++

```
__fastcall Boolean EndConnect();
```

Description

The Execute (see page 101) method calls EndConnect when the ClientExecute (see page 98) method finishes. It frees all memory associated with the current client connection and checks whether the Thread is being reused for another client connection. If the thread should be reused, EndConnect returns true to indicate that the ClientExecute (see page 98) method should be called again.

See Also

ClientExecute (see page 98)

Execute (see page 101)

2.1.9.4.4 TServerClientThread.Error Method

Generates an OnClientError event for the associated server socket.

Pascal

```
procedure Error(ErrorEvent: TErrorEvent; var ErrorCode: Integer); virtual;
```

C++

```
virtual __fastcall Error(TErrorEvent ErrorEvent, int ErrorCode);
```

Description

TServerClientThread (see page 95) assigns the Error method as an event handler for the OnErrorEvent method of the associated TClientWinSocket (see page 35) object. This allows errors received from the client to generate OnClientError events on the server socket.

Warning: Error is not thread-safe. It should not be called from the server client thread. It is only intended for use by the associated client socket, which is not local to the server client thread.

See Also

OnClientError (see page 52)

OnErrorEvent (see page 89)

2.1.9.4.5 TServerClientThread.Event Method

Generates an OnSocketEvent event for the associated server socket.

Pascal

```
procedure Event(SocketEvent: TSocketEvent); virtual;
```

C++

```
virtual __fastcall Event(TSocketEvent SocketEvent);
```

Description

TServerClientThread (see page 95) assigns the Event method as an event handler for the OnSocketEvent method of the associated TClientWinSocket (see page 35) object. This allows events generated by the client to trigger client events on the server socket.

Warning: Event is not thread-safe. It should not be called from the server client thread. It is only intended for use by the associated client socket, which is not local to the server client thread.

See Also

OnClientError (see page 52)

OnErrorEvent (see page 89)

2.1.9.4.6 TServerClientThread.Execute Method

Manages the caching of server client threads and calls the ClientExecute (see page 98) method to run the thread for individual client connections.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Execute is called automatically when the thread runs. A thread runs when Create (see page 97) is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Do not override the Execute method to implement the thread function of a TServerClientThread (see page 95) descendant. Instead, override the ClientExecute (see page 98) method, which is run after the thread is initialized for a particular client connection.

See Also

ClientExecute (see page 98)

ReActivate (see page 101)

2.1.9.4.7 TServerClientThread.HandleException Method

Displays an error message using the main VCL thread.

Pascal

```
procedure HandleException; virtual;
```

C++

```
virtual __fastcall HandleException();
```

Description

HandleException obtains the current exception object and calls TApplication.ShowException (see page 4240) (for descendants of Exception (see page 3949)) or the global ShowException (see page 4240) procedure (for other objects) to display an error message. HandleException wraps the call to ShowException (see page 4240) in the Synchronize method to ensure that the call is thread-safe.

See Also

ShowException

ShowException (see page 4240)

Synchronize

2.1.9.4.8 TServerClientThread.ReActivate Method

Initializes the thread for a particular client connection.

Pascal

```
procedure ReActivate(ASocket: TServerClientWinSocket);
```

C++

```
__fastcall ReActivate(TServerClientWinSocket ASocket);
```

Description

ReActivate is called when the thread is created or reactivated from the server socket's cache of threads. ReActivate assigns the Error (see page 100) and Event (see page 100) methods as event handlers for the associated TServerWinSocket (see page 120) and informs the server socket that the thread is now active.

See Also

Error (see page 100)

TServerClientThread (see page 95)

2.1.9.4.9 TServerClientThread.StartConnect Method

Indicates when the thread is ready to execute.

Pascal

```
function StartConnect: Boolean;
```

C++

```
__fastcall Boolean StartConnect();
```

Description

The Execute (see page 101) method calls StartConnect to determine when to call ClientExecute (see page 98). StartConnect waits until the ReActivate (see page 101) method has signaled that the thread is properly initialized, and then returns true if the thread is ready to run. If the thread is terminated for any reason before it is fully initialized, StartConnect returns false.

See Also

ReActivate (see page 101)

ClientExecute (see page 98)

Execute (see page 101)

2.1.9.5 TServerClientThread Properties

The properties of the TServerClientThread class are listed here.

Public Properties

	Name	Description
R	ClientSocket (see page 102)	Specifies the TServerClientWinSocket (see page 104) object that uses the thread for all reading and writing over the socket connection.
	Data (see page 103)	Points to application-specific data associated with the thread.
	KeepInCache (see page 103)	Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 104).
R	ServerSocket (see page 104)	Specifies the TServerWinSocket (see page 120) object that manages all the TServerClientThread (see page 95) objects for its client connections.

2.1.9.5.1 TServerClientThread.ClientSocket Property

Specifies the TServerClientWinSocket (see page 104) object that uses the thread for all reading and writing over the socket connection.

Pascal

```
property ClientSocket: TServerClientWinSocket;
```

C++

```
__property TServerClientWinSocket ClientSocket;
```

Description

Use ClientSocket to gain direct access to the Windows socket component that uses the thread. If ClientSocket is nil (Delphi) or NULL (C++), the thread is idle and available for reuse by another TServerClientWinSocket (see page 104) object. TServerClientThread (see page 95) uses ClientSocket to read from or write to the socket connection.

See Also

TServerClientWinSocket (see page 104)

2.1.9.5.2 TServerClientThread.Data Property

Points to application-specific data associated with the thread.

Pascal

```
property Data: Pointer;
```

C++

```
__property void * Data;
```

Description

Use Data to associate information with the thread object. Data can store information in a thread-safe manner that would otherwise require the use of global variables.

See Also

Data (see page 84)

2.1.9.5.3 TServerClientThread.KeepInCache Property

Determines whether the thread should be kept in the thread cache maintained by ServerSocket (see page 104).

Pascal

```
property KeepInCache: Boolean;
```

C++

```
__property Boolean KeepInCache;
```

Description

The TServerWinSocket (see page 120) that spawns server client threads for its connections to client sockets maintains a cache of server client threads for reuse. As server client threads are activated, the TServerWinSocket (see page 120) object sets the KeepInCache property to indicate whether the thread will fit in the thread cache.

When KeepInCache is true, FreeOnTerminate is set to false before the thread terminates, and the thread is left in the thread cache. When KeepInCache is false, the thread is freed when execution terminates.

See Also

FreeOnTerminate

ServerSocket (see page 104)

ThreadCacheSize (see page 132)

2.1.9.5.4 TServerClientThread.ServerSocket Property

Specifies the TServerWinSocket (see page 120) object that manages all the TServerClientThread (see page 95) objects for its client connections.

Pascal

```
property ServerSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerSocket;
```

Description

Use ServerSocket to get direct access to the listening Windows socket object that spawned the thread for a client connection it accepted. TServerClientThread (see page 95) passes events and errors to the appropriate event handlers assigned to ServerSocket. TServerClientThread (see page 95) adds itself to the cache of threads maintained by ServerSocket from its constructor, and removes itself when the thread terminates.

See Also

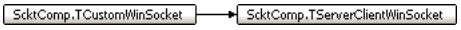
Create (see page 97)

TServerWinSocket (see page 120)

2.1.10 TServerClientWinSocket Class

TServerClientWinSocket describes the server endpoint of a socket connection to a client socket.

Class Hierarchy



Pascal

```
TServerClientWinSocket = class(TCustomWinSocket);
```

C++

```
class TServerClientWinSocket : public TCustomWinSocket;
```

File

SckComp

Description

TServerClientWinSocket is used by server socket components to manage the Windows socket API calls for the socket connection to an individual client socket. TServerClientWinSocket objects are created when a client socket connection is accepted by a listening server socket.

Applications can create and customize the properties of a TServerClientWinSocket object in an OnGetSocket event handler for a server socket component. Component writers can create descendants of TServerClientWinSocket that may be substituted for TServerClientWinSocket in the OnGetSocket event handler.

See Also

TClientWinSocket (see page 35)

TCustomServerSocket (see page 39)

TServerSocket (see page 108)

TServerWinSocket (see page 120)

2.1.10.1 TServerClientWinSocket Members

The following tables list the members exposed by TServerClientWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 68)	Looks up information or forms socket connections asynchronously.
	DoListen (see page 70)	Opens a listening connection.
	DoOpen (see page 71)	Opens a connection to a remote socket.
	Error (see page 71)	Generates an OnErrorEvent (see page 89) event.
	Event (see page 72)	Triggers an OnSocketEvent (see page 90) event.
	InitSocket (see page 72)	Generates a windows Internet socket address structure from a description of the desired socket port.
	SetAsyncStyles (see page 81)	Sets the ASyncStyles (see page 83) property.

Public Events

	Name	Description
	OnErrorEvent (see page 89)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnSocketEvent (see page 90)	Occurs when the Windows socket receives notification of asynchronous events.

Public Methods

	Name	Description
	Accept (see page 68)	Provides the interface for a method that accepts a connection to a client socket.
	Close (see page 69)	Shuts down the socket connection if it is open.
	Connect (see page 69)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 106)	Creates and initializes an instance of TServerClientWinSocket (see page 104).
	DefaultHandler (see page 70)	Processes message records.
	Destroy (see page 107)	Destroys an instance of TServerClientWinSocket (see page 104).
	Disconnect (see page 70)	Shuts down the socket connection represented by a Windows socket handle.
	Listen (see page 73)	Opens a listening connection for a server socket.
	Lock (see page 74)	Starts a critical section, blocking all other execution threads until the Unlock (see page 81) method is called.
	LookupName (see page 74)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 75)	Returns the port number associated with a specific service.
	Open (see page 75)	Opens a connection to a remote socket.
	Read (see page 76)	Generates an OnSocketEvent (see page 90) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 77)	Reads up to Count bytes from the socket connection into the Buf parameter.
	ReceiveLength (see page 77)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 78)	Reads a string from the socket connection.
	SendBuf (see page 78)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection.
	SendStreamThenDrop (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 80)	Writes the string S to the socket connection.
	Unlock (see page 81)	Reenables execution in other threads after a critical section begun by the Lock (see page 74) method.
	Write (see page 81)	Generates an OnSocketEvent (see page 90) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
	Addr (see page 82)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 83)	Determines which asynchronous events the socket can receive.
	Connected (see page 83)	Indicates whether the socket connection is open and available for communication with other machines.
	Data (see page 84)	Points to application-specific data associated with the socket.
	Handle (see page 84)	Specifies the handle of the window that receives notifications from the Windows socket.
	LocalAddress (see page 85)	Specifies the IP address of the local system that was bound to the socket connection.
	LocalHost (see page 85)	Specifies the name of the local system.
	LocalPort (see page 86)	Specifies the ID number of the port used by the socket connection.
	LookupState (see page 86)	Indicates any asynchronous activities occurring while opening a connection.
	RemoteAddr (see page 87)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
	RemoteAddress (see page 87)	Specifies the IP address of the remote system that was bound to the socket connection.
	RemoteHost (see page 88)	Specifies the name of the remote system.
	RemotePort (see page 88)	Specifies the ID number of the port used by the RemoteHost (see page 88) to form the socket connection.
	ServerWinSocket (see page 107)	Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 104).
	SocketHandle (see page 89)	Specifies the Windows handle for the socket.

2.1.10.2 TServerClientWinSocket.Create Constructor

Creates and initializes an instance of TServerClientWinSocket (see page 104).

Pascal

```
constructor Create(Socket: TSocket; ServerWinSocket: TServerWinSocket);
```

C++

```
__fastcall TServerClientWinSocket(TSocket Socket, TServerWinSocket ServerWinSocket);
```

Description

Use Create to create an instance of TServerClientWinSocket (see page 104) in the OnGetSocket event handler of a server socket component. The Socket parameter is the Windows socket handle for the accepted socket connection to a client socket. The ServerWinSocket (see page 107) parameter is the Windows socket object for the listening connection that accepted the connection to the client socket.

Create adds the new TServerClientWinSocket (see page 104) object to the list of connections maintained by the listening server socket. It then sets the OnSocketEvent (see page 90) event handler so that notifications from the connection to the client can be passed on to the server socket. After setting up the connection to the server socket, Create calls the inherited constructor.

See Also

OnSocketEvent (see page 90)

Create (see page 66)

Connections (see page 130)

2.1.10.3 TServerClientWinSocket.Destroy Destructor

Destroys an instance of TServerClientWinSocket (see page 104).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerClientWinSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TServerClientWinSocket (see page 104) reference is not nil, and only then calls Destroy. Instances of TServerClientWinSocket (see page 104) are freed automatically when the connection is closed.

Destroy removes the TServerClientWinSocket (see page 104) from the list of connections maintained by the listening windows socket.

See Also

System::TObject::Free (see page 3372)

Connections (see page 130)

2.1.10.4 TServerClientWinSocket Properties

The properties of the TServerClientWinSocket class are listed here.

Public Properties

	Name	Description
	ServerWinSocket (see page 107)	Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 104).

2.1.10.4.1 TServerClientWinSocket.ServerWinSocket Property

Describes the endpoint of the listening connection that accepted the connection to the client socket which is managed by this TServerClientWinSocket (see page 104).

Pascal

```
property ServerWinSocket: TServerWinSocket;
```

C++

```
__property TServerWinSocket ServerWinSocket;
```

Description

Use ServerWinSocket to gain access to the Windows socket object for the listening connection that spawned this TServerClientWinSocket (see page 104). Use the ServerWinSocket to trigger events that can be handled by the event handlers of the server socket, to get execution threads from the cache maintained by the Windows server socket, or to get information about the other active connections accepted by the server socket.

See Also

TServerWinSocket (see page 120)

2.1.11 TServerSocket Class

TServerSocket manages server socket connections for a TCP/IP server.

Class Hierarchy

Pascal

```
TServerSocket = class(TCustomServerSocket);
```

C++

```
class TServerSocket : public TCustomServerSocket;
```

File

SckComp

Description

Add a TServerSocket object to a form or data module to turn an application into a TCP/IP server. TServerSocket listens for requests for TCP/IP connections from other machines, and establishes connections when requests are received.

See Also

TClientSocket (see page 24)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

2.1.11.1 TServerSocket Members

The following tables list the members exposed by TServerSocket.

Protected Events

	Name	Description
	OnConnect (see page 60)	Occurs on client sockets just after the connection to the server is opened.
	OnConnecting (see page 60)	Occurs for a client socket after the server socket has been located, but before the connection is established.
	OnDisconnect (see page 61)	Occurs just before a client socket closes the connection to a server socket.
	OnError (see page 61)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnLookup (see page 62)	Occurs when a client socket is about to look up the server socket with which it wants to connect.
	OnRead (see page 63)	Occurs when a client socket should read information from the socket connection.
	OnWrite (see page 63)	Occurs when a client socket should write information to the socket connection.

Protected Fields

	Name	Description
	FServerSocket (see page 42)	This is FServerSocket, a member of class TCustomServerSocket.

Protected Methods

	Name	Description
A	DoActivate (see page 16)	Provides the prototype of a method to open or close the socket connection.
V	DoActivate (see page 42)	Opens or Closes the socket connection.
A	Error (see page 16)	Provides the prototype of a method that responds to error conditions.

	Error (see page 58)	Generates an OnError (see page 61) event.
	Event (see page 17)	Provides the prototype of a method that responds when specific socket events occur.
	Event (see page 58)	Dispatches the events that occur in the normal operation of the socket to the appropriate event handlers.
	GetGetSocketEvent (see page 43)	Returns the OnGetSocket (see page 54) event handler.
	GetGetThreadEvent (see page 43)	Returns the OnGetThread (see page 54) event handler.
	GetOnClientError (see page 44)	Returns the OnClientError (see page 52) event handler.
	GetOnClientEvent (see page 44)	Provides indexed access to the various event handlers that respond to client events.
	GetOnThreadEnd (see page 44)	Returns the OnThreadEnd (see page 55) event handler.
	GetOnThreadStart (see page 45)	Returns the OnThreadStart (see page 55) event handler.
	GetServerType (see page 45)	Returns the ServerType (see page 49) property.
	GetThreadCacheSize (see page 45)	Returns the value of the ThreadCacheSize (see page 50) property.
	InitSocket (see page 18)	Tells a Windows socket object to call the Error (see page 16) and Event (see page 17) methods.
	Loaded (see page 18)	Opens or closes the socket connection when the socket is loaded from a file, depending on the saved value of the Active (see page 21) property.
	SetActive (see page 19)	Sets the Active (see page 21) property.
	SetAddress (see page 20)	Sets the Address (see page 22) property.
	SetGetSocketEvent (see page 46)	Sets the OnGetSocket (see page 54) event handler.
	SetGetThreadEvent (see page 46)	Sets the OnGetThread (see page 54) event handler.
	SetHost (see page 20)	Sets the Host (see page 22) property.
	SetOnClientError (see page 47)	Sets the OnClientError (see page 52) event handler.
	SetOnClientEvent (see page 47)	Sets the client event handlers of the associated TServerWinSocket (see page 120) object.
	SetOnThreadEnd (see page 47)	Sets the OnThreadEnd (see page 55) event handler.
	SetOnThreadStart (see page 48)	Sets the OnThreadStart (see page 55) event handler.
	SetPort (see page 20)	Sets the Port (see page 23) property.
	SetServerType (see page 48)	Sets the ServerType (see page 49) property.
	SetService (see page 21)	Sets the Service (see page 23) property.
	SetThreadCacheSize (see page 49)	Sets the ThreadCacheSize (see page 50) property.

Protected Properties

	Name	Description
	Address (see page 22)	Specifies the IP address of the server system.
	Host (see page 22)	Specifies an alias for the IP address of the server system.

Public Methods

	Name	Description
	Close (see page 16)	Shuts down the socket connection.
	Create (see page 110)	Creates an instance of TServerSocket (see page 108).
	Destroy (see page 41)	Destroys an instance of TCustomServerSocket (see page 39).
	Open (see page 19)	Opens the socket connection.

Public Properties

	Name	Description
	Socket (see page 113)	Specifies the TServerWinSocket (see page 120) object that describes the endpoint of the listening connection.

Published Events

	Name	Description
	OnAccept (see page 114)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 115)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 116)	Occurs when one of the connections to a client socket is closed.

	OnClientError (see page 116)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 117)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 117)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 118)	Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 118)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 119)	Occurs just before a server socket is opened for listening.
	OnThreadEnd (see page 119)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 120)	Occurs when the execution thread for a connection to a client socket starts up.

Published Properties

	Name	Description
	Active (see page 111)	Indicates whether the socket connection is open and available for communication with other machines.
	Port (see page 111)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 112)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 112)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 113)	Specifies the maximum number of threads that can be reused for new client connections.

2.1.11.2 TServerSocket.Create Constructor

Creates an instance of TServerSocket (see page 108).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TServerSocket(TComponent * AOwner);
```

Description

Call Create to create and initialize a server socket component at runtime. TServerSocket (see page 108) objects placed at design time are created automatically.

After calling the inherited constructor, Create

Allocates the TServerWinSocket (see page 120) object that encapsulates the Windows socket connection for the server socket.

Initializes the event and error dispatchers for the server socket.

Initializes ThreadCacheSize (see page 113) to 10.

See Also

Create

ThreadCacheSize (see page 50)

2.1.11.3 TServerSocket Properties

The properties of the TServerSocket class are listed here.

Public Properties

	Name	Description
	Socket (see page 113)	Specifies the TServerWinSocket (see page 120) object that describes the endpoint of the listening connection.

Published Properties

	Name	Description
	Active (see page 111)	Indicates whether the socket connection is open and available for communication with other machines.
	Port (see page 111)	Specifies the ID number that identifies the server socket connection.
	ServerType (see page 112)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	Service (see page 112)	Specifies the name of the service for which the socket connection is used.
	ThreadCacheSize (see page 113)	Specifies the maximum number of threads that can be reused for new client connections.

2.1.11.3.1 TServerSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to true to make the socket open a connection when the application starts running. At runtime, use the Open or Close method to open or close the connection.

See Also

Close (see page 16)

Open (see page 19)

SetActive (see page 19)

2.1.11.3.2 TServerSocket.Port Property

Specifies the ID number that identifies the server socket connection.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Port numbers allow a single system, identified by the Host or Address property, to host multiple connections simultaneously. Many values of Port are associated by convention with a particular service such as ftp or http.

For server sockets, Port is the ID of the connection on which the server socket listens for client requests. Server sockets

generally set Port to a predefined value which clients can use to initiate connections.

For client sockets, Port is the ID of the desired server connection. The value of Port is usually associated with the service the client wishes to use on the server application.

Note: Trying to change Port when the connection is open will raise an ESocketError exception.

See Also

Address (see page 22)

ESocketError (see page 13)

Host (see page 22)

Service (see page 23)

SetPort (see page 20)

2.1.11.3.3 TServerSocket.ServerType Property

Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the server socket. When ServerType is stThreadBlocking, the execution of the connection thread is suspended while reading or writing until all information has been transferred over the connection. The thread for each connection generates OnClientRead or OnClientWrite events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over the socket connections asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default. OnClientRead or OnClientWrite events occur when the client socket on the other end of one of the connections tries to send or receive information over the connection.

Use a non-blocking socket when the socket needs to synchronize reading and writing with client sockets.

Note: When ServerType is stThreadBlocking, it is important that the OnClient... event handlers contain thread-safe code.

See Also

OnClientRead (see page 53)

OnClientWrite (see page 53)

TServerClientThread (see page 95)

2.1.11.3.4 TServerSocket.Service Property

Specifies the name of the service for which the socket connection is used.

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

Use Service to identify the use of the connection. Windows provides There are a number of standard service names such as ftp, http, finger, and time. Servers can specify additional services and their associated ports in a SERVICES file. For more information, see the Microsoft documentation for Windows sockets.

Certain port numbers are reserved for specific values of service. Thus, Service provides a more meaningful way to specify the server Port to use for the socket connection. For server sockets, using Service rather than Port ensures that the server will listen for TCP/IP requests on the appropriate port.

Note: Trying to change Service when the connection is open will raise an ESocketError exception.

See Also

ESocketError (see page 13)

Port (see page 23)

SetService (see page 21)

LookupService (see page 75)

2.1.11.3.5 TServerSocket.Socket Property

Specifies the TServerWinSocket (see page 120) object that describes the endpoint of the listening connection.

Pascal

```
property Socket: TServerWinSocket;
```

C++

```
__property TServerWinSocket Socket;
```

Description

Use Socket to obtain

Information about the connections currently active on the server socket.

Information about the connection threads that are cached for use by the server socket.

Access to the Windows socket handle for the listening connection that is needed for Windows socket API calls.

See Also

ThreadCacheSize (see page 50)

2.1.11.3.6 TServerSocket.ThreadCacheSize Property

Specifies the maximum number of threads that can be reused for new client connections.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution

thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

See Also

ServerType (see page 49)

TServerClientThread (see page 95)

ActiveThreads (see page 130)

IdleThreads (see page 131)

2.1.11.4 TServerSocket Events

The events of the TServerSocket class are listed here.

Published Events

	Name	Description
	OnAccept (see page 114)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnClientConnect (see page 115)	Occurs when a client socket completes a connection accepted by the server socket.
	OnClientDisconnect (see page 116)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 116)	Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 117)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 117)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 118)	Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.
	OnGetThread (see page 118)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnListen (see page 119)	Occurs just before a server socket is opened for listening.
	OnThreadEnd (see page 119)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 120)	Occurs when the execution thread for a connection to a client socket starts up.

2.1.11.4.1 TServerSocket.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that will be used in the server endpoint of the connection. The port number, and possibly the IP address, will differ from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect event occurs for the client socket.

See Also

OnConnect (see page 60)

OnListen (see page 61)

2.1.11.4.2 TServerSocket.OnClientConnect Event

Occurs when a client socket completes a connection accepted by the server socket.

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write an OnClientConnect event handler to take specific action when a client socket completes the socket connection to the server socket. For example, the socket may start reading or writing over the connection in an OnClientConnect event handler.

The order of server socket events leading up to OnClientConnect is as follows:

1An OnListen event occurs just before the server socket is opened for listening.

2The server socket receives client requests in a listening queue. The server socket accepts one of those requests, and receives a Windows socket handle for the new socket connection.

3The server socket generates an OnGetSocket event, passing in the Windows socket handle. If a TServerClientWinSocket object for the server endpoint of the new connection is not created in an OnGetSocket event handler, the server socket creates one. The TServerWinSocket object continues to listen for other clients.

4An OnAccept event occurs, using the new TServerClientWinSocket object.

5If ServerType is stThreadBlocking and no thread is available in the cache, an OnGetThread event occurs. If the OnGetThread event handler does not create a thread, the server socket creates a TServerClientThread.

6If ServerType is stThreadBlocking, an OnThreadStart event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket object and an OnClientConnect event occurs.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientConnect event handler for TServerSocket is also set when setting the OnClientConnect event handler of the associated TServerWinSocket.

See Also

OnAccept (see page 59)

OnGetSocket (see page 54)

OnGetThread (see page 54)

OnListen (see page 61)

OnThreadStart (see page 55)

ServerType (see page 49)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

GetClientThread (see page 127)

2.1.11.4.3 TServerSocket.OnClientDisconnect Event

Occurs when one of the connections to a client socket is closed.

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client connection does not close the server socket. The server socket remains open and listening for client requests on its listening connection.

The TServerClientWinSocket that describes the server endpoint of the client connection is freed after OnClientDisconnect. If ServerType is stThreadBlocking, OnThreadEnd occurs after OnClientDisconnect.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientDisconnect event handler for TServerSocket is also set when setting the OnClientDisconnect event handler of the associated TServerWinSocket.

See Also

OnThreadEnd (see page 55)

ServerType (see page 49)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

GetClientThread (see page 127)

2.1.11.4.4 TServerSocket.OnClientError Event

Occurs when there is a failure in establishing, using, or terminating the socket connection to an individual client socket.

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write an OnClientError event handler to respond to errors that arise with a connection to an individual client. Set the ErrorCode

parameter to 0 if the OnClientError event handler successfully deals with the error condition to prevent an ESocketError from being raised.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

Note: The OnClientError event handler for TServerSocket is also set when setting the OnClientError event handler of the associated TServerWinSocket.

See Also

ESocketError (see page 13)

ServerType (see page 49)

TServerClientWinSocket (see page 104)

GetClientThread (see page 127)

2.1.11.4.5 TServerSocket.OnClientRead Event

Occurs when the server socket should read information from a client socket.

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write an OnClientRead event handler to read from the socket connection. If the ServerType property is stThreadBlocking, use a TWinSocketStream object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, use the methods of the Socket parameter to perform the actual reading.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 49)

ReceiveBuf (see page 77)

ReceiveText (see page 78)

TServerClientWinSocket (see page 104)

GetClientThread (see page 127)

TWinSocketStream (see page 138)

2.1.11.4.6 TServerSocket.OnClientWrite Event

Occurs when the server socket should write information to a client socket.

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write an OnClientWrite event handler to write to the socket connection. If the ServerType property is stThreadBlocking, use a TWinSocketStream object to prevent problems that arise while writing from causing the execution thread to hang indefinitely. Otherwise, use the methods of the Socket parameter to perform the actual writing.

Note: If ServerType is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread method of the Sender parameter to access thread-specific information.

See Also

ServerType (see page 49)

SendBuf (see page 78)

SendStream (see page 79)

SendText (see page 80)

GetClientThread (see page 127)

TWinSocketStream (see page 138)

2.1.11.4.7 TServerSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket object to represent the server endpoint of a connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write an OnGetSocket event handler to create a specialized descendant of TServerClientWinSocket for the server socket component to use. Return the new TServerClientWinSocket object in the ClientSocket parameter. The Sender parameter is the TServerWinSocket object that describes the server endpoint of the listening connection.

See Also

TServerClientWinSocket (see page 104)

Create (see page 106)

TServerWinSocket (see page 120)

2.1.11.4.8 TServerSocket.OnGetThread Event

Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write an OnGetThread event handler to create a specialized descendant of TServerClientThread for the connection to the client socket. Return the new TServerClientThread object in the SocketThread parameter.

Most applications that use thread-blocking server components will want to provide an OnGetThread event handler and implement a TServerClientThread that handles its own reading and writing in a thread-safe manner rather than relying on the default TServerClientThread which triggers OnClientRead and OnClientWrite events. This is because the OnClientRead and OnClientWrite event handlers reside on the server socket, which is in global memory.

The Sender parameter is the TServerWinSocket object that describes the server endpoint of the listening connection. The ClientSocket parameter is the TServerClientWinSocket object that describes the server endpoint of the connection to the client socket that is about to be formed.

See Also

ServerType (see page 49)

TServerClientThread (see page 95)

TServerClientWinSocket (see page 104)

TServerWinSocket (see page 120)

2.1.11.4.9 TServerSocket.OnListen Event

Occurs just before a server socket is opened for listening.

Pascal

```
property OnListen: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnListen;
```

Description

Write an OnListen event handler for a server socket to take specific action just before the socket is opened for listening. OnListen occurs after Address and Port have been bound to the socket connection, but before it is opened. This is the last opportunity to make changes to the socket endpoint before it is opened for listening. Use the SocketHandle property of the Socket parameter for Windows API calls that change the socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. The connection to a client socket is completed when a request from the queue is accepted.

See Also

Accept (see page 68)

Address (see page 22)

Port (see page 23)

2.1.11.4.10 TServerSocket.OnThreadEnd Event

Occurs when a client socket connection is terminated and the associated thread finishes execution.

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect event.

The Sender parameter is the TServerWinSocket that describes the server endpoint of the listening connection. The Thread parameter is the thread that is finishing up.

See Also

OnClientDisconnect (see page 52)

TServerWinSocket (see page 120)

2.1.11.4.11 TServerSocket.OnThreadStart Event

Occurs when the execution thread for a connection to a client socket starts up.

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write an OnThreadStart to initialize the TServerClientThread for the connection to a single client socket. The Sender parameter is the TServerWinSocket that describes the server endpoint of the listening connection. The Thread parameter is the thread that is about to start execution.

See Also

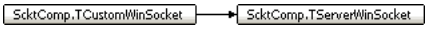
TServerClientThread (see page 95)

TServerWinSocket (see page 120)

2.1.12 TServerWinSocket Class

TServerWinSocket describes the endpoint of a listening server socket connection.

Class Hierarchy



Pascal

```
TServerWinSocket = class(TCustomWinSocket);
```

C++

```
class TServerWinSocket : public TCustomWinSocket;
```

File

ScktComp

Description

TServerWinSocket is used by server socket components to manage the Windows socket API calls for a TCP/IP listening connection. TServerWinSocket introduces new properties, events, and methods to manage multiple socket connections to different client sockets, and to generate or reuse TServerClientThread (see page 95) objects that allow each client connection to have its own execution thread.

See Also

TClientWinSocket (see page 35)

TServerClientThread (see page 95)

TServerClientWinSocket (see page 104)

TServerSocket (see page 108)

2.1.12.1 TServerWinSocket Members

The following tables list the members exposed by TServerWinSocket.

Protected Methods

	Name	Description
	AsyncInitSocket (see page 68)	Looks up information or forms socket connections asynchronously.
	ClientConnect (see page 125)	This is ClientConnect, a member of class TServerWinSocket.
	ClientDisconnect (see page 125)	This is ClientDisconnect, a member of class TServerWinSocket.
	ClientErrorEvent (see page 125)	This is ClientErrorEvent, a member of class TServerWinSocket.
	ClientRead (see page 125)	This is ClientRead, a member of class TServerWinSocket.
	ClientWrite (see page 126)	This is ClientWrite, a member of class TServerWinSocket.
	DoCreateThread (see page 126)	This is DoCreateThread, a member of class TServerWinSocket.
	DoListen (see page 70)	Opens a listening connection.
	DoOpen (see page 71)	Opens a connection to a remote socket.
	Error (see page 71)	Generates an OnErrorEvent (see page 89) event.
	Event (see page 72)	Triggers an OnSocketEvent (see page 90) event.
	GetClientSocket (see page 127)	This is GetClientSocket, a member of class TServerWinSocket.
	GetServerThread (see page 127)	This is GetServerThread, a member of class TServerWinSocket.
	InitSocket (see page 72)	Generates a windows Internet socket address structure from a description of the desired socket port.
	Listen (see page 127)	Opens a listening connection for a server socket.
	SetAsyncStyles (see page 81)	Sets the ASyncStyles (see page 83) property.
	SetServerType (see page 128)	This is SetServerType, a member of class TServerWinSocket.
	SetThreadCacheSize (see page 128)	This is SetThreadCacheSize, a member of class TServerWinSocket.
	ThreadEnd (see page 129)	This is ThreadEnd, a member of class TServerWinSocket.
	ThreadStart (see page 129)	This is ThreadStart, a member of class TServerWinSocket.

Public Events

	Name	Description
	OnClientConnect (see page 133)	Occurs when a client socket completes a connection accepted by the listening socket.
	OnClientDisconnect (see page 134)	Occurs when one of the connections to a client socket is closed.
	OnClientError (see page 135)	Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 135)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 136)	Occurs when the server socket should write information to a client socket.
	OnErrorEvent (see page 89)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetSocket (see page 137)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 104) object to form the connection to a client socket.

	OnGetThread (see page 137)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnSocketEvent (see page 90)	Occurs when the Windows socket receives notification of asynchronous events.
	OnThreadEnd (see page 138)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 138)	Occurs when the execution thread for a connection to a client socket starts up.

Public Methods

	Name	Description
V	Accept (see page 124)	Accepts a connection to a client socket.
	Close (see page 69)	Shuts down the socket connection if it is open.
V	Connect (see page 69)	Provides the interface for a method that responds to connection notifications from client sockets.
	Create (see page 123)	Creates an instance of TServerWinSocket (see page 120) from a Windows socket handle.
V	DefaultHandler (see page 70)	Processes message records.
V	Destroy (see page 123)	Destroys an instance of TServerWinSocket (see page 120).
V	Disconnect (see page 126)	Shuts down the socket connection represented by a Windows socket handle.
	GetClientThread (see page 127)	Returns the execution thread used by the client connection managed by the ClientSocket parameter.
	Lock (see page 74)	Starts a critical section, blocking all other execution threads until the Unlock (see page 81) method is called.
	LookupName (see page 74)	Generates the portion of a Windows Internet socket address structure that describes the host.
	LookupService (see page 75)	Returns the port number associated with a specific service.
	Open (see page 75)	Opens a connection to a remote socket.
V	Read (see page 76)	Generates an OnSocketEvent (see page 90) of type seRead in response to notifications that the socket connection is ready.
	ReceiveBuf (see page 77)	Reads up to Count bytes from the socket connection into the Buf parameter.
	ReceiveLength (see page 77)	Returns the number of bytes ready to be sent over the socket connection.
	ReceiveText (see page 78)	Reads a string from the socket connection.
	SendBuf (see page 78)	Writes Count bytes to the socket connection from the Buf parameter.
	SendStream (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection.
	SendStreamThenDrop (see page 79)	Writes all the information that can be read from the AStream parameter to the socket connection and then terminates the connection.
	SendText (see page 80)	Writes the string S to the socket connection.
	Unlock (see page 81)	Reenables execution in other threads after a critical section begun by the Lock (see page 74) method.
V	Write (see page 81)	Generates an OnSocketEvent (see page 90) of type seWrite in response to notifications that the socket connection is ready.

Public Properties

	Name	Description
R	ActiveConnections (see page 129)	Indicates the number of connections to client sockets accepted by this listening socket that are currently open.
R	ActiveThreads (see page 130)	Indicates the number of TServerClientThread (see page 95) objects currently in use by the socket connections listed in the Connections (see page 130) array.
R	Addr (see page 82)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the socket port.
	ASyncStyles (see page 83)	Determines which asynchronous events the socket can receive.
R	Connected (see page 83)	Indicates whether the socket connection is open and available for communication with other machines.
R	Connections (see page 130)	Lists all TServerClientWinSocket (see page 104) objects that represent open socket connections to client sockets that were accepted by this listening socket.
	Data (see page 84)	Points to application-specific data associated with the socket.
R	Handle (see page 84)	Specifies the handle of the window that receives notifications from the Windows socket.

	IdleThreads (see page 131)	Indicates the number of unused TServerClientThread (see page 95) objects available in the cache.
	LocalAddress (see page 85)	Specifies the IP address of the local system that was bound to the socket connection.
	LocalHost (see page 85)	Specifies the name of the local system.
	LocalPort (see page 86)	Specifies the ID number of the port used by the socket connection.
	LookupState (see page 86)	Indicates any asynchronous activities occurring while opening a connection.
	RemoteAddr (see page 87)	Specifies the Internet socket address structure used in Windows socket API calls to represent the full specification of the remote socket port.
	RemoteAddress (see page 87)	Specifies the IP address of the remote system that was bound to the socket connection.
	RemoteHost (see page 88)	Specifies the name of the remote system.
	RemotePort (see page 88)	Specifies the ID number of the port used by the RemoteHost (see page 88) to form the socket connection.
	ServerType (see page 131)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	SocketHandle (see page 89)	Specifies the Windows handle for the socket.
	ThreadCacheSize (see page 132)	Indicates the number of TServerClientThread (see page 95) objects cached for reuse by new connections to client sockets.

2.1.12.2 TServerWinSocket.Create Constructor

Creates an instance of TServerWinSocket (see page 120) from a Windows socket handle.

Pascal

```
constructor Create(ASocket: TSocket);
```

C++

```
__fastcall TServerWinSocket(TSocket ASocket);
```

Description

Call Create to create a Windows listening socket object. Windows listening socket objects are created automatically by the server socket component that uses them.

Create allocates the helper objects that manage the list of active client connections and the thread cache before calling the inherited constructor. Create also initializes the ASyncStyles (see page 83) property to [asAccept].

See Also

ASyncStyles (see page 83)

Connections (see page 130)

Create (see page 66)

2.1.12.3 TServerWinSocket.Destroy Destructor

Destroys an instance of TServerWinSocket (see page 120).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerWinSocket();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the Windows listening socket object is not nil,

and only then calls Destroy. Unless the Windows socket object is not used by a server socket component, it is freed automatically by the socket component.

Destroy frees the helper objects that manage the list of active client connections and the thread cache.

See Also

System::TObject::Free (see page 3372)

TServerSocket (see page 108)

2.1.12.4 TServerWinSocket Methods

The methods of the TServerWinSocket class are listed here.

Protected Methods

	Name	Description
	ClientConnect (see page 125)	This is ClientConnect, a member of class TServerWinSocket.
	ClientDisconnect (see page 125)	This is ClientDisconnect, a member of class TServerWinSocket.
	ClientErrorEvent (see page 125)	This is ClientErrorEvent, a member of class TServerWinSocket.
	ClientRead (see page 125)	This is ClientRead, a member of class TServerWinSocket.
	ClientWrite (see page 126)	This is ClientWrite, a member of class TServerWinSocket.
	DoCreateThread (see page 126)	This is DoCreateThread, a member of class TServerWinSocket.
	GetClientSocket (see page 127)	This is GetClientSocket, a member of class TServerWinSocket.
	GetServerThread (see page 127)	This is GetServerThread, a member of class TServerWinSocket.
	Listen (see page 127)	Opens a listening connection for a server socket.
	SetServerType (see page 128)	This is SetServerType, a member of class TServerWinSocket.
	SetThreadCacheSize (see page 128)	This is SetThreadCacheSize, a member of class TServerWinSocket.
	ThreadEnd (see page 129)	This is ThreadEnd, a member of class TServerWinSocket.
	ThreadStart (see page 129)	This is ThreadStart, a member of class TServerWinSocket.

Public Methods

	Name	Description
	Accept (see page 124)	Accepts a connection to a client socket.
	Disconnect (see page 126)	Shuts down the socket connection represented by a Windows socket handle.
	GetClientThread (see page 127)	Returns the execution thread used by the client connection managed by the ClientSocket parameter.

2.1.12.4.1 TServerWinSocket.Accept Method

Accepts a connection to a client socket.

Pascal

```
procedure Accept(Socket: TSocket); override;
```

C++

```
virtual __fastcall Accept(TSocket Socket);
```

Description

Accept is called automatically in response to notifications that a client socket is requesting a connection. The Socket parameter is the Windows socket handle for the listening connection.

Accept accepts the client connection request, obtaining the Windows socket handle for the client connection. It then generates an OnGetSocket (see page 137) event, passing this socket handle as an argument to any event handler. If a TServerClientWinSocket (see page 104) is not created by an OnGetSocket (see page 137) event handler, Accept creates a TServerClientWinSocket (see page 104) to represent the server endpoint of the accepted connection. Next, Accept obtains a

running TServerClientThread (see page 95), either by restarting a thread in the cache, obtaining one from an OnGetThread (see page 137) event, or creating a new TServerClientThread (see page 95) object. This new thread handles the connection to the client socket.

Accept is only called if the ASyncStyles (see page 83) property includes asAccept.

See Also

ASyncStyles (see page 83)

OnGetSocket (see page 137)

OnGetThread (see page 137)

2.1.12.4.2 TServerWinSocket.ClientConnect Method

Pascal

```
procedure ClientConnect(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientConnect(TCustomWinSOcket Socket);
```

Description

This is ClientConnect, a member of class TServerWinSocket.

2.1.12.4.3 TServerWinSocket.ClientDisconnect Method

Pascal

```
procedure ClientDisconnect(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientDisconnect(TCustomWinSOcket Socket);
```

Description

This is ClientDisconnect, a member of class TServerWinSocket.

2.1.12.4.4 TServerWinSocket.ClientErrorEvent Method

Pascal

```
procedure ClientErrorEvent(Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode:
Integer); dynamic;
```

C++

```
__fastcall ClientErrorEvent(TCustomWinSocket Socket, TErrorEvent ErrorEvent, int ErrorCode);
```

Description

This is ClientErrorEvent, a member of class TServerWinSocket.

2.1.12.4.5 TServerWinSocket.ClientRead Method

Pascal

```
procedure ClientRead(Socket: TCustomWinSocket); dynamic;
```

C++

```
__fastcall ClientRead(TCustomWinSocket Socket);
```

Description

This is ClientRead, a member of class TServerWinSocket.

2.1.12.4.6 TServerWinSocket.ClientWrite Method

Pascal

```
procedure ClientWrite(Socket: TCustomWinSOcket); dynamic;
```

C++

```
__fastcall ClientWrite(TCustomWinSOcket Socket);
```

Description

This is ClientWrite, a member of class TServerWinSocket.

2.1.12.4.7 TServerWinSocket.Disconnect Method

Shuts down the socket connection represented by a Windows socket handle.

Pascal

```
procedure Disconnect(Socket: TSocket); override;
```

C++

```
virtual __fastcall Disconnect(TSocket Socket);
```

Description

Disconnect is called by the Close (see page 69) method to terminate the socket connection.

Disconnect calls the Lock (see page 74) method to make sure the method is thread-safe. Then, it terminates all open connections to client sockets. Next it generates an OnSocketEvent (see page 90) of type seDisconnect. Finally, it closes the listening connection and calls Unlock (see page 81) to end the critical section started by Lock (see page 74).

See Also

Close (see page 69)

Lock (see page 74)

OnSocketEvent (see page 90)

Unlock (see page 81)

2.1.12.4.8 TServerWinSocket.DoCreateThread Method

Pascal

```
function DoCreateThread(ClientSocket: TServerClientWinSocket): TServerClientThread; virtual;
```

C++

```
virtual __fastcall TServerClientThread DoCreateThread(TServerClientWinSocket ClientSocket);
```

Description

This is DoCreateThread, a member of class TServerWinSocket.

2.1.12.4.9 TServerWinSocket.GetClientSocket Method

Pascal

```
function GetClientSocket(Socket: TSocket): TServerClientWinSocket; dynamic;
```

C++

```
__fastcall TServerClientWinSocket GetClientSocket(TSocket Socket);
```

Description

This is GetClientSocket, a member of class TServerWinSocket.

2.1.12.4.10 TServerWinSocket.GetClientThread Method

Returns the execution thread used by the client connection managed by the ClientSocket parameter.

Pascal

```
function GetClientThread(ClientSocket: TServerClientWinSocket): TServerClientThread;
```

C++

```
__fastcall TServerClientThread GetClientThread(TServerClientWinSocket ClientSocket);
```

Description

Use GetClientThread to obtain access to the TServerClientThread (see page 95) for a particular client connection. GetClientThread allows an OnClient... event handler to access thread-global information saved in the Data (see page 84) property of the thread.

See Also

OnClientConnect (see page 133)

OnClientDisconnect (see page 134)

OnClientError (see page 135)

OnClientRead (see page 135)

OnClientWrite (see page 136)

Data (see page 103)

2.1.12.4.11 TServerWinSocket.GetServerThread Method

Pascal

```
function GetServerThread(ClientSocket: TServerClientWinSocket): TServerClientThread; dynamic;
```

C++

```
__fastcall TServerClientThread GetServerThread(TServerClientWinSocket ClientSocket);
```

Description

This is GetServerThread, a member of class TServerWinSocket.

2.1.12.4.12 TServerWinSocket.Listen Method

Opens a listening connection for a server socket.

Pascal

```
procedure Listen(var Name: string; var Address: string; var Service: string; Port: Word;
QueueSize: Integer);
```

C++

```
__fastcall Listen(UnicodeString Name, UnicodeString Address, UnicodeString Service, Word Port,
int QueueSize);
```

Description

Server socket components call Listen to open a listening connection.

Listen creates the Windows socket that will make the listening connection, and binds the Windows Internet socket address derived from the parameters to the socket. Listen then generates an OnSocketEvent (see page 90) of type seListen. Finally, Listen opens the socket as a listening connection with a queue that can hold QueueSize client requests, setting the Connected (see page 83) property to true.

The Name, Address, Service, and Port parameters are taken from the Host, Address, Service and Port properties of the server socket component. The QueueSize is set to 5 by TServerSocket (see page 108).

See Also

Connected (see page 83)

InitSocket (see page 72)

OnSocketEvent (see page 90)

Address (see page 22)

Host (see page 22)

Port (see page 23)

Service (see page 23)

TServerSocket (see page 108)

2.1.12.4.13 TServerWinSocket.SetServerType Method

Pascal

```
procedure SetServerType(Value: TServerType);
```

C++

```
__fastcall SetServerType(TServerType Value);
```

Description

This is SetServerType, a member of class TServerWinSocket.

2.1.12.4.14 TServerWinSocket.SetThreadCacheSize Method

Pascal

```
procedure SetThreadCacheSize(Value: Integer);
```

C++

```
__fastcall SetThreadCacheSize(int Value);
```

Description

This is SetThreadCacheSize, a member of class TServerWinSocket.

2.1.12.4.15 TServerWinSocket.ThreadEnd Method

Pascal

```
procedure ThreadEnd(AThread: TServerClientThread); dynamic;
```

C++

```
__fastcall ThreadEnd(TServerClientThread AThread);
```

Description

This is ThreadEnd, a member of class TServerWinSocket.

2.1.12.4.16 TServerWinSocket.ThreadStart Method

Pascal

```
procedure ThreadStart(AThread: TServerClientThread); dynamic;
```

C++

```
__fastcall ThreadStart(TServerClientThread AThread);
```

Description

This is ThreadStart, a member of class TServerWinSocket.

2.1.12.5 TServerWinSocket Properties

The properties of the TServerWinSocket class are listed here.

Public Properties

	Name	Description
R	ActiveConnections (see page 129)	Indicates the number of connections to client sockets accepted by this listening socket that are currently open.
R	ActiveThreads (see page 130)	Indicates the number of TServerClientThread (see page 95) objects currently in use by the socket connections listed in the Connections (see page 130) array.
R	Connections (see page 130)	Lists all TServerClientWinSocket (see page 104) objects that represent open socket connections to client sockets that were accepted by this listening socket.
R	IdleThreads (see page 131)	Indicates the number of unused TServerClientThread (see page 95) objects available in the cache.
	ServerType (see page 131)	Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.
	ThreadCacheSize (see page 132)	Indicates the number of TServerClientThread (see page 95) objects cached for reuse by new connections to client sockets.

2.1.12.5.1 TServerWinSocket.ActiveConnections Property

Indicates the number of connections to client sockets accepted by this listening socket that are currently open.

Pascal

```
property ActiveConnections: Integer;
```

C++

```
__property int ActiveConnections;
```

Description

Use ActiveConnections to monitor the use of the server socket. ActiveConnections is the number of entries in the Connections

(see page 130) array.

See Also

Connections (see page 130)

2.1.12.5.2 TServerWinSocket.ActiveThreads Property

Indicates the number of TServerClientThread (see page 95) objects currently in use by the socket connections listed in the Connections (see page 130) array.

Pascal

```
property ActiveThreads: Integer;
```

C++

```
__property int ActiveThreads;
```

Description

Use ActiveThreads to monitor thread usage by the listening socket. If ActiveThreads is usually much higher than ThreadCacheSize (see page 132), increasing ThreadCacheSize (see page 132) should improve performance. To determine whether ThreadCacheSize (see page 132) is too high, use the IdleThreads (see page 131) property.

When the ServerType (see page 131) property is stThreadBlocking, each client connection automatically spawns a thread which is added to the cache and counted by ActiveThreads.

See Also

Connections (see page 130)

GetClientThread (see page 127)

IdleThreads (see page 131)

ServerType (see page 131)

ThreadCacheSize (see page 132)

2.1.12.5.3 TServerWinSocket.Connections Property

Lists all TServerClientWinSocket (see page 104) objects that represent open socket connections to client sockets that were accepted by this listening socket.

Pascal

```
property Connections [Index: Integer]: TCustomWinSocket;
```

C++

```
__property TCustomWinSocket Connections[int Index];
```

Description

Use Connections with the ActiveConnections (see page 129) property to iterate over all open socket connections accepted by the listening socket. Each entry in the Connections array is a TServerClientWinSocket (see page 104) object, where an Index of 0 gives the first TServerClientWinSocket (see page 104), an Index of 1 returns the second TServerClientWinSocket (see page 104), and so on.

If ServerType (see page 131) is stThreadBlocking, the thread associated with each of the TServerClientWinSocket (see page 104) objects in the Connections array can be obtained using the GetClientThread (see page 127) method.

See Also

ActiveConnections (see page 129)

GetClientThread (see page 127)

ServerType (see page 131)

TServerClientWinSocket (see page 104)

2.1.12.5.4 TServerWinSocket.IdleThreads Property

Indicates the number of unused TServerClientThread (see page 95) objects available in the cache.

Pascal

```
property IdleThreads: Integer;
```

C++

```
__property int IdleThreads;
```

Description

Use IdleThreads to monitor thread cache usage by the listening socket. If IdleThreads often very high, the thread cache is tying up a lot of unused memory storing extra threads. Monitor the ActiveThreads (see page 130) property to determine an appropriate value for ThreadCacheSize (see page 132).

When the ServerType (see page 131) property is stThreadBlocking, each client connection automatically spawns a thread which is added to the cache. As the threads in the cache complete their execution, they become available for reuse, and are counted by IdleThreads. When there are more active threads than ThreadCacheSize (see page 132), the extra threads are freed when they finish execution instead.

See Also

ActiveThreads (see page 130)

ServerType (see page 131)

ThreadCacheSize (see page 132)

2.1.12.5.5 TServerWinSocket.ServerType Property

Specifies whether each connection accepted by the server socket is non-blocking, or if it is automatically given a separate execution thread.

Pascal

```
property ServerType: TServerType;
```

C++

```
__property TServerType ServerType;
```

Description

Set ServerType to stThreadBlocking to automatically spawn a new thread for each socket connection accepted by the listening socket connection. When ServerType is stThreadBlocking, the TServerClientThread (see page 95) object for a connection generates OnClientRead (see page 135) or OnClientWrite (see page 136) events when the server socket needs to read or write.

Set ServerType to stNonBlocking to handle all reading and writing over socket connections accepted by the listening socket asynchronously. When ServerType is stNonBlocking, all client connections are handled in a single execution thread by default.

OnClientRead (see page 135) or OnClientWrite (see page 136) events occur when the server socket needs to attend to I/O over one of the connections.

Use a non-blocking socket when reading and writing must be synchronized with client sockets.

Note: When ServerType is stThreadBlocking, it is important that the OnClient...

event handlers contain thread-safe code.

See Also

ASyncStyles (see page 83)

OnClientRead (see page 135)

OnClientWrite (see page 136)

OnGetThread (see page 137)

TServerClientThread (see page 95)

2.1.12.5.6 TServerWinSocket.ThreadCacheSize Property

Indicates the number of TServerClientThread (see page 95) objects cached for reuse by new connections to client sockets.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

When ServerType (see page 131) is stThreadBlocking, each new connection that is accepted by the server socket is given a separate execution thread. In order to improve performance, server sockets store these threads in a cache rather than freeing them when the connection is closed. New (see page 3619) connections can then reuse threads from the cache, rather than requiring the socket server to create a new thread every time a connection is accepted.

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

See Also

ActiveThreads (see page 130)

IdleThreads (see page 131)

ServerType (see page 131)

2.1.12.6 TServerWinSocket Events

The events of the TServerWinSocket class are listed here.

Public Events

	Name	Description
	OnClientConnect (see page 133)	Occurs when a client socket completes a connection accepted by the listening socket.
	OnClientDisconnect (see page 134)	Occurs when one of the connections to a client socket is closed.

	OnClientError (see page 135)	Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.
	OnClientRead (see page 135)	Occurs when the server socket should read information from a client socket.
	OnClientWrite (see page 136)	Occurs when the server socket should write information to a client socket.
	OnGetSocket (see page 137)	Occurs when the server socket needs to create a new TServerClientWinSocket (see page 104) object to form the connection to a client socket.
	OnGetThread (see page 137)	Occurs when the server socket needs to create a new execution thread for a connection to a client socket.
	OnThreadEnd (see page 138)	Occurs when a client socket connection is terminated and the associated thread finishes execution.
	OnThreadStart (see page 138)	Occurs when the execution thread for a connection to a client socket starts up.

2.1.12.6.1 TServerWinSocket.OnClientConnect Event

Occurs when a client socket completes a connection accepted by the listening socket.

Pascal

```
property OnClientConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientConnect;
```

Description

Write (see page 81) an OnClientConnect event handler to take specific action when a client socket completes the socket connection to a TServerClientWinSocket (see page 104) object. Depending on the service, this is the point when the socket should begin reading or writing.

The order of Windows server socket events leading up to OnClientConnect is as follows:

1An OnSocketEvent (see page 90) event of type seListen occurs just before the Windows socket is opened for listening.

2The listening Windows socket receives client requests in a listening queue. The listening socket accepts one of those requests, and receives a Windows socket handle for the new socket connection.

3The listening socket generates an OnGetSocket (see page 137) event, passing in the Windows socket handle. If a TServerClientWinSocket (see page 104) object for the Windows socket handle was not created in the OnGetSocket (see page 137) event handler, the Windows socket creates one. The Windows socket continues to listen for other clients.

4An OnSocketEvent (see page 90) event of type seAccept occurs, using the new TServerClientWinSocket (see page 104) object.

5If ServerType (see page 131) is stThreadBlocking and no thread is available in the cache, an OnGetThread (see page 137) event occurs. If the OnGetThread (see page 137) event handler does not create a thread, the Windows socket object creates a TServerClientThread (see page 95).

6If ServerType (see page 131) is stThreadBlocking, an OnThreadStart (see page 138) event occurs as the thread begins execution.

7The client completes the connection to the TServerClientWinSocket (see page 104) object and an OnClientConnect event occurs.

The Sender parameter is the TServerWinClient object that listens for client requests. The Socket parameter is the TServerClientWinSocket (see page 104) object that forms a connection to the client socket.

Note: If ServerType (see page 131) is stThreadBlocking, make sure that all code in an OnClientConnect event handler is thread-safe. Use the GetClientThread (see page 127) method to access thread-specific information.

Note: The OnClientConnect event handler for TServerWinSocket (see page 120) is also set when setting the

OnClientConnect event handler of the associated TServerSocket (see page 108).

See Also

GetClientThread (see page 127)

OnGetSocket (see page 137)

OnGetThread (see page 137)

OnSocketEvent (see page 90)

OnThreadStart (see page 138)

ServerType (see page 131)

TServerClientWinSocket (see page 104)

OnClientConnect (see page 51)

2.1.12.6.2 TServerWinSocket.OnClientDisconnect Event

Occurs when one of the connections to a client socket is closed.

Pascal

```
property OnClientDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientDisconnect;
```

Description

Write (see page 81) an OnClientDisconnect event handler to take specific action when the connection to a client socket is ending. The termination of a client request does not close the listening socket. The listening connection remains open for accepting new client connection requests.

The TServerClientWinSocket (see page 104) object associated with the client connection is freed after OnClientDisconnect. If ServerType (see page 131) is stThreadBlocking, OnThreadEnd (see page 138) occurs after OnClientDisconnect.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 104) object that is ending a connection with the client socket.

Note: If ServerType (see page 131) is stThreadBlocking, make sure that all code in an OnClientDisconnect event handler is thread-safe.Use the GetClientThread (see page 127) method to access thread-specific information.

The OnClientDisconnect event handler for TServerWinSocket (see page 120) is also set when setting the OnClientDisconnect event handler of the associated TServerSocket (see page 108).

See Also

GetClientThread (see page 127)

OnThreadEnd (see page 138)

ServerType (see page 131)

TServerClientWinSocket (see page 104)

OnClientDisconnect (see page 52)

2.1.12.6.3 TServerWinSocket.OnClientError Event

Occurs when a failure is encountered when establishing, using, or terminating the socket connection to an individual client socket.

Pascal

```
property OnClientError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnClientError;
```

Description

Write (see page 81) an OnClientError event handler to respond to errors that arise with the connection to a client socket. Set the ErrorCode parameter to 0 if the OnClientError event handler successfully deals with the error condition, to prevent an ESocketError (see page 13) from being raised.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 104) object that manages the connection to the client. The ErrorEvent parameter indicates what Socket was attempting to do when the error occurred. The ErrorCode parameter is the error code returned by the Windows socket API call.

Note: If ServerType (see page 131) is stThreadBlocking, make sure that all code in an OnClientError event handler is thread-safe. Use the GetClientThread (see page 127) method to access thread-specific information.

Note: The OnClientError event handler for TServerWinSocket (see page 120) is also set when setting the OnClientError event handler of the associated TServerSocket (see page 108).

See Also

ESocketError (see page 13)

GetClientThread (see page 127)

ServerType (see page 131)

TServerClientWinSocket (see page 104)

OnClientError (see page 52)

2.1.12.6.4 TServerWinSocket.OnClientRead Event

Occurs when the server socket should read information from a client socket.

Pascal

```
property OnClientRead: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientRead;
```

Description

Write (see page 81) an OnClientRead event handler to read from the socket connection. Use a TWinSocketStream (see page 138) object or the methods of the Socket parameter to do the reading.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 104) object that encapsulates the Windows socket connection.

Note: If ServerType (see page 131) is stThreadBlocking, make sure that all code in an OnClientRead event handler is thread-safe. Use the GetClientThread (see page 127) method to access thread-specific information.

Note: The OnClientRead event handler for TServerWinSocket (see page 120) is also set when setting the OnClientRead event handler of the associated TServerSocket (see page 108).

See Also

GetClientThread (see page 127)

ServerType (see page 131)

ReceiveBuf (see page 77)

ReceiveText (see page 78)

TServerClientWinSocket (see page 104)

OnClientRead (see page 53)

TWinSocketStream (see page 138)

2.1.12.6.5 TServerWinSocket.OnClientWrite Event

Occurs when the server socket should write information to a client socket.

Pascal

```
property OnClientWrite: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnClientWrite;
```

Description

Write (see page 81) an OnClientWrite event handler to write to the socket connection. Use a TWinSocketStream (see page 138) object or the methods of the Socket parameter to do the actual writing.

The Sender parameter is the TServerWinClient object that received the client request. The Socket parameter is the TServerClientWinSocket (see page 104) object that encapsulates the Windows socket connection.

Note: If ServerType (see page 131) is stThreadBlocking, make sure that all code in an OnClientWrite event handler is thread-safe. Use the GetClientThread (see page 127) method to access thread-specific information.

Note: The OnClientWrite event handler for TServerWinSocket (see page 120) is also set when setting the OnClientWrite event handler of the associated TServerSocket (see page 108).

See Also

GetClientThread (see page 127)

ServerType (see page 131)

SendBuf (see page 78)

SendStream (see page 79)

SendText (see page 80)

OnClientWrite (see page 53)

2.1.12.6.6 TServerWinSocket.OnGetSocket Event

Occurs when the server socket needs to create a new TServerClientWinSocket (see page 104) object to form the connection to a client socket.

Pascal

```
property OnGetSocket: TGetSocketEvent;
```

C++

```
__property TGetSocketEvent OnGetSocket;
```

Description

Write (see page 81) an OnGetSocket event handler to create a specialized TServerClientWinSocket (see page 104) object to use in the connection represented by the Socket parameter. Return the new TServerClientWinSocket (see page 104) object in the ClientSocket parameter.

The Sender parameter is the TServerWinSocket (see page 120) object that received the client request. The Socket parameter is the Windows socket handle for the socket connection to the client socket.

Note: The OnGetSocket event handler for TServerWinSocket (see page 120) is also set when setting the OnGetSocket event handler of the associated TServerSocket (see page 108).

See Also

Create (see page 106)

OnGetSocket (see page 54)

2.1.12.6.7 TServerWinSocket.OnGetThread Event

Occurs when the server socket needs to create a new execution thread for a connection to a client socket.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

Write (see page 81) an OnGetThread event handler to create a specialized descendant of TServerClientThread (see page 95) for the connection to the client socket. Create (see page 123) the new thread with the CreateSuspended parameter set to false, and return it in the SocketThread parameter. OnGetThread only occurs if there are no idle threads in the cache.

Most applications that use thread-blocking connections will want to create a new descendant of TServerClientThread (see page 95) in an OnGetThread event handler. This is because the default behavior of TServerClientThread (see page 95) uses the OnClientRead (see page 135) and OnClientWrite (see page 136) event handlers for reading and writing. These events occur on the server socket, which is not thread-local.

The Sender parameter is the TServerWinSocket (see page 120) object that received the client request. The ClientSocket parameter is the TServerClientWinSocket (see page 104) object that will communicate with the client socket.

Note: The OnGetThread event handler for TServerWinSocket (see page 120) is also set when setting the OnGetThread event handler of the associated TServerSocket (see page 108).

See Also

IdleThreads (see page 131)

ServerType (see page 131)

OnGetThread (see page 54)

2.1.12.6.8 TServerWinSocket.OnThreadEnd Event

Occurs when a client socket connection is terminated and the associated thread finishes execution.

Pascal

```
property OnThreadEnd: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadEnd;
```

Description

Write (see page 81) an OnThreadEnd to take specific action when the thread for a client connection finishes execution. OnThreadEnd occurs after the OnClientDisconnect (see page 134) event.

The Sender parameter is the TServerWinSocket (see page 120) that received the client request. The Thread parameter is the thread that is finishing up.

Note: The OnThreadEnd event handler for TServerWinSocket (see page 120) is also set when setting the OnThreadEnd event handler of the associated TServerSocket (see page 108).

See Also

OnClientDisconnect (see page 134)

OnThreadEnd (see page 55)

2.1.12.6.9 TServerWinSocket.OnThreadStart Event

Occurs when the execution thread for a connection to a client socket starts up.

Pascal

```
property OnThreadStart: TThreadNotifyEvent;
```

C++

```
__property TThreadNotifyEvent OnThreadStart;
```

Description

Write (see page 81) an OnThreadStart to initialize the TServerClientThread (see page 95) for the connection to a single client socket. The Sender parameter is the TServerWinSocket (see page 120) that received the client request. The Thread parameter is the thread that is about to start execution.

Note: The OnThreadStart event handler for TServerWinSocket (see page 120) is also set when setting the OnThreadStart event handler of the associated TServerSocket (see page 108).

See Also

OnThreadStart (see page 55)

2.1.13 TWinSocketStream Class

TWinSocketStream is a stream that provides services which allow applications to read from or write to socket connections.

Class Hierarchy

Pascal

```
TWinSocketStream = class(TStream);
```

C++

```
class TWinSocketStream : public TStream;
```

File

ScktComp

Description

Use TWinSocketStream to read or write information over a blocking socket connection. Windows socket objects include methods to read from or write to the socket connection they represent. However, these methods do not provide a mechanism for timing out when the socket connection is dropped or for waiting until the socket connection is ready before reading.

When the socket is a non-blocking socket, this lack of a time-out or waiting mechanism is not a problem, because reading and writing occur asynchronously in response to notifications from the socket connection. For blocking sockets, however, these mechanisms provided by TWinSocketStream are necessary so that the application using the socket does not hang indefinitely.

To use a Windows socket stream, create an instance of TWinSocketStream, use the methods of the stream to read or write the data, and then free the Windows socket stream.

Note: TWinSocketStream does not work with non-blocking sockets.

See Also

TClientSocket (see page 24)

TClientWinSocket (see page 35)

TServerClientWinSocket (see page 104)

TServerSocket (see page 108)

2.1.13.1 TWinSocketStream Members

The following tables list the members exposed by TWinSocketStream.

Public Methods

	Name	Description
	Create (see page 140)	Creates an instance of TWinSocketStream (see page 138) for a Windows socket object.
	Destroy (see page 140)	Destroys an instance of TWinSocketStream (see page 138).
	Read (see page 141)	Reads up to Count bytes from the socket connection into Buffer.
	Seek (see page 141)	Always returns 0.
	WaitForData (see page 142)	Waits up to TimeOut (see page 143) milliseconds for the socket connection to be ready to transfer data.
	Write (see page 142)	Writes Count bytes from Buffer to the socket connection.

Public Properties

	Name	Description
	TimeOut (see page 143)	Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

2.1.13.2 TWinSocketStream.Create Constructor

Creates an instance of TWinSocketStream (see page 138) for a Windows socket object.

Pascal

```
constructor Create(ASocket: TCustomWinSocket; TimeOut: Longint);
```

C++

```
__fastcall TWinSocketStream(TCustomWinSocket ASocket, Longint TimeOut);
```

Description

Call Create to obtain an instance of TWinSocketStream (see page 138) for reading from or writing to the socket connection represented by the ASocket parameter. ASocket should be the endpoint of an open blocking socket connection to another Windows socket.

Create associates the stream with the socket connection, and allocates helper objects to allow it to determine when the socket connection is ready for reading or writing. The TimeOut (see page 143) property (the number of milliseconds before a read or write operation aborts) is initialized to the value of the TimeOut (see page 143) parameter.

If ASocket represents a non-blocking socket connection, Create raises an ESocketError (see page 13) exception.

See Also

ESocketError (see page 13)

ClientType (see page 38)

TCustomWinSocket (see page 64)

ServerType (see page 131)

TimeOut (see page 143)

2.1.13.3 TWinSocketStream.Destroy Destructor

Destroys an instance of TWinSocketStream (see page 138).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWinSocketStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TWinSocketStream (see page 138) reference is not nil and only then calls Destroy.

Free the TWinSocketStream (see page 138) object when it is no longer needed for reading from or writing to the socket connection.

Destroy frees the helper objects allocated in the Create (see page 140) method before calling the default destructor.

See Also

Create (see page 140)

System::TObject::Free (see page 3372)

2.1.13.4 TWinSocketStream Methods

The methods of the TWinSocketStream class are listed here.

Public Methods

	Name	Description
	Read (see page 141)	Reads up to Count bytes from the socket connection into Buffer.
	Seek (see page 141)	Always returns 0.
	WaitForData (see page 142)	Waits up to TimeOut (see page 143) milliseconds for the socket connection to be ready to transfer data.
	Write (see page 142)	Writes Count bytes from Buffer to the socket connection.

2.1.13.4.1 TWinSocketStream.Read Method

Reads up to Count bytes from the socket connection into Buffer.

Pascal

```
function Read(var Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( Buffer, Longint Count);
```

Description

Use Read to read data from the socket connection when the number of bytes to be transferred is unknown. Buffer must have at least Count bytes allocated to hold the data that is read from the connection. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

Read may return 0 if the socket connection is extremely slow and the read operation has not completed after TimeOut (see page 143) milliseconds. This ensures that the Read method does not hang indefinitely when a problem occurs with the socket connection.

To guard against the Read method timing out because of a slow connection, set Count fairly low, and make several calls to Read, rather than fewer calls with a large value of Count.

To ensure that the socket connection is ready to send data before calling Read, use the WaitForData (see page 142) method.

Unlike the ReadBuffer method, Read does not raise an exception if Count bytes are not read from the socket connection.

See Also

ReadBuffer

WaitForData (see page 142)

Write (see page 142)

TimeOut (see page 143)

2.1.13.4.2 TWinSocketStream.Seek Method

Always returns 0.

Pascal

```
function Seek(Offset: Longint; Origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint Offset, Word Origin);
```

Description

Seek overrides the inherited method to always set the Position property of the TWinSocketStream (see page 138) object to 0. Socket connections do not support random access. The Position property of a Windows socket stream can only reflect the position at the beginning of all remaining data.

See Also

Position

2.1.13.4.3 TWinSocketStream.WaitForData Method

Waits up to TimeOut (see page 143) milliseconds for the socket connection to be ready to transfer data.

Pascal

```
function WaitForData(Timeout: Longint): Boolean;
```

C++

```
__fastcall Boolean WaitForData(Longint Timeout);
```

Description

Call WaitForData to ensure that the socket connection is ready to read or write information. WaitForData returns true if the socket connection is ready. WaitForData returns false if the socket connection is not ready after TimeOut (see page 143) milliseconds elapse.

Call WaitForData before reading or writing information over the socket connection. Otherwise, Read (see page 141) or Write (see page 142) method calls may time out before any data is transferred.

See Also

Read (see page 141)

Write (see page 142)

2.1.13.4.4 TWinSocketStream.Write Method

Writes Count bytes from Buffer to the socket connection.

Pascal

```
function Write(const Buffer; Count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  Buffer, Longint Count);
```

Description

Use Write to write data from a Buffer to the socket connection. Write returns the number of bytes transferred.

Write may return 0 if the socket connection is extremely slow and the write operation has not completed after TimeOut (see page 143) milliseconds. This ensures that the Write method does not hang indefinitely when a problem occurs with the socket connection.

To guard against the Write method timing out because of a slow connection, set Count fairly low, and make several calls to Write, rather than fewer calls with a large value of Count.

Unlike the WriteBuffer method, Write does not raise an exception if Count bytes are not written. For example, if the connection is

too slow to transfer all the data, Write will return 0, where WriteBuffer will raise an exception.

See Also

Read (see page 141)

WaitForData (see page 142)

WriteBuffer

TimeOut (see page 143)

2.1.13.5 TWinSocketStream Properties

The properties of the TWinSocketStream class are listed here.

Public Properties

	Name	Description
	TimeOut (see page 143)	Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

2.1.13.5.1 TWinSocketStream.TimeOut Property

Specifies the time, in milliseconds, that the socket stream should wait before timing out on a read or write call.

Pascal

```
property TimeOut: Longint;
```

C++

```
__property Longint TimeOut;
```

Description

Set TimeOut to the amount of time that should elapse before a read or write operation aborts under the assumption that the socket connection has failed.

See Also

Read (see page 141)

Write (see page 142)

2.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	SetErrorProc (see page 144)	Replaces the exception handler for error messages that are received from a Windows socket connection.

2.2.1 ScktComp.SetErrorProc Function

Replaces the exception handler for error messages that are received from a Windows socket connection.

Pascal

```
function SetErrorProc(ErrorProc: TSocketErrorProc): TSocketErrorProc;
```

C++

```
TSocketErrorProc SetErrorProc(TSocketErrorProc ErrorProc);
```

File

ScktComp

Description

Call SetErrorProc to assign an exception handler for error messages from Windows socket API calls. By default, socket components have no special error handler and simply raise an ESocketError (see page 13) exception when they receive error messages. If an exception handler is assigned using SetErrorProc, the socket component does not raise this exception.

The ErrorProc (see page 3818) parameter specifies the exception handler. It is passed the Windows socket error code. Inside this exception handler an application can compensate for the error or raise an exception. Passing a value of nil (Delphi) or NULL (C++) for the ErrorProc (see page 3818) parameter restores the default behavior, where the socket component raises an ESocketError (see page 13) exception.

SetErrorProc returns the current exception handler. This value is nil (Delphi) or NULL (C++) when SetErrorProc is called for the first time, and is the value of the ErrorProc (see page 3818) parameter from the previous invocation of SetErrorProc thereafter.

See Also

ESocketError (see page 13)

2.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAsyncStyle (see page 145)	This is record SckttComp.TAsyncStyle.
	TClientType (see page 146)	This is record SckttComp.TClientType.
	TErrorEvent (see page 147)	This is record SckttComp.TErrorEvent.
	TLookupState (see page 148)	This is record SckttComp.TLookupState.
	TServerType (see page 148)	This is record SckttComp.TServerType.
	TSocketEvent (see page 148)	TSocketEvent identifies the operation that a socket is performing.

Records

	Name	Description
	TCMLookupComplete (see page 146)	This is record SckttComp.TCMLookupComplete.
	TCMSocketMessage (see page 147)	This is record SckttComp.TCMSocketMessage.

2.3.1 SckttComp.TAsyncStyle Enumeration

Pascal

```
TAsyncStyle = (
  asRead,
  asWrite,
  asOOB,
  asAccept,
  asConnect,
  asClose
);
```

C++

```
enum TAsyncStyle {
  asRead,
  asWrite,
  asOOB,
  asAccept,
  asConnect,
  asClose
};
```

File

SckttComp

Description

This is record SckttComp.TAsyncStyle.

2.3.2 ScktComp.TClientType Enumeration

Pascal

```
TClientType = (
  ctNonBlocking,
  ctBlocking
);
```

C++

```
enum TClientType {
  ctNonBlocking,
  ctBlocking
};
```

File

ScktComp

Description

This is record ScktComp.TClientType.

2.3.3 ScktComp.TCMLookupComplete Record

Pascal

```
TCMLookupComplete = record
  Msg: Cardinal;
  LookupHandle: THandle;
  AsyncBufLen: Word;
  AsyncError: Word;
  Result: Longint;
end;
```

C++

```
struct TCMLookupComplete {
  unsigned Msg;
  THandle LookupHandle;
  Word AsyncBufLen;
  Word AsyncError;
  Longint Result;
};
```

File

ScktComp

Description

This is record ScktComp.TCMLookupComplete.

2.3.4 SocktComp.TCMSocketMessage Record

Pascal

```
TCMSocketMessage = record
  Msg: Cardinal;
  Socket: TSocket;
  SelectEvent: Word;
  SelectError: Word;
  Result: Longint;
end;
```

C++

```
struct TCMSocketMessage {
  unsigned Msg;
  TSocket Socket;
  Word SelectEvent;
  Word SelectError;
  Longint Result;
};
```

File

SocktComp

Description

This is record SocktComp.TCMSocketMessage.

2.3.5 SocktComp.TErrorEvent Enumeration

Pascal

```
TErrorEvent = (
  eeGeneral,
  eeSend,
  eeReceive,
  eeConnect,
  eeDisconnect,
  eeAccept,
  eeLookup
);
```

C++

```
enum TErrorEvent {
  eeGeneral,
  eeSend,
  eeReceive,
  eeConnect,
  eeDisconnect,
  eeAccept,
  eeLookup
};
```

File

SocktComp

Description

This is record ScktComp.TErrorEvent.

2.3.6 SocktComp.TLookupState Enumeration

Pascal

```
TLookupState = (
  lsIdle,
  lsLookupAddress,
  lsLookupService
);
```

C++

```
enum TLookupState {
  lsIdle,
  lsLookupAddress,
  lsLookupService
};
```

File

ScktComp

Description

This is record ScktComp.TLookupState.

2.3.7 SocktComp.TServerType Enumeration

Pascal

```
TServerType = (
  stNonBlocking,
  stThreadBlocking
);
```

C++

```
enum TServerType {
  stNonBlocking,
  stThreadBlocking
};
```

File

ScktComp

Description

This is record ScktComp.TServerType.

2.3.8 SocktComp.TSocketEvent Enumeration

TSocketEvent identifies the operation that a socket is performing.

Pascal

```
TSocketEvent = (
  seLookup,
  seConnecting,
  seConnect,
  seDisconnect,
  seListen,
  seAccept,
  seWrite,
  seRead
);
```

C++

```
enum TSocketEvent {
  seLookup,
  seConnecting,
  seConnect,
  seDisconnect,
  seListen,
  seAccept,
  seWrite,
  seRead
};
```

File

ScktComp

Description

TSocketEvent is used to identify what a socket is doing when various methods or event handlers are called. It has one of the following values:

Value	Meaning
seLookup	(client sockets only) The Windows socket object is about to locate the server socket. Changing the properties of the client socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will affect the attempted connection.
seConnecting	(client sockets only) The server socket has been located but the connection isn't completed. This is the first opportunity to obtain the actual server port and IP address used for the connection, which may differ from those of the listening socket contacted initially.
seConnect	The socket connection has just been established. This is the point when a socket starts reading or writing over the connection.
seListen	(server sockets only) The Windows socket object has been initialized and is about to open a listening connection. Changing the properties of the server socket has no effect at this point, but changes to the Windows socket object specified by the Socket parameter will the listening connection.
seAccept	(server sockets only) A client connection request has just been accepted. This is the first opportunity to obtain the port and IP address of the individual client connection.
seWrite	The socket is ready for information to be written.
seRead	The socket is receiving information from a socket on the other end of the connection.

See Also

Procedural types

Creating events

2.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAsyncStyles (see page 150)	This is type ScktComp.TAsyncStyles.
TGetSocketEvent (see page 150)	This is type ScktComp.TGetSocketEvent.
TGetThreadEvent (see page 151)	This is type ScktComp.TGetThreadEvent.
TSocketErrorEvent (see page 151)	TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.
TSocketErrorProc (see page 152)	This is type ScktComp.TSocketErrorProc.
TSocketEventEvent (see page 152)	This is type ScktComp.TSocketEventEvent.
TSocketNotifyEvent (see page 152)	The type for event handlers which respond to events during the expected course of forming or using a socket connection.
TThreadNotifyEvent (see page 153)	This is type ScktComp.TThreadNotifyEvent.

2.4.1 ScktComp.TAsyncStyles Type

Pascal

```
TAsyncStyles = set of TAsyncStyle;
```

C++

```
TAsyncStyle TAsyncStyles;
```

File

ScktComp

Description

This is type ScktComp.TAsyncStyles.

2.4.2 ScktComp.TGetSocketEvent Type

Pascal

```
TGetSocketEvent = procedure (Sender: TObject; Socket: TSocket; var ClientSocket:
TServerClientWinSocket) of object;
```

C++

```
(Sender: TObject; Socket: TSocket; var ClientSocket: TServerClientWinSocket) (
TGetSocketEvent)();
```

File

ScktComp

Description

This is type ScktComp.TGetSocketEvent.

2.4.3 SocktComp.TGetThreadEvent Type

Pascal

```
TGetThreadEvent = procedure (Sender: TObject; ClientSocket: TServerClientWinSocket; var
SocketThread: TServerClientThread) of object;
```

C++

```
(Sender: TObject; ClientSocket: TServerClientWinSocket; var SocketThread: TServerClientThread)
( TGetThreadEvent)();
```

File

SocktComp

Description

This is type SocktComp.TGetThreadEvent.

2.4.4 SocktComp.TSocketErrorEvent Type

TSocketErrorEvent is the type for event handlers that respond when a Windows socket reports an error.

Pascal

```
TSocketErrorEvent = procedure (Sender: TObject; Socket: TCustomWinSocket; ErrorEvent:
TErrorEvent; var ErrorCode: Integer) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer) (
TSocketErrorEvent)();
```

File

SocktComp

Description

The Sender parameter is the socket component that encountered the error.

The Socket parameter is the TCustomWinSocket (see page 64) object that receives the error notification.

The ErrorCode parameter is the error code returned by the Windows socket API call. Changing this to 0 within an error handler prevents an exception from being raised.

The ErrorEvent parameter indicates what Socket was attempting to do when the error occurred. It is of type TErrorEvent (see page 147), which can have any of the following values:

Value	Meaning
eeGeneral	The socket received an error message that does not fit into any of the following categories.
eeSend	An error occurred when trying to write to the socket connection.
eeReceive	An error occurred when trying to read from the socket connection.
eeConnect	A connection request that was already accepted could not be completed.
eeDisconnect	An error occurred when trying to close a connection.

eeAccept	A problem occurred when trying to accept a client connection request.

See Also

Procedural types

Creating events

2.4.5 ScktComp.TSocketErrorProc Type

Pascal

```
TSocketErrorProc = procedure (ErrorCode: Integer);
```

C++

```
(ErrorCode: Integer) ( TSocketErrorProc)();
```

File

ScktComp

Description

This is type ScktComp.TSocketErrorProc.

2.4.6 ScktComp.TSocketEventEvent Type

Pascal

```
TSocketEventEvent = procedure (Sender: TObject; Socket: TCustomWinSocket; SocketEvent:
TSocketEvent) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket; SocketEvent: TSocketEvent) ( TSocketEventEvent)();
```

File

ScktComp

Description

This is type ScktComp.TSocketEventEvent.

2.4.7 ScktComp.TSocketNotifyEvent Type

The type for event handlers which respond to events during the expected course of forming or using a socket connection.

Pascal

```
TSocketNotifyEvent = procedure (Sender: TObject; Socket: TCustomWinSocket) of object;
```

C++

```
(Sender: TObject; Socket: TCustomWinSocket) ( TSocketNotifyEvent)();
```

File

SocktComp

Description

The Sender parameter is the socket component or Windows socket object to which the event occurred.

The Socket parameter is the TCustomWinSocket (see page 64) object that represents the endpoint of the connection.

When the event handler belongs to a client Windows socket object, these two parameters are the same. For server Windows socket objects, the Sender parameter represents the listening connection while the Socket parameter represents a client connection.

See Also

TCustomSocket (see page 56)

Procedural types

Creating events

2.4.8 SocktComp.TThreadNotifyEvent Type

Pascal

```
TThreadNotifyEvent = procedure (Sender: TObject; Thread: TServerClientThread) of object;
```

C++

```
(Sender: TObject; Thread: TServerClientThread) ( TThreadNotifyEvent)();
```

File

SocktComp

Description

This is type SocktComp.TThreadNotifyEvent.

2.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_DEFERFREE (see page 154)	This is constant ScktComp.CM_DEFERFREE.
CM_LOOKUPCOMPLETE (see page 154)	This is constant ScktComp.CM_LOOKUPCOMPLETE.
CM_SOCKETMESSAGE (see page 154)	This is constant ScktComp.CM_SOCKETMESSAGE.

2.5.1 ScktComp.CM_DEFERFREE Constant

Pascal

```
CM_DEFERFREE = WM_USER + $0002;
```

C++

```
CM_DEFERFREE = WM_USER + $0002;
```

File

ScktComp

Description

This is constant ScktComp.CM_DEFERFREE.

2.5.2 ScktComp.CM_LOOKUPCOMPLETE Constant

Pascal

```
CM_LOOKUPCOMPLETE = WM_USER + $0003;
```

C++

```
CM_LOOKUPCOMPLETE = WM_USER + $0003;
```

File

ScktComp

Description

This is constant ScktComp.CM_LOOKUPCOMPLETE.

2.5.3 ScktComp.CM_SOCKETMESSAGE Constant

Pascal

```
CM_SOCKETMESSAGE = WM_USER + $0001;
```

C++

```
CM_SOCKETMESSAGE = WM_USER + $0001;
```

File

SckttComp

Description

This is constant SckttComp.CM_SOCKETMESSAGE.

3 ScktMain Namespace

This is namespace ScktMain.

Topics

Name	Description
Classes (see page 158)	The following table lists classes in this documentation.
Types (see page 186)	The following table lists types in this documentation.
Variables (see page 187)	The following table lists variables in this documentation.
Constants (see page 188)	The following table lists constants in this documentation.

Classes

	Name	Description
	TSocketForm (see page 158)	This is class ScktMain.TSocketForm.
	TSocketService (see page 182)	This is class ScktMain.TSocketService.

Constants

Name	Description
UI_INITIALIZE (see page 188)	This is constant ScktMain.UI_INITIALIZE.
WM_MIDASICON (see page 188)	This is constant ScktMain.WM_MIDASICON.

Types

Name	Description
TSocketProc (see page 186)	This is type ScktMain.TSocketProc.

Variables

Name	Description
SocketForm (see page 187)	This is variable ScktMain.SocketForm.
SocketService (see page 187)	This is variable ScktMain.SocketService.

3.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSocketForm (see page 158)	This is class ScktMain.TSocketForm.
	TSocketService (see page 182)	This is class ScktMain.TSocketService.

3.1.1 TSocketForm Class

Class Hierarchy

Pascal

```
TSocketForm = class(TForm);
```

C++

```
class TSocketForm : public TForm;
```

File

ScktMain

Description

This is class ScktMain.TSocketForm.

3.1.1.1 TSocketForm Members

The following tables list the members exposed by TSocketForm.

Protected Methods

	Name	Description
	AddClient (see page 174)	This is AddClient, a member of class TSocketForm.
	AddIcon (see page 174)	This is AddIcon, a member of class TSocketForm.
	ClearModifications (see page 175)	This is ClearModifications, a member of class TSocketForm.
	ReadSettings (see page 178)	This is ReadSettings, a member of class TSocketForm.
	RemoveClient (see page 179)	This is RemoveClient, a member of class TSocketForm.
	UIInitialize (see page 180)	This is UIInitialize, a member of class TSocketForm.
	WMMIDASIcon (see page 180)	This is WMMIDASIcon, a member of class TSocketForm.
V	WndProc (see page 180)	Provides specific message responses for the form.
	WriteSettings (see page 181)	This is WriteSettings, a member of class TSocketForm.

Public Fields

	Name	Description
	ActionList1 (see page 161)	This is ActionList1, a member of class TSocketForm.
	AllowXML (see page 161)	This is AllowXML, a member of class TSocketForm.

	ApplyAction (see page 162)	This is ApplyAction, a member of class TSocketForm.
	ApplyButton (see page 162)	This is ApplyButton, a member of class TSocketForm.
	ConnectionList (see page 162)	This is ConnectionList, a member of class TSocketForm.
	Connections1 (see page 162)	This is Connections1, a member of class TSocketForm.
	DisconnectAction (see page 163)	This is DisconnectAction, a member of class TSocketForm.
	ExportedObjectOnly1 (see page 163)	This is ExportedObjectOnly1, a member of class TSocketForm.
	GUIDDesc (see page 163)	This is GUIDDesc, a member of class TSocketForm.
	HeaderControl1 (see page 163)	This is HeaderControl1, a member of class TSocketForm.
	InterceptGroup (see page 163)	This is InterceptGroup, a member of class TSocketForm.
	InterceptGUID (see page 164)	This is InterceptGUID, a member of class TSocketForm.
	Label1 (see page 164)	This is Label1, a member of class TSocketForm.
	Label4 (see page 164)	This is Label4, a member of class TSocketForm.
	Label5 (see page 164)	This is Label5, a member of class TSocketForm.
	Label7 (see page 165)	This is Label7, a member of class TSocketForm.
	MainMenu1 (see page 165)	This is MainMenu1, a member of class TSocketForm.
	miAdd (see page 165)	This is miAdd, a member of class TSocketForm.
	miClose (see page 165)	This is miClose, a member of class TSocketForm.
	miDisconnect (see page 165)	This is miDisconnect, a member of class TSocketForm.
	miExit (see page 166)	This is miExit, a member of class TSocketForm.
	miPorts (see page 166)	This is miPorts, a member of class TSocketForm.
	miProperties (see page 166)	This is miProperties, a member of class TSocketForm.
	miRemove (see page 166)	This is miRemove, a member of class TSocketForm.
	miShowHostName (see page 167)	This is miShowHostName, a member of class TSocketForm.
	N1 (see page 167)	This is N1, a member of class TSocketForm.
	N2 (see page 167)	This is N2, a member of class TSocketForm.
	N3 (see page 167)	This is N3, a member of class TSocketForm.
	Pages (see page 167)	This is Pages, a member of class TSocketForm.
	Panel1 (see page 168)	This is Panel1, a member of class TSocketForm.
	PopupMenu (see page 168)	This is PopupMenu, a member of class TSocketForm.
	PortDesc (see page 168)	This is PortDesc, a member of class TSocketForm.
	PortGroup (see page 168)	This is PortGroup, a member of class TSocketForm.
	PortList (see page 169)	This is PortList, a member of class TSocketForm.
	PortNo (see page 169)	This is PortNo, a member of class TSocketForm.
	PortUpDown (see page 169)	This is PortUpDown, a member of class TSocketForm.
	PropPage (see page 169)	This is PropPage, a member of class TSocketForm.
	RegisteredAction (see page 169)	This is RegisteredAction, a member of class TSocketForm.
	RemovePortAction (see page 170)	This is RemovePortAction, a member of class TSocketForm.
	ShowHostAction (see page 170)	This is ShowHostAction, a member of class TSocketForm.
	StatPage (see page 170)	This is StatPage, a member of class TSocketForm.
	ThreadDesc (see page 170)	This is ThreadDesc, a member of class TSocketForm.
	ThreadGroup (see page 171)	This is ThreadGroup, a member of class TSocketForm.
	ThreadSize (see page 171)	This is ThreadSize, a member of class TSocketForm.
	ThreadUpDown (see page 171)	This is ThreadUpDown, a member of class TSocketForm.
	Timeout (see page 171)	This is Timeout, a member of class TSocketForm.
	TimeoutDesc (see page 171)	This is TimeoutDesc, a member of class TSocketForm.
	TimeoutGroup (see page 172)	This is TimeoutGroup, a member of class TSocketForm.
	TimeoutUpDown (see page 172)	This is TimeoutUpDown, a member of class TSocketForm.
	UpdateTimer (see page 172)	This is UpdateTimer, a member of class TSocketForm.
	UserStatus (see page 172)	This is UserStatus, a member of class TSocketForm.
	XMLPacket1 (see page 173)	This is XMLPacket1, a member of class TSocketForm.

Public Methods

	Name	Description
	AllowXMLExecute (see page 174)	This is AllowXMLExecute, a member of class TSocketForm.

	ApplyActionExecute (see page 174)	This is ApplyActionExecute, a member of class TSocketForm.
	ApplyActionUpdate (see page 174)	This is ApplyActionUpdate, a member of class TSocketForm.
	ConnectionListColumnClick (see page 175)	This is ConnectionListColumnClick, a member of class TSocketForm.
	ConnectionListCompare (see page 175)	This is ConnectionListCompare, a member of class TSocketForm.
	DisconnectActionUpdate (see page 175)	This is DisconnectActionUpdate, a member of class TSocketForm.
	FormCloseQuery (see page 176)	This is FormCloseQuery, a member of class TSocketForm.
	FormCreate (see page 176)	This is FormCreate, a member of class TSocketForm.
	FormDestroy (see page 176)	This is FormDestroy, a member of class TSocketForm.
	FormShow (see page 176)	This is FormShow, a member of class TSocketForm.
	Initialize (see page 176)	This is Initialize, a member of class TSocketForm.
	IntegerExit (see page 177)	This is IntegerExit, a member of class TSocketForm.
	miAddClick (see page 177)	This is miAddClick, a member of class TSocketForm.
	miCloseClick (see page 177)	This is miCloseClick, a member of class TSocketForm.
	miDisconnectClick (see page 177)	This is miDisconnectClick, a member of class TSocketForm.
	miExitClick (see page 178)	This is miExitClick, a member of class TSocketForm.
	miPropertiesClick (see page 178)	This is miPropertiesClick, a member of class TSocketForm.
	PortListClick (see page 178)	This is PortListClick, a member of class TSocketForm.
	RegisteredActionExecute (see page 178)	This is RegisteredActionExecute, a member of class TSocketForm.
	RemovePortActionExecute (see page 179)	This is RemovePortActionExecute, a member of class TSocketForm.
	RemovePortActionUpdate (see page 179)	This is RemovePortActionUpdate, a member of class TSocketForm.
	ShowHostActionExecute (see page 179)	This is ShowHostActionExecute, a member of class TSocketForm.
	UpdateTimerTimer (see page 180)	This is UpdateTimerTimer, a member of class TSocketForm.
	UpDownClick (see page 180)	This is UpDownClick, a member of class TSocketForm.

Public Properties

	Name	Description
	ItemIndex (see page 181)	This is ItemIndex, a member of class TSocketForm.
R	SelectedSocket (see page 181)	This is SelectedSocket, a member of class TSocketForm.

3.1.1.2 TSocketForm Fields

The fields of the TSocketForm class are listed here.

Public Fields

	Name	Description
	ActionList1 (see page 161)	This is ActionList1, a member of class TSocketForm.
	AllowXML (see page 161)	This is AllowXML, a member of class TSocketForm.
	ApplyAction (see page 162)	This is ApplyAction, a member of class TSocketForm.
	ApplyButton (see page 162)	This is ApplyButton, a member of class TSocketForm.
	ConnectionList (see page 162)	This is ConnectionList, a member of class TSocketForm.
	Connections1 (see page 162)	This is Connections1, a member of class TSocketForm.
	DisconnectAction (see page 163)	This is DisconnectAction, a member of class TSocketForm.
	ExportedObjectOnly1 (see page 163)	This is ExportedObjectOnly1, a member of class TSocketForm.
	GUIDDesc (see page 163)	This is GUIDDesc, a member of class TSocketForm.
	HeaderControl1 (see page 163)	This is HeaderControl1, a member of class TSocketForm.
	InterceptGroup (see page 163)	This is InterceptGroup, a member of class TSocketForm.
	InterceptGUID (see page 164)	This is InterceptGUID, a member of class TSocketForm.
	Label1 (see page 164)	This is Label1, a member of class TSocketForm.
	Label4 (see page 164)	This is Label4, a member of class TSocketForm.
	Label5 (see page 164)	This is Label5, a member of class TSocketForm.
	Label7 (see page 165)	This is Label7, a member of class TSocketForm.
	MainMenu1 (see page 165)	This is MainMenu1, a member of class TSocketForm.
	miAdd (see page 165)	This is miAdd, a member of class TSocketForm.

	miClose (see page 165)	This is miClose, a member of class TSocketForm.
	miDisconnect (see page 165)	This is miDisconnect, a member of class TSocketForm.
	miExit (see page 166)	This is miExit, a member of class TSocketForm.
	miPorts (see page 166)	This is miPorts, a member of class TSocketForm.
	miProperties (see page 166)	This is miProperties, a member of class TSocketForm.
	miRemove (see page 166)	This is miRemove, a member of class TSocketForm.
	miShowHostName (see page 167)	This is miShowHostName, a member of class TSocketForm.
	N1 (see page 167)	This is N1, a member of class TSocketForm.
	N2 (see page 167)	This is N2, a member of class TSocketForm.
	N3 (see page 167)	This is N3, a member of class TSocketForm.
	Pages (see page 167)	This is Pages, a member of class TSocketForm.
	Panel1 (see page 168)	This is Panel1, a member of class TSocketForm.
	PopupMenu (see page 168)	This is PopupMenu, a member of class TSocketForm.
	PortDesc (see page 168)	This is PortDesc, a member of class TSocketForm.
	PortGroup (see page 168)	This is PortGroup, a member of class TSocketForm.
	PortList (see page 169)	This is PortList, a member of class TSocketForm.
	PortNo (see page 169)	This is PortNo, a member of class TSocketForm.
	PortUpDown (see page 169)	This is PortUpDown, a member of class TSocketForm.
	PropPage (see page 169)	This is PropPage, a member of class TSocketForm.
	RegisteredAction (see page 169)	This is RegisteredAction, a member of class TSocketForm.
	RemovePortAction (see page 170)	This is RemovePortAction, a member of class TSocketForm.
	ShowHostAction (see page 170)	This is ShowHostAction, a member of class TSocketForm.
	StatPage (see page 170)	This is StatPage, a member of class TSocketForm.
	ThreadDesc (see page 170)	This is ThreadDesc, a member of class TSocketForm.
	ThreadGroup (see page 171)	This is ThreadGroup, a member of class TSocketForm.
	ThreadSize (see page 171)	This is ThreadSize, a member of class TSocketForm.
	ThreadUpDown (see page 171)	This is ThreadUpDown, a member of class TSocketForm.
	Timeout (see page 171)	This is Timeout, a member of class TSocketForm.
	TimeoutDesc (see page 171)	This is TimeoutDesc, a member of class TSocketForm.
	TimeoutGroup (see page 172)	This is TimeoutGroup, a member of class TSocketForm.
	TimeoutUpDown (see page 172)	This is TimeoutUpDown, a member of class TSocketForm.
	UpdateTimer (see page 172)	This is UpdateTimer, a member of class TSocketForm.
	UserStatus (see page 172)	This is UserStatus, a member of class TSocketForm.
	XMLPacket1 (see page 173)	This is XMLPacket1, a member of class TSocketForm.

3.1.1.2.1 TSocketForm.ActionList1 Field

Pascal

```
ActionList1: TActionList;
```

C++

```
TActionList ActionList1;
```

Description

This is ActionList1, a member of class TSocketForm.

3.1.1.2.2 TSocketForm.AllowXML Field

Pascal

```
AllowXML: TAction;
```

C++

```
TAction AllowXML;
```

Description

This is AllowXML, a member of class TSocketForm.

3.1.1.2.3 TSocketForm.ApplyAction Field

Pascal

```
ApplyAction: TAction;
```

C++

```
TAction ApplyAction;
```

Description

This is ApplyAction, a member of class TSocketForm.

3.1.1.2.4 TSocketForm.ApplyButton Field

Pascal

```
ApplyButton: TButton;
```

C++

```
TButton ApplyButton;
```

Description

This is ApplyButton, a member of class TSocketForm.

3.1.1.2.5 TSocketForm.ConnectionList Field

Pascal

```
ConnectionList: TListView;
```

C++

```
TListView ConnectionList;
```

Description

This is ConnectionList, a member of class TSocketForm.

3.1.1.2.6 TSocketForm.Connections1 Field

Pascal

```
Connections1: TMenuItem;
```

C++

```
TMenuItem Connections1;
```

Description

This is Connections1, a member of class TSocketForm.

3.1.1.2.7 TSocketForm.DisconnectAction Field

Pascal

```
DisconnectAction: TAction;
```

C++

```
TAction DisconnectAction;
```

Description

This is DisconnectAction, a member of class TSocketForm.

3.1.1.2.8 TSocketForm.ExportedObjectOnly1 Field

Pascal

```
ExportedObjectOnly1: TMenuItem;
```

C++

```
TMenuItem ExportedObjectOnly1;
```

Description

This is ExportedObjectOnly1, a member of class TSocketForm.

3.1.1.2.9 TSocketForm.GUIDDesc Field

Pascal

```
GUIDDesc: TLabel;
```

C++

```
TLabel GUIDDesc;
```

Description

This is GUIDDesc, a member of class TSocketForm.

3.1.1.2.10 TSocketForm.HeaderControl1 Field

Pascal

```
HeaderControl1: THeaderControl;
```

C++

```
THeaderControl HeaderControl1;
```

Description

This is HeaderControl1, a member of class TSocketForm.

3.1.1.2.11 TSocketForm.InterceptGroup Field

Pascal

```
InterceptGroup: TGroupBox;
```

C++

```
TGroupBox InterceptGroup;
```

Description

This is InterceptGroup, a member of class TSocketForm.

3.1.1.2.12 TSocketForm.InterceptGUID Field

Pascal

```
InterceptGUID: TEdit;
```

C++

```
TEdit InterceptGUID;
```

Description

This is InterceptGUID, a member of class TSocketForm.

3.1.1.2.13 TSocketForm.Label1 Field

Pascal

```
Label1: TLabel;
```

C++

```
TLabel Label1;
```

Description

This is Label1, a member of class TSocketForm.

3.1.1.2.14 TSocketForm.Label4 Field

Pascal

```
Label4: TLabel;
```

C++

```
TLabel Label4;
```

Description

This is Label4, a member of class TSocketForm.

3.1.1.2.15 TSocketForm.Label5 Field

Pascal

```
Label5: TLabel;
```

C++

```
TLabel Label5;
```

Description

This is Label5, a member of class TSocketForm.

3.1.1.2.16 TSocketForm.Label7 Field

Pascal

```
Label7: TLabel;
```

C++

```
TLabel Label7;
```

Description

This is Label7, a member of class TSocketForm.

3.1.1.2.17 TSocketForm.MainMenu1 Field

Pascal

```
MainMenu1: TMainMenu;
```

C++

```
TMainMenu MainMenu1;
```

Description

This is MainMenu1, a member of class TSocketForm.

3.1.1.2.18 TSocketForm.miAdd Field

Pascal

```
miAdd: TMenuItem;
```

C++

```
TMenuItem miAdd;
```

Description

This is miAdd, a member of class TSocketForm.

3.1.1.2.19 TSocketForm.miClose Field

Pascal

```
miClose: TMenuItem;
```

C++

```
TMenuItem miClose;
```

Description

This is miClose, a member of class TSocketForm.

3.1.1.2.20 TSocketForm.miDisconnect Field

Pascal

```
miDisconnect: TMenuItem;
```

C++

```
TMenuItem miDisconnect;
```

Description

This is miDisconnect, a member of class TSocketForm.

3.1.1.2.21 TSocketForm.miExit Field

Pascal

```
miExit: TMenuItem;
```

C++

```
TMenuItem miExit;
```

Description

This is miExit, a member of class TSocketForm.

3.1.1.2.22 TSocketForm.miPorts Field

Pascal

```
miPorts: TMenuItem;
```

C++

```
TMenuItem miPorts;
```

Description

This is miPorts, a member of class TSocketForm.

3.1.1.2.23 TSocketForm.miProperties Field

Pascal

```
miProperties: TMenuItem;
```

C++

```
TMenuItem miProperties;
```

Description

This is miProperties, a member of class TSocketForm.

3.1.1.2.24 TSocketForm.miRemove Field

Pascal

```
miRemove: TMenuItem;
```

C++

```
TMenuItem miRemove;
```

Description

This is miRemove, a member of class TSocketForm.

3.1.1.2.25 TSocketForm.miShowHostName Field

Pascal

```
miShowHostName: TMenuItem;
```

C++

```
TMenuItem miShowHostName;
```

Description

This is miShowHostName, a member of class TSocketForm.

3.1.1.2.26 TSocketForm.N1 Field

Pascal

```
N1: TMenuItem;
```

C++

```
TMenuItem N1;
```

Description

This is N1, a member of class TSocketForm.

3.1.1.2.27 TSocketForm.N2 Field

Pascal

```
N2: TMenuItem;
```

C++

```
TMenuItem N2;
```

Description

This is N2, a member of class TSocketForm.

3.1.1.2.28 TSocketForm.N3 Field

Pascal

```
N3: TMenuItem;
```

C++

```
TMenuItem N3;
```

Description

This is N3, a member of class TSocketForm.

3.1.1.2.29 TSocketForm.Pages Field

Pascal

```
Pages: TPageControl;
```

C++

```
TPageControl Pages;
```

Description

This is Pages, a member of class TSocketForm.

3.1.1.2.30 TSocketForm.Panel1 Field

Pascal

```
Panel1: TPanel;
```

C++

```
TPanel Panel1;
```

Description

This is Panel1, a member of class TSocketForm.

3.1.1.2.31 TSocketForm.PopupMenu Field

Pascal

```
PopupMenu: TPopupMenu;
```

C++

```
TPopupMenu PopupMenu;
```

Description

This is PopupMenu, a member of class TSocketForm.

3.1.1.2.32 TSocketForm.PortDesc Field

Pascal

```
PortDesc: TLabel;
```

C++

```
TLabel PortDesc;
```

Description

This is PortDesc, a member of class TSocketForm.

3.1.1.2.33 TSocketForm.PortGroup Field

Pascal

```
PortGroup: TGroupBox;
```

C++

```
TGroupBox PortGroup;
```

Description

This is PortGroup, a member of class TSocketForm.

3.1.1.2.34 TSocketForm.PortList Field

Pascal

```
PortList: TListBox;
```

C++

```
TListBox PortList;
```

Description

This is PortList, a member of class TSocketForm.

3.1.1.2.35 TSocketForm.PortNo Field

Pascal

```
PortNo: TEdit;
```

C++

```
TEdit PortNo;
```

Description

This is PortNo, a member of class TSocketForm.

3.1.1.2.36 TSocketForm.PortUpDown Field

Pascal

```
PortUpDown: TUpDown;
```

C++

```
TUpDown PortUpDown;
```

Description

This is PortUpDown, a member of class TSocketForm.

3.1.1.2.37 TSocketForm.PropPage Field

Pascal

```
PropPage: TTabSheet;
```

C++

```
TTabSheet PropPage;
```

Description

This is PropPage, a member of class TSocketForm.

3.1.1.2.38 TSocketForm.RegisteredAction Field

Pascal

```
RegisteredAction: TAction;
```

C++

```
TAction RegisteredAction;
```

Description

This is RegisteredAction, a member of class TSocketForm.

3.1.1.2.39 TSocketForm.RemovePortAction Field

Pascal

```
RemovePortAction: TAction;
```

C++

```
TAction RemovePortAction;
```

Description

This is RemovePortAction, a member of class TSocketForm.

3.1.1.2.40 TSocketForm.ShowHostAction Field

Pascal

```
ShowHostAction: TAction;
```

C++

```
TAction ShowHostAction;
```

Description

This is ShowHostAction, a member of class TSocketForm.

3.1.1.2.41 TSocketForm.StatPage Field

Pascal

```
StatPage: TTabSheet;
```

C++

```
TTabSheet StatPage;
```

Description

This is StatPage, a member of class TSocketForm.

3.1.1.2.42 TSocketForm.ThreadDesc Field

Pascal

```
ThreadDesc: TLabel;
```

C++

```
TLabel ThreadDesc;
```

Description

This is ThreadDesc, a member of class TSocketForm.

3.1.1.2.43 TSocketForm.ThreadGroup Field

Pascal

```
ThreadGroup: TGroupBox;
```

C++

```
TGroupBox ThreadGroup;
```

Description

This is ThreadGroup, a member of class TSocketForm.

3.1.1.2.44 TSocketForm.ThreadSize Field

Pascal

```
ThreadSize: TEdit;
```

C++

```
TEdit ThreadSize;
```

Description

This is ThreadSize, a member of class TSocketForm.

3.1.1.2.45 TSocketForm.ThreadUpDown Field

Pascal

```
ThreadUpDown: TUpDown;
```

C++

```
TUpDown ThreadUpDown;
```

Description

This is ThreadUpDown, a member of class TSocketForm.

3.1.1.2.46 TSocketForm.Timeout Field

Pascal

```
Timeout: TEdit;
```

C++

```
TEdit Timeout;
```

Description

This is Timeout, a member of class TSocketForm.

3.1.1.2.47 TSocketForm.TimeoutDesc Field

Pascal

```
TimeoutDesc: TLabel;
```

C++

```
TLabel TimeoutDesc;
```

Description

This is TimeoutDesc, a member of class TSocketForm.

3.1.1.2.48 TSocketForm.TimeoutGroup Field

Pascal

```
TimeoutGroup: TGroupBox;
```

C++

```
TGroupBox TimeoutGroup;
```

Description

This is TimeoutGroup, a member of class TSocketForm.

3.1.1.2.49 TSocketForm.TimeoutUpDown Field

Pascal

```
TimeoutUpDown: TUpDown;
```

C++

```
TUpDown TimeoutUpDown;
```

Description

This is TimeoutUpDown, a member of class TSocketForm.

3.1.1.2.50 TSocketForm.UpdateTimer Field

Pascal

```
UpdateTimer: TTimer;
```

C++

```
TTimer UpdateTimer;
```

Description

This is UpdateTimer, a member of class TSocketForm.

3.1.1.2.51 TSocketForm.UserStatus Field

Pascal

```
UserStatus: TStatusBar;
```

C++

```
TStatusBar UserStatus;
```

Description

This is UserStatus, a member of class TSocketForm.

3.1.1.2.52 TSocketForm.XMLPacket1 Field

Pascal

```
XMLPacket1: TMenuItem;
```

C++

```
TMenuItem XMLPacket1;
```

Description

This is XMLPacket1, a member of class TSocketForm.

3.1.1.3 TSocketForm Methods

The methods of the TSocketForm class are listed here.

Protected Methods

	Name	Description
	AddClient (see page 174)	This is AddClient, a member of class TSocketForm.
	AddIcon (see page 174)	This is AddIcon, a member of class TSocketForm.
	ClearModifications (see page 175)	This is ClearModifications, a member of class TSocketForm.
	ReadSettings (see page 178)	This is ReadSettings, a member of class TSocketForm.
	RemoveClient (see page 179)	This is RemoveClient, a member of class TSocketForm.
	UIInitialize (see page 180)	This is UIInitialize, a member of class TSocketForm.
	WMMIDASIcon (see page 180)	This is WMMIDASIcon, a member of class TSocketForm.
	WndProc (see page 180)	Provides specific message responses for the form.
	WriteSettings (see page 181)	This is WriteSettings, a member of class TSocketForm.

Public Methods

	Name	Description
	AllowXMLExecute (see page 174)	This is AllowXMLExecute, a member of class TSocketForm.
	ApplyActionExecute (see page 174)	This is ApplyActionExecute, a member of class TSocketForm.
	ApplyActionUpdate (see page 174)	This is ApplyActionUpdate, a member of class TSocketForm.
	ConnectionListColumnClick (see page 175)	This is ConnectionListColumnClick, a member of class TSocketForm.
	ConnectionListCompare (see page 175)	This is ConnectionListCompare, a member of class TSocketForm.
	DisconnectActionUpdate (see page 175)	This is DisconnectActionUpdate, a member of class TSocketForm.
	FormCloseQuery (see page 176)	This is FormCloseQuery, a member of class TSocketForm.
	FormCreate (see page 176)	This is FormCreate, a member of class TSocketForm.
	FormDestroy (see page 176)	This is FormDestroy, a member of class TSocketForm.
	FormShow (see page 176)	This is FormShow, a member of class TSocketForm.
	Initialize (see page 176)	This is Initialize, a member of class TSocketForm.
	IntegerExit (see page 177)	This is IntegerExit, a member of class TSocketForm.
	miAddClick (see page 177)	This is miAddClick, a member of class TSocketForm.
	miCloseClick (see page 177)	This is miCloseClick, a member of class TSocketForm.
	miDisconnectClick (see page 177)	This is miDisconnectClick, a member of class TSocketForm.
	miExitClick (see page 178)	This is miExitClick, a member of class TSocketForm.
	miPropertiesClick (see page 178)	This is miPropertiesClick, a member of class TSocketForm.
	PortListClick (see page 178)	This is PortListClick, a member of class TSocketForm.
	RegisteredActionExecute (see page 178)	This is RegisteredActionExecute, a member of class TSocketForm.
	RemovePortActionExecute (see page 179)	This is RemovePortActionExecute, a member of class TSocketForm.
	RemovePortActionUpdate (see page 179)	This is RemovePortActionUpdate, a member of class TSocketForm.
	ShowHostActionExecute (see page 179)	This is ShowHostActionExecute, a member of class TSocketForm.
	UpdateTimerTimer (see page 180)	This is UpdateTimerTimer, a member of class TSocketForm.

	UpDownClick (see page 180)	This is UpDownClick, a member of class TSocketForm.

3.1.1.3.1 TSocketForm.AddClient Method

Pascal

```
procedure AddClient(Thread: TServerClientThread);
```

C++

```
__fastcall AddClient(TServerClientThread Thread);
```

Description

This is AddClient, a member of class TSocketForm.

3.1.1.3.2 TSocketForm.AddIcon Method

Pascal

```
procedure AddIcon;
```

C++

```
__fastcall AddIcon();
```

Description

This is AddIcon, a member of class TSocketForm.

3.1.1.3.3 TSocketForm.AllowXMLExecute Method

Pascal

```
procedure AllowXMLExecute(Sender: TObject);
```

C++

```
__fastcall AllowXMLExecute(TObject * Sender);
```

Description

This is AllowXMLExecute, a member of class TSocketForm.

3.1.1.3.4 TSocketForm.ApplyActionExecute Method

Pascal

```
procedure ApplyActionExecute(Sender: TObject);
```

C++

```
__fastcall ApplyActionExecute(TObject * Sender);
```

Description

This is ApplyActionExecute, a member of class TSocketForm.

3.1.1.3.5 TSocketForm.ApplyActionUpdate Method

Pascal

```
procedure ApplyActionUpdate(Sender: TObject);
```

C++

```
__fastcall ApplyActionUpdate(TObject * Sender);
```

Description

This is ApplyActionUpdate, a member of class TSocketForm.

3.1.1.3.6 TSocketForm.ClearModifications Method

Pascal

```
procedure ClearModifications;
```

C++

```
__fastcall ClearModifications();
```

Description

This is ClearModifications, a member of class TSocketForm.

3.1.1.3.7 TSocketForm.ConnectionListColumnClick Method

Pascal

```
procedure ConnectionListColumnClick(Sender: TObject; Column: TListColumn);
```

C++

```
__fastcall ConnectionListColumnClick(TObject * Sender, TListColumn Column);
```

Description

This is ConnectionListColumnClick, a member of class TSocketForm.

3.1.1.3.8 TSocketForm.ConnectionListCompare Method

Pascal

```
procedure ConnectionListCompare(Sender: TObject; Item1: TListItem; Item2: TListItem; Data:
Integer; var Compare: Integer);
```

C++

```
__fastcall ConnectionListCompare(TObject * Sender, TListItem Item1, TListItem Item2, int Data,
int Compare);
```

Description

This is ConnectionListCompare, a member of class TSocketForm.

3.1.1.3.9 TSocketForm.DisconnectActionUpdate Method

Pascal

```
procedure DisconnectActionUpdate(Sender: TObject);
```

C++

```
__fastcall DisconnectActionUpdate(TObject * Sender);
```

Description

This is DisconnectActionUpdate, a member of class TSocketForm.

3.1.1.3.10 TSocketForm.FormCloseQuery Method

Pascal

```
procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
```

C++

```
__fastcall FormCloseQuery(TObject * Sender, Boolean CanClose);
```

Description

This is FormCloseQuery, a member of class TSocketForm.

3.1.1.3.11 TSocketForm.FormCreate Method

Pascal

```
procedure FormCreate(Sender: TObject);
```

C++

```
__fastcall FormCreate(TObject * Sender);
```

Description

This is FormCreate, a member of class TSocketForm.

3.1.1.3.12 TSocketForm.FormDestroy Method

Pascal

```
procedure FormDestroy(Sender: TObject);
```

C++

```
__fastcall FormDestroy(TObject * Sender);
```

Description

This is FormDestroy, a member of class TSocketForm.

3.1.1.3.13 TSocketForm.FormShow Method

Pascal

```
procedure FormShow(Sender: TObject);
```

C++

```
__fastcall FormShow(TObject * Sender);
```

Description

This is FormShow, a member of class TSocketForm.

3.1.1.3.14 TSocketForm.Initialize Method

Pascal

```
procedure Initialize(FromService: Boolean);
```

C++

```
__fastcall Initialize(Boolean FromService);
```

Description

This is Initialize, a member of class TSocketForm.

3.1.1.3.15 TSocketForm.IntegerExit Method

Pascal

```
procedure IntegerExit(Sender: TObject);
```

C++

```
__fastcall IntegerExit(TObject * Sender);
```

Description

This is IntegerExit, a member of class TSocketForm.

3.1.1.3.16 TSocketForm.miAddClick Method

Pascal

```
procedure miAddClick(Sender: TObject);
```

C++

```
__fastcall miAddClick(TObject * Sender);
```

Description

This is miAddClick, a member of class TSocketForm.

3.1.1.3.17 TSocketForm.miCloseClick Method

Pascal

```
procedure miCloseClick(Sender: TObject);
```

C++

```
__fastcall miCloseClick(TObject * Sender);
```

Description

This is miCloseClick, a member of class TSocketForm.

3.1.1.3.18 TSocketForm.miDisconnectClick Method

Pascal

```
procedure miDisconnectClick(Sender: TObject);
```

C++

```
__fastcall miDisconnectClick(TObject * Sender);
```

Description

This is miDisconnectClick, a member of class TSocketForm.

3.1.1.3.19 TSocketForm.miExitClick Method

Pascal

```
procedure miExitClick(Sender: TObject);
```

C++

```
__fastcall miExitClick(TObject * Sender);
```

Description

This is miExitClick, a member of class TSocketForm.

3.1.1.3.20 TSocketForm.miPropertiesClick Method

Pascal

```
procedure miPropertiesClick(Sender: TObject);
```

C++

```
__fastcall miPropertiesClick(TObject * Sender);
```

Description

This is miPropertiesClick, a member of class TSocketForm.

3.1.1.3.21 TSocketForm.PortListClick Method

Pascal

```
procedure PortListClick(Sender: TObject);
```

C++

```
__fastcall PortListClick(TObject * Sender);
```

Description

This is PortListClick, a member of class TSocketForm.

3.1.1.3.22 TSocketForm.ReadSettings Method

Pascal

```
procedure ReadSettings;
```

C++

```
__fastcall ReadSettings();
```

Description

This is ReadSettings, a member of class TSocketForm.

3.1.1.3.23 TSocketForm.RegisteredActionExecute Method

Pascal

```
procedure RegisteredActionExecute(Sender: TObject);
```

C++

```
__fastcall RegisteredActionExecute(TObject * Sender);
```

Description

This is RegisteredActionExecute, a member of class TSocketForm.

3.1.1.3.24 TSocketForm.RemoveClient Method

Pascal

```
procedure RemoveClient(Thread: TServerClientThread);
```

C++

```
__fastcall RemoveClient(TServerClientThread Thread);
```

Description

This is RemoveClient, a member of class TSocketForm.

3.1.1.3.25 TSocketForm.RemovePortActionExecute Method

Pascal

```
procedure RemovePortActionExecute(Sender: TObject);
```

C++

```
__fastcall RemovePortActionExecute(TObject * Sender);
```

Description

This is RemovePortActionExecute, a member of class TSocketForm.

3.1.1.3.26 TSocketForm.RemovePortActionUpdate Method

Pascal

```
procedure RemovePortActionUpdate(Sender: TObject);
```

C++

```
__fastcall RemovePortActionUpdate(TObject * Sender);
```

Description

This is RemovePortActionUpdate, a member of class TSocketForm.

3.1.1.3.27 TSocketForm.ShowHostActionExecute Method

Pascal

```
procedure ShowHostActionExecute(Sender: TObject);
```

C++

```
__fastcall ShowHostActionExecute(TObject * Sender);
```

Description

This is ShowHostActionExecute, a member of class TSocketForm.

3.1.1.3.28 TSocketForm.UIInitialize Method

Pascal

```
procedure UIInitialize(var Message: TMessage); message UI_INITIALIZE;
```

C++

```
__fastcall UIInitialize(TMessage Message);
```

Description

This is UIInitialize, a member of class TSocketForm.

3.1.1.3.29 TSocketForm.UpdateTimerTimer Method

Pascal

```
procedure UpdateTimerTimer(Sender: TObject);
```

C++

```
__fastcall UpdateTimerTimer(TObject * Sender);
```

Description

This is UpdateTimerTimer, a member of class TSocketForm.

3.1.1.3.30 TSocketForm.UpDownClick Method

Pascal

```
procedure UpDownClick(Sender: TObject; Button: TUDBtnType);
```

C++

```
__fastcall UpDownClick(TObject * Sender, TUDBtnType Button);
```

Description

This is UpDownClick, a member of class TSocketForm.

3.1.1.3.31 TSocketForm.WMMIDASIcon Method

Pascal

```
procedure WMMIDASIcon(var Message: TMessage); message WM_MIDASICON;
```

C++

```
__fastcall WMMIDASIcon(TMessage Message);
```

Description

This is WMMIDASIcon, a member of class TSocketForm.

3.1.1.3.32 TSocketForm.WndProc Method

Provides specific message responses for the form.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method that receives messages for a form. After calling the inherited method, WndProc performs some window activation, focus, and position message handling necessary to keep in sync with Windows. Override WndProc to change how the form responds to Windows messages.

See Also

WndProc

3.1.1.3.33 TSocketForm.WriteSettings Method

Pascal

```
procedure WriteSettings;
```

C++

```
__fastcall WriteSettings();
```

Description

This is WriteSettings, a member of class TSocketForm.

3.1.1.4 TSocketForm Properties

The properties of the TSocketForm class are listed here.

Public Properties

	Name	Description
	ItemIndex (see page 181)	This is ItemIndex, a member of class TSocketForm.
R	SelectedSocket (see page 181)	This is SelectedSocket, a member of class TSocketForm.

3.1.1.4.1 TSocketForm.ItemIndex Property

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

This is ItemIndex, a member of class TSocketForm.

3.1.1.4.2 TSocketForm.SelectedSocket Property

Pascal

```
property SelectedSocket: TServerSocket;
```

C++

```
__property TServerSocket SelectedSocket;
```

Description

This is SelectedSocket, a member of class TSocketForm.

3.1.2 TSocketService Class

Class Hierarchy

Pascal

```
TSocketService = class(TService);
```

C++

```
class TSocketService : public TService;
```

File

ScktMain

Description

This is class ScktMain.TSocketService.

3.1.2.1 TSocketService Members

The following tables list the members exposed by TSocketService.

Protected Methods

	Name	Description
	Controller (see page 3074)	This is Controller, a member of class TService.
	DoContinue (see page 3074)	This is DoContinue, a member of class TService.
	DoCustomControl (see page 3074)	This is DoCustomControl, a member of class TService.
	DoInterrogate (see page 3074)	This is DoInterrogate, a member of class TService.
	DoPause (see page 3075)	This is DoPause, a member of class TService.
	DoShutdown (see page 3075)	This is DoShutdown, a member of class TService.
	DoStart (see page 3075)	This is DoStart, a member of class TService.
	DoStop (see page 3075)	This is DoStop, a member of class TService.
	Main (see page 3077)	This is Main, a member of class TService.
	Start (see page 184)	This is Start, a member of class TSocketService.
	Stop (see page 185)	This is Stop, a member of class TSocketService.

Public Methods

	Name	Description
	CreateNew (see page 183)	Instantiates and initializes a TService object.
	Destroy (see page 3073)	Disposes of a TService (see page 3071) instance.
	GetServiceController (see page 184)	Returns the service's registered handler function.
	LogMessage (see page 3076)	Sends an error message to the event log.
	ReportStatus (see page 3077)	Updates the Service Control Manager's status information for the calling service.

Public Properties

	Name	Description
	ErrCode (see page 3079)	Contains the service specific error code returned on starting or stopping the service.

	Param (see page 3081)	Lists the parameters to the service.
	ParamCount (see page 3081)	Indicates the number of startup parameters for the service.
	ServiceThread (see page 3082)	Identifies the dedicated thread for the service.
	Status (see page 3083)	Indicates the current state of the service.
	Terminated (see page 3084)	Indicates whether the service thread has terminated.
	Win32ErrCode (see page 3085)	Indicates the Win32 error code reported by the service if a service specific code is not defined.

Published Events

	Name	Description
	AfterInstall (see page 3085)	Occurs immediately after the service is registered with the Windows Service Control manager.
	AfterUninstall (see page 3086)	Occurs immediately after the service is removed from the Service Control manager's database.
	BeforeInstall (see page 3086)	Occurs before the service is first registered with the Windows Service Control manager.
	BeforeUninstall (see page 3087)	Occurs immediately before the service is removed from the Service Control manager's database.
	OnContinue (see page 3087)	Occurs when the Service Control manager resumes the service after it has been paused.
	OnExecute (see page 3088)	Occurs when the thread associated with the service starts up.
	OnPause (see page 3088)	Occurs when the Service Control manager pauses the service temporarily.
	OnShutdown (see page 3088)	Occurs when the system running the service application is about to shut down.
	OnStart (see page 3089)	OnStartup occurs when the service first starts up, before the OnExecute (see page 3088) event.
	OnStop (see page 3089)	Occurs when the Service Control manager stops the service.

Published Properties

	Name	Description
	AllowPause (see page 3078)	Indicates whether clients are allowed to pause the service.
	AllowStop (see page 3078)	Indicates whether clients are allowed to stop the service.
	Dependencies (see page 3078)	Lists the services or load ordering groups on which the service depends.
	DisplayName (see page 3079)	Identifies the service in the Windows Service Control Manager.
	ErrorSeverity (see page 3080)	Indicates the severity of service startup failure.
	Interactive (see page 3080)	Indicates whether the service is interactive with the Windows desktop.
	LoadGroup (see page 3080)	Names of the load ordering group that includes the service.
	Password (see page 3081)	Specifies the password, optionally used depending upon the ServiceStartName (see page 3082), for accessing the service.
	ServiceStartName (see page 3082)	ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.
	ServiceType (see page 3083)	Identifies the type of the service as Win32, device driver, or file system driver.
	StartType (see page 3083)	Indicates when or how the service starts.
	TagID (see page 3084)	Specifies a unique tag for the service within its LoadGroup (see page 3080).
	WaitHint (see page 3084)	Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

3.1.2.2 TSocketService.CreateNew Constructor

Instantiates and initializes a TService object.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TSocketService(TComponent * AOwner, int Dummy = 0);
```

Description

Call CreateNew to instantiate a service object at runtime. Services added at design time are created automatically. The AOwner parameter specifies the TServiceApplication object that is responsible for ultimately freeing the service. The Dummy parameter is not used. Use CreateNew rather than the Create constructor to instantiate a new service object at runtime. CreateNew does not attempt to stream in an associated DFM file that indicates any components added to the service at design time.

See Also

Owner

Create (see page 3064)

TServiceApplication (see page 3089)

3.1.2.3 TSocketService Methods

The methods of the TSocketService class are listed here.

Protected Methods

	Name	Description
	Start (see page 184)	This is Start, a member of class TSocketService.
	Stop (see page 185)	This is Stop, a member of class TSocketService.

Public Methods

	Name	Description
	GetServiceController (see page 184)	Returns the service's registered handler function.

3.1.2.3.1 TSocketService.GetServiceController Method

Returns the service's registered handler function.

Pascal

```
function GetServiceController: TServiceController; override;
```

C++

```
virtual __fastcall TServiceController GetServiceController();
```

Description

GetServiceController returns the handler for the service. You do not need to call this function directly. The service object's main function calls it automatically.

Note: Each service in the service application has a main function for that service. When a request is made to start the service, the main thread of the service calls the RegisterServiceCtrlHandler function to register this control handler function. TService automatically calls RegisterServiceCtrlHandler and returns this handler function. Because this function cannot be a class method (member function), the Service application wizard generates code that creates a non-member handler, (ServiceController) for your service object. This routine returns the service object's Handler (Controller).

3.1.2.3.2 TSocketService.Start Method

Pascal

```
procedure Start(Sender: TService; var Started: Boolean);
```

C++

```
__fastcall Start(TService Sender, Boolean Started);
```

Description

This is Start, a member of class TSocketService.

3.1.2.3.3 TSocketService.Stop Method

Pascal

```
procedure Stop(Sender: TService; var Stopped: Boolean);
```

C++

```
__fastcall Stop(TService Sender, Boolean Stopped);
```

Description

This is Stop, a member of class TSocketService.

3.2 Types

The following table lists types in this documentation.

Types

Name	Description
TSocketProc (see page 186)	This is type SckMain.TSocketProc.

3.2.1 SckMain.TSocketProc Type

Pascal

```
TSocketProc = procedure (Item: TListItem; Socket: TCustomWinSocket) of object;
```

C++

```
(Item: TListItem; Socket: TCustomWinSocket) ( TSocketProc)();
```

File

SckMain

Description

This is type SckMain.TSocketProc.

3.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
SocketForm (see page 187)	This is variable SocketMain.SocketForm.
SocketService (see page 187)	This is variable SocketMain.SocketService.

3.3.1 SocketMain.SocketForm Variable

Pascal

```
SocketForm: TSocketForm;
```

C++

```
TSocketForm SocketForm;
```

File

SocketMain

Description

This is variable SocketMain.SocketForm.

3.3.2 SocketMain.SocketService Variable

Pascal

```
SocketService: TSocketService;
```

C++

```
TSocketService SocketService;
```

File

SocketMain

Description

This is variable SocketMain.SocketService.

3.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
UI_INITIALIZE (see page 188)	This is constant SckttMain.UI_INITIALIZE.
WM_MIDASICON (see page 188)	This is constant SckttMain.WM_MIDASICON.

3.4.1 SckttMain.UI_INITIALIZE Constant

Pascal

```
UI_INITIALIZE = WM_MIDASICON + 1;
```

C++

```
UI_INITIALIZE = WM_MIDASICON + 1;
```

File

SckttMain

Description

This is constant SckttMain.UI_INITIALIZE.

3.4.2 SckttMain.WM_MIDASICON Constant

Pascal

```
WM_MIDASICON = WM_USER + 1;
```

C++

```
WM_MIDASICON = WM_USER + 1;
```

File

SckttMain

Description

This is constant SckttMain.WM_MIDASICON.

4 SConnect Namespace

This is namespace SConnect.

Topics

Name	Description
Classes (see page 191)	The following table lists classes in this documentation.
Interfaces (see page 266)	The following table lists interfaces in this documentation.
Functions (see page 282)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 283)	The following table lists structs, records, enums in this documentation.
Types (see page 284)	The following table lists types in this documentation.
Variables (see page 286)	The following table lists variables in this documentation.
Constants (see page 288)	The following table lists constants in this documentation.

Classes

	Name	Description
	EInterpreterError (see page 191)	EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.
	ESocketConnectionError (see page 192)	ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 221).
	TCustomDataBlockInterpreter (see page 192)	TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.
	TDataBlock (see page 197)	This is class SConnect.TDataBlock.
	TDataBlockInterpreter (see page 207)	TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.
	TDataDispatch (see page 215)	This is class SConnect.TDataDispatch.
	TPacketInterceptFactory (see page 219)	TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.
	TSocketConnection (see page 221)	TSocketConnection uses Windows sockets to manage the connection to an application server.
	TSocketTransport (see page 229)	This is class SConnect.TSocketTransport.
	TStreamedConnection (see page 236)	TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.
	TTransportThread (see page 248)	This is class SConnect.TTransportThread.
	TWebConnection (see page 254)	TWebConnection uses HTTP to manage the connection to an application server.

Constants

Name	Description
asCreateObject (see page 288)	This is constant SConnect.asCreateObject.

asError (see page 288)	This is constant SConnect.asError.
asFreeObject (see page 289)	This is constant SConnect.asFreeObject.
asGetAppServers (see page 289)	This is constant SConnect.asGetAppServers.
asGetGUID (see page 289)	This is constant SConnect.asGetGUID.
asGetID (see page 290)	This is constant SConnect.asGetID.
asGetServers (see page 290)	This is constant SConnect.asGetServers.
asInvoke (see page 290)	This is constant SConnect.asInvoke.
asMask (see page 291)	This is constant SConnect.asMask.
asSoapCommand (see page 291)	This is constant SConnect.asSoapCommand.
CallSig (see page 291)	This is constant SConnect.CallSig.
ResultSig (see page 292)	This is constant SConnect.ResultSig.
THREAD_EXCEPTION (see page 292)	This is constant SConnect.THREAD_EXCEPTION.
THREAD_RECEIVEDSTREAM (see page 292)	This is constant SConnect.THREAD_RECEIVEDSTREAM.
THREAD_REPLACETRANSPORT (see page 293)	This is constant SConnect.THREAD_REPLACETRANSPORT.
THREAD_SENDNOTIFY (see page 293)	This is constant SConnect.THREAD_SENDNOTIFY.
THREAD_SENDSTREAM (see page 293)	This is constant SConnect.THREAD_SENDSTREAM.

Functions

	Name	Description
	GetPacketInterceptorList (see page 282)	This is function SConnect.GetPacketInterceptorList.
	LoadWinSock2 (see page 282)	This is function SConnect.LoadWinSock2.

Interfaces

	Name	Description
	IDataBlock (see page 266)	IDataBlock is the interface used by data block interpreters to access memory buffers.
	IDataIntercept (see page 274)	IDataIntercept is the interface used by TSocketConnection (see page 221) components to transform data that is communicated over the connection.
	ISendDataBlock (see page 276)	ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.
	ITransport (see page 278)	ITransport is the interface used by TStreamedConnection (see page 236) to connect to, read from, and write to a remote application server.

Structs, Records, Enums

	Name	Description
	TVarFlag (see page 283)	This is record SConnect.TVarFlag.

Types

Name	Description
PIntArray (see page 284)	This is type SConnect.PIntArray.
PVariantArray (see page 284)	This is type SConnect.PVariantArray.
TIntArray (see page 285)	This is type SConnect.TIntArray.
TVarFlags (see page 285)	This is type SConnect.TVarFlags.
TVariantArray (see page 285)	This is type SConnect.TVariantArray.

Variables

Name	Description
WSACloseEvent (see page 286)	This is variable SConnect.WSACloseEvent.
WSACreateEvent (see page 286)	This is variable SConnect.WSACreateEvent.
WSAEventSelect (see page 287)	This is variable SConnect.WSAEventSelect.
WSAResetEvent (see page 287)	This is variable SConnect.WSAResetEvent.

4.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EInterpreterError (see page 191)	EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.
	ESocketConnectionError (see page 192)	ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 221).
	TCustomDataBlockInterpreter (see page 192)	TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.
	TDataBlock (see page 197)	This is class SConnect.TDataBlock.
	TDataBlockInterpreter (see page 207)	TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.
	TDataDispatch (see page 215)	This is class SConnect.TDataDispatch.
	TPacketInterceptFactory (see page 219)	TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.
	TSocketConnection (see page 221)	TSocketConnection uses Windows sockets to manage the connection to an application server.
	TSocketTransport (see page 229)	This is class SConnect.TSocketTransport.
	TStreamedConnection (see page 236)	TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.
	TTransportThread (see page 248)	This is class SConnect.TTransportThread.
	TWebConnection (see page 254)	TWebConnection uses HTTP to manage the connection to an application server.

4.1.1 EInterpreterError Class

EInterpreterError is the exception class for errors that arise when trying to interpret COM calls that have been marshaled to a data block.

Class Hierarchy

Pascal

```
EInterpreterError = class(Exception);
```

C++

```
class EInterpreterError : public Exception;
```

File

SConnect

Description

Data block interpreters (TCustomDataBlockInterpreter (see page 192) descendants) raise EInterpreterError when they can't interpret the information in a data block. Streamed connection components such as TSocketConnection (see page 221) use data block interpreters to send interface calls to and receive interface calls from a remote application server.

See Also

TCustomDataBlockInterpreter (see page 192)

TSocketConnection (see page 221)

TSoapConnection (see page 1188)

TWebConnection (see page 254)

4.1.2 ESocketConnectionError Class

ESocketConnectionError is the exception class for errors that arise when sending or receiving messages using TSocketConnection (see page 221).

Class Hierarchy

Pascal

```
ESocketConnectionError = class(Exception);
```

C++

```
class ESocketConnectionError : public Exception;
```

File

SConnect

Description

Socket connection components raise ESocketConnectionError when the socket they use to send and receive messages reports an error, or when the information read from the socket is incomplete.

See Also

TSocketConnection (see page 221)

4.1.3 TCustomDataBlockInterpreter Class

TCustomDataBlockInterpreter marshals COM interface calls for streamed connection components.

Class Hierarchy

Pascal

```
TCustomDataBlockInterpreter = class;
```

C++

```
class TCustomDataBlockInterpreter;
```

File

SConnect

Description

TCustomDataBlockInterpreter introduces the methods needed for marshaling the messages that pass between a client application and an application server in a multi-tiered database application.

Streamed connection components use data block interpreters to marshal of messages sent to and received from an application server. An object that supports the IDataBlock (see page 266) interface stores the messages themselves.

See Also

TStreamedConnection (see page 236)

IDataBlock (see page 266)

4.1.3.1 TCustomDataBlockInterpreter Members

The following tables list the members exposed by TCustomDataBlockInterpreter.

Protected Methods

	Name	Description
	AddDispatch (see page 194)	This is AddDispatch, a member of class TCustomDataBlockInterpreter.
	CallCreateObject (see page 194)	This is CallCreateObject, a member of class TCustomDataBlockInterpreter.
	CallFreeObject (see page 194)	This is CallFreeObject, a member of class TCustomDataBlockInterpreter.
	CallGetIDsOfNames (see page 194)	This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.
	CallGetServerList (see page 195)	This is CallGetServerList, a member of class TCustomDataBlockInterpreter.
	CallInvoke (see page 195)	This is CallInvoke, a member of class TCustomDataBlockInterpreter.
	CanCreateObject (see page 195)	This is CanCreateObject, a member of class TCustomDataBlockInterpreter.
	CreateObject (see page 195)	This is CreateObject, a member of class TCustomDataBlockInterpreter.
	InternalCreateObject (see page 195)	This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.
	LockObject (see page 196)	This is LockObject, a member of class TCustomDataBlockInterpreter.
	ReleaseObject (see page 196)	This is ReleaseObject, a member of class TCustomDataBlockInterpreter.
	RemoveDispatch (see page 196)	This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.
	StoreObject (see page 197)	This is StoreObject, a member of class TCustomDataBlockInterpreter.
	UnlockObject (see page 197)	This is UnlockObject, a member of class TCustomDataBlockInterpreter.

Public Methods

	Name	Description
	InterpretData (see page 196)	This is InterpretData, a member of class TCustomDataBlockInterpreter.

4.1.3.2 TCustomDataBlockInterpreter Methods

The methods of the TCustomDataBlockInterpreter class are listed here.

Protected Methods

	Name	Description
	AddDispatch (see page 194)	This is AddDispatch, a member of class TCustomDataBlockInterpreter.
	CallCreateObject (see page 194)	This is CallCreateObject, a member of class TCustomDataBlockInterpreter.
	CallFreeObject (see page 194)	This is CallFreeObject, a member of class TCustomDataBlockInterpreter.
	CallGetIDsOfNames (see page 194)	This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.
	CallGetServerList (see page 195)	This is CallGetServerList, a member of class TCustomDataBlockInterpreter.
	CallInvoke (see page 195)	This is CallInvoke, a member of class TCustomDataBlockInterpreter.
	CanCreateObject (see page 195)	This is CanCreateObject, a member of class TCustomDataBlockInterpreter.
	CreateObject (see page 195)	This is CreateObject, a member of class TCustomDataBlockInterpreter.
	InternalCreateObject (see page 195)	This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.
	LockObject (see page 196)	This is LockObject, a member of class TCustomDataBlockInterpreter.
	ReleaseObject (see page 196)	This is ReleaseObject, a member of class TCustomDataBlockInterpreter.
	RemoveDispatch (see page 196)	This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.
	StoreObject (see page 197)	This is StoreObject, a member of class TCustomDataBlockInterpreter.

	UnlockObject (see page 197)	This is UnlockObject, a member of class TCustomDataBlockInterpreter.

Public Methods

	Name	Description
	InterpretData (see page 196)	This is InterpretData, a member of class TCustomDataBlockInterpreter.

4.1.3.2.1 TCustomDataBlockInterpreter.AddDispatch Method

Pascal

```
procedure AddDispatch(Value: TDataDispatch); virtual; abstract;
```

C++

```
virtual __fastcall AddDispatch(TDataDispatch Value) = 0;
```

Description

This is AddDispatch, a member of class TCustomDataBlockInterpreter.

4.1.3.2.2 TCustomDataBlockInterpreter.CallCreateObject Method

Pascal

```
function CallCreateObject(Name: string): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CallCreateObject(UnicodeString Name) = 0;
```

Description

This is CallCreateObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.3 TCustomDataBlockInterpreter.CallFreeObject Method

Pascal

```
procedure CallFreeObject(DispatchIndex: Integer); virtual; abstract;
```

C++

```
virtual __fastcall CallFreeObject(int DispatchIndex) = 0;
```

Description

This is CallFreeObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.4 TCustomDataBlockInterpreter.CallGetIDsOfNames Method

Pascal

```
function CallGetIDsOfNames(DispatchIndex: Integer; const IID: TGUID; Names: Pointer;
NameCount: Integer; LocaleID: Integer; DispIDs: Pointer): HResult; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT CallGetIDsOfNames(int DispatchIndex, const TGUID IID,
void * Names, int NameCount, int LocaleID, void * DispIDs) = 0;
```

Description

This is CallGetIDsOfNames, a member of class TCustomDataBlockInterpreter.

4.1.3.2.5 TCustomDataBlockInterpreter.CallGetServerList Method

Pascal

```
function CallGetServerList: OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CallGetServerList() = 0;
```

Description

This is CallGetServerList, a member of class TCustomDataBlockInterpreter.

4.1.3.2.6 TCustomDataBlockInterpreter.CallInvoke Method

Pascal

```
function CallInvoke(DispatchIndex: Integer; DispID: Integer; const IID: TGUID; LocaleID:
Integer; Flags: Word; var Params; VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer):
HResult; virtual; abstract; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT CallInvoke(int DispatchIndex, int DispID, const TGUID
IID, int LocaleID, Word Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr) = 0;
```

Description

This is CallInvoke, a member of class TCustomDataBlockInterpreter.

4.1.3.2.7 TCustomDataBlockInterpreter.CanCreateObject Method

Pascal

```
function CanCreateObject(const ClassID: TGUID): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean CanCreateObject(const TGUID ClassID) = 0;
```

Description

This is CanCreateObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.8 TCustomDataBlockInterpreter.CreateObject Method

Pascal

```
function CreateObject(const Name: string): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant CreateObject(const UnicodeString Name) = 0;
```

Description

This is CreateObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.9 TCustomDataBlockInterpreter.InternalCreateObject Method

Pascal

```
function InternalCreateObject(const ClassID: TGUID): OleVariant; virtual; abstract;
```

C++

```
virtual __fastcall OleVariant InternalCreateObject(const TGUID ClassID) = 0;
```

Description

This is InternalCreateObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.10 TCustomDataBlockInterpreter.InterpretData Method

Pascal

```
procedure InterpretData(const Data: IDataBlock); virtual; abstract;
```

C++

```
virtual __fastcall InterpretData(const IDataBlock Data) = 0;
```

Description

This is InterpretData, a member of class TCustomDataBlockInterpreter.

4.1.3.2.11 TCustomDataBlockInterpreter.LockObject Method

Pascal

```
function LockObject(ID: Integer): IDispatch; virtual; abstract;
```

C++

```
virtual __fastcall IDispatch LockObject(int ID) = 0;
```

Description

This is LockObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.12 TCustomDataBlockInterpreter.ReleaseObject Method

Pascal

```
procedure ReleaseObject(ID: Integer); virtual; abstract;
```

C++

```
virtual __fastcall ReleaseObject(int ID) = 0;
```

Description

This is ReleaseObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.13 TCustomDataBlockInterpreter.RemoveDispatch Method

Pascal

```
procedure RemoveDispatch(Value: TDataDispatch); virtual; abstract;
```

C++

```
virtual __fastcall RemoveDispatch(TDataDispatch Value) = 0;
```

Description

This is RemoveDispatch, a member of class TCustomDataBlockInterpreter.

4.1.3.2.14 TCustomDataBlockInterpreter.StoreObject Method

Pascal

```
function StoreObject(const Value: OleVariant): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int StoreObject(const OleVariant Value) = 0;
```

Description

This is StoreObject, a member of class TCustomDataBlockInterpreter.

4.1.3.2.15 TCustomDataBlockInterpreter.UnlockObject Method

Pascal

```
procedure UnlockObject(ID: Integer; const Disp: IDispatch); virtual; abstract;
```

C++

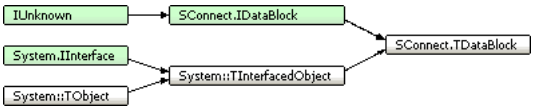
```
virtual __fastcall UnlockObject(int ID, const IDispatch Disp) = 0;
```

Description

This is UnlockObject, a member of class TCustomDataBlockInterpreter.

4.1.4 TDataBlock Class

Class Hierarchy



Pascal

```
TDataBlock = class(TInterfacedObject, IDataBlock);
```

C++

```
class TDataBlock : public TInterfacedObject, public IDataBlock;
```

File

SConnect

Description

This is class SConnect.TDataBlock.

4.1.4.1 TDataBlock Members

The following tables list the members exposed by TDataBlock.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	Clear (see page 200)	Removes all data from Memory except for the reserved bytes.
	GetBytesReserved (see page 201)	Returns the value of the BytesReserved property.
	GetMemory (see page 201)	Returns the value of the Memory property.
	GetSignature (see page 201)	Returns the value of the Signature property.
	GetSize (see page 202)	Returns the value of the Size property.
	GetStream (see page 202)	Returns the value of the Stream property.
	IgnoreStream (see page 202)	Informs the data block when its stream is invalidated.
	InitData (see page 202)	Adds data to the data block.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	Read (see page 203)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 203)	Sets the value of the Signature property.
	SetSize (see page 204)	Sets the value of the Size property.
	Write (see page 204)	Writes a specified number of bytes from a buffer.

Protected Properties

	Name	Description
R	BytesReserved (see page 205)	Indicates the number of bytes from Memory that are reserved for internal use.
R	Memory (see page 205)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 206)	Indicates the contents of the data block.
	Size (see page 206)	Indicates the size of the data in the memory buffer.
R	Stream (see page 206)	Provides access to an internal stream that contains the data block's memory.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 199)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 199)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.

	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

4.1.4.2 TDataBlock.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TDataBlock();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

4.1.4.3 TDataBlock.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataBlock();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

4.1.4.4 TDataBlock Methods

The methods of the TDataBlock class are listed here.

Protected Methods

	Name	Description
	Clear (see page 200)	Removes all data from Memory except for the reserved bytes.
	GetBytesReserved (see page 201)	Returns the value of the BytesReserved property.
	GetMemory (see page 201)	Returns the value of the Memory property.
	GetSignature (see page 201)	Returns the value of the Signature property.
	GetSize (see page 202)	Returns the value of the Size property.
	GetStream (see page 202)	Returns the value of the Stream property.
	IgnoreStream (see page 202)	Informs the data block when its stream is invalidated.
	InitData (see page 202)	Adds data to the data block.
	Read (see page 203)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 203)	Sets the value of the Signature property.
	SetSize (see page 204)	Sets the value of the Size property.
	Write (see page 204)	Writes a specified number of bytes from a buffer.

4.1.4.4.1 TDataBlock.Clear Method

Removes all data from Memory except for the reserved bytes.

Pascal

```
procedure Clear; stdcall;
```

C++

```
__fastcall __stdcall Clear();
```

Description

Call Clear when finished reading from Memory or when the information written to Memory has been used. Clear reclaims the

resources used to store information that is read from or written to the data block.

See Also

BytesReserved (see page 272)

Memory (see page 272)

4.1.4.4.2 TDataBlock.GetBytesReserved Method

Returns the value of the BytesReserved property.

Pascal

```
function GetBytesReserved: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetBytesReserved();
```

Description

GetBytesReserved is the read implementer for the BytesReserved property.

See Also

BytesReserved (see page 272)

4.1.4.4.3 TDataBlock.GetMemory Method

Returns the value of the Memory property.

Pascal

```
function GetMemory: Pointer; stdcall;
```

C++

```
__fastcall __stdcall void * GetMemory();
```

Description

GetMemory is the read implementer for the Memory property.

See Also

Memory (see page 272)

4.1.4.4.4 TDataBlock.GetSignature Method

Returns the value of the Signature property.

Pascal

```
function GetSignature: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSignature();
```

Description

GetSignature is the read implementer for the Signature property.

See Also

Signature (see page 273)

4.1.4.4.5 TDataBlock.GetSize Method

Returns the value of the Size property.

Pascal

```
function GetSize: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSize();
```

Description

GetSize is the read implementer for the Size property.

See Also

Size (see page 273)

4.1.4.4.6 TDataBlock.GetStream Method

Returns the value of the Stream property.

Pascal

```
function GetStream: TStream; stdcall;
```

C++

```
__fastcall __stdcall TStream GetStream();
```

Description

GetStream is the read implementer for the Stream property.

See Also

Stream (see page 274)

4.1.4.4.7 TDataBlock.IgnoreStream Method

Informs the data block when its stream is invalidated.

Pascal

```
procedure IgnoreStream; stdcall;
```

C++

```
__fastcall __stdcall IgnoreStream();
```

Description

Call IgnoreStream after taking responsibility for freeing the component specified by Stream. After a call to IgnoreStream, the data block will not free the memory for its internal Stream component.

See Also

Stream (see page 274)

4.1.4.4.8 TDataBlock.InitData Method

Adds data to the data block.

Pascal

```
function InitData(Data: Pointer; DataLen: Integer; CheckLen: Boolean): Integer; stdcall;
```

C++

```
__fastcall __stdcall int InitData(void * Data, int DataLen, Boolean CheckLen);
```

Description

Data is the data to be added to the data block. It must include a valid signature or InitData raises an exception.

DataLen is the number of bytes referenced by Data.

CheckLen specifies whether InitData should verify that the signature indicates the same amount of data as is indicated by the DataLen parameter.

See Also

Signature (see page 273)

4.1.4.4.9 TDataBlock.Read Method

Reads a specified number of bytes into a buffer.

Pascal

```
function Read(var Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Read( Buffer, int Count);
```

Description

Call Read to copy the contents stored in the memory buffer to the buffer supplied by the Buffer parameter. Read reads up to Count bytes and returns the number of bytes actually read.

The first call to Read begins reading from Memory at the first position after the reserved bytes. Subsequent calls begin reading at the point where the last call to Read left off.

Note: The Count parameter should specify a value less than or equal to Size - BytesReserved.

See Also

Write (see page 271)

Memory (see page 272)

Size (see page 273)

BytesReserved (see page 272)

4.1.4.4.10 TDataBlock.SetSignature Method

Sets the value of the Signature property.

Pascal

```
procedure SetSignature(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSignature(int Value);
```

Description

SetSignature is the write implementer for the Signature property.

See Also

Signature (see page 273)

4.1.4.4.11 TDataBlock.SetSize Method

Sets the value of the Size property.

Pascal

```
procedure SetSize(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSize(int Value);
```

Description

SetSize is the write implementer for the Size property.

See Also

Size (see page 273)

4.1.4.4.12 TDataBlock.Write Method

Writes a specified number of bytes from a buffer.

Pascal

```
function Write(const Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Write(const  Buffer, int Count);
```

Description

Call Write to copy the contents of the buffer supplied as the Buffer parameter to the next position in Memory. Write writes up to Count bytes and returns the number of bytes actually written.

The first call to Write begins writing to Memory at the first position after the reserved bytes. Subsequent calls begin writing at the point where the last call to Write left off.

Note: The Count parameter should specify a value less than or equal to Size - BytesReserved.

See Also

Read (see page 270)

Memory (see page 272)

Size (see page 273)

BytesReserved (see page 272)

4.1.4.5 TDataBlock Properties

The properties of the TDataBlock class are listed here.

Protected Properties

	Name	Description
	BytesReserved (see page 205)	Indicates the number of bytes from Memory that are reserved for internal use.
	Memory (see page 205)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 206)	Indicates the contents of the data block.
	Size (see page 206)	Indicates the size of the data in the memory buffer.
	Stream (see page 206)	Provides access to an internal stream that contains the data block's memory.

4.1.4.5.1 TDataBlock.BytesReserved Property

Indicates the number of bytes from Memory that are reserved for internal use.

Pascal

```
property BytesReserved: Integer;
```

C++

```
__property int BytesReserved;
```

Description

BytesReserved indicates how many bytes of Memory can't be used for reading and writing. When interpreting the value of the Size property, use BytesReserved to determine how much buffer space is available for reading and writing data.

See Also

Memory (see page 272)

Size (see page 273)

4.1.4.5.2 TDataBlock.Memory Property

Points to a buffer from which data is read or to which data is written.

Pascal

```
property Memory: Pointer;
```

C++

```
__property void * Memory;
```

Description

Memory provides access to the buffer that contains data that is read or written. Memory includes a reserved area (of size BytesReserved) for internal information such as the value of Signature,

Note: When implementing the Receive and Send methods of the ITransport interface, component writers should read directly from and write directly to Memory, rather than using the Read and Write methods. Otherwise, the information in the reserved area will not be sent to or from the application server.

See Also

Size (see page 273)

ITransport (see page 278)

BytesReserved (see page 272)

Signature (see page 273)

Stream (see page 274)

4.1.4.5.3 TDataBlock.Signature Property

Indicates the contents of the data block.

Pascal

```
property Signature: Integer;
```

C++

```
__property int Signature;
```

Description

Write Signature values to indicate what type of data is written to the data block. Read Signature to determine what type of data can be read from the data block.

The value of signature is stored somewhere in the first BytesReserved bytes of Memory.

See Also

BytesReserved (see page 272)

Memory (see page 272)

4.1.4.5.4 TDataBlock.Size Property

Indicates the size of the data in the memory buffer.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is the number of bytes available in Memory, excluding the reserved bytes. Set Size to ensure that Memory is large enough to contain the data that the data block will read or write.

See Also

BytesReserved (see page 272)

Memory (see page 272)

Signature (see page 273)

4.1.4.5.5 TDataBlock.Stream Property

Provides access to an internal stream that contains the data block's memory.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Stream allows component writers more convenient access to the data block's memory buffer. This memory buffer contains the data that is read or written.

See Also

Memory (see page 272)

IgnoreStream (see page 269)

TMemoryStream

4.1.5 TDataBlockInterpreter Class

TDataBlockInterpreter marshals COM interface calls for non COM-based connection components.

Class Hierarchy

Pascal

```
TDataBlockInterpreter = class(TCustomDataBlockInterpreter);
```

C++

```
class TDataBlockInterpreter : public TCustomDataBlockInterpreter;
```

File

SConnect

Description

Many streamed connection components use TDataBlockInterpreter to handle the marshaling of messages sent to and received from an application server. An object that supports the IDataBlock (see page 266) interface stores the messages themselves.

See Also

TStreamedConnection (see page 236)

IDataBlock (see page 266)

TSocketConnection (see page 221)

TWebConnection (see page 254)

4.1.5.1 TDataBlockInterpreter Members

The following tables list the members exposed by TDataBlockInterpreter.

Protected Methods

	Name	Description
	AddDispatch (see page 209)	This is AddDispatch, a member of class TDataBlockInterpreter.
	CallFreeObject (see page 210)	This is CallFreeObject, a member of class TDataBlockInterpreter.
	CallGetIDsOfNames (see page 210)	This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.
	CallGetServerList (see page 210)	This is CallGetServerList, a member of class TDataBlockInterpreter.
	CallInvoke (see page 211)	This is CallInvoke, a member of class TDataBlockInterpreter.
	CanCreateObject (see page 211)	This is CanCreateObject, a member of class TDataBlockInterpreter.
	CreateObject (see page 211)	This is CreateObject, a member of class TDataBlockInterpreter.
	DoCreateObject (see page 211)	This is DoCreateObject, a member of class TDataBlockInterpreter.
	DoCustomAction (see page 211)	This is DoCustomAction, a member of class TDataBlockInterpreter.
	DoFreeObject (see page 212)	This is DoFreeObject, a member of class TDataBlockInterpreter.

	DoGetAppServerList (see page 212)	This is DoGetAppServerList, a member of class TDataBlockInterpreter.
	DoGetIDsOfNames (see page 212)	This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.
	DoGetServerList (see page 212)	This is DoGetServerList, a member of class TDataBlockInterpreter.
	DoInvoke (see page 213)	This is DoInvoke, a member of class TDataBlockInterpreter.
V	InternalCreateObject (see page 213)	This is InternalCreateObject, a member of class TDataBlockInterpreter.
V	LockObject (see page 213)	This is LockObject, a member of class TDataBlockInterpreter.
V	ReleaseObject (see page 214)	This is ReleaseObject, a member of class TDataBlockInterpreter.
V	RemoveDispatch (see page 214)	This is RemoveDispatch, a member of class TDataBlockInterpreter.
V	StoreObject (see page 214)	This is StoreObject, a member of class TDataBlockInterpreter.
V	UnlockObject (see page 214)	This is UnlockObject, a member of class TDataBlockInterpreter.

Public Methods

	Name	Description
V	CallCreateObject (see page 210)	Creates a COM object on a remote application server given its class name.
	Create (see page 208)	Creates and initializes an instance of TDataBlockInterpreter (see page 207).
V	Destroy (see page 208)	Destroys an instance of TDataBlockInterpreter (see page 207).
V	InterpretData (see page 213)	Interprets a call received from a remote application server.

4.1.5.2 TDataBlockInterpreter.Create Constructor

Creates and initializes an instance of TDataBlockInterpreter (see page 207).

Pascal

```
constructor Create(SendDataBlock: ISendDataBlock; CheckRegValue: string);
```

C++

```
__fastcall TDataBlockInterpreter(ISendDataBlock SendDataBlock, UnicodeString CheckRegValue);
```

Description

Applications should not need to instantiate data block interpreters. Typically, they are instantiated by the streamed connection components that use them to marshal data. To access a streamed connection component's data block interpreter instance, use its Interpreter property.

Create takes two arguments:

An ISendDataBlock (see page 276) interface that the data block interpreter can use to send information in a data block to its destination object (usually the remote data module of an application server).

A string that specifies a registry key value which must appear for application servers that can be accessed using the data block interpreter.

See Also

Interpreter (see page 247)

4.1.5.3 TDataBlockInterpreter.Destroy Destructor

Destroys an instance of TDataBlockInterpreter (see page 207).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataBlockInterpreter();
```

Description

Applications have no need to free data block interpreters. They are freed by the streamed connection component that uses them to marshal interface calls.

See Also

System::TObject::Free (see page 3372)

4.1.5.4 TDataBlockInterpreter Methods

The methods of the TDataBlockInterpreter class are listed here.

Protected Methods

	Name	Description
	AddDispatch (see page 209)	This is AddDispatch, a member of class TDataBlockInterpreter.
	CallFreeObject (see page 210)	This is CallFreeObject, a member of class TDataBlockInterpreter.
	CallGetIDsOfNames (see page 210)	This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.
	CallGetServerList (see page 210)	This is CallGetServerList, a member of class TDataBlockInterpreter.
	CallInvoke (see page 211)	This is CallInvoke, a member of class TDataBlockInterpreter.
	CanCreateObject (see page 211)	This is CanCreateObject, a member of class TDataBlockInterpreter.
	CreateObject (see page 211)	This is CreateObject, a member of class TDataBlockInterpreter.
	DoCreateObject (see page 211)	This is DoCreateObject, a member of class TDataBlockInterpreter.
	DoCustomAction (see page 211)	This is DoCustomAction, a member of class TDataBlockInterpreter.
	DoFreeObject (see page 212)	This is DoFreeObject, a member of class TDataBlockInterpreter.
	DoGetAppServerList (see page 212)	This is DoGetAppServerList, a member of class TDataBlockInterpreter.
	DoGetIDsOfNames (see page 212)	This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.
	DoGetServerList (see page 212)	This is DoGetServerList, a member of class TDataBlockInterpreter.
	DoInvoke (see page 213)	This is DoInvoke, a member of class TDataBlockInterpreter.
	InternalCreateObject (see page 213)	This is InternalCreateObject, a member of class TDataBlockInterpreter.
	LockObject (see page 213)	This is LockObject, a member of class TDataBlockInterpreter.
	ReleaseObject (see page 214)	This is ReleaseObject, a member of class TDataBlockInterpreter.
	RemoveDispatch (see page 214)	This is RemoveDispatch, a member of class TDataBlockInterpreter.
	StoreObject (see page 214)	This is StoreObject, a member of class TDataBlockInterpreter.
	UnlockObject (see page 214)	This is UnlockObject, a member of class TDataBlockInterpreter.

Public Methods

	Name	Description
	CallCreateObject (see page 210)	Creates a COM object on a remote application server given its class name.
	InterpretData (see page 213)	Interprets a call received from a remote application server.

4.1.5.4.1 TDataBlockInterpreter.AddDispatch Method

Pascal

```
procedure AddDispatch(Value: TDataDispatch); override;
```

C++

```
virtual __fastcall AddDispatch(TDataDispatch Value);
```

Description

This is AddDispatch, a member of class TDataBlockInterpreter.

4.1.5.4.2 TDataBlockInterpreter.CallCreateObject Method

Creates a COM object on a remote application server given its class name.

Pascal

```
function CallCreateObject(Name: string): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CallCreateObject(UnicodeString Name);
```

Description

CallCreateObject sends a message to the remote application server, telling it to create an object of the class specified by the Name parameter. Name is the string representation of the Class ID (CLSID). CreateObject (see page 211) returns a reference to an interface that can be used to communicate with the remote COM object.

See Also

CreateOleObject

4.1.5.4.3 TDataBlockInterpreter.CallFreeObject Method

Pascal

```
procedure CallFreeObject(DispatchIndex: Integer); override;
```

C++

```
virtual __fastcall CallFreeObject(int DispatchIndex);
```

Description

This is CallFreeObject, a member of class TDataBlockInterpreter.

4.1.5.4.4 TDataBlockInterpreter.CallGetIDsOfNames Method

Pascal

```
function CallGetIDsOfNames(DispatchIndex: Integer; const IID: TGUID; Names: Pointer;
NameCount: Integer; LocaleID: Integer; DispIDs: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT CallGetIDsOfNames(int DispatchIndex, const TGUID IID, void * Names,
int NameCount, int LocaleID, void * DispIDs);
```

Description

This is CallGetIDsOfNames, a member of class TDataBlockInterpreter.

4.1.5.4.5 TDataBlockInterpreter.CallGetServerList Method

Pascal

```
function CallGetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CallGetServerList();
```

Description

This is CallGetServerList, a member of class TDataBlockInterpreter.

4.1.5.4.6 TDataBlockInterpreter.CallInvoke Method

Pascal

```
function CallInvoke(DispatchIndex: Integer; DispID: Integer; const IID: TGUID; LocaleID:
Integer; Flags: Word; var Params; VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer):
HResult; override;
```

C++

```
virtual __fastcall HRESULT CallInvoke(int DispatchIndex, int DispID, const TGUID IID, int
LocaleID, Word Flags,  Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

This is CallInvoke, a member of class TDataBlockInterpreter.

4.1.5.4.7 TDataBlockInterpreter.CanCreateObject Method

Pascal

```
function CanCreateObject(const ClassID: TGUID): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanCreateObject(const TGUID ClassID);
```

Description

This is CanCreateObject, a member of class TDataBlockInterpreter.

4.1.5.4.8 TDataBlockInterpreter.CreateObject Method

Pascal

```
function CreateObject(const Name: string): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant CreateObject(const UnicodeString Name);
```

Description

This is CreateObject, a member of class TDataBlockInterpreter.

4.1.5.4.9 TDataBlockInterpreter.DoCreateObject Method

Pascal

```
procedure DoCreateObject(const Data: IDataBlock);
```

C++

```
__fastcall DoCreateObject(const IDataBlock Data);
```

Description

This is DoCreateObject, a member of class TDataBlockInterpreter.

4.1.5.4.10 TDataBlockInterpreter.DoCustomAction Method

Pascal

```
function DoCustomAction(Action: Integer; const Data: IDataBlock): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCustomAction(int Action, const IDataBlock Data);
```

Description

This is DoCustomAction, a member of class TDataBlockInterpreter.

4.1.5.4.11 TDataBlockInterpreter.DoFreeObject Method

Pascal

```
procedure DoFreeObject(const Data: IDataBlock);
```

C++

```
__fastcall DoFreeObject(const IDataBlock Data);
```

Description

This is DoFreeObject, a member of class TDataBlockInterpreter.

4.1.5.4.12 TDataBlockInterpreter.DoGetAppServerList Method

Pascal

```
procedure DoGetAppServerList(const Data: IDataBlock);
```

C++

```
__fastcall DoGetAppServerList(const IDataBlock Data);
```

Description

This is DoGetAppServerList, a member of class TDataBlockInterpreter.

4.1.5.4.13 TDataBlockInterpreter.DoGetIDsOfNames Method

Pascal

```
procedure DoGetIDsOfNames(const Data: IDataBlock);
```

C++

```
__fastcall DoGetIDsOfNames(const IDataBlock Data);
```

Description

This is DoGetIDsOfNames, a member of class TDataBlockInterpreter.

4.1.5.4.14 TDataBlockInterpreter.DoGetServerList Method

Pascal

```
procedure DoGetServerList(const Data: IDataBlock);
```

C++

```
__fastcall DoGetServerList(const IDataBlock Data);
```

Description

This is DoGetServerList, a member of class TDataBlockInterpreter.

4.1.5.4.15 TDataBlockInterpreter.DoInvoke Method

Pascal

```
procedure DoInvoke(const Data: IDataBlock);
```

C++

```
__fastcall DoInvoke(const IDataBlock Data);
```

Description

This is DoInvoke, a member of class TDataBlockInterpreter.

4.1.5.4.16 TDataBlockInterpreter.InternalCreateObject Method

Pascal

```
function InternalCreateObject(const ClassID: TGUID): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalCreateObject(const TGUID ClassID);
```

Description

This is InternalCreateObject, a member of class TDataBlockInterpreter.

4.1.5.4.17 TDataBlockInterpreter.InterpretData Method

Interprets a call received from a remote application server.

Pascal

```
procedure InterpretData(const Data: IDataBlock); override;
```

C++

```
virtual __fastcall InterpretData(const IDataBlock Data);
```

Description

Call InterpretData to respond to a COM interface call that is marshaled to a data block. The Data parameter specifies an IDataBlock (see page 266) interface that can be used to read the marshaled version of the message.

InterpretData reads the call, interprets it as an IDispatch (see page 3494) call, and performs the appropriate actions (such as invoking a local COM interface).

4.1.5.4.18 TDataBlockInterpreter.LockObject Method

Pascal

```
function LockObject(ID: Integer): IDispatch; override;
```

C++

```
virtual __fastcall IDispatch LockObject(int ID);
```

Description

This is LockObject, a member of class TDataBlockInterpreter.

4.1.5.4.19 TDataBlockInterpreter.ReleaseObject Method

Pascal

```
procedure ReleaseObject(ID: Integer); override;
```

C++

```
virtual __fastcall ReleaseObject(int ID);
```

Description

This is ReleaseObject, a member of class TDataBlockInterpreter.

4.1.5.4.20 TDataBlockInterpreter.RemoveDispatch Method

Pascal

```
procedure RemoveDispatch(Value: TDataDispatch); override;
```

C++

```
virtual __fastcall RemoveDispatch(TDataDispatch Value);
```

Description

This is RemoveDispatch, a member of class TDataBlockInterpreter.

4.1.5.4.21 TDataBlockInterpreter.StoreObject Method

Pascal

```
function StoreObject(const Value: OleVariant): Integer; override;
```

C++

```
virtual __fastcall int StoreObject(const OleVariant Value);
```

Description

This is StoreObject, a member of class TDataBlockInterpreter.

4.1.5.4.22 TDataBlockInterpreter.UnlockObject Method

Pascal

```
procedure UnlockObject(ID: Integer; const Disp: IDispatch); override;
```

C++

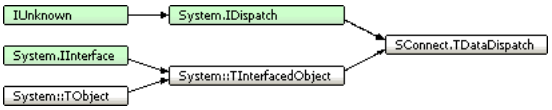
```
virtual __fastcall UnlockObject(int ID, const IDispatch Disp);
```

Description

This is UnlockObject, a member of class TDataBlockInterpreter.

4.1.6 TDataDispatch Class

Class Hierarchy



Pascal

```
TDataDispatch = class(TInterfacedObject, IDispatch);
```

C++

```
class TDataDispatch : public TInterfacedObject, public IDispatch;
```

File

SConnect

Description

This is class SConnect.TDataDispatch.

4.1.6.1 TDataDispatch Members

The following tables list the members exposed by TDataDispatch.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetIDsOfNames (see page 217)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 218)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 218)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 219)	Invoke performs a call to a late-binded property or method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	DispatchIndex (see page 219)	This is DispatchIndex, a member of class TDataDispatch.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.

	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 216)	This is Create, a member of class TDataDispatch.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 217)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

4.1.6.2 TDataDispatch.Create Constructor

Pascal

```
constructor Create(Interpreter: TCustomDataBlockInterpreter; DispatchIndex: Integer);
```

C++

```
__fastcall TDataDispatch(TCustomDataBlockInterpreter Interpreter, int DispatchIndex);
```

Description

This is Create, a member of class TDataDispatch.

4.1.6.3 TDataDispatch.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDataDispatch();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

4.1.6.4 TDataDispatch Methods

The methods of the TDataDispatch class are listed here.

Protected Methods

	Name	Description
	GetIDsOfNames (see page 217)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 218)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 218)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 219)	Invoke performs a call to a late-binded property or method.

4.1.6.4.1 TDataDispatch.GetIDsOfNames Method

GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name. GetIDsOfNames is used primarily to obtain dispatch identifiers for the late-binded methods exposed by the COM object through the IDispatch interface.

See Also

GetTypeInfoCount (see page 3496)

GetTypeInfo (see page 3496)

Invoke (see page 3497)

4.1.6.4.2 TDataDispatch.GetTypeInfo Method

GetTypeInfo returns the object's type information.

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

GetTypeInfo returns the type information for a specified index. The index is previously obtained by a call to the GetTypeInfoCount method. If the method returns S_OK, the `TypeInfo` parameter is set to point to the requested type information; otherwise an error code is returned.

See Also

GetTypeInfoCount (see page 3496)

GetIDsOfNames (see page 3495)

Invoke (see page 3497)

4.1.6.4.3 TDataDispatch.GetTypeInfoCount Method

GetTypeInfoCount returns the count of type information interfaces.

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

GetTypeInfoCount returns the count of type information interfaces supported by the implementing COM object. If the method returns S_OK, the `Count` parameter is set to the number of type information interfaces, which is either 1 (if the object supports type information) or 0 (if the object does not support type information). A return value of E_NOTIMPL means that the method is not implemented in the COM object.

See Also

GetTypeInfo (see page 3496)

GetIDsOfNames (see page 3495)

Invoke (see page 3497)

4.1.6.4.4 TDataDispatch.Invoke Method

Invoke performs a call to a late-binded property or method.

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

Invoke actually calls a method or a property exposed by the COM object through IDispatch. Invoke requires a previously obtained dispatch identifier. The identifier uniquely identifies a method, a property, and a list of parameters to be passed to the called method and exception information. The dispatch identifier of a method or property (also known as **dispid**) can be obtained by a calling the GetIDsOfNames method.

See Also

GetTypeInfoCount (see page 3496)

GetTypeInfo (see page 3496)

GetIDsOfNames (see page 3495)

4.1.6.5 TDataDispatch Properties

The properties of the TDataDispatch class are listed here.

Protected Properties

	Name	Description
	DispatchIndex (see page 219)	This is DispatchIndex, a member of class TDataDispatch.

4.1.6.5.1 TDataDispatch.DispatchIndex Property

Pascal

```
property DispatchIndex: Integer;
```

C++

```
__property int DispatchIndex;
```

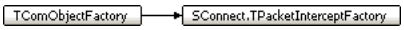
Description

This is DispatchIndex, a member of class TDataDispatch.

4.1.7 TPacketInterceptFactory Class

TPacketInterceptFactory is the type of the class factory used to instantiate a data packet interceptor.

Class Hierarchy



Pascal

```
TPacketInterceptFactory = class(TComObjectFactory);
```

C++

```
class TPacketInterceptFactory : public TComObjectFactory;
```

File

SConnect

Description

In Delphi, use TPacketInterceptFactory as the class factory for COM objects that implement the IDataIntercept (see page 274) interface. Data packet interceptors implement the IDataIntercept (see page 274) interface, which they use in multi-tiered database applications to modify the data that passes between a socket connection component in the client application and scktsrvr.exe on the server. TPacketInterceptFactory registers the data packet interceptor so that can appear in the drop-down list of the Object Inspector when setting the socket connection's InterceptName property and in the corresponding list in the properties dialog of scktsrvr.exe.

TPacketInterceptFactory supports the IUnknown (see page 3756), IClassFactory, and IClassFactory2 interfaces. The IClassFactory interface is implemented by class objects associated with class identifiers (CLSIDs) for the classes that they instantiate. IClassFactory must be implemented for every object definition, so that objects of that type can be created. The IClassFactory2 interface is implemented by class factories for classes that need support for licensing.

See Also

ComClassManager

CreateComObject

CreateOleObject

TStreamedConnection (see page 236)

IDataIntercept (see page 274)

Specifying a Connection Using Sockets

4.1.7.1 TPacketInterceptFactory Members

The following tables list the members exposed by TPacketInterceptFactory.

Public Methods

	Name	Description
	UpdateRegistry (see page 221)	Registers or unregisters COM objects created by the factory.

4.1.7.2 TPacketInterceptFactory Methods

The methods of the TPacketInterceptFactory class are listed here.

Public Methods

	Name	Description
	UpdateRegistry (see page 221)	Registers or unregisters COM objects created by the factory.

4.1.7.2.1 TPacketInterceptFactory.UpdateRegistry Method

Registers or unregisters COM objects created by the factory.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically register or unregister the server.

In addition to the standard COM registration that allows clients to instantiate the data packet interceptor that this factory creates, TPacketInterceptFactory (see page 219) also registers its associated COM object as a data packet interceptor. This allows the object's ProgID to appear in the drop-down list for a socket connection's InterceptName property, and to appear in the corresponding list on the properties dialog for scktsrvr.exe.

See Also

DllRegisterServer

DllUnregisterServer

RegisterClassObject

InterceptName (see page 247)

4.1.8 TSocketConnection Class

TSocketConnection uses Windows sockets to manage the connection to an application server.

Class Hierarchy

Pascal

```
TSocketConnection = class(TStreamedConnection);
```

C++

```
class TSocketConnection : public TStreamedConnection;
```

File

SConnect

Description

TSocketConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The Socket connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer for the application server.

Call into a separate COM object that encodes and decodes the content of messages.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TSocketConnection establishes the initial connection between the client application and a remote application server using TCP/IP. To use TSocketConnection, the application server must be running SocktSrvr.exe.

Once the connection is established, the client registers any or all of its client datasets, and these client datasets use the IAppServer interface from the socket connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using HTTP, use the TWebConnection (see page 254) component instead. To connect using SOAP, use the TSoapConnection component.

See Also

TDataSetProvider

TClientDataSet

TDCOMConnection

TRemoteDataModule

IAppServer

TWebConnection (see page 254)

TSoapConnection (see page 1188)

Specifying a Connection Using Sockets

4.1.8.1 TSocketConnection Members

The following tables list the members exposed by TSocketConnection.

Protected Methods

	Name	Description
	_AddRef (see page 239)	Increments an internal reference counter.
	_Release (see page 239)	Decrements an internal reference counter.
V	CreateTransport (see page 224)	Returns an ITransport (see page 278) interface that can be used to connect to and communicate with the application server.
V	DoConnect (see page 224)	Connects to a remote application server.
V	DoDisconnect (see page 240)	Disconnects from an application server.
V	DoError (see page 241)	raises an exception for a specified exception object.
V	GetInterpreter (see page 242)	Returns the value of the Interpreter (see page 247) property.
V	InternalClose (see page 242)	Terminates a connection to the application server.
V	InternalOpen (see page 243)	Opens a connection to the application server.
	QueryInterface (see page 243)	Implements the IUnknown (see page 3756) QueryInterface method.
	Send (see page 244)	Sends an interface call to the application server and returns the response.
	ThreadException (see page 244)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 244)	Handles callbacks from the application server.
	WndProc (see page 245)	Dispatches messages from the transport layer.

Protected Properties

	Name	Description
R	Handle (see page 246)	Provides access to an internal window handle that can receive messages from the transport layer.

	Interpreter (see page 247)	Provides access to the helper object that marshals interface calls for the streamed connection.

Public Methods

	Name	Description
	Create (see page 223)	Creates and initializes an instance of TSocketConnection (see page 221).
	Destroy (see page 238)	Disposes of a TStreamedConnection (see page 236) instance.
	GetInterceptorList (see page 241)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
	GetServerList (see page 242)	Retrieves a list of registered data brokers from the remote server system.
	Send (see page 277)	Sends information retrieved from an IDataBlock (see page 266) interface to the application server.

Published Properties

	Name	Description
	Address (see page 225)	Specifies the IP address of the server.
	Host (see page 226)	Specifies the host name of the server.
	InterceptGUID (see page 226)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 227)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
	ObjectBroker (see page 228)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Port (see page 228)	Specifies the port number of the socket dispatcher application on the server system.
	SupportCallbacks (see page 229)	Determines whether the streamed connection component can handle callbacks in the application server interface.

4.1.8.2 TSocketConnection.Create Constructor

Creates and initializes an instance of TSocketConnection (see page 221).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSocketConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a TSocketConnection (see page 221) object at runtime. Socket connection components added to forms or data modules at design time are created automatically.

Create overrides the inherited constructor to initialize the Port (see page 228) property to 211.

See Also

Create (see page 237)

Port (see page 228)

4.1.8.3 TSocketConnection Methods

The methods of the TSocketConnection class are listed here.

Protected Methods

	Name	Description
	CreateTransport (see page 224)	Returns an ITransport (see page 278) interface that can be used to connect to and communicate with the application server.
	DoConnect (see page 224)	Connects to a remote application server.

4.1.8.3.1 TSocketConnection.CreateTransport Method

Returns an ITransport (see page 278) interface that can be used to connect to and communicate with the application server.

Pascal

```
function CreateTransport: ITransport; override;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Socket connection components use an ITransport (see page 278) interface to connect to and communicate with the application server. CreateTransport instantiates the internal helper object that connects to the socket dispatcher on the server and sends or receives messages. CreateTransport passes the values of the Host (see page 226), Address (see page 225) and Port (see page 228) properties to this helper object and returns its ITransport (see page 278) interface.

See Also

Host (see page 226)

Address (see page 225)

Port (see page 228)

4.1.8.3.2 TSocketConnection.DoConnect Method

Connects to a remote application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is used internally to connect to the application server when the Connected property is set to true. If the Host (see page 226) or Address (see page 225) and Port (see page 228) properties are set, DoConnect uses these to locate the socket on the server machine. Otherwise, if the ObjectBroker (see page 228) property is set, it obtains these values from the object broker.

Once the server machine is located, DoConnect opens a connection.

Note: When ObjectBroker (see page 228) is nil (Delphi) or NULL (C++), the values of Host (see page 226), Address (see page 225), and Port (see page 228) are saved with the socket connection component. If ObjectBroker (see page 228) is not nil (Delphi) or NULL (C++), these values are not saved from design time. However, once these values are set (either by values obtained from the Object Broker or by values specified in code at runtime), they are used every time a connection must be re-established. If these values subsequently fail, the socket connection component fetches new values from the Object Broker.

See Also

Connected

Host (see page 226)

Address (see page 225)

Port (see page 228)

ObjectBroker

4.1.8.4 TSocketConnection Properties

The properties of the TSocketConnection class are listed here.

Published Properties

	Name	Description
	Address (see page 225)	Specifies the IP address of the server.
	Host (see page 226)	Specifies the host name of the server.
	InterceptGUID (see page 226)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 227)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
	ObjectBroker (see page 228)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Port (see page 228)	Specifies the port number of the socket dispatcher application on the server system.
	SupportCallbacks (see page 229)	Determines whether the streamed connection component can handle callbacks in the application server interface.

4.1.8.4.1 TSocketConnection.Address Property

Specifies the IP address of the server.

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

Use Address to specify the IP address of the application server to which the client application connects. In addition to the address, you must provide a value for the Port (see page 228) property so that the socket connection component can locate the listening socket on the server system.

Sockets (see page 1367) require either an IP address or a Host (see page 226) name to locate the server system. You can use the Host (see page 226) property instead of the Address property to specify the server system using a host name. The Address and Host (see page 226) properties are mutually exclusive: Setting Address causes Host (see page 226) to be set to an empty string, and vice versa.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Address property. Instead, the Object Broker supplies an Address for a server that is located dynamically at runtime. Explicitly setting the Address property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Host (see page 226)

Port (see page 228)

ObjectBroker

Specifying a Connection Using Sockets

4.1.8.4.2 TSocketConnection.Host Property

Specifies the host name of the server.

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

Use Host to specify the host name of the server system to which the client application should connect. In addition to the host name, you must provide a value for the Port (see page 228) property so that the socket connection component can locate the listening socket on the server system.

Sockets (see page 1367) require either a Host name or an IP address to locate the server system. You can use the Address (see page 225) property instead of the Host property to specify the server system using an IP address. The Host and Address (see page 225) properties are mutually exclusive: Setting Host causes Address (see page 225) to be set to an empty string, and vice versa.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Host property. Instead, the Object Broker supplies a host name for a server that is located dynamically at runtime. Explicitly setting the Host property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection component.

See Also

Address (see page 225)

Port (see page 228)

ObjectBroker

Specifying a Connection Using Sockets

4.1.8.4.3 TSocketConnection.InterceptGUID Property

Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptGUID: string;
```

C++

```
__property UnicodeString InterceptGUID;
```

Description

Set InterceptGUID to the GUID of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

The COM object must implement the IDataIntercept interface. Use the IDataIntercept methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a ProgID instead of a GUID, use the InterceptName property

instead.

Note: When using InterceptGUID, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptGUID. For servers that respond to TSocketConnection, you can set the server's InterceptGUID to the same GUID using the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 274)

InterceptName (see page 247)

GetInterceptorList (see page 241)

Specifying a Connection Using Sockets

4.1.8.4.4 TSocketConnection.InterceptName Property

Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptName: string;
```

C++

```
__property UnicodeString InterceptName;
```

Description

Set InterceptName to the programmatic ID (ProgID) of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

Note: In Delphi, you can choose the ProgID of any COM objects that have been registered to perform this task at Design time, using a drop-down list in the Object Inspector, if they use TPacketInterceptFactory as a class factory.

The COM object must implement the IDataIntercept interface. Use the IDataIntercept methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a GUID instead of a ProgID, use the InterceptGUID property instead.

Note: When using InterceptName, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptName. For servers that respond to TSocketConnection, you can set the COM object that the server uses with the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 274)

InterceptGUID (see page 246)

GetInterceptorList (see page 241)

TPacketInterceptFactory (see page 219)

Specifying a Connection Using Sockets

4.1.8.4.5 TSocketConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker;
```

C++

```
__property ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect

ServerName

ServerGUID

TSimpleObjectBroker

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

4.1.8.4.6 TSocketConnection.Port Property

Specifies the port number of the socket dispatcher application on the server system.

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

Use Port to specify the port that the socket dispatcher uses to listen for client connection requests. By default, Port is 211, which is the default port number of ScktSrvr.exe (the socket dispatcher application). Change the value of Port if the socket dispatcher is configured to use a different port.

Note: If the socket connection component uses an Object Broker, you do not need to specify the Port property. Instead, the Object Broker supplies a port number for a server that is located dynamically at runtime. Explicitly setting the Port property overrides the value supplied by the Object Broker, but such overridden values are not saved with the socket connection

component.

See Also

Address (see page 225)

Host (see page 226)

ObjectBroker

Specifying a Connection Using Sockets

4.1.8.4.7 TSocketConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection, callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

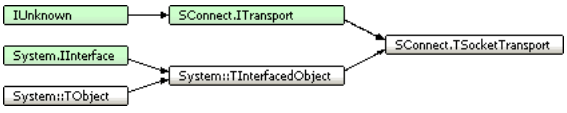
See Also

Send (see page 244)

Extending the Interface of the Application Server

4.1.9 TSocketTransport Class

Class Hierarchy



Pascal

```
TSocketTransport = class(TInterfacedObject, ITransport);
```

C++

```
class TSocketTransport : public TInterfacedObject, public ITransport;
```

File

SConnect

Description

This is class SConnect.TSocketTransport.

4.1.9.1 TSocketTransport Members

The following tables list the members exposed by TSocketTransport.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetConnected (see page 232)	Returns the value of the Connected property.
	GetWaitEvent (see page 233)	Returns the handle for an event object that is managed by the object that implements ITransport.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	Receive (see page 233)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 234)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 234)	Sets the value of the Connected property.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 231)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 231)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.

	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Address (see page 234)	This is Address, a member of class TSocketTransport.
	Connected (see page 281)	Indicates whether the connection to the application server is established.
	Host (see page 235)	This is Host, a member of class TSocketTransport.
	InterceptGUID (see page 235)	This is InterceptGUID, a member of class TSocketTransport.
	Port (see page 235)	This is Port, a member of class TSocketTransport.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Socket (see page 235)	This is Socket, a member of class TSocketTransport.

4.1.9.2 TSocketTransport.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSocketTransport();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

4.1.9.3 TSocketTransport.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSocketTransport();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

4.1.9.4 TSocketTransport Methods

The methods of the TSocketTransport class are listed here.

Protected Methods

	Name	Description
	GetConnected (see page 232)	Returns the value of the Connected property.
	GetWaitEvent (see page 233)	Returns the handle for an event object that is managed by the object that implements ITransport.
	Receive (see page 233)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 234)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 234)	Sets the value of the Connected property.

4.1.9.4.1 TSocketTransport.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean GetConnected();
```

Description

GetConnected is the read property implementer for the Connected property.

See Also

Connected (see page 281)

4.1.9.4.2 TSocketTransport.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport.

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport.

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport, create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 3133)

4.1.9.4.3 TSocketTransport.Receive Method

Returns an IDataBlock interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport. Its value is the return value of the Send method.

See Also

Memory (see page 272)

Send (see page 280)

4.1.9.4.4 TSocketTransport.Send Method

Sends information retrieved from an IDataBlock interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 272)

Receive (see page 280)

4.1.9.4.5 TSocketTransport.SetConnected Method

Sets the value of the Connected property.

Pascal

```
procedure SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall SetConnected(Boolean Value);
```

Description

SetConnected is the write implementer for the Connected property.

4.1.9.5 TSocketTransport Properties

The properties of the TSocketTransport class are listed here.

Public Properties

	Name	Description
	Address (see page 234)	This is Address, a member of class TSocketTransport.
	Host (see page 235)	This is Host, a member of class TSocketTransport.
	InterceptGUID (see page 235)	This is InterceptGUID, a member of class TSocketTransport.
	Port (see page 235)	This is Port, a member of class TSocketTransport.
	Socket (see page 235)	This is Socket, a member of class TSocketTransport.

4.1.9.5.1 TSocketTransport.Address Property

Pascal

```
property Address: string;
```

C++

```
__property UnicodeString Address;
```

Description

This is Address, a member of class TSocketTransport.

4.1.9.5.2 TSocketTransport.Host Property

Pascal

```
property Host: string;
```

C++

```
__property UnicodeString Host;
```

Description

This is Host, a member of class TSocketTransport.

4.1.9.5.3 TSocketTransport.InterceptGUID Property

Pascal

```
property InterceptGUID: string;
```

C++

```
__property UnicodeString InterceptGUID;
```

Description

This is InterceptGUID, a member of class TSocketTransport.

4.1.9.5.4 TSocketTransport.Port Property

Pascal

```
property Port: Integer;
```

C++

```
__property int Port;
```

Description

This is Port, a member of class TSocketTransport.

4.1.9.5.5 TSocketTransport.Socket Property

Pascal

```
property Socket: TCustomWinSocket;
```

C++

```
__property TCustomWinSocket Socket;
```

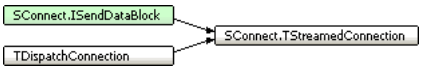
Description

This is Socket, a member of class TSocketTransport.

4.1.10 TStreamedConnection Class

TStreamedConnection is the base class for components that handle their own marshaling of interface calls when connecting to application servers.

Class Hierarchy



Pascal

```
TStreamedConnection = class(TDispatchConnection, ISendDataBlock);
```

C++

```
class TStreamedConnection : public TDispatchConnection, public ISendDataBlock;
```

File

SConnect

Description

Do not create instances of TStreamedConnection. Instead, instantiate a descendant of TStreamedConnection such as TSocketConnection (see page 221) or TWebConnection (see page 254).

TStreamedConnection handles the marshaling of all COM data types so that the streamed connection component can generate, interpret, and respond to remote COM interface calls without using DCOM. It uses an ITransport (see page 278) interface to connect to application servers and to send and receive messages.

TStreamedConnection can't actually connect to or communicate with an application server. Descendants of TStreamedConnection override the CreateTransport (see page 239) method to obtain an ITransport (see page 278) interface that enables these behaviors.

TStreamedConnection implements the ISendDataBlock (see page 276) interface.

See Also

TClientDataSet

TSocketConnection (see page 221)

TWebConnection (see page 254)

TStream

ITransport (see page 278)

TRemoteDataModule

IAppServer

4.1.10.1 TStreamedConnection Members

The following tables list the members exposed by TStreamedConnection.

Protected Methods

	Name	Description
	_AddRef (see page 239)	Increments an internal reference counter.
	_Release (see page 239)	Decrements an internal reference counter.
V	CreateTransport (see page 239)	Provides the backstop for a method that obtains an ITransport (see page 278) interface.
V	DoConnect (see page 240)	Connects to an application server.
V	DoDisconnect (see page 240)	Disconnects from an application server.
V	DoError (see page 241)	raises an exception for a specified exception object.
V	GetInterpreter (see page 242)	Returns the value of the Interpreter (see page 247) property.
V	InternalClose (see page 242)	Terminates a connection to the application server.
V	InternalOpen (see page 243)	Opens a connection to the application server.
	QueryInterface (see page 243)	Implements the IUnknown (see page 3756) QueryInterface method.
	Send (see page 244)	Sends an interface call to the application server and returns the response.
	ThreadException (see page 244)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 244)	Handles callbacks from the application server.
	WndProc (see page 245)	Dispatches messages from the transport layer.

Protected Properties

	Name	Description
R	Handle (see page 246)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 246)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 247)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 247)	Provides access to the helper object that marshals interface calls for the streamed connection.
	SupportCallbacks (see page 248)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Public Methods

	Name	Description
V	Create (see page 237)	Creates and initializes an instance of TStreamedConnection (see page 236).
V	Destroy (see page 238)	Disposes of a TStreamedConnection (see page 236) instance.
V	GetInterceptorList (see page 241)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
V	GetServerList (see page 242)	Retrieves a list of registered data brokers from the remote server system.

4.1.10.2 TStreamedConnection.Create Constructor

Creates and initializes an instance of TStreamedConnection (see page 236).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TStreamedConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a streamed connection component at runtime. Connection components placed in forms or data modules at design time are instantiated automatically.

After calling the inherited constructor, Create instantiates the data block interpreter accessed by the Interpreter (see page 247) property.

See Also

Interpreter (see page 247)

Create

4.1.10.3 TStreamedConnection.Destroy Destructor

Disposes of a TStreamedConnection (see page 236) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TStreamedConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the streamed connection is not nil, and only then calls Destroy.

Destroy ensures that the connection to the server is closed. It then frees all resources and helper objects before calling the inherited destructor.

See Also

System::TObject::Free (see page 3372)

4.1.10.4 TStreamedConnection Methods

The methods of the TStreamedConnection class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 239)	Increments an internal reference counter.
	_Release (see page 239)	Decrements an internal reference counter.
V	CreateTransport (see page 239)	Provides the backstop for a method that obtains an ITransport (see page 278) interface.
V	DoConnect (see page 240)	Connects to an application server.
V	DoDisconnect (see page 240)	Disconnects from an application server.
V	DoError (see page 241)	raises an exception for a specified exception object.
V	GetInterpreter (see page 242)	Returns the value of the Interpreter (see page 247) property.
V	InternalClose (see page 242)	Terminates a connection to the application server.
V	InternalOpen (see page 243)	Opens a connection to the application server.
	QueryInterface (see page 243)	Implements the IUnknown (see page 3756) QueryInterface method.
	Send (see page 244)	Sends an interface call to the application server and returns the response.
	ThreadException (see page 244)	raises an exception for the thread that communicates with the application server.
	ThreadReceivedStream (see page 244)	Handles callbacks from the application server.
	WndProc (see page 245)	Dispatches messages from the transport layer.

Public Methods

	Name	Description
V	GetInterceptorList (see page 241)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
V	GetServerList (see page 242)	Retrieves a list of registered data brokers from the remote server system.

4.1.10.4.1 TStreamedConnection._AddRef Method

Increments an internal reference counter.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef implements the IUnknown (see page 3756) interface method, _AddRef, by incrementing an internal reference counter. The reference count is decreased by calling _Release (see page 239).

See Also

_Release (see page 239)

QueryInterface (see page 243)

4.1.10.4.2 TStreamedConnection._Release Method

Decrements an internal reference counter.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release implements the IUnknown (see page 3756) interface method, _Release, by decrementing an internal reference counter. The reference count is increased by calling _AddRef (see page 239)

See Also

_AddRef (see page 239)

QueryInterface (see page 243)

4.1.10.4.3 TStreamedConnection.CreateTransport Method

Provides the backstop for a method that obtains an ITransport (see page 278) interface.

Pascal

```
function CreateTransport: ITransport; virtual;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Streamed connection components use an ITransport (see page 278) interface to connect to and communicate with the application server. As implemented in TStreamedConnection (see page 236), CreateTransport always returns nil (Delphi) or NULL (C++).

Descendants of TStreamedConnection (see page 236) override CreateTransport to obtain an ITransport (see page 278)

interface that connects to and communicates with the application server.

Component writers wanting to create connection components for specialized transport protocols need only override CreateTransport. Once the streamed connection has a working ITransport (see page 278) interface, all other functions of the connection component are handled automatically.

See Also

Send (see page 244)

DoConnect (see page 240)

DoDisconnect (see page 240)

ThreadReceivedStream (see page 244)

4.1.10.4.4 TStreamedConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected property is set to true. DoConnect creates a separate thread for managing the connection so that communication with the application server does not block execution in the client application. Using the ITransport (see page 278) interface, DoConnect connects to the application server and sets the value of the AppServer property to its interface.

Note: Descendants of TStreamedConnection (see page 236) override CreateTransport (see page 239) rather than DoConnect to change the way the client application forms a connection to application servers.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

CreateTransport (see page 239)

4.1.10.4.5 TStreamedConnection.DoDisconnect Method

Disconnects from an application server.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

DoDisconnect is called internally when the Connected property is set to false. Using the ITransport (see page 278) interface,

DoDisconnect drops the connection to the application server.

See Also

Connected

CreateTransport (see page 239)

4.1.10.4.6 TStreamedConnection.DoError Method

raises an exception for a specified exception object.

Pascal

```
procedure DoError(E: Exception); virtual;
```

C++

```
virtual __fastcall DoError(Exception E);
```

Description

TStreamedConnection (see page 236) uses a separate thread when making interface calls to the application server. When that thread encounters an exception, it sends a message to the Window handle specified by the Handle (see page 246) property. The connection component uses DoError to raise those exceptions.

Note: DoError is called by the ThreadException (see page 244) method, which is the message handler for thread exceptions. It is also called when thread exception messages are taken from the message queue before they are received by the internal window procedure.

See Also

ThreadException (see page 244)

Handle (see page 246)

EInterpreterError (see page 191)

4.1.10.4.7 TStreamedConnection.GetInterceptorList Method

Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.

Pascal

```
function GetInterceptorList: OleVariant; virtual;
```

C++

```
virtual __fastcall OleVariant GetInterceptorList();
```

Description

Call GetInterceptorList to obtain a list of ProgIDs that can be assigned to the InterceptName (see page 247) property. GetInterceptorList retrieves a list of all COM objects that have been registered by TPacketInterceptFactory (see page 219) as data packet interceptors (objects that support the IDataIntercept (see page 274) interface that can be used to intercept and transform messages passed between a connection component and an application server.)

GetInterceptorList returns a Variant (see page 3421) array of strings. Any of these strings can be used as a valid setting for the InterceptName (see page 247) property.

At design time, TStreamedConnection (see page 236) uses the GetInterceptorList method to populate the InterceptName (see page 247) drop-down in the Object Inspector.

See Also

IDataIntercept (see page 274)

InterceptName (see page 247)

InterceptGUID (see page 246)

TPacketInterceptFactory (see page 219)

4.1.10.4.8 TStreamedConnection.GetInterpreter Method

Returns the value of the Interpreter (see page 247) property.

Pascal

```
function GetInterpreter: TCustomDataBlockInterpreter; virtual;
```

C++

```
virtual __fastcall TCustomDataBlockInterpreter GetInterpreter();
```

Description

GetInterpreter is the protected read implementation of the Interpreter (see page 247) property. As implemented in TStreamedConnection (see page 236), GetInterpreter instantiates a TDataBlockInterpreter (see page 207) object.

See Also

Interpreter (see page 247)

TDataBlockInterpreter (see page 207)

4.1.10.4.9 TStreamedConnection.GetServerList Method

Retrieves a list of registered data brokers from the remote server system.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Call GetServerList to obtain a list of servers that are registered on the system to which the streamed connection communicates. GetServerList returns a Variant (see page 3421) array of strings. Any of these strings can be used as a valid setting for the ServerName property.

At design time, TStreamedConnection (see page 236) uses the GetServerList method to populate the ServerName drop-down in the Object Inspector. Note that, unlike COM-based connection components (which populate that list with servers that are registered on the client machine), this results in a list of application servers that are registered on the server machine.

See Also

ServerName

4.1.10.4.10 TStreamedConnection.InternalClose Method

Terminates a connection to the application server.

Pascal

```
procedure InternalClose; virtual;
```

C++

```
virtual __fastcall InternalClose();
```

Description

Streamed connection components use InternalClose to handle the details of closing a connection to the application server. This method is called by the DoDisconnect (see page 240) method (which implements the Connected property) and at other times when the connection component must terminate a connection.

See Also

Connected

DoDisconnect (see page 240)

4.1.10.4.11 TStreamedConnection.InternalOpen Method

Opens a connection to the application server.

Pascal

```
procedure InternalOpen; virtual;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

Streamed connection components use InternalOpen to handle the details of establishing a connection to the application server. This method is called by the DoConnect (see page 240) method (which implements the Connected property) and at other times when the connection component must establish a connection.

See Also

Connected

DoConnect (see page 240)

4.1.10.4.12 TStreamedConnection.QueryInterface Method

Implements the IUnknown (see page 3756) QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall; reintroduce;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface returns a reference to the interface specified by the IID parameter as the Obj parameter. It the streamed connection supports the specified interface, QueryInterface returns 0. If the streamed connection does not support the interface, the Obj parameter is nil (Delphi) or NULL (C++) and QueryInterface returns E_NOINTERFACE (see page 3845).

See Also

_AddRef (see page 239)

ISendDataBlock (see page 276)

4.1.10.4.13 TStreamedConnection.Send Method

Sends an interface call to the application server and returns the response.

Pascal

```
function Send(const Data: IDataBlock; WaitForResult: Boolean): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Send(const IDataBlock Data, Boolean WaitForResult);
```

Description

The data block interpreter (specified by the Interpreter (see page 247) property) calls Send to dispatch interface messages to the application server. It supplies an IDataBlock (see page 266) interface that encapsulates the interface from the application server.

Communication with the application server occurs on a separate thread. The WaitForResult parameter indicates whether execution on the main thread should halt until the interface call returns a result (WaitForResult == true), or whether the interface call can occur asynchronously (WaitForResult == false). If WaitForResult is true, Send returns an IDataBlock (see page 266) interface (_di_IDataBlock in C++) that supplies the result.

See Also

Interpreter (see page 247)

ThreadReceivedStream (see page 244)

4.1.10.4.14 TStreamedConnection.ThreadException Method

raises an exception for the thread that communicates with the application server.

Pascal

```
procedure ThreadException(var Message: TMessage); message THREAD_EXCEPTION;
```

C++

```
__fastcall ThreadException(TMessage Message);
```

Description

TStreamedConnection (see page 236) uses a separate thread when making interface calls to the application server. When that thread encounters an exception, it sends a message to the Window handle specified by the Handle (see page 246) property. ThreadException is the message handler for that Windows message. It raises an exception in the main execution thread.

See Also

Handle (see page 246)

4.1.10.4.15 TStreamedConnection.ThreadReceivedStream Method

Handles callbacks from the application server.

Pascal

```
procedure ThreadReceivedStream(var Message: TMessage); message THREAD_RECEIVEDSTREAM;
```

C++

```
__fastcall ThreadReceivedStream(TMessage Message);
```

Description

TStreamedConnection (see page 236) uses a separate thread for interface calls to and from the application server. When the application server calls a client application interface, that thread sends a message to the Window handle specified by the Handle (see page 246) property. ThreadReceivedStream is the message handler for the Windows message from the application server.

ThreadReceivedStream passes the interface call to the data block interpreter specified by the Interpreter (see page 247) property.

See Also

Handle (see page 246)

Interpreter (see page 247)

4.1.10.4.16 TStreamedConnection.WndProc Method

Dispatches messages from the transport layer.

Pascal

```
procedure WndProc(var Message: TMessage);
```

C++

```
__fastcall WndProc(TMessage Message);
```

Description

TStreamedConnection (see page 236) uses a separate thread for interface calls to and from the application server. That thread communicates with the streamed connection component by sending Windows messages to the handle specified by the Handle (see page 246) property. WndProc is the window procedure associated with that Window handle. It dispatches any messages that have message handlers such as ThreadException (see page 244) or ThreadReceivedStream (see page 244). All other messages are ignored.

See Also

Handle (see page 246)

ThreadException (see page 244)

ThreadReceivedStream (see page 244)

4.1.10.5 TStreamedConnection Properties

The properties of the TStreamedConnection class are listed here.

Protected Properties

	Name	Description
R	Handle (see page 246)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 246)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 247)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 247)	Provides access to the helper object that marshals interface calls for the streamed connection.
	SupportCallbacks (see page 248)	Determines whether the streamed connection component can handle callbacks in the application server interface.

4.1.10.5.1 TStreamedConnection.Handle Property

Provides access to an internal window handle that can receive messages from the transport layer.

Pascal

```
property Handle: THandle;
```

C++

```
__property THandle Handle;
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received.

The internal thread that handles these communication tasks uses Windows messages to coordinate events. TStreamedConnection (see page 236) uses the Handle property to provide this internal thread with a window handle that it can use.

See Also

WndProc (see page 245)

ThreadReceivedStream (see page 244)

ThreadException (see page 244)

4.1.10.5.2 TStreamedConnection.InterceptGUID Property

Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptGUID: string;
```

C++

```
__property UnicodeString InterceptGUID;
```

Description

Set InterceptGUID to the GUID of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

The COM object must implement the IDataIntercept (see page 274) interface. Use the IDataIntercept (see page 274) methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a ProgID instead of a GUID, use the InterceptName (see page 247) property instead.

Note: When using InterceptGUID, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptGUID. For servers that respond to TSocketConnection (see page 221), you can set the server's InterceptGUID to the same GUID using the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 274)

InterceptName (see page 247)

GetInterceptorList (see page 241)

Specifying a Connection Using Sockets

4.1.10.5.3 TStreamedConnection.InterceptName Property

Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.

Pascal

```
property InterceptName: string;
```

C++

```
__property UnicodeString InterceptName;
```

Description

Set InterceptName to the programmatic ID (ProgID) of a COM object that intercepts messages between the connection component and an application server before they are sent and after they are received.

Note: In Delphi, you can choose the ProgID of any COM objects that have been registered to perform this task at Design time, using a drop-down list in the Object Inspector, if they use TPacketInterceptFactory (see page 219) as a class factory.

The COM object must implement the IDataIntercept (see page 274) interface. Use the IDataIntercept (see page 274) methods to manipulate the data that is communicated using the socket connection. For example, applications can use these methods to encrypt and decrypt data or to compress and decompress data.

Tip: To specify the COM object that intercepts messages using a GUID instead of a ProgID, use the InterceptGUID (see page 246) property instead.

Note: When using InterceptName, the server must use the same COM object to undo the transformation performed by the socket connection component's InterceptName. For servers that respond to TSocketConnection (see page 221), you can set the COM object that the server uses with the Socket server dialog of ScktSrvr.exe.

See Also

IDataIntercept (see page 274)

InterceptGUID (see page 246)

GetInterceptorList (see page 241)

TPacketInterceptFactory (see page 219)

Specifying a Connection Using Sockets

4.1.10.5.4 TStreamedConnection.Interpreter Property

Provides access to the helper object that marshals interface calls for the streamed connection.

Pascal

```
property Interpreter: TCustomDataBlockInterpreter;
```

C++

```
__property TCustomDataBlockInterpreter Interpreter;
```

Description

Streamed connection components use a data block interpreter internally to provide marshaling support.

See Also

Send (see page 244)

4.1.10.5.5 TStreamedConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection (see page 221), callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection (see page 254) can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

See Also

Send (see page 244)

Extending the Interface of the Application Server

4.1.11 TTransportThread Class

Class Hierarchy

Pascal

```
TTransportThread = class(TThread);
```

C++

```
class TTransportThread : public TThread;
```

File

SConnect

Description

This is class SConnect.TTransportThread.

4.1.11.1 TTransportThread Members

The following tables list the members exposed by TTransportThread.

Public Methods

	Name	Description
	Create (see page 249)	This is Create, a member of class TTransportThread.
	Destroy (see page 249)	Destroys the thread object and releases the memory allocated to it.
	Execute (see page 250)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Properties

	Name	Description
	Semaphore (see page 254)	This is Semaphore, a member of class TTransportThread.

4.1.11.2 TTransportThread.Create Constructor

Pascal

```
constructor Create(AHandle: THandle; Transport: ITransport); virtual;
```

C++

```
virtual __fastcall TTransportThread(THandle AHandle, ITransport Transport);
```

Description

This is Create, a member of class TTransportThread.

4.1.11.3 TTransportThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTransportThread();
```

Description

Do not call Destroy in an application. Instead, use Free to destroy a thread when FreeOnTerminate is False.

Destroy signals the thread to terminate and then waits for the thread to return before calling the inherited Destroy method.

See Also

Create

Terminate

WaitFor

FreeOnTerminate

System::TObject::Free (see page 3372)

4.1.11.4 TTransportThread Methods

The methods of the TTransportThread class are listed here.

Public Methods

	Name	Description
	Execute (see page 250)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

4.1.11.4.1 TTransportThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}
```

```
class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended - secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
```

```
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended – secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
```

```
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create

Terminated

Synchronize

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

4.1.11.5 TTransportThread Properties

The properties of the TTransportThread class are listed here.

Public Properties

	Name	Description
	Semaphore (see page 254)	This is Semaphore, a member of class TTransportThread.

4.1.11.5.1 TTransportThread.Semaphore Property

Pascal

```
property Semaphore: THandle;
```

C++

```
__property THandle Semaphore;
```

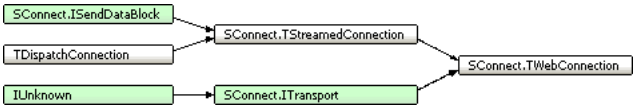
Description

This is Semaphore, a member of class TTransportThread.

4.1.12 TWebConnection Class

TWebConnection uses HTTP to manage the connection to an application server.

Class Hierarchy



Pascal

```
TWebConnection = class(TStreamedConnection, ITransport);
```

C++

```
class TWebConnection : public TStreamedConnection, public ITransport;
```

File

SConnect

Description

TWebConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server. The Web connection object can

Establish an initial connection to a remote application server.

Obtain an IAppServer for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TWebConnection establishes the initial connection between the client application and a remote application server using HTTP. To use TWebConnection, you must have the following:

Wininet.dll must be installed on the client system. Wininet.dll is found in the Windows system directory if you have IE3 or higher installed.

The server system must have IIS version 4 or better or Netscape Enterprise version 3.6 or better.

HTTPsrvr.dll must be installed with the Web server to which TWebConnection forms a connection. HTTPsrvr.dll launches the application server and marshals all calls to the application server interface.

Once the connection is established, the client registers any or all of its client datasets, and these client datasets use the IAppServer interface from the Web connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using TCP/IP, use the TSocketConnection (see page 221) component instead. To connect to remote application servers using SOAP, use the TSoapConnection component.

See Also

TDataSetProvider

TClientDataSet

TDCOMConnection

TRemoteDataModule

IAppServer

TSocketConnection (see page 221)

TSoapConnection (see page 1188)

Extending the Interface of the Application Server

Specifying a Connection Using HTTP

4.1.12.1 TWebConnection Members

The following tables list the members exposed by TWebConnection.

Protected Methods

	Name	Description
	_AddRef (see page 239)	Increments an internal reference counter.
	_Release (see page 239)	Decrements an internal reference counter.
V	CreateTransport (see page 257)	Provides the backstop for a method that obtains an ITransport interface.
V	DoConnect (see page 258)	Connects to an application server.
V	DoDisconnect (see page 240)	Disconnects from an application server.
V	DoError (see page 241)	raises an exception for a specified exception object.
V	GetInterpreter (see page 258)	Returns the value of the Interpreter property.
	GetWaitEvent (see page 259)	Returns the handle for an event object that is managed by the object that implements ITransport.
V	InternalClose (see page 242)	Terminates a connection to the application server.
V	InternalOpen (see page 243)	Opens a connection to the application server.
	QueryInterface (see page 243)	Implements the IUnknown (see page 3756) QueryInterface method.
	Receive (see page 259)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 260)	Sends information retrieved from an IDataBlock interface to the application server.
V	SetConnected (see page 260)	Sets the Connected property.
	ThreadException (see page 244)	raises an exception for the thread that communicates with the application server.

	ThreadReceivedStream (see page 244)	Handles callbacks from the application server.
	Transport_GetConnected (see page 261)	This is Transport_GetConnected, a member of class TWebConnection.
	Transport_SetConnected (see page 261)	This is Transport_SetConnected, a member of class TWebConnection.
	WndProc (see page 245)	Dispatches messages from the transport layer.

Protected Properties

	Name	Description
R	Handle (see page 246)	Provides access to an internal window handle that can receive messages from the transport layer.
	InterceptGUID (see page 246)	Specifies the GUID of a COM object that manipulates interface calls before they are sent or after return values are received.
	InterceptName (see page 247)	Specifies the ProgID of a COM object that manipulates interface calls before they are sent or after return values are received.
R	Interpreter (see page 247)	Provides access to the helper object that marshals interface calls for the streamed connection.
	SupportCallbacks (see page 264)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Public Methods

	Name	Description
V	Create (see page 256)	Creates and initializes an instance of TWebConnection (see page 254).
V	Destroy (see page 257)	Destroys an instance of TWebConnection (see page 254).
	GetConnected (see page 279)	Returns the value of the Connected (see page 281) property.
V	GetInterceptorList (see page 241)	Retrieves a list of registered interceptors that can be used to transform messages between the connection component and the application server.
V	GetServerList (see page 242)	Retrieves a list of registered data brokers from the remote server system.

Public Properties

	Name	Description
	Connected (see page 281)	Indicates whether the connection to the application server is established.

Published Properties

	Name	Description
	Agent (see page 262)	Provides the value of the User-Agent header for HTTP messages to the server.
	ObjectBroker (see page 262)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Password (see page 263)	Specifies the password that allows the client application to log on to the application server.
	Proxy (see page 263)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 263)	Lists the host names of locally registered servers that do not require a proxy.
	URL (see page 264)	Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.
	UserName (see page 265)	Specifies the user name that allows the client application to log on to the application server.

4.1.12.2 TWebConnection.Create Constructor

Creates and initializes an instance of TWebConnection (see page 254).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a TWebConnection (see page 254) object at runtime. Web connection components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually a form or data module) that is responsible for freeing the Web connection component.

See Also

Create (see page 237)

Owner

4.1.12.3 TWebConnection.Destroy Destructor

Destroys an instance of TWebConnection (see page 254).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebConnection();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the Web connection is not nil before calling Destroy.

4.1.12.4 TWebConnection Methods

The methods of the TWebConnection class are listed here.

Protected Methods

	Name	Description
	CreateTransport (see page 257)	Provides the backstop for a method that obtains an ITransport interface.
	DoConnect (see page 258)	Connects to an application server.
	GetInterpreter (see page 258)	Returns the value of the Interpreter property.
	GetWaitEvent (see page 259)	Returns the handle for an event object that is managed by the object that implements ITransport.
	Receive (see page 259)	Returns an IDataBlock interface that can be used to read information received from the application server.
	Send (see page 260)	Sends information retrieved from an IDataBlock interface to the application server.
	SetConnected (see page 260)	Sets the Connected property.
	Transport_GetConnected (see page 261)	This is Transport_GetConnected, a member of class TWebConnection.
	Transport_SetConnected (see page 261)	This is Transport_SetConnected, a member of class TWebConnection.

4.1.12.4.1 TWebConnection.CreateTransport Method

Provides the backstop for a method that obtains an ITransport interface.

Pascal

```
function CreateTransport: ITransport; override;
```

C++

```
virtual __fastcall ITransport CreateTransport();
```

Description

Streamed connection components use an ITransport interface to connect to and communicate with the application server. As implemented in TStreamedConnection, CreateTransport always returns nil (Delphi) or NULL (C++).

Descendants of TStreamedConnection override CreateTransport to obtain an ITransport interface that connects to and communicates with the application server.

Component writers wanting to create connection components for specialized transport protocols need only override CreateTransport. Once the streamed connection has a working ITransport interface, all other functions of the connection component are handled automatically.

See Also

Send (see page 244)

DoConnect (see page 240)

DoDisconnect (see page 240)

ThreadReceivedStream (see page 244)

4.1.12.4.2 TWebConnection.DoConnect Method

Connects to an application server.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

DoConnect is called internally when the Connected property is set to true. DoConnect creates a separate thread for managing the connection so that communication with the application server does not block execution in the client application. Using the ITransport interface, DoConnect connects to the application server and sets the value of the AppServer property to its interface.

Note: Descendants of TStreamedConnection override CreateTransport rather than DoConnect to change the way the client application forms a connection to application servers.

See Also

AfterConnect

BeforeConnect

Connected

AppServer

CreateTransport (see page 239)

4.1.12.4.3 TWebConnection.GetInterpreter Method

Returns the value of the Interpreter property.

Pascal

```
function GetInterpreter: TCustomDataBlockInterpreter; override;
```

C++

```
virtual __fastcall TCustomDataBlockInterpreter GetInterpreter();
```

Description

GetInterpreter is the protected read implementation of the Interpreter property. As implemented in TStreamedConnection,

GetInterpreter instantiates a TDataBlockInterpreter object.

See Also

Interpreter (see page 247)

TDataBlockInterpreter (see page 207)

4.1.12.4.4 TWebConnection.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport.

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport.

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport, create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 3133)

4.1.12.4.5 TWebConnection.Receive Method

Returns an IDataBlock interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport. Its value is the return value of the Send method.

See Also

Memory (see page 272)

Send (see page 280)

4.1.12.4.6 TWebConnection.Send Method

Sends information retrieved from an IDataBlock interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 272)

Receive (see page 280)

4.1.12.4.7 TWebConnection.SetConnected Method

Sets the Connected property.

Pascal

```
procedure SetConnected(Value: Boolean); override;
```

C++

```
virtual __fastcall SetConnected(Boolean Value);
```

Description

SetConnected is the protected implementation of the Connected property. It establishes a connection to the Application server if Value is true, and disconnects from the application server if Value is false.

If Value is false, SetConnected generates a BeforeDisconnect event, closes the connection, and generates an AfterDisconnect event.

If Value is true, SetConnected performs the following actions:

1. If LoginPrompt is true, and the connection is not already open, SetConnected generates an OnGetUsername event, and uses that to initialize a login dialog. It displays the login dialog, and aborts the connection if the user cancels.
2. It generates a BeforeConnect event.
3. It opens the connection.
4. It generates an AfterConnect event.
5. If LoginPrompt is true, and the connection opened successfully, SetConnected generates an OnLogin event.

See Also

Connected

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

LoginPrompt

OnGetUsername

OnLogin

4.1.12.4.8 TWebConnection.Transport_GetConnected Method

Pascal

```
function Transport_GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean Transport_GetConnected();
```

Description

This is Transport_GetConnected, a member of class TWebConnection.

4.1.12.4.9 TWebConnection.Transport_SetConnected Method

Pascal

```
procedure Transport_SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall Transport_SetConnected(Boolean Value);
```

Description

This is Transport_SetConnected, a member of class TWebConnection.

4.1.12.5 TWebConnection Properties

The properties of the TWebConnection class are listed here.

Protected Properties

	Name	Description
	SupportCallbacks (see page 264)	Determines whether the streamed connection component can handle callbacks in the application server interface.

Published Properties

	Name	Description
	Agent (see page 262)	Provides the value of the User-Agent header for HTTP messages to the server.
	ObjectBroker (see page 262)	Specifies an Object Broker that allows the connection component to choose from several application servers.
	Password (see page 263)	Specifies the password that allows the client application to log on to the application server.
	Proxy (see page 263)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 263)	Lists the host names of locally registered servers that do not require a proxy.
	URL (see page 264)	Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.
	UserName (see page 265)	Specifies the user name that allows the client application to log on to the application server.

4.1.12.5.1 TWebConnection.Agent Property

Provides the value of the User-Agent header for HTTP messages to the server.

Pascal

```
property Agent: string;
```

C++

```
__property UnicodeString Agent;
```

Description

Use Agent to supply HTTP messages generated by the Web connection component with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the HTTP message.

See Also

Title

4.1.12.5.2 TWebConnection.ObjectBroker Property

Specifies an Object Broker that allows the connection component to choose from several application servers.

Pascal

```
property ObjectBroker;
```

C++

```
__property ObjectBroker;
```

Description

Set ObjectBroker to use a broker that maintains a list of servers that support the application specified by ServerName or ServerGUID. When the connection component needs to open a connection, it requests information to identify a server machine from the Object Broker.

Descendants of TDispatchConnection define properties that specify how to locate server machines. These properties are not saved with the connection component if the ObjectBroker property is set. Instead, the Object Broker provides this information dynamically at runtime from its list of available servers.

Applications can use TSimpleObjectBroker for an object broker, or derive their own object brokers from TCustomObjectBroker.

See Also

DoConnect

ServerName

ServerGUID

TSimpleObjectBroker

Brokering Connections

Specifying a Connection Using HTTP

Specifying a Connection Using DCOM

Specifying a Connection Using Sockets

Using TSimpleObjectBroker

4.1.12.5.3 TWebConnection.Password Property

Specifies the password that allows the client application to log on to the application server.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

When the Web connection component connects to the application server, it supplies the values of the UserName (see page 265) and Password properties so that it can log on to the host or proxy. The value of Password can be an empty string if neither the host nor proxy requires authentication.

See Also

UserName (see page 265)

Specifying a Connection Using HTTP

4.1.12.5.4 TWebConnection.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

Use Proxy if the Web connection component can't resolve the value of Host locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 263)

4.1.12.5.5 TWebConnection.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property UnicodeString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the Web connection component can make a direct connection. When the Host property specifies a server in this list, the Web connection component makes a direct connection even if the Proxy (see page 263) property is specified.

When ProxyByPass is an empty string, the Web connection component checks the registry for any locally known servers and

bypasses the proxy server if Host refers to a registered local server.

See Also

Proxy (see page 263)

4.1.12.5.6 TWebConnection.SupportCallbacks Property

Determines whether the streamed connection component can handle callbacks in the application server interface.

Pascal

```
property SupportCallbacks: Boolean;
```

C++

```
__property Boolean SupportCallbacks;
```

Description

Use SupportCallbacks to indicate whether the connection component can marshal callback functions in the application server's interface. When SupportCallbacks is true, the connection component can marshal calls from the application server to the client application over an interface supplied as a callback.

For TSocketConnection, callbacks are optional. Set SupportCallbacks to true if you are using callbacks as part of the application server's interface. When SupportCallbacks is true, the socket connection component requires WinSock2. Set SupportCallbacks to false if you are not using callback functions and want to eliminate the dependency on WinSock2.

TWebConnection can't support callback functions. It initializes SupportCallbacks to false in the constructor and never changes this value.

See Also

Send (see page 244)

Extending the Interface of the Application Server

4.1.12.5.7 TWebConnection.URL Property

Specifies the Uniform Resource Locator for httpsrvr.dll on the server system.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

Use URL to specify the Uniform Resource Locator that the Web connection component uses to locate httpsrvr.dll. This DLL, which ships in the BIN directory, is deployed on the same machine as the Web server so that it can accept HTTP request messages and convert them into COM calls that communicate with the application server.

URL includes the protocol (http or https), host name, and scriptname for httpsrvr.dll. Typically, it has the form

`http://DataHost.org/scripts/httpsrvr.dll`

Note: URL is a standard Uniform Resource Locator. As such, you can also use it to override the port associated with http (80) or https (443) if the Web server is configured differently. The port number follows the host name, preceded by a colon.

See Also

Specifying a Connection Using HTTP

4.1.12.5.8 TWebConnection.UserName Property

Specifies the user name that allows the client application to log on to the application server.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

When the Web connection component connects to the application server on a local host or to the proxy server, it supplies the values of the UserName and Password (see page 263) properties so that it can log in. If neither the host nor the proxy server requires authentication, UserName can be an empty string.

See Also

Password (see page 263)

Specifying a Connection Using HTTP

4.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDataBlock (see page 266)	IDataBlock is the interface used by data block interpreters to access memory buffers.
	IDataIntercept (see page 274)	IDataIntercept is the interface used by TSocketConnection (see page 221) components to transform data that is communicated over the connection.
	ISendDataBlock (see page 276)	ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.
	ITransport (see page 278)	ITransport is the interface used by TStreamedConnection (see page 236) to connect to, read from, and write to a remote application server.

4.2.1 IDataBlock Interface

IDataBlock is the interface used by data block interpreters to access memory buffers.

Class Hierarchy

Pascal

```
IDataBlock = interface(IUnknown);
```

C++

```
__interface IDataBlock : public IUnknown;
```

File

SConnect

Description

Data block interpreters uses IDataBlock when marshaling COM interface calls. Streamed connection components send data obtained from IDataBlock to an application server once the data block interpreter has filled its memory by marshaling interface calls. They also read data from the application server into IDataBlock and pass it to a data block interpreter for marshaling.

IDataBlock is an interface, and as such can't be directly instantiated. Instead, objects that implement the IDataBlock properties and methods include IDataBlock in their class definition (Delphi) or be derived from IDataBlock (C++). Such objects can be cast to an IDataBlock so that they can be used by data block interpreters.

Note: C++ method declarations that use IDataBlock use the _di_IDataBlock type instead. This is a DelphiInterface (see page 3256) wrapper around the IDataBlock interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDataBlock >  _di_IDataBlock;
```

See Also

TCustomDataBlockInterpreter (see page 192)

TStreamedConnection (see page 236)

4.2.1.1 IDataBlock Members

The following tables list the members exposed by IDataBlock.

Public Methods

	Name	Description
	Clear (see page 267)	Removes all data from Memory (see page 272) except for the reserved bytes.
	GetBytesReserved (see page 268)	Returns the value of the BytesReserved (see page 272) property.
	GetMemory (see page 268)	Returns the value of the Memory (see page 272) property.
	GetSignature (see page 268)	Returns the value of the Signature (see page 273) property.
	GetSize (see page 269)	Returns the value of the Size (see page 273) property.
	GetStream (see page 269)	Returns the value of the Stream (see page 274) property.
	IgnoreStream (see page 269)	Informs the data block when its stream is invalidated.
	InitData (see page 270)	Adds data to the data block.
	Read (see page 270)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 271)	Sets the value of the Signature (see page 273) property.
	SetSize (see page 271)	Sets the value of the Size (see page 273) property.
	Write (see page 271)	Writes a specified number of bytes from a buffer.

Public Properties

	Name	Description
R	BytesReserved (see page 272)	Indicates the number of bytes from Memory (see page 272) that are reserved for internal use.
R	Memory (see page 272)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 273)	Indicates the contents of the data block.
	Size (see page 273)	Indicates the size of the data in the memory buffer.
R	Stream (see page 274)	Provides access to an internal stream that contains the data block's memory.

4.2.1.2 IDataBlock Methods

The methods of the IDataBlock class are listed here.

Public Methods

	Name	Description
	Clear (see page 267)	Removes all data from Memory (see page 272) except for the reserved bytes.
	GetBytesReserved (see page 268)	Returns the value of the BytesReserved (see page 272) property.
	GetMemory (see page 268)	Returns the value of the Memory (see page 272) property.
	GetSignature (see page 268)	Returns the value of the Signature (see page 273) property.
	GetSize (see page 269)	Returns the value of the Size (see page 273) property.
	GetStream (see page 269)	Returns the value of the Stream (see page 274) property.
	IgnoreStream (see page 269)	Informs the data block when its stream is invalidated.
	InitData (see page 270)	Adds data to the data block.
	Read (see page 270)	Reads a specified number of bytes into a buffer.
	SetSignature (see page 271)	Sets the value of the Signature (see page 273) property.
	SetSize (see page 271)	Sets the value of the Size (see page 273) property.
	Write (see page 271)	Writes a specified number of bytes from a buffer.

4.2.1.2.1 IDataBlock.Clear Method

Removes all data from Memory (see page 272) except for the reserved bytes.

Pascal

```
procedure Clear; stdcall;
```

C++

```
__fastcall __stdcall Clear();
```

Description

Call Clear when finished reading from Memory (see page 272) or when the information written to Memory (see page 272) has been used. Clear reclaims the resources used to store information that is read from or written to the data block.

See Also

BytesReserved (see page 272)

Memory (see page 272)

4.2.1.2.2 IDataBlock.GetBytesReserved Method

Returns the value of the BytesReserved (see page 272) property.

Pascal

```
function GetBytesReserved: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetBytesReserved();
```

Description

GetBytesReserved is the read implementer for the BytesReserved (see page 272) property.

See Also

BytesReserved (see page 272)

4.2.1.2.3 IDataBlock.GetMemory Method

Returns the value of the Memory (see page 272) property.

Pascal

```
function GetMemory: Pointer; stdcall;
```

C++

```
__fastcall __stdcall void * GetMemory();
```

Description

GetMemory is the read implementer for the Memory (see page 272) property.

See Also

Memory (see page 272)

4.2.1.2.4 IDataBlock.GetSignature Method

Returns the value of the Signature (see page 273) property.

Pascal

```
function GetSignature: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSignature();
```

Description

GetSignature is the read implementer for the Signature (see page 273) property.

See Also

Signature (see page 273)

4.2.1.2.5 IDataBlock.GetSize Method

Returns the value of the Size (see page 273) property.

Pascal

```
function GetSize: Integer; stdcall;
```

C++

```
__fastcall __stdcall int GetSize();
```

Description

GetSize is the read implementer for the Size (see page 273) property.

See Also

Size (see page 273)

4.2.1.2.6 IDataBlock.GetStream Method

Returns the value of the Stream (see page 274) property.

Pascal

```
function GetStream: TStream; stdcall;
```

C++

```
__fastcall __stdcall TStream GetStream();
```

Description

GetStream is the read implementer for the Stream (see page 274) property.

See Also

Stream (see page 274)

4.2.1.2.7 IDataBlock.IgnoreStream Method

Informs the data block when its stream is invalidated.

Pascal

```
procedure IgnoreStream; stdcall;
```

C++

```
__fastcall __stdcall IgnoreStream();
```

Description

Call IgnoreStream after taking responsibility for freeing the component specified by Stream (see page 274). After a call to

IgnoreStream, the data block will not free the memory for its internal Stream (see page 274) component.

See Also

Stream (see page 274)

4.2.1.2.8 IDataBlock.InitData Method

Adds data to the data block.

Pascal

```
function InitData(Data: Pointer; DataLen: Integer; CheckLen: Boolean): Integer; stdcall;
```

C++

```
__fastcall __stdcall int InitData(void * Data, int DataLen, Boolean CheckLen);
```

Description

Data is the data to be added to the data block. It must include a valid signature or InitData raises an exception.

DataLen is the number of bytes referenced by Data.

CheckLen specifies whether InitData should verify that the signature indicates the same amount of data as is indicated by the DataLen parameter.

See Also

Signature (see page 273)

4.2.1.2.9 IDataBlock.Read Method

Reads a specified number of bytes into a buffer.

Pascal

```
function Read(var Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Read( Buffer, int Count);
```

Description

Call Read to copy the contents stored in the memory buffer to the buffer supplied by the Buffer parameter. Read reads up to Count bytes and returns the number of bytes actually read.

The first call to Read begins reading from Memory (see page 272) at the first position after the reserved bytes. Subsequent calls begin reading at the point where the last call to Read left off.

Note: The Count parameter should specify a value less than or equal to Size (see page 273) - BytesReserved (see page 272).

See Also

Write (see page 271)

Memory (see page 272)

Size (see page 273)

BytesReserved (see page 272)

4.2.1.2.10 IDataBlock.SetSignature Method

Sets the value of the Signature (see page 273) property.

Pascal

```
procedure SetSignature(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSignature(int Value);
```

Description

SetSignature is the write implementer for the Signature (see page 273) property.

See Also

Signature (see page 273)

4.2.1.2.11 IDataBlock.SetSize Method

Sets the value of the Size (see page 273) property.

Pascal

```
procedure SetSize(Value: Integer); stdcall;
```

C++

```
__fastcall __stdcall SetSize(int Value);
```

Description

SetSize is the write implementer for the Size (see page 273) property.

See Also

Size (see page 273)

4.2.1.2.12 IDataBlock.Write Method

Writes a specified number of bytes from a buffer.

Pascal

```
function Write(const Buffer; Count: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Write(const  Buffer, int Count);
```

Description

Call Write to copy the contents of the buffer supplied as the Buffer parameter to the next position in Memory (see page 272). Write writes up to Count bytes and returns the number of bytes actually written.

The first call to Write begins writing to Memory (see page 272) at the first position after the reserved bytes. Subsequent calls begin writing at the point where the last call to Write left off.

Note: The Count parameter should specify a value less than or equal to Size (see page 273) - BytesReserved (see page 272).

See Also

Read (see page 270)

Memory (see page 272)

Size (see page 273)

BytesReserved (see page 272)

4.2.1.3 IDataBlock Properties

The properties of the IDataBlock class are listed here.

Public Properties

	Name	Description
	BytesReserved (see page 272)	Indicates the number of bytes from Memory (see page 272) that are reserved for internal use.
	Memory (see page 272)	Points to a buffer from which data is read or to which data is written.
	Signature (see page 273)	Indicates the contents of the data block.
	Size (see page 273)	Indicates the size of the data in the memory buffer.
	Stream (see page 274)	Provides access to an internal stream that contains the data block's memory.

4.2.1.3.1 IDataBlock.BytesReserved Property

Indicates the number of bytes from Memory (see page 272) that are reserved for internal use.

Pascal

```
property BytesReserved: Integer;
```

C++

```
__property int BytesReserved;
```

Description

BytesReserved indicates how many bytes of Memory (see page 272) can't be used for reading and writing. When interpreting the value of the Size (see page 273) property, use BytesReserved to determine how much buffer space is available for reading and writing data.

See Also

Memory (see page 272)

Size (see page 273)

4.2.1.3.2 IDataBlock.Memory Property

Points to a buffer from which data is read or to which data is written.

Pascal

```
property Memory: Pointer;
```

C++

```
__property void * Memory;
```

Description

Memory provides access to the buffer that contains data that is read or written. Memory includes a reserved area (of size BytesReserved (see page 272)) for internal information such as the value of Signature (see page 273),

Note: When implementing the Receive and Send methods of the ITransport (see page 278) interface, component writers should read directly from and write directly to Memory, rather than using the Read (see page 270) and Write (see page 271)

methods. Otherwise, the information in the reserved area will not be sent to or from the application server.

See Also

Size (see page 273)

ITransport (see page 278)

BytesReserved (see page 272)

Signature (see page 273)

Stream (see page 274)

4.2.1.3.3 IDataBlock.Signature Property

Indicates the contents of the data block.

Pascal

```
property Signature: Integer;
```

C++

```
__property int Signature;
```

Description

Write (see page 271) Signature values to indicate what type of data is written to the data block. Read (see page 270) Signature to determine what type of data can be read from the data block.

The value of signature is stored somewhere in the first BytesReserved (see page 272) bytes of Memory (see page 272).

See Also

BytesReserved (see page 272)

Memory (see page 272)

4.2.1.3.4 IDataBlock.Size Property

Indicates the size of the data in the memory buffer.

Pascal

```
property Size: Integer;
```

C++

```
__property int Size;
```

Description

Size is the number of bytes available in Memory (see page 272), excluding the reserved bytes. Set Size to ensure that Memory (see page 272) is large enough to contain the data that the data block will read or write.

See Also

BytesReserved (see page 272)

Memory (see page 272)

Signature (see page 273)

4.2.1.3.5 IDataBlock.Stream Property

Provides access to an internal stream that contains the data block's memory.

Pascal

```
property Stream: TStream;
```

C++

```
__property TStream Stream;
```

Description

Stream allows component writers more convenient access to the data block's memory buffer. This memory buffer contains the data that is read or written.

See Also

Memory (see page 272)

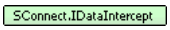
IgnoreStream (see page 269)

TMemoryStream

4.2.2 IDataIntercept Interface

IDataIntercept is the interface used by TSocketConnection (see page 221) components to transform data that is communicated over the connection.

Class Hierarchy



Pascal

```
IDataIntercept = interface;
```

C++

```
__interface IDataIntercept;
```

File

SConnect

Description

IDataIntercept is the interface implemented by data packet interceptors. A data packet interceptor is a COM object that implements IDataIntercept to transform data that is accessed through an IDataBlock (see page 266) interface. TSocketConnection (see page 221) can use a data packet interceptor before messages are sent and after responses are received.

Developers can create a data packet interceptor to encrypt and decrypt data or to compress and decompress messages. The transformation must leave the reserved bytes of the data blocks unchanged or the socket connection component won't be able to communicate properly.

Note: When creating a data packet interceptor in Delphi, use TPacketInterceptFactory (see page 219) as its class factory.

Once you have created and registered a data packet interceptor, Set the InterceptName or InterceptGUID property of a socket connection component to identify the data packet interceptor. This tells the socket connection to instantiate the interceptor and use it to transform messages before they are sent and after they are received. On the server side, right click on the Tray Icon for

scktsrvr.exe and assign the interceptor using the properties dialog.

See Also

IDataBlock (see page 266)

TStreamedConnection (see page 236)

TPacketInterceptFactory (see page 219)

Specifying a Connection Using Sockets

4.2.2.1 IDataIntercept Members

The following tables list the members exposed by IDataIntercept.

Public Methods

	Name	Description
	DataIn (see page 275)	Undoes the transformation applied by the DataOut (see page 276) method.
	DataOut (see page 276)	Returns the value of the BytesReserved property.

4.2.2.2 IDataIntercept Methods

The methods of the IDataIntercept class are listed here.

Public Methods

	Name	Description
	DataIn (see page 275)	Undoes the transformation applied by the DataOut (see page 276) method.
	DataOut (see page 276)	Returns the value of the BytesReserved property.

4.2.2.2.1 IDataIntercept.DataIn Method

Undoes the transformation applied by the DataOut (see page 276) method.

Pascal

```
procedure DataIn(const Data: IDataBlock); stdcall;
```

C++

```
__fastcall __stdcall DataIn(const IDataBlock Data);
```

Description

Call DataIn to undo any transformations that were applied to information received by the connection. For example, use this method to decrypt or decompress data.

The Data parameter is an IDataBlock (see page 266) interface to the data that was received by the socket connection component. Use the properties and methods of IDataBlock (see page 266) to reverse the transformation that was applied by the DataOut (see page 276) method.

Warning: Do not transform the reserved bytes in the data block.

See Also

DataOut (see page 276)

BytesReserved (see page 272)

4.2.2.2.2 IDataIntercept.DataOut Method

Returns the value of the BytesReserved property.

Pascal

```
procedure DataOut(const Data: IDataBlock); stdcall;
```

C++

```
__fastcall __stdcall DataOut(const IDataBlock Data);
```

Description

Call DataOut to transform data before it is sent using a connection component. For example, use this method to encrypt or compress data.

The Data parameter is an IDataBlock (see page 266) interface to the data that is about to be sent by the socket connection component. Use the properties and methods of IDataBlock (see page 266) to transform the data.

Warning: Do not transform the reserved bytes in the data block.

See Also

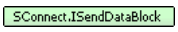
DataIn (see page 275)

BytesReserved (see page 272)

4.2.3 ISendDataBlock Interface

ISendDataBlock is the interface used by data block interpreters to send messages to an application server after they are marshaled.

Class Hierarchy



Pascal

```
ISendDataBlock = interface;
```

C++

```
__interface ISendDataBlock;
```

File

SConnect

Description

Streamed connection components implement ISendDataBlock to allow a data block interpreter (which handles marshaling of COM interface calls) to send interface calls to the application server.

ISendDataBlock is an interface, and as such can't be directly instantiated. Instead, streamed connection components (which implement ISendDataBlock) can be cast to an ISendDataBlock (Delphi) or have a conversion operator to obtain this interface (C++).

Note: TDataBlockInterpreter (see page 207) uses _di_ISendDataBlock in place of ISendDataBlock. This is a DelphiInterface (see page 3256) wrapper around the ISendDataBlock interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< ISendDataBlock > _di_ISendDataBlock;
```

See Also

TStreamedConnection (see page 236)

TCustomDataBlockInterpreter (see page 192)

IDataBlock (see page 266)

4.2.3.1 ISendDataBlock Members

The following tables list the members exposed by ISendDataBlock.

Public Methods

	Name	Description
	Send (see page 277)	Sends information retrieved from an IDataBlock (see page 266) interface to the application server.

4.2.3.2 ISendDataBlock Methods

The methods of the ISendDataBlock class are listed here.

Public Methods

	Name	Description
	Send (see page 277)	Sends information retrieved from an IDataBlock (see page 266) interface to the application server.

4.2.3.2.1 ISendDataBlock.Send Method

Sends information retrieved from an IDataBlock (see page 266) interface to the application server.

Pascal

```
function Send(const Data: IDataBlock; WaitForResult: Boolean): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Send(const IDataBlock Data, Boolean WaitForResult);
```

Description

ISendDataBlock (see page 276) dispatches information to an application server. The information must be available as the Memory property of an IDataBlock (see page 266) interface.

The WaitForResult parameter indicates whether the caller requires a response when the information is sent. If WaitForResult is true, Send does not return until a response is received from the application server. This response is returned as an IDataBlock (see page 266) interface (_di_IDataBlock). If WaitForResult is false, Send may still return a response, but only if the application server responds immediately.

See Also

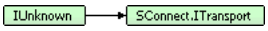
Memory (see page 272)

Send (see page 244)

4.2.4 ITransport Interface

ITransport is the interface used by TStreamedConnection (see page 236) to connect to, read from, and write to a remote application server.

Class Hierarchy



Pascal

```
ITransport = interface(IUnknown);
```

C++

```
__interface ITransport : public IUnknown;
```

File

SConnect

Description

Streamed connection components use ITransport to connect to an application server and to send or receive data over that connection. ITransport is responsible only for managing the communication protocols. It does not provide any support for marshaling or storing information.

ITransport is an interface, and as such can't be directly instantiated. Instead, objects that implement the ITransport properties and methods include ITransport in their class definition (Delphi) or are derived from ITransport (C++). Such objects can be cast to an ITransport so that they can be used by streamed connection components.

Note: C++ method declarations that use ITransport use the _di_ITransport type instead. This is a DelphiInterface (see page 3256) wrapper around the ITransport interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< ITransport> _di_ITransport;
```

See Also

TStreamedConnection (see page 236)

4.2.4.1 ITransport Members

The following tables list the members exposed by ITransport.

Public Methods

	Name	Description
	GetConnected (see page 279)	Returns the value of the Connected (see page 281) property.
	GetWaitEvent (see page 279)	Returns the handle for an event object that is managed by the object that implements ITransport (see page 278).
	Receive (see page 280)	Returns an IDataBlock (see page 266) interface that can be used to read information received from the application server.
	Send (see page 280)	Sends information retrieved from an IDataBlock (see page 266) interface to the application server.
	SetConnected (see page 280)	Sets the value of the Connected (see page 281) property.

Public Properties

	Name	Description
	Connected (see page 281)	Indicates whether the connection to the application server is established.

4.2.4.2 ITransport Methods

The methods of the ITransport class are listed here.

Public Methods

	Name	Description
	GetConnected (see page 279)	Returns the value of the Connected (see page 281) property.
	GetWaitEvent (see page 279)	Returns the handle for an event object that is managed by the object that implements ITransport (see page 278).
	Receive (see page 280)	Returns an IDataBlock (see page 266) interface that can be used to read information received from the application server.
	Send (see page 280)	Sends information retrieved from an IDataBlock (see page 266) interface to the application server.
	SetConnected (see page 280)	Sets the value of the Connected (see page 281) property.

4.2.4.2.1 ITransport.GetConnected Method

Returns the value of the Connected (see page 281) property.

Pascal

```
function GetConnected: Boolean; stdcall;
```

C++

```
__fastcall __stdcall Boolean GetConnected();
```

Description

GetConnected is the read property implementer for the Connected (see page 281) property.

See Also

Connected (see page 281)

4.2.4.2.2 ITransport.GetWaitEvent Method

Returns the handle for an event object that is managed by the object that implements ITransport (see page 278).

Pascal

```
function GetWaitEvent: THandle; stdcall;
```

C++

```
__fastcall __stdcall THandle GetWaitEvent();
```

Description

Streamed connection components perform all reading from and writing to the application server using a separate execution thread. This limits the impact that communication with the application server has on the speed of the client application, because execution does not need to wait while messages are marshaled and sent or received. The streamed connection component needs to coordinate its use of the communication thread. To do this, it uses an event object that is a member of the object that implements ITransport (see page 278).

GetWaitEvent returns a handle to this event object. When implementing an object that supports ITransport (see page 278),

create a TEvent object as a member and return its handle as the return value of GetWaitEvent. Do not use the event object for internal purposes or it will interfere with the streamed connection's use of the event.

See Also

TEvent (see page 3133)

4.2.4.2.3 ITransport.Receive Method

Returns an IDataBlock (see page 266) interface that can be used to read information received from the application server.

Pascal

```
function Receive(WaitForInput: Boolean; Context: Integer): IDataBlock; stdcall;
```

C++

```
__fastcall __stdcall IDataBlock Receive(Boolean WaitForInput, int Context);
```

Description

Receive creates a COM object that implements the IDataBlock (see page 266) interface and reads information from the application server into its Memory property.

If WaitForInput is true, Receive halts execution until the application server sends the expected information. If WaitForInput is false, Receive can fail (reading no information) if the application server does not send any information immediately (or after a brief time-out period).

Context is transport-specific data that can be used when implementing a class that supports ITransport (see page 278). Its value is the return value of the Send (see page 280) method.

See Also

Memory (see page 272)

Send (see page 280)

4.2.4.2.4 ITransport.Send Method

Sends information retrieved from an IDataBlock (see page 266) interface to the application server.

Pascal

```
function Send(const Data: IDataBlock): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Send(const IDataBlock Data);
```

Description

Send sends the entire contents of the Data parameter's Memory property to the application server. It returns a transport-specific value that is used by the next call to the Receive (see page 280) method. For example, HTTP transport classes return the handle for the Internet connection.

See Also

Memory (see page 272)

Receive (see page 280)

4.2.4.2.5 ITransport.SetConnected Method

Sets the value of the Connected (see page 281) property.

Pascal

```
procedure SetConnected(Value: Boolean); stdcall;
```

C++

```
__fastcall __stdcall SetConnected(Boolean Value);
```

Description

SetConnected is the write implementer for the Connected (see page 281) property.

4.2.4.3 ITransport Properties

The properties of the ITransport class are listed here.

Public Properties

	Name	Description
	Connected (see page 281)	Indicates whether the connection to the application server is established.

4.2.4.3.1 ITransport.Connected Property

Indicates whether the connection to the application server is established.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Set Connected to true to connect to the application server. Set Connected to false to close a connection.

Read (see page 3634) the Connected property to verify that the connection is open before reading from or writing to the application server.

See Also

Receive (see page 280)

Send (see page 280)

4.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetPacketInterceptorList (see page 282)	This is function SConnect.GetPacketInterceptorList.
	LoadWinSock2 (see page 282)	This is function SConnect.LoadWinSock2.

4.3.1 SConnect.GetPacketInterceptorList Function

Pascal

```
procedure GetPacketInterceptorList(List: TStringList);
```

C++

```
GetPacketInterceptorList(TStringList List);
```

File

SConnect

Description

This is function SConnect.GetPacketInterceptorList.

4.3.2 SConnect.LoadWinSock2 Function

Pascal

```
function LoadWinSock2: Boolean;
```

C++

```
Boolean LoadWinSock2();
```

File

SConnect

Description

This is function SConnect.LoadWinSock2.

4.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TVarFlag (see page 283)	This is record SConnect.TVarFlag.

4.4.1 SConnect.TVarFlag Enumeration

Pascal

```
TVarFlag = (
  vfByRef,
  vfVariant
);
```

C++

```
enum TVarFlag {
  vfByRef,
  vfVariant
};
```

File

SConnect

Description

This is record SConnect.TVarFlag.

4.5 Types

The following table lists types in this documentation.

Types

Name	Description
PIntArray (see page 284)	This is type SConnect.PIntArray.
PVariantArray (see page 284)	This is type SConnect.PVariantArray.
TIntArray (see page 285)	This is type SConnect.TIntArray.
TVarFlags (see page 285)	This is type SConnect.TVarFlags.
TVariantArray (see page 285)	This is type SConnect.TVariantArray.

4.5.1 SConnect.PIntArray Type

Pascal

```
PIntArray = ^TIntArray;
```

C++

```
^TIntArray PIntArray;
```

File

SConnect

Description

This is type SConnect.PIntArray.

4.5.2 SConnect.PVariantArray Type

Pascal

```
PVariantArray = ^TVariantArray;
```

C++

```
^TVariantArray PVariantArray;
```

File

SConnect

Description

This is type SConnect.PVariantArray.

4.5.3 SConnect.TIntArray Type

Pascal

```
TIntArray = array[0..0] of Integer;
```

C++

```
array[0..0] of Integer TIntArray;
```

File

SConnect

Description

This is type SConnect.TIntArray.

4.5.4 SConnect.TVarFlags Type

Pascal

```
TVarFlags = set of TVarFlag;
```

C++

```
TVarFlag TVarFlags;
```

File

SConnect

Description

This is type SConnect.TVarFlags.

4.5.5 SConnect.TVariantArray Type

Pascal

```
TVariantArray = array[0..0] of OleVariant;
```

C++

```
array[0..0] of OleVariant TVariantArray;
```

File

SConnect

Description

This is type SConnect.TVariantArray.

4.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
WSACloseEvent (see page 286)	This is variable SConnect.WSACloseEvent.
WSACreateEvent (see page 286)	This is variable SConnect.WSACreateEvent.
WSAEventSelect (see page 287)	This is variable SConnect.WSAEventSelect.
WSAResetEvent (see page 287)	This is variable SConnect.WSAResetEvent.

4.6.1 SConnect.WSACloseEvent Variable

Pascal

```
WSACloseEvent: function(hEvent: THandle): Boolean stdcall;
```

C++

```
function(hEvent: THandle): Boolean stdcall WSACloseEvent;
```

File

SConnect

Description

This is variable SConnect.WSACloseEvent.

4.6.2 SConnect.WSACreateEvent Variable

Pascal

```
WSACreateEvent: function: THandle stdcall;
```

C++

```
function: THandle stdcall WSACreateEvent;
```

File

SConnect

Description

This is variable SConnect.WSACreateEvent.

4.6.3 SConnect.WSAEventSelect Variable

Pascal

```
WSAEventSelect: function(s: TSocket; hEventObject: THandle; lNetworkEvents: Integer): Integer
stdcall;
```

C++

```
function(s: TSocket; hEventObject: THandle; lNetworkEvents: Integer): Integer stdcall
WSAEventSelect;
```

File

SConnect

Description

This is variable SConnect.WSAEventSelect.

4.6.4 SConnect.WSAResetEvent Variable

Pascal

```
WSAResetEvent: function(hEvent: THandle): Boolean stdcall;
```

C++

```
function(hEvent: THandle): Boolean stdcall WSAResetEvent;
```

File

SConnect

Description

This is variable SConnect.WSAResetEvent.

4.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
asCreateObject (see page 288)	This is constant SConnect.asCreateObject.
asError (see page 288)	This is constant SConnect.asError.
asFreeObject (see page 289)	This is constant SConnect.asFreeObject.
asGetAppServers (see page 289)	This is constant SConnect.asGetAppServers.
asGetGUID (see page 289)	This is constant SConnect.asGetGUID.
asGetID (see page 290)	This is constant SConnect.asGetID.
asGetServers (see page 290)	This is constant SConnect.asGetServers.
asInvoke (see page 290)	This is constant SConnect.asInvoke.
asMask (see page 291)	This is constant SConnect.asMask.
asSoapCommand (see page 291)	This is constant SConnect.asSoapCommand.
CallSig (see page 291)	This is constant SConnect.CallSig.
ResultSig (see page 292)	This is constant SConnect.ResultSig.
THREAD_EXCEPTION (see page 292)	This is constant SConnect.THREAD_EXCEPTION.
THREAD_RECEIVEDSTREAM (see page 292)	This is constant SConnect.THREAD_RECEIVEDSTREAM.
THREAD_REPLACETRANSPORT (see page 293)	This is constant SConnect.THREAD_REPLACETRANSPORT.
THREAD_SENDNOTIFY (see page 293)	This is constant SConnect.THREAD_SENDNOTIFY.
THREAD_SENDSTREAM (see page 293)	This is constant SConnect.THREAD_SENDSTREAM.

4.7.1 SConnect.asCreateObject Constant

Pascal

```
asCreateObject = $04;
```

C++

```
asCreateObject = 0x04;
```

File

SConnect

Description

This is constant SConnect.asCreateObject.

4.7.2 SConnect.asError Constant

Pascal

```
asError = $01;
```

C++

```
asError = 0x01;
```

File

SConnect

Description

This is constant SConnect.asError.

4.7.3 SConnect.asFreeObject Constant

Pascal

```
asFreeObject = $05;
```

C++

```
asFreeObject = 0x05;
```

File

SConnect

Description

This is constant SConnect.asFreeObject.

4.7.4 SConnect.asGetAppServers Constant

Pascal

```
asGetAppServers = $12;
```

C++

```
asGetAppServers = 0x12;
```

File

SConnect

Description

This is constant SConnect.asGetAppServers.

4.7.5 SConnect.asGetGUID Constant

Pascal

```
asGetGUID = $11;
```

C++

```
asGetGUID = 0x11;
```

File

SConnect

Description

This is constant SConnect.asGetGUID.

4.7.6 SConnect.asGetID Constant

Pascal

```
asGetID = $03;
```

C++

```
asGetID = 0x03;
```

File

SConnect

Description

This is constant SConnect.asGetID.

4.7.7 SConnect.asGetServers Constant

Pascal

```
asGetServers = $10;
```

C++

```
asGetServers = 0x10;
```

File

SConnect

Description

This is constant SConnect.asGetServers.

4.7.8 SConnect.asInvoke Constant

Pascal

```
asInvoke = $02;
```

C++

```
asInvoke = 0x02;
```

File

SConnect

Description

This is constant SConnect.asInvoke.

4.7.9 SConnect.asMask Constant

Pascal

```
asMask = $FF;
```

C++

```
asMask = 0xFF;
```

File

SConnect

Description

This is constant SConnect.asMask.

4.7.10 SConnect.asSoapCommand Constant

Pascal

```
asSoapCommand = $14;
```

C++

```
asSoapCommand = 0x14;
```

File

SConnect

Description

This is constant SConnect.asSoapCommand.

4.7.11 SConnect.CallSig Constant

Pascal

```
CallSig = $DA00;
```

C++

```
CallSig = 0xDA00;
```

File

SConnect

Description

This is constant SConnect.CallSig.

4.7.12 SConnect.ResultSig Constant

Pascal

```
ResultSig = $DB00;
```

C++

```
ResultSig = 0xDB00;
```

File

SConnect

Description

This is constant SConnect.ResultSig.

4.7.13 SConnect.THREAD_EXCEPTION Constant

Pascal

```
THREAD_EXCEPTION = THREAD_RECEIVEDSTREAM + 1;
```

C++

```
THREAD_EXCEPTION = THREAD_RECEIVEDSTREAM + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_EXCEPTION.

4.7.14 SConnect.THREAD_RECEIVEDSTREAM Constant

Pascal

```
THREAD_RECEIVEDSTREAM = THREAD_SENDSTREAM + 1;
```

C++

```
THREAD_RECEIVEDSTREAM = THREAD_SENDSTREAM + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_RECEIVEDSTREAM.

4.7.15 SConnect.THREAD_REPLACETRANSPORT Constant

Pascal

```
THREAD_REPLACETRANSPORT = THREAD_SENDNOTIFY + 1;
```

C++

```
THREAD_REPLACETRANSPORT = THREAD_SENDNOTIFY + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_REPLACETRANSPORT.

4.7.16 SConnect.THREAD_SENDNOTIFY Constant

Pascal

```
THREAD_SENDNOTIFY = THREAD_EXCEPTION + 1;
```

C++

```
THREAD_SENDNOTIFY = THREAD_EXCEPTION + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_SENDNOTIFY.

4.7.17 SConnect.THREAD_SENDSTREAM Constant

Pascal

```
THREAD_SENDSTREAM = WM_USER + 1;
```

C++

```
THREAD_SENDSTREAM = WM_USER + 1;
```

File

SConnect

Description

This is constant SConnect.THREAD_SENDSTREAM.

5 ScreenTips Namespace

This is namespace ScreenTips.

Topics

Name	Description
Classes (see page 297)	The following table lists classes in this documentation.
Interfaces (see page 361)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 363)	The following table lists structs, records, enums in this documentation.
Types (see page 366)	The following table lists types in this documentation.
Constants (see page 369)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAllocatedActionsException (see page 297)	This is class ScreenTips.EAllocatedActionsException.
	TAllocatedActions (see page 297)	This is class ScreenTips.TAllocatedActions.
	TAssociateItem (see page 301)	This is class ScreenTips.TAssociateItem.
	TAssociateItemList (see page 302)	This is class ScreenTips.TAssociateItemList.
	TCustomScreenTipsManager (see page 305)	This is class ScreenTips.TCustomScreenTipsManager.
	TCustomScreenTipsPopup (see page 316)	This is class ScreenTips.TCustomScreenTipsPopup.
	TDisplayOffset (see page 324)	This is class ScreenTips.TDisplayOffset.
	TScreenTipBorderStyle (see page 326)	This is class ScreenTips.TScreenTipBorderStyle.
	TScreenTipItem (see page 328)	This is class ScreenTips.TScreenTipItem.
	TScreenTipsCollection (see page 334)	This is class ScreenTips.TScreenTipsCollection.
	TScreenTipsManager (see page 337)	This is class ScreenTips.TScreenTipsManager.
	TScreenTipsPopup (see page 343)	This is class ScreenTips.TScreenTipsPopup.
	TScreenTipsWindow (see page 351)	This is class ScreenTips.TScreenTipsWindow.

Constants

Name	Description
cDefaultBorderColor (see page 369)	This is constant ScreenTips.cDefaultBorderColor.
cDefaultEndColor (see page 369)	This is constant ScreenTips.cDefaultEndColor.
cDefaultStartColor (see page 370)	This is constant ScreenTips.cDefaultStartColor.
cScreenTipHeaderSpacing (see page 370)	This is constant ScreenTips.cScreenTipHeaderSpacing.
cScreenTipMinimumHeight (see page 370)	This is constant ScreenTips.cScreenTipMinimumHeight.
cScreenTipTextAndImageWidth (see page 371)	This is constant ScreenTips.cScreenTipTextAndImageWidth.
cScreenTipTextOnlyWidth (see page 371)	This is constant ScreenTips.cScreenTipTextOnlyWidth.

Interfaces

	Name	Description
	IScreenTipItem (see page 361)	This is class ScreenTips.IScreenTipItem.

Structs, Records, Enums

	Name	Description
	TDescriptionType (see page 363)	This is record ScreenTips.TDescriptionType.
	TImageAlign (see page 363)	This is record ScreenTips.TImageAlign.
	TItemImageAlign (see page 364)	This is record ScreenTips.TItemImageAlign.
	TScreenTipOption (see page 364)	This is record ScreenTips.TScreenTipOption.
	TScreenTipPopupType (see page 365)	This is record ScreenTips.TScreenTipPopupType.
	TTipBorderStyle (see page 365)	This is record ScreenTips.TTipBorderStyle.

Types

Name	Description
TCreateScreenTipEvent (see page 366)	This is type ScreenTips.TCreateScreenTipEvent.
TDuplicateCreateErrorEvent (see page 366)	This is type ScreenTips.TDuplicateCreateErrorEvent.
TScreenTipOptions (see page 367)	This is type ScreenTips.TScreenTipOptions.
TScreenTipsWindowClass (see page 367)	This is type ScreenTips.TScreenTipsWindowClass.
TScreenTipWinClass (see page 367)	This is type ScreenTips.TScreenTipWinClass.
TShowScreenTipEvent (see page 368)	This is type ScreenTips.TShowScreenTipEvent.

5.1 Classes

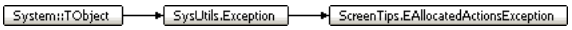
The following table lists classes in this documentation.

Classes

	Name	Description
	EAllocatedActionsException (see page 297)	This is class ScreenTips.EAllocatedActionsException.
	TAllocatedActions (see page 297)	This is class ScreenTips.TAllocatedActions.
	TAssociateItem (see page 301)	This is class ScreenTips.TAssociateItem.
	TAssociateItemList (see page 302)	This is class ScreenTips.TAssociateItemList.
	TCustomScreenTipsManager (see page 305)	This is class ScreenTips.TCustomScreenTipsManager.
	TCustomScreenTipsPopup (see page 316)	This is class ScreenTips.TCustomScreenTipsPopup.
	TDisplayOffset (see page 324)	This is class ScreenTips.TDisplayOffset.
	TScreenTipBorderStyle (see page 326)	This is class ScreenTips.TScreenTipBorderStyle.
	TScreenTipItem (see page 328)	This is class ScreenTips.TScreenTipItem.
	TScreenTipsCollection (see page 334)	This is class ScreenTips.TScreenTipsCollection.
	TScreenTipsManager (see page 337)	This is class ScreenTips.TScreenTipsManager.
	TScreenTipsPopup (see page 343)	This is class ScreenTips.TScreenTipsPopup.
	TScreenTipsWindow (see page 351)	This is class ScreenTips.TScreenTipsWindow.

5.1.1 EAllocatedActionsException Class

Class Hierarchy



Pascal

```
EAllocatedActionsException = class(Exception);
```

C++

```
class EAllocatedActionsException : public Exception;
```

File

ScreenTips

Description

This is class ScreenTips.EAllocatedActionsException.

5.1.2 TAllocatedActions Class

Class Hierarchy

Pascal

```
TAllocatedActions = class;
```

C++

```
class TAllocatedActions;
```

File

ScreenTips

Description

This is class ScreenTips.TAllocatedActions.

5.1.2.1 TAllocatedActions Members

The following tables list the members exposed by TAllocatedActions.

Public Methods

	Name	Description
	AddAction (see page 299)	This is AddAction, a member of class TAllocatedActions.
	Clear (see page 299)	This is Clear, a member of class TAllocatedActions.
	Create (see page 298)	This is Create, a member of class TAllocatedActions.
	Delete (see page 299)	This is Delete, a member of class TAllocatedActions.
V	Destroy (see page 298)	This is Destroy, a member of class TAllocatedActions.
	Exists (see page 299)	This is Exists, a member of class TAllocatedActions.
	Find (see page 300)	This is Find, a member of class TAllocatedActions.
	GetItem (see page 300)	This is GetItem, a member of class TAllocatedActions.

Public Properties

	Name	Description
R	Count (see page 300)	This is Count, a member of class TAllocatedActions.

5.1.2.2 TAllocatedActions.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TAllocatedActions();
```

Description

This is Create, a member of class TAllocatedActions.

5.1.2.3 TAllocatedActions.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAllocatedActions();
```

Description

This is Destroy, a member of class TAllocatedActions.

5.1.2.4 TAllocatedActions Methods

The methods of the TAllocatedActions class are listed here.

Public Methods

	Name	Description
	AddAction (see page 299)	This is AddAction, a member of class TAllocatedActions.
	Clear (see page 299)	This is Clear, a member of class TAllocatedActions.
	Delete (see page 299)	This is Delete, a member of class TAllocatedActions.
	Exists (see page 299)	This is Exists, a member of class TAllocatedActions.
	Find (see page 300)	This is Find, a member of class TAllocatedActions.
	GetItem (see page 300)	This is GetItem, a member of class TAllocatedActions.

5.1.2.4.1 TAllocatedActions.AddAction Method

Pascal

```
procedure AddAction(const aName: string; aScreenTipItem: TObject);
```

C++

```
__fastcall AddAction(const UnicodeString aName, TObject * aScreenTipItem);
```

Description

This is AddAction, a member of class TAllocatedActions.

5.1.2.4.2 TAllocatedActions.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TAllocatedActions.

5.1.2.4.3 TAllocatedActions.Delete Method

Pascal

```
function Delete(const aName: string; out aIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean Delete(const UnicodeString aName, int aIndex);
```

Description

This is Delete, a member of class TAllocatedActions.

5.1.2.4.4 TAllocatedActions.Exists Method

Pascal

```
function Exists(const aName: string): Boolean;
```

C++

```
__fastcall Boolean Exists(const UnicodeString aName);
```

Description

This is Exists, a member of class TAllocatedActions.

5.1.2.4.5 TAllocatedActions.Find Method

Pascal

```
function Find(const aName: string; out aIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean Find(const UnicodeString aName, int aIndex);
```

Description

This is Find, a member of class TAllocatedActions.

5.1.2.4.6 TAllocatedActions.GetItem Method

Pascal

```
function GetItem(const aName: string): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetItem(const UnicodeString aName);
```

Description

This is GetItem, a member of class TAllocatedActions.

5.1.2.5 TAllocatedActions Properties

The properties of the TAllocatedActions class are listed here.

Public Properties

	Name	Description
	Count (see page 300)	This is Count, a member of class TAllocatedActions.

5.1.2.5.1 TAllocatedActions.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TAllocatedActions.

5.1.3 TAssociateItem Class

Class Hierarchy

Pascal

```
TAssociateItem = class;
```

C++

```
class TAssociateItem;
```

File

ScreenTips

Description

This is class ScreenTips.TAssociateItem.

5.1.3.1 TAssociateItem Members

The following tables list the members exposed by TAssociateItem.

Public Methods

	Name	Description
	Create (see page 301)	This is Create, a member of class TAssociateItem.
	Destroy (see page 301)	This is Destroy, a member of class TAssociateItem.

Public Properties

	Name	Description
	Associate (see page 302)	This is Associate, a member of class TAssociateItem.
	PopupTip (see page 302)	This is PopupTip, a member of class TAssociateItem.

5.1.3.2 TAssociateItem.Create Constructor

Pascal

```
constructor Create(APopupTip: TControl; AAssociate: TControl);
```

C++

```
__fastcall TAssociateItem(TControl * APopupTip, TControl * AAssociate);
```

Description

This is Create, a member of class TAssociateItem.

5.1.3.3 TAssociateItem.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAssociateItem();
```

Description

This is Destroy, a member of class TAssociateItem.

5.1.3.4 TAssociateItem Properties

The properties of the TAssociateItem class are listed here.

Public Properties

	Name	Description
	Associate (see page 302)	This is Associate, a member of class TAssociateItem.
	PopupTip (see page 302)	This is PopupTip, a member of class TAssociateItem.

5.1.3.4.1 TAssociateItem.Associate Property

Pascal

```
property Associate: TControl;
```

C++

```
__property TControl * Associate;
```

Description

This is Associate, a member of class TAssociateItem.

5.1.3.4.2 TAssociateItem.PopupTip Property

Pascal

```
property PopupTip: TControl;
```

C++

```
__property TControl * PopupTip;
```

Description

This is PopupTip, a member of class TAssociateItem.

5.1.4 TAssociateItemList Class

Class Hierarchy

Pascal

```
TAssociateItemList = class;
```

C++

```
class TAssociateItemList;
```

File

ScreenTips

Description

This is class ScreenTips.TAssociateItemList.

5.1.4.1 TAssociateItemList Members

The following tables list the members exposed by TAssociateItemList.

Public Methods

	Name	Description
	Add (see page 304)	This is Add, a member of class TAssociateItemList.
	Create (see page 303)	This is Create, a member of class TAssociateItemList.
	Destroy (see page 303)	This is Destroy, a member of class TAssociateItemList.
	Find (see page 304)	This is Find, a member of class TAssociateItemList.
	Get (see page 304)	This is Get, a member of class TAssociateItemList.
	Remove (see page 304)	This is Remove, a member of class TAssociateItemList.

5.1.4.2 TAssociateItemList.Create Constructor

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TAssociateItemList();
```

Description

This is Create, a member of class TAssociateItemList.

5.1.4.3 TAssociateItemList.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAssociateItemList();
```

Description

This is Destroy, a member of class TAssociateItemList.

5.1.4.4 TAssociateItemList Methods

The methods of the TAssociateItemList class are listed here.

Public Methods

	Name	Description
	Add (see page 304)	This is Add, a member of class TAssociateItemList.

	Find (see page 304)	This is Find, a member of class TAssociateItemList.
	Get (see page 304)	This is Get, a member of class TAssociateItemList.
	Remove (see page 304)	This is Remove, a member of class TAssociateItemList.

5.1.4.4.1 TAssociateItemList.Add Method

Pascal

```
procedure Add(Popup: TControl; Associate: TControl);
```

C++

```
__fastcall Add(TControl * Popup, TControl * Associate);
```

Description

This is Add, a member of class TAssociateItemList.

5.1.4.4.2 TAssociateItemList.Find Method

Pascal

```
function Find(Associate: TControl): Integer;
```

C++

```
__fastcall int Find(TControl * Associate);
```

Description

This is Find, a member of class TAssociateItemList.

5.1.4.4.3 TAssociateItemList.Get Method

Pascal

```
function Get(const Idx: Integer): TAssociateItem;
```

C++

```
__fastcall TAssociateItem Get(const int Idx);
```

Description

This is Get, a member of class TAssociateItemList.

5.1.4.4.4 TAssociateItemList.Remove Method

Pascal

```
procedure Remove(Associate: TControl);
```

C++

```
__fastcall Remove(TControl * Associate);
```

Description

This is Remove, a member of class TAssociateItemList.

5.1.5 TCustomScreenTipsManager Class

Class Hierarchy

Pascal

```
TCustomScreenTipsManager = class(TComponent);
```

C++

```
class TCustomScreenTipsManager : public TComponent;
```

File

ScreenTips

Description

This is class ScreenTips.TCustomScreenTipsManager.

5.1.5.1 TCustomScreenTipsManager Members

The following tables list the members exposed by TCustomScreenTipsManager.

Protected Methods

	Name	Description
	DoCreateScreenTip (see page 309)	This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.
	DoDuplicateCreateError (see page 309)	This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.
	DoGetScreenTipClass (see page 309)	This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.
	DoScreenTipsLoaded (see page 309)	This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	DoShow (see page 309)	This is DoShow, a member of class TCustomScreenTipsManager.
	DoShowScreenTip (see page 310)	This is DoShowScreenTip, a member of class TCustomScreenTipsManager.
	Loaded (see page 310)	Initializes the component after the form file has been read into memory.
	Notification (see page 311)	Forwards notification messages to all owned components.
	UpdateActionInfo (see page 312)	This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

Public Events

	Name	Description
	OnCreateScreenTip (see page 315)	This is OnCreateScreenTip, a member of class TCustomScreenTipsManager.
	OnDuplicateCreateError (see page 315)	This is OnDuplicateCreateError, a member of class TCustomScreenTipsManager.
	OnGetScreenTipWinClass (see page 316)	This is OnGetScreenTipWinClass, a member of class TCustomScreenTipsManager.
	OnScreenTipsLoaded (see page 316)	This is OnScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	OnShowScreenTip (see page 316)	This is OnShowScreenTip, a member of class TCustomScreenTipsManager.

Public Methods

	Name	Description
	AddPopup (see page 308)	This is AddPopup, a member of class TCustomScreenTipsManager.
	ClearScreenTips (see page 308)	This is ClearScreenTips, a member of class TCustomScreenTipsManager.
	Create (see page 306)	Allocates memory and constructs a safely initialized instance of a component.
	CreateScreenTip (see page 308)	This is CreateScreenTip, a member of class TCustomScreenTipsManager.
	Destroy (see page 307)	Disposes of the component and its owned components.
	GenerateScreenTips (see page 310)	This is GenerateScreenTips, a member of class TCustomScreenTipsManager.

	GetScreenTip (see page 310)	This is GetScreenTip, a member of class TCustomScreenTipsManager.
	RemovePopup (see page 311)	This is RemovePopup, a member of class TCustomScreenTipsManager.
	RemoveScreenTip (see page 312)	This is RemoveScreenTip, a member of class TCustomScreenTipsManager.
	UpdateOptions (see page 312)	This is UpdateOptions, a member of class TCustomScreenTipsManager.

Public Properties

	Name	Description
	BorderStyle (see page 313)	This is BorderStyle, a member of class TCustomScreenTipsManager.
	DisplayOffset (see page 313)	This is DisplayOffset, a member of class TCustomScreenTipsManager.
	Font (see page 313)	This is Font, a member of class TCustomScreenTipsManager.
	Footer (see page 313)	This is Footer, a member of class TCustomScreenTipsManager.
	FooterImage (see page 314)	This is FooterImage, a member of class TCustomScreenTipsManager.
	GradientEndColor (see page 314)	This is GradientEndColor, a member of class TCustomScreenTipsManager.
	GradientStartColor (see page 314)	This is GradientStartColor, a member of class TCustomScreenTipsManager.
	LinkedActionLists (see page 314)	This is LinkedActionLists, a member of class TCustomScreenTipsManager.
	Options (see page 314)	This is Options, a member of class TCustomScreenTipsManager.
	ScreenTips (see page 315)	This is ScreenTips, a member of class TCustomScreenTipsManager.

5.1.5.2 TCustomScreenTipsManager.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomScreenTipsManager(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3355)

Destroy

System::TObject::Free (see page 3372)

Owner

5.1.5.3 TCustomScreenTipsManager.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomScreenTipsManager();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3356)

DestroyComponents

Destroying

System::TObject::Free (see page 3372)

Notification

Owner

Release

Components

5.1.5.4 TCustomScreenTipsManager Methods

The methods of the TCustomScreenTipsManager class are listed here.

Protected Methods

	Name	Description
	DoCreateScreenTip (see page 309)	This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.

	DoDuplicateCreateError (see page 309)	This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.
	DoGetScreenTipClass (see page 309)	This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.
	DoScreenTipsLoaded (see page 309)	This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	DoShow (see page 309)	This is DoShow, a member of class TCustomScreenTipsManager.
	DoShowScreenTip (see page 310)	This is DoShowScreenTip, a member of class TCustomScreenTipsManager.
	Loaded (see page 310)	Initializes the component after the form file has been read into memory.
	Notification (see page 311)	Forwards notification messages to all owned components.
	UpdateActionInfo (see page 312)	This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

Public Methods

	Name	Description
	AddPopup (see page 308)	This is AddPopup, a member of class TCustomScreenTipsManager.
	ClearScreenTips (see page 308)	This is ClearScreenTips, a member of class TCustomScreenTipsManager.
	CreateScreenTip (see page 308)	This is CreateScreenTip, a member of class TCustomScreenTipsManager.
	GenerateScreenTips (see page 310)	This is GenerateScreenTips, a member of class TCustomScreenTipsManager.
	GetScreenTip (see page 310)	This is GetScreenTip, a member of class TCustomScreenTipsManager.
	RemovePopup (see page 311)	This is RemovePopup, a member of class TCustomScreenTipsManager.
	RemoveScreenTip (see page 312)	This is RemoveScreenTip, a member of class TCustomScreenTipsManager.
	UpdateOptions (see page 312)	This is UpdateOptions, a member of class TCustomScreenTipsManager.

5.1.5.4.1 TCustomScreenTipsManager.AddPopup Method

Pascal

```
procedure AddPopup(AScreenTipPopup: TCustomControl);
```

C++

```
__fastcall AddPopup(TCustomControl AScreenTipPopup);
```

Description

This is AddPopup, a member of class TCustomScreenTipsManager.

5.1.5.4.2 TCustomScreenTipsManager.ClearScreenTips Method

Pascal

```
procedure ClearScreenTips;
```

C++

```
__fastcall ClearScreenTips();
```

Description

This is ClearScreenTips, a member of class TCustomScreenTipsManager.

5.1.5.4.3 TCustomScreenTipsManager.CreateScreenTip Method

Pascal

```
function CreateScreenTip(aAction: TBasicAction): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem CreateScreenTip(TBasicAction aAction);
```

Description

This is CreateScreenTip, a member of class TCustomScreenTipsManager.

5.1.5.4.4 TCustomScreenTipsManager.DoCreateScreenTip Method

Pascal

```
procedure DoCreateScreenTip(aItem: TScreenTipItem); virtual;
```

C++

```
virtual __fastcall DoCreateScreenTip(TScreenTipItem aItem);
```

Description

This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.

5.1.5.4.5 TCustomScreenTipsManager.DoDuplicateCreateError Method

Pascal

```
procedure DoDuplicateCreateError(aAction: TBasicAction); virtual;
```

C++

```
virtual __fastcall DoDuplicateCreateError(TBasicAction aAction);
```

Description

This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.

5.1.5.4.6 TCustomScreenTipsManager.DoGetScreenTipClass Method

Pascal

```
procedure DoGetScreenTipClass(var WindowClass: TScreenTipsWindowClass); virtual;
```

C++

```
virtual __fastcall DoGetScreenTipClass(TScreenTipsWindowClass WindowClass);
```

Description

This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.

5.1.5.4.7 TCustomScreenTipsManager.DoScreenTipsLoaded Method

Pascal

```
procedure DoScreenTipsLoaded; virtual;
```

C++

```
virtual __fastcall DoScreenTipsLoaded();
```

Description

This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.

5.1.5.4.8 TCustomScreenTipsManager.DoShow Method

Pascal

```
procedure DoShow(Item: TScreenTipItem; var HintInfo: THintInfo); virtual;
```

C++

```
virtual __fastcall DoShow(TScreenTipItem Item, THintInfo HintInfo);
```

Description

This is DoShow, a member of class TCustomScreenTipsManager.

5.1.5.4.9 TCustomScreenTipsManager.DoShowScreenTip Method

Pascal

```
procedure DoShowScreenTip(Action: TBasicAction; Item: TScreenTipItem; var HintInfo:
THintInfo); virtual;
```

C++

```
virtual __fastcall DoShowScreenTip(TBasicAction Action, TScreenTipItem Item, THintInfo
HintInfo);
```

Description

This is DoShowScreenTip, a member of class TCustomScreenTipsManager.

5.1.5.4.10 TCustomScreenTipsManager.GenerateScreenTips Method

Pascal

```
function GenerateScreenTips: Boolean;
```

C++

```
__fastcall Boolean GenerateScreenTips();
```

Description

This is GenerateScreenTips, a member of class TCustomScreenTipsManager.

5.1.5.4.11 TCustomScreenTipsManager.GetScreenTip Method

Pascal

```
function GetScreenTip(aAction: TBasicAction): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetScreenTip(TBasicAction aAction);
```

Description

This is GetScreenTip, a member of class TCustomScreenTipsManager.

5.1.5.4.12 TCustomScreenTipsManager.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

5.1.5.4.13 TCustomScreenTipsManager.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

5.1.5.4.14 TCustomScreenTipsManager.RemovePopup Method

Pascal

```
procedure RemovePopup(AScreenTipPopup: TCustomControl);
```

C++

```
__fastcall RemovePopup(TCustomControl AScreenTipPopup);
```

Description

This is RemovePopup, a member of class TCustomScreenTipsManager.

5.1.5.4.15 TCustomScreenTipsManager.RemoveScreenTip Method

Pascal

```
function RemoveScreenTip(aScreenTip: TScreenTipItem): Boolean;
```

C++

```
__fastcall Boolean RemoveScreenTip(TScreenTipItem aScreenTip);
```

Description

This is RemoveScreenTip, a member of class TCustomScreenTipsManager.

5.1.5.4.16 TCustomScreenTipsManager.UpdateActionInfo Method

Pascal

```
procedure UpdateActionInfo(aAction: TContainedAction; aTooltip: TScreenTipItem);
```

C++

```
__fastcall UpdateActionInfo(TContainedAction aAction, TScreenTipItem aTooltip);
```

Description

This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

5.1.5.4.17 TCustomScreenTipsManager.UpdateOptions Method

Pascal

```
procedure UpdateOptions;
```

C++

```
__fastcall UpdateOptions();
```

Description

This is UpdateOptions, a member of class TCustomScreenTipsManager.

5.1.5.5 TCustomScreenTipsManager Properties

The properties of the TCustomScreenTipsManager class are listed here.

Public Properties

	Name	Description
	BorderStyle (see page 313)	This is BorderStyle, a member of class TCustomScreenTipsManager.
	DisplayOffset (see page 313)	This is DisplayOffset, a member of class TCustomScreenTipsManager.
	Font (see page 313)	This is Font, a member of class TCustomScreenTipsManager.
	Footer (see page 313)	This is Footer, a member of class TCustomScreenTipsManager.
	FooterImage (see page 314)	This is FooterImage, a member of class TCustomScreenTipsManager.

	GradientEndColor (see page 314)	This is GradientEndColor, a member of class TCustomScreenTipsManager.
	GradientStartColor (see page 314)	This is GradientStartColor, a member of class TCustomScreenTipsManager.
	LinkedActionLists (see page 314)	This is LinkedActionLists, a member of class TCustomScreenTipsManager.
	Options (see page 314)	This is Options, a member of class TCustomScreenTipsManager.
	ScreenTips (see page 315)	This is ScreenTips, a member of class TCustomScreenTipsManager.

5.1.5.5.1 TCustomScreenTipsManager.BorderStyle Property

Pascal

```
property BorderStyle: TScreenTipBorderStyle;
```

C++

```
__property TScreenTipBorderStyle BorderStyle;
```

Description

This is BorderStyle, a member of class TCustomScreenTipsManager.

5.1.5.5.2 TCustomScreenTipsManager.DisplayOffset Property

Pascal

```
property DisplayOffset: TDisplayOffset;
```

C++

```
__property TDisplayOffset DisplayOffset;
```

Description

This is DisplayOffset, a member of class TCustomScreenTipsManager.

5.1.5.5.3 TCustomScreenTipsManager.Font Property

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

This is Font, a member of class TCustomScreenTipsManager.

5.1.5.5.4 TCustomScreenTipsManager.Footer Property

Pascal

```
property Footer: string;
```

C++

```
__property UnicodeString Footer;
```

Description

This is Footer, a member of class TCustomScreenTipsManager.

5.1.5.5.5 TCustomScreenTipsManager.FooterImage Property

Pascal

```
property FooterImage: TPicture;
```

C++

```
__property TPicture FooterImage;
```

Description

This is FooterImage, a member of class TCustomScreenTipsManager.

5.1.5.5.6 TCustomScreenTipsManager.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TCustomScreenTipsManager.

5.1.5.5.7 TCustomScreenTipsManager.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TCustomScreenTipsManager.

5.1.5.5.8 TCustomScreenTipsManager.LinkedActionLists Property

Pascal

```
property LinkedActionLists: TActionListCollection;
```

C++

```
__property TActionListCollection LinkedActionLists;
```

Description

This is LinkedActionLists, a member of class TCustomScreenTipsManager.

5.1.5.5.9 TCustomScreenTipsManager.Options Property

Pascal

```
property Options: TScreenTipOptions;
```

C++

```
__property TScreenTipOptions Options;
```

Description

This is Options, a member of class TCustomScreenTipsManager.

5.1.5.5.10 TCustomScreenTipsManager.ScreenTips Property

Pascal

```
property ScreenTips: TScreenTipsCollection;
```

C++

```
__property TScreenTipsCollection ScreenTips;
```

Description

This is ScreenTips, a member of class TCustomScreenTipsManager.

5.1.5.6 TCustomScreenTipsManager Events

The events of the TCustomScreenTipsManager class are listed here.

Public Events

	Name	Description
	OnCreateScreenTip (see page 315)	This is OnCreateScreenTip, a member of class TCustomScreenTipsManager.
	OnDuplicateCreateError (see page 315)	This is OnDuplicateCreateError, a member of class TCustomScreenTipsManager.
	OnGetScreenTipWinClass (see page 316)	This is OnGetScreenTipWinClass, a member of class TCustomScreenTipsManager.
	OnScreenTipsLoaded (see page 316)	This is OnScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	OnShowScreenTip (see page 316)	This is OnShowScreenTip, a member of class TCustomScreenTipsManager.

5.1.5.6.1 TCustomScreenTipsManager.OnCreateScreenTip Event

Pascal

```
property OnCreateScreenTip: TCreateScreenTipEvent;
```

C++

```
__property TCreateScreenTipEvent OnCreateScreenTip;
```

Description

This is OnCreateScreenTip, a member of class TCustomScreenTipsManager.

5.1.5.6.2 TCustomScreenTipsManager.OnDuplicateCreateError Event

Pascal

```
property OnDuplicateCreateError: TDuplicateCreateErrorEvent;
```

C++

```
__property TDuplicateCreateErrorEvent OnDuplicateCreateError;
```

Description

This is OnDuplicateCreateError, a member of class TCustomScreenTipsManager.

5.1.5.6.3 TCustomScreenTipsManager.OnGetScreenTipWinClass Event

Pascal

```
property OnGetScreenTipWinClass: TScreenTipWinClass;
```

C++

```
__property TScreenTipWinClass OnGetScreenTipWinClass;
```

Description

This is OnGetScreenTipWinClass, a member of class TCustomScreenTipsManager.

5.1.5.6.4 TCustomScreenTipsManager.OnScreenTipsLoaded Event

Pascal

```
property OnScreenTipsLoaded: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnScreenTipsLoaded;
```

Description

This is OnScreenTipsLoaded, a member of class TCustomScreenTipsManager.

5.1.5.6.5 TCustomScreenTipsManager.OnShowScreenTip Event

Pascal

```
property OnShowScreenTip: TShowScreenTipEvent;
```

C++

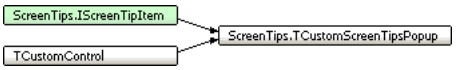
```
__property TShowScreenTipEvent OnShowScreenTip;
```

Description

This is OnShowScreenTip, a member of class TCustomScreenTipsManager.

5.1.6 TCustomScreenTipsPopup Class

Class Hierarchy



Pascal

```
TCustomScreenTipsPopup = class(TCustomControl, IScreenTipItem);
```

C++

```
class TCustomScreenTipsPopup : public TCustomControl, public IScreenTipItem;
```

File

ScreenTips

Description

This is class ScreenTips.TCustomScreenTipsPopup.

5.1.6.1 TCustomScreenTipsPopup Members

The following tables list the members exposed by TCustomScreenTipsPopup.

Protected Methods

	Name	Description
	LinkPopupToManager (see page 319)	This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.
	Notification (see page 319)	Responds to notifications that components are being created or destroyed.
	Paint (see page 320)	Renders the image of a custom control.
	RemoveLinkToManager (see page 320)	This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.
	UpdateAssociateToManager (see page 320)	This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

Public Methods

	Name	Description
	Create (see page 317)	Creates an instance of TCustomControl.
	Destroy (see page 318)	Destroys an instance of TCustomControl.
	GetScreenTipItem (see page 319)	This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

Public Properties

	Name	Description
	Associate (see page 321)	This is Associate, a member of class TCustomScreenTipsPopup.
	Glyph (see page 321)	This is Glyph, a member of class TCustomScreenTipsPopup.
	PopupType (see page 321)	This is PopupType, a member of class TCustomScreenTipsPopup.
	ScreenTip (see page 322)	This is ScreenTip, a member of class TCustomScreenTipsPopup.
	ScreenTipManager (see page 322)	This is ScreenTipManager, a member of class TCustomScreenTipsPopup.
	Transparent (see page 324)	This is Transparent, a member of class TCustomScreenTipsPopup.

Published Properties

	Name	Description
	ShowHint (see page 322)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

5.1.6.2 TCustomScreenTipsPopup.Create Constructor

Creates an instance of TCustomControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomScreenTipsPopup(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

5.1.6.3 TCustomScreenTipsPopup.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomScreenTipsPopup();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free (see page 3372)

delete

Create

5.1.6.4 TCustomScreenTipsPopup Methods

The methods of the TCustomScreenTipsPopup class are listed here.

Protected Methods

	Name	Description
	LinkPopupToManager (see page 319)	This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.
	Notification (see page 319)	Responds to notifications that components are being created or destroyed.
	Paint (see page 320)	Renders the image of a custom control.
	RemoveLinkToManager (see page 320)	This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.
	UpdateAssociateToManager (see page 320)	This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

Public Methods

	Name	Description
	GetScreenTipItem (see page 319)	This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

5.1.6.4.1 TCustomScreenTipsPopup.GetScreenTipItem Method

Pascal

```
function GetScreenTipItem: TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetScreenTipItem();
```

Description

This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

5.1.6.4.2 TCustomScreenTipsPopup.LinkPopupToManager Method

Pascal

```
procedure LinkPopupToManager; virtual;
```

C++

```
virtual __fastcall LinkPopupToManager();
```

Description

This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.

5.1.6.4.3 TCustomScreenTipsPopup.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware

controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

5.1.6.4.4 TCustomScreenTipsPopup.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

5.1.6.4.5 TCustomScreenTipsPopup.RemoveLinkToManager Method

Pascal

```
procedure RemoveLinkToManager; virtual;
```

C++

```
virtual __fastcall RemoveLinkToManager();
```

Description

This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.

5.1.6.4.6 TCustomScreenTipsPopup.UpdateAssociateToManager Method

Pascal

```
procedure UpdateAssociateToManager; virtual;
```

C++

```
virtual __fastcall UpdateAssociateToManager();
```

Description

This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

5.1.6.5 TCustomScreenTipsPopup Properties

The properties of the TCustomScreenTipsPopup class are listed here.

Public Properties

	Name	Description
	Associate (see page 321)	This is Associate, a member of class TCustomScreenTipsPopup.
	Glyph (see page 321)	This is Glyph, a member of class TCustomScreenTipsPopup.
	PopupType (see page 321)	This is PopupType, a member of class TCustomScreenTipsPopup.
	ScreenTip (see page 322)	This is ScreenTip, a member of class TCustomScreenTipsPopup.
	ScreenTipManager (see page 322)	This is ScreenTipManager, a member of class TCustomScreenTipsPopup.
	Transparent (see page 324)	This is Transparent, a member of class TCustomScreenTipsPopup.

Published Properties

	Name	Description
	ShowHint (see page 322)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

5.1.6.5.1 TCustomScreenTipsPopup.Associate Property

Pascal

```
property Associate: TControl;
```

C++

```
__property TControl * Associate;
```

Description

This is Associate, a member of class TCustomScreenTipsPopup.

5.1.6.5.2 TCustomScreenTipsPopup.Glyph Property

Pascal

```
property Glyph: TPicture;
```

C++

```
__property TPicture Glyph;
```

Description

This is Glyph, a member of class TCustomScreenTipsPopup.

5.1.6.5.3 TCustomScreenTipsPopup.PopupType Property

Pascal

```
property PopupType: TScreenTipPopupType;
```

C++

```
__property TScreenTipPopupType PopupType;
```

Description

This is PopupType, a member of class TCustomScreenTipsPopup.

5.1.6.5.4 TCustomScreenTipsPopup.ScreenTip Property

Pascal

```
property ScreenTip: TScreenTipItem;
```

C++

```
__property TScreenTipItem ScreenTip;
```

Description

This is ScreenTip, a member of class TCustomScreenTipsPopup.

5.1.6.5.5 TCustomScreenTipsPopup.ScreenTipManager Property

Pascal

```
property ScreenTipManager: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager ScreenTipManager;
```

Description

This is ScreenTipManager, a member of class TCustomScreenTipsPopup.

5.1.6.5.6 TCustomScreenTipsPopup.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
```

```
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

5.1.6.5.7 TCustomScreenTipsPopup.Transparent Property

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

This is Transparent, a member of class TCustomScreenTipsPopup.

5.1.7 TDisplayOffset Class

Class Hierarchy

Pascal

```
TDisplayOffset = class(TPersistent);
```

C++

```
class TDisplayOffset : public TPersistent;
```

File

ScreenTips

Description

This is class ScreenTips.TDisplayOffset.

5.1.7.1 TDisplayOffset Members

The following tables list the members exposed by TDisplayOffset.

Public Methods

	Name	Description
	Create (see page 324)	Constructs an object and initializes its data before the object is first used.

Published Properties

	Name	Description
	Control (see page 325)	This is Control, a member of class TDisplayOffset.
	X (see page 325)	This is X, a member of class TDisplayOffset.
	Y (see page 325)	This is Y, a member of class TDisplayOffset.

5.1.7.2 TDisplayOffset.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TDisplayOffset();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

5.1.7.3 TDisplayOffset Properties

The properties of the TDisplayOffset class are listed here.

Published Properties

	Name	Description
	Control (see page 325)	This is Control, a member of class TDisplayOffset.
	X (see page 325)	This is X, a member of class TDisplayOffset.
	Y (see page 325)	This is Y, a member of class TDisplayOffset.

5.1.7.3.1 TDisplayOffset.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TDisplayOffset.

5.1.7.3.2 TDisplayOffset.X Property

Pascal

```
property X: Integer;
```

C++

```
__property int X;
```

Description

This is X, a member of class TDisplayOffset.

5.1.7.3.3 TDisplayOffset.Y Property

Pascal

```
property Y: Integer;
```

C++

```
__property int Y;
```

Description

This is Y, a member of class TDisplayOffset.

5.1.8 TScreenTipBorderStyle Class

Class Hierarchy

Pascal

```
TScreenTipBorderStyle = class(TPersistent);
```

C++

```
class TScreenTipBorderStyle : public TPersistent;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipBorderStyle.

5.1.8.1 TScreenTipBorderStyle Members

The following tables list the members exposed by TScreenTipBorderStyle.

Public Methods

	Name	Description
	Create (see page 326)	Constructs an object and initializes its data before the object is first used.

Published Properties

	Name	Description
	Color (see page 327)	This is Color, a member of class TScreenTipBorderStyle.
	CornerSize (see page 327)	This is CornerSize, a member of class TScreenTipBorderStyle.
	Style (see page 327)	This is Style, a member of class TScreenTipBorderStyle.

5.1.8.2 TScreenTipBorderStyle.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScreenTipBorderStyle();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The

System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

5.1.8.3 TScreenTipBorderStyle Properties

The properties of the TScreenTipBorderStyle class are listed here.

Published Properties

	Name	Description
	Color (see page 327)	This is Color, a member of class TScreenTipBorderStyle.
	CornerSize (see page 327)	This is CornerSize, a member of class TScreenTipBorderStyle.
	Style (see page 327)	This is Style, a member of class TScreenTipBorderStyle.

5.1.8.3.1 TScreenTipBorderStyle.Color Property

Pascal

```
property Color: TColor;
```

C++

```
__property TColor Color;
```

Description

This is Color, a member of class TScreenTipBorderStyle.

5.1.8.3.2 TScreenTipBorderStyle.CornerSize Property

Pascal

```
property CornerSize: Integer;
```

C++

```
__property int CornerSize;
```

Description

This is CornerSize, a member of class TScreenTipBorderStyle.

5.1.8.3.3 TScreenTipBorderStyle.Style Property

Pascal

```
property Style: TTipBorderStyle;
```

C++

```
__property TTipBorderStyle Style;
```

Description

This is Style, a member of class TScreenTipBorderStyle.

5.1.9 TScreenTipItem Class

Class Hierarchy

Pascal

```
TScreenTipItem = class(TCollectionItem);
```

C++

```
class TScreenTipItem : public TCollectionItem;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipItem.

5.1.9.1 TScreenTipItem Members

The following tables list the members exposed by TScreenTipItem.

Protected Methods

	Name	Description
	GetDisplayName (see page 330)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 330)	Copies the contents of another, similar object.
	Create (see page 329)	Creates and initializes a TCollectionItem instance.
	Destroy (see page 329)	Destroys the TCollectionItem instance and frees its memory.
	IsEnabled (see page 331)	This is IsEnabled, a member of class TScreenTipItem.
	SetManager (see page 331)	This is SetManager, a member of class TScreenTipItem.

Public Properties

	Name	Description
	Height (see page 333)	This is Height, a member of class TScreenTipItem.
	Width (see page 334)	This is Width, a member of class TScreenTipItem.

Published Properties

	Name	Description
	Action (see page 332)	This is Action, a member of class TScreenTipItem.
	Description (see page 332)	This is Description, a member of class TScreenTipItem.
	DisabledDescription (see page 332)	This is DisabledDescription, a member of class TScreenTipItem.
	DisabledHeader (see page 332)	This is DisabledHeader, a member of class TScreenTipItem.
	Header (see page 332)	This is Header, a member of class TScreenTipItem.
	Image (see page 333)	This is Image, a member of class TScreenTipItem.
	Manager (see page 333)	This is Manager, a member of class TScreenTipItem.
	ShowFooter (see page 333)	This is ShowFooter, a member of class TScreenTipItem.
	ShowImage (see page 334)	This is ShowImage, a member of class TScreenTipItem.

5.1.9.2 TScreenTipItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TScreenTipItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add

Collection

5.1.9.3 TScreenTipItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScreenTipItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear

Delete

System::TObject::Free (see page 3372)

5.1.9.4 TScreenTipItem Methods

The methods of the TScreenTipItem class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 330)	Returns the name of the collection item as it appears in the collection editor.

Public Methods

	Name	Description
	Assign (see page 330)	Copies the contents of another, similar object.
	IsEnabled (see page 331)	This is IsEnabled, a member of class TScreenTipItem.
	SetManager (see page 331)	This is SetManager, a member of class TScreenTipItem.

5.1.9.4.1 TScreenTipItem.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError (see page 3919)

5.1.9.4.2 TScreenTipItem.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

5.1.9.4.3 TScreenTipItem.IsEnabled Method

Pascal

```
function IsEnabled: Boolean;
```

C++

```
__fastcall Boolean IsEnabled();
```

Description

This is IsEnabled, a member of class TScreenTipItem.

5.1.9.4.4 TScreenTipItem.SetManager Method

Pascal

```
procedure SetManager(AManager: TCustomScreenTipsManager);
```

C++

```
__fastcall SetManager(TCustomScreenTipsManager AManager);
```

Description

This is SetManager, a member of class TScreenTipItem.

5.1.9.5 TScreenTipItem Properties

The properties of the TScreenTipItem class are listed here.

Public Properties

	Name	Description
	Height (see page 333)	This is Height, a member of class TScreenTipItem.
	Width (see page 334)	This is Width, a member of class TScreenTipItem.

Published Properties

	Name	Description
	Action (see page 332)	This is Action, a member of class TScreenTipItem.
	Description (see page 332)	This is Description, a member of class TScreenTipItem.
	DisabledDescription (see page 332)	This is DisabledDescription, a member of class TScreenTipItem.
	DisabledHeader (see page 332)	This is DisabledHeader, a member of class TScreenTipItem.
	Header (see page 332)	This is Header, a member of class TScreenTipItem.
	Image (see page 333)	This is Image, a member of class TScreenTipItem.
R	Manager (see page 333)	This is Manager, a member of class TScreenTipItem.
	ShowFooter (see page 333)	This is ShowFooter, a member of class TScreenTipItem.
	ShowImage (see page 334)	This is ShowImage, a member of class TScreenTipItem.

5.1.9.5.1 TScreenTipItem.Action Property

Pascal

```
property Action: TContainedAction;
```

C++

```
__property TContainedAction Action;
```

Description

This is Action, a member of class TScreenTipItem.

5.1.9.5.2 TScreenTipItem.Description Property

Pascal

```
property Description: TStrings;
```

C++

```
__property TStrings Description;
```

Description

This is Description, a member of class TScreenTipItem.

5.1.9.5.3 TScreenTipItem.DisabledDescription Property

Pascal

```
property DisabledDescription: TStrings;
```

C++

```
__property TStrings DisabledDescription;
```

Description

This is DisabledDescription, a member of class TScreenTipItem.

5.1.9.5.4 TScreenTipItem.DisabledHeader Property

Pascal

```
property DisabledHeader: string;
```

C++

```
__property UnicodeString DisabledHeader;
```

Description

This is DisabledHeader, a member of class TScreenTipItem.

5.1.9.5.5 TScreenTipItem.Header Property

Pascal

```
property Header: string;
```

C++

```
__property UnicodeString Header;
```

Description

This is Header, a member of class TScreenTipItem.

5.1.9.5.6 TScreenTipItem.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class TScreenTipItem.

5.1.9.5.7 TScreenTipItem.Image Property

Pascal

```
property Image: TPicture;
```

C++

```
__property TPicture Image;
```

Description

This is Image, a member of class TScreenTipItem.

5.1.9.5.8 TScreenTipItem.Manager Property

Pascal

```
property Manager: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager Manager;
```

Description

This is Manager, a member of class TScreenTipItem.

5.1.9.5.9 TScreenTipItem.ShowFooter Property

Pascal

```
property ShowFooter: Boolean;
```

C++

```
__property Boolean ShowFooter;
```

Description

This is ShowFooter, a member of class TScreenTipItem.

5.1.9.5.10 TScreenTipItem.ShowImage Property

Pascal

```
property ShowImage: Boolean;
```

C++

```
__property Boolean ShowImage;
```

Description

This is ShowImage, a member of class TScreenTipItem.

5.1.9.5.11 TScreenTipItem.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class TScreenTipItem.

5.1.10 TScreenTipsCollection Class

Class Hierarchy

Pascal

```
TScreenTipsCollection = class(TCollection);
```

C++

```
class TScreenTipsCollection : public TCollection;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsCollection.

5.1.10.1 TScreenTipsCollection Members

The following tables list the members exposed by TScreenTipsCollection.

Protected Methods

	Name	Description
	GetOwner (see page 336)	Returns the owner of an object.

Public Methods

	Name	Description
	Add (see page 335)	Creates a new TCollectionItem instance and adds it to the Items array.
	Create (see page 335)	This is Create, a member of class TScreenTipsCollection.
	Insert (see page 336)	Creates a new TCollectionItem instance and adds it to the Items array.

Public Properties

	Name	Description
	Items (see page 337)	Lists the items in the collection.

5.1.10.2 TScreenTipsCollection.Create Constructor

Pascal

```
constructor Create(Manager: TCustomScreenTipsManager);
```

C++

```
__fastcall TScreenTipsCollection(TCustomScreenTipsManager Manager);
```

Description

This is Create, a member of class TScreenTipsCollection.

5.1.10.3 TScreenTipsCollection Methods

The methods of the TScreenTipsCollection class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 336)	Returns the owner of an object.

Public Methods

	Name	Description
	Add (see page 335)	Creates a new TCollectionItem instance and adds it to the Items array.
	Insert (see page 336)	Creates a new TCollectionItem instance and adds it to the Items array.

5.1.10.3.1 TScreenTipsCollection.Add Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Add: TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem Add();
```

Description

Call Add to create an item in the collection. The new item is placed at the end of the Items array.

Add returns the new collection item.

See Also

Items

Insert

Create

Delete

5.1.10.3.2 TScreenTipsCollection.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

5.1.10.3.3 TScreenTipsCollection.Insert Method

Creates a new TCollectionItem instance and adds it to the Items array.

Pascal

```
function Insert(Index: Integer): TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem Insert(int Index);
```

Description

Call Insert to add a new item at a specified position in the collection. Existing items (starting from the specified position) are moved up in the Items array.

Insert returns the new collection item.

See Also

Add

Items

5.1.10.4 TScreenTipsCollection Properties

The properties of the TScreenTipsCollection class are listed here.

Public Properties

	Name	Description
	Items (see page 337)	Lists the items in the collection.

5.1.10.4.1 TScreenTipsCollection.Items Property

Lists the items in the collection.

Pascal

```
property Items [Index: Integer]: TScreenTipItem;
```

C++

```
__property TScreenTipItem Items[int Index];
```

Description

Use Items to access individual items in the collection. The value of the `Index` parameter corresponds to the `Index` property of TCollectionItem. It represents the position of the item in the collection.

See Also

Count

ItemClass

Index

Add

Insert

5.1.11 TScreenTipsManager Class

Class Hierarchy

Pascal

```
TScreenTipsManager = class(TCustomScreenTipsManager);
```

C++

```
class TScreenTipsManager : public TCustomScreenTipsManager;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsManager.

5.1.11.1 TScreenTipsManager Members

The following tables list the members exposed by TScreenTipsManager.

Protected Methods

	Name	Description
	DoCreateScreenTip (see page 309)	This is DoCreateScreenTip, a member of class TCustomScreenTipsManager.
	DoDuplicateCreateError (see page 309)	This is DoDuplicateCreateError, a member of class TCustomScreenTipsManager.
	DoGetScreenTipClass (see page 309)	This is DoGetScreenTipClass, a member of class TCustomScreenTipsManager.
	DoScreenTipsLoaded (see page 309)	This is DoScreenTipsLoaded, a member of class TCustomScreenTipsManager.
	DoShow (see page 309)	This is DoShow, a member of class TCustomScreenTipsManager.
	DoShowScreenTip (see page 310)	This is DoShowScreenTip, a member of class TCustomScreenTipsManager.
	Loaded (see page 310)	Initializes the component after the form file has been read into memory.
	Notification (see page 311)	Forwards notification messages to all owned components.
	UpdateActionInfo (see page 312)	This is UpdateActionInfo, a member of class TCustomScreenTipsManager.

Public Methods

	Name	Description
	AddPopup (see page 308)	This is AddPopup, a member of class TCustomScreenTipsManager.
	ClearScreenTips (see page 308)	This is ClearScreenTips, a member of class TCustomScreenTipsManager.
	Create (see page 306)	Allocates memory and constructs a safely initialized instance of a component.
	CreateScreenTip (see page 308)	This is CreateScreenTip, a member of class TCustomScreenTipsManager.
	Destroy (see page 307)	Disposes of the component and its owned components.
	GenerateScreenTips (see page 310)	This is GenerateScreenTips, a member of class TCustomScreenTipsManager.
	GetScreenTip (see page 310)	This is GetScreenTip, a member of class TCustomScreenTipsManager.
	RemovePopup (see page 311)	This is RemovePopup, a member of class TCustomScreenTipsManager.
	RemoveScreenTip (see page 312)	This is RemoveScreenTip, a member of class TCustomScreenTipsManager.
	UpdateOptions (see page 312)	This is UpdateOptions, a member of class TCustomScreenTipsManager.

Published Events

	Name	Description
	OnCreateScreenTip (see page 341)	This is OnCreateScreenTip, a member of class TScreenTipsManager.
	OnDuplicateCreateError (see page 342)	This is OnDuplicateCreateError, a member of class TScreenTipsManager.
	OnGetScreenTipWinClass (see page 342)	This is OnGetScreenTipWinClass, a member of class TScreenTipsManager.
	OnScreenTipsLoaded (see page 342)	This is OnScreenTipsLoaded, a member of class TScreenTipsManager.
	OnShowScreenTip (see page 342)	This is OnShowScreenTip, a member of class TScreenTipsManager.

Published Properties

	Name	Description
	BorderStyle (see page 339)	This is BorderStyle, a member of class TScreenTipsManager.
	DisplayOffset (see page 339)	This is DisplayOffset, a member of class TScreenTipsManager.
	Font (see page 339)	This is Font, a member of class TScreenTipsManager.
	Footer (see page 340)	This is Footer, a member of class TScreenTipsManager.
	FooterImage (see page 340)	This is FooterImage, a member of class TScreenTipsManager.
	GradientEndColor (see page 340)	This is GradientEndColor, a member of class TScreenTipsManager.
	GradientStartColor (see page 340)	This is GradientStartColor, a member of class TScreenTipsManager.
	LinkedActionLists (see page 340)	This is LinkedActionLists, a member of class TScreenTipsManager.
	Options (see page 341)	This is Options, a member of class TScreenTipsManager.
	ScreenTips (see page 341)	This is ScreenTips, a member of class TScreenTipsManager.

5.1.11.2 TScreenTipsManager Properties

The properties of the TScreenTipsManager class are listed here.

Published Properties

	Name	Description
	BorderStyle (see page 339)	This is BorderStyle, a member of class TScreenTipsManager.
	DisplayOffset (see page 339)	This is DisplayOffset, a member of class TScreenTipsManager.
	Font (see page 339)	This is Font, a member of class TScreenTipsManager.
	Footer (see page 340)	This is Footer, a member of class TScreenTipsManager.
	FooterImage (see page 340)	This is FooterImage, a member of class TScreenTipsManager.
	GradientEndColor (see page 340)	This is GradientEndColor, a member of class TScreenTipsManager.
	GradientStartColor (see page 340)	This is GradientStartColor, a member of class TScreenTipsManager.
	LinkedActionLists (see page 340)	This is LinkedActionLists, a member of class TScreenTipsManager.
	Options (see page 341)	This is Options, a member of class TScreenTipsManager.
	ScreenTips (see page 341)	This is ScreenTips, a member of class TScreenTipsManager.

5.1.11.2.1 TScreenTipsManager.BorderStyle Property

Pascal

```
property BorderStyle: TScreenTipBorderStyle;
```

C++

```
__property TScreenTipBorderStyle BorderStyle;
```

Description

This is BorderStyle, a member of class TScreenTipsManager.

5.1.11.2.2 TScreenTipsManager.DisplayOffset Property

Pascal

```
property DisplayOffset: TDisplayOffset;
```

C++

```
__property TDisplayOffset DisplayOffset;
```

Description

This is DisplayOffset, a member of class TScreenTipsManager.

5.1.11.2.3 TScreenTipsManager.Font Property

Pascal

```
property Font: TFont;
```

C++

```
__property TFont Font;
```

Description

This is Font, a member of class TScreenTipsManager.

5.1.11.2.4 TScreenTipsManager.Footer Property

Pascal

```
property Footer: string;
```

C++

```
__property UnicodeString Footer;
```

Description

This is Footer, a member of class TScreenTipsManager.

5.1.11.2.5 TScreenTipsManager.FooterImage Property

Pascal

```
property FooterImage: TPicture;
```

C++

```
__property TPicture FooterImage;
```

Description

This is FooterImage, a member of class TScreenTipsManager.

5.1.11.2.6 TScreenTipsManager.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TScreenTipsManager.

5.1.11.2.7 TScreenTipsManager.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TScreenTipsManager.

5.1.11.2.8 TScreenTipsManager.LinkedActionLists Property

Pascal

```
property LinkedActionLists: TActionListCollection;
```

C++

```
__property TActionListCollection LinkedActionLists;
```

Description

This is LinkedActionLists, a member of class TScreenTipsManager.

5.1.11.2.9 TScreenTipsManager.Options Property

Pascal

```
property Options: TScreenTipOptions;
```

C++

```
__property TScreenTipOptions Options;
```

Description

This is Options, a member of class TScreenTipsManager.

5.1.11.2.10 TScreenTipsManager.ScreenTips Property

Pascal

```
property ScreenTips: TScreenTipsCollection;
```

C++

```
__property TScreenTipsCollection ScreenTips;
```

Description

This is ScreenTips, a member of class TScreenTipsManager.

5.1.11.3 TScreenTipsManager Events

The events of the TScreenTipsManager class are listed here.

Published Events

	Name	Description
	OnCreateScreenTip (see page 341)	This is OnCreateScreenTip, a member of class TScreenTipsManager.
	OnDuplicateCreateError (see page 342)	This is OnDuplicateCreateError, a member of class TScreenTipsManager.
	OnGetScreenTipWinClass (see page 342)	This is OnGetScreenTipWinClass, a member of class TScreenTipsManager.
	OnScreenTipsLoaded (see page 342)	This is OnScreenTipsLoaded, a member of class TScreenTipsManager.
	OnShowScreenTip (see page 342)	This is OnShowScreenTip, a member of class TScreenTipsManager.

5.1.11.3.1 TScreenTipsManager.OnCreateScreenTip Event

Pascal

```
property OnCreateScreenTip: TCreateScreenTipEvent;
```

C++

```
__property TCreateScreenTipEvent OnCreateScreenTip;
```

Description

This is OnCreateScreenTip, a member of class TScreenTipsManager.

5.1.11.3.2 TScreenTipsManager.OnDuplicateCreateError Event

Pascal

```
property OnDuplicateCreateError: TDuplicateCreateErrorEvent;
```

C++

```
__property TDuplicateCreateErrorEvent OnDuplicateCreateError;
```

Description

This is OnDuplicateCreateError, a member of class TScreenTipsManager.

5.1.11.3.3 TScreenTipsManager.OnGetScreenTipWinClass Event

Pascal

```
property OnGetScreenTipWinClass: TScreenTipWinClass;
```

C++

```
__property TScreenTipWinClass OnGetScreenTipWinClass;
```

Description

This is OnGetScreenTipWinClass, a member of class TScreenTipsManager.

5.1.11.3.4 TScreenTipsManager.OnScreenTipsLoaded Event

Pascal

```
property OnScreenTipsLoaded: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnScreenTipsLoaded;
```

Description

This is OnScreenTipsLoaded, a member of class TScreenTipsManager.

5.1.11.3.5 TScreenTipsManager.OnShowScreenTip Event

Pascal

```
property OnShowScreenTip: TShowScreenTipEvent;
```

C++

```
__property TShowScreenTipEvent OnShowScreenTip;
```

Description

This is OnShowScreenTip, a member of class TScreenTipsManager.

5.1.12 TScreenTipsPopup Class

Class Hierarchy

Pascal

```
TScreenTipsPopup = class(TCustomScreenTipsPopup);
```

C++

```
class TScreenTipsPopup : public TCustomScreenTipsPopup;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsPopup.

5.1.12.1 TScreenTipsPopup Members

The following tables list the members exposed by TScreenTipsPopup.

Protected Methods

	Name	Description
	LinkPopupToManager (see page 319)	This is LinkPopupToManager, a member of class TCustomScreenTipsPopup.
	Notification (see page 319)	Responds to notifications that components are being created or destroyed.
	Paint (see page 320)	Renders the image of a custom control.
	RemoveLinkToManager (see page 320)	This is RemoveLinkToManager, a member of class TCustomScreenTipsPopup.
	UpdateAssociateToManager (see page 320)	This is UpdateAssociateToManager, a member of class TCustomScreenTipsPopup.

Public Methods

	Name	Description
	Create (see page 317)	Creates an instance of TCustomControl.
	Destroy (see page 318)	Destroys an instance of TCustomControl.
	GetScreenTipItem (see page 362)	This is GetScreenTipItem, a member of class IScreenTipItem.
	GetScreenTipItem (see page 319)	This is GetScreenTipItem, a member of class TCustomScreenTipsPopup.

Published Properties

	Name	Description
	Align (see page 344)	Determines how the control aligns within its container (parent control).
	Anchors (see page 347)	Specifies how the control is anchored to its parent.
	Associate (see page 348)	This is Associate, a member of class TScreenTipsPopup.
	Glyph (see page 348)	This is Glyph, a member of class TScreenTipsPopup.
	PopupType (see page 348)	This is PopupType, a member of class TScreenTipsPopup.
	ScreenTip (see page 349)	This is ScreenTip, a member of class TScreenTipsPopup.
	ScreenTipManager (see page 349)	This is ScreenTipManager, a member of class TScreenTipsPopup.
	ShowHint (see page 322)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 349)	This is Transparent, a member of class TScreenTipsPopup.

	Visible (see page 349)	Determines whether the component appears onscreen.

5.1.12.2 TScreenTipsPopup Properties

The properties of the TScreenTipsPopup class are listed here.

Published Properties

	Name	Description
	Align (see page 344)	Determines how the control aligns within its container (parent control).
	Anchors (see page 347)	Specifies how the control is anchored to its parent.
	Associate (see page 348)	This is Associate, a member of class TScreenTipsPopup.
	Glyph (see page 348)	This is Glyph, a member of class TScreenTipsPopup.
	PopupType (see page 348)	This is PopupType, a member of class TScreenTipsPopup.
	ScreenTip (see page 349)	This is ScreenTip, a member of class TScreenTipsPopup.
	ScreenTipManager (see page 349)	This is ScreenTipManager, a member of class TScreenTipsPopup.
	Transparent (see page 349)	This is Transparent, a member of class TScreenTipsPopup.
	Visible (see page 349)	Determines whether the component appears onscreen.

5.1.12.2.1 TScreenTipsPopup.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
```

```
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
```

```
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
```

```
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

5.1.12.2.2 TScreenTipsPopup.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

5.1.12.2.3 TScreenTipsPopup.Associate Property

Pascal

```
property Associate: TControl;
```

C++

```
__property TControl * Associate;
```

Description

This is Associate, a member of class TScreenTipsPopup.

5.1.12.2.4 TScreenTipsPopup.Glyph Property

Pascal

```
property Glyph: TPicture;
```

C++

```
__property TPicture Glyph;
```

Description

This is Glyph, a member of class TScreenTipsPopup.

5.1.12.2.5 TScreenTipsPopup.PopupType Property

Pascal

```
property PopupType: TScreenTipPopupType;
```

C++

```
__property TScreenTipPopupType PopupType;
```

Description

This is PopupType, a member of class TScreenTipsPopup.

5.1.12.2.6 TScreenTipsPopup.ScreenTip Property

Pascal

```
property ScreenTip: TScreenTipItem;
```

C++

```
__property TScreenTipItem ScreenTip;
```

Description

This is ScreenTip, a member of class TScreenTipsPopup.

5.1.12.2.7 TScreenTipsPopup.ScreenTipManager Property

Pascal

```
property ScreenTipManager: TCustomScreenTipsManager;
```

C++

```
__property TCustomScreenTipsManager ScreenTipManager;
```

Description

This is ScreenTipManager, a member of class TScreenTipsPopup.

5.1.12.2.8 TScreenTipsPopup.Transparent Property

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

This is Transparent, a member of class TScreenTipsPopup.

5.1.12.2.9 TScreenTipsPopup.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

5.1.13 TScreenTipsWindow Class

Class Hierarchy

Pascal

```
TScreenTipsWindow = class(THintWindow);
```

C++

```
class TScreenTipsWindow : public THintWindow;
```

File

ScreenTips

Description

This is class ScreenTips.TScreenTipsWindow.

5.1.13.1 TScreenTipsWindow Members

The following tables list the members exposed by TScreenTipsWindow.

Protected Methods

	Name	Description
	CMMouseenter (see page 353)	This is CMMouseenter, a member of class TScreenTipsWindow.
	CMMouseLeave (see page 354)	This is CMMouseLeave, a member of class TScreenTipsWindow.
V	CreateParams (see page 354)	Initializes a window-creation parameter data structure.
V	CreateRoundedStyle (see page 354)	This is CreateRoundedStyle, a member of class TScreenTipsWindow.
V	CreateTipBorderStyle (see page 355)	This is CreateTipBorderStyle, a member of class TScreenTipsWindow.
V	CreateWnd (see page 355)	Creates the underlying window.
V	DrawBackground (see page 355)	This is DrawBackground, a member of class TScreenTipsWindow.
V	DrawBorder (see page 356)	This is DrawBorder, a member of class TScreenTipsWindow.
V	DrawFooter (see page 356)	This is DrawFooter, a member of class TScreenTipsWindow.
V	DrawHeader (see page 356)	This is DrawHeader, a member of class TScreenTipsWindow.
V	Paint (see page 356)	Renders the image of a custom control.
	WMNCHitTest (see page 357)	This is WMNCHitTest, a member of class TScreenTipsWindow.

Public Methods

	Name	Description
V	ActivateToolTip (see page 353)	This is ActivateToolTip, a member of class TScreenTipsWindow.
V	Create (see page 352)	Creates and initializes an instance of THintWindow.
V	Destroy (see page 352)	Destroys an instance of TCustomControl.

	HideHint (see page 356)	This is HideHint, a member of class TScreenTipsWindow.

Public Properties

	Name	Description
R	ActiveItem (see page 357)	This is ActiveItem, a member of class TScreenTipsWindow.
	Body (see page 358)	This is Body, a member of class TScreenTipsWindow.
	CornerSize (see page 358)	This is CornerSize, a member of class TScreenTipsWindow.
	Footer (see page 358)	This is Footer, a member of class TScreenTipsWindow.
	FooterImage (see page 358)	This is FooterImage, a member of class TScreenTipsWindow.
	GradientEndColor (see page 359)	This is GradientEndColor, a member of class TScreenTipsWindow.
	GradientStartColor (see page 359)	This is GradientStartColor, a member of class TScreenTipsWindow.
	HintControl (see page 359)	This is HintControl, a member of class TScreenTipsWindow.
	Image (see page 359)	This is Image, a member of class TScreenTipsWindow.
	TipBorderStyle (see page 359)	This is TipBorderStyle, a member of class TScreenTipsWindow.

5.1.13.2 TScreenTipsWindow.Create Constructor

Creates and initializes an instance of THintWindow.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScreenTipsWindow(TComponent * AOwner);
```

Description

Call Create to instantiate a hint window at runtime. Hint windows activated by the application in response to the ShowHint property of a control are created automatically.

See Also

ActivateHint

ShowHint

5.1.13.3 TScreenTipsWindow.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScreenTipsWindow();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a

descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free (see page 3372)

delete

Create

5.1.13.4 TScreenTipsWindow Methods

The methods of the TScreenTipsWindow class are listed here.

Protected Methods

	Name	Description
	CMMouseenter (see page 353)	This is CMMouseenter, a member of class TScreenTipsWindow.
	CMMouseLeave (see page 354)	This is CMMouseLeave, a member of class TScreenTipsWindow.
V	CreateParams (see page 354)	Initializes a window-creation parameter data structure.
V	CreateRoundedStyle (see page 354)	This is CreateRoundedStyle, a member of class TScreenTipsWindow.
V	CreateTipBorderStyle (see page 355)	This is CreateTipBorderStyle, a member of class TScreenTipsWindow.
V	CreateWnd (see page 355)	Creates the underlying window.
V	DrawBackground (see page 355)	This is DrawBackground, a member of class TScreenTipsWindow.
V	DrawBorder (see page 356)	This is DrawBorder, a member of class TScreenTipsWindow.
V	DrawFooter (see page 356)	This is DrawFooter, a member of class TScreenTipsWindow.
V	DrawHeader (see page 356)	This is DrawHeader, a member of class TScreenTipsWindow.
V	Paint (see page 356)	Renders the image of a custom control.
	WMNCHitTest (see page 357)	This is WMNCHitTest, a member of class TScreenTipsWindow.

Public Methods

	Name	Description
V	ActivateToolTip (see page 353)	This is ActivateToolTip, a member of class TScreenTipsWindow.
	HideHint (see page 356)	This is HideHint, a member of class TScreenTipsWindow.

5.1.13.4.1 TScreenTipsWindow.ActivateToolTip Method

Pascal

```
procedure ActivateToolTip(Rect: TRect; Item: TScreenTipItem); virtual;
```

C++

```
virtual __fastcall ActivateToolTip(TRect Rect, TScreenTipItem Item);
```

Description

This is ActivateToolTip, a member of class TScreenTipsWindow.

5.1.13.4.2 TScreenTipsWindow.CMMouseenter Method

Pascal

```
procedure CMMouseenter(var Message: TMessage); message CM_MOUSEENTER;
```

C++

```
__fastcall CMMouseenter(TMessage Message);
```

Description

This is CMMouseenter, a member of class TScreenTipsWindow.

5.1.13.4.3 TScreenTipsWindow.CMMouseLeave Method

Pascal

```
procedure CMMouseLeave(var Message: TMessage); message CM_MOUSELEAVE;
```

C++

```
__fastcall CMMouseLeave(TMessage Message);
```

Description

This is CMMouseLeave, a member of class TScreenTipsWindow.

5.1.13.4.4 TScreenTipsWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

5.1.13.4.5 TScreenTipsWindow.CreateRoundedStyle Method

Pascal

```
procedure CreateRoundedStyle; virtual;
```

C++

```
virtual __fastcall CreateRoundedStyle();
```

Description

This is CreateRoundedStyle, a member of class TScreenTipsWindow.

5.1.13.4.6 TScreenTipsWindow.CreateTipBorderStyle Method

Pascal

```
procedure CreateTipBorderStyle; virtual;
```

C++

```
virtual __fastcall CreateTipBorderStyle();
```

Description

This is CreateTipBorderStyle, a member of class TScreenTipsWindow.

5.1.13.4.7 TScreenTipsWindow.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

5.1.13.4.8 TScreenTipsWindow.DrawBackground Method

Pascal

```
procedure DrawBackground(const aRect: TRect); virtual;
```

C++

```
virtual __fastcall DrawBackground(const TRect aRect);
```

Description

This is DrawBackground, a member of class TScreenTipsWindow.

5.1.13.4.9 TScreenTipsWindow.DrawBorder Method

Pascal

```
procedure DrawBorder(const aRect: TRect); virtual;
```

C++

```
virtual __fastcall DrawBorder(const TRect aRect);
```

Description

This is DrawBorder, a member of class TScreenTipsWindow.

5.1.13.4.10 TScreenTipsWindow.DrawFooter Method

Pascal

```
procedure DrawFooter; virtual;
```

C++

```
virtual __fastcall DrawFooter();
```

Description

This is DrawFooter, a member of class TScreenTipsWindow.

5.1.13.4.11 TScreenTipsWindow.DrawHeader Method

Pascal

```
function DrawHeader: Integer; virtual;
```

C++

```
virtual __fastcall int DrawHeader();
```

Description

This is DrawHeader, a member of class TScreenTipsWindow.

5.1.13.4.12 TScreenTipsWindow.HideHint Method

Pascal

```
procedure HideHint;
```

C++

```
__fastcall HideHint();
```

Description

This is HideHint, a member of class TScreenTipsWindow.

5.1.13.4.13 TScreenTipsWindow.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

5.1.13.4.14 TScreenTipsWindow.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TScreenTipsWindow.

5.1.13.5 TScreenTipsWindow Properties

The properties of the TScreenTipsWindow class are listed here.

Public Properties

	Name	Description
R	ActiveItem (see page 357)	This is ActiveItem, a member of class TScreenTipsWindow.
	Body (see page 358)	This is Body, a member of class TScreenTipsWindow.
	CornerSize (see page 358)	This is CornerSize, a member of class TScreenTipsWindow.
	Footer (see page 358)	This is Footer, a member of class TScreenTipsWindow.
	FooterImage (see page 358)	This is FooterImage, a member of class TScreenTipsWindow.
	GradientEndColor (see page 359)	This is GradientEndColor, a member of class TScreenTipsWindow.
	GradientStartColor (see page 359)	This is GradientStartColor, a member of class TScreenTipsWindow.
	HintControl (see page 359)	This is HintControl, a member of class TScreenTipsWindow.
	Image (see page 359)	This is Image, a member of class TScreenTipsWindow.
	TipBorderStyle (see page 359)	This is TipBorderStyle, a member of class TScreenTipsWindow.

5.1.13.5.1 TScreenTipsWindow.ActiveItem Property

Pascal

```
property ActiveItem: TScreenTipItem;
```

C++

```
__property TScreenTipItem ActiveItem;
```

Description

This is ActiveItem, a member of class TScreenTipsWindow.

5.1.13.5.2 TScreenTipsWindow.Body Property

Pascal

```
property Body: string;
```

C++

```
__property UnicodeString Body;
```

Description

This is Body, a member of class TScreenTipsWindow.

5.1.13.5.3 TScreenTipsWindow.CornerSize Property

Pascal

```
property CornerSize: Integer;
```

C++

```
__property int CornerSize;
```

Description

This is CornerSize, a member of class TScreenTipsWindow.

5.1.13.5.4 TScreenTipsWindow.Footer Property

Pascal

```
property Footer: string;
```

C++

```
__property UnicodeString Footer;
```

Description

This is Footer, a member of class TScreenTipsWindow.

5.1.13.5.5 TScreenTipsWindow.FooterImage Property

Pascal

```
property FooterImage: TBitmap;
```

C++

```
__property TBitmap FooterImage;
```

Description

This is FooterImage, a member of class TScreenTipsWindow.

5.1.13.5.6 TScreenTipsWindow.GradientEndColor Property

Pascal

```
property GradientEndColor: TColor;
```

C++

```
__property TColor GradientEndColor;
```

Description

This is GradientEndColor, a member of class TScreenTipsWindow.

5.1.13.5.7 TScreenTipsWindow.GradientStartColor Property

Pascal

```
property GradientStartColor: TColor;
```

C++

```
__property TColor GradientStartColor;
```

Description

This is GradientStartColor, a member of class TScreenTipsWindow.

5.1.13.5.8 TScreenTipsWindow.HintControl Property

Pascal

```
property HintControl: TControl;
```

C++

```
__property TControl * HintControl;
```

Description

This is HintControl, a member of class TScreenTipsWindow.

5.1.13.5.9 TScreenTipsWindow.Image Property

Pascal

```
property Image: TPicture;
```

C++

```
__property TPicture Image;
```

Description

This is Image, a member of class TScreenTipsWindow.

5.1.13.5.10 TScreenTipsWindow.TipBorderStyle Property

Pascal

```
property TipBorderStyle: TTipBorderStyle;
```

C++

```
__property TTipBorderStyle TipBorderStyle;
```

Description

This is TipBorderStyle, a member of class TScreenTipsWindow.

5.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IScreenTipItem (see page 361)	This is class ScreenTips.IScreenTipItem.

5.2.1 IScreenTipItem Interface

Class Hierarchy

ScreenTips.IScreenTipItem

Pascal

```
IScreenTipItem = interface;
```

C++

```
__interface IScreenTipItem;
```

File

ScreenTips

Description

This is class ScreenTips.IScreenTipItem.

5.2.1.1 IScreenTipItem Members

The following tables list the members exposed by IScreenTipItem.

Public Methods

	Name	Description
	GetScreenTipItem (see page 362)	This is GetScreenTipItem, a member of class IScreenTipItem.

5.2.1.2 IScreenTipItem Methods

The methods of the IScreenTipItem class are listed here.

Public Methods

	Name	Description
	GetScreenTipItem (see page 362)	This is GetScreenTipItem, a member of class IScreenTipItem.

5.2.1.2.1 IScreenTipItem.GetScreenTipItem Method

Pascal

```
function GetScreenTipItem: TScreenTipItem;
```

C++

```
__fastcall TScreenTipItem GetScreenTipItem();
```

Description

This is GetScreenTipItem, a member of class IScreenTipItem.

5.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDescriptionType (see page 363)	This is record ScreenTips.TDescriptionType.
	TImageAlign (see page 363)	This is record ScreenTips.TImageAlign.
	TItemImageAlign (see page 364)	This is record ScreenTips.TItemImageAlign.
	TScreenTipOption (see page 364)	This is record ScreenTips.TScreenTipOption.
	TScreenTipPopupType (see page 365)	This is record ScreenTips.TScreenTipPopupType.
	TTipBorderStyle (see page 365)	This is record ScreenTips.TTipBorderStyle.

5.3.1 ScreenTips.TDescriptionType Enumeration

Pascal

```
TDescriptionType = (
  dtManual,
  dtHelp,
  dtResource
);
```

C++

```
enum TDescriptionType {
  dtManual,
  dtHelp,
  dtResource
};
```

File

ScreenTips

Description

This is record ScreenTips.TDescriptionType.

5.3.2 ScreenTips.TImageAlign Enumeration

Pascal

```
TImageAlign = (
  iaLeft,
  iaRight
);
```

C++

```
enum TImageAlign {
  iaLeft,
  iaRight
```

```
};
```

File

ScreenTips

Description

This is record ScreenTips.TImageAlign.

5.3.3 ScreenTips.TItemImageAlign Enumeration

Pascal

```
TItemImageAlign = (
  iiaLeft,
  iiaCenter,
  iiaRight
);
```

C++

```
enum TItemImageAlign {
  iiaLeft,
  iiaCenter,
  iiaRight
};
```

File

ScreenTips

Description

This is record ScreenTips.TItemImageAlign.

5.3.4 ScreenTips.TScreenTipOption Enumeration

Pascal

```
TScreenTipOption = (
  soShowHeader,
  soShowShortCut
);
```

C++

```
enum TScreenTipOption {
  soShowHeader,
  soShowShortCut
};
```

File

ScreenTips

Description

This is record ScreenTips.TScreenTipOption.

5.3.5 ScreenTips.TScreenTipPopupType Enumeration

Pascal

```
TScreenTipPopupType = (
  ptCustom,
  ptHelp,
  ptInfo
);
```

C++

```
enum TScreenTipPopupType {
  ptCustom,
  ptHelp,
  ptInfo
};
```

File

ScreenTips

Description

This is record ScreenTips.TScreenTipPopupType.

5.3.6 ScreenTips.TTipBorderStyle Enumeration

Pascal

```
TTipBorderStyle = (
  hbsNormal,
  hbsRounded
);
```

C++

```
enum TTipBorderStyle {
  hbsNormal,
  hbsRounded
};
```

File

ScreenTips

Description

This is record ScreenTips.TTipBorderStyle.

5.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCreateScreenTipEvent (see page 366)	This is type ScreenTips.TCreateScreenTipEvent.
TDuplicateCreateErrorEvent (see page 366)	This is type ScreenTips.TDuplicateCreateErrorEvent.
TScreenTipOptions (see page 367)	This is type ScreenTips.TScreenTipOptions.
TScreenTipsWindowClass (see page 367)	This is type ScreenTips.TScreenTipsWindowClass.
TScreenTipWinClass (see page 367)	This is type ScreenTips.TScreenTipWinClass.
TShowScreenTipEvent (see page 368)	This is type ScreenTips.TShowScreenTipEvent.

5.4.1 ScreenTips.TCreateScreenTipEvent Type

Pascal

```
TCreateScreenTipEvent = procedure (Manager: TObject; Item: TScreenTipItem) of object;
```

C++

```
(Manager: TObject; Item: TScreenTipItem) ( TCreateScreenTipEvent)();
```

File

ScreenTips

Description

This is type ScreenTips.TCreateScreenTipEvent.

5.4.2 ScreenTips.TDuplicateCreateErrorEvent Type

Pascal

```
TDuplicateCreateErrorEvent = procedure (Manager: TObject; Action: TBasicAction) of object;
```

C++

```
(Manager: TObject; Action: TBasicAction) ( TDuplicateCreateErrorEvent)();
```

File

ScreenTips

Description

This is type ScreenTips.TDuplicateCreateErrorEvent.

5.4.3 ScreenTips.TScreenTipOptions Type

Pascal

```
TScreenTipOptions = set of TScreenTipOption;
```

C++

```
TScreenTipOption TScreenTipOptions;
```

File

ScreenTips

Description

This is type ScreenTips.TScreenTipOptions.

5.4.4 ScreenTips.TScreenTipsWindowClass Type

Pascal

```
TScreenTipsWindowClass = class of TScreenTipsWindow;
```

C++

```
TScreenTipsWindow TScreenTipsWindowClass;
```

File

ScreenTips

Description

This is type ScreenTips.TScreenTipsWindowClass.

5.4.5 ScreenTips.TScreenTipWinClass Type

Pascal

```
TScreenTipWinClass = procedure (Manager: TObject; var ScreenTipWinClass:
TScreenTipsWindowClass) of object;
```

C++

```
(Manager: TObject; var ScreenTipWinClass: TScreenTipsWindowClass) ( TScreenTipWinClass)();
```

File

ScreenTips

Description

This is type ScreenTips.TScreenTipWinClass.

5.4.6 ScreenTips.TShowScreenTipEvent Type

Pascal

```
TShowScreenTipEvent = procedure (Manager: TObject; Action: TBasicAction; var ShowScreenTip:
Boolean) of object;
```

C++

```
(Manager: TObject; Action: TBasicAction; var ShowScreenTip: Boolean) ( TShowScreenTipEvent)();
```

File

ScreenTips

Description

This is type ScreenTips.TShowScreenTipEvent.

5.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
cDefaultBorderColor (see page 369)	This is constant ScreenTips.cDefaultBorderColor.
cDefaultEndColor (see page 369)	This is constant ScreenTips.cDefaultEndColor.
cDefaultStartColor (see page 370)	This is constant ScreenTips.cDefaultStartColor.
cScreenTipHeaderSpacing (see page 370)	This is constant ScreenTips.cScreenTipHeaderSpacing.
cScreenTipMinimumHeight (see page 370)	This is constant ScreenTips.cScreenTipMinimumHeight.
cScreenTipTextAndImageWidth (see page 371)	This is constant ScreenTips.cScreenTipTextAndImageWidth.
cScreenTipTextOnlyWidth (see page 371)	This is constant ScreenTips.cScreenTipTextOnlyWidth.

5.5.1 ScreenTips.cDefaultBorderColor Constant

Pascal

```
cDefaultBorderColor = $00767676;
```

C++

```
cDefaultBorderColor = 0x00767676;
```

File

ScreenTips

Description

This is constant ScreenTips.cDefaultBorderColor.

5.5.2 ScreenTips.cDefaultEndColor Constant

Pascal

```
cDefaultEndColor = clBtnFace;
```

C++

```
cDefaultEndColor = clBtnFace;
```

File

ScreenTips

Description

This is constant ScreenTips.cDefaultEndColor.

5.5.3 ScreenTips.cDefaultStartColor Constant

Pascal

```
cDefaultStartColor = clWhite;
```

C++

```
cDefaultStartColor = clWhite;
```

File

ScreenTips

Description

This is constant ScreenTips.cDefaultStartColor.

5.5.4 ScreenTips.cScreenTipHeaderSpacing Constant

Pascal

```
cScreenTipHeaderSpacing = 14;
```

C++

```
cScreenTipHeaderSpacing = 14;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipHeaderSpacing.

5.5.5 ScreenTips.cScreenTipMinimumHeight Constant

Pascal

```
cScreenTipMinimumHeight = 50;
```

C++

```
cScreenTipMinimumHeight = 50;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipMinimumHeight.

5.5.6 ScreenTips.cScreenTipTextAndImageWidth Constant

Pascal

```
cScreenTipTextAndImageWidth = 318;
```

C++

```
cScreenTipTextAndImageWidth = 318;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipTextAndImageWidth.

5.5.7 ScreenTips.cScreenTipTextOnlyWidth Constant

Pascal

```
cScreenTipTextOnlyWidth = 210;
```

C++

```
cScreenTipTextOnlyWidth = 210;
```

File

ScreenTips

Description

This is constant ScreenTips.cScreenTipTextOnlyWidth.

6 SessColn Namespace

This is namespace SessColn.

Topics

Name	Description
Classes (see page 374)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 396)	The following table lists structs, records, enums in this documentation.
Types (see page 397)	The following table lists types in this documentation.
Variables (see page 398)	The following table lists variables in this documentation.

Classes

	Name	Description
	ESessionsException (see page 374)	ESessionsException is the exception that is raised when TSessionsService encounters a problem.
	TAbstractSessionIDGen (see page 374)	This is class SessColn.TAbstractSessionIDGen.
	TAbstractSessionStatusChange (see page 375)	This is class SessColn.TAbstractSessionStatusChange.
	TSessionIDGen (see page 377)	This is class SessColn.TSessionIDGen.
	TSessionItem (see page 378)	
	TSessions (see page 384)	

Structs, Records, Enums

	Name	Description
	TSessionStatusFlag (see page 396)	This is record SessColn.TSessionStatusFlag.

Types

Name	Description
TSessionStatusFlags (see page 397)	This is type SessColn.TSessionStatusFlags.

Variables

Name	Description
Sessions (see page 398)	This is variable SessColn.Sessions.

6.1 Classes

The following table lists classes in this documentation.

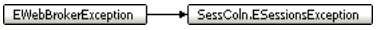
Classes

	Name	Description
	ESessionsException (see page 374)	ESessionsException is the exception that is raised when TSessionsService encounters a problem.
	TAbstractSessionIDGen (see page 374)	This is class SessColn.TAbstractSessionIDGen.
	TAbstractSessionStatusChange (see page 375)	This is class SessColn.TAbstractSessionStatusChange.
	TSessionIDGen (see page 377)	This is class SessColn.TSessionIDGen.
	TSessionItem (see page 378)	
	TSessions (see page 384)	

6.1.1 ESessionsException Class

ESessionsException is the exception that is raised when TSessionsService encounters a problem.

Class Hierarchy



Pascal

```
ESessionsException = class(EWebBrokerException);
```

C++

```
class ESessionsException : public EWebBrokerException;
```

File

SessColn

Description

ESessionsException represents exceptions that occur when managing the Web sessions in a WebSnap application. For example, this exception is raised when the application tries to create a new session and the sessions service is already storing the maximum allowed number of sessions.

See Also

TSessionsService (see page 6756)

6.1.2 TAbstractSessionIDGen Class

Class Hierarchy

Pascal

```
TAbstractSessionIDGen = class;
```

C++

```
class TAbstractSessionIDGen;
```

File

SessColn

Description

This is class SessColn.TAbstractSessionIDGen.

6.1.2.1 TAbstractSessionIDGen Members

The following tables list the members exposed by TAbstractSessionIDGen.

Public Methods

	Name	Description
	NextID (see page 375)	This is NextID, a member of class TAbstractSessionIDGen.

6.1.2.2 TAbstractSessionIDGen Methods

The methods of the TAbstractSessionIDGen class are listed here.

Public Methods

	Name	Description
	NextID (see page 375)	This is NextID, a member of class TAbstractSessionIDGen.

6.1.2.2.1 TAbstractSessionIDGen.NextID Method

Pascal

```
function NextID: string; virtual; abstract;
```

C++

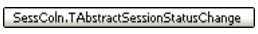
```
virtual __fastcall UnicodeString NextID() = 0;
```

Description

This is NextID, a member of class TAbstractSessionIDGen.

6.1.3 TAbstractSessionStatusChange Class

Class Hierarchy



Pascal

```
TAbstractSessionStatusChange = class;
```

C++

```
class TAbstractSessionStatusChange;
```

File

SessColn

Description

This is class SessColn.TAbstractSessionStatusChange.

6.1.3.1 TAbstractSessionStatusChange Members

The following tables list the members exposed by TAbstractSessionStatusChange.

Protected Methods

	Name	Description
	SessionStarted (see page 376)	This is SessionStarted, a member of class TAbstractSessionStatusChange.
	SessionTerminated (see page 376)	This is SessionTerminated, a member of class TAbstractSessionStatusChange.
	SessionTimeout (see page 377)	This is SessionTimeout, a member of class TAbstractSessionStatusChange.

6.1.3.2 TAbstractSessionStatusChange Methods

The methods of the TAbstractSessionStatusChange class are listed here.

Protected Methods

	Name	Description
	SessionStarted (see page 376)	This is SessionStarted, a member of class TAbstractSessionStatusChange.
	SessionTerminated (see page 376)	This is SessionTerminated, a member of class TAbstractSessionStatusChange.
	SessionTimeout (see page 377)	This is SessionTimeout, a member of class TAbstractSessionStatusChange.

6.1.3.2.1 TAbstractSessionStatusChange.SessionStarted Method

Pascal

```
procedure SessionStarted(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionStarted(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionStarted, a member of class TAbstractSessionStatusChange.

6.1.3.2.2 TAbstractSessionStatusChange.SessionTerminated Method

Pascal

```
procedure SessionTerminated(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionTerminated(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionTerminated, a member of class TAbstractSessionStatusChange.

6.1.3.2.3 TAbstractSessionStatusChange.SessionTimeout Method

Pascal

```
procedure SessionTimeout(AID: TSessionID; var AHandled: Boolean); virtual; abstract;
```

C++

```
virtual __fastcall SessionTimeout(TSessionID AID, Boolean AHandled) = 0;
```

Description

This is SessionTimeout, a member of class TAbstractSessionStatusChange.

6.1.4 TSessionIDGen Class

Class Hierarchy

Pascal

```
TSessionIDGen = class(TAbstractSessionIDGen);
```

C++

```
class TSessionIDGen : public TAbstractSessionIDGen;
```

File

SessColn

Description

This is class SessColn.TSessionIDGen.

6.1.4.1 TSessionIDGen Members

The following tables list the members exposed by TSessionIDGen.

Public Methods

	Name	Description
	Create (see page 377)	This is Create, a member of class TSessionIDGen.
V	NextID (see page 378)	This is NextID, a member of class TSessionIDGen.

6.1.4.2 TSessionIDGen.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TSessionIDGen();
```

Description

This is Create, a member of class TSessionIDGen.

6.1.4.3 TSessionIDGen Methods

The methods of the TSessionIDGen class are listed here.

Public Methods

	Name	Description
	NextID (see page 378)	This is NextID, a member of class TSessionIDGen.

6.1.4.3.1 TSessionIDGen.NextID Method

Pascal

```
function NextID: string; override;
```

C++

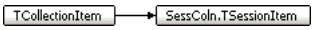
```
virtual __fastcall UnicodeString NextID();
```

Description

This is NextID, a member of class TSessionIDGen.

6.1.5 TSessionItem Class

Class Hierarchy



Pascal

```
TSessionItem = class(TCollectionItem);
```

C++

```
class TSessionItem : public TCollectionItem;
```

File

SessColn

See Also

TSessions (see page 384)

6.1.5.1 TSessionItem Members

The following tables list the members exposed by TSessionItem.

Public Methods

	Name	Description
	Assign (see page 380)	Copies the properties of one web session item to another web session item.
	Create (see page 379)	Creates and initializes a TCollectionItem instance.
	Destroy (see page 379)	Destroys the TCollectionItem instance and frees its memory.
	IsActive (see page 380)	Sets the state of the web session.

Public Properties

	Name	Description
	ExpirationTime (see page 381)	Indicates when the Web session is to be terminated.

Published Properties

	Name	Description
	Expired (see page 381)	Indicates whether the web session is expired.
	ID (see page 382)	
	Items (see page 382)	
	StartTime (see page 382)	Indicates when the Web session is to be started.
	StatusFlags (see page 382)	Gets or sets the status for the web session.
	Terminated (see page 383)	Sets the status of a web session to terminated.
	TimeoutMinutes (see page 383)	Sets the number of minutes before an idle web session is timed-out.
	TouchTime (see page 384)	

6.1.5.2 TSessionItem.Create Constructor

Creates and initializes a TCollectionItem instance.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TSessionItem(TCollection Collection);
```

Description

Do not call Create to instantiate a collection item. Instead, call the Add method of the collection to which the item should belong.

Collection identifies the TCollection instance to which the new item belongs.

See Also

Add

Collection

6.1.5.3 TSessionItem.Destroy Destructor

Destroys the TCollectionItem instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSessionItem();
```

Description

Destroy is called indirectly by TCollection's Clear or Delete method.

See Also

Clear

Delete

System::TObject::Free (see page 3372)

6.1.5.4 TSessionItem Methods

The methods of the TSessionItem class are listed here.

Public Methods

	Name	Description
	Assign (see page 380)	Copies the properties of one web session item to another web session item.
	IsActive (see page 380)	Sets the state of the web session.

6.1.5.4.1 TSessionItem.Assign Method

Copies the properties of one web session item to another web session item.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use the Assign method to assign the value of one web session item to another web session item. (more here)

See Also

Assign

6.1.5.4.2 TSessionItem.IsActive Method

Sets the state of the web session.

Pascal

```
function IsActive: Boolean;
```

C++

```
__fastcall Boolean IsActive();
```

Description

Use the IsActive method to set the web session state. Set IsActive to True true to continue a web session. Otherwise, set it to False false. (More here).

See Also

StatusFlags (see page 382)

Terminated (see page 383)

GetIsActive (see page 389)

6.1.5.5 TSessionItem Properties

The properties of the TSessionItem class are listed here.

Public Properties

	Name	Description
	ExpirationTime (see page 381)	Indicates when the Web session is to be terminated.

Published Properties

	Name	Description
	Expired (see page 381)	Indicates whether the web session is expired.
	ID (see page 382)	
	Items (see page 382)	
	StartTime (see page 382)	Indicates when the Web session is to be started.
	StatusFlags (see page 382)	Gets or sets the status for the web session.
	Terminated (see page 383)	Sets the status of a web session to terminated.
	TimeoutMinutes (see page 383)	Sets the number of minutes before an idle web session is timed-out.
	TouchTime (see page 384)	

6.1.5.5.1 TSessionItem.ExpirationTime Property

Indicates when the Web session is to be terminated.

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

Read (see page 3634) ExpirationTime to see when the web session is (was?) to be terminated. The expiration time is represented by a TDateTime (see page 3323) type.

See Also

Expired (see page 381)

StartTime (see page 382)

GetExpirationTime (see page 388)

6.1.5.5.2 TSessionItem.Expired Property

Indicates whether the web session is expired.

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

Set Expired to True true to end a web session. Otherwise, set it to False false (More here).

See Also

ExpirationTime (see page 381)

Terminated (see page 383)

GetExpired (see page 388)

6.1.5.5.3 TSessionItem.ID Property

Pascal

```
property ID: string;
```

C++

```
__property UnicodeString ID;
```

6.1.5.5.4 TSessionItem.Items Property

Pascal

```
property Items: TAbstractNamedVariants;
```

C++

```
__property TAbstractNamedVariants Items;
```

6.1.5.5.5 TSessionItem.StartTime Property

Indicates when the Web session is to be started.

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

Use StartTime to start a web session at a certain date and time. The start time is represented by a TDateTime (see page 3323) type.

See Also

ExpirationTime (see page 381)

TouchTime (see page 384)

GetStartTime (see page 390)

6.1.5.5.6 TSessionItem.StatusFlags Property

Gets or sets the status for the web session.

Pascal

```
property StatusFlags: TSessionStatusFlags;
```

C++

```
__property TSessionStatusFlags StatusFlags;
```

Description

Set StatusFlags to change the state of the web session. (more here)

The TSessionStatusFlags (see page 397) type defines the set of possible state flags for the StatusFlags property. The following table lists the possible values for the TSessionStatusFlags (see page 397) type and the meaning corresponding to each flag:

Flag	Web session state
ssStarted	The web session is started.
ssTerminated	The web session is terminated.
ssTimeout	The web session is timed-out.

See Also

StartTime (see page 382)

Terminated (see page 383)

TimeoutMinutes (see page 383)

CheckStatusChange (see page 386)

6.1.5.5.7 TSessionItem.Terminated Property

Sets the status of a web session to terminated.

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

Set Terminated to True true to end a web session. Otherwise, set it to False false. (More here).

See Also

StatusFlags (see page 382)

TimeoutMinutes (see page 383)

GetTerminated (see page 390)

6.1.5.5.8 TSessionItem.TimeoutMinutes Property

Sets the number of minutes before an idle web session is timed-out.

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

Set TimeoutMinutes to the number of minutes a web session that a web session can be idle before it is timed out by the server. (More here)

See Also

StatusFlags (see page 382)

Terminated (see page 383)

GetTimeoutMinutes (see page 390)

6.1.5.5.9 TSessionItem.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

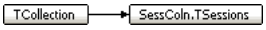
```
__property TDateTime TouchTime;
```

See Also

GetTouchTime (see page 391)

6.1.6 TSessions Class

Class Hierarchy



Pascal

```
TSessions = class(TCollection);
```

C++

```
class TSessions : public TCollection;
```

File

SessColn

6.1.6.1 TSessions Members

The following tables list the members exposed by TSessions.

Public Methods

	Name	Description
	CheckStatusChange (see page 386)	CheckStatusChanges checks if the session status has changed.
	ConnectActiveSession (see page 387)	This is ConnectActiveSession, a member of class TSessions.
	ConnectSession (see page 387)	
	Create (see page 385)	Constructs an object and initializes its data before the object is first used.
V	CreateSessionIDGen (see page 387)	This is CreateSessionIDGen, a member of class TSessions.
V	Destroy (see page 385)	Destroys the collection and each item in it.
	DisconnectSession (see page 387)	Disconnects the specified session.
	GetExpirationTime (see page 388)	Gets the expiration time for the specified session.
	GetExpired (see page 388)	
	GetIDCount (see page 388)	This is GetIDCount, a member of class TSessions.
	GetIDs (see page 388)	This is GetIDs, a member of class TSessions.
	GetIsActive (see page 389)	
	GetItemCount (see page 389)	Gets the number of items in the specified session.
	GetItems (see page 389)	
	GetSession (see page 389)	
	GetSessionIDGen (see page 390)	This is GetSessionIDGen, a member of class TSessions.
	GetStartTime (see page 390)	Gets the start time for the specified session.
	GetTerminated (see page 390)	

	GetTimeoutMinutes (see page 390)	Gets the number of minutes before the session times out.
	GetTouchTime (see page 391)	
	ItemValue (see page 391)	
	LoadFromFile (see page 391)	Loads the session information from a file.
	LoadFromStream (see page 391)	Loads the session information from a stream.
	NotifyStatusChange (see page 392)	
	SaveToFile (see page 392)	Saves the session information to a file.
	SaveToStream (see page 392)	Saves the session output to a stream.
	SessionExists (see page 393)	Indicates whether the specified session exists.
	SetItemValue (see page 393)	
	SetTimeoutMinutes (see page 393)	Sets the timeout minutes for the specified session.
	StartSession (see page 393)	Starts a new session.
	TerminateSession (see page 394)	Terminates the specified session.
	TouchSession (see page 394)	

Public Properties

	Name	Description
R	ActiveCount (see page 394)	This is ActiveCount, a member of class TSessions.
	FileName (see page 395)	Gets or sets the file name for the session.
	MaxSessions (see page 395)	This is MaxSessions, a member of class TSessions.

6.1.6.2 TSessions.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSessions();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

6.1.6.3 TSessions.Destroy Destructor

Destroys the collection and each item in it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSessions();
```

Description

Destroy calls the Clear method to free each item referenced in the Items array, then destroys the collection itself.

See Also

Clear

System::TObject::Free (see page 3372)

6.1.6.4 TSessions Methods

The methods of the TSessions class are listed here.

Public Methods

	Name	Description
	CheckStatusChange (see page 386)	CheckStatusChanges checks if the session status has changed.
	ConnectActiveSession (see page 387)	This is ConnectActiveSession, a member of class TSessions.
	ConnectSession (see page 387)	
V	CreateSessionIDGen (see page 387)	This is CreateSessionIDGen, a member of class TSessions.
	DisconnectSession (see page 387)	Disconnects the specified session.
	GetExpirationTime (see page 388)	Gets the expiration time for the specified session.
	GetExpired (see page 388)	
	GetIDCount (see page 388)	This is GetIDCount, a member of class TSessions.
	GetIDs (see page 388)	This is GetIDs, a member of class TSessions.
	GetIsActive (see page 389)	
	GetItemCount (see page 389)	Gets the number of items in the specified session.
	GetItems (see page 389)	
	GetSession (see page 389)	
	GetSessionIDGen (see page 390)	This is GetSessionIDGen, a member of class TSessions.
	GetStartTime (see page 390)	Gets the start time for the specified session.
	GetTerminated (see page 390)	
	GetTimeoutMinutes (see page 390)	Gets the number of minutes before the session times out.
	GetTouchTime (see page 391)	
	ItemValue (see page 391)	
	LoadFromFile (see page 391)	Loads the session information from a file.
	LoadFromStream (see page 391)	Loads the session information from a stream.
	NotifyStatusChange (see page 392)	
	SaveToFile (see page 392)	Saves the session information to a file.
	SaveToStream (see page 392)	Saves the session output to a stream.
	SessionExists (see page 393)	Indicates whether the specified session exists.
	SetItemValue (see page 393)	
	SetTimeoutMinutes (see page 393)	Sets the timeout minutes for the specified session.
	StartSession (see page 393)	Starts a new session.
	TerminateSession (see page 394)	Terminates the specified session.
	TouchSession (see page 394)	

6.1.6.4.1 TSessions.CheckStatusChange Method

CheckStatusChanges checks if the session status has changed.

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

Call CheckStatusChange to check if the session status has changed. Set CheckStatusChange to T true to monitor changes to the session's status. Otherwise, set it to F false. (more here)

See Also

StatusFlags (see page 382)

6.1.6.4.2 TSessions.ConnectActiveSession Method

Pascal

```
function ConnectActiveSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean ConnectActiveSession(TSessionID AID);
```

Description

This is ConnectActiveSession, a member of class TSessions.

6.1.6.4.3 TSessions.ConnectSession Method

Pascal

```
function ConnectSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean ConnectSession(TSessionID AID);
```

See Also

DisconnectSession (see page 387)

6.1.6.4.4 TSessions.CreateSessionIDGen Method

Pascal

```
function CreateSessionIDGen: TAbstractSessionIDGen; virtual;
```

C++

```
virtual __fastcall TAbstractSessionIDGen CreateSessionIDGen();
```

Description

This is CreateSessionIDGen, a member of class TSessions.

6.1.6.4.5 TSessions.DisconnectSession Method

Disconnects the specified session.

Pascal

```
procedure DisconnectSession(AID: TSessionID);
```

C++

```
__fastcall DisconnectSession(TSessionID AID);
```

Description

Call DisconnectSession to disconnect the session specified by the TSessionID (see page 1129).

See Also

ConnectSession (see page 387)

6.1.6.4.6 TSessions.GetExpirationTime Method

Gets the expiration time for the specified session.

Pascal

```
function GetExpirationTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime(TSessionID AID);
```

Description

Call GetExpirationTime to get the expiration time for the session specified by the TSessionID (see page 1129) in a TDateTime (see page 3323) format.

See Also

ExpirationTime (see page 381)

6.1.6.4.7 TSessions.GetExpired Method

Pascal

```
function GetExpired(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetExpired(TSessionID AID);
```

See Also

Expired (see page 381)

6.1.6.4.8 TSessions.GetIDCount Method

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

This is GetIDCount, a member of class TSessions.

6.1.6.4.9 TSessions.GetIDs Method

Pascal

```
procedure GetIDs(var AList: TList);
```

C++

```
__fastcall GetIDs(TList AList);
```

Description

This is GetIDs, a member of class TSessions.

6.1.6.4.10 TSessions.GetIsActive Method

Pascal

```
function GetIsActive(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetIsActive(TSessionID AID);
```

6.1.6.4.11 TSessions.GetItemCount Method

Gets the number of items in the specified session.

Pascal

```
function GetItemCount(AID: TSessionID): Integer;
```

C++

```
__fastcall int GetItemCount(TSessionID AID);
```

Description

Call GetItemCount to get the number of items (returned as an integer) in the session specified by TSessionID (see page 1129). more here

See Also

Items (see page 382)

Items

6.1.6.4.12 TSessions.GetItems Method

Pascal

```
procedure GetItems(AID: TSessionID; var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TSessionID AID, TAbstractNamedVariants AItems);
```

See Also

Items (see page 382)

GetItemCount (see page 389)

6.1.6.4.13 TSessions.GetSession Method

Pascal

```
function GetSession(AID: TSessionID; var AItem: TSessionItem): Boolean;
```

C++

```
__fastcall Boolean GetSession(TSessionID AID, TSessionItem AItem);
```

6.1.6.4.14 TSessions.GetSessionIDGen Method

Pascal

```
function GetSessionIDGen: TAbstractSessionIDGen;
```

C++

```
__fastcall TAbstractSessionIDGen GetSessionIDGen();
```

Description

This is GetSessionIDGen, a member of class TSessions.

6.1.6.4.15 TSessions.GetStartTime Method

Gets the start time for the specified session.

Pascal

```
function GetStartTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime(TSessionID AID);
```

Description

Call GetStartTime to get the start time for a session specified by TSessionID (see page 1129) in a TDateTime (see page 3323) format.

See Also

StartTime (see page 382)

6.1.6.4.16 TSessions.GetTerminated Method

Pascal

```
function GetTerminated(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean GetTerminated(TSessionID AID);
```

See Also

Terminated (see page 383)

6.1.6.4.17 TSessions.GetTimeoutMinutes Method

Gets the number of minutes before the session times out.

Pascal

```
function GetTimeoutMinutes(AID: TSessionID): Integer;
```

C++

```
__fastcall int GetTimeoutMinutes(TSessionID AID);
```

Description

Call GetTimeoutMinutes to get the number of minutes (represented as an Integer value) before the session specified by TSessionID (see page 1129) is timed out.

See Also

TimeoutMinutes (see page 383)

6.1.6.4.18 TSessions.GetTouchTime Method

Pascal

```
function GetTouchTime(AID: TSessionID): TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime(TSessionID AID);
```

See Also

TouchTime (see page 384)

6.1.6.4.19 TSessions.ItemValue Method

Pascal

```
function ItemValue(AID: TSessionID; const AName: string): Variant;
```

C++

```
__fastcall Variant ItemValue(TSessionID AID, const UnicodeString AName);
```

See Also

Items (see page 382)

6.1.6.4.20 TSessions.LoadFromFile Method

Loads the session information from a file.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString Filename);
```

Description

Call LoadFromFile to load the session information from the specified file.

See Also

LoadFromStream (see page 391)

SaveToFile (see page 392)

6.1.6.4.21 TSessions.LoadFromStream Method

Loads the session information from a stream.

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

Call LoadFromStream to load session information from the specified stream. (more here)

See Also

LoadFromFile (see page 391)

SaveToStream (see page 392)

6.1.6.4.22 TSessions.NotifyStatusChange Method

Pascal

```
procedure NotifyStatusChange(AStatusChange: TAbstractSessionStatusChange);
```

C++

```
__fastcall NotifyStatusChange(TAbstractSessionStatusChange AStatusChange);
```

See Also

StatusFlags (see page 382)

CheckStatusChange (see page 386)

6.1.6.4.23 TSessions.SaveToFile Method

Saves the session information to a file.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString Filename);
```

Description

Call SaveToFile to save the session information to the specified file. (more here)

See Also

LoadFromFile (see page 391)

SaveToStream (see page 392)

6.1.6.4.24 TSessions.SaveToStream Method

Saves the session output to a stream.

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

Call SaveToStream to save the session output to a stream. (more here)

See Also

LoadFromStream (see page 391)

SaveToFile (see page 392)

6.1.6.4.25 TSessions.SessionExists Method

Indicates whether the specified session exists.

Pascal

```
function SessionExists(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean SessionExists(TSessionID AID);
```

Description

Call SessionExists to indicate whether the specified session exists. Set SessionExists to true to indicate to the server that the session is still alive, or false to indicate that it is not.

6.1.6.4.26 TSessions.SetItemValue Method

Pascal

```
procedure SetItemValue(AID: TSessionID; const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetItemValue(TSessionID AID, const UnicodeString AName, const Variant AValue);
```

See Also

ItemValue (see page 391)

6.1.6.4.27 TSessions.SetTimeoutMinutes Method

Sets the timeout minutes for the specified session.

Pascal

```
procedure SetTimeoutMinutes(AID: TSessionID; AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(TSessionID AID, int AValue);
```

Description

Call SetTimeoutMinutes to set the number of minutes before a specified session is timed-out.

See Also

TimeoutMinutes (see page 383)

6.1.6.4.28 TSessions.StartSession Method

Starts a new session.

Pascal

```
function StartSession: TSessionID;
```

C++

```
__fastcall TSessionID StartSession();
```

Description

Call StartSession to start a new session and assign it a session ID. (more here).

See Also

TerminateSession (see page 394)

6.1.6.4.29 TSessions.TerminateSession Method

Terminates the specified session.

Pascal

```
function TerminateSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean TerminateSession(TSessionID AID);
```

Description

Call TerminateSession to terminated the session specified by the TSessionID (see page 1129). Set TerminateSession to T true to terminate the session, otherwise set it to F false.

See Also

StartSession (see page 393)

6.1.6.4.30 TSessions.TouchSession Method

Pascal

```
function TouchSession(AID: TSessionID): Boolean;
```

C++

```
__fastcall Boolean TouchSession(TSessionID AID);
```

See Also

TouchTime (see page 384)

GetTouchTime (see page 391)

6.1.6.5 TSessions Properties

The properties of the TSessions class are listed here.

Public Properties

	Name	Description
	ActiveCount (see page 394)	This is ActiveCount, a member of class TSessions.
	FileName (see page 395)	Gets or sets the file name for the session.
	MaxSessions (see page 395)	This is MaxSessions, a member of class TSessions.

6.1.6.5.1 TSessions.ActiveCount Property

Pascal

```
property ActiveCount: Integer;
```

C++

```
__property int ActiveCount;
```

Description

This is ActiveCount, a member of class TSessions.

6.1.6.5.2 TSessions.FileName Property

Gets or sets the file name for the session.

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

Use FileName to get or set the file name for the session. (more here)

See Also

SaveToFile (see page 392)

6.1.6.5.3 TSessions.MaxSessions Property

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

This is MaxSessions, a member of class TSessions.

6.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSessionStatusFlag (see page 396)	This is record SessColn.TSessionStatusFlag.

6.2.1 SessColn.TSessionStatusFlag Enumeration

Pascal

```
TSessionStatusFlag = (
  ssStarted,
  ssTerminated,
  ssTimeOut
);
```

C++

```
enum TSessionStatusFlag {
  ssStarted,
  ssTerminated,
  ssTimeOut
};
```

File

SessColn

Description

This is record SessColn.TSessionStatusFlag.

6.3 Types

The following table lists types in this documentation.

Types

Name	Description
TSessionStatusFlags (see page 397)	This is type SessColn.TSessionStatusFlags.

6.3.1 SessColn.TSessionStatusFlags Type

Pascal

```
TSessionStatusFlags = set of TSessionStatusFlag;
```

C++

```
TSessionStatusFlag TSessionStatusFlags;
```

File

SessColn

Description

This is type SessColn.TSessionStatusFlags.

6.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Sessions (see page 398)	This is variable SessColn.Sessions.

6.4.1 SessColn.Sessions Variable

Pascal

```
Sessions: TSessions;
```

C++

```
TSessions Sessions;
```

File

SessColn

Description

This is variable SessColn.Sessions.

7 ShadowWnd Namespace

This is namespace ShadowWnd.

Topics

Name	Description
Classes (see page 400)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 406)	The following table lists structs, records, enums in this documentation.

Classes

	Name	Description
	TShadowWindow (see page 400)	This is class ShadowWnd.TShadowWindow.

Structs, Records, Enums

	Name	Description
	TControlSide (see page 406)	This is record ShadowWnd.TControlSide.

7.1 Classes

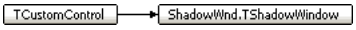
The following table lists classes in this documentation.

Classes

	Name	Description
	TShadowWindow (see page 400)	This is class ShadowWnd.TShadowWindow.

7.1.1 TShadowWindow Class

Class Hierarchy



Pascal

```
TShadowWindow = class(TCustomControl);
```

C++

```
class TShadowWindow : public TCustomControl;
```

File

ShadowWnd

Description

This is class ShadowWnd.TShadowWindow.

7.1.1.1 TShadowWindow Members

The following tables list the members exposed by TShadowWindow.

Protected Methods

	Name	Description
	CreateParams (see page 402)	Initializes a window-creation parameter data structure.
	Notification (see page 403)	Responds to notifications that components are being created or destroyed.
	Paint (see page 403)	Renders the image of a custom control.
	WMEraseBkgnd (see page 404)	This is WMEraseBkgnd, a member of class TShadowWindow.

Public Methods

	Name	Description
	Create (see page 401)	Creates an instance of TCustomControl.
	CreateShadow (see page 401)	This is CreateShadow, a member of class TShadowWindow.
	Destroy (see page 401)	Destroys an instance of TCustomControl.
	SetBounds (see page 404)	Sets the windowed control's boundary properties all at once.

Public Properties

	Name	Description
	Control (see page 405)	This is Control, a member of class TShadowWindow.

	Side (see page 405)	This is Side, a member of class TShadowWindow.

7.1.1.2 TShadowWindow.Create Constructor

Creates an instance of TCustomControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShadowWindow(TComponent * AOwner);
```

Description

Calling Create constructs and initializes an instance of TCustomControl. However, you should never attempt to instantiate a TCustomControl. This class is intended solely as a base class from which other control classes descend and you should only call Create to instantiate one of these descendants. Controls placed on forms at design time are created automatically.

After calling the inherited constructor, Create initializes the control, including the following property settings:

Initializes a TCanvas object for its Canvas property.

When overriding Create, always call the inherited Create method first, then proceed with the control's initialization. Remember to specify the override directive when overriding the Create method.

Note: If a component's constructor allocates resources or memory, also override the destructor to free those resources.

See Also

Create

Destroy

Canvas

Owner

7.1.1.3 TShadowWindow.CreateShadow Constructor

Pascal

```
constructor CreateShadow(AOwner: TComponent; ControlSide: TControlSide); virtual;
```

C++

```
virtual __fastcall TShadowWindow(TComponent * AOwner, TControlSide ControlSide);
```

Description

This is CreateShadow, a member of class TShadowWindow.

7.1.1.4 TShadowWindow.Destroy Destructor

Destroys an instance of TCustomControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShadowWindow();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the control is destroyed, it:

Destroys the TCanvas object in its Canvas property.

Override Destroy to free any memory or resources allocated in the Create method. When declaring a Destroy method in a descendant control type, always add the override directive to the declaration and call the inherited Destroy as the last statement in the redeclared method.

When a control is created, Delphi automatically provides exception handling around the constructor call. The destructor is called if an exception escapes from the constructor. This means that the destructor code must be prepared to clean up a partially constructed instance. Check all data fields for zero before disposing of their contents.

See Also

System::TObject::Free (see page 3372)

delete

Create

7.1.1.5 TShadowWindow Methods

The methods of the TShadowWindow class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 402)	Initializes a window-creation parameter data structure.
	Notification (see page 403)	Responds to notifications that components are being created or destroyed.
	Paint (see page 403)	Renders the image of a custom control.
	WMEraseBkgnd (see page 404)	This is WMEraseBkgnd, a member of class TShadowWindow.

Public Methods

	Name	Description
	SetBounds (see page 404)	Sets the windowed control's boundary properties all at once.

7.1.1.5.1 TShadowWindow.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override

CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

7.1.1.5.2 TShadowWindow.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

7.1.1.5.3 TShadowWindow.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT

messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

7.1.1.5.4 TShadowWindow.SetBounds Method

Sets the windowed control's boundary properties all at once.

Pascal

```
procedure SetBounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer);
override;
```

C++

```
virtual __fastcall SetBounds(int ALeft, int ATop, int AWidth, int AHeight);
```

Description

Use SetBounds to change all of the control's boundary properties at once. The same effect can be achieved by setting the Left, Top, Width, and Height properties separately. By setting all four properties at once, SetBounds ensures that the control will not repaint between changes

Specify the values for the Left, Top, Width, and Height properties as the value of the ALeft, ATop, AWidth, and AHeight parameters, respectively.

See Also

Height

Left

Top

Width

7.1.1.5.5 TShadowWindow.WMEraseBkgnd Method

Pascal

```
procedure WMEraseBkgnd(var Message: TWMEraseBkgnd); message WM_ERASEBKGND;
```

C++

```
__fastcall WMEraseBkgnd(TWMEraseBkgnd Message);
```

Description

This is WMEraseBkgnd, a member of class TShadowWindow.

7.1.1.6 TShadowWindow Properties

The properties of the TShadowWindow class are listed here.

Public Properties

	Name	Description
	Control (see page 405)	This is Control, a member of class TShadowWindow.
	Side (see page 405)	This is Side, a member of class TShadowWindow.

7.1.1.6.1 TShadowWindow.Control Property

Pascal

```
property Control: TControl;
```

C++

```
__property TControl * Control;
```

Description

This is Control, a member of class TShadowWindow.

7.1.1.6.2 TShadowWindow.Side Property

Pascal

```
property Side: TControlSide;
```

C++

```
__property TControlSide Side;
```

Description

This is Side, a member of class TShadowWindow.

7.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TControlSide (see page 406)	This is record ShadowWnd.TControlSide.

7.2.1 ShadowWnd.TControlSide Enumeration

Pascal

```
TControlSide = (
  csRight,
  csBottom
);
```

C++

```
enum TControlSide {
  csRight,
  csBottom
};
```

File

ShadowWnd

Description

This is record ShadowWnd.TControlSide.

8 SHDocVw Namespace

This is namespace SHDocVw.

Topics

Name	Description
Classes (see page 413)	The following table lists classes in this documentation.
Interfaces (see page 616)	The following table lists interfaces in this documentation.
Types (see page 802)	The following table lists types in this documentation.
Constants (see page 838)	The following table lists constants in this documentation.

Classes

	Name	Description
	CoCScriptErrorList (see page 413)	This is class SHDocVw.CoCScriptErrorList.
	CoInternetExplorer (see page 414)	This is class SHDocVw.CoInternetExplorer.
	CoSearchAssistantOC (see page 415)	This is class SHDocVw.CoSearchAssistantOC.
	CoShellBrowserWindow (see page 417)	This is class SHDocVw.CoShellBrowserWindow.
	CoShellNameSpace (see page 418)	This is class SHDocVw.CoShellNameSpace.
	CoShellUIHelper (see page 419)	This is class SHDocVw.CoShellUIHelper.
	CoShellWindows (see page 420)	This is class SHDocVw.CoShellWindows.
	TInternetExplorer (see page 421)	This is class SHDocVw.TInternetExplorer.
	TShellNameSpace (see page 466)	This is class SHDocVw.TShellNameSpace.
	TShellUIHelper (see page 483)	This is class SHDocVw.TShellUIHelper.
	TShellWindows (see page 494)	This is class SHDocVw.TShellWindows.
	TWebBrowser (see page 504)	TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	TWebBrowser_V1 (see page 569)	This is class SHDocVw.TWebBrowser_V1.

Constants

Name	Description
CGID_IWebBrowser (see page 840)	This is constant SHDocVw.CGID_IWebBrowser.
CLASS_CScriptErrorList (see page 841)	This is constant SHDocVw.CLASS_CScriptErrorList.
CLASS_InternetExplorer (see page 841)	This is constant SHDocVw.CLASS_InternetExplorer.
CLASS_SearchAssistantOC (see page 841)	This is constant SHDocVw.CLASS_SearchAssistantOC.
CLASS_ShellBrowserWindow (see page 842)	This is constant SHDocVw.CLASS_ShellBrowserWindow.
CLASS_ShellNameSpace (see page 842)	This is constant SHDocVw.CLASS_ShellNameSpace.
CLASS_ShellUIHelper (see page 842)	This is constant SHDocVw.CLASS_ShellUIHelper.

CLASS_ShellWindows (see page 843)	This is constant SHDocVw.CLASS_ShellWindows.
CLASS_WebBrowser (see page 843)	This is constant SHDocVw.CLASS_WebBrowser.
CLASS_WebBrowser_V1 (see page 843)	This is constant SHDocVw.CLASS_WebBrowser_V1.
CSC_NAVIGATEBACK (see page 844)	This is constant SHDocVw.CSC_NAVIGATEBACK.
CSC_NAVIGATEFORWARD (see page 844)	This is constant SHDocVw.CSC_NAVIGATEFORWARD.
CSC_UPDATECOMMANDS (see page 844)	This is constant SHDocVw.CSC_UPDATECOMMANDS.
DIID__SearchAssistantEvents (see page 845)	This is constant SHDocVw.DIID__SearchAssistantEvents.
DIID_DShellNameSpaceEvents (see page 845)	This is constant SHDocVw.DIID_DShellNameSpaceEvents.
DIID_DShellWindowsEvents (see page 845)	This is constant SHDocVw.DIID_DShellWindowsEvents.
DIID_DWebBrowserEvents (see page 846)	This is constant SHDocVw.DIID_DWebBrowserEvents.
DIID_DWebBrowserEvents2 (see page 846)	This is constant SHDocVw.DIID_DWebBrowserEvents2.
dtlOcxPage (see page 846)	This is constant SHDocVw.dtlOcxPage.
dtlServerPage (see page 847)	This is constant SHDocVw.dtlServerPage.
HTMLID_FIND (see page 847)	This is constant SHDocVw.HTMLID_FIND.
HTMLID_OPTIONS (see page 847)	This is constant SHDocVw.HTMLID_OPTIONS.
HTMLID_VIEWSOURCE (see page 848)	This is constant SHDocVw.HTMLID_VIEWSOURCE.
IID_IScriptErrorList (see page 848)	This is constant SHDocVw.IID_IScriptErrorList.
IID_ISearch (see page 848)	This is constant SHDocVw.IID_ISearch.
IID_ISearchAssistantOC (see page 849)	This is constant SHDocVw.IID_ISearchAssistantOC.
IID_ISearchAssistantOC2 (see page 849)	This is constant SHDocVw.IID_ISearchAssistantOC2.
IID_ISearchAssistantOC3 (see page 849)	This is constant SHDocVw.IID_ISearchAssistantOC3.
IID_ISearches (see page 850)	This is constant SHDocVw.IID_ISearches.
IID_IShellFavoritesNameSpace (see page 850)	This is constant SHDocVw.IID_IShellFavoritesNameSpace.
IID_IShellNameSpace (see page 850)	This is constant SHDocVw.IID_IShellNameSpace.
IID_IShellUIHelper (see page 851)	This is constant SHDocVw.IID_IShellUIHelper.
IID_IShellWindows (see page 851)	This is constant SHDocVw.IID_IShellWindows.
IID_IWebBrowser (see page 851)	This is constant SHDocVw.IID_IWebBrowser.
IID_IWebBrowser2 (see page 852)	This is constant SHDocVw.IID_IWebBrowser2.
IID_IWebBrowserApp (see page 852)	This is constant SHDocVw.IID_IWebBrowserApp.
LIBID_SHDocVw (see page 852)	This is constant SHDocVw.LIBID_SHDocVw.
navAllowAutosearch (see page 853)	This is constant SHDocVw.navAllowAutosearch.
navBrowserBar (see page 853)	This is constant SHDocVw.navBrowserBar.
navNoHistory (see page 853)	This is constant SHDocVw.navNoHistory.
navNoReadFromCache (see page 854)	This is constant SHDocVw.navNoReadFromCache.
navNoWriteToCache (see page 854)	This is constant SHDocVw.navNoWriteToCache.
navOpenInNewWindow (see page 854)	This is constant SHDocVw.navOpenInNewWindow.
OLECMDEXECOPT_DODEFAULT (see page 855)	This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.
OLECMDEXECOPT_DONTPROMPTUSER (see page 855)	This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.
OLECMDEXECOPT_PROMPTUSER (see page 855)	This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.
OLECMDEXECOPT_SHOWHELP (see page 856)	This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.
OLECMDF_DEFHIDEONCTXTMENU (see page 856)	This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.
OLECMDF_ENABLED (see page 856)	This is constant SHDocVw.OLECMDF_ENABLED.
OLECMDF_INVISIBLE (see page 857)	This is constant SHDocVw.OLECMDF_INVISIBLE.
OLECMDF_LATCHED (see page 857)	This is constant SHDocVw.OLECMDF_LATCHED.
OLECMDF_NINCHED (see page 857)	This is constant SHDocVw.OLECMDF_NINCHED.
OLECMDF_SUPPORTED (see page 858)	This is constant SHDocVw.OLECMDF_SUPPORTED.
OLECMDID_ALLOWUILESSSAVEAS (see page 858)	This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.
OLECMDID_CLEARSELECTION (see page 858)	This is constant SHDocVw.OLECMDID_CLEARSELECTION.
OLECMDID_CLOSE (see page 859)	This is constant SHDocVw.OLECMDID_CLOSE.
OLECMDID_COPY (see page 859)	This is constant SHDocVw.OLECMDID_COPY.
OLECMDID_CUT (see page 859)	This is constant SHDocVw.OLECMDID_CUT.
OLECMDID_DELETE (see page 860)	This is constant SHDocVw.OLECMDID_DELETE.
OLECMDID_DONTDOWNLOADCSS (see page 860)	This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.
OLECMDID_ENABLE_INTERACTION (see page 860)	This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.

OLECMDID_FIND (see page 861)	This is constant SHDocVw.OLECMDID_FIND.
OLECMDID_GETPRINTTEMPLATE (see page 861)	This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.
OLECMDID_GETZOOMRANGE (see page 861)	This is constant SHDocVw.OLECMDID_GETZOOMRANGE.
OLECMDID_HIDETOOLBARS (see page 862)	This is constant SHDocVw.OLECMDID_HIDETOOLBARS.
OLECMDID_HTTPEQUIV (see page 862)	This is constant SHDocVw.OLECMDID_HTTPEQUIV.
OLECMDID_HTTPEQUIV_DONE (see page 862)	This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.
OLECMDID_NEW (see page 863)	This is constant SHDocVw.OLECMDID_NEW.
OLECMDID_ONTOOLBARACTIVATED (see page 863)	This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.
OLECMDID_ONUNLOAD (see page 863)	This is constant SHDocVw.OLECMDID_ONUNLOAD.
OLECMDID_OPEN (see page 864)	This is constant SHDocVw.OLECMDID_OPEN.
OLECMDID_PAGESETUP (see page 864)	This is constant SHDocVw.OLECMDID_PAGESETUP.
OLECMDID_PASTE (see page 864)	This is constant SHDocVw.OLECMDID_PASTE.
OLECMDID_PASTESPECIAL (see page 865)	This is constant SHDocVw.OLECMDID_PASTESPECIAL.
OLECMDID_PREREFRESH (see page 865)	This is constant SHDocVw.OLECMDID_PREREFRESH.
OLECMDID_PRINT (see page 865)	This is constant SHDocVw.OLECMDID_PRINT.
OLECMDID_PRINT2 (see page 866)	This is constant SHDocVw.OLECMDID_PRINT2.
OLECMDID_PRINTPREVIEW (see page 866)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW.
OLECMDID_PRINTPREVIEW2 (see page 866)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.
OLECMDID_PROPERTIES (see page 867)	This is constant SHDocVw.OLECMDID_PROPERTIES.
OLECMDID_PROPERTYBAG2 (see page 867)	This is constant SHDocVw.OLECMDID_PROPERTYBAG2.
OLECMDID_REDO (see page 867)	This is constant SHDocVw.OLECMDID_REDO.
OLECMDID_REFRESH (see page 868)	This is constant SHDocVw.OLECMDID_REFRESH.
OLECMDID_SAVE (see page 868)	This is constant SHDocVw.OLECMDID_SAVE.
OLECMDID_SAVEAS (see page 868)	This is constant SHDocVw.OLECMDID_SAVEAS.
OLECMDID_SAVECOPYAS (see page 869)	This is constant SHDocVw.OLECMDID_SAVECOPYAS.
OLECMDID_SELECTALL (see page 869)	This is constant SHDocVw.OLECMDID_SELECTALL.
OLECMDID_SETDOWNLOADSTATE (see page 869)	This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.
OLECMDID_SETPRINTTEMPLATE (see page 870)	This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.
OLECMDID_SETPROGRESSMAX (see page 870)	This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.
OLECMDID_SETPROGRESSPOS (see page 870)	This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.
OLECMDID_SETPROGRESSTEXT (see page 871)	This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.
OLECMDID_SETTITLE (see page 871)	This is constant SHDocVw.OLECMDID_SETTITLE.
OLECMDID_SHOWFIND (see page 871)	This is constant SHDocVw.OLECMDID_SHOWFIND.
OLECMDID_SHOWMESSAGE (see page 872)	This is constant SHDocVw.OLECMDID_SHOWMESSAGE.
OLECMDID_SHOWPAGESETUP (see page 872)	This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.
OLECMDID_SHOWPRINT (see page 872)	This is constant SHDocVw.OLECMDID_SHOWPRINT.
OLECMDID_SHOWSCRIPTERROR (see page 873)	This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.
OLECMDID_SPELL (see page 873)	This is constant SHDocVw.OLECMDID_SPELL.
OLECMDID_STOP (see page 873)	This is constant SHDocVw.OLECMDID_STOP.
OLECMDID_STOPDOWNLOAD (see page 874)	This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.
OLECMDID_UNDO (see page 874)	This is constant SHDocVw.OLECMDID_UNDO.
OLECMDID_UPDATECOMMANDS (see page 874)	This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.
OLECMDID_UPDATEPAGESTATUS (see page 875)	This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.
OLECMDID_ZOOM (see page 875)	This is constant SHDocVw.OLECMDID_ZOOM.
READYSTATE_COMPLETE (see page 875)	This is constant SHDocVw.READYSTATE_COMPLETE.
READYSTATE_INTERACTIVE (see page 876)	This is constant SHDocVw.READYSTATE_INTERACTIVE.
READYSTATE_LOADED (see page 876)	This is constant SHDocVw.READYSTATE_LOADED.
READYSTATE_LOADING (see page 876)	This is constant SHDocVw.READYSTATE_LOADING.
READYSTATE_UNINITIALIZED (see page 877)	This is constant SHDocVw.READYSTATE_UNINITIALIZED.
REFRESH_COMPLETELY (see page 877)	This is constant SHDocVw.REFRESH_COMPLETELY.
REFRESH_CONTINUE (see page 877)	This is constant SHDocVw.REFRESH_CONTINUE.
REFRESH_IFEXPIRED (see page 878)	This is constant SHDocVw.REFRESH_IFEXPIRED.
REFRESH_NORMAL (see page 878)	This is constant SHDocVw.REFRESH_NORMAL.
secureLockIconMixed (see page 878)	This is constant SHDocVw.secureLockIconMixed.

secureLockIconSecure128Bit (see page 879)	This is constant SHDocVw.secureLockIconSecure128Bit.
secureLockIconSecure40Bit (see page 879)	This is constant SHDocVw.secureLockIconSecure40Bit.
secureLockIconSecure56Bit (see page 879)	This is constant SHDocVw.secureLockIconSecure56Bit.
secureLockIconSecureFortezza (see page 880)	This is constant SHDocVw.secureLockIconSecureFortezza.
secureLockIconSecureUnknownBits (see page 880)	This is constant SHDocVw.secureLockIconSecureUnknownBits.
secureLockIconUnsecure (see page 880)	This is constant SHDocVw.secureLockIconUnsecure.
SHDocVwMajorVersion (see page 881)	This is constant SHDocVw.SHDocVwMajorVersion.
SHDocVwMinorVersion (see page 881)	This is constant SHDocVw.SHDocVwMinorVersion.
SWC_3RDPARTY (see page 881)	This is constant SHDocVw.SWC_3RDPARTY.
SWC_BROWSER (see page 882)	This is constant SHDocVw.SWC_BROWSER.
SWC_CALLBACK (see page 882)	This is constant SHDocVw.SWC_CALLBACK.
SWC_EXPLORER (see page 882)	This is constant SHDocVw.SWC_EXPLORER.
SWFO_COOKIEPASSED (see page 883)	This is constant SHDocVw.SWFO_COOKIEPASSED.
SWFO_INCLUDEPENDING (see page 883)	This is constant SHDocVw.SWFO_INCLUDEPENDING.
SWFO_NEEDDISPATCH (see page 883)	This is constant SHDocVw.SWFO_NEEDDISPATCH.

Interfaces

	Name	Description
	_SearchAssistantEvents (see page 616)	This is class SHDocVw._SearchAssistantEvents.
	DShellNameSpaceEvents (see page 618)	This is class SHDocVw.DShellNameSpaceEvents.
	DShellWindowsEvents (see page 619)	This is class SHDocVw.DShellWindowsEvents.
	DWebBrowserEvents (see page 621)	This is class SHDocVw.DWebBrowserEvents.
	DWebBrowserEvents2 (see page 626)	This is class SHDocVw.DWebBrowserEvents2.
	IScriptErrorList (see page 635)	This is class SHDocVw.IScriptErrorList.
	IScriptErrorListDisp (see page 639)	This is class SHDocVw.IScriptErrorListDisp.
	ISearch (see page 644)	This is class SHDocVw.ISearch.
	ISearchAssistantOC (see page 646)	This is class SHDocVw.ISearchAssistantOC.
	ISearchAssistantOC2 (see page 656)	This is class SHDocVw.ISearchAssistantOC2.
	ISearchAssistantOC2Disp (see page 658)	This is class SHDocVw.ISearchAssistantOC2Disp.
	ISearchAssistantOC3 (see page 666)	This is class SHDocVw.ISearchAssistantOC3.
	ISearchAssistantOC3Disp (see page 669)	This is class SHDocVw.ISearchAssistantOC3Disp.
	ISearchAssistantOCDisp (see page 677)	This is class SHDocVw.ISearchAssistantOCDisp.
	ISearchDisp (see page 685)	This is class SHDocVw.ISearchDisp.
	ISearches (see page 686)	This is class SHDocVw.ISearches.
	ISearchesDisp (see page 689)	This is class SHDocVw.ISearchesDisp.
	IShellFavoritesNameSpace (see page 691)	This is class SHDocVw.IShellFavoritesNameSpace.
	IShellFavoritesNameSpaceDisp (see page 695)	This is class SHDocVw.IShellFavoritesNameSpaceDisp.
	IShellNameSpace (see page 700)	This is class SHDocVw.IShellNameSpace.
	IShellNameSpaceDisp (see page 709)	This is class SHDocVw.IShellNameSpaceDisp.
	IShellUIHelper (see page 716)	This is class SHDocVw.IShellUIHelper.
	IShellUIHelperDisp (see page 721)	This is class SHDocVw.IShellUIHelperDisp.
	IShellWindows (see page 725)	This is class SHDocVw.IShellWindows.
	IShellWindowsDisp (see page 729)	This is class SHDocVw.IShellWindowsDisp.
	IWebBrowser (see page 733)	IWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	IWebBrowser2 (see page 749)	IWebBrowser2 provides access to the Web browser functionality of Microsoft Shell Doc Object and Control Library (SHDOCVW.DLL).
	IWebBrowser2Disp (see page 763)	This is class SHDocVw.IWebBrowser2Disp.
	IWebBrowserApp (see page 776)	This is class SHDocVw.IWebBrowserApp.
	IWebBrowserAppDisp (see page 785)	This is class SHDocVw.IWebBrowserAppDisp.
	IWebBrowserDisp (see page 795)	This is class SHDocVw.IWebBrowserDisp.

Types

Name	Description
BrowserNavConstants (see page 804)	This is type SHDocVw.BrowserNavConstants.

CommandStateChangeConstants (see page 804)	Represents an alias for TOleEnum.
CScriptErrorList (see page 804)	This is type SHDocVw.CScriptErrorList.
InternetExplorer (see page 805)	This is type SHDocVw.InternetExplorer.
OLECMDEXECOPT (see page 805)	OLECMDF (see page 806) influences the way in which an OLECMDID (see page 806) command is executed.
OLECMDF (see page 806)	OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.
OLECMDID (see page 806)	OLECMDID represents a command that can be executed by an Automation object.
POleVariant1 (see page 807)	POleVariant1 is a pointer to a System::OleVariant (see page 3274) variable.
RefreshConstants (see page 807)	This is type SHDocVw.RefreshConstants.
SearchAssistantOC (see page 808)	This is type SHDocVw.SearchAssistantOC.
SecureLockIconConstants (see page 808)	This is type SHDocVw.SecureLockIconConstants.
ShellBrowserWindow (see page 808)	This is type SHDocVw.ShellBrowserWindow.
ShellNameSpace (see page 809)	This is type SHDocVw.ShellNameSpace.
ShellUIHelper (see page 809)	This is type SHDocVw.ShellUIHelper.
ShellWindowFindWindowOptions (see page 809)	This is type SHDocVw.ShellWindowFindWindowOptions.
ShellWindows (see page 810)	This is type SHDocVw.ShellWindows.
ShellWindowTypeConstants (see page 810)	This is type SHDocVw.ShellWindowTypeConstants.
tagREADYSTATE (see page 810)	This is type SHDocVw.tagREADYSTATE.
TInternetExplorerBeforeNavigate2 (see page 811)	This is type SHDocVw.TInternetExplorerBeforeNavigate2.
TInternetExplorerClientToHostWindow (see page 811)	This is type SHDocVw.TInternetExplorerClientToHostWindow.
TInternetExplorerCommandStateChange (see page 811)	This is type SHDocVw.TInternetExplorerCommandStateChange.
TInternetExplorerDocumentComplete (see page 812)	This is type SHDocVw.TInternetExplorerDocumentComplete.
TInternetExplorerFileDownload (see page 812)	This is type SHDocVw.TInternetExplorerFileDownload.
TInternetExplorerNavigateComplete2 (see page 812)	This is type SHDocVw.TInternetExplorerNavigateComplete2.
TInternetExplorerNavigateError (see page 813)	This is type SHDocVw.TInternetExplorerNavigateError.
TInternetExplorerNewWindow2 (see page 813)	This is type SHDocVw.TInternetExplorerNewWindow2.
TInternetExplorerOnFullScreen (see page 813)	This is type SHDocVw.TInternetExplorerOnFullScreen.
TInternetExplorerOnMenuBar (see page 814)	This is type SHDocVw.TInternetExplorerOnMenuBar.
TInternetExplorerOnStatusBar (see page 814)	This is type SHDocVw.TInternetExplorerOnStatusBar.
TInternetExplorerOnTheaterMode (see page 814)	This is type SHDocVw.TInternetExplorerOnTheaterMode.
TInternetExplorerOnToolBar (see page 815)	This is type SHDocVw.TInternetExplorerOnToolBar.
TInternetExplorerOnVisible (see page 815)	This is type SHDocVw.TInternetExplorerOnVisible.
TInternetExplorerPrintTemplateInstantiation (see page 815)	This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.
TInternetExplorerPrintTemplateTeardown (see page 816)	This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.
TInternetExplorerPrivacyImpactedStateChange (see page 816)	This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.
TInternetExplorerProgressChange (see page 816)	This is type SHDocVw.TInternetExplorerProgressChange.
TInternetExplorerPropertyChange (see page 817)	This is type SHDocVw.TInternetExplorerPropertyChange.
TInternetExplorerSetSecureLockIcon (see page 817)	This is type SHDocVw.TInternetExplorerSetSecureLockIcon.
TInternetExplorerStatusTextChange (see page 817)	This is type SHDocVw.TInternetExplorerStatusTextChange.
TInternetExplorerTitleChange (see page 818)	This is type SHDocVw.TInternetExplorerTitleChange.
TInternetExplorerUpdatePageStatus (see page 818)	This is type SHDocVw.TInternetExplorerUpdatePageStatus.
TInternetExplorerWindowClosing (see page 818)	This is type SHDocVw.TInternetExplorerWindowClosing.
TInternetExplorerWindowSetHeight (see page 819)	This is type SHDocVw.TInternetExplorerWindowSetHeight.
TInternetExplorerWindowSetLeft (see page 819)	This is type SHDocVw.TInternetExplorerWindowSetLeft.
TInternetExplorerWindowSetResizable (see page 819)	This is type SHDocVw.TInternetExplorerWindowSetResizable.
TInternetExplorerWindowSetTop (see page 820)	This is type SHDocVw.TInternetExplorerWindowSetTop.
TInternetExplorerWindowSetWidth (see page 820)	This is type SHDocVw.TInternetExplorerWindowSetWidth.
TShellNameSpaceFavoritesSelectionChange (see page 820)	This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.
TShellWindowsWindowRegistered (see page 821)	This is type SHDocVw.TShellWindowsWindowRegistered.
TShellWindowsWindowRevoked (see page 821)	This is type SHDocVw.TShellWindowsWindowRevoked.
TWebBrowser_V1BeforeNavigate (see page 821)	This is type SHDocVw.TWebBrowser_V1BeforeNavigate.
TWebBrowser_V1CommandStateChange (see page 822)	This is type SHDocVw.TWebBrowser_V1CommandStateChange.
TWebBrowser_V1FrameBeforeNavigate (see page 822)	This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.

TWebBrowser_V1FrameNavigateComplete (see page 823)	This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.
TWebBrowser_V1FrameNewWindow (see page 823)	This is type SHDocVw.TWebBrowser_V1FrameNewWindow.
TWebBrowser_V1NavigateComplete (see page 823)	This is type SHDocVw.TWebBrowser_V1NavigateComplete.
TWebBrowser_V1NewWindow (see page 824)	This is type SHDocVw.TWebBrowser_V1NewWindow.
TWebBrowser_V1ProgressChange (see page 824)	This is type SHDocVw.TWebBrowser_V1ProgressChange.
TWebBrowser_V1PropertyChange (see page 824)	This is type SHDocVw.TWebBrowser_V1PropertyChange.
TWebBrowser_V1Quit (see page 825)	This is type SHDocVw.TWebBrowser_V1Quit.
TWebBrowser_V1StatusTextChange (see page 825)	This is type SHDocVw.TWebBrowser_V1StatusTextChange.
TWebBrowser_V1TitleChange (see page 825)	This is type SHDocVw.TWebBrowser_V1TitleChange.
TWebBrowserBeforeNavigate2 (see page 826)	TWebBrowserBeforeNavigate2 is the type for the BeforeNavigate2 event handler on TWebBrowser (see page 504).
TWebBrowserClientToHostWindow (see page 826)	This is type SHDocVw.TWebBrowserClientToHostWindow.
TWebBrowserCommandStateChange (see page 827)	TWebBrowserCommandStateChange is the type for the OnCommandStateChange event handler on TWebBrowser (see page 504).
TWebBrowserDocumentComplete (see page 827)	TWebBrowserDocumentComplete is the type for event handlers that respond when the Service Control manager stops a service.
TWebBrowserFileDownload (see page 828)	This is type SHDocVw.TWebBrowserFileDownload.
TWebBrowserNavigateComplete2 (see page 828)	TWebBrowserNavigateComplete2 is the type for the OnNavigateComplete2 event handler on TWebBrowser (see page 504).
TWebBrowserNavigateError (see page 829)	This is type SHDocVw.TWebBrowserNavigateError.
TWebBrowserNewWindow2 (see page 829)	TWebBrowserNewWindow2 is the type for the OnNewWindow2 event handler on TWebBrowser (see page 504).
TWebBrowserOnFullScreen (see page 830)	This is type SHDocVw.TWebBrowserOnFullScreen.
TWebBrowserOnMenuBar (see page 830)	This is type SHDocVw.TWebBrowserOnMenuBar.
TWebBrowserOnStatusBar (see page 830)	This is type SHDocVw.TWebBrowserOnStatusBar.
TWebBrowserOnTheaterMode (see page 831)	This is type SHDocVw.TWebBrowserOnTheaterMode.
TWebBrowserOnToolBar (see page 831)	This is type SHDocVw.TWebBrowserOnToolBar.
TWebBrowserOnVisible (see page 831)	TWebBrowserOnVisible is the type for the OnVisible event handler on TWebBrowser (see page 504).
TWebBrowserPrintTemplateInstantiation (see page 832)	This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.
TWebBrowserPrintTemplateTeardown (see page 832)	This is type SHDocVw.TWebBrowserPrintTemplateTeardown.
TWebBrowserPrivacyImpactedStateChange (see page 832)	This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.
TWebBrowserProgressChange (see page 833)	TWebBrowserProgressChange is the type for the OnProgressChange event handler on TWebBrowser (see page 504).
TWebBrowserPropertyChange (see page 833)	This is type SHDocVw.TWebBrowserPropertyChange.
TWebBrowserSetSecureLockIcon (see page 834)	This is type SHDocVw.TWebBrowserSetSecureLockIcon.
TWebBrowserStatusTextChange (see page 834)	This is type SHDocVw.TWebBrowserStatusTextChange.
TWebBrowserTitleChange (see page 834)	TWebBrowserTitleChange is the type for the OnTitleChange event handler on TWebBrowser (see page 504).
TWebBrowserUpdatePageStatus (see page 835)	This is type SHDocVw.TWebBrowserUpdatePageStatus.
TWebBrowserWindowClosing (see page 835)	This is type SHDocVw.TWebBrowserWindowClosing.
TWebBrowserWindowSetHeight (see page 835)	This is type SHDocVw.TWebBrowserWindowSetHeight.
TWebBrowserWindowSetLeft (see page 836)	This is type SHDocVw.TWebBrowserWindowSetLeft.
TWebBrowserWindowSetResizable (see page 836)	This is type SHDocVw.TWebBrowserWindowSetResizable.
TWebBrowserWindowSetTop (see page 836)	This is type SHDocVw.TWebBrowserWindowSetTop.
TWebBrowserWindowSetWidth (see page 837)	This is type SHDocVw.TWebBrowserWindowSetWidth.
WebBrowser (see page 837)	This is type SHDocVw.WebBrowser.
WebBrowser_V1 (see page 837)	This is type SHDocVw.WebBrowser_V1.

8.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	CoCScriptErrorList (see page 413)	This is class SHDocVw.CoCScriptErrorList.
	CoInternetExplorer (see page 414)	This is class SHDocVw.CoInternetExplorer.
	CoSearchAssistantOC (see page 415)	This is class SHDocVw.CoSearchAssistantOC.
	CoShellBrowserWindow (see page 417)	This is class SHDocVw.CoShellBrowserWindow.
	CoShellNameSpace (see page 418)	This is class SHDocVw.CoShellNameSpace.
	CoShellUIHelper (see page 419)	This is class SHDocVw.CoShellUIHelper.
	CoShellWindows (see page 420)	This is class SHDocVw.CoShellWindows.
	TInternetExplorer (see page 421)	This is class SHDocVw.TInternetExplorer.
	TShellNameSpace (see page 466)	This is class SHDocVw.TShellNameSpace.
	TShellUIHelper (see page 483)	This is class SHDocVw.TShellUIHelper.
	TShellWindows (see page 494)	This is class SHDocVw.TShellWindows.
	TWebBrowser (see page 504)	TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	TWebBrowser_V1 (see page 569)	This is class SHDocVw.TWebBrowser_V1.

8.1.1 CoCScriptErrorList Class

Class Hierarchy

Pascal

```
CoCScriptErrorList = class;
```

C++

```
class CoCScriptErrorList;
```

File

SHDocVw

Description

This is class SHDocVw.CoCScriptErrorList.

8.1.1.1 CoCScriptErrorList Members

The following tables list the members exposed by CoCScriptErrorList.

Public Methods

	Name	Description
	Create (see page 414)	This is Create, a member of class CoCScriptErrorList.
	CreateRemote (see page 414)	This is CreateRemote, a member of class CoCScriptErrorList.

8.1.1.2 CoCScriptErrorList Methods

The methods of the CoCScriptErrorList class are listed here.

Public Methods

	Name	Description
	Create (see page 414)	This is Create, a member of class CoCScriptErrorList.
	CreateRemote (see page 414)	This is CreateRemote, a member of class CoCScriptErrorList.

8.1.1.2.1 CoCScriptErrorList.Create Method

Pascal

```
class function Create: IScriptErrorList;
```

C++

```
__fastcall IScriptErrorList Create();
```

Description

This is Create, a member of class CoCScriptErrorList.

8.1.1.2.2 CoCScriptErrorList.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IScriptErrorList;
```

C++

```
__fastcall IScriptErrorList CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoCScriptErrorList.

8.1.2 CoInternetExplorer Class

Class Hierarchy

SHDocVw.CoInternetExplorer

Pascal

```
CoInternetExplorer = class;
```

C++

```
class CoInternetExplorer;
```

File

SHDocVw

Description

This is class SHDocVw.CoInternetExplorer.

8.1.2.1 CoInternetExplorer Members

The following tables list the members exposed by CoInternetExplorer.

Public Methods

	Name	Description
	Create (see page 415)	This is Create, a member of class CoInternetExplorer.
	CreateRemote (see page 415)	This is CreateRemote, a member of class CoInternetExplorer.

8.1.2.2 CoInternetExplorer Methods

The methods of the CoInternetExplorer class are listed here.

Public Methods

	Name	Description
	Create (see page 415)	This is Create, a member of class CoInternetExplorer.
	CreateRemote (see page 415)	This is CreateRemote, a member of class CoInternetExplorer.

8.1.2.2.1 CoInternetExplorer.Create Method

Pascal

```
class function Create: IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 Create();
```

Description

This is Create, a member of class CoInternetExplorer.

8.1.2.2.2 CoInternetExplorer.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoInternetExplorer.

8.1.3 CoSearchAssistantOC Class

Class Hierarchy

SHDocVw.CoSearchAssistantOC

Pascal

```
CoSearchAssistantOC = class;
```

C++

```
class CoSearchAssistantOC;
```

File

SHDocVw

Description

This is class SHDocVw.CoSearchAssistantOC.

8.1.3.1 CoSearchAssistantOC Members

The following tables list the members exposed by CoSearchAssistantOC.

Public Methods

	Name	Description
	Create (see page 416)	This is Create, a member of class CoSearchAssistantOC.
	CreateRemote (see page 416)	This is CreateRemote, a member of class CoSearchAssistantOC.

8.1.3.2 CoSearchAssistantOC Methods

The methods of the CoSearchAssistantOC class are listed here.

Public Methods

	Name	Description
	Create (see page 416)	This is Create, a member of class CoSearchAssistantOC.
	CreateRemote (see page 416)	This is CreateRemote, a member of class CoSearchAssistantOC.

8.1.3.2.1 CoSearchAssistantOC.Create Method

Pascal

```
class function Create: ISearchAssistantOC3;
```

C++

```
__fastcall ISearchAssistantOC3 Create();
```

Description

This is Create, a member of class CoSearchAssistantOC.

8.1.3.2.2 CoSearchAssistantOC.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): ISearchAssistantOC3;
```

C++

```
__fastcall ISearchAssistantOC3 CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoSearchAssistantOC.

8.1.4 CoShellBrowserWindow Class

Class Hierarchy

Pascal

```
CoShellBrowserWindow = class;
```

C++

```
class CoShellBrowserWindow;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellBrowserWindow.

8.1.4.1 CoShellBrowserWindow Members

The following tables list the members exposed by CoShellBrowserWindow.

Public Methods

	Name	Description
	Create (see page 417)	This is Create, a member of class CoShellBrowserWindow.
	CreateRemote (see page 418)	This is CreateRemote, a member of class CoShellBrowserWindow.

8.1.4.2 CoShellBrowserWindow Methods

The methods of the CoShellBrowserWindow class are listed here.

Public Methods

	Name	Description
	Create (see page 417)	This is Create, a member of class CoShellBrowserWindow.
	CreateRemote (see page 418)	This is CreateRemote, a member of class CoShellBrowserWindow.

8.1.4.2.1 CoShellBrowserWindow.Create Method

Pascal

```
class function Create: IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 Create();
```

Description

This is Create, a member of class CoShellBrowserWindow.

8.1.4.2.2 CoShellBrowserWindow.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IWebBrowser2;
```

C++

```
__fastcall IWebBrowser2 CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellBrowserWindow.

8.1.5 CoShellNameSpace Class

Class Hierarchy

Pascal

```
CoShellNameSpace = class;
```

C++

```
class CoShellNameSpace;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellNameSpace.

8.1.5.1 CoShellNameSpace Members

The following tables list the members exposed by CoShellNameSpace.

Public Methods

	Name	Description
	Create (see page 419)	This is Create, a member of class CoShellNameSpace.
	CreateRemote (see page 419)	This is CreateRemote, a member of class CoShellNameSpace.

8.1.5.2 CoShellNameSpace Methods

The methods of the CoShellNameSpace class are listed here.

Public Methods

	Name	Description
	Create (see page 419)	This is Create, a member of class CoShellNameSpace.
	CreateRemote (see page 419)	This is CreateRemote, a member of class CoShellNameSpace.

8.1.5.2.1 CoShellNameSpace.Create Method

Pascal

```
class function Create: IShellNameSpace;
```

C++

```
__fastcall IShellNameSpace Create();
```

Description

This is Create, a member of class CoShellNameSpace.

8.1.5.2.2 CoShellNameSpace.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellNameSpace;
```

C++

```
__fastcall IShellNameSpace CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellNameSpace.

8.1.6 CoShellUIHelper Class

Class Hierarchy

SHDocVw.CoShellUIHelper

Pascal

```
CoShellUIHelper = class;
```

C++

```
class CoShellUIHelper;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellUIHelper.

8.1.6.1 CoShellUIHelper Members

The following tables list the members exposed by CoShellUIHelper.

Public Methods

	Name	Description
	Create (see page 420)	This is Create, a member of class CoShellUIHelper.
	CreateRemote (see page 420)	This is CreateRemote, a member of class CoShellUIHelper.

8.1.6.2 CoShellUIHelper Methods

The methods of the CoShellUIHelper class are listed here.

Public Methods

	Name	Description
	Create (see page 420)	This is Create, a member of class CoShellUIHelper.
	CreateRemote (see page 420)	This is CreateRemote, a member of class CoShellUIHelper.

8.1.6.2.1 CoShellUIHelper.Create Method

Pascal

```
class function Create: IShellUIHelper;
```

C++

```
__fastcall IShellUIHelper Create();
```

Description

This is Create, a member of class CoShellUIHelper.

8.1.6.2.2 CoShellUIHelper.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellUIHelper;
```

C++

```
__fastcall IShellUIHelper CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellUIHelper.

8.1.7 CoShellWindows Class

Class Hierarchy

Pascal

```
CoShellWindows = class;
```

C++

```
class CoShellWindows;
```

File

SHDocVw

Description

This is class SHDocVw.CoShellWindows.

8.1.7.1 CoShellWindows Members

The following tables list the members exposed by CoShellWindows.

Public Methods

	Name	Description
	Create (see page 421)	This is Create, a member of class CoShellWindows.
	CreateRemote (see page 421)	This is CreateRemote, a member of class CoShellWindows.

8.1.7.2 CoShellWindows Methods

The methods of the CoShellWindows class are listed here.

Public Methods

	Name	Description
	Create (see page 421)	This is Create, a member of class CoShellWindows.
	CreateRemote (see page 421)	This is CreateRemote, a member of class CoShellWindows.

8.1.7.2.1 CoShellWindows.Create Method

Pascal

```
class function Create: IShellWindows;
```

C++

```
__fastcall IShellWindows Create();
```

Description

This is Create, a member of class CoShellWindows.

8.1.7.2.2 CoShellWindows.CreateRemote Method

Pascal

```
class function CreateRemote(const MachineName: string): IShellWindows;
```

C++

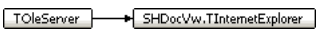
```
__fastcall IShellWindows CreateRemote(const UnicodeString MachineName);
```

Description

This is CreateRemote, a member of class CoShellWindows.

8.1.8 TInternetExplorer Class

Class Hierarchy



Pascal

```
TInternetExplorer = class(TOleServer);
```

C++

```
class TInternetExplorer : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TInternetExplorer.

8.1.8.1 TInternetExplorer Members

The following tables list the members exposed by TInternetExplorer.

Protected Methods

	Name	Description
	Get_AddressBar (see page 430)	This is Get_AddressBar, a member of class TInternetExplorer.
	Get_Application (see page 430)	This is Get_Application, a member of class TInternetExplorer.
	Get_Busy (see page 430)	This is Get_Busy, a member of class TInternetExplorer.
	Get_Container (see page 431)	This is Get_Container, a member of class TInternetExplorer.
	Get_Document (see page 431)	This is Get_Document, a member of class TInternetExplorer.
	Get_FullName (see page 431)	This is Get_FullName, a member of class TInternetExplorer.
	Get_FullScreen (see page 431)	This is Get_FullScreen, a member of class TInternetExplorer.
	Get_Height (see page 431)	This is Get_Height, a member of class TInternetExplorer.
	Get_HWnd (see page 432)	This is Get_HWnd, a member of class TInternetExplorer.
	Get_Left (see page 432)	This is Get_Left, a member of class TInternetExplorer.
	Get_LocationName (see page 432)	This is Get_LocationName, a member of class TInternetExplorer.
	Get_LocationURL (see page 432)	This is Get_LocationURL, a member of class TInternetExplorer.
	Get_MenuBar (see page 433)	This is Get_MenuBar, a member of class TInternetExplorer.
	Get_Name (see page 433)	This is Get_Name, a member of class TInternetExplorer.
	Get_Offline (see page 433)	This is Get_Offline, a member of class TInternetExplorer.
	Get_Parent (see page 433)	This is Get_Parent, a member of class TInternetExplorer.
	Get_Path (see page 433)	This is Get_Path, a member of class TInternetExplorer.
	Get_ReadyState (see page 434)	This is Get_ReadyState, a member of class TInternetExplorer.
	Get_RegisterAsBrowser (see page 434)	This is Get_RegisterAsBrowser, a member of class TInternetExplorer.
	Get_RegisterAsDropTarget (see page 434)	This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.
	Get_Resizable (see page 434)	This is Get_Resizable, a member of class TInternetExplorer.
	Get_Silent (see page 435)	This is Get_Silent, a member of class TInternetExplorer.
	Get_StatusBar (see page 435)	This is Get_StatusBar, a member of class TInternetExplorer.
	Get_StatusText (see page 435)	This is Get_StatusText, a member of class TInternetExplorer.
	Get_TheaterMode (see page 435)	This is Get_TheaterMode, a member of class TInternetExplorer.
	Get_ToolBar (see page 435)	This is Get_ToolBar, a member of class TInternetExplorer.
	Get_Top (see page 436)	This is Get_Top, a member of class TInternetExplorer.
	Get_TopLevelContainer (see page 436)	This is Get_TopLevelContainer, a member of class TInternetExplorer.
	Get_type_ (see page 436)	This is Get_type_, a member of class TInternetExplorer.
	Get_Visible (see page 436)	This is Get_Visible, a member of class TInternetExplorer.
	Get_Width (see page 437)	This is Get_Width, a member of class TInternetExplorer.
	InitServerData (see page 438)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 438)	Dispatches an event from the COM server to the appropriate event handler.
	Set_AddressBar (see page 443)	This is Set_AddressBar, a member of class TInternetExplorer.
	Set_FullScreen (see page 444)	This is Set_FullScreen, a member of class TInternetExplorer.
	Set_Height (see page 444)	This is Set_Height, a member of class TInternetExplorer.
	Set_Left (see page 444)	This is Set_Left, a member of class TInternetExplorer.

	Set_MenuBar (see page 444)	This is Set_MenuBar, a member of class TInternetExplorer.
	Set_Offline (see page 444)	This is Set_Offline, a member of class TInternetExplorer.
	Set_RegisterAsBrowser (see page 445)	This is Set_RegisterAsBrowser, a member of class TInternetExplorer.
	Set_RegisterAsDropTarget (see page 445)	This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.
	Set_Resizable (see page 445)	This is Set_Resizable, a member of class TInternetExplorer.
	Set_Silent (see page 445)	This is Set_Silent, a member of class TInternetExplorer.
	Set_StatusBar (see page 446)	This is Set_StatusBar, a member of class TInternetExplorer.
	Set_StatusText (see page 446)	This is Set_StatusText, a member of class TInternetExplorer.
	Set_TheaterMode (see page 446)	This is Set_TheaterMode, a member of class TInternetExplorer.
	Set_ToolBar (see page 446)	This is Set_ToolBar, a member of class TInternetExplorer.
	Set_Top (see page 446)	This is Set_Top, a member of class TInternetExplorer.
	Set_Visible (see page 447)	This is Set_Visible, a member of class TInternetExplorer.
	Set_Width (see page 447)	This is Set_Width, a member of class TInternetExplorer.

Public Methods

	Name	Description
	ClientToWindow (see page 428)	This is ClientToWindow, a member of class TInternetExplorer.
V	Connect (see page 428)	Establishes a connection to the COM server.
	ConnectTo (see page 428)	This is ConnectTo, a member of class TInternetExplorer.
V	Create (see page 425)	Allocates memory and constructs a safely initialized instance of a component.
V	Destroy (see page 425)	Disposes of the component and its owned components.
V	Disconnect (see page 429)	Terminates the connection to the COM server.
	ExecWB (see page 429)	This is the overview for the ExecWB method overload.
	GetProperty (see page 437)	This is GetProperty, a member of class TInternetExplorer.
	GoBack (see page 437)	This is GoBack, a member of class TInternetExplorer.
	GoForward (see page 437)	This is GoForward, a member of class TInternetExplorer.
	GoHome (see page 437)	This is GoHome, a member of class TInternetExplorer.
	GoSearch (see page 438)	This is GoSearch, a member of class TInternetExplorer.
	Navigate (see page 439)	This is the overview for the Navigate method overload.
	Navigate2 (see page 440)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 442)	This is PutProperty, a member of class TInternetExplorer.
	QueryStatusWB (see page 442)	This is QueryStatusWB, a member of class TInternetExplorer.
	Quit (see page 442)	This is Quit, a member of class TInternetExplorer.
	Refresh (see page 442)	This is Refresh, a member of class TInternetExplorer.
	Refresh2 (see page 443)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 447)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 448)	This is Stop, a member of class TInternetExplorer.

Public Properties

	Name	Description
	AddressBar (see page 449)	This is AddressBar, a member of class TInternetExplorer.
R	Application (see page 449)	This is Application, a member of class TInternetExplorer.
R	Busy (see page 449)	This is Busy, a member of class TInternetExplorer.
R	Container (see page 450)	This is Container, a member of class TInternetExplorer.
R	DefaultInterface (see page 450)	This is DefaultInterface, a member of class TInternetExplorer.
R	Document (see page 450)	This is Document, a member of class TInternetExplorer.
R	FullName (see page 450)	This is FullName, a member of class TInternetExplorer.
	FullScreen (see page 450)	This is FullScreen, a member of class TInternetExplorer.
	Height (see page 451)	This is Height, a member of class TInternetExplorer.
R	HWnd (see page 451)	This is HWnd, a member of class TInternetExplorer.
	Left (see page 451)	This is Left, a member of class TInternetExplorer.
R	LocationName (see page 451)	This is LocationName, a member of class TInternetExplorer.
R	LocationURL (see page 452)	This is LocationURL, a member of class TInternetExplorer.
	MenuBar (see page 452)	This is MenuBar, a member of class TInternetExplorer.
R	Name (see page 452)	Specifies the name of the component as referenced in code.

	Offline (see page 454)	This is Offline, a member of class TInternetExplorer.
R	Parent (see page 454)	This is Parent, a member of class TInternetExplorer.
R	Path (see page 455)	This is Path, a member of class TInternetExplorer.
R	ReadyState (see page 455)	This is ReadyState, a member of class TInternetExplorer.
	RegisterAsBrowser (see page 455)	This is RegisterAsBrowser, a member of class TInternetExplorer.
	RegisterAsDropTarget (see page 455)	This is RegisterAsDropTarget, a member of class TInternetExplorer.
	Resizable (see page 455)	This is Resizable, a member of class TInternetExplorer.
	Silent (see page 456)	This is Silent, a member of class TInternetExplorer.
	StatusBar (see page 456)	This is StatusBar, a member of class TInternetExplorer.
	StatusText (see page 456)	This is StatusText, a member of class TInternetExplorer.
	TheaterMode (see page 456)	This is TheaterMode, a member of class TInternetExplorer.
	ToolBar (see page 457)	This is ToolBar, a member of class TInternetExplorer.
	Top (see page 457)	This is Top, a member of class TInternetExplorer.
R	TopLevelContainer (see page 457)	This is TopLevelContainer, a member of class TInternetExplorer.
R	type_ (see page 457)	This is type_, a member of class TInternetExplorer.
	Visible (see page 457)	This is Visible, a member of class TInternetExplorer.
	Width (see page 458)	This is Width, a member of class TInternetExplorer.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 459)	This is OnBeforeNavigate2, a member of class TInternetExplorer.
	OnClientToHostWindow (see page 459)	This is OnClientToHostWindow, a member of class TInternetExplorer.
	OnCommandStateChange (see page 459)	This is OnCommandStateChange, a member of class TInternetExplorer.
	OnDocumentComplete (see page 459)	This is OnDocumentComplete, a member of class TInternetExplorer.
	OnDownloadBegin (see page 459)	This is OnDownloadBegin, a member of class TInternetExplorer.
	OnDownloadComplete (see page 460)	This is OnDownloadComplete, a member of class TInternetExplorer.
	OnFileDownload (see page 460)	This is OnFileDownload, a member of class TInternetExplorer.
	OnFullScreen (see page 460)	This is OnFullScreen, a member of class TInternetExplorer.
	OnMenuBar (see page 460)	This is OnMenuBar, a member of class TInternetExplorer.
	OnNavigateComplete2 (see page 461)	This is OnNavigateComplete2, a member of class TInternetExplorer.
	OnNavigateError (see page 461)	This is OnNavigateError, a member of class TInternetExplorer.
	OnNewWindow2 (see page 461)	This is OnNewWindow2, a member of class TInternetExplorer.
	OnPrintTemplateInstantiation (see page 461)	This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.
	OnPrintTemplateTeardown (see page 461)	This is OnPrintTemplateTeardown, a member of class TInternetExplorer.
	OnPrivacyImpactedStateChange (see page 462)	This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.
	OnProgressChange (see page 462)	This is OnProgressChange, a member of class TInternetExplorer.
	OnPropertyChange (see page 462)	This is OnPropertyChange, a member of class TInternetExplorer.
	OnQuit (see page 462)	This is OnQuit, a member of class TInternetExplorer.
	OnSetSecureLockIcon (see page 463)	This is OnSetSecureLockIcon, a member of class TInternetExplorer.
	OnStatusBar (see page 463)	This is OnStatusBar, a member of class TInternetExplorer.
	OnStatusTextChange (see page 463)	This is OnStatusTextChange, a member of class TInternetExplorer.
	OnTheaterMode (see page 463)	This is OnTheaterMode, a member of class TInternetExplorer.
	OnTitleChange (see page 463)	This is OnTitleChange, a member of class TInternetExplorer.
	OnToolBar (see page 464)	This is OnToolBar, a member of class TInternetExplorer.
	OnUpdatePageStatus (see page 464)	This is OnUpdatePageStatus, a member of class TInternetExplorer.
	OnVisible (see page 464)	This is OnVisible, a member of class TInternetExplorer.
	OnWindowClosing (see page 464)	This is OnWindowClosing, a member of class TInternetExplorer.
	OnWindowSetHeight (see page 465)	This is OnWindowSetHeight, a member of class TInternetExplorer.
	OnWindowSetLeft (see page 465)	This is OnWindowSetLeft, a member of class TInternetExplorer.
	OnWindowSetResizable (see page 465)	This is OnWindowSetResizable, a member of class TInternetExplorer.
	OnWindowSetTop (see page 465)	This is OnWindowSetTop, a member of class TInternetExplorer.
	OnWindowSetWidth (see page 465)	This is OnWindowSetWidth, a member of class TInternetExplorer.

8.1.8.2 TInternetExplorer.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TInternetExplorer(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3355)

Destroy

System::TObject::Free (see page 3372)

Owner

8.1.8.3 TInternetExplorer.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TInternetExplorer();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3356)

DestroyComponents

Destroying

System::TObject::Free (see page 3372)

Notification

Owner

Release

Components

8.1.8.4 TInternetExplorer Methods

The methods of the TInternetExplorer class are listed here.

Protected Methods

	Name	Description
	Get_AddressBar (see page 430)	This is Get_AddressBar, a member of class TInternetExplorer.
	Get_Application (see page 430)	This is Get_Application, a member of class TInternetExplorer.
	Get_Busy (see page 430)	This is Get_Busy, a member of class TInternetExplorer.
	Get_Container (see page 431)	This is Get_Container, a member of class TInternetExplorer.
	Get_Document (see page 431)	This is Get_Document, a member of class TInternetExplorer.
	Get_FullName (see page 431)	This is Get_FullName, a member of class TInternetExplorer.
	Get_FullScreen (see page 431)	This is Get_FullScreen, a member of class TInternetExplorer.
	Get_Height (see page 431)	This is Get_Height, a member of class TInternetExplorer.
	Get_HWnd (see page 432)	This is Get_HWnd, a member of class TInternetExplorer.
	Get_Left (see page 432)	This is Get_Left, a member of class TInternetExplorer.
	Get_LocationName (see page 432)	This is Get_LocationName, a member of class TInternetExplorer.
	Get_LocationURL (see page 432)	This is Get_LocationURL, a member of class TInternetExplorer.
	Get_MenuBar (see page 433)	This is Get_MenuBar, a member of class TInternetExplorer.
	Get_Name (see page 433)	This is Get_Name, a member of class TInternetExplorer.
	Get_Offline (see page 433)	This is Get_Offline, a member of class TInternetExplorer.
	Get_Parent (see page 433)	This is Get_Parent, a member of class TInternetExplorer.
	Get_Path (see page 433)	This is Get_Path, a member of class TInternetExplorer.
	Get_ReadyState (see page 434)	This is Get_ReadyState, a member of class TInternetExplorer.

	Get_RegisterAsBrowser (see page 434)	This is Get_RegisterAsBrowser, a member of class TInternetExplorer.
	Get_RegisterAsDropTarget (see page 434)	This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.
	Get_Resizable (see page 434)	This is Get_Resizable, a member of class TInternetExplorer.
	Get_Silent (see page 435)	This is Get_Silent, a member of class TInternetExplorer.
	Get_StatusBar (see page 435)	This is Get_StatusBar, a member of class TInternetExplorer.
	Get_StatusText (see page 435)	This is Get_StatusText, a member of class TInternetExplorer.
	Get_TheaterMode (see page 435)	This is Get_TheaterMode, a member of class TInternetExplorer.
	Get_ToolBar (see page 435)	This is Get_ToolBar, a member of class TInternetExplorer.
	Get_Top (see page 436)	This is Get_Top, a member of class TInternetExplorer.
	Get_TopLevelContainer (see page 436)	This is Get_TopLevelContainer, a member of class TInternetExplorer.
	Get_type_ (see page 436)	This is Get_type_, a member of class TInternetExplorer.
	Get_Visible (see page 436)	This is Get_Visible, a member of class TInternetExplorer.
	Get_Width (see page 437)	This is Get_Width, a member of class TInternetExplorer.
	InitServerData (see page 438)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 438)	Dispatches an event from the COM server to the appropriate event handler.
	Set_AddressBar (see page 443)	This is Set_AddressBar, a member of class TInternetExplorer.
	Set_FullScreen (see page 444)	This is Set_FullScreen, a member of class TInternetExplorer.
	Set_Height (see page 444)	This is Set_Height, a member of class TInternetExplorer.
	Set_Left (see page 444)	This is Set_Left, a member of class TInternetExplorer.
	Set_MenuBar (see page 444)	This is Set_MenuBar, a member of class TInternetExplorer.
	Set_Offline (see page 444)	This is Set_Offline, a member of class TInternetExplorer.
	Set_RegisterAsBrowser (see page 445)	This is Set_RegisterAsBrowser, a member of class TInternetExplorer.
	Set_RegisterAsDropTarget (see page 445)	This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.
	Set_Resizable (see page 445)	This is Set_Resizable, a member of class TInternetExplorer.
	Set_Silent (see page 445)	This is Set_Silent, a member of class TInternetExplorer.
	Set_StatusBar (see page 446)	This is Set_StatusBar, a member of class TInternetExplorer.
	Set_StatusText (see page 446)	This is Set_StatusText, a member of class TInternetExplorer.
	Set_TheaterMode (see page 446)	This is Set_TheaterMode, a member of class TInternetExplorer.
	Set_ToolBar (see page 446)	This is Set_ToolBar, a member of class TInternetExplorer.
	Set_Top (see page 446)	This is Set_Top, a member of class TInternetExplorer.
	Set_Visible (see page 447)	This is Set_Visible, a member of class TInternetExplorer.
	Set_Width (see page 447)	This is Set_Width, a member of class TInternetExplorer.

Public Methods

	Name	Description
	ClientToWindow (see page 428)	This is ClientToWindow, a member of class TInternetExplorer.
	Connect (see page 428)	Establishes a connection to the COM server.
	ConnectTo (see page 428)	This is ConnectTo, a member of class TInternetExplorer.
	Disconnect (see page 429)	Terminates the connection to the COM server.
	ExecWB (see page 429)	This is the overview for the ExecWB method overload.
	GetProperty (see page 437)	This is GetProperty, a member of class TInternetExplorer.
	GoBack (see page 437)	This is GoBack, a member of class TInternetExplorer.
	GoForward (see page 437)	This is GoForward, a member of class TInternetExplorer.
	GoHome (see page 437)	This is GoHome, a member of class TInternetExplorer.
	GoSearch (see page 438)	This is GoSearch, a member of class TInternetExplorer.
	Navigate (see page 439)	This is the overview for the Navigate method overload.
	Navigate2 (see page 440)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 442)	This is PutProperty, a member of class TInternetExplorer.
	QueryStatusWB (see page 442)	This is QueryStatusWB, a member of class TInternetExplorer.
	Quit (see page 442)	This is Quit, a member of class TInternetExplorer.
	Refresh (see page 442)	This is Refresh, a member of class TInternetExplorer.
	Refresh2 (see page 443)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 447)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 448)	This is Stop, a member of class TInternetExplorer.

8.1.8.4.1 TInternetExplorer.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class TInternetExplorer.

8.1.8.4.2 TInternetExplorer.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

8.1.8.4.3 TInternetExplorer.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IWebBrowser2);
```

C++

```
__fastcall ConnectTo(IWebBrowser2 svrIntf);
```

Description

This is ConnectTo, a member of class TInternetExplorer.

8.1.8.4.4 TInternetExplorer.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

8.1.8.4.5 ExecWB Method

This is the overview for the ExecWB method overload.

Overload List

	Name	Description
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT) (see page 429)	This is ExecWB, a member of class TInternetExplorer.
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant) (see page 429)	This is ExecWB, a member of class TInternetExplorer.
	TInternetExplorer.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant) (see page 430)	This is ExecWB, a member of class TInternetExplorer.

8.1.8.4.5.1 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt);
```

Description

This is ExecWB, a member of class TInternetExplorer.

8.1.8.4.5.2 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT, OleVariant)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn);
```

Description

This is ExecWB, a member of class TInternetExplorer.

8.1.8.4.5.3 TInternetExplorer.ExecWB Method (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant)

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class TInternetExplorer.

8.1.8.4.6 TInternetExplorer.Get_AddressBar Method

Pascal

```
function Get_AddressBar: WordBool;
```

C++

```
__fastcall WordBool Get_AddressBar();
```

Description

This is Get_AddressBar, a member of class TInternetExplorer.

8.1.8.4.7 TInternetExplorer.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TInternetExplorer.

8.1.8.4.8 TInternetExplorer.Get_Busy Method

Pascal

```
function Get_Busy: WordBool;
```

C++

```
__fastcall WordBool Get_Busy();
```

Description

This is Get_Busy, a member of class TInternetExplorer.

8.1.8.4.9 TInternetExplorer.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TInternetExplorer.

8.1.8.4.10 TInternetExplorer.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TInternetExplorer.

8.1.8.4.11 TInternetExplorer.Get_FullName Method

Pascal

```
function Get_FullName: WideString;
```

C++

```
__fastcall BSTR Get_FullName();
```

Description

This is Get_FullName, a member of class TInternetExplorer.

8.1.8.4.12 TInternetExplorer.Get_FullScreen Method

Pascal

```
function Get_FullScreen: WordBool;
```

C++

```
__fastcall WordBool Get_FullScreen();
```

Description

This is Get_FullScreen, a member of class TInternetExplorer.

8.1.8.4.13 TInternetExplorer.Get_Height Method

Pascal

```
function Get_Height: Integer;
```

C++

```
__fastcall int Get_Height();
```

Description

This is Get_Height, a member of class TInternetExplorer.

8.1.8.4.14 TInternetExplorer.Get_HWnd Method

Pascal

```
function Get_HWnd: Integer;
```

C++

```
__fastcall int Get_HWnd();
```

Description

This is Get_HWnd, a member of class TInternetExplorer.

8.1.8.4.15 TInternetExplorer.Get_Left Method

Pascal

```
function Get_Left: Integer;
```

C++

```
__fastcall int Get_Left();
```

Description

This is Get_Left, a member of class TInternetExplorer.

8.1.8.4.16 TInternetExplorer.Get_LocationName Method

Pascal

```
function Get_LocationName: WideString;
```

C++

```
__fastcall BSTR Get_LocationName();
```

Description

This is Get_LocationName, a member of class TInternetExplorer.

8.1.8.4.17 TInternetExplorer.Get_LocationURL Method

Pascal

```
function Get_LocationURL: WideString;
```

C++

```
__fastcall BSTR Get_LocationURL();
```

Description

This is Get_LocationURL, a member of class TInternetExplorer.

8.1.8.4.18 TInternetExplorer.Get_MenuBar Method

Pascal

```
function Get_MenuBar: WordBool;
```

C++

```
__fastcall WordBool Get_MenuBar();
```

Description

This is Get_MenuBar, a member of class TInternetExplorer.

8.1.8.4.19 TInternetExplorer.Get_Name Method

Pascal

```
function Get_Name: WideString;
```

C++

```
__fastcall BSTR Get_Name();
```

Description

This is Get_Name, a member of class TInternetExplorer.

8.1.8.4.20 TInternetExplorer.Get_Offline Method

Pascal

```
function Get_Offline: WordBool;
```

C++

```
__fastcall WordBool Get_Offline();
```

Description

This is Get_Offline, a member of class TInternetExplorer.

8.1.8.4.21 TInternetExplorer.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TInternetExplorer.

8.1.8.4.22 TInternetExplorer.Get_Path Method

Pascal

```
function Get_Path: WideString;
```

C++

```
__fastcall BSTR Get_Path();
```

Description

This is Get_Path, a member of class TInternetExplorer.

8.1.8.4.23 TInternetExplorer.Get_ReadyState Method

Pascal

```
function Get_ReadyState: tagREADYSTATE;
```

C++

```
__fastcall tagREADYSTATE Get_ReadyState();
```

Description

This is Get_ReadyState, a member of class TInternetExplorer.

8.1.8.4.24 TInternetExplorer.Get_RegisterAsBrowser Method

Pascal

```
function Get_RegisterAsBrowser: WordBool;
```

C++

```
__fastcall WordBool Get_RegisterAsBrowser();
```

Description

This is Get_RegisterAsBrowser, a member of class TInternetExplorer.

8.1.8.4.25 TInternetExplorer.Get_RegisterAsDropTarget Method

Pascal

```
function Get_RegisterAsDropTarget: WordBool;
```

C++

```
__fastcall WordBool Get_RegisterAsDropTarget();
```

Description

This is Get_RegisterAsDropTarget, a member of class TInternetExplorer.

8.1.8.4.26 TInternetExplorer.Get_Resizable Method

Pascal

```
function Get_Resizable: WordBool;
```

C++

```
__fastcall WordBool Get_Resizable();
```

Description

This is Get_Resizable, a member of class TInternetExplorer.

8.1.8.4.27 TInternetExplorer.Get_Silent Method

Pascal

```
function Get_Silent: WordBool;
```

C++

```
__fastcall WordBool Get_Silent();
```

Description

This is Get_Silent, a member of class TInternetExplorer.

8.1.8.4.28 TInternetExplorer.Get_StatusBar Method

Pascal

```
function Get_StatusBar: WordBool;
```

C++

```
__fastcall WordBool Get_StatusBar();
```

Description

This is Get_StatusBar, a member of class TInternetExplorer.

8.1.8.4.29 TInternetExplorer.Get_StatusText Method

Pascal

```
function Get_StatusText: WideString;
```

C++

```
__fastcall BSTR Get_StatusText();
```

Description

This is Get_StatusText, a member of class TInternetExplorer.

8.1.8.4.30 TInternetExplorer.Get_TheaterMode Method

Pascal

```
function Get_TheaterMode: WordBool;
```

C++

```
__fastcall WordBool Get_TheaterMode();
```

Description

This is Get_TheaterMode, a member of class TInternetExplorer.

8.1.8.4.31 TInternetExplorer.Get_ToolBar Method

Pascal

```
function Get_ToolBar: SYSINT;
```

C++

```
__fastcall SYSINT Get_ToolBar();
```

Description

This is Get_ToolBar, a member of class TInternetExplorer.

8.1.8.4.32 TInternetExplorer.Get_Top Method

Pascal

```
function Get_Top: Integer;
```

C++

```
__fastcall int Get_Top();
```

Description

This is Get_Top, a member of class TInternetExplorer.

8.1.8.4.33 TInternetExplorer.Get_TopLevelContainer Method

Pascal

```
function Get_TopLevelContainer: WordBool;
```

C++

```
__fastcall WordBool Get_TopLevelContainer();
```

Description

This is Get_TopLevelContainer, a member of class TInternetExplorer.

8.1.8.4.34 TInternetExplorer.Get_type_ Method

Pascal

```
function Get_type_: WideString;
```

C++

```
__fastcall BSTR Get_type_();
```

Description

This is Get_type_, a member of class TInternetExplorer.

8.1.8.4.35 TInternetExplorer.Get_Visible Method

Pascal

```
function Get_Visible: WordBool;
```

C++

```
__fastcall WordBool Get_Visible();
```

Description

This is Get_Visible, a member of class TInternetExplorer.

8.1.8.4.36 TInternetExplorer.Get_Width Method

Pascal

```
function Get_Width: Integer;
```

C++

```
__fastcall int Get_Width();
```

Description

This is Get_Width, a member of class TInternetExplorer.

8.1.8.4.37 TInternetExplorer.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class TInternetExplorer.

8.1.8.4.38 TInternetExplorer.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class TInternetExplorer.

8.1.8.4.39 TInternetExplorer.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class TInternetExplorer.

8.1.8.4.40 TInternetExplorer.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class TInternetExplorer.

8.1.8.4.41 TInternetExplorer.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class TInternetExplorer.

8.1.8.4.42 TInternetExplorer.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

8.1.8.4.43 TInternetExplorer.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the

appropriate event handlers.

See Also

EventDispatch

8.1.8.4.44 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TInternetExplorer.Navigate (WideString) (see page 439)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant) (see page 439)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant) (see page 439)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 440)	This is Navigate, a member of class TInternetExplorer.
	TInternetExplorer.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 440)	This is Navigate, a member of class TInternetExplorer.

8.1.8.4.44.1 TInternetExplorer.Navigate Method (WideString)

Pascal

```
procedure Navigate(const URL: WideString); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
```

Description

This is Navigate, a member of class TInternetExplorer.

8.1.8.4.44.2 TInternetExplorer.Navigate Method (WideString, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
```

Description

This is Navigate, a member of class TInternetExplorer.

8.1.8.4.44.3 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
```

Description

This is Navigate, a member of class TInternetExplorer.

8.1.8.4.44.4 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
```

Description

This is Navigate, a member of class TInternetExplorer.

8.1.8.4.44.5 TInternetExplorer.Navigate Method (WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class TInternetExplorer.

8.1.8.4.45 Navigate2 Method

This is the overview for the Navigate2 method overload.

Overload List

	Name	Description
	TInternetExplorer.Navigate2 (OleVariant) (see page 441)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant) (see page 441)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant) (see page 441)	This is Navigate2, a member of class TInternetExplorer.
	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant) (see page 441)	This is Navigate2, a member of class TInternetExplorer.

	TInternetExplorer.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant) (see page 442)	This is Navigate2, a member of class TInternetExplorer.

8.1.8.4.45.1 TInternetExplorer.Navigate2 Method (OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL);
```

Description

This is Navigate2, a member of class TInternetExplorer.

8.1.8.4.45.2 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags);
```

Description

This is Navigate2, a member of class TInternetExplorer.

8.1.8.4.45.3 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName);
```

Description

This is Navigate2, a member of class TInternetExplorer.

8.1.8.4.45.4 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData);
```

Description

This is Navigate2, a member of class TInternetExplorer.

8.1.8.4.45.5 TInternetExplorer.Navigate2 Method (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant; var Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData, OleVariant Headers);
```

Description

This is Navigate2, a member of class TInternetExplorer.

8.1.8.4.46 TInternetExplorer.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class TInternetExplorer.

8.1.8.4.47 TInternetExplorer.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class TInternetExplorer.

8.1.8.4.48 TInternetExplorer.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class TInternetExplorer.

8.1.8.4.49 TInternetExplorer.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class TInternetExplorer.

8.1.8.4.50 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TInternetExplorer.Refresh2 () (see page 443)	This is Refresh2, a member of class TInternetExplorer.
	TInternetExplorer.Refresh2 (OleVariant) (see page 443)	This is Refresh2, a member of class TInternetExplorer.

8.1.8.4.50.1 TInternetExplorer.Refresh2 Method ()

Pascal

```
procedure Refresh2; overload;
```

C++

```
__fastcall Refresh2();
```

Description

This is Refresh2, a member of class TInternetExplorer.

8.1.8.4.50.2 TInternetExplorer.Refresh2 Method (OleVariant)

Pascal

```
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class TInternetExplorer.

8.1.8.4.51 TInternetExplorer.Set_AddressBar Method

Pascal

```
procedure Set_AddressBar(Value: WordBool);
```

C++

```
__fastcall Set_AddressBar(WordBool Value);
```

Description

This is Set_AddressBar, a member of class TInternetExplorer.

8.1.8.4.52 TInternetExplorer.Set_FullScreen Method

Pascal

```
procedure Set_FullScreen(pbFullScreen: WordBool);
```

C++

```
__fastcall Set_FullScreen(WordBool pbFullScreen);
```

Description

This is Set_FullScreen, a member of class TInternetExplorer.

8.1.8.4.53 TInternetExplorer.Set_Height Method

Pascal

```
procedure Set_Height(pl: Integer);
```

C++

```
__fastcall Set_Height(int pl);
```

Description

This is Set_Height, a member of class TInternetExplorer.

8.1.8.4.54 TInternetExplorer.Set_Left Method

Pascal

```
procedure Set_Left(pl: Integer);
```

C++

```
__fastcall Set_Left(int pl);
```

Description

This is Set_Left, a member of class TInternetExplorer.

8.1.8.4.55 TInternetExplorer.Set_MenuBar Method

Pascal

```
procedure Set_MenuBar(Value: WordBool);
```

C++

```
__fastcall Set_MenuBar(WordBool Value);
```

Description

This is Set_MenuBar, a member of class TInternetExplorer.

8.1.8.4.56 TInternetExplorer.Set_Offline Method

Pascal

```
procedure Set_Offline(pbOffline: WordBool);
```

C++

```
__fastcall Set_Offline(WordBool pbOffline);
```

Description

This is Set_Offline, a member of class TInternetExplorer.

8.1.8.4.57 TInternetExplorer.Set_RegisterAsBrowser Method

Pascal

```
procedure Set_RegisterAsBrowser(pbRegister: WordBool);
```

C++

```
__fastcall Set_RegisterAsBrowser(WordBool pbRegister);
```

Description

This is Set_RegisterAsBrowser, a member of class TInternetExplorer.

8.1.8.4.58 TInternetExplorer.Set_RegisterAsDropTarget Method

Pascal

```
procedure Set_RegisterAsDropTarget(pbRegister: WordBool);
```

C++

```
__fastcall Set_RegisterAsDropTarget(WordBool pbRegister);
```

Description

This is Set_RegisterAsDropTarget, a member of class TInternetExplorer.

8.1.8.4.59 TInternetExplorer.Set_Resizable Method

Pascal

```
procedure Set_Resizable(Value: WordBool);
```

C++

```
__fastcall Set_Resizable(WordBool Value);
```

Description

This is Set_Resizable, a member of class TInternetExplorer.

8.1.8.4.60 TInternetExplorer.Set_Silent Method

Pascal

```
procedure Set_Silent(pbSilent: WordBool);
```

C++

```
__fastcall Set_Silent(WordBool pbSilent);
```

Description

This is Set_Silent, a member of class TInternetExplorer.

8.1.8.4.61 TInternetExplorer.Set_StatusBar Method

Pascal

```
procedure Set_StatusBar(pBool: WordBool);
```

C++

```
__fastcall Set_StatusBar(WordBool pBool);
```

Description

This is Set_StatusBar, a member of class TInternetExplorer.

8.1.8.4.62 TInternetExplorer.Set_StatusText Method

Pascal

```
procedure Set_StatusText(const StatusText: WideString);
```

C++

```
__fastcall Set_StatusText(const BSTR StatusText);
```

Description

This is Set_StatusText, a member of class TInternetExplorer.

8.1.8.4.63 TInternetExplorer.Set_TheaterMode Method

Pascal

```
procedure Set_TheaterMode(pbRegister: WordBool);
```

C++

```
__fastcall Set_TheaterMode(WordBool pbRegister);
```

Description

This is Set_TheaterMode, a member of class TInternetExplorer.

8.1.8.4.64 TInternetExplorer.Set_ToolBar Method

Pascal

```
procedure Set_ToolBar(Value: SYSINT);
```

C++

```
__fastcall Set_ToolBar(SYSINT Value);
```

Description

This is Set_ToolBar, a member of class TInternetExplorer.

8.1.8.4.65 TInternetExplorer.Set_Top Method

Pascal

```
procedure Set_Top(pl: Integer);
```

C++

```
__fastcall Set_Top(int pl);
```

Description

This is Set_Top, a member of class TInternetExplorer.

8.1.8.4.66 TInternetExplorer.Set_Visible Method

Pascal

```
procedure Set_Visible(pBool: WordBool);
```

C++

```
__fastcall Set_Visible(WordBool pBool);
```

Description

This is Set_Visible, a member of class TInternetExplorer.

8.1.8.4.67 TInternetExplorer.Set_Width Method

Pascal

```
procedure Set_Width(pl: Integer);
```

C++

```
__fastcall Set_Width(int pl);
```

Description

This is Set_Width, a member of class TInternetExplorer.

8.1.8.4.68 ShowBrowserBar Method

This is the overview for the ShowBrowserBar method overload.

Overload List

	Name	Description
	TInternetExplorer.ShowBrowserBar (OleVariant) (see page 447)	This is ShowBrowserBar, a member of class TInternetExplorer.
	TInternetExplorer.ShowBrowserBar (OleVariant, OleVariant) (see page 448)	This is ShowBrowserBar, a member of class TInternetExplorer.
	TInternetExplorer.ShowBrowserBar (OleVariant, OleVariant, OleVariant) (see page 448)	This is ShowBrowserBar, a member of class TInternetExplorer.

8.1.8.4.68.1 TInternetExplorer.ShowBrowserBar Method (OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

8.1.8.4.68.2 TInternetExplorer.ShowBrowserBar Method (OleVariant, OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

8.1.8.4.68.3 TInternetExplorer.ShowBrowserBar Method (OleVariant, OleVariant, OleVariant)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class TInternetExplorer.

8.1.8.4.69 TInternetExplorer.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class TInternetExplorer.

8.1.8.5 TInternetExplorer Properties

The properties of the TInternetExplorer class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 449)	This is AddressBar, a member of class TInternetExplorer.
R	Application (see page 449)	This is Application, a member of class TInternetExplorer.
R	Busy (see page 449)	This is Busy, a member of class TInternetExplorer.
R	Container (see page 450)	This is Container, a member of class TInternetExplorer.
R	DefaultInterface (see page 450)	This is DefaultInterface, a member of class TInternetExplorer.
R	Document (see page 450)	This is Document, a member of class TInternetExplorer.
R	FullName (see page 450)	This is FullName, a member of class TInternetExplorer.
	FullScreen (see page 450)	This is FullScreen, a member of class TInternetExplorer.
	Height (see page 451)	This is Height, a member of class TInternetExplorer.
R	HWnd (see page 451)	This is HWnd, a member of class TInternetExplorer.
	Left (see page 451)	This is Left, a member of class TInternetExplorer.
R	LocationName (see page 451)	This is LocationName, a member of class TInternetExplorer.

R	LocationURL (see page 452)	This is LocationURL, a member of class TInternetExplorer.
	MenuBar (see page 452)	This is MenuBar, a member of class TInternetExplorer.
R	Name (see page 452)	Specifies the name of the component as referenced in code.
	Offline (see page 454)	This is Offline, a member of class TInternetExplorer.
R	Parent (see page 454)	This is Parent, a member of class TInternetExplorer.
R	Path (see page 455)	This is Path, a member of class TInternetExplorer.
R	ReadyState (see page 455)	This is ReadyState, a member of class TInternetExplorer.
	RegisterAsBrowser (see page 455)	This is RegisterAsBrowser, a member of class TInternetExplorer.
	RegisterAsDropTarget (see page 455)	This is RegisterAsDropTarget, a member of class TInternetExplorer.
	Resizable (see page 455)	This is Resizable, a member of class TInternetExplorer.
	Silent (see page 456)	This is Silent, a member of class TInternetExplorer.
	StatusBar (see page 456)	This is StatusBar, a member of class TInternetExplorer.
	StatusText (see page 456)	This is StatusText, a member of class TInternetExplorer.
	TheaterMode (see page 456)	This is TheaterMode, a member of class TInternetExplorer.
	ToolBar (see page 457)	This is ToolBar, a member of class TInternetExplorer.
	Top (see page 457)	This is Top, a member of class TInternetExplorer.
R	TopLevelContainer (see page 457)	This is TopLevelContainer, a member of class TInternetExplorer.
R	type_ (see page 457)	This is type_, a member of class TInternetExplorer.
	Visible (see page 457)	This is Visible, a member of class TInternetExplorer.
	Width (see page 458)	This is Width, a member of class TInternetExplorer.

8.1.8.5.1 TInternetExplorer.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class TInternetExplorer.

8.1.8.5.2 TInternetExplorer.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class TInternetExplorer.

8.1.8.5.3 TInternetExplorer.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class TInternetExplorer.

8.1.8.5.4 TInternetExplorer.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class TInternetExplorer.

8.1.8.5.5 TInternetExplorer.DefaultInterface Property

Pascal

```
property DefaultInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 DefaultInterface;
```

Description

This is DefaultInterface, a member of class TInternetExplorer.

8.1.8.5.6 TInternetExplorer.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class TInternetExplorer.

8.1.8.5.7 TInternetExplorer.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class TInternetExplorer.

8.1.8.5.8 TInternetExplorer.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class TInternetExplorer.

8.1.8.5.9 TInternetExplorer.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class TInternetExplorer.

8.1.8.5.10 TInternetExplorer.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class TInternetExplorer.

8.1.8.5.11 TInternetExplorer.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class TInternetExplorer.

8.1.8.5.12 TInternetExplorer.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class TInternetExplorer.

8.1.8.5.13 TInternetExplorer.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class TInternetExplorer.

8.1.8.5.14 TInternetExplorer.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class TInternetExplorer.

8.1.8.5.15 TInternetExplorer.Name Property

Specifies the name of the component as referenced in code.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Use Name to change the name of a component to reflect its purpose in the current application. By default, the IDE assigns sequential names based on the type of the component, such as 'Button1', 'Button2', and so on.

Use Name to refer to the component in code.

Warning: Changing Name at runtime causes any references to the old name to become undefined. Any subsequent code that uses the old name will cause an exception.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
```

```
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  dynamic_cast<TObject *>(Components[i]));
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
        break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
```

```
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
  if (ListBox1.Items.Objects[Index] is TControl) then
  begin
    TheObject :=
      (ListBox1.Items.Objects[Index] as TControl);
    StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                             IntToStr(TheObject.Left) + ', ' +
                             IntToStr(TheObject.Top) + ') ';
  end
  else
    Beep;
  end;
end;
```

See Also

ChangeName

SetName

ValidateRename

Defining New Persistent Fields

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

8.1.8.5.16 TInternetExplorer.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class TInternetExplorer.

8.1.8.5.17 TInternetExplorer.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class TInternetExplorer.

8.1.8.5.18 TInternetExplorer.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class TInternetExplorer.

8.1.8.5.19 TInternetExplorer.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class TInternetExplorer.

8.1.8.5.20 TInternetExplorer.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class TInternetExplorer.

8.1.8.5.21 TInternetExplorer.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class TInternetExplorer.

8.1.8.5.22 TInternetExplorer.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class TInternetExplorer.

8.1.8.5.23 TInternetExplorer.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class TInternetExplorer.

8.1.8.5.24 TInternetExplorer.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class TInternetExplorer.

8.1.8.5.25 TInternetExplorer.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class TInternetExplorer.

8.1.8.5.26 TInternetExplorer.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class TInternetExplorer.

8.1.8.5.27 TInternetExplorer.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class TInternetExplorer.

8.1.8.5.28 TInternetExplorer.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class TInternetExplorer.

8.1.8.5.29 TInternetExplorer.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class TInternetExplorer.

8.1.8.5.30 TInternetExplorer.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TInternetExplorer.

8.1.8.5.31 TInternetExplorer.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class TInternetExplorer.

8.1.8.5.32 TInternetExplorer.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class TInternetExplorer.

8.1.8.6 TInternetExplorer Events

The events of the TInternetExplorer class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 459)	This is OnBeforeNavigate2, a member of class TInternetExplorer.
	OnClientToHostWindow (see page 459)	This is OnClientToHostWindow, a member of class TInternetExplorer.
	OnCommandStateChange (see page 459)	This is OnCommandStateChange, a member of class TInternetExplorer.
	OnDocumentComplete (see page 459)	This is OnDocumentComplete, a member of class TInternetExplorer.
	OnDownloadBegin (see page 459)	This is OnDownloadBegin, a member of class TInternetExplorer.
	OnDownloadComplete (see page 460)	This is OnDownloadComplete, a member of class TInternetExplorer.
	OnFileDownload (see page 460)	This is OnFileDownload, a member of class TInternetExplorer.
	OnFullScreen (see page 460)	This is OnFullScreen, a member of class TInternetExplorer.
	OnMenuBar (see page 460)	This is OnMenuBar, a member of class TInternetExplorer.
	OnNavigateComplete2 (see page 461)	This is OnNavigateComplete2, a member of class TInternetExplorer.
	OnNavigateError (see page 461)	This is OnNavigateError, a member of class TInternetExplorer.
	OnNewWindow2 (see page 461)	This is OnNewWindow2, a member of class TInternetExplorer.
	OnPrintTemplateInstantiation (see page 461)	This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.
	OnPrintTemplateTeardown (see page 461)	This is OnPrintTemplateTeardown, a member of class TInternetExplorer.
	OnPrivacyImpactedStateChange (see page 462)	This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.
	OnProgressChange (see page 462)	This is OnProgressChange, a member of class TInternetExplorer.
	OnPropertyChange (see page 462)	This is OnPropertyChange, a member of class TInternetExplorer.
	OnQuit (see page 462)	This is OnQuit, a member of class TInternetExplorer.
	OnSetSecureLockIcon (see page 463)	This is OnSetSecureLockIcon, a member of class TInternetExplorer.
	OnStatusBar (see page 463)	This is OnStatusBar, a member of class TInternetExplorer.
	OnStatusTextChange (see page 463)	This is OnStatusTextChange, a member of class TInternetExplorer.
	OnTheaterMode (see page 463)	This is OnTheaterMode, a member of class TInternetExplorer.
	OnTitleChange (see page 463)	This is OnTitleChange, a member of class TInternetExplorer.
	OnToolBar (see page 464)	This is OnToolBar, a member of class TInternetExplorer.
	OnUpdatePageStatus (see page 464)	This is OnUpdatePageStatus, a member of class TInternetExplorer.
	OnVisible (see page 464)	This is OnVisible, a member of class TInternetExplorer.
	OnWindowClosing (see page 464)	This is OnWindowClosing, a member of class TInternetExplorer.
	OnWindowSetHeight (see page 465)	This is OnWindowSetHeight, a member of class TInternetExplorer.
	OnWindowSetLeft (see page 465)	This is OnWindowSetLeft, a member of class TInternetExplorer.
	OnWindowSetResizable (see page 465)	This is OnWindowSetResizable, a member of class TInternetExplorer.
	OnWindowSetTop (see page 465)	This is OnWindowSetTop, a member of class TInternetExplorer.
	OnWindowSetWidth (see page 465)	This is OnWindowSetWidth, a member of class TInternetExplorer.

8.1.8.6.1 TInternetExplorer.OnBeforeNavigate2 Event

Pascal

```
property OnBeforeNavigate2: TInternetExplorerBeforeNavigate2;
```

C++

```
__property TInternetExplorerBeforeNavigate2 OnBeforeNavigate2;
```

Description

This is OnBeforeNavigate2, a member of class TInternetExplorer.

8.1.8.6.2 TInternetExplorer.OnClientToHostWindow Event

Pascal

```
property OnClientToHostWindow: TInternetExplorerClientToHostWindow;
```

C++

```
__property TInternetExplorerClientToHostWindow OnClientToHostWindow;
```

Description

This is OnClientToHostWindow, a member of class TInternetExplorer.

8.1.8.6.3 TInternetExplorer.OnCommandStateChange Event

Pascal

```
property OnCommandStateChange: TInternetExplorerCommandStateChange;
```

C++

```
__property TInternetExplorerCommandStateChange OnCommandStateChange;
```

Description

This is OnCommandStateChange, a member of class TInternetExplorer.

8.1.8.6.4 TInternetExplorer.OnDocumentComplete Event

Pascal

```
property OnDocumentComplete: TInternetExplorerDocumentComplete;
```

C++

```
__property TInternetExplorerDocumentComplete OnDocumentComplete;
```

Description

This is OnDocumentComplete, a member of class TInternetExplorer.

8.1.8.6.5 TInternetExplorer.OnDownloadBegin Event

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

This is OnDownloadBegin, a member of class TInternetExplorer.

8.1.8.6.6 TInternetExplorer.OnDownloadComplete Event

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

This is OnDownloadComplete, a member of class TInternetExplorer.

8.1.8.6.7 TInternetExplorer.OnFileDownload Event

Pascal

```
property OnFileDownload: TInternetExplorerFileDownload;
```

C++

```
__property TInternetExplorerFileDownload OnFileDownload;
```

Description

This is OnFileDownload, a member of class TInternetExplorer.

8.1.8.6.8 TInternetExplorer.OnFullScreen Event

Pascal

```
property OnFullScreen: TInternetExplorerOnFullScreen;
```

C++

```
__property TInternetExplorerOnFullScreen OnFullScreen;
```

Description

This is OnFullScreen, a member of class TInternetExplorer.

8.1.8.6.9 TInternetExplorer.OnMenuBar Event

Pascal

```
property OnMenuBar: TInternetExplorerOnMenuBar;
```

C++

```
__property TInternetExplorerOnMenuBar OnMenuBar;
```

Description

This is OnMenuBar, a member of class TInternetExplorer.

8.1.8.6.10 TInternetExplorer.OnNavigateComplete2 Event

Pascal

```
property OnNavigateComplete2: TInternetExplorerNavigateComplete2;
```

C++

```
__property TInternetExplorerNavigateComplete2 OnNavigateComplete2;
```

Description

This is OnNavigateComplete2, a member of class TInternetExplorer.

8.1.8.6.11 TInternetExplorer.OnNavigateError Event

Pascal

```
property OnNavigateError: TInternetExplorerNavigateError;
```

C++

```
__property TInternetExplorerNavigateError OnNavigateError;
```

Description

This is OnNavigateError, a member of class TInternetExplorer.

8.1.8.6.12 TInternetExplorer.OnNewWindow2 Event

Pascal

```
property OnNewWindow2: TInternetExplorerNewWindow2;
```

C++

```
__property TInternetExplorerNewWindow2 OnNewWindow2;
```

Description

This is OnNewWindow2, a member of class TInternetExplorer.

8.1.8.6.13 TInternetExplorer.OnPrintTemplateInstantiation Event

Pascal

```
property OnPrintTemplateInstantiation: TInternetExplorerPrintTemplateInstantiation;
```

C++

```
__property TInternetExplorerPrintTemplateInstantiation OnPrintTemplateInstantiation;
```

Description

This is OnPrintTemplateInstantiation, a member of class TInternetExplorer.

8.1.8.6.14 TInternetExplorer.OnPrintTemplateTeardown Event

Pascal

```
property OnPrintTemplateTeardown: TInternetExplorerPrintTemplateTeardown;
```

C++

```
__property TInternetExplorerPrintTemplateTeardown OnPrintTemplateTeardown;
```

Description

This is OnPrintTemplateTeardown, a member of class TInternetExplorer.

8.1.8.6.15 TInternetExplorer.OnPrivacyImpactedStateChange Event

Pascal

```
property OnPrivacyImpactedStateChange: TInternetExplorerPrivacyImpactedStateChange;
```

C++

```
__property TInternetExplorerPrivacyImpactedStateChange OnPrivacyImpactedStateChange;
```

Description

This is OnPrivacyImpactedStateChange, a member of class TInternetExplorer.

8.1.8.6.16 TInternetExplorer.OnProgressChange Event

Pascal

```
property OnProgressChange: TInternetExplorerProgressChange;
```

C++

```
__property TInternetExplorerProgressChange OnProgressChange;
```

Description

This is OnProgressChange, a member of class TInternetExplorer.

8.1.8.6.17 TInternetExplorer.OnPropertyChange Event

Pascal

```
property OnPropertyChange: TInternetExplorerPropertyChange;
```

C++

```
__property TInternetExplorerPropertyChange OnPropertyChange;
```

Description

This is OnPropertyChange, a member of class TInternetExplorer.

8.1.8.6.18 TInternetExplorer.OnQuit Event

Pascal

```
property OnQuit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnQuit;
```

Description

This is OnQuit, a member of class TInternetExplorer.

8.1.8.6.19 TInternetExplorer.OnSetSecureLockIcon Event

Pascal

```
property OnSetSecureLockIcon: TInternetExplorerSetSecureLockIcon;
```

C++

```
__property TInternetExplorerSetSecureLockIcon OnSetSecureLockIcon;
```

Description

This is OnSetSecureLockIcon, a member of class TInternetExplorer.

8.1.8.6.20 TInternetExplorer.OnStatusBar Event

Pascal

```
property OnStatusBar: TInternetExplorerOnStatusBar;
```

C++

```
__property TInternetExplorerOnStatusBar OnStatusBar;
```

Description

This is OnStatusBar, a member of class TInternetExplorer.

8.1.8.6.21 TInternetExplorer.OnStatusTextChange Event

Pascal

```
property OnStatusTextChange: TInternetExplorerStatusTextChange;
```

C++

```
__property TInternetExplorerStatusTextChange OnStatusTextChange;
```

Description

This is OnStatusTextChange, a member of class TInternetExplorer.

8.1.8.6.22 TInternetExplorer.OnTheaterMode Event

Pascal

```
property OnTheaterMode: TInternetExplorerOnTheaterMode;
```

C++

```
__property TInternetExplorerOnTheaterMode OnTheaterMode;
```

Description

This is OnTheaterMode, a member of class TInternetExplorer.

8.1.8.6.23 TInternetExplorer.OnTitleChange Event

Pascal

```
property OnTitleChange: TInternetExplorerTitleChange;
```

C++

```
__property TInternetExplorerTitleChange OnTitleChange;
```

Description

This is OnTitleChange, a member of class TInternetExplorer.

8.1.8.6.24 TInternetExplorer.OnToolBar Event

Pascal

```
property OnToolBar: TInternetExplorerOnToolBar;
```

C++

```
__property TInternetExplorerOnToolBar OnToolBar;
```

Description

This is OnToolBar, a member of class TInternetExplorer.

8.1.8.6.25 TInternetExplorer.OnUpdatePageStatus Event

Pascal

```
property OnUpdatePageStatus: TInternetExplorerUpdatePageStatus;
```

C++

```
__property TInternetExplorerUpdatePageStatus OnUpdatePageStatus;
```

Description

This is OnUpdatePageStatus, a member of class TInternetExplorer.

8.1.8.6.26 TInternetExplorer.OnVisible Event

Pascal

```
property OnVisible: TInternetExplorerOnVisible;
```

C++

```
__property TInternetExplorerOnVisible OnVisible;
```

Description

This is OnVisible, a member of class TInternetExplorer.

8.1.8.6.27 TInternetExplorer.OnWindowClosing Event

Pascal

```
property OnWindowClosing: TInternetExplorerWindowClosing;
```

C++

```
__property TInternetExplorerWindowClosing OnWindowClosing;
```

Description

This is OnWindowClosing, a member of class TInternetExplorer.

8.1.8.6.28 TInternetExplorer.OnWindowSetHeight Event

Pascal

```
property OnWindowSetHeight: TInternetExplorerWindowSetHeight;
```

C++

```
__property TInternetExplorerWindowSetHeight OnWindowSetHeight;
```

Description

This is OnWindowSetHeight, a member of class TInternetExplorer.

8.1.8.6.29 TInternetExplorer.OnWindowSetLeft Event

Pascal

```
property OnWindowSetLeft: TInternetExplorerWindowSetLeft;
```

C++

```
__property TInternetExplorerWindowSetLeft OnWindowSetLeft;
```

Description

This is OnWindowSetLeft, a member of class TInternetExplorer.

8.1.8.6.30 TInternetExplorer.OnWindowSetResizable Event

Pascal

```
property OnWindowSetResizable: TInternetExplorerWindowSetResizable;
```

C++

```
__property TInternetExplorerWindowSetResizable OnWindowSetResizable;
```

Description

This is OnWindowSetResizable, a member of class TInternetExplorer.

8.1.8.6.31 TInternetExplorer.OnWindowSetTop Event

Pascal

```
property OnWindowSetTop: TInternetExplorerWindowSetTop;
```

C++

```
__property TInternetExplorerWindowSetTop OnWindowSetTop;
```

Description

This is OnWindowSetTop, a member of class TInternetExplorer.

8.1.8.6.32 TInternetExplorer.OnWindowSetWidth Event

Pascal

```
property OnWindowSetWidth: TInternetExplorerWindowSetWidth;
```

C++

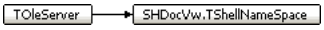
```
__property TInternetExplorerWindowSetWidth OnWindowSetWidth;
```

Description

This is OnWindowSetWidth, a member of class TInternetExplorer.

8.1.9 TShellNameSpace Class

Class Hierarchy



Pascal

```
TShellNameSpace = class(TOleServer);
```

C++

```
class TShellNameSpace : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellNameSpace.

8.1.9.1 TShellNameSpace Members

The following tables list the members exposed by TShellNameSpace.

Protected Methods

	Name	Description
	Get_Columns (see page 472)	This is Get_Columns, a member of class TShellNameSpace.
	Get_CountViewTypes (see page 472)	This is Get_CountViewTypes, a member of class TShellNameSpace.
	Get_Depth (see page 472)	This is Get_Depth, a member of class TShellNameSpace.
	Get_EnumOptions (see page 473)	This is Get_EnumOptions, a member of class TShellNameSpace.
	Get_Flags (see page 473)	This is Get_Flags, a member of class TShellNameSpace.
	Get_Mode (see page 473)	This is Get_Mode, a member of class TShellNameSpace.
	Get_Root (see page 473)	This is Get_Root, a member of class TShellNameSpace.
	Get_SelectedItem (see page 474)	This is Get_SelectedItem, a member of class TShellNameSpace.
	Get_SubscriptionsEnabled (see page 474)	This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.
	Get_TVFlags (see page 474)	This is Get_TVFlags, a member of class TShellNameSpace.
	InitServerData (see page 474)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 475)	Dispatches an event from the COM server to the appropriate event handler.
	Set_Columns (see page 477)	This is Set_Columns, a member of class TShellNameSpace.
	Set_Depth (see page 477)	This is Set_Depth, a member of class TShellNameSpace.
	Set_EnumOptions (see page 477)	This is Set_EnumOptions, a member of class TShellNameSpace.
	Set_Flags (see page 477)	This is Set_Flags, a member of class TShellNameSpace.
	Set_Mode (see page 478)	This is Set_Mode, a member of class TShellNameSpace.
	Set_Root (see page 478)	This is Set_Root, a member of class TShellNameSpace.
	Set_SelectedItem (see page 478)	This is Set_SelectedItem, a member of class TShellNameSpace.
	Set_TVFlags (see page 478)	This is Set_TVFlags, a member of class TShellNameSpace.

Public Methods

	Name	Description
V	Connect (see page 470)	Establishes a connection to the COM server.
	ConnectTo (see page 470)	This is ConnectTo, a member of class TShellNameSpace.
V	Create (see page 467)	Allocates memory and constructs a safely initialized instance of a component.
	CreateSubscriptionForSelection (see page 471)	This is CreateSubscriptionForSelection, a member of class TShellNameSpace.
	DeleteSubscriptionForSelection (see page 471)	This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.
V	Destroy (see page 468)	Disposes of the component and its owned components.
V	Disconnect (see page 471)	Terminates the connection to the COM server.
	Expand (see page 472)	This is Expand, a member of class TShellNameSpace.
	Export (see page 472)	This is Export, a member of class TShellNameSpace.
	Import (see page 474)	This is Import, a member of class TShellNameSpace.
	InvokeContextMenuCommand (see page 475)	This is InvokeContextMenuCommand, a member of class TShellNameSpace.
	MoveSelectionDown (see page 475)	This is MoveSelectionDown, a member of class TShellNameSpace.
	MoveSelectionTo (see page 476)	This is MoveSelectionTo, a member of class TShellNameSpace.
	MoveSelectionUp (see page 476)	This is MoveSelectionUp, a member of class TShellNameSpace.
	NewFolder (see page 476)	This is NewFolder, a member of class TShellNameSpace.
	ResetSort (see page 476)	This is ResetSort, a member of class TShellNameSpace.
	SelectedItems (see page 477)	This is SelectedItems, a member of class TShellNameSpace.
	SetRoot (see page 479)	This is SetRoot, a member of class TShellNameSpace.
	SetViewType (see page 479)	This is SetViewType, a member of class TShellNameSpace.
	Synchronize (see page 479)	This is Synchronize, a member of class TShellNameSpace.
	UnselectAll (see page 479)	This is UnselectAll, a member of class TShellNameSpace.

Public Properties

	Name	Description
	Columns (see page 480)	This is Columns, a member of class TShellNameSpace.
R	CountViewTypes (see page 480)	This is CountViewTypes, a member of class TShellNameSpace.
R	DefaultInterface (see page 480)	This is DefaultInterface, a member of class TShellNameSpace.
	Depth (see page 480)	This is Depth, a member of class TShellNameSpace.
	EnumOptions (see page 481)	This is EnumOptions, a member of class TShellNameSpace.
	Flags (see page 481)	This is Flags, a member of class TShellNameSpace.
	Mode (see page 481)	This is Mode, a member of class TShellNameSpace.
	Root (see page 481)	This is Root, a member of class TShellNameSpace.
	SelectedItem (see page 482)	This is SelectedItem, a member of class TShellNameSpace.
R	SubscriptionsEnabled (see page 482)	This is SubscriptionsEnabled, a member of class TShellNameSpace.
	TVFlags (see page 482)	This is TVFlags, a member of class TShellNameSpace.

Published Events

	Name	Description
	OnDoubleClick (see page 482)	This is OnDoubleClick, a member of class TShellNameSpace.
	OnFavoritesSelectionChange (see page 483)	This is OnFavoritesSelectionChange, a member of class TShellNameSpace.
	OnInitialized (see page 483)	This is OnInitialized, a member of class TShellNameSpace.
	OnSelectionChange (see page 483)	This is OnSelectionChange, a member of class TShellNameSpace.

8.1.9.2 TShellNameSpace.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShellNameSpace(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3355)

Destroy

System::TObject::Free (see page 3372)

Owner

8.1.9.3 TShellNameSpace.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellNameSpace();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3356)

DestroyComponents

Destroying

System::TObject::Free (see page 3372)

Notification

Owner

Release

Components

8.1.9.4 TShellNameSpace Methods

The methods of the TShellNameSpace class are listed here.

Protected Methods

	Name	Description
	Get_Columns (see page 472)	This is Get_Columns, a member of class TShellNameSpace.
	Get_CountViewTypes (see page 472)	This is Get_CountViewTypes, a member of class TShellNameSpace.
	Get_Depth (see page 472)	This is Get_Depth, a member of class TShellNameSpace.
	Get_EnumOptions (see page 473)	This is Get_EnumOptions, a member of class TShellNameSpace.
	Get_Flags (see page 473)	This is Get_Flags, a member of class TShellNameSpace.
	Get_Mode (see page 473)	This is Get_Mode, a member of class TShellNameSpace.
	Get_Root (see page 473)	This is Get_Root, a member of class TShellNameSpace.
	Get_SelectedItem (see page 474)	This is Get_SelectedItem, a member of class TShellNameSpace.
	Get_SubscriptionsEnabled (see page 474)	This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.
	Get_TVFlags (see page 474)	This is Get_TVFlags, a member of class TShellNameSpace.
	InitServerData (see page 474)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 475)	Dispatches an event from the COM server to the appropriate event handler.
	Set_Columns (see page 477)	This is Set_Columns, a member of class TShellNameSpace.
	Set_Depth (see page 477)	This is Set_Depth, a member of class TShellNameSpace.
	Set_EnumOptions (see page 477)	This is Set_EnumOptions, a member of class TShellNameSpace.
	Set_Flags (see page 477)	This is Set_Flags, a member of class TShellNameSpace.
	Set_Mode (see page 478)	This is Set_Mode, a member of class TShellNameSpace.
	Set_Root (see page 478)	This is Set_Root, a member of class TShellNameSpace.
	Set_SelectedItem (see page 478)	This is Set_SelectedItem, a member of class TShellNameSpace.
	Set_TVFlags (see page 478)	This is Set_TVFlags, a member of class TShellNameSpace.

Public Methods

	Name	Description
	Connect (see page 470)	Establishes a connection to the COM server.
	ConnectTo (see page 470)	This is ConnectTo, a member of class TShellNameSpace.
	CreateSubscriptionForSelection (see page 471)	This is CreateSubscriptionForSelection, a member of class TShellNameSpace.

	DeleteSubscriptionForSelection (see page 471)	This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.
	Disconnect (see page 471)	Terminates the connection to the COM server.
	Expand (see page 472)	This is Expand, a member of class TShellNameSpace.
	Export (see page 472)	This is Export, a member of class TShellNameSpace.
	Import (see page 474)	This is Import, a member of class TShellNameSpace.
	InvokeContextMenuCommand (see page 475)	This is InvokeContextMenuCommand, a member of class TShellNameSpace.
	MoveSelectionDown (see page 475)	This is MoveSelectionDown, a member of class TShellNameSpace.
	MoveSelectionTo (see page 476)	This is MoveSelectionTo, a member of class TShellNameSpace.
	MoveSelectionUp (see page 476)	This is MoveSelectionUp, a member of class TShellNameSpace.
	NewFolder (see page 476)	This is NewFolder, a member of class TShellNameSpace.
	ResetSort (see page 476)	This is ResetSort, a member of class TShellNameSpace.
	SelectedItems (see page 477)	This is SelectedItems, a member of class TShellNameSpace.
	SetRoot (see page 479)	This is SetRoot, a member of class TShellNameSpace.
	SetViewType (see page 479)	This is SetViewType, a member of class TShellNameSpace.
	Synchronize (see page 479)	This is Synchronize, a member of class TShellNameSpace.
	UnselectAll (see page 479)	This is UnselectAll, a member of class TShellNameSpace.

8.1.9.4.1 TShellNameSpace.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

8.1.9.4.2 TShellNameSpace.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellNameSpace);
```

C++

```
__fastcall ConnectTo(IShellNameSpace svrIntf);
```

Description

This is ConnectTo, a member of class TShellNameSpace.

8.1.9.4.3 TShellNameSpace.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class TShellNameSpace.

8.1.9.4.4 TShellNameSpace.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class TShellNameSpace.

8.1.9.4.5 TShellNameSpace.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

8.1.9.4.6 TShellNameSpace.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT);
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class TShellNameSpace.

8.1.9.4.7 TShellNameSpace.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class TShellNameSpace.

8.1.9.4.8 TShellNameSpace.Get_Columns Method

Pascal

```
function Get_Columns: WideString;
```

C++

```
__fastcall BSTR Get_Columns();
```

Description

This is Get_Columns, a member of class TShellNameSpace.

8.1.9.4.9 TShellNameSpace.Get_CountViewTypes Method

Pascal

```
function Get_CountViewTypes: SYSINT;
```

C++

```
__fastcall SYSINT Get_CountViewTypes();
```

Description

This is Get_CountViewTypes, a member of class TShellNameSpace.

8.1.9.4.10 TShellNameSpace.Get_Depth Method

Pascal

```
function Get_Depth: SYSINT;
```

C++

```
__fastcall SYSINT Get_Depth();
```

Description

This is Get_Depth, a member of class TShellNameSpace.

8.1.9.4.11 TShellNameSpace.Get_EnumOptions Method

Pascal

```
function Get_EnumOptions: Integer;
```

C++

```
__fastcall int Get_EnumOptions();
```

Description

This is Get_EnumOptions, a member of class TShellNameSpace.

8.1.9.4.12 TShellNameSpace.Get_Flags Method

Pascal

```
function Get_Flags: LongWord;
```

C++

```
__fastcall LongWord Get_Flags();
```

Description

This is Get_Flags, a member of class TShellNameSpace.

8.1.9.4.13 TShellNameSpace.Get_Mode Method

Pascal

```
function Get_Mode: SYSUINT;
```

C++

```
__fastcall SYSUINT Get_Mode();
```

Description

This is Get_Mode, a member of class TShellNameSpace.

8.1.9.4.14 TShellNameSpace.Get_Root Method

Pascal

```
function Get_Root: OleVariant;
```

C++

```
__fastcall OleVariant Get_Root();
```

Description

This is Get_Root, a member of class TShellNameSpace.

8.1.9.4.15 TShellNameSpace.Get_SelectedItem Method

Pascal

```
function Get_SelectedItem: IDispatch;
```

C++

```
__fastcall IDispatch Get_SelectedItem();
```

Description

This is Get_SelectedItem, a member of class TShellNameSpace.

8.1.9.4.16 TShellNameSpace.Get_SubscriptionsEnabled Method

Pascal

```
function Get_SubscriptionsEnabled: WordBool;
```

C++

```
__fastcall WordBool Get_SubscriptionsEnabled();
```

Description

This is Get_SubscriptionsEnabled, a member of class TShellNameSpace.

8.1.9.4.17 TShellNameSpace.Get_TVFlags Method

Pascal

```
function Get_TVFlags: LongWord;
```

C++

```
__fastcall LongWord Get_TVFlags();
```

Description

This is Get_TVFlags, a member of class TShellNameSpace.

8.1.9.4.18 TShellNameSpace.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class TShellNameSpace.

8.1.9.4.19 TShellNameSpace.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

8.1.9.4.20 TShellNameSpace.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class TShellNameSpace.

8.1.9.4.21 TShellNameSpace.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the appropriate event handlers.

See Also

EventDispatch

8.1.9.4.22 TShellNameSpace.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class TShellNameSpace.

8.1.9.4.23 TShellNameSpace.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class TShellNameSpace.

8.1.9.4.24 TShellNameSpace.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class TShellNameSpace.

8.1.9.4.25 TShellNameSpace.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class TShellNameSpace.

8.1.9.4.26 TShellNameSpace.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class TShellNameSpace.

8.1.9.4.27 TShellNameSpace.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class TShellNameSpace.

8.1.9.4.28 TShellNameSpace.Set_Columns Method

Pascal

```
procedure Set_Columns(const bstrColumns: WideString);
```

C++

```
__fastcall Set_Columns(const BSTR bstrColumns);
```

Description

This is Set_Columns, a member of class TShellNameSpace.

8.1.9.4.29 TShellNameSpace.Set_Depth Method

Pascal

```
procedure Set_Depth(piDepth: SYSINT);
```

C++

```
__fastcall Set_Depth(SYSINT piDepth);
```

Description

This is Set_Depth, a member of class TShellNameSpace.

8.1.9.4.30 TShellNameSpace.Set_EnumOptions Method

Pascal

```
procedure Set_EnumOptions(pgrfEnumFlags: Integer);
```

C++

```
__fastcall Set_EnumOptions(int pgrfEnumFlags);
```

Description

This is Set_EnumOptions, a member of class TShellNameSpace.

8.1.9.4.31 TShellNameSpace.Set_Flags Method

Pascal

```
procedure Set_Flags(pdwFlags: LongWord);
```

C++

```
__fastcall Set_Flags(LongWord pdwFlags);
```

Description

This is Set_Flags, a member of class TShellNameSpace.

8.1.9.4.32 TShellNameSpace.Set_Mode Method

Pascal

```
procedure Set_Mode(puMode: SYSUINT);
```

C++

```
__fastcall Set_Mode(SYSUINT puMode);
```

Description

This is Set_Mode, a member of class TShellNameSpace.

8.1.9.4.33 TShellNameSpace.Set_Root Method

Pascal

```
procedure Set_Root(pvar: OleVariant);
```

C++

```
__fastcall Set_Root(OleVariant pvar);
```

Description

This is Set_Root, a member of class TShellNameSpace.

8.1.9.4.34 TShellNameSpace.Set_SelectedItem Method

Pascal

```
procedure Set_SelectedItem(const pItem: IDispatch);
```

C++

```
__fastcall Set_SelectedItem(const IDispatch pItem);
```

Description

This is Set_SelectedItem, a member of class TShellNameSpace.

8.1.9.4.35 TShellNameSpace.Set_TVFlags Method

Pascal

```
procedure Set_TVFlags(dwFlags: LongWord);
```

C++

```
__fastcall Set_TVFlags(LongWord dwFlags);
```

Description

This is Set_TVFlags, a member of class TShellNameSpace.

8.1.9.4.36 TShellNameSpace.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class TShellNameSpace.

8.1.9.4.37 TShellNameSpace.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT);
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class TShellNameSpace.

8.1.9.4.38 TShellNameSpace.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class TShellNameSpace.

8.1.9.4.39 TShellNameSpace.UnselectAll Method

Pascal

```
procedure UnselectAll;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class TShellNameSpace.

8.1.9.5 TShellNameSpace Properties

The properties of the TShellNameSpace class are listed here.

Public Properties

	Name	Description
	Columns (see page 480)	This is Columns, a member of class TShellNameSpace.
R	CountViewTypes (see page 480)	This is CountViewTypes, a member of class TShellNameSpace.
R	DefaultInterface (see page 480)	This is DefaultInterface, a member of class TShellNameSpace.
	Depth (see page 480)	This is Depth, a member of class TShellNameSpace.
	EnumOptions (see page 481)	This is EnumOptions, a member of class TShellNameSpace.
	Flags (see page 481)	This is Flags, a member of class TShellNameSpace.
	Mode (see page 481)	This is Mode, a member of class TShellNameSpace.
	Root (see page 481)	This is Root, a member of class TShellNameSpace.
	SelectedItem (see page 482)	This is SelectedItem, a member of class TShellNameSpace.
R	SubscriptionsEnabled (see page 482)	This is SubscriptionsEnabled, a member of class TShellNameSpace.
	TVFlags (see page 482)	This is TVFlags, a member of class TShellNameSpace.

8.1.9.5.1 TShellNameSpace.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class TShellNameSpace.

8.1.9.5.2 TShellNameSpace.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class TShellNameSpace.

8.1.9.5.3 TShellNameSpace.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellNameSpace;
```

C++

```
__property IShellNameSpace DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellNameSpace.

8.1.9.5.4 TShellNameSpace.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class TShellNameSpace.

8.1.9.5.5 TShellNameSpace.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class TShellNameSpace.

8.1.9.5.6 TShellNameSpace.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class TShellNameSpace.

8.1.9.5.7 TShellNameSpace.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class TShellNameSpace.

8.1.9.5.8 TShellNameSpace.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class TShellNameSpace.

8.1.9.5.9 TShellNameSpace.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class TShellNameSpace.

8.1.9.5.10 TShellNameSpace.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class TShellNameSpace.

8.1.9.5.11 TShellNameSpace.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class TShellNameSpace.

8.1.9.6 TShellNameSpace Events

The events of the TShellNameSpace class are listed here.

Published Events

	Name	Description
	OnDoubleClick (see page 482)	This is OnDoubleClick, a member of class TShellNameSpace.
	OnFavoritesSelectionChange (see page 483)	This is OnFavoritesSelectionChange, a member of class TShellNameSpace.
	OnInitialized (see page 483)	This is OnInitialized, a member of class TShellNameSpace.
	OnSelectionChange (see page 483)	This is OnSelectionChange, a member of class TShellNameSpace.

8.1.9.6.1 TShellNameSpace.OnDoubleClick Event

Pascal

```
property OnDoubleClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDoubleClick;
```

Description

This is OnDoubleClick, a member of class TShellNameSpace.

8.1.9.6.2 TShellNameSpace.OnFavoritesSelectionChange Event

Pascal

```
property OnFavoritesSelectionChange: TShellNameSpaceFavoritesSelectionChange;
```

C++

```
__property TShellNameSpaceFavoritesSelectionChange OnFavoritesSelectionChange;
```

Description

This is OnFavoritesSelectionChange, a member of class TShellNameSpace.

8.1.9.6.3 TShellNameSpace.OnInitialized Event

Pascal

```
property OnInitialized: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnInitialized;
```

Description

This is OnInitialized, a member of class TShellNameSpace.

8.1.9.6.4 TShellNameSpace.OnSelectionChange Event

Pascal

```
property OnSelectionChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelectionChange;
```

Description

This is OnSelectionChange, a member of class TShellNameSpace.

8.1.10 TShellUIHelper Class

Class Hierarchy

Pascal

```
TShellUIHelper = class(TOleServer);
```

C++

```
class TShellUIHelper : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellUIHelper.

8.1.10.1 TShellUIHelper Members

The following tables list the members exposed by TShellUIHelper.

Protected Methods

	Name	Description
	InitServerData (see page 492)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	AddChannel (see page 486)	This is AddChannel, a member of class TShellUIHelper.
	AddDesktopComponent (see page 487)	This is the overview for the AddDesktopComponent method overload.
	AddFavorite (see page 488)	This is the overview for the AddFavorite method overload.
	AutoCompleteAttach (see page 489)	This is the overview for the AutoCompleteAttach method overload.
	AutoCompleteSaveForm (see page 490)	This is the overview for the AutoCompleteSaveForm method overload.
	AutoScan (see page 490)	This is the overview for the AutoScan method overload.
	Connect (see page 491)	Establishes a connection to the COM server.
	ConnectTo (see page 491)	This is ConnectTo, a member of class TShellUIHelper.
	Create (see page 484)	Allocates memory and constructs a safely initialized instance of a component.
	Destroy (see page 485)	Disposes of the component and its owned components.
	Disconnect (see page 492)	Terminates the connection to the COM server.
	ImportExportFavorites (see page 492)	This is ImportExportFavorites, a member of class TShellUIHelper.
	IsSubscribed (see page 493)	This is IsSubscribed, a member of class TShellUIHelper.
	NavigateAndFind (see page 493)	This is NavigateAndFind, a member of class TShellUIHelper.
	RefreshOfflineDesktop (see page 493)	This is RefreshOfflineDesktop, a member of class TShellUIHelper.
	ResetFirstBootMode (see page 493)	This is ResetFirstBootMode, a member of class TShellUIHelper.
	ResetSafeMode (see page 494)	This is ResetSafeMode, a member of class TShellUIHelper.
	ShowBrowserUI (see page 494)	This is ShowBrowserUI, a member of class TShellUIHelper.

Public Properties

	Name	Description
	DefaultInterface (see page 494)	This is DefaultInterface, a member of class TShellUIHelper.

8.1.10.2 TShellUIHelper.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShellUIHelper(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3355)

Destroy

System::TObject::Free (see page 3372)

Owner

8.1.10.3 TShellUIHelper.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellUIHelper();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3356)

DestroyComponents

Destroying

System::TObject::Free (see page 3372)

Notification

Owner

Release

Components

8.1.10.4 TShellUIHelper Methods

The methods of the TShellUIHelper class are listed here.

Protected Methods

	Name	Description
	InitServerData (see page 492)	Initializes the ServerData property to reflect the type library information of the COM server.

Public Methods

	Name	Description
	AddChannel (see page 486)	This is AddChannel, a member of class TShellUIHelper.
	AddDesktopComponent (see page 487)	This is the overview for the AddDesktopComponent method overload.
	AddFavorite (see page 488)	This is the overview for the AddFavorite method overload.
	AutoCompleteAttach (see page 489)	This is the overview for the AutoCompleteAttach method overload.
	AutoCompleteSaveForm (see page 490)	This is the overview for the AutoCompleteSaveForm method overload.
	AutoScan (see page 490)	This is the overview for the AutoScan method overload.
	Connect (see page 491)	Establishes a connection to the COM server.
	ConnectTo (see page 491)	This is ConnectTo, a member of class TShellUIHelper.
	Disconnect (see page 492)	Terminates the connection to the COM server.
	ImportExportFavorites (see page 492)	This is ImportExportFavorites, a member of class TShellUIHelper.
	IsSubscribed (see page 493)	This is IsSubscribed, a member of class TShellUIHelper.
	NavigateAndFind (see page 493)	This is NavigateAndFind, a member of class TShellUIHelper.
	RefreshOfflineDesktop (see page 493)	This is RefreshOfflineDesktop, a member of class TShellUIHelper.
	ResetFirstBootMode (see page 493)	This is ResetFirstBootMode, a member of class TShellUIHelper.
	ResetSafeMode (see page 494)	This is ResetSafeMode, a member of class TShellUIHelper.
	ShowBrowserUI (see page 494)	This is ShowBrowserUI, a member of class TShellUIHelper.

8.1.10.4.1 TShellUIHelper.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString);
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class TShellUIHelper.

8.1.10.4.2 AddDesktopComponent Method

This is the overview for the AddDesktopComponent method overload.

Overload List

	Name	Description
	TShellUIHelper.AddDesktopComponent (WideString, WideString) (see page 487)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant) (see page 487)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant) (see page 487)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant, OleVariant) (see page 488)	This is AddDesktopComponent, a member of class TShellUIHelper.
	TShellUIHelper.AddDesktopComponent (WideString, WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 488)	This is AddDesktopComponent, a member of class TShellUIHelper.

8.1.10.4.2.1 TShellUIHelper.AddDesktopComponent Method (WideString, WideString)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

8.1.10.4.2.2 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

8.1.10.4.2.3 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

8.1.10.4.2.4 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

8.1.10.4.2.5 TShellUIHelper.AddDesktopComponent Method (WideString, WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant); overload;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class TShellUIHelper.

8.1.10.4.3 AddFavorite Method

This is the overview for the AddFavorite method overload.

Overload List

	Name	Description
	TShellUIHelper.AddFavorite (WideString) (see page 488)	This is AddFavorite, a member of class TShellUIHelper.
	TShellUIHelper.AddFavorite (WideString, OleVariant) (see page 489)	This is AddFavorite, a member of class TShellUIHelper.

8.1.10.4.3.1 TShellUIHelper.AddFavorite Method (WideString)

Pascal

```
procedure AddFavorite(const URL: WideString); overload;
```

C++

```
__fastcall AddFavorite(const BSTR URL);
```

Description

This is AddFavorite, a member of class TShellUIHelper.

8.1.10.4.3.2 TShellUIHelper.AddFavorite Method (WideString, OleVariant)

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant); overload;
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class TShellUIHelper.

8.1.10.4.4 AutoCompleteAttach Method

This is the overview for the AutoCompleteAttach method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoCompleteAttach () (see page 489)	This is AutoCompleteAttach, a member of class TShellUIHelper.
	TShellUIHelper.AutoCompleteAttach (OleVariant) (see page 489)	This is AutoCompleteAttach, a member of class TShellUIHelper.

8.1.10.4.4.1 TShellUIHelper.AutoCompleteAttach Method ()

Pascal

```
procedure AutoCompleteAttach; overload;
```

C++

```
__fastcall AutoCompleteAttach();
```

Description

This is AutoCompleteAttach, a member of class TShellUIHelper.

8.1.10.4.4.2 TShellUIHelper.AutoCompleteAttach Method (OleVariant)

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant); overload;
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class TShellUIHelper.

8.1.10.4.5 AutoCompleteSaveForm Method

This is the overview for the AutoCompleteSaveForm method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoCompleteSaveForm () (see page 490)	This is AutoCompleteSaveForm, a member of class TShellUIHelper.
	TShellUIHelper.AutoCompleteSaveForm (OleVariant) (see page 490)	This is AutoCompleteSaveForm, a member of class TShellUIHelper.

8.1.10.4.5.1 TShellUIHelper.AutoCompleteSaveForm Method ()

Pascal

```
procedure AutoCompleteSaveForm; overload;
```

C++

```
__fastcall AutoCompleteSaveForm();
```

Description

This is AutoCompleteSaveForm, a member of class TShellUIHelper.

8.1.10.4.5.2 TShellUIHelper.AutoCompleteSaveForm Method (OleVariant)

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant); overload;
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class TShellUIHelper.

8.1.10.4.6 AutoScan Method

This is the overview for the AutoScan method overload.

Overload List

	Name	Description
	TShellUIHelper.AutoScan (WideString, WideString) (see page 490)	This is AutoScan, a member of class TShellUIHelper.
	TShellUIHelper.AutoScan (WideString, WideString, OleVariant) (see page 491)	This is AutoScan, a member of class TShellUIHelper.

8.1.10.4.6.1 TShellUIHelper.AutoScan Method (WideString, WideString)

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString); overload;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl);
```

Description

This is AutoScan, a member of class TShellUIHelper.

8.1.10.4.6.2 TShellUIHelper.AutoScan Method (WideString, WideString, OleVariant)

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant); overload;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class TShellUIHelper.

8.1.10.4.7 TShellUIHelper.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

8.1.10.4.8 TShellUIHelper.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellUIHelper);
```

C++

```
__fastcall ConnectTo(IShellUIHelper svrIntf);
```

Description

This is ConnectTo, a member of class TShellUIHelper.

8.1.10.4.9 TShellUIHelper.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

8.1.10.4.10 TShellUIHelper.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString);
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class TShellUIHelper.

8.1.10.4.11 TShellUIHelper.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

8.1.10.4.12 TShellUIHelper.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class TShellUIHelper.

8.1.10.4.13 TShellUIHelper.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant);
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class TShellUIHelper.

8.1.10.4.14 TShellUIHelper.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class TShellUIHelper.

8.1.10.4.15 TShellUIHelper.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class TShellUIHelper.

8.1.10.4.16 TShellUIHelper.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class TShellUIHelper.

8.1.10.4.17 TShellUIHelper.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class TShellUIHelper.

8.1.10.5 TShellUIHelper Properties

The properties of the TShellUIHelper class are listed here.

Public Properties

	Name	Description
	DefaultInterface (see page 494)	This is DefaultInterface, a member of class TShellUIHelper.

8.1.10.5.1 TShellUIHelper.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellUIHelper;
```

C++

```
__property IShellUIHelper DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellUIHelper.

8.1.11 TShellWindows Class

Class Hierarchy

Pascal

```
TShellWindows = class(TOleServer);
```

C++

```
class TShellWindows : public TOleServer;
```

File

SHDocVw

Description

This is class SHDocVw.TShellWindows.

8.1.11.1 TShellWindows Members

The following tables list the members exposed by TShellWindows.

Protected Methods

	Name	Description
	Get_Count (see page 499)	This is Get_Count, a member of class TShellWindows.
V	InitServerData (see page 499)	Initializes the ServerData property to reflect the type library information of the COM server.
V	InvokeEvent (see page 500)	Dispatches an event from the COM server to the appropriate event handler.

Public Methods

	Name	Description
	_NewEnum (see page 498)	This is _NewEnum, a member of class TShellWindows.
V	Connect (see page 498)	Establishes a connection to the COM server.
	ConnectTo (see page 498)	This is ConnectTo, a member of class TShellWindows.
V	Create (see page 495)	Allocates memory and constructs a safely initialized instance of a component.
V	Destroy (see page 496)	Disposes of the component and its owned components.
V	Disconnect (see page 499)	Terminates the connection to the COM server.
	FindWindowSW (see page 499)	This is FindWindowSW, a member of class TShellWindows.
	Item (see page 500)	This is the overview for the Item method overload.
	OnActivated (see page 501)	This is OnActivated, a member of class TShellWindows.
	OnCreated (see page 501)	This is OnCreated, a member of class TShellWindows.
	OnNavigate (see page 501)	This is OnNavigate, a member of class TShellWindows.
	ProcessAttachDetach (see page 502)	This is ProcessAttachDetach, a member of class TShellWindows.
	Register (see page 502)	This is Register, a member of class TShellWindows.
	RegisterPending (see page 502)	This is RegisterPending, a member of class TShellWindows.
	Revoke (see page 502)	This is Revoke, a member of class TShellWindows.

Public Properties

	Name	Description
R	Count (see page 503)	This is Count, a member of class TShellWindows.
R	DefaultInterface (see page 503)	This is DefaultInterface, a member of class TShellWindows.

Published Events

	Name	Description
	OnWindowRegistered (see page 503)	This is OnWindowRegistered, a member of class TShellWindows.
	OnWindowRevoked (see page 504)	This is OnWindowRevoked, a member of class TShellWindows.

8.1.11.2 TShellWindows.Create Constructor

Allocates memory and constructs a safely initialized instance of a component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TShellWindows(TComponent * AOwner);
```

Description

All objects have a Create method that constructs the object. TComponent redefines Create so that, for components, Create also:

establishes the relationship of a component and its Owner, as indicated by the AOwner parameter

sets the ComponentStyle property to csInheritable, meaning that the component can be inherited by a descendant form type

It is not necessary to explicitly create components added in the form designer. These components are created automatically when the application is run, and they are destroyed when the application is closed.

For components created programmatically, that is, not created in the form designer, call Create and pass in an owner component as the AOwner parameter. The owner disposes of the component when it is destroyed. If the component is not owned, then use Free when it needs to be destroyed.

Tip: When passing in Self as the Owner parameter, consider what Self references. If a component creates another component in one of its methods, then Self refers to the first component and not the component being created, which is then owned by the first component.

Note: The TComponent constructor is virtual in part to allow polymorphic instantiation of class references. This is critical to the streaming system and to the form designer. Do not forget to use the override directive when declaring a new component's Create constructor.

See Also

Components

ComponentStyle

System::TObject::Create (see page 3355)

Destroy

System::TObject::Free (see page 3372)

Owner

8.1.11.3 TShellWindows.Destroy Destructor

Disposes of the component and its owned components.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TShellWindows();
```

Description

Do not call Destroy directly. Call Free instead. Free verifies that the component is not nil, and only then calls Destroy.

Never explicitly free a component in one of its own event handlers, nor free a component from the event handler of a component that it owns or contains.

To destroy a form, call its Release method. Release waits for all the form's event handlers and the event handlers of the form's components to finish executing before destroying the form.

Note: A form owns all the controls and non-visual components that are placed on it in design mode. When it is freed, all of these components are automatically freed as well. By default, all forms are owned by the global Application object. When an application terminates, it frees the global Application object, which frees all forms. For objects that are not components, and for components created with a nil owner, be sure to call Free after finishing with the object; otherwise the memory allocated for the object will be lost until the application terminates.

See Also

System::TObject::Destroy (see page 3356)

DestroyComponents

Destroying

System::TObject::Free (see page 3372)

Notification

Owner

Release

Components

8.1.11.4 TShellWindows Methods

The methods of the TShellWindows class are listed here.

Protected Methods

	Name	Description
	Get_Count (see page 499)	This is Get_Count, a member of class TShellWindows.
	InitServerData (see page 499)	Initializes the ServerData property to reflect the type library information of the COM server.
	InvokeEvent (see page 500)	Dispatches an event from the COM server to the appropriate event handler.

Public Methods

	Name	Description
	_NewEnum (see page 498)	This is _NewEnum, a member of class TShellWindows.
	Connect (see page 498)	Establishes a connection to the COM server.
	ConnectTo (see page 498)	This is ConnectTo, a member of class TShellWindows.
	Disconnect (see page 499)	Terminates the connection to the COM server.
	FindWindowSW (see page 499)	This is FindWindowSW, a member of class TShellWindows.
	Item (see page 500)	This is the overview for the Item method overload.
	OnActivated (see page 501)	This is OnActivated, a member of class TShellWindows.
	OnCreated (see page 501)	This is OnCreated, a member of class TShellWindows.
	OnNavigate (see page 501)	This is OnNavigate, a member of class TShellWindows.
	ProcessAttachDetach (see page 502)	This is ProcessAttachDetach, a member of class TShellWindows.
	Register (see page 502)	This is Register, a member of class TShellWindows.
	RegisterPending (see page 502)	This is RegisterPending, a member of class TShellWindows.
	Revoke (see page 502)	This is Revoke, a member of class TShellWindows.

8.1.11.4.1 TShellWindows._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class TShellWindows.

8.1.11.4.2 TShellWindows.Connect Method

Establishes a connection to the COM server.

Pascal

```
procedure Connect; override;
```

C++

```
virtual __fastcall Connect();
```

Description

Call Connect to connect the Connect component to the COM server it represents. If the AutoConnect property is True true, the Connect component automatically calls Connect when the application is first run.

Do not use Connect if the value of ConnectKind is ckAttachToInterface. When attaching to an interface, use the ConnectTo method, which is introduced by descendant classes, instead.

If the value of ConnectKind is ckRemote, the RemoteMachineName property must be set before calling Connect.

Connect introduces Connect as an abstract a pure virtual method. Descendant classes must override this method to let the Connect component connect to a specific type of COM server.

See Also

AutoConnect

ConnectKind

RemoteMachineName

Disconnect

Using Component Wrappers

8.1.11.4.3 TShellWindows.ConnectTo Method

Pascal

```
procedure ConnectTo(svrIntf: IShellWindows);
```

C++

```
__fastcall ConnectTo(IShellWindows svrIntf);
```

Description

This is ConnectTo, a member of class TShellWindows.

8.1.11.4.4 TShellWindows.Disconnect Method

Terminates the connection to the COM server.

Pascal

```
procedure Disconnect; override;
```

C++

```
virtual __fastcall Disconnect();
```

Description

Call Disconnect to terminate a connection formed using the Connect method.

Disconnect introduces Disconnect as an abstract a pure virtual method. Descendant classes must override this method to let the Disconnect component disconnect from a specific type of COM server.

See Also

Connect

Using Component Wrappers

8.1.11.4.5 TShellWindows.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class TShellWindows.

8.1.11.4.6 TShellWindows.Get_Count Method

Pascal

```
function Get_Count: Integer;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class TShellWindows.

8.1.11.4.7 TShellWindows.InitServerData Method

Initializes the ServerData property to reflect the type library information of the COM server.

Pascal

```
procedure InitServerData; override;
```

C++

```
virtual __fastcall InitServerData();
```

Description

The InitServerData constructor calls InitServerData to initialize ServerData with information from the COM server's type library.

InitServerData is an abstract a pure virtual method in InitServerData. Each InitServerData descendant implements this method so that the ServerData property reflects its particular COM server.

See Also

ServerData

Create

8.1.11.4.8 TShellWindows.InvokeEvent Method

Dispatches an event from the COM server to the appropriate event handler.

Pascal

```
procedure InvokeEvent(DispID: TDispID; var Params: TVariantArray); override;
```

C++

```
virtual __fastcall InvokeEvent(TDispID DispID, TVariantArray Params);
```

Description

InvokeEvent descendants use InvokeEvent internally to link their event handlers to events on the COM server.

DispID is the dispatch ID for the appropriate event handler.

Params passes the parameters of the event handler as an array of OleVariants.

As implemented in InvokeEvent, InvokeEvent does nothing, because InvokeEvent has no events. Descendants that surface events from a COM server override this method so that the event sink specified by EventDispatch can pass events on to the appropriate event handlers.

See Also

EventDispatch

8.1.11.4.9 Item Method

This is the overview for the Item method overload.

Overload List

	Name	Description
	TShellWindows.Item () (see page 500)	This is Item, a member of class TShellWindows.
	TShellWindows.Item (OleVariant) (see page 501)	This is Item, a member of class TShellWindows.

8.1.11.4.9.1 TShellWindows.Item Method ()

Pascal

```
function Item: IDispatch; overload;
```

C++

```
__fastcall IDispatch Item();
```

Description

This is Item, a member of class TShellWindows.

8.1.11.4.9.2 TShellWindows.Item Method (OleVariant)

Pascal

```
function Item(index: OleVariant): IDispatch; overload;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class TShellWindows.

8.1.11.4.10 TShellWindows.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool);
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class TShellWindows.

8.1.11.4.11 TShellWindows.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown);
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class TShellWindows.

8.1.11.4.12 TShellWindows.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant);
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class TShellWindows.

8.1.11.4.13 TShellWindows.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool);
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class TShellWindows.

8.1.11.4.14 TShellWindows.Register Method

Pascal

```
procedure Register(const pid: IDispatch; HWND: Integer; swClass: SYSINT; out plCookie:
Integer);
```

C++

```
__fastcall Register(const IDispatch pid, int HWND, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class TShellWindows.

8.1.11.4.15 TShellWindows.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer);
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class TShellWindows.

8.1.11.4.16 TShellWindows.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer);
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class TShellWindows.

8.1.11.5 TShellWindows Properties

The properties of the TShellWindows class are listed here.

Public Properties

	Name	Description
	Count (see page 503)	This is Count, a member of class TShellWindows.
	DefaultInterface (see page 503)	This is DefaultInterface, a member of class TShellWindows.

8.1.11.5.1 TShellWindows.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TShellWindows.

8.1.11.5.2 TShellWindows.DefaultInterface Property

Pascal

```
property DefaultInterface: IShellWindows;
```

C++

```
__property IShellWindows DefaultInterface;
```

Description

This is DefaultInterface, a member of class TShellWindows.

8.1.11.6 TShellWindows Events

The events of the TShellWindows class are listed here.

Published Events

	Name	Description
	OnWindowRegistered (see page 503)	This is OnWindowRegistered, a member of class TShellWindows.
	OnWindowRevoked (see page 504)	This is OnWindowRevoked, a member of class TShellWindows.

8.1.11.6.1 TShellWindows.OnWindowRegistered Event

Pascal

```
property OnWindowRegistered: TShellWindowsWindowRegistered;
```

C++

```
__property TShellWindowsWindowRegistered OnWindowRegistered;
```

Description

This is OnWindowRegistered, a member of class TShellWindows.

8.1.11.6.2 TShellWindows.OnWindowRevoked Event

Pascal

```
property OnWindowRevoked: TShellWindowsWindowRevoked;
```

C++

```
__property TShellWindowsWindowRevoked OnWindowRevoked;
```

Description

This is OnWindowRevoked, a member of class TShellWindows.

8.1.12 TWebBrowser Class

TWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).

Class Hierarchy

Pascal

```
TWebBrowser = class(TOleControl);
```

C++

```
class TWebBrowser : public TOleControl;
```

File

SHDocVw

Description

TWebBrowser wraps the IWebBrowser2 (see page 749) interface from Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL) to allow you to create a customized Web browsing application or to add Internet, file and network browsing, document viewing, and data downloading capabilities to your applications.

Note: Because TWebBrowser wraps the SHDOCVW.DLL, you must have SHDOCVW.DLL installed to use this component. This DLL ships with Internet Explorer 4 and later.

8.1.12.1 TWebBrowser Members

The following tables list the members exposed by TWebBrowser.

Protected Methods

	Name	Description
	CreateControl (see page 508)	This is CreateControl, a member of class TWebBrowser.
	Get_Application (see page 509)	This is Get_Application, a member of class TWebBrowser.
	Get_Container (see page 509)	This is Get_Container, a member of class TWebBrowser.
	Get_Document (see page 509)	This is Get_Document, a member of class TWebBrowser.
	Get_Parent (see page 509)	This is Get_Parent, a member of class TWebBrowser.
	InitControlData (see page 512)	This is InitControlData, a member of class TWebBrowser.

Public Methods

	Name	Description
	ClientToWindow (see page 507)	Not used in TWebBrowser (see page 504).
	ExecWB (see page 508)	This is the overview for the ExecWB method overload.
	GetProperty (see page 510)	Not used in TWebBrowser (see page 504).
	GoBack (see page 510)	Navigates to the previous item in the history list.
	GoForward (see page 510)	Navigates to the next item in the history list.
	GoHome (see page 511)	Navigates to the current start page.
	GoSearch (see page 511)	Navigates to the current search page.
	Navigate (see page 512)	This is the overview for the Navigate method overload.
	Navigate2 (see page 514)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 515)	Not used in TWebBrowser (see page 504).
	QueryStatusWB (see page 516)	Returns information about what commands the Web browser supports.
	Quit (see page 516)	Not used in TWebBrowser (see page 504).
	Refresh (see page 516)	Reloads the current document.
	Refresh2 (see page 517)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 518)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 518)	Stops the current operation.

Public Properties

	Name	Description
R	Application (see page 524)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, the interface for the WebBrowser (see page 837) control.
R	Busy (see page 524)	Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.
R	Container (see page 525)	Provides access to the automation interface of the Web browser's container.
R	ControlInterface (see page 525)	Provides access to the interface for this Web browser component.
R	DefaultInterface (see page 526)	Provides access to the default interface for the component.
R	Document (see page 526)	Provides access to the Automation interface of the active document.
R	FullName (see page 528)	Indicates the fully qualified path name for the executable that hosts the Web Browser component.
R	HWnd (see page 529)	Provides access to the Window handle for the Web browser.
R	LocationName (see page 529)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 530)	Identifies the URL or path of the document or folder currently displayed.
R	Name (see page 531)	Indicates the name for the executable that hosts the Web Browser component.
R	Parent (see page 531)	Provides access to the Automation interface of the Web browser's container.
R	Path (see page 532)	Indicates the name for the executable that hosts the Web Browser component.
R	ReadyState (see page 538)	Indicates the state of the Web browser component.
R	TopLevelContainer (see page 543)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 544)	Indicates the type name of the contained document object.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 545)	Occurs immediately before the Web browser navigates to a new resource.
	OnClientToHostWindow (see page 546)	This is OnClientToHostWindow, a member of class TWebBrowser.
	OnCommandStateChange (see page 546)	Occurs when the ability to execute certain TWebBrowser (see page 504) methods changes.
	OnDocumentComplete (see page 546)	Occurs when the document being navigated to reaches ReadyState_Complete.
	OnDownloadBegin (see page 547)	Occurs when the Web browser starts downloading a document.
	OnDownloadComplete (see page 547)	Occurs when a navigation operation finishes, is halted, or fails.
	OnDragDrop (see page 548)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 551)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 554)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 555)	Occurs when a control receives the input focus.

	OnExit (see page 558)	Occurs when the input focus shifts away from one control to another.
	OnFileDownload (see page 562)	This is OnFileDownload, a member of class TWebBrowser.
	OnFullScreen (see page 562)	Not used in TWebBrowser (see page 504).
	OnMenuBar (see page 562)	Not used in TWebBrowser (see page 504).
	OnNavigateComplete2 (see page 562)	Occurs immediately after the Web browser successfully navigates to a new location.
	OnNavigateError (see page 563)	This is OnNavigateError, a member of class TWebBrowser.
	OnNewWindow2 (see page 563)	Occurs when a new, hidden, non-navigated Web browser window is needed.
	OnPrintTemplateInstantiation (see page 563)	This is OnPrintTemplateInstantiation, a member of class TWebBrowser.
	OnPrintTemplateTeardown (see page 564)	This is OnPrintTemplateTeardown, a member of class TWebBrowser.
	OnPrivacyImpactedStateChange (see page 564)	This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.
	OnProgressChange (see page 564)	Occurs when the progress of a download operation is updated.
	OnPropertyChange (see page 565)	Not used in TWebBrowser (see page 504).
	OnQuit (see page 565)	Not used in TWebBrowser (see page 504).
	OnSetSecureLockIcon (see page 565)	This is OnSetSecureLockIcon, a member of class TWebBrowser.
	OnStartDrag (see page 565)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusBar (see page 566)	Not used in TWebBrowser (see page 504).
	OnStatusTextChange (see page 566)	Not used in TWebBrowser (see page 504).
	OnTheaterMode (see page 567)	Not used in TWebBrowser (see page 504).
	OnTitleChange (see page 567)	Occurs when the title of a document in the WebBrowser (see page 837) control becomes available or changes.
	OnToolBar (see page 567)	Not used in TWebBrowser (see page 504).
	OnUpdatePageStatus (see page 567)	This is OnUpdatePageStatus, a member of class TWebBrowser.
	OnVisible (see page 568)	Occurs when the Web browser window is about to be shown or hidden.
	OnWindowClosing (see page 568)	This is OnWindowClosing, a member of class TWebBrowser.
	OnWindowSetHeight (see page 568)	This is OnWindowSetHeight, a member of class TWebBrowser.
	OnWindowSetLeft (see page 568)	This is OnWindowSetLeft, a member of class TWebBrowser.
	OnWindowSetResizable (see page 569)	This is OnWindowSetResizable, a member of class TWebBrowser.
	OnWindowSetTop (see page 569)	This is OnWindowSetTop, a member of class TWebBrowser.
	OnWindowSetWidth (see page 569)	This is OnWindowSetWidth, a member of class TWebBrowser.

Published Properties

	Name	Description
	AddressBar (see page 520)	Not used in TWebBrowser (see page 504).
	Align (see page 520)	Determines how the control aligns within its container (parent control).
	Anchors (see page 523)	Specifies how the control is anchored to its parent.
	DragCursor (see page 526)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 527)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FullScreen (see page 529)	Not used in TWebBrowser (see page 504).
	MenuBar (see page 530)	Not used in TWebBrowser (see page 504).
	Offline (see page 531)	Indicates whether the Web browser gets its document from the local cache.
	ParentShowHint (see page 532)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 533)	Identifies the pop-up menu associated with the control.
	RegisterAsBrowser (see page 538)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 539)	Specifies whether users can load a document using drag-and-drop.
	Resizable (see page 539)	Not used in TWebBrowser (see page 504).
	ShowHint (see page 539)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Silent (see page 541)	Specifies whether the Web browser can display dialog boxes.
	StatusBar (see page 541)	Not used in TWebBrowser (see page 504).
	StatusText (see page 541)	Not used in TWebBrowser (see page 504).
	TabOrder (see page 542)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 542)	Determines if the user can tab to a control.
	TheaterMode (see page 543)	Not used in TWebBrowser (see page 504).

	ToolBar (see page 543)	Not used in TWebBrowser (see page 504).
	Visible (see page 544)	Specifies whether the Web browser component appears onscreen.

8.1.12.2 TWebBrowser Methods

The methods of the TWebBrowser class are listed here.

Protected Methods

	Name	Description
	CreateControl (see page 508)	This is CreateControl, a member of class TWebBrowser.
	Get_Application (see page 509)	This is Get_Application, a member of class TWebBrowser.
	Get_Container (see page 509)	This is Get_Container, a member of class TWebBrowser.
	Get_Document (see page 509)	This is Get_Document, a member of class TWebBrowser.
	Get_Parent (see page 509)	This is Get_Parent, a member of class TWebBrowser.
	InitControlData (see page 512)	This is InitControlData, a member of class TWebBrowser.

Public Methods

	Name	Description
	ClientToWindow (see page 507)	Not used in TWebBrowser (see page 504).
	ExecWB (see page 508)	This is the overview for the ExecWB method overload.
	GetProperty (see page 510)	Not used in TWebBrowser (see page 504).
	GoBack (see page 510)	Navigates to the previous item in the history list.
	GoForward (see page 510)	Navigates to the next item in the history list.
	GoHome (see page 511)	Navigates to the current start page.
	GoSearch (see page 511)	Navigates to the current search page.
	Navigate (see page 512)	This is the overview for the Navigate method overload.
	Navigate2 (see page 514)	This is the overview for the Navigate2 method overload.
	PutProperty (see page 515)	Not used in TWebBrowser (see page 504).
	QueryStatusWB (see page 516)	Returns information about what commands the Web browser supports.
	Quit (see page 516)	Not used in TWebBrowser (see page 504).
	Refresh (see page 516)	Reloads the current document.
	Refresh2 (see page 517)	This is the overview for the Refresh2 method overload.
	ShowBrowserBar (see page 518)	This is the overview for the ShowBrowserBar method overload.
	Stop (see page 518)	Stops the current operation.

8.1.12.2.1 TWebBrowser.ClientToWindow Method

Not used in TWebBrowser (see page 504).

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. Internet Explorer uses ClientToWindow to convert between client coordinates and window coordinates.

8.1.12.2.2 TWebBrowser.CreateControl Method

Pascal

```
procedure CreateControl;
```

C++

```
__fastcall CreateControl();
```

Description

This is CreateControl, a member of class TWebBrowser.

8.1.12.2.3 ExecWB Method

This is the overview for the ExecWB method overload.

Overload List

	Name	Description
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT) (see page 508)	Executes a command.
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant) (see page 508)	Executes a command.
	TWebBrowser.ExecWB (OLECMDID, OLECMDEXECOPT, OleVariant, OleVariant) (see page 508)	Executes a command.

8.1.12.2.3.1 TWebBrowser.ExecWB Method (OLECMDID, OLECMDEXECOPT)

Executes a command.

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT); overload;
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant); overload;
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); overload;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt);
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn);
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

ExecWB lets you use the IOleCommandTarget interface to pass a command to the Web browser control. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command to execute. The Web Browser does not support all of the commands for which Microsoft providesconstants. Use the QueryStatusWB (see page 516) method to determine what commands can be used.

cmdexecopt modifies how the command is executed.

pvaIn specifies a Variant (see page 3421) array that contains any input arguments. This value can be omitted if there are not input arguments.

pvaOutspecifies a Variant (see page 3421) that receives the results of the command, if any. This value can be omitted if there are no results.

See Also

QueryStatusWB (see page 516)

DoObjectVerb

GetObjectVerbs

8.1.12.2.4 TWebBrowser.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TWebBrowser.

8.1.12.2.5 TWebBrowser.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TWebBrowser.

8.1.12.2.6 TWebBrowser.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TWebBrowser.

8.1.12.2.7 TWebBrowser.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TWebBrowser.

8.1.12.2.8 TWebBrowser.GetProperty Method

Not used in TWebBrowser (see page 504).

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. Internet Explorer uses GetProperty to return a property value.

8.1.12.2.9 TWebBrowser.GoBack Method

Navigates to the previous item in the history list.

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

Call GoBack to return to the last document in the history list. GoBack has no effect unless the history list contains additional documents.

Note: The Navigate (see page 512) or Navigate2 (see page 514) command can change the history list, depending on the specified flags.

See Also

OnCommandStateChange (see page 546)

GoForward (see page 510)

GoHome (see page 511)

GoSearch (see page 511)

Navigate (see page 512)

Navigate2 (see page 514)

8.1.12.2.10 TWebBrowser.GoForward Method

Navigates to the next item in the history list.

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

Call GoForward to return to the next document in the history list. GoForward reverses the effect of the last GoBack (see page

510) command.

See Also

OnCommandStateChange (see page 546)

GoBack (see page 510)

GoHome (see page 511)

GoSearch (see page 511)

8.1.12.2.11 TWebBrowser.GoHome Method

Navigates to the current start page.

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

Call GoHome to navigate to the registered home or start page. The URL for this page is specified in the Registry at

`HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main`

under the name Startpage. You can change this value using the TRegistry or TRegistryIniFile object, or using Navigation tab of the Internet Explorer's Options dialog.

See Also

GoBack (see page 510)

GoForward (see page 510)

GoSearch (see page 511)

TRegistry

TRegistryIniFile

8.1.12.2.12 TWebBrowser.GoSearch Method

Navigates to the current search page.

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

Call GoSearch to navigate to the registered search page. The URL for this page is specified in the Registry at

`HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main`

under the name SearchPage. You can change this value using the TRegistry or TRegistryIniFile object, or using Navigation tab of the Internet Explorer's Options dialog.

See Also

GoBack (see page 510)

GoForward (see page 510)

GoHome (see page 511)

TRegistry

TRegistryIniFile

8.1.12.2.13 TWebBrowser.InitControlData Method

Pascal

```
procedure InitControlData; override;
```

C++

```
virtual __fastcall InitControlData();
```

Description

This is InitControlData, a member of class TWebBrowser.

8.1.12.2.14 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TWebBrowser.Navigate (WideString) (see page 512)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant) (see page 512)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant) (see page 512)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 512)	Navigates to a resource identified by a path name or URL.
	TWebBrowser.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 512)	Navigates to a resource identified by a path name or URL.

8.1.12.2.14.1 TWebBrowser.Navigate Method (WideString)

Navigates to a resource identified by a path name or URL.

Pascal

```
procedure Navigate(const URL: WideString); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

Use Navigate to locate and download a specific resource. Navigate can send an HTTP message to a specified URL and display the results or simply display the results of a specified file.

URL specifies the UNC path name of a file or the Uniform Resource Locator of an Internet resource that the Web browser should display.

Flags is a set of values that specify whether to add the resource to the history list, whether to read from or write to the cache, and whether to display the resource in a new window. It can be a sum of zero or more of the following:

Constant	Value	Meaning
navOpenInNewWindow (see page 854)	1	Open the resource or file in a new window.
navNoHistory (see page 853)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 854)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 854)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 853)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate to generate an HTTP POST message. If PostData is NULL, Navigate generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

Note: To navigate to a location that can't be expressed as an URL (for example, to an item identifier list), use the Navigate2 (see page 514) method instead.

See Also

Navigate2 (see page 514)

GoBack (see page 510)

GoForward (see page 510)

GoHome (see page 511)

GoSearch (see page 511)

OnBeforeNavigate2 (see page 545)

OnNavigateComplete2 (see page 562)

TWebRequest

8.1.12.2.15 Navigate2 Method

This is the overview for the Navigate2 method overload.

Overload List

	Name	Description
	TWebBrowser.Navigate2 (OleVariant) (see page 514)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant) (see page 514)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant) (see page 514)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant) (see page 514)	Navigates to a specified resource.
	TWebBrowser.Navigate2 (OleVariant, OleVariant, OleVariant, OleVariant, OleVariant) (see page 514)	Navigates to a specified resource.

8.1.12.2.15.1 TWebBrowser.Navigate2 Method (OleVariant)

Navigates to a specified resource.

Pascal

```
procedure Navigate2(var URL: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
procedure Navigate2(var URL: OleVariant; var Flags: OleVariant; var TargetFrameName:
OleVariant; var PostData: OleVariant; var Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate2(OleVariant URL);
__fastcall Navigate2(OleVariant URL, OleVariant Flags);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData);
__fastcall Navigate2(OleVariant URL, OleVariant Flags, OleVariant TargetFrameName, OleVariant
PostData, OleVariant Headers);
```

Description

Use Navigate2 to locate and download a specific resource. Navigate (see page 512) can send an HTTP message to a specified URL and display the results, display the results of a specified file, or navigate to a resource that can't be expressed as an URL such as an item identifier list.

URL specifies the UNC path name of a file, the Uniform Resource Locator of an Internet resource, or a pointer to an item identifier list (PIDL).

Flags is a set of values that specify whether to add the resource to the history list, whether to read from or write to the cache, and whether to display the resource in a new window. It can be a sum of zero or more of the following:

Constant	Value	Meaning
navOpenInNewWindow (see page 854)	1	Open the resource or file in a new window.
navNoHistory (see page 853)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 854)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 854)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 853)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate (see page 512) to generate an HTTP POST message. If PostData is NULL, Navigate (see page 512) generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

See Also

Navigate (see page 512)

GoBack (see page 510)

GoForward (see page 510)

GoHome (see page 511)

GoSearch (see page 511)

OnBeforeNavigate2 (see page 545)

OnNavigateComplete2 (see page 562)

TWebRequest

8.1.12.2.16 TWebBrowser.PutProperty Method

Not used in TWebBrowser (see page 504).

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. Internet Explorer uses PutProperty to set a property value.

8.1.12.2.17 TWebBrowser.QueryStatusWB Method

Returns information about what commands the Web browser supports.

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

Use QueryStatusWB to determine what commands can be executed using the ExecWB (see page 508) method. QueryStatusWB uses the QueryStatus method of the IOleCommandTarget interface to obtain information about the status of commands. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command about which you want information.

QueryStatusWB returns a value made by adding together flags that indicate the status of the command.

See Also

ExecWB (see page 508)

8.1.12.2.18 TWebBrowser.Quit Method

Not used in TWebBrowser (see page 504).

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

Do not use this method. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. Internet Explorer responds to Quit by shutting down.

8.1.12.2.19 TWebBrowser.Refresh Method

Reloads the current document.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to reload the current document. This is useful when the Web browser is displaying the resource at an URL that may have changed.

Refresh always sends a pragma:nocache header, telling the server not to return a cached copy. This can cause problems with some servers. To prevent the pragma:nocache header, or to refresh only expired pages, use the Refresh2 (see page 517) method instead.

See Also

Refresh2 (see page 517)

Document (see page 526)

8.1.12.2.20 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TWebBrowser.Refresh2 () (see page 517)	Reloads the current document if necessary.
	TWebBrowser.Refresh2 (OleVariant) (see page 517)	Reloads the current document if necessary.

8.1.12.2.20.1 TWebBrowser.Refresh2 Method ()

Reloads the current document if necessary.

Pascal

```
procedure Refresh2; overload;
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2();
__fastcall Refresh2(OleVariant Level);
```

Description

Call Refresh2 to reload the current document. Unlike the Refresh (see page 516) method, Refresh2 lets you specify what level of refresh to perform.

Level indicates what type of information is refreshed. The following table lists the possible values:

Constant	Value	Meaning
REFRESH_NORMAL (see page 878)	0	Perform a lightweight refresh that does not include the pragma:nocache header. The pragma:nocache header tells the server not to return a cached copy. This can cause problems with some servers.
REFRESH_IFEXPIRED (see page 878)	1	Only refresh if the page has expired. Do not include the pragma:nocache header.
REFRESH_CONTINUE (see page 877)	2	For internal use only.
REFRESH_COMPLETELY (see page 877)	3	Perform a full refresh, including the pragma:nocache header. Using this option is the same as calling the Refresh (see page 516) method.

When Level is omitted, a value of REFRESH_COMPLETELY (see page 877) is assumed.

See Also

Refresh (see page 516)

8.1.12.2.21 ShowBrowserBar Method

This is the overview for the ShowBrowserBar method overload.

Overload List

	Name	Description
	TWebBrowser.ShowBrowserBar (OleVariant) (see page 518)	Not used in TWebBrowser (see page 504)
	TWebBrowser.ShowBrowserBar (OleVariant, OleVariant) (see page 518)	Not used in TWebBrowser (see page 504)
	TWebBrowser.ShowBrowserBar (OleVariant, OleVariant, OleVariant) (see page 518)	Not used in TWebBrowser (see page 504)

8.1.12.2.21.1 TWebBrowser.ShowBrowserBar Method (OleVariant)

Not used in TWebBrowser (see page 504)

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant); overload;
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant); overload;
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); overload;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid);
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow);
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

Do not use this method. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. Internet Explorer uses ShowBrowserBar to show or hide a specified browser bar.

8.1.12.2.22 TWebBrowser.Stop Method

Stops the current operation.

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

Stop cancels any pending navigation or download operation and terminates any dynamic page elements, such as background sounds and animations.

See Also

OnCommandStateChange (see page 546)

GoForward (see page 510)

GoHome (see page 511)

GoSearch (see page 511)

GoBack (see page 510)

Navigate (see page 512)

Navigate2 (see page 514)

8.1.12.3 TWebBrowser Properties

The properties of the TWebBrowser class are listed here.

Public Properties

	Name	Description
	Application (see page 524)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, the interface for the WebBrowser (see page 837) control.
	Busy (see page 524)	Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.
	Container (see page 525)	Provides access to the automation interface of the Web browser's container.
	ControlInterface (see page 525)	Provides access to the interface for this Web browser component.
	DefaultInterface (see page 526)	Provides access to the default interface for the component.
	Document (see page 526)	Provides access to the Automation interface of the active document.
	FullName (see page 528)	Indicates the fully qualified path name for the executable that hosts the Web Browser component.
	HWnd (see page 529)	Provides access to the Window handle for the Web browser.
	LocationName (see page 529)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
	LocationURL (see page 530)	Identifies the URL or path of the document or folder currently displayed.
	Name (see page 531)	Indicates the name for the executable that hosts the Web Browser component.
	Parent (see page 531)	Provides access to the Automation interface of the Web browser's container.
	Path (see page 532)	Indicates the name for the executable that hosts the Web Browser component.
	ReadyState (see page 538)	Indicates the state of the Web browser component.
	TopLevelContainer (see page 543)	Indicates whether the Web browser is a top-level Automation object.
	type_ (see page 544)	Indicates the type name of the contained document object.

Published Properties

	Name	Description
	AddressBar (see page 520)	Not used in TWebBrowser (see page 504).
	Align (see page 520)	Determines how the control aligns within its container (parent control).
	Anchors (see page 523)	Specifies how the control is anchored to its parent.
	DragCursor (see page 526)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 527)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	FullScreen (see page 529)	Not used in TWebBrowser (see page 504).
	MenuBar (see page 530)	Not used in TWebBrowser (see page 504).
	Offline (see page 531)	Indicates whether the Web browser gets its document from the local cache.
	ParentShowHint (see page 532)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 533)	Identifies the pop-up menu associated with the control.
	RegisterAsBrowser (see page 538)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 539)	Specifies whether users can load a document using drag-and-drop.
	Resizable (see page 539)	Not used in TWebBrowser (see page 504).
	ShowHint (see page 539)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Silent (see page 541)	Specifies whether the Web browser can display dialog boxes.
	StatusBar (see page 541)	Not used in TWebBrowser (see page 504).
	StatusText (see page 541)	Not used in TWebBrowser (see page 504).
	TabOrder (see page 542)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 542)	Determines if the user can tab to a control.

	TheaterMode (see page 543)	Not used in TWebBrowser (see page 504).
	ToolBar (see page 543)	Not used in TWebBrowser (see page 504).
	Visible (see page 544)	Specifies whether the Web browser component appears onscreen.

8.1.12.3.1 TWebBrowser.AddressBar Property

Not used in TWebBrowser (see page 504).

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

Do not use AddressBar. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. AddressBar shows or hides the URL address bar on Internet Explorer. TWebBrowser (see page 504) does not have an address bar.

8.1.12.3.2 TWebBrowser.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
```

```
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

8.1.12.3.3 TWebBrowser.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

8.1.12.3.4 TWebBrowser.Application Property

Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, the interface for the WebBrowser (see page 837) control.

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

Use Application to access the interface for the application containing the WebBrowser (see page 837) control. If the application does not have an Automation interface, or if the interface is not accessible, Application provides access to the Automation interface of the Web browser control.

See Also

Container (see page 525)

Parent (see page 531)

ControlInterface (see page 525)

8.1.12.3.5 TWebBrowser.Busy Property

Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

Read (see page 3634) Busy to determine whether the Web browser component is currently navigating to a new URL or

downloading a document. If Busy is true, you can interrupt the current operation using the Stop (see page 518) method before starting a new operation.

Note: The Web browser receives events at the beginning and end of each operation, as well as at discrete points during a download operation.

See Also

ReadyState (see page 538)

OnDocumentComplete (see page 546)

OnBeforeNavigate2 (see page 545)

OnNavigateComplete2 (see page 562)

OnDownloadComplete (see page 547)

OnDownloadBegin (see page 547)

OnProgressChange (see page 564)

Stop (see page 518)

8.1.12.3.6 TWebBrowser.Container Property

Provides access to the automation interface of the Web browser's container.

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

Use Container to access the interface for the Web browser's parent or container. If the Web browser does not have a container with an Automation interface, Container is nil.

See Also

Application (see page 524)

Parent (see page 531)

ControlInterface (see page 525)

TopLevelContainer (see page 543)

8.1.12.3.7 TWebBrowser.ControlInterface Property

Provides access to the interface for this Web browser component.

Pascal

```
property ControlInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 ControlInterface;
```

Description

Use ControlInterface to access the interface for the WebBrowser (see page 837) component. This interface lets an Automation

controller access the properties, methods, and events of the Web browser.

See Also

Application (see page 524)

Document (see page 526)

Parent (see page 531)

DefaultInterface (see page 526)

8.1.12.3.8 TWebBrowser.DefaultInterface Property

Provides access to the default interface for the component.

Pascal

```
property DefaultInterface: IWebBrowser2;
```

C++

```
__property IWebBrowser2 DefaultInterface;
```

Description

Use DefaultInterface to access the default interface for the component. As implemented in TWebBrowser (see page 504), this is the same as the ControlInterface (see page 525) property.

See Also

ControlInterface (see page 525)

8.1.12.3.9 TWebBrowser.Document Property

Provides access to the Automation interface of the active document.

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

Use Document to access an IDispatch (see page 3494) interface for manipulating the contents of the HTML document through the COM object model. If there is no active document, Document is nil. That is, you must first load a document using the Navigate (see page 512) or Navigate2 (see page 514) method before you can manipulate it using the Document interface.

For more information about this interface, see the document object section of Microsoft's Dynamic HTML reference.

See Also

ControlInterface (see page 525)

Navigate (see page 512)

Navigate2 (see page 514)

8.1.12.3.10 TWebBrowser.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

8.1.12.3.11 TWebBrowser.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}
```

```
void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

8.1.12.3.12 TWebBrowser.FullName Property

Indicates the fully qualified path name for the executable that hosts the Web Browser component.

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

Read (see page 3634) FullName to obtain the name of the executable as a WideString (see page 3467). Unlike the Name (see page 531) property, FullName specifies a fully qualified path name.

See Also

Application (see page 524)

ExeName

Name (see page 531)

Path (see page 532)

8.1.12.3.13 TWebBrowser.FullScreen Property

Not used in TWebBrowser (see page 504).

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

Do not use FullScreen. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. FullScreen determines whether Internet Explorer is maximized.

8.1.12.3.14 TWebBrowser.HWnd Property

Provides access to the Window handle for the Web browser.

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

Use HWnd for Windows API calls that require a window handle. Use HWnd instead of the Handle property when working with TWebBrowser (see page 504) controls.

See Also

Handle

8.1.12.3.15 TWebBrowser.LocationName Property

Identifies the short (UI-friendly) name of the URL or file currently displayed.

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

Read (see page 3634) LocationName to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName is the title of that page. If the resource is a folder or file on the local network or on a disk, LocationName is the full UNC name of the folder or file.

See Also

LocationURL (see page 530)

OnTitleChange (see page 567)

8.1.12.3.16 TWebBrowser.LocationURL Property

Identifies the URL or path of the document or folder currently displayed.

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

Read (see page 3634) LocationURL to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName (see page 529) is its URL. If the resource is a folder or file on the local network or on a disk, LocationName (see page 529) is the full UNC name of the folder or file.

Note: LocationURL is a read-only property. To direct the Web browser to display the resource at a specific URL, folder, or file, use the Navigate (see page 512) or Navigate2 (see page 514) method.

See Also

LocationName (see page 529)

Navigate (see page 512)

Navigate2 (see page 514)

8.1.12.3.17 TWebBrowser.MenuBar Property

Not used in TWebBrowser (see page 504).

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

Do not use MenuBar. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. MenuBar shows or hides the menu bar in Internet Explorer. TWebBrowser (see page 504) does not have a menu bar.

8.1.12.3.18 TWebBrowser.Name Property

Indicates the name for the executable that hosts the Web Browser component.

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

Read (see page 3634) Name to obtain the name of the executable that hosts the Web browser. Unlike the FullName (see page 528) property, Name does not include the path.

See Also

Application (see page 524)

FullName (see page 528)

Path (see page 532)

8.1.12.3.19 TWebBrowser.Offline Property

Indicates whether the Web browser gets its document from the local cache.

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

Set Offline to true to work offline with the Web browser. When the Web browser is offline, it can't read HTML pages using an Internet connection. Instead, you must specify a source document as a UNC filename.

Note: You can use Internet Explorer to save HTML pages to a local cache so that they are available for offline work.

See Also

Navigate (see page 512)

Navigate2 (see page 514)

8.1.12.3.20 TWebBrowser.Parent Property

Provides access to the Automation interface of the Web browser's container.

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

Use Parent to access the interface for the Web browser's parent or container. If the Web browser does not have a container with an Automation interface, the value of Parent is the same as ControlInterface (see page 525).

Use the TopLevelContainer (see page 543) property to determine which interface is the value of Parent.

See Also

Container (see page 525)

ControlInterface (see page 525)

Application (see page 524)

TopLevelContainer (see page 543)

8.1.12.3.21 TWebBrowser.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

8.1.12.3.22 TWebBrowser.Path Property

Indicates the name for the executable that hosts the Web Browser component.

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

Read (see page 3634) Path to obtain the path of the executable that hosts the Web browser.

See Also

FullName (see page 528)

Name (see page 531)

8.1.12.3.23 TWebBrowser.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
      TNotifyEvent();
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
          {
              case 0:
              {
                  ShowMessage("first item clicked");
                  break;
              }
              case 1:
              {
                  ShowMessage("second item clicked");
                  break;
              }
              case 2:
              {
                  ShowMessage("third item clicked");
```

```
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
```

```
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

8.1.12.3.24 TWebBrowser.ReadyState Property

Indicates the state of the Web browser component.

Pascal

```
property ReadyState: TOleEnum;
```

C++

```
__property TOleEnum ReadyState;
```

Description

Read (see page 3634) ReadyState to obtain state information specific to the Web browser control. Use this information to avoid attempting to use the Web browser control when it is not ready to perform an action.

The following table lists the possible values:

Constant	Value	Meaning
READYSTATE_UNINITIALIZED (see page 877)	0	Default initialization state. The Control has not begun loading any properties.
READYSTATE_LOADING (see page 876)	1	The control is loading its properties.
READYSTATE_LOADED (see page 876)	2	The control has initialized its properties but is not yet interactive.
READYSTATE_INTERACTIVE (see page 876)	3	The control is interactive, but not all of its data is available. (It is downloading a document or frame).
READYSTATE_COMPLETE (see page 875)	4	The control is interactive and all of its data is available. (The current document is fully downloaded).

See Also

ComponentState

ControlState

Busy (see page 524)

8.1.12.3.25 TWebBrowser.RegisterAsBrowser Property

Registers the control as a top-level browser for target name resolution.

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

Set RegisterAsBrowser to true when using the Web browser control to implement a top-level browser.

8.1.12.3.26 TWebBrowser.RegisterAsDropTarget Property

Specifies whether users can load a document using drag-and-drop.

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

Set RegisterAsDropTarget to true to allow users to drop HTML documents onto the Web browser. When RegisterAsDropTarget is true, users can load a document by dragging it onto the Web browser. When RegisterAsDropTarget is false, dragging a document onto the Web browser has no effect.

See Also

Navigate (see page 512)

Navigate2 (see page 514)

8.1.12.3.27 TWebBrowser.Resizable Property

Not used in TWebBrowser (see page 504).

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

Do not use Resizable. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. Resizable determines whether the Internet Explorer window can be resized.

8.1.12.3.28 TWebBrowser.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

8.1.12.3.29 TWebBrowser.Silent Property

Specifies whether the Web browser can display dialog boxes.

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

Set Silent to true to prevent the Web browser control from displaying dialog boxes.

8.1.12.3.30 TWebBrowser.StatusBar Property

Not used in TWebBrowser (see page 504).

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

Do not use StatusBar. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. StatusBar shows or hides the status bar in Internet Explorer. TWebBrowser (see page 504) does not have a status bar.

8.1.12.3.31 TWebBrowser.StatusText Property

Not used in TWebBrowser (see page 504).

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

Do not use StatusText. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. StatusText specifies the text that appears on the status bar in Internet Explorer. TWebBrowser (see page 504) does not have a status bar.

8.1.12.3.32 TWebBrowser.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

8.1.12.3.33 TWebBrowser.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

8.1.12.3.34 TWebBrowser.TheaterMode Property

Not used in TWebBrowser (see page 504).

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

Do not use TheaterMode. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. TheaterMode changes the display of Internet Explorer to use theater or normal mode.

8.1.12.3.35 TWebBrowser.ToolBar Property

Not used in TWebBrowser (see page 504).

Pascal

```
property ToolBar: Integer;
```

C++

```
__property int ToolBar;
```

Description

Do not use ToolBar. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. ToolBar determines which toolbar is displayed in Internet Explorer. TWebBrowser (see page 504) does not have any tool bars.

8.1.12.3.36 TWebBrowser.TopLevelContainer Property

Indicates whether the Web browser is a top-level Automation object.

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

Read (see page 3634) TopLevelContainer to determine whether the Web browser has a parent with an Automation interface. When TopLevelContainer is false, the Container (see page 525) and Parent (see page 531) properties provide access to the parent's interface. When TopLevelContainer is true, the value of Container (see page 525) is nil and the value of Parent (see page 531) is the Web browser's own interface.

See Also

Container (see page 525)

Parent (see page 531)

8.1.12.3.37 TWebBrowser.type_ Property

Indicates the type name of the contained document object.

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

Read (see page 3634) type_ to determine the type of the Automation object whose interface is represented by the Document (see page 526) property.

See Also

Container (see page 525)

Parent (see page 531)

Document (see page 526)

8.1.12.3.38 TWebBrowser.Visible Property

Specifies whether the Web browser component appears onscreen.

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

Use Visible to show or hide the Web browser control.

See Also

Enabled

8.1.12.4 TWebBrowser Events

The events of the TWebBrowser class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate2 (see page 545)	Occurs immediately before the Web browser navigates to a new resource.
	OnClientToHostWindow (see page 546)	This is OnClientToHostWindow, a member of class TWebBrowser.
	OnCommandStateChange (see page 546)	Occurs when the ability to execute certain TWebBrowser (see page 504) methods changes.
	OnDocumentComplete (see page 546)	Occurs when the document being navigated to reaches ReadyState_Complete.
	OnDownloadBegin (see page 547)	Occurs when the Web browser starts downloading a document.
	OnDownloadComplete (see page 547)	Occurs when a navigation operation finishes, is halted, or fails.
	OnDragDrop (see page 548)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 551)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 554)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

	OnEnter (see page 555)	Occurs when a control receives the input focus.
	OnExit (see page 558)	Occurs when the input focus shifts away from one control to another.
	OnFileDownload (see page 562)	This is OnFileDownload, a member of class TWebBrowser.
	OnFullScreen (see page 562)	Not used in TWebBrowser (see page 504).
	OnMenuBar (see page 562)	Not used in TWebBrowser (see page 504).
	OnNavigateComplete2 (see page 562)	Occurs immediately after the Web browser successfully navigates to a new location.
	OnNavigateError (see page 563)	This is OnNavigateError, a member of class TWebBrowser.
	OnNewWindow2 (see page 563)	Occurs when a new, hidden, non-navigated Web browser window is needed.
	OnPrintTemplateInstantiation (see page 563)	This is OnPrintTemplateInstantiation, a member of class TWebBrowser.
	OnPrintTemplateTeardown (see page 564)	This is OnPrintTemplateTeardown, a member of class TWebBrowser.
	OnPrivacyImpactedStateChange (see page 564)	This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.
	OnProgressChange (see page 564)	Occurs when the progress of a download operation is updated.
	OnPropertyChange (see page 565)	Not used in TWebBrowser (see page 504).
	OnQuit (see page 565)	Not used in TWebBrowser (see page 504).
	OnSetSecureLockIcon (see page 565)	This is OnSetSecureLockIcon, a member of class TWebBrowser.
	OnStartDrag (see page 565)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusBar (see page 566)	Not used in TWebBrowser (see page 504).
	OnStatusTextChange (see page 566)	Not used in TWebBrowser (see page 504).
	OnTheaterMode (see page 567)	Not used in TWebBrowser (see page 504).
	OnTitleChange (see page 567)	Occurs when the title of a document in the WebBrowser (see page 837) control becomes available or changes.
	OnToolBar (see page 567)	Not used in TWebBrowser (see page 504).
	OnUpdatePageStatus (see page 567)	This is OnUpdatePageStatus, a member of class TWebBrowser.
	OnVisible (see page 568)	Occurs when the Web browser window is about to be shown or hidden.
	OnWindowClosing (see page 568)	This is OnWindowClosing, a member of class TWebBrowser.
	OnWindowSetHeight (see page 568)	This is OnWindowSetHeight, a member of class TWebBrowser.
	OnWindowSetLeft (see page 568)	This is OnWindowSetLeft, a member of class TWebBrowser.
	OnWindowSetResizable (see page 569)	This is OnWindowSetResizable, a member of class TWebBrowser.
	OnWindowSetTop (see page 569)	This is OnWindowSetTop, a member of class TWebBrowser.
	OnWindowSetWidth (see page 569)	This is OnWindowSetWidth, a member of class TWebBrowser.

8.1.12.4.1 TWebBrowser.OnBeforeNavigate2 Event

Occurs immediately before the Web browser navigates to a new resource.

Pascal

```
property OnBeforeNavigate2: TWebBrowserBeforeNavigate2;
```

C++

```
__property TWebBrowserBeforeNavigate2 OnBeforeNavigate2;
```

Description

Write (see page 3707) an OnBeforeNavigate2 event handler to redirect or cancel a change to a different URL. This event may occur as the result of a call to the Navigate (see page 512) or Navigate2 (see page 514) method, or the user clicking a link.

See Also

Navigate (see page 512)

Navigate2 (see page 514)

OnNavigateComplete2 (see page 562)

TWebRequest

8.1.12.4.2 TWebBrowser.OnClientToHostWindow Event

Pascal

```
property OnClientToHostWindow: TWebBrowserClientToHostWindow;
```

C++

```
__property TWebBrowserClientToHostWindow OnClientToHostWindow;
```

Description

This is OnClientToHostWindow, a member of class TWebBrowser.

8.1.12.4.3 TWebBrowser.OnCommandStateChange Event

Occurs when the ability to execute certain TWebBrowser (see page 504) methods changes.

Pascal

```
property OnCommandStateChange: TWebBrowserCommandStateChange;
```

C++

```
__property TWebBrowserCommandStateChange OnCommandStateChange;
```

Description

Write (see page 3707) an OnCommandStateChange to update any controls in the application whose state depends on the ability to execute the Web browser's methods.

See Also

Stop (see page 518)

GoBack (see page 510)

GoForward (see page 510)

Busy (see page 524)

QueryStatusWB (see page 516)

8.1.12.4.4 TWebBrowser.OnDocumentComplete Event

Occurs when the document being navigated to reaches ReadyState_Complete.

Pascal

```
property OnDocumentComplete: TWebBrowserDocumentComplete;
```

C++

```
__property TWebBrowserDocumentComplete OnDocumentComplete;
```

Description

Write (see page 3707) an OnDocumentComplete event handler to take specific action when a frame or document is fully loaded into the Web browser .For a document without frames, this event occurs once when the document finishes loading. On a document containing multiple frames, this event occurs once for each frame. When the multiple-frame document finishes loading, the Web browser fires the event one final time.

See Also

ReadyState (see page 538)

OnDownloadBegin (see page 547)

OnDownloadComplete (see page 547)

Busy (see page 524)

Navigate (see page 512)

Navigate2 (see page 514)

OnBeforeNavigate2 (see page 545)

8.1.12.4.5 TWebBrowser.OnDownloadBegin Event

Occurs when the Web browser starts downloading a document.

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

Write (see page 3707) an OnDownloadBegin event handler to take specific action after the Web browser has located a document and immediately before it starts downloading the document. For example, use the OnDownloadBegin event to launch an animation control the represents downloading or a progress bar that is updated by an OnProgressChange (see page 564) event handler. The control can then be stopped in an OnDownloadComplete (see page 547) event handler.

Note: To take specific action when the Web browser looks up the resource, rather than when it begins downloading, use the OnBeforeNavigate2 (see page 545) event. OnDownloadBegin occurs shortly after OnBeforeNavigate2 (see page 545).

See Also

OnDocumentComplete (see page 546)

OnDownloadComplete (see page 547)

Navigate (see page 512)

Navigate2 (see page 514)

OnBeforeNavigate2 (see page 545)

TAnimate

OnNavigateComplete2 (see page 562)

OnProgressChange (see page 564)

8.1.12.4.6 TWebBrowser.OnDownloadComplete Event

Occurs when a navigation operation finishes, is halted, or fails.

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

Write (see page 3707) an OnDownloadComplete event handler to take specific action after the Web browser stops a downloading operation. For example, use the OnDownloadComplete event to stop an download indication that is started in an OnDownloadBegin (see page 547) event handler.

Note: Unlike the OnNavigateComplete2 (see page 562) event, OnDownloadComplete occurs even if the Web browser does not successfully navigate to an URL.

See Also

OnDocumentComplete (see page 546)

OnDownloadBegin (see page 547)

Navigate (see page 512)

Navigate2 (see page 514)

OnNavigateComplete2 (see page 562)

8.1.12.4.7 TWebBrowser.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
```

```
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
```

```
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

8.1.12.4.8 TWebBrowser.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}
```

```
void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
```

```
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

8.1.12.4.9 TWebBrowser.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
```

```
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

8.1.12.4.10 TWebBrowser.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
  applied to the active control. Clicking the button sets the
  ActiveControl to the button.  That is why we need to save
  the ActiveControl in a shared OnEnter procedure.  In the rich
  edit, only the text selected converts.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
     FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
     FontDialog1.Execute;
  end;

  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    myActiveControl := ActiveControl;
  end;

  procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
  begin
    if myActiveControl is TEdit then
      with myActiveControl as TEdit do
         Font.Assign(TFontDialog(Sender).Font)
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

8.1.12.4.11 TWebBrowser.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
```

```
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
```

```
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

8.1.12.4.12 TWebBrowser.OnFileDownload Event

Pascal

```
property OnFileDownload: TWebBrowserFileDownload;
```

C++

```
__property TWebBrowserFileDownload OnFileDownload;
```

Description

This is OnFileDownload, a member of class TWebBrowser.

8.1.12.4.13 TWebBrowser.OnFullScreen Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnFullScreen: TWebBrowserOnFullScreen;
```

C++

```
__property TWebBrowserOnFullScreen OnFullScreen;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnFullScreen occurs with Internet Explorer when the application is maximized or restored from a maximized state.

8.1.12.4.14 TWebBrowser.OnMenuBar Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnMenuBar: TWebBrowserOnMenuBar;
```

C++

```
__property TWebBrowserOnMenuBar OnMenuBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnMenuBar occurs with Internet Explorer when the menu bar is hidden or shown.

8.1.12.4.15 TWebBrowser.OnNavigateComplete2 Event

Occurs immediately after the Web browser successfully navigates to a new location.

Pascal

```
property OnNavigateComplete2: TWebBrowserNavigateComplete2;
```

C++

```
__property TWebBrowserNavigateComplete2 OnNavigateComplete2;
```

Description

Write (see page 3707) an OnNavigateComplete2 event handler to take specific action when the Web browser successfully navigates to a new resource. The event can occur before the document is fully downloaded, but when it occurs at least part of

the document must be received and a viewer for the document created.

Note: Unlike the OnDownloadComplete (see page 547) event, OnNavigateComplete2 does not occur if the operation is not successful.

See Also

OnDocumentComplete (see page 546)

OnDownloadBegin (see page 547)

Navigate (see page 512)

Navigate2 (see page 514)

OnDownloadComplete (see page 547)

OnBeforeNavigate2 (see page 545)

8.1.12.4.16 TWebBrowser.OnNavigateError Event

Pascal

```
property OnNavigateError: TWebBrowserNavigateError;
```

C++

```
__property TWebBrowserNavigateError OnNavigateError;
```

Description

This is OnNavigateError, a member of class TWebBrowser.

8.1.12.4.17 TWebBrowser.OnNewWindow2 Event

Occurs when a new, hidden, non-navigated Web browser window is needed.

Pascal

```
property OnNewWindow2: TWebBrowserNewWindow2;
```

C++

```
__property TWebBrowserNewWindow2 OnNewWindow2;
```

Description

Write (see page 3707) an OnNewWindow2 to take specific action immediately before the Web browser creates a new window for displaying a resource. This window may be needed because the user shift-clicked on a link, the user right-clicked on a link and chose "open in new window", the frame for the target URL does not yet exist, or the Navigate (see page 512) or Navigate2 (see page 514) requested that the target document appear in a new window.

See Also

OnBeforeNavigate2 (see page 545)

8.1.12.4.18 TWebBrowser.OnPrintTemplateInstantiation Event

Pascal

```
property OnPrintTemplateInstantiation: TWebBrowserPrintTemplateInstantiation;
```

C++

```
__property TWebBrowserPrintTemplateInstantiation OnPrintTemplateInstantiation;
```

Description

This is OnPrintTemplateInstantiation, a member of class TWebBrowser.

8.1.12.4.19 TWebBrowser.OnPrintTemplateTeardown Event

Pascal

```
property OnPrintTemplateTeardown: TWebBrowserPrintTemplateTeardown;
```

C++

```
__property TWebBrowserPrintTemplateTeardown OnPrintTemplateTeardown;
```

Description

This is OnPrintTemplateTeardown, a member of class TWebBrowser.

8.1.12.4.20 TWebBrowser.OnPrivacyImpactedStateChange Event

Pascal

```
property OnPrivacyImpactedStateChange: TWebBrowserPrivacyImpactedStateChange;
```

C++

```
__property TWebBrowserPrivacyImpactedStateChange OnPrivacyImpactedStateChange;
```

Description

This is OnPrivacyImpactedStateChange, a member of class TWebBrowser.

8.1.12.4.21 TWebBrowser.OnProgressChange Event

Occurs when the progress of a download operation is updated.

Pascal

```
property OnProgressChange: TWebBrowserProgressChange;
```

C++

```
__property TWebBrowserProgressChange OnProgressChange;
```

Description

Write (see page 3707) an OnProgressChange event handler to provide visual feedback about the download process. For example, an OnProgressChange event handler can update a TProgressBar component or display the number of bytes downloaded so far.

See Also

OnDocumentComplete (see page 546)

OnDownloadBegin (see page 547)

Navigate (see page 512)

Navigate2 (see page 514)

OnDownloadComplete (see page 547)

TProgressBar

8.1.12.4.22 TWebBrowser.OnPropertyChange Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnPropertyChange: TWebBrowserPropertyChange;
```

C++

```
__property TWebBrowserPropertyChange OnPropertyChange;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnPropertyChange occurs with Internet Explorer when a property is modified using the PutProperty (see page 515) method.

8.1.12.4.23 TWebBrowser.OnQuit Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnQuit: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnQuit;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnQuit occurs when the Internet Explorer is about to shut down.

8.1.12.4.24 TWebBrowser.OnSetSecureLockIcon Event

Pascal

```
property OnSetSecureLockIcon: TWebBrowserSetSecureLockIcon;
```

C++

```
__property TWebBrowserSetSecureLockIcon OnSetSecureLockIcon;
```

Description

This is OnSetSecureLockIcon, a member of class TWebBrowser.

8.1.12.4.25 TWebBrowser.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it

contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

8.1.12.4.26 TWebBrowser.OnStatusBar Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnStatusBar: TWebBrowserOnStatusBar;
```

C++

```
__property TWebBrowserOnStatusBar OnStatusBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnStatusBar occurs when the Internet Explorer first shows or hides its status bar.

8.1.12.4.27 TWebBrowser.OnStatusTextChange Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnStatusTextChange: TWebBrowserStatusTextChange;
```

C++

```
__property TWebBrowserStatusTextChange OnStatusTextChange;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnStatusTextChange occurs when the text displayed in the Internet Explorer's status bar changes.

8.1.12.4.28 TWebBrowser.OnTheaterMode Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnTheaterMode: TWebBrowserOnTheaterMode;
```

C++

```
__property TWebBrowserOnTheaterMode OnTheaterMode;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnTheaterMode occurs when the Internet Explorer changes into or out of theater mode.

8.1.12.4.29 TWebBrowser.OnTitleChange Event

Occurs when the title of a document in the WebBrowser (see page 837) control becomes available or changes.

Pascal

```
property OnTitleChange: TWebBrowserTitleChange;
```

C++

```
__property TWebBrowserTitleChange OnTitleChange;
```

Description

Write (see page 3707) an OnTitleChange event handler to respond when the Web browser obtains information about the document title. Because the title can change while an HTML page is downloading, LocationName (see page 529) is originally set to the URL of the document. After the title specified in the HTML page, if any, becomes available, LocationName (see page 529) is changed to reflect the actual title.

See Also

LocationName (see page 529)

8.1.12.4.30 TWebBrowser.OnToolBar Event

Not used in TWebBrowser (see page 504).

Pascal

```
property OnToolBar: TWebBrowserOnToolBar;
```

C++

```
__property TWebBrowserOnToolBar OnToolBar;
```

Description

Do not use this event. It is included in the TWebBrowser (see page 504) interface because the IWebBrowser2 (see page 749) interface is shared with Internet Explorer. OnToolBar occurs when the Internet Explorer changes which tool bars are visible.

8.1.12.4.31 TWebBrowser.OnUpdatePageStatus Event

Pascal

```
property OnUpdatePageStatus: TWebBrowserUpdatePageStatus;
```

C++

```
__property TWebBrowserUpdatePageStatus OnUpdatePageStatus;
```

Description

This is OnUpdatePageStatus, a member of class TWebBrowser.

8.1.12.4.32 TWebBrowser.OnVisible Event

Occurs when the Web browser window is about to be shown or hidden.

Pascal

```
property OnVisible: TWebBrowserOnVisible;
```

C++

```
__property TWebBrowserOnVisible OnVisible;
```

Description

Write (see page 3707) an OnVisible event handler to make adjustments to compensate for the appearance or disappearance of the Web browser window.

See Also

Visible (see page 544)

8.1.12.4.33 TWebBrowser.OnWindowClosing Event

Pascal

```
property OnWindowClosing: TWebBrowserWindowClosing;
```

C++

```
__property TWebBrowserWindowClosing OnWindowClosing;
```

Description

This is OnWindowClosing, a member of class TWebBrowser.

8.1.12.4.34 TWebBrowser.OnWindowSetHeight Event

Pascal

```
property OnWindowSetHeight: TWebBrowserWindowSetHeight;
```

C++

```
__property TWebBrowserWindowSetHeight OnWindowSetHeight;
```

Description

This is OnWindowSetHeight, a member of class TWebBrowser.

8.1.12.4.35 TWebBrowser.OnWindowSetLeft Event

Pascal

```
property OnWindowSetLeft: TWebBrowserWindowSetLeft;
```

C++

```
__property TWebBrowserWindowSetLeft OnWindowSetLeft;
```

Description

This is OnWindowSetLeft, a member of class TWebBrowser.

8.1.12.4.36 TWebBrowser.OnWindowSetResizable Event

Pascal

```
property OnWindowSetResizable: TWebBrowserWindowSetResizable;
```

C++

```
__property TWebBrowserWindowSetResizable OnWindowSetResizable;
```

Description

This is OnWindowSetResizable, a member of class TWebBrowser.

8.1.12.4.37 TWebBrowser.OnWindowSetTop Event

Pascal

```
property OnWindowSetTop: TWebBrowserWindowSetTop;
```

C++

```
__property TWebBrowserWindowSetTop OnWindowSetTop;
```

Description

This is OnWindowSetTop, a member of class TWebBrowser.

8.1.12.4.38 TWebBrowser.OnWindowSetWidth Event

Pascal

```
property OnWindowSetWidth: TWebBrowserWindowSetWidth;
```

C++

```
__property TWebBrowserWindowSetWidth OnWindowSetWidth;
```

Description

This is OnWindowSetWidth, a member of class TWebBrowser.

8.1.13 TWebBrowser_V1 Class

Class Hierarchy

Pascal

```
TWebBrowser_V1 = class(TOleControl);
```

C++

```
class TWebBrowser_V1 : public TOleControl;
```

File

SHDocVw

Description

This is class SHDocVw.TWebBrowser_V1.

8.1.13.1 TWebBrowser_V1 Members

The following tables list the members exposed by TWebBrowser_V1.

Protected Methods

	Name	Description
	CreateControl (see page 572)	This is CreateControl, a member of class TWebBrowser_V1.
	Get_Application (see page 572)	This is Get_Application, a member of class TWebBrowser_V1.
	Get_Container (see page 572)	This is Get_Container, a member of class TWebBrowser_V1.
	Get_Document (see page 572)	This is Get_Document, a member of class TWebBrowser_V1.
	Get_Parent (see page 572)	This is Get_Parent, a member of class TWebBrowser_V1.
	InitControlData (see page 574)	This is InitControlData, a member of class TWebBrowser_V1.

Public Methods

	Name	Description
	GoBack (see page 573)	This is GoBack, a member of class TWebBrowser_V1.
	GoForward (see page 573)	This is GoForward, a member of class TWebBrowser_V1.
	GoHome (see page 573)	This is GoHome, a member of class TWebBrowser_V1.
	GoSearch (see page 573)	This is GoSearch, a member of class TWebBrowser_V1.
	Navigate (see page 574)	This is the overview for the Navigate method overload.
	Refresh (see page 575)	Repaints the control on the screen.
	Refresh2 (see page 576)	This is the overview for the Refresh2 method overload.
	Stop (see page 576)	This is Stop, a member of class TWebBrowser_V1.

Public Properties

	Name	Description
	Application (see page 581)	This is Application, a member of class TWebBrowser_V1.
	Busy (see page 581)	This is Busy, a member of class TWebBrowser_V1.
	Container (see page 582)	This is Container, a member of class TWebBrowser_V1.
	ControlInterface (see page 582)	This is ControlInterface, a member of class TWebBrowser_V1.
	DefaultInterface (see page 582)	This is DefaultInterface, a member of class TWebBrowser_V1.
	Document (see page 582)	This is Document, a member of class TWebBrowser_V1.
	LocationName (see page 584)	This is LocationName, a member of class TWebBrowser_V1.
	LocationURL (see page 585)	This is LocationURL, a member of class TWebBrowser_V1.
	Parent (see page 585)	Indicates the parent of the control.
	TopLevelContainer (see page 595)	This is TopLevelContainer, a member of class TWebBrowser_V1.
	type_ (see page 595)	This is type_, a member of class TWebBrowser_V1.

Published Events

	Name	Description
	OnBeforeNavigate (see page 597)	This is OnBeforeNavigate, a member of class TWebBrowser_V1.
	OnCommandStateChange (see page 597)	This is OnCommandStateChange, a member of class TWebBrowser_V1.
	OnDownloadBegin (see page 598)	This is OnDownloadBegin, a member of class TWebBrowser_V1.
	OnDownloadComplete (see page 598)	This is OnDownloadComplete, a member of class TWebBrowser_V1.
	OnDragDrop (see page 598)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 601)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 604)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 605)	Occurs when a control receives the input focus.
	OnExit (see page 608)	Occurs when the input focus shifts away from one control to another.

	OnFrameBeforeNavigate (see page 612)	This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.
	OnFrameNavigateComplete (see page 612)	This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.
	OnFrameNewWindow (see page 612)	This is OnFrameNewWindow, a member of class TWebBrowser_V1.
	OnNavigateComplete (see page 612)	This is OnNavigateComplete, a member of class TWebBrowser_V1.
	OnNewWindow (see page 613)	This is OnNewWindow, a member of class TWebBrowser_V1.
	OnProgressChange (see page 613)	This is OnProgressChange, a member of class TWebBrowser_V1.
	OnPropertyChange (see page 613)	This is OnPropertyChange, a member of class TWebBrowser_V1.
	OnQuit (see page 613)	This is OnQuit, a member of class TWebBrowser_V1.
	OnStartDrag (see page 613)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusTextChange (see page 614)	This is OnStatusTextChange, a member of class TWebBrowser_V1.
	OnTitleChange (see page 614)	This is OnTitleChange, a member of class TWebBrowser_V1.
	OnWindowActivate (see page 615)	This is OnWindowActivate, a member of class TWebBrowser_V1.
	OnWindowMove (see page 615)	This is OnWindowMove, a member of class TWebBrowser_V1.
	OnWindowResize (see page 615)	This is OnWindowResize, a member of class TWebBrowser_V1.

Published Properties

	Name	Description
	Align (see page 577)	Determines how the control aligns within its container (parent control).
	Anchors (see page 581)	Specifies how the control is anchored to its parent.
	DragCursor (see page 582)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 583)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ParentShowHint (see page 586)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 587)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 592)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 593)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 594)	Determines if the user can tab to a control.
	Visible (see page 595)	Determines whether the component appears onscreen.

8.1.13.2 TWebBrowser_V1 Methods

The methods of the TWebBrowser_V1 class are listed here.

Protected Methods

	Name	Description
	CreateControl (see page 572)	This is CreateControl, a member of class TWebBrowser_V1.
	Get_Application (see page 572)	This is Get_Application, a member of class TWebBrowser_V1.
	Get_Container (see page 572)	This is Get_Container, a member of class TWebBrowser_V1.
	Get_Document (see page 572)	This is Get_Document, a member of class TWebBrowser_V1.
	Get_Parent (see page 572)	This is Get_Parent, a member of class TWebBrowser_V1.
	InitControlData (see page 574)	This is InitControlData, a member of class TWebBrowser_V1.

Public Methods

	Name	Description
	GoBack (see page 573)	This is GoBack, a member of class TWebBrowser_V1.
	GoForward (see page 573)	This is GoForward, a member of class TWebBrowser_V1.
	GoHome (see page 573)	This is GoHome, a member of class TWebBrowser_V1.
	GoSearch (see page 573)	This is GoSearch, a member of class TWebBrowser_V1.
	Navigate (see page 574)	This is the overview for the Navigate method overload.
	Refresh (see page 575)	Repaints the control on the screen.
	Refresh2 (see page 576)	This is the overview for the Refresh2 method overload.
	Stop (see page 576)	This is Stop, a member of class TWebBrowser_V1.

8.1.13.2.1 TWebBrowser_V1.CreateControl Method

Pascal

```
procedure CreateControl;
```

C++

```
__fastcall CreateControl();
```

Description

This is CreateControl, a member of class TWebBrowser_V1.

8.1.13.2.2 TWebBrowser_V1.Get_Application Method

Pascal

```
function Get_Application: IDispatch;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

This is Get_Application, a member of class TWebBrowser_V1.

8.1.13.2.3 TWebBrowser_V1.Get_Container Method

Pascal

```
function Get_Container: IDispatch;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

This is Get_Container, a member of class TWebBrowser_V1.

8.1.13.2.4 TWebBrowser_V1.Get_Document Method

Pascal

```
function Get_Document: IDispatch;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

This is Get_Document, a member of class TWebBrowser_V1.

8.1.13.2.5 TWebBrowser_V1.Get_Parent Method

Pascal

```
function Get_Parent: IDispatch;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

This is Get_Parent, a member of class TWebBrowser_V1.

8.1.13.2.6 TWebBrowser_V1.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class TWebBrowser_V1.

8.1.13.2.7 TWebBrowser_V1.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class TWebBrowser_V1.

8.1.13.2.8 TWebBrowser_V1.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class TWebBrowser_V1.

8.1.13.2.9 TWebBrowser_V1.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class TWebBrowser_V1.

8.1.13.2.10 TWebBrowser_V1.InitControlData Method

Pascal

```
procedure InitControlData; override;
```

C++

```
virtual __fastcall InitControlData();
```

Description

This is InitControlData, a member of class TWebBrowser_V1.

8.1.13.2.11 Navigate Method

This is the overview for the Navigate method overload.

Overload List

	Name	Description
	TWebBrowser_V1.Navigate (WideString) (see page 574)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant) (see page 574)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant) (see page 575)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant, OleVariant) (see page 575)	This is Navigate, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Navigate (WideString, OleVariant, OleVariant, OleVariant, OleVariant) (see page 575)	This is Navigate, a member of class TWebBrowser_V1.

8.1.13.2.11.1 TWebBrowser_V1.Navigate Method (WideString)

Pascal

```
procedure Navigate(const URL: WideString); overload;
```

C++

```
__fastcall Navigate(const BSTR URL);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

8.1.13.2.11.2 TWebBrowser_V1.Navigate Method (WideString, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

8.1.13.2.11.3 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

8.1.13.2.11.4 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

8.1.13.2.11.5 TWebBrowser_V1.Navigate Method (WideString, OleVariant, OleVariant, OleVariant, OleVariant)

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); overload;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class TWebBrowser_V1.

8.1.13.2.12 TWebBrowser_V1.Refresh Method

Repaints the control on the screen.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh method to repaint the control immediately. Refresh calls the Repaint method. Use the Refresh and Repaint methods interchangeably.

See Also

Invalidate

Parent

Repaint

Update

8.1.13.2.13 Refresh2 Method

This is the overview for the Refresh2 method overload.

Overload List

	Name	Description
	TWebBrowser_V1.Refresh2 () (see page 576)	This is Refresh2, a member of class TWebBrowser_V1.
	TWebBrowser_V1.Refresh2 (OleVariant) (see page 576)	This is Refresh2, a member of class TWebBrowser_V1.

8.1.13.2.13.1 TWebBrowser_V1.Refresh2 Method ()

Pascal

```
procedure Refresh2; overload;
```

C++

```
__fastcall Refresh2();
```

Description

This is Refresh2, a member of class TWebBrowser_V1.

8.1.13.2.13.2 TWebBrowser_V1.Refresh2 Method (OleVariant)

Pascal

```
procedure Refresh2(var Level: OleVariant); overload;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class TWebBrowser_V1.

8.1.13.2.14 TWebBrowser_V1.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class TWebBrowser_V1.

8.1.13.3 TWebBrowser_V1 Properties

The properties of the TWebBrowser_V1 class are listed here.

Public Properties

	Name	Description
	Application (see page 581)	This is Application, a member of class TWebBrowser_V1.
	Busy (see page 581)	This is Busy, a member of class TWebBrowser_V1.
	Container (see page 582)	This is Container, a member of class TWebBrowser_V1.
	ControlInterface (see page 582)	This is ControlInterface, a member of class TWebBrowser_V1.
	DefaultInterface (see page 582)	This is DefaultInterface, a member of class TWebBrowser_V1.
	Document (see page 582)	This is Document, a member of class TWebBrowser_V1.
	LocationName (see page 584)	This is LocationName, a member of class TWebBrowser_V1.
	LocationURL (see page 585)	This is LocationURL, a member of class TWebBrowser_V1.
	Parent (see page 585)	Indicates the parent of the control.
	TopLevelContainer (see page 595)	This is TopLevelContainer, a member of class TWebBrowser_V1.
	type_ (see page 595)	This is type_, a member of class TWebBrowser_V1.

Published Properties

	Name	Description
	Align (see page 577)	Determines how the control aligns within its container (parent control).
	Anchors (see page 581)	Specifies how the control is anchored to its parent.
	DragCursor (see page 582)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragMode (see page 583)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	ParentShowHint (see page 586)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 587)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 592)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 593)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 594)	Determines if the user can tab to a control.
	Visible (see page 595)	Determines whether the component appears onscreen.

8.1.13.3.1 TWebBrowser_V1.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
```

```
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
```

```
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

8.1.13.3.2 TWebBrowser_V1.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

8.1.13.3.3 TWebBrowser_V1.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class TWebBrowser_V1.

8.1.13.3.4 TWebBrowser_V1.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class TWebBrowser_V1.

8.1.13.3.5 TWebBrowser_V1.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class TWebBrowser_V1.

8.1.13.3.6 TWebBrowser_V1.ControlInterface Property

Pascal

```
property ControlInterface: IWebBrowser;
```

C++

```
__property IWebBrowser ControlInterface;
```

Description

This is ControlInterface, a member of class TWebBrowser_V1.

8.1.13.3.7 TWebBrowser_V1.DefaultInterface Property

Pascal

```
property DefaultInterface: IWebBrowser;
```

C++

```
__property IWebBrowser DefaultInterface;
```

Description

This is DefaultInterface, a member of class TWebBrowser_V1.

8.1.13.3.8 TWebBrowser_V1.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class TWebBrowser_V1.

8.1.13.3.9 TWebBrowser_V1.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

8.1.13.3.10 TWebBrowser_V1.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}
```

```
void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

8.1.13.3.11 TWebBrowser_V1.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class TWebBrowser_V1.

8.1.13.3.12 TWebBrowser_V1.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class TWebBrowser_V1.

8.1.13.3.13 TWebBrowser_V1.Parent Property

Indicates the parent of the control.

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

Use the Parent property to get or set the parent of this control. The parent of a control is the control that contains the control. For example, if an application includes three radio buttons in a group box, the group box is the parent of the three radio buttons, and the radio buttons are the child controls of the group box.

To serve as a parent, a control must be an instance of a descendant of TWinControl.

When creating a new control at runtime, assign a Parent property value for the new control. Usually, this is a form, panel, group box, or some control that is designed to contain another. Changing the parent of a control moves the control onscreen so that it is displayed within the new parent. When the parent control moves, the child moves with the parent.

Some controls (such as ActiveX controls) are contained in native windows rather than in a parent VCL control. For these controls, the value of Parent is nil (Delphi) or NULL (C++) and the ParentWindow property specifies the window.

Note: The Parent property declared in TControl is similar to the Owner property declared in TComponent, in that the Parent of a control frees the control just as the Owner of a component frees that Component. However, the Parent of a control is always a windowed control that visually contains the control, and is responsible for writing the control to a stream when the form is saved. The Owner of a component is the component that was passed as a parameter in the constructor and, if assigned, initiates the process of saving all objects (including the control and its parent) when the form is saved.

C++ Examples:

```
/*
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (Button1->Parent != GroupBox1) // second click?
  {
    RemoveControl(Button1);
    GroupBox1->InsertControl(Button1);
  }
}

/*
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
*/

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Button2->Parent = GroupBox1;
}
```

Delphi Examples:

```
{
This example uses a button placed next to a group box. When the
user clicks the button, the group box becomes the parent of the
button, so the button moves inside the group box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Button1.Parent <> GroupBox1) then // second click?
  begin
    RemoveControl(Button1);
    GroupBox1.InsertControl(Button1);
  end;
end;

{
Note that it was necessary to remove the button from the Controls
property of the form before the button actually moves into the
group box. This code accomplishes the same thing:
}

procedure TForm1.Button2Click(Sender: TObject);
begin
  Button2.Parent := GroupBox1;
end;
```

See Also

Owner

Controls

HostDockSite

ParentWindow

8.1.13.3.14 TWebBrowser_V1.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

8.1.13.3.15 TWebBrowser_V1.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
```

```
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
```

```
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
```

```
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
```

```
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

8.1.13.3.16 TWebBrowser_V1.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

8.1.13.3.17 TWebBrowser_V1.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

8.1.13.3.18 TWebBrowser_V1.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

8.1.13.3.19 TWebBrowser_V1.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class TWebBrowser_V1.

8.1.13.3.20 TWebBrowser_V1.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class TWebBrowser_V1.

8.1.13.3.21 TWebBrowser_V1.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

8.1.13.4 TWebBrowser_V1 Events

The events of the TWebBrowser_V1 class are listed here.

Published Events

	Name	Description
	OnBeforeNavigate (see page 597)	This is OnBeforeNavigate, a member of class TWebBrowser_V1.
	OnCommandStateChange (see page 597)	This is OnCommandStateChange, a member of class TWebBrowser_V1.
	OnDownloadBegin (see page 598)	This is OnDownloadBegin, a member of class TWebBrowser_V1.
	OnDownloadComplete (see page 598)	This is OnDownloadComplete, a member of class TWebBrowser_V1.
	OnDragDrop (see page 598)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 601)	Occurs when the user drags an object over a control.
	OnEndDrag (see page 604)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 605)	Occurs when a control receives the input focus.
	OnExit (see page 608)	Occurs when the input focus shifts away from one control to another.
	OnFrameBeforeNavigate (see page 612)	This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.
	OnFrameNavigateComplete (see page 612)	This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.
	OnFrameNewWindow (see page 612)	This is OnFrameNewWindow, a member of class TWebBrowser_V1.
	OnNavigateComplete (see page 612)	This is OnNavigateComplete, a member of class TWebBrowser_V1.
	OnNewWindow (see page 613)	This is OnNewWindow, a member of class TWebBrowser_V1.
	OnProgressChange (see page 613)	This is OnProgressChange, a member of class TWebBrowser_V1.
	OnPropertyChange (see page 613)	This is OnPropertyChange, a member of class TWebBrowser_V1.
	OnQuit (see page 613)	This is OnQuit, a member of class TWebBrowser_V1.
	OnStartDrag (see page 613)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStatusTextChange (see page 614)	This is OnStatusTextChange, a member of class TWebBrowser_V1.
	OnTitleChange (see page 614)	This is OnTitleChange, a member of class TWebBrowser_V1.
	OnWindowActivate (see page 615)	This is OnWindowActivate, a member of class TWebBrowser_V1.
	OnWindowMove (see page 615)	This is OnWindowMove, a member of class TWebBrowser_V1.
	OnWindowResize (see page 615)	This is OnWindowResize, a member of class TWebBrowser_V1.

8.1.13.4.1 TWebBrowser_V1.OnBeforeNavigate Event

Pascal

```
property OnBeforeNavigate: TWebBrowser_V1BeforeNavigate;
```

C++

```
__property TWebBrowser_V1BeforeNavigate OnBeforeNavigate;
```

Description

This is OnBeforeNavigate, a member of class TWebBrowser_V1.

8.1.13.4.2 TWebBrowser_V1.OnCommandStateChange Event

Pascal

```
property OnCommandStateChange: TWebBrowser_V1CommandStateChange;
```

C++

```
__property TWebBrowser_V1CommandStateChange OnCommandStateChange;
```

Description

This is OnCommandStateChange, a member of class TWebBrowser_V1.

8.1.13.4.3 TWebBrowser_V1.OnDownloadBegin Event

Pascal

```
property OnDownloadBegin: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadBegin;
```

Description

This is OnDownloadBegin, a member of class TWebBrowser_V1.

8.1.13.4.4 TWebBrowser_V1.OnDownloadComplete Event

Pascal

```
property OnDownloadComplete: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDownloadComplete;
```

Description

This is OnDownloadComplete, a member of class TWebBrowser_V1.

8.1.13.4.5 TWebBrowser_V1.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}
```

```
void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
```

```
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

8.1.13.4.6 TWebBrowser_V1.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
```

```
  dragging of the label as soon as the user drags the label
  onto the panel:
  Note: Set the label's DragMode property to dmAutomatic.
  */
  void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

  {
    reinterpret_cast<TForm1 *>(Source)->Left = X;
    reinterpret_cast<TForm1 *>(Source)->Top = Y;
  }

  void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
            TDragState State, bool &Accept)
  {
    Accept = Source->ClassNameIs("TLabel");
  }

  void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
            TDragState State, bool &Accept)
  {
    Accept = false;
    if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
      dynamic_cast<TLabel *>(Source)->EndDrag(false);
  }
  /*
  This code comes from an application that contains a list box
  and three labels, each with a different font and color.  The
  DragMode property for each of the labels is dmAutomatic.
  The user can select a label and drag it to a list box and
  drop it. When the label is dropped, the items in the list
  box assume the color and font of the dropped label.
  */

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:
  void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
  TDragState State, bool &Accept)
  {
     Accept = Source->ClassNameIs("TLabel");
  }

  /*
  This OnDragDrop event handler implements the drop behavior.
  */
  void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
  {
    if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
    {
      TListBox *DestList = dynamic_cast<TListBox *>(Sender);
      DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
      DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
      DestList->DoubleBuffered = true;
      DestList->Color = clWindow;
    }
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    ListBox1->Items->Add("Not");
    ListBox1->Items->Add("In");
    ListBox1->Items->Add("Alphabetical");
    ListBox1->Items->Add("Order");

  }
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

8.1.13.4.7 TWebBrowser_V1.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
```

```
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

8.1.13.4.8 TWebBrowser_V1.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to

another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

8.1.13.4.9 TWebBrowser_V1.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
```

```
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;
```

```
begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

8.1.13.4.10 TWebBrowser_V1.OnFrameBeforeNavigate Event

Pascal

```
property OnFrameBeforeNavigate: TWebBrowser_V1FrameBeforeNavigate;
```

C++

```
__property TWebBrowser_V1FrameBeforeNavigate OnFrameBeforeNavigate;
```

Description

This is OnFrameBeforeNavigate, a member of class TWebBrowser_V1.

8.1.13.4.11 TWebBrowser_V1.OnFrameNavigateComplete Event

Pascal

```
property OnFrameNavigateComplete: TWebBrowser_V1FrameNavigateComplete;
```

C++

```
__property TWebBrowser_V1FrameNavigateComplete OnFrameNavigateComplete;
```

Description

This is OnFrameNavigateComplete, a member of class TWebBrowser_V1.

8.1.13.4.12 TWebBrowser_V1.OnFrameNewWindow Event

Pascal

```
property OnFrameNewWindow: TWebBrowser_V1FrameNewWindow;
```

C++

```
__property TWebBrowser_V1FrameNewWindow OnFrameNewWindow;
```

Description

This is OnFrameNewWindow, a member of class TWebBrowser_V1.

8.1.13.4.13 TWebBrowser_V1.OnNavigateComplete Event

Pascal

```
property OnNavigateComplete: TWebBrowser_V1NavigateComplete;
```

C++

```
__property TWebBrowser_V1NavigateComplete OnNavigateComplete;
```

Description

This is OnNavigateComplete, a member of class TWebBrowser_V1.

8.1.13.4.14 TWebBrowser_V1.OnNewWindow Event

Pascal

```
property OnNewWindow: TWebBrowser_V1NewWindow;
```

C++

```
__property TWebBrowser_V1NewWindow OnNewWindow;
```

Description

This is OnNewWindow, a member of class TWebBrowser_V1.

8.1.13.4.15 TWebBrowser_V1.OnProgressChange Event

Pascal

```
property OnProgressChange: TWebBrowser_V1ProgressChange;
```

C++

```
__property TWebBrowser_V1ProgressChange OnProgressChange;
```

Description

This is OnProgressChange, a member of class TWebBrowser_V1.

8.1.13.4.16 TWebBrowser_V1.OnPropertyChange Event

Pascal

```
property OnPropertyChange: TWebBrowser_V1PropertyChange;
```

C++

```
__property TWebBrowser_V1PropertyChange OnPropertyChange;
```

Description

This is OnPropertyChange, a member of class TWebBrowser_V1.

8.1.13.4.17 TWebBrowser_V1.OnQuit Event

Pascal

```
property OnQuit: TWebBrowser_V1Quit;
```

C++

```
__property TWebBrowser_V1Quit OnQuit;
```

Description

This is OnQuit, a member of class TWebBrowser_V1.

8.1.13.4.18 TWebBrowser_V1.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

8.1.13.4.19 TWebBrowser_V1.OnStatusTextChange Event

Pascal

```
property OnStatusTextChange: TWebBrowser_V1StatusTextChange;
```

C++

```
__property TWebBrowser_V1StatusTextChange OnStatusTextChange;
```

Description

This is OnStatusTextChange, a member of class TWebBrowser_V1.

8.1.13.4.20 TWebBrowser_V1.OnTitleChange Event

Pascal

```
property OnTitleChange: TWebBrowser_V1TitleChange;
```

C++

```
__property TWebBrowser_V1TitleChange OnTitleChange;
```

Description

This is OnTitleChange, a member of class TWebBrowser_V1.

8.1.13.4.21 TWebBrowser_V1.OnWindowActivate Event

Pascal

```
property OnWindowActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowActivate;
```

Description

This is OnWindowActivate, a member of class TWebBrowser_V1.

8.1.13.4.22 TWebBrowser_V1.OnWindowMove Event

Pascal

```
property OnWindowMove: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowMove;
```

Description

This is OnWindowMove, a member of class TWebBrowser_V1.

8.1.13.4.23 TWebBrowser_V1.OnWindowResize Event

Pascal

```
property OnWindowResize: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnWindowResize;
```

Description

This is OnWindowResize, a member of class TWebBrowser_V1.

8.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	_SearchAssistantEvents (see page 616)	This is class SHDocVw._SearchAssistantEvents.
	DShellNameSpaceEvents (see page 618)	This is class SHDocVw.DShellNameSpaceEvents.
	DShellWindowsEvents (see page 619)	This is class SHDocVw.DShellWindowsEvents.
	DWebBrowserEvents (see page 621)	This is class SHDocVw.DWebBrowserEvents.
	DWebBrowserEvents2 (see page 626)	This is class SHDocVw.DWebBrowserEvents2.
	IScriptErrorList (see page 635)	This is class SHDocVw.IScriptErrorList.
	IScriptErrorListDisp (see page 639)	This is class SHDocVw.IScriptErrorListDisp.
	ISearch (see page 644)	This is class SHDocVw.ISearch.
	ISearchAssistantOC (see page 646)	This is class SHDocVw.ISearchAssistantOC.
	ISearchAssistantOC2 (see page 656)	This is class SHDocVw.ISearchAssistantOC2.
	ISearchAssistantOC2Disp (see page 658)	This is class SHDocVw.ISearchAssistantOC2Disp.
	ISearchAssistantOC3 (see page 666)	This is class SHDocVw.ISearchAssistantOC3.
	ISearchAssistantOC3Disp (see page 669)	This is class SHDocVw.ISearchAssistantOC3Disp.
	ISearchAssistantOCDisp (see page 677)	This is class SHDocVw.ISearchAssistantOCDisp.
	ISearchDisp (see page 685)	This is class SHDocVw.ISearchDisp.
	ISearches (see page 686)	This is class SHDocVw.ISearches.
	ISearchesDisp (see page 689)	This is class SHDocVw.ISearchesDisp.
	IShellFavoritesNameSpace (see page 691)	This is class SHDocVw.IShellFavoritesNameSpace.
	IShellFavoritesNameSpaceDisp (see page 695)	This is class SHDocVw.IShellFavoritesNameSpaceDisp.
	IShellNameSpace (see page 700)	This is class SHDocVw.IShellNameSpace.
	IShellNameSpaceDisp (see page 709)	This is class SHDocVw.IShellNameSpaceDisp.
	IShellUIHelper (see page 716)	This is class SHDocVw.IShellUIHelper.
	IShellUIHelperDisp (see page 721)	This is class SHDocVw.IShellUIHelperDisp.
	IShellWindows (see page 725)	This is class SHDocVw.IShellWindows.
	IShellWindowsDisp (see page 729)	This is class SHDocVw.IShellWindowsDisp.
	IWebBrowser (see page 733)	IWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).
	IWebBrowser2 (see page 749)	IWebBrowser2 provides access to the Web browser functionality of Microsoft Shell Doc Object and Control Library (SHDOCVW.DLL).
	IWebBrowser2Disp (see page 763)	This is class SHDocVw.IWebBrowser2Disp.
	IWebBrowserApp (see page 776)	This is class SHDocVw.IWebBrowserApp.
	IWebBrowserAppDisp (see page 785)	This is class SHDocVw.IWebBrowserAppDisp.
	IWebBrowserDisp (see page 795)	This is class SHDocVw.IWebBrowserDisp.

8.2.1 _SearchAssistantEvents Interface

Class Hierarchy

SHDocVw._SearchAssistantEvents

Pascal

```
_SearchAssistantEvents = dispinterface;
```

C++

```
class _SearchAssistantEvents;
```

File

SHDocVw

Description

This is class SHDocVw._SearchAssistantEvents.

8.2.1.1 _SearchAssistantEvents Members

The following tables list the members exposed by _SearchAssistantEvents.

Public Methods

	Name	Description
	OnNewSearch (see page 617)	This is OnNewSearch, a member of class _SearchAssistantEvents.
	OnNextMenuSelect (see page 617)	This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

8.2.1.2 _SearchAssistantEvents Methods

The methods of the _SearchAssistantEvents class are listed here.

Public Methods

	Name	Description
	OnNewSearch (see page 617)	This is OnNewSearch, a member of class _SearchAssistantEvents.
	OnNextMenuSelect (see page 617)	This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

8.2.1.2.1 _SearchAssistantEvents.OnNewSearch Method

Pascal

```
procedure OnNewSearch;
```

C++

```
__fastcall OnNewSearch();
```

Description

This is OnNewSearch, a member of class _SearchAssistantEvents.

8.2.1.2.2 _SearchAssistantEvents.OnNextMenuSelect Method

Pascal

```
procedure OnNextMenuSelect(idItem: Integer);
```

C++

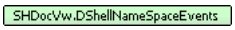
```
__fastcall OnNextMenuSelect(int idItem);
```

Description

This is OnNextMenuSelect, a member of class _SearchAssistantEvents.

8.2.2 DShellNameSpaceEvents Interface

Class Hierarchy



Pascal

```
DShellNameSpaceEvents = dispinterface;
```

C++

```
class DShellNameSpaceEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DShellNameSpaceEvents.

8.2.2.1 DShellNameSpaceEvents Members

The following tables list the members exposed by DShellNameSpaceEvents.

Public Methods

	Name	Description
	DoubleClick (see page 618)	This is DoubleClick, a member of class DShellNameSpaceEvents.
	FavoritesSelectionChange (see page 619)	This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.
	Initialized (see page 619)	This is Initialized, a member of class DShellNameSpaceEvents.
	SelectionChange (see page 619)	This is SelectionChange, a member of class DShellNameSpaceEvents.

8.2.2.2 DShellNameSpaceEvents Methods

The methods of the DShellNameSpaceEvents class are listed here.

Public Methods

	Name	Description
	DoubleClick (see page 618)	This is DoubleClick, a member of class DShellNameSpaceEvents.
	FavoritesSelectionChange (see page 619)	This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.
	Initialized (see page 619)	This is Initialized, a member of class DShellNameSpaceEvents.
	SelectionChange (see page 619)	This is SelectionChange, a member of class DShellNameSpaceEvents.

8.2.2.2.1 DShellNameSpaceEvents.DoubleClick Method

Pascal

```
procedure DoubleClick;
```

C++

```
__fastcall DoubleClick();
```

Description

This is DoubleClick, a member of class DShellNameSpaceEvents.

8.2.2.2.2 DShellNameSpaceEvents.FavoritesSelectionChange Method

Pascal

```
procedure FavoritesSelectionChange(cItems: Integer; hItem: Integer; const strName: WideString;
const strUrl: WideString; cVisits: Integer; const strDate: WideString; fAvailableOffline:
Integer);
```

C++

```
__fastcall FavoritesSelectionChange(int cItems, int hItem, const BSTR strName, const BSTR
strUrl, int cVisits, const BSTR strDate, int fAvailableOffline);
```

Description

This is FavoritesSelectionChange, a member of class DShellNameSpaceEvents.

8.2.2.2.3 DShellNameSpaceEvents.Initialized Method

Pascal

```
procedure Initialized;
```

C++

```
__fastcall Initialized();
```

Description

This is Initialized, a member of class DShellNameSpaceEvents.

8.2.2.2.4 DShellNameSpaceEvents.SelectionChange Method

Pascal

```
procedure SelectionChange;
```

C++

```
__fastcall SelectionChange();
```

Description

This is SelectionChange, a member of class DShellNameSpaceEvents.

8.2.3 DShellWindowsEvents Interface

Class Hierarchy

Pascal

```
DShellWindowsEvents = dispinterface;
```

C++

```
class DShellWindowsEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DShellWindowsEvents.

8.2.3.1 DShellWindowsEvents Members

The following tables list the members exposed by DShellWindowsEvents.

Public Methods

	Name	Description
	WindowRegistered (see page 620)	This is WindowRegistered, a member of class DShellWindowsEvents.
	WindowRevoked (see page 620)	This is WindowRevoked, a member of class DShellWindowsEvents.

8.2.3.2 DShellWindowsEvents Methods

The methods of the DShellWindowsEvents class are listed here.

Public Methods

	Name	Description
	WindowRegistered (see page 620)	This is WindowRegistered, a member of class DShellWindowsEvents.
	WindowRevoked (see page 620)	This is WindowRevoked, a member of class DShellWindowsEvents.

8.2.3.2.1 DShellWindowsEvents.WindowRegistered Method

Pascal

```
procedure WindowRegistered(lCookie: Integer);
```

C++

```
__fastcall WindowRegistered(int lCookie);
```

Description

This is WindowRegistered, a member of class DShellWindowsEvents.

8.2.3.2.2 DShellWindowsEvents.WindowRevoked Method

Pascal

```
procedure WindowRevoked(lCookie: Integer);
```

C++

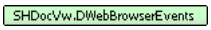
```
__fastcall WindowRevoked(int lCookie);
```

Description

This is WindowRevoked, a member of class DShellWindowsEvents.

8.2.4 DWebBrowserEvents Interface

Class Hierarchy



Pascal

```
DWebBrowserEvents = dispinterface;
```

C++

```
class DWebBrowserEvents;
```

File

SHDocVw

Description

This is class SHDocVw.DWebBrowserEvents.

8.2.4.1 DWebBrowserEvents Members

The following tables list the members exposed by DWebBrowserEvents.

Public Methods

	Name	Description
	BeforeNavigate (see page 622)	This is BeforeNavigate, a member of class DWebBrowserEvents.
	CommandStateChange (see page 622)	This is CommandStateChange, a member of class DWebBrowserEvents.
	DownloadBegin (see page 622)	This is DownloadBegin, a member of class DWebBrowserEvents.
	DownloadComplete (see page 623)	This is DownloadComplete, a member of class DWebBrowserEvents.
	FrameBeforeNavigate (see page 623)	This is FrameBeforeNavigate, a member of class DWebBrowserEvents.
	FrameNavigateComplete (see page 623)	This is FrameNavigateComplete, a member of class DWebBrowserEvents.
	FrameNewWindow (see page 623)	This is FrameNewWindow, a member of class DWebBrowserEvents.
	NavigateComplete (see page 624)	This is NavigateComplete, a member of class DWebBrowserEvents.
	NewWindow (see page 624)	This is NewWindow, a member of class DWebBrowserEvents.
	ProgressChange (see page 624)	This is ProgressChange, a member of class DWebBrowserEvents.
	PropertyChange (see page 624)	This is PropertyChange, a member of class DWebBrowserEvents.
	Quit (see page 624)	This is Quit, a member of class DWebBrowserEvents.
	StatusTextChange (see page 625)	This is StatusTextChange, a member of class DWebBrowserEvents.
	TitleChange (see page 625)	This is TitleChange, a member of class DWebBrowserEvents.
	WindowActivate (see page 625)	This is WindowActivate, a member of class DWebBrowserEvents.
	WindowMove (see page 625)	This is WindowMove, a member of class DWebBrowserEvents.
	WindowResize (see page 626)	This is WindowResize, a member of class DWebBrowserEvents.

8.2.4.2 DWebBrowserEvents Methods

The methods of the DWebBrowserEvents class are listed here.

Public Methods

	Name	Description
	BeforeNavigate (see page 622)	This is BeforeNavigate, a member of class DWebBrowserEvents.

	CommandStateChange (see page 622)	This is CommandStateChange, a member of class DWebBrowserEvents.
	DownloadBegin (see page 622)	This is DownloadBegin, a member of class DWebBrowserEvents.
	DownloadComplete (see page 623)	This is DownloadComplete, a member of class DWebBrowserEvents.
	FrameBeforeNavigate (see page 623)	This is FrameBeforeNavigate, a member of class DWebBrowserEvents.
	FrameNavigateComplete (see page 623)	This is FrameNavigateComplete, a member of class DWebBrowserEvents.
	FrameNewWindow (see page 623)	This is FrameNewWindow, a member of class DWebBrowserEvents.
	NavigateComplete (see page 624)	This is NavigateComplete, a member of class DWebBrowserEvents.
	NewWindow (see page 624)	This is NewWindow, a member of class DWebBrowserEvents.
	ProgressChange (see page 624)	This is ProgressChange, a member of class DWebBrowserEvents.
	PropertyChange (see page 624)	This is PropertyChange, a member of class DWebBrowserEvents.
	Quit (see page 624)	This is Quit, a member of class DWebBrowserEvents.
	StatusTextChange (see page 625)	This is StatusTextChange, a member of class DWebBrowserEvents.
	TitleChange (see page 625)	This is TitleChange, a member of class DWebBrowserEvents.
	WindowActivate (see page 625)	This is WindowActivate, a member of class DWebBrowserEvents.
	WindowMove (see page 625)	This is WindowMove, a member of class DWebBrowserEvents.
	WindowResize (see page 626)	This is WindowResize, a member of class DWebBrowserEvents.

8.2.4.2.1 DWebBrowserEvents.BeforeNavigate Method

Pascal

```
procedure BeforeNavigate(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool);
```

C++

```
__fastcall BeforeNavigate(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Cancel);
```

Description

This is BeforeNavigate, a member of class DWebBrowserEvents.

8.2.4.2.2 DWebBrowserEvents.CommandStateChange Method

Pascal

```
procedure CommandStateChange(Command: Integer; Enable: WordBool);
```

C++

```
__fastcall CommandStateChange(int Command, WordBool Enable);
```

Description

This is CommandStateChange, a member of class DWebBrowserEvents.

8.2.4.2.3 DWebBrowserEvents.DownloadBegin Method

Pascal

```
procedure DownloadBegin;
```

C++

```
__fastcall DownloadBegin();
```

Description

This is DownloadBegin, a member of class DWebBrowserEvents.

8.2.4.2.4 DWebBrowserEvents.DownloadComplete Method

Pascal

```
procedure DownloadComplete;
```

C++

```
__fastcall DownloadComplete();
```

Description

This is DownloadComplete, a member of class DWebBrowserEvents.

8.2.4.2.5 DWebBrowserEvents.FrameBeforeNavigate Method

Pascal

```
procedure FrameBeforeNavigate(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool);
```

C++

```
__fastcall FrameBeforeNavigate(const BSTR URL, int Flags, const BSTR TargetFrameName,
OleVariant PostData, const BSTR Headers, WordBool Cancel);
```

Description

This is FrameBeforeNavigate, a member of class DWebBrowserEvents.

8.2.4.2.6 DWebBrowserEvents.FrameNavigateComplete Method

Pascal

```
procedure FrameNavigateComplete(const URL: WideString);
```

C++

```
__fastcall FrameNavigateComplete(const BSTR URL);
```

Description

This is FrameNavigateComplete, a member of class DWebBrowserEvents.

8.2.4.2.7 DWebBrowserEvents.FrameNewWindow Method

Pascal

```
procedure FrameNewWindow(const URL: WideString; Flags: Integer; const TargetFrameName:
WideString; var PostData: OleVariant; const Headers: WideString; var Processed: WordBool);
```

C++

```
__fastcall FrameNewWindow(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Processed);
```

Description

This is FrameNewWindow, a member of class DWebBrowserEvents.

8.2.4.2.8 DWebBrowserEvents.NavigateComplete Method

Pascal

```
procedure NavigateComplete(const URL: WideString);
```

C++

```
__fastcall NavigateComplete(const BSTR URL);
```

Description

This is NavigateComplete, a member of class DWebBrowserEvents.

8.2.4.2.9 DWebBrowserEvents.NewWindow Method

Pascal

```
procedure NewWindow(const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool);
```

C++

```
__fastcall NewWindow(const BSTR URL, int Flags, const BSTR TargetFrameName, OleVariant
PostData, const BSTR Headers, WordBool Processed);
```

Description

This is NewWindow, a member of class DWebBrowserEvents.

8.2.4.2.10 DWebBrowserEvents.ProgressChange Method

Pascal

```
procedure ProgressChange(Progress: Integer; ProgressMax: Integer);
```

C++

```
__fastcall ProgressChange(int Progress, int ProgressMax);
```

Description

This is ProgressChange, a member of class DWebBrowserEvents.

8.2.4.2.11 DWebBrowserEvents.PropertyChange Method

Pascal

```
procedure PropertyChange(const Property_: WideString);
```

C++

```
__fastcall PropertyChange(const BSTR Property_);
```

Description

This is PropertyChange, a member of class DWebBrowserEvents.

8.2.4.2.12 DWebBrowserEvents.Quit Method

Pascal

```
procedure Quit(var Cancel: WordBool);
```

C++

```
__fastcall Quit(WordBool Cancel);
```

Description

This is Quit, a member of class DWebBrowserEvents.

8.2.4.2.13 DWebBrowserEvents.StatusTextChange Method

Pascal

```
procedure StatusTextChange(const Text: WideString);
```

C++

```
__fastcall StatusTextChange(const BSTR Text);
```

Description

This is StatusTextChange, a member of class DWebBrowserEvents.

8.2.4.2.14 DWebBrowserEvents.TitleChange Method

Pascal

```
procedure TitleChange(const Text: WideString);
```

C++

```
__fastcall TitleChange(const BSTR Text);
```

Description

This is TitleChange, a member of class DWebBrowserEvents.

8.2.4.2.15 DWebBrowserEvents.WindowActivate Method

Pascal

```
procedure WindowActivate;
```

C++

```
__fastcall WindowActivate();
```

Description

This is WindowActivate, a member of class DWebBrowserEvents.

8.2.4.2.16 DWebBrowserEvents.WindowMove Method

Pascal

```
procedure WindowMove;
```

C++

```
__fastcall WindowMove();
```

Description

This is WindowMove, a member of class DWebBrowserEvents.

8.2.4.2.17 DWebBrowserEvents.WindowResize Method

Pascal

```
procedure WindowResize;
```

C++

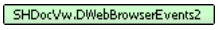
```
__fastcall WindowResize();
```

Description

This is WindowResize, a member of class DWebBrowserEvents.

8.2.5 DWebBrowserEvents2 Interface

Class Hierarchy



Pascal

```
DWebBrowserEvents2 = dispinterface;
```

C++

```
class DWebBrowserEvents2;
```

File

SHDocVw

Description

This is class SHDocVw.DWebBrowserEvents2.

8.2.5.1 DWebBrowserEvents2 Members

The following tables list the members exposed by DWebBrowserEvents2.

Public Methods

	Name	Description
	BeforeNavigate2 (see page 628)	This is BeforeNavigate2, a member of class DWebBrowserEvents2.
	ClientToHostWindow (see page 628)	This is ClientToHostWindow, a member of class DWebBrowserEvents2.
	CommandStateChange (see page 628)	This is CommandStateChange, a member of class DWebBrowserEvents2.
	DocumentComplete (see page 628)	This is DocumentComplete, a member of class DWebBrowserEvents2.
	DownloadBegin (see page 629)	This is DownloadBegin, a member of class DWebBrowserEvents2.
	DownloadComplete (see page 629)	This is DownloadComplete, a member of class DWebBrowserEvents2.
	FileDownload (see page 629)	This is FileDownload, a member of class DWebBrowserEvents2.
	NavigateComplete2 (see page 629)	This is NavigateComplete2, a member of class DWebBrowserEvents2.
	NavigateError (see page 629)	This is NavigateError, a member of class DWebBrowserEvents2.
	NewWindow2 (see page 630)	This is NewWindow2, a member of class DWebBrowserEvents2.
	OnFullScreen (see page 630)	This is OnFullScreen, a member of class DWebBrowserEvents2.
	OnMenuBar (see page 630)	This is OnMenuBar, a member of class DWebBrowserEvents2.
	OnQuit (see page 630)	This is OnQuit, a member of class DWebBrowserEvents2.
	OnStatusBar (see page 631)	This is OnStatusBar, a member of class DWebBrowserEvents2.
	OnTheaterMode (see page 631)	This is OnTheaterMode, a member of class DWebBrowserEvents2.

	OnToolBar (see page 631)	This is OnToolBar, a member of class DWebBrowserEvents2.
	OnVisible (see page 631)	This is OnVisible, a member of class DWebBrowserEvents2.
	PrintTemplateInstantiation (see page 631)	This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.
	PrintTemplateTeardown (see page 632)	This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.
	PrivacyImpactedStateChange (see page 632)	This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.
	ProgressChange (see page 632)	This is ProgressChange, a member of class DWebBrowserEvents2.
	PropertyChange (see page 632)	This is PropertyChange, a member of class DWebBrowserEvents2.
	SetSecureLockIcon (see page 633)	This is SetSecureLockIcon, a member of class DWebBrowserEvents2.
	StatusTextChange (see page 633)	This is StatusTextChange, a member of class DWebBrowserEvents2.
	TitleChange (see page 633)	This is TitleChange, a member of class DWebBrowserEvents2.
	UpdatePageStatus (see page 633)	This is UpdatePageStatus, a member of class DWebBrowserEvents2.
	WindowClosing (see page 633)	This is WindowClosing, a member of class DWebBrowserEvents2.
	WindowSetHeight (see page 634)	This is WindowSetHeight, a member of class DWebBrowserEvents2.
	WindowSetLeft (see page 634)	This is WindowSetLeft, a member of class DWebBrowserEvents2.
	WindowSetResizable (see page 634)	This is WindowSetResizable, a member of class DWebBrowserEvents2.
	WindowSetTop (see page 634)	This is WindowSetTop, a member of class DWebBrowserEvents2.
	WindowSetWidth (see page 635)	This is WindowSetWidth, a member of class DWebBrowserEvents2.

8.2.5.2 DWebBrowserEvents2 Methods

The methods of the DWebBrowserEvents2 class are listed here.

Public Methods

	Name	Description
	BeforeNavigate2 (see page 628)	This is BeforeNavigate2, a member of class DWebBrowserEvents2.
	ClientToHostWindow (see page 628)	This is ClientToHostWindow, a member of class DWebBrowserEvents2.
	CommandStateChange (see page 628)	This is CommandStateChange, a member of class DWebBrowserEvents2.
	DocumentComplete (see page 628)	This is DocumentComplete, a member of class DWebBrowserEvents2.
	DownloadBegin (see page 629)	This is DownloadBegin, a member of class DWebBrowserEvents2.
	DownloadComplete (see page 629)	This is DownloadComplete, a member of class DWebBrowserEvents2.
	FileDownload (see page 629)	This is FileDownload, a member of class DWebBrowserEvents2.
	NavigateComplete2 (see page 629)	This is NavigateComplete2, a member of class DWebBrowserEvents2.
	NavigateError (see page 629)	This is NavigateError, a member of class DWebBrowserEvents2.
	NewWindow2 (see page 630)	This is NewWindow2, a member of class DWebBrowserEvents2.
	OnFullScreen (see page 630)	This is OnFullScreen, a member of class DWebBrowserEvents2.
	OnMenuBar (see page 630)	This is OnMenuBar, a member of class DWebBrowserEvents2.
	OnQuit (see page 630)	This is OnQuit, a member of class DWebBrowserEvents2.
	OnStatusBar (see page 631)	This is OnStatusBar, a member of class DWebBrowserEvents2.
	OnTheaterMode (see page 631)	This is OnTheaterMode, a member of class DWebBrowserEvents2.
	OnToolBar (see page 631)	This is OnToolBar, a member of class DWebBrowserEvents2.
	OnVisible (see page 631)	This is OnVisible, a member of class DWebBrowserEvents2.
	PrintTemplateInstantiation (see page 631)	This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.
	PrintTemplateTeardown (see page 632)	This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.
	PrivacyImpactedStateChange (see page 632)	This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.
	ProgressChange (see page 632)	This is ProgressChange, a member of class DWebBrowserEvents2.
	PropertyChange (see page 632)	This is PropertyChange, a member of class DWebBrowserEvents2.
	SetSecureLockIcon (see page 633)	This is SetSecureLockIcon, a member of class DWebBrowserEvents2.
	StatusTextChange (see page 633)	This is StatusTextChange, a member of class DWebBrowserEvents2.
	TitleChange (see page 633)	This is TitleChange, a member of class DWebBrowserEvents2.
	UpdatePageStatus (see page 633)	This is UpdatePageStatus, a member of class DWebBrowserEvents2.
	WindowClosing (see page 633)	This is WindowClosing, a member of class DWebBrowserEvents2.
	WindowSetHeight (see page 634)	This is WindowSetHeight, a member of class DWebBrowserEvents2.
	WindowSetLeft (see page 634)	This is WindowSetLeft, a member of class DWebBrowserEvents2.

	WindowSetResizable (see page 634)	This is WindowSetResizable, a member of class DWebBrowserEvents2.
	WindowSetTop (see page 634)	This is WindowSetTop, a member of class DWebBrowserEvents2.
	WindowSetWidth (see page 635)	This is WindowSetWidth, a member of class DWebBrowserEvents2.

8.2.5.2.1 DWebBrowserEvents2.BeforeNavigate2 Method

Pascal

```
procedure BeforeNavigate2(const pDisp: IDispatch; const URL: OleVariant; const Flags:
OleVariant; const TargetFrameName: OleVariant; const PostData: OleVariant; const Headers:
OleVariant; var Cancel: WordBool);
```

C++

```
__fastcall BeforeNavigate2(const IDispatch pDisp, const OleVariant URL, const OleVariant
Flags, const OleVariant TargetFrameName, const OleVariant PostData, const OleVariant Headers,
WordBool Cancel);
```

Description

This is BeforeNavigate2, a member of class DWebBrowserEvents2.

8.2.5.2.2 DWebBrowserEvents2.ClientToHostWindow Method

Pascal

```
procedure ClientToHostWindow(var CX: Integer; var CY: Integer);
```

C++

```
__fastcall ClientToHostWindow(int CX, int CY);
```

Description

This is ClientToHostWindow, a member of class DWebBrowserEvents2.

8.2.5.2.3 DWebBrowserEvents2.CommandStateChange Method

Pascal

```
procedure CommandStateChange(Command: Integer; Enable: WordBool);
```

C++

```
__fastcall CommandStateChange(int Command, WordBool Enable);
```

Description

This is CommandStateChange, a member of class DWebBrowserEvents2.

8.2.5.2.4 DWebBrowserEvents2.DocumentComplete Method

Pascal

```
procedure DocumentComplete(const pDisp: IDispatch; var URL: OleVariant);
```

C++

```
__fastcall DocumentComplete(const IDispatch pDisp, OleVariant URL);
```

Description

This is DocumentComplete, a member of class DWebBrowserEvents2.

8.2.5.2.5 DWebBrowserEvents2.DownloadBegin Method

Pascal

```
procedure DownloadBegin;
```

C++

```
__fastcall DownloadBegin();
```

Description

This is DownloadBegin, a member of class DWebBrowserEvents2.

8.2.5.2.6 DWebBrowserEvents2.DownloadComplete Method

Pascal

```
procedure DownloadComplete;
```

C++

```
__fastcall DownloadComplete();
```

Description

This is DownloadComplete, a member of class DWebBrowserEvents2.

8.2.5.2.7 DWebBrowserEvents2.FileDownload Method

Pascal

```
procedure FileDownload(var Cancel: WordBool);
```

C++

```
__fastcall FileDownload(WordBool Cancel);
```

Description

This is FileDownload, a member of class DWebBrowserEvents2.

8.2.5.2.8 DWebBrowserEvents2.NavigateComplete2 Method

Pascal

```
procedure NavigateComplete2(const pDisp: IDispatch; var URL: OleVariant);
```

C++

```
__fastcall NavigateComplete2(const IDispatch pDisp, OleVariant URL);
```

Description

This is NavigateComplete2, a member of class DWebBrowserEvents2.

8.2.5.2.9 DWebBrowserEvents2.NavigateError Method

Pascal

```
procedure NavigateError(const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant;
var StatusCode: OleVariant; var Cancel: WordBool);
```

C++

```
__fastcall NavigateError(const IDispatch pDisp, OleVariant URL, OleVariant Frame, OleVariant
StatusCode, WordBool Cancel);
```

Description

This is NavigateError, a member of class DWebBrowserEvents2.

8.2.5.2.10 DWebBrowserEvents2.NewWindow2 Method

Pascal

```
procedure NewWindow2(var ppDisp: IDispatch; var Cancel: WordBool);
```

C++

```
__fastcall NewWindow2(IDispatch ppDisp, WordBool Cancel);
```

Description

This is NewWindow2, a member of class DWebBrowserEvents2.

8.2.5.2.11 DWebBrowserEvents2.OnFullScreen Method

Pascal

```
procedure OnFullScreen(FullScreen: WordBool);
```

C++

```
__fastcall OnFullScreen(WordBool FullScreen);
```

Description

This is OnFullScreen, a member of class DWebBrowserEvents2.

8.2.5.2.12 DWebBrowserEvents2.OnMenuBar Method

Pascal

```
procedure OnMenuBar(MenuBar: WordBool);
```

C++

```
__fastcall OnMenuBar(WordBool MenuBar);
```

Description

This is OnMenuBar, a member of class DWebBrowserEvents2.

8.2.5.2.13 DWebBrowserEvents2.OnQuit Method

Pascal

```
procedure OnQuit;
```

C++

```
__fastcall OnQuit();
```

Description

This is OnQuit, a member of class DWebBrowserEvents2.

8.2.5.2.14 DWebBrowserEvents2.OnStatusBar Method

Pascal

```
procedure OnStatusBar(StatusBar: WordBool);
```

C++

```
__fastcall OnStatusBar(WordBool StatusBar);
```

Description

This is OnStatusBar, a member of class DWebBrowserEvents2.

8.2.5.2.15 DWebBrowserEvents2.OnTheaterMode Method

Pascal

```
procedure OnTheaterMode(TheaterMode: WordBool);
```

C++

```
__fastcall OnTheaterMode(WordBool TheaterMode);
```

Description

This is OnTheaterMode, a member of class DWebBrowserEvents2.

8.2.5.2.16 DWebBrowserEvents2.OnToolBar Method

Pascal

```
procedure OnToolBar(ToolBar: WordBool);
```

C++

```
__fastcall OnToolBar(WordBool ToolBar);
```

Description

This is OnToolBar, a member of class DWebBrowserEvents2.

8.2.5.2.17 DWebBrowserEvents2.OnVisible Method

Pascal

```
procedure OnVisible(Visible: WordBool);
```

C++

```
__fastcall OnVisible(WordBool Visible);
```

Description

This is OnVisible, a member of class DWebBrowserEvents2.

8.2.5.2.18 DWebBrowserEvents2.PrintTemplateInstantiation Method

Pascal

```
procedure PrintTemplateInstantiation(const pDisp: IDispatch);
```

C++

```
__fastcall PrintTemplateInstantiation(const IDispatch pDisp);
```

Description

This is PrintTemplateInstantiation, a member of class DWebBrowserEvents2.

8.2.5.2.19 DWebBrowserEvents2.PrintTemplateTeardown Method

Pascal

```
procedure PrintTemplateTeardown(const pDisp: IDispatch);
```

C++

```
__fastcall PrintTemplateTeardown(const IDispatch pDisp);
```

Description

This is PrintTemplateTeardown, a member of class DWebBrowserEvents2.

8.2.5.2.20 DWebBrowserEvents2.PrivacyImpactedStateChange Method

Pascal

```
procedure PrivacyImpactedStateChange(bImpacted: WordBool);
```

C++

```
__fastcall PrivacyImpactedStateChange(WordBool bImpacted);
```

Description

This is PrivacyImpactedStateChange, a member of class DWebBrowserEvents2.

8.2.5.2.21 DWebBrowserEvents2.ProgressChange Method

Pascal

```
procedure ProgressChange(Progress: Integer; ProgressMax: Integer);
```

C++

```
__fastcall ProgressChange(int Progress, int ProgressMax);
```

Description

This is ProgressChange, a member of class DWebBrowserEvents2.

8.2.5.2.22 DWebBrowserEvents2.PropertyChange Method

Pascal

```
procedure PropertyChange(const szProperty: WideString);
```

C++

```
__fastcall PropertyChange(const BSTR szProperty);
```

Description

This is PropertyChange, a member of class DWebBrowserEvents2.

8.2.5.2.23 DWebBrowserEvents2.SetSecureLockIcon Method

Pascal

```
procedure SetSecureLockIcon(SecureLockIcon: Integer);
```

C++

```
__fastcall SetSecureLockIcon(int SecureLockIcon);
```

Description

This is SetSecureLockIcon, a member of class DWebBrowserEvents2.

8.2.5.2.24 DWebBrowserEvents2.StatusTextChange Method

Pascal

```
procedure StatusTextChange(const Text: WideString);
```

C++

```
__fastcall StatusTextChange(const BSTR Text);
```

Description

This is StatusTextChange, a member of class DWebBrowserEvents2.

8.2.5.2.25 DWebBrowserEvents2.TitleChange Method

Pascal

```
procedure TitleChange(const Text: WideString);
```

C++

```
__fastcall TitleChange(const BSTR Text);
```

Description

This is TitleChange, a member of class DWebBrowserEvents2.

8.2.5.2.26 DWebBrowserEvents2.UpdatePageStatus Method

Pascal

```
procedure UpdatePageStatus(const pDisp: IDispatch; var nPage: OleVariant; var fDone:
OleVariant);
```

C++

```
__fastcall UpdatePageStatus(const IDispatch pDisp, OleVariant nPage, OleVariant fDone);
```

Description

This is UpdatePageStatus, a member of class DWebBrowserEvents2.

8.2.5.2.27 DWebBrowserEvents2.WindowClosing Method

Pascal

```
procedure WindowClosing(IsChildWindow: WordBool; var Cancel: WordBool);
```

C++

```
__fastcall WindowClosing(WordBool IsChildWindow, WordBool Cancel);
```

Description

This is WindowClosing, a member of class DWebBrowserEvents2.

8.2.5.2.28 DWebBrowserEvents2.WindowSetHeight Method

Pascal

```
procedure WindowSetHeight(Height: Integer);
```

C++

```
__fastcall WindowSetHeight(int Height);
```

Description

This is WindowSetHeight, a member of class DWebBrowserEvents2.

8.2.5.2.29 DWebBrowserEvents2.WindowSetLeft Method

Pascal

```
procedure WindowSetLeft(Left: Integer);
```

C++

```
__fastcall WindowSetLeft(int Left);
```

Description

This is WindowSetLeft, a member of class DWebBrowserEvents2.

8.2.5.2.30 DWebBrowserEvents2.WindowSetResizable Method

Pascal

```
procedure WindowSetResizable(Resizable: WordBool);
```

C++

```
__fastcall WindowSetResizable(WordBool Resizable);
```

Description

This is WindowSetResizable, a member of class DWebBrowserEvents2.

8.2.5.2.31 DWebBrowserEvents2.WindowSetTop Method

Pascal

```
procedure WindowSetTop(Top: Integer);
```

C++

```
__fastcall WindowSetTop(int Top);
```

Description

This is WindowSetTop, a member of class DWebBrowserEvents2.

8.2.5.2.32 DWebBrowserEvents2.WindowSetWidth Method

Pascal

```
procedure WindowSetWidth(Width: Integer);
```

C++

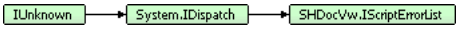
```
__fastcall WindowSetWidth(int Width);
```

Description

This is WindowSetWidth, a member of class DWebBrowserEvents2.

8.2.6 IScriptErrorList Interface

Class Hierarchy



Pascal

```
IScriptErrorList = interface(IDispatch);
```

C++

```
__interface IScriptErrorList : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IScriptErrorList.

8.2.6.1 IScriptErrorList Members

The following tables list the members exposed by IScriptErrorList.

Public Methods

	Name	Description
	advanceError (see page 636)	This is advanceError, a member of class IScriptErrorList.
	canAdvanceError (see page 636)	This is canAdvanceError, a member of class IScriptErrorList.
	canRetreatError (see page 637)	This is canRetreatError, a member of class IScriptErrorList.
	getAlwaysShowLockState (see page 637)	This is getAlwaysShowLockState, a member of class IScriptErrorList.
	getDetailsPaneOpen (see page 637)	This is getDetailsPaneOpen, a member of class IScriptErrorList.
	getErrorChar (see page 637)	This is getErrorChar, a member of class IScriptErrorList.
	getErrorCode (see page 637)	This is getErrorCode, a member of class IScriptErrorList.
	getErrorLine (see page 638)	This is getErrorLine, a member of class IScriptErrorList.
	getErrorMsg (see page 638)	This is getErrorMsg, a member of class IScriptErrorList.
	getErrorUrl (see page 638)	This is getErrorUrl, a member of class IScriptErrorList.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	getPerErrorDisplay (see page 638)	This is getPerErrorDisplay, a member of class IScriptErrorList.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.

	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	retreatError (see page 639)	This is retreatError, a member of class IScriptErrorList.
	setDetailsPaneOpen (see page 639)	This is setDetailsPaneOpen, a member of class IScriptErrorList.
	setPerErrorDisplay (see page 639)	This is setPerErrorDisplay, a member of class IScriptErrorList.

8.2.6.2 IScriptErrorList Methods

The methods of the IScriptErrorList class are listed here.

Public Methods

	Name	Description
	advanceError (see page 636)	This is advanceError, a member of class IScriptErrorList.
	canAdvanceError (see page 636)	This is canAdvanceError, a member of class IScriptErrorList.
	canRetreatError (see page 637)	This is canRetreatError, a member of class IScriptErrorList.
	getAlwaysShowLockState (see page 637)	This is getAlwaysShowLockState, a member of class IScriptErrorList.
	getDetailsPaneOpen (see page 637)	This is getDetailsPaneOpen, a member of class IScriptErrorList.
	getErrorChar (see page 637)	This is getErrorChar, a member of class IScriptErrorList.
	getErrorCode (see page 637)	This is getErrorCode, a member of class IScriptErrorList.
	getErrorLine (see page 638)	This is getErrorLine, a member of class IScriptErrorList.
	getErrorMsg (see page 638)	This is getErrorMsg, a member of class IScriptErrorList.
	getErrorUrl (see page 638)	This is getErrorUrl, a member of class IScriptErrorList.
	getPerErrorDisplay (see page 638)	This is getPerErrorDisplay, a member of class IScriptErrorList.
	retreatError (see page 639)	This is retreatError, a member of class IScriptErrorList.
	setDetailsPaneOpen (see page 639)	This is setDetailsPaneOpen, a member of class IScriptErrorList.
	setPerErrorDisplay (see page 639)	This is setPerErrorDisplay, a member of class IScriptErrorList.

8.2.6.2.1 IScriptErrorList.advanceError Method

Pascal

```
procedure advanceError; safecall;
```

C++

```
__fastcall advanceError();
```

Description

This is advanceError, a member of class IScriptErrorList.

8.2.6.2.2 IScriptErrorList.canAdvanceError Method

Pascal

```
function canAdvanceError: Integer; safecall;
```

C++

```
__fastcall int canAdvanceError();
```

Description

This is canAdvanceError, a member of class IScriptErrorList.

8.2.6.2.3 IScriptErrorList.canRetreatError Method

Pascal

```
function canRetreatError: Integer; safecall;
```

C++

```
__fastcall int canRetreatError();
```

Description

This is canRetreatError, a member of class IScriptErrorList.

8.2.6.2.4 IScriptErrorList.getAlwaysShowLockState Method

Pascal

```
function getAlwaysShowLockState: Integer; safecall;
```

C++

```
__fastcall int getAlwaysShowLockState();
```

Description

This is getAlwaysShowLockState, a member of class IScriptErrorList.

8.2.6.2.5 IScriptErrorList.getDetailsPaneOpen Method

Pascal

```
function getDetailsPaneOpen: Integer; safecall;
```

C++

```
__fastcall int getDetailsPaneOpen();
```

Description

This is getDetailsPaneOpen, a member of class IScriptErrorList.

8.2.6.2.6 IScriptErrorList.getErrorChar Method

Pascal

```
function getErrorChar: Integer; safecall;
```

C++

```
__fastcall int getErrorChar();
```

Description

This is getErrorChar, a member of class IScriptErrorList.

8.2.6.2.7 IScriptErrorList.getErrorCode Method

Pascal

```
function getErrorCode: Integer; safecall;
```

C++

```
__fastcall int getErrorCode();
```

Description

This is getErrorCode, a member of class IScriptErrorList.

8.2.6.2.8 IScriptErrorList.getErrorLine Method

Pascal

```
function getErrorLine: Integer; safecall;
```

C++

```
__fastcall int getErrorLine();
```

Description

This is getErrorLine, a member of class IScriptErrorList.

8.2.6.2.9 IScriptErrorList.getErrorMsg Method

Pascal

```
function getErrorMsg: WideString; safecall;
```

C++

```
__fastcall BSTR getErrorMsg();
```

Description

This is getErrorMsg, a member of class IScriptErrorList.

8.2.6.2.10 IScriptErrorList.getErrorUrl Method

Pascal

```
function getErrorUrl: WideString; safecall;
```

C++

```
__fastcall BSTR getErrorUrl();
```

Description

This is getErrorUrl, a member of class IScriptErrorList.

8.2.6.2.11 IScriptErrorList.getPerErrorDisplay Method

Pascal

```
function getPerErrorDisplay: Integer; safecall;
```

C++

```
__fastcall int getPerErrorDisplay();
```

Description

This is getPerErrorDisplay, a member of class IScriptErrorList.

8.2.6.2.12 IScriptErrorList.retreatError Method

Pascal

```
procedure retreatError; safecall;
```

C++

```
__fastcall retreatError();
```

Description

This is retreatError, a member of class IScriptErrorList.

8.2.6.2.13 IScriptErrorList.setDetailsPaneOpen Method

Pascal

```
procedure setDetailsPaneOpen(fDetailsPaneOpen: Integer); safecall;
```

C++

```
__fastcall setDetailsPaneOpen(int fDetailsPaneOpen);
```

Description

This is setDetailsPaneOpen, a member of class IScriptErrorList.

8.2.6.2.14 IScriptErrorList.setPerErrorDisplay Method

Pascal

```
procedure setPerErrorDisplay(fPerErrorDisplay: Integer); safecall;
```

C++

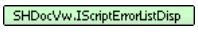
```
__fastcall setPerErrorDisplay(int fPerErrorDisplay);
```

Description

This is setPerErrorDisplay, a member of class IScriptErrorList.

8.2.7 IScriptErrorListDisp Interface

Class Hierarchy



Pascal

```
IScriptErrorListDisp = dispinterface;
```

C++

```
class IScriptErrorListDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IScriptErrorListDisp.

8.2.7.1 IScriptErrorListDisp Members

The following tables list the members exposed by IScriptErrorListDisp.

Public Methods

	Name	Description
	advanceError (see page 640)	This is advanceError, a member of class IScriptErrorListDisp.
	canAdvanceError (see page 641)	This is canAdvanceError, a member of class IScriptErrorListDisp.
	canRetreatError (see page 641)	This is canRetreatError, a member of class IScriptErrorListDisp.
	getAlwaysShowLockState (see page 641)	This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.
	getDetailsPaneOpen (see page 641)	This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.
	getErrorChar (see page 642)	This is getErrorChar, a member of class IScriptErrorListDisp.
	getErrorCode (see page 642)	This is getErrorCode, a member of class IScriptErrorListDisp.
	getErrorLine (see page 642)	This is getErrorLine, a member of class IScriptErrorListDisp.
	getErrorMsg (see page 642)	This is getErrorMsg, a member of class IScriptErrorListDisp.
	getErrorUrl (see page 642)	This is getErrorUrl, a member of class IScriptErrorListDisp.
	getPerErrorDisplay (see page 643)	This is getPerErrorDisplay, a member of class IScriptErrorListDisp.
	retreatError (see page 643)	This is retreatError, a member of class IScriptErrorListDisp.
	setDetailsPaneOpen (see page 643)	This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.
	setPerErrorDisplay (see page 643)	This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

8.2.7.2 IScriptErrorListDisp Methods

The methods of the IScriptErrorListDisp class are listed here.

Public Methods

	Name	Description
	advanceError (see page 640)	This is advanceError, a member of class IScriptErrorListDisp.
	canAdvanceError (see page 641)	This is canAdvanceError, a member of class IScriptErrorListDisp.
	canRetreatError (see page 641)	This is canRetreatError, a member of class IScriptErrorListDisp.
	getAlwaysShowLockState (see page 641)	This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.
	getDetailsPaneOpen (see page 641)	This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.
	getErrorChar (see page 642)	This is getErrorChar, a member of class IScriptErrorListDisp.
	getErrorCode (see page 642)	This is getErrorCode, a member of class IScriptErrorListDisp.
	getErrorLine (see page 642)	This is getErrorLine, a member of class IScriptErrorListDisp.
	getErrorMsg (see page 642)	This is getErrorMsg, a member of class IScriptErrorListDisp.
	getErrorUrl (see page 642)	This is getErrorUrl, a member of class IScriptErrorListDisp.
	getPerErrorDisplay (see page 643)	This is getPerErrorDisplay, a member of class IScriptErrorListDisp.
	retreatError (see page 643)	This is retreatError, a member of class IScriptErrorListDisp.
	setDetailsPaneOpen (see page 643)	This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.
	setPerErrorDisplay (see page 643)	This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

8.2.7.2.1 IScriptErrorListDisp.advanceError Method

Pascal

```
procedure advanceError;
```

C++

```
__fastcall advanceError();
```

Description

This is advanceError, a member of class IScriptErrorListDisp.

8.2.7.2.2 IScriptErrorListDisp.canAdvanceError Method

Pascal

```
function canAdvanceError: Integer;
```

C++

```
__fastcall int canAdvanceError();
```

Description

This is canAdvanceError, a member of class IScriptErrorListDisp.

8.2.7.2.3 IScriptErrorListDisp.canRetreatError Method

Pascal

```
function canRetreatError: Integer;
```

C++

```
__fastcall int canRetreatError();
```

Description

This is canRetreatError, a member of class IScriptErrorListDisp.

8.2.7.2.4 IScriptErrorListDisp.getAlwaysShowLockState Method

Pascal

```
function getAlwaysShowLockState: Integer;
```

C++

```
__fastcall int getAlwaysShowLockState();
```

Description

This is getAlwaysShowLockState, a member of class IScriptErrorListDisp.

8.2.7.2.5 IScriptErrorListDisp.getDetailsPaneOpen Method

Pascal

```
function getDetailsPaneOpen: Integer;
```

C++

```
__fastcall int getDetailsPaneOpen();
```

Description

This is getDetailsPaneOpen, a member of class IScriptErrorListDisp.

8.2.7.2.6 IScriptErrorListDisp.getErrorChar Method

Pascal

```
function getErrorChar: Integer;
```

C++

```
__fastcall int getErrorChar();
```

Description

This is getErrorChar, a member of class IScriptErrorListDisp.

8.2.7.2.7 IScriptErrorListDisp.getErrorCode Method

Pascal

```
function getErrorCode: Integer;
```

C++

```
__fastcall int getErrorCode();
```

Description

This is getErrorCode, a member of class IScriptErrorListDisp.

8.2.7.2.8 IScriptErrorListDisp.getErrorLine Method

Pascal

```
function getErrorLine: Integer;
```

C++

```
__fastcall int getErrorLine();
```

Description

This is getErrorLine, a member of class IScriptErrorListDisp.

8.2.7.2.9 IScriptErrorListDisp.getErrorMsg Method

Pascal

```
function getErrorMsg: WideString;
```

C++

```
__fastcall BSTR getErrorMsg();
```

Description

This is getErrorMsg, a member of class IScriptErrorListDisp.

8.2.7.2.10 IScriptErrorListDisp.getErrorUrl Method

Pascal

```
function getErrorUrl: WideString;
```

C++

```
__fastcall BSTR getErrorUrl();
```

Description

This is getErrorUrl, a member of class IScriptErrorListDisp.

8.2.7.2.11 IScriptErrorListDisp.getPerErrorDisplay Method

Pascal

```
function getPerErrorDisplay: Integer;
```

C++

```
__fastcall int getPerErrorDisplay();
```

Description

This is getPerErrorDisplay, a member of class IScriptErrorListDisp.

8.2.7.2.12 IScriptErrorListDisp.retreatError Method

Pascal

```
procedure retreatError;
```

C++

```
__fastcall retreatError();
```

Description

This is retreatError, a member of class IScriptErrorListDisp.

8.2.7.2.13 IScriptErrorListDisp.setDetailsPaneOpen Method

Pascal

```
procedure setDetailsPaneOpen(fDetailsPaneOpen: Integer);
```

C++

```
__fastcall setDetailsPaneOpen(int fDetailsPaneOpen);
```

Description

This is setDetailsPaneOpen, a member of class IScriptErrorListDisp.

8.2.7.2.14 IScriptErrorListDisp.setPerErrorDisplay Method

Pascal

```
procedure setPerErrorDisplay(fPerErrorDisplay: Integer);
```

C++

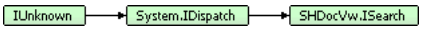
```
__fastcall setPerErrorDisplay(int fPerErrorDisplay);
```

Description

This is setPerErrorDisplay, a member of class IScriptErrorListDisp.

8.2.8 ISearch Interface

Class Hierarchy



Pascal

```
ISearch = interface(IDispatch);
```

C++

```
__interface ISearch : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearch.

8.2.8.1 ISearch Members

The following tables list the members exposed by ISearch.

Public Methods

	Name	Description
	Get_Id (see page 645)	This is Get_Id, a member of class ISearch.
	Get_Title (see page 645)	This is Get_Title, a member of class ISearch.
	Get_URL (see page 645)	This is Get_URL, a member of class ISearch.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.

Public Properties

	Name	Description
R	Id (see page 645)	This is Id, a member of class ISearch.
R	Title (see page 646)	This is Title, a member of class ISearch.
R	URL (see page 646)	This is URL, a member of class ISearch.

8.2.8.2 ISearch Methods

The methods of the ISearch class are listed here.

Public Methods

	Name	Description
	Get_Id (see page 645)	This is Get_Id, a member of class ISearch.
	Get_Title (see page 645)	This is Get_Title, a member of class ISearch.
	Get_URL (see page 645)	This is Get_URL, a member of class ISearch.

8.2.8.2.1 ISearch.Get_Id Method

Pascal

```
function Get_Id: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Id();
```

Description

This is Get_Id, a member of class ISearch.

8.2.8.2.2 ISearch.Get_Title Method

Pascal

```
function Get_Title: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Title();
```

Description

This is Get_Title, a member of class ISearch.

8.2.8.2.3 ISearch.Get_URL Method

Pascal

```
function Get_URL: WideString; safecall;
```

C++

```
__fastcall BSTR Get_URL();
```

Description

This is Get_URL, a member of class ISearch.

8.2.8.3 ISearch Properties

The properties of the ISearch class are listed here.

Public Properties

	Name	Description
	Id (see page 645)	This is Id, a member of class ISearch.
	Title (see page 646)	This is Title, a member of class ISearch.
	URL (see page 646)	This is URL, a member of class ISearch.

8.2.8.3.1 ISearch.Id Property

Pascal

```
property Id: WideString;
```

C++

```
__property BSTR Id;
```

Description

This is Id, a member of class ISearch.

8.2.8.3.2 ISearch.Title Property

Pascal

```
property Title: WideString;
```

C++

```
__property BSTR Title;
```

Description

This is Title, a member of class ISearch.

8.2.8.3.3 ISearch.URL Property

Pascal

```
property URL: WideString;
```

C++

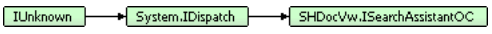
```
__property BSTR URL;
```

Description

This is URL, a member of class ISearch.

8.2.9 ISearchAssistantOC Interface

Class Hierarchy



Pascal

```
ISearchAssistantOC = interface(IDispatch);
```

C++

```
__interface ISearchAssistantOC : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC.

8.2.9.1 ISearchAssistantOC Members

The following tables list the members exposed by ISearchAssistantOC.

Public Methods

	Name	Description
	AddNextMenuItem (see page 648)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 648)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 649)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 649)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 649)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 649)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 649)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 650)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 650)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 650)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 650)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 651)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 651)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_Version (see page 651)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetProperty (see page 651)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 651)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	IsRestricted (see page 652)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 652)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 652)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 652)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 653)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 653)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 653)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 653)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 653)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 654)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 654)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 654)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 655)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 655)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 655)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 655)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 656)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 656)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	Version (see page 656)	This is Version, a member of class ISearchAssistantOC.

8.2.9.2 ISearchAssistantOC Methods

The methods of the ISearchAssistantOC class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 648)	This is AddNextMenuItem, a member of class ISearchAssistantOC.

	EncodeString (see page 648)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 649)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 649)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 649)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 649)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 649)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 650)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 650)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 650)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 650)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 651)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 651)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_Version (see page 651)	This is Get_Version, a member of class ISearchAssistantOC.
	GetProperty (see page 651)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 651)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	IsRestricted (see page 652)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 652)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 652)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 652)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 653)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 653)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 653)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 653)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 653)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 654)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 654)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

8.2.9.2.1 ISearchAssistantOC.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer); safecall;
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC.

8.2.9.2.2 ISearchAssistantOC.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString; safecall;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC.

8.2.9.2.3 ISearchAssistantOC.FindComputer Method

Pascal

```
procedure FindComputer; safecall;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC.

8.2.9.2.4 ISearchAssistantOC.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders; safecall;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC.

8.2.9.2.5 ISearchAssistantOC.FindOnWeb Method

Pascal

```
procedure FindOnWeb; safecall;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC.

8.2.9.2.6 ISearchAssistantOC.FindPeople Method

Pascal

```
procedure FindPeople; safecall;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC.

8.2.9.2.7 ISearchAssistantOC.FindPrinter Method

Pascal

```
procedure FindPrinter; safecall;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC.

8.2.9.2.8 ISearchAssistantOC.Get_ASProvider Method

Pascal

```
function Get_ASProvider: WideString; safecall;
```

C++

```
__fastcall BSTR Get_ASProvider();
```

Description

This is Get_ASProvider, a member of class ISearchAssistantOC.

8.2.9.2.9 ISearchAssistantOC.Get_ASSetting Method

Pascal

```
function Get_ASSetting: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_ASSetting();
```

Description

This is Get_ASSetting, a member of class ISearchAssistantOC.

8.2.9.2.10 ISearchAssistantOC.Get_InWebFolder Method

Pascal

```
function Get_InWebFolder: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_InWebFolder();
```

Description

This is Get_InWebFolder, a member of class ISearchAssistantOC.

8.2.9.2.11 ISearchAssistantOC.Get_SearchAssistantDefault Method

Pascal

```
function Get_SearchAssistantDefault: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SearchAssistantDefault();
```

Description

This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.

8.2.9.2.12 ISearchAssistantOC.Get_Searches Method

Pascal

```
function Get_Searches: ISearches; safecall;
```

C++

```
__fastcall ISearches Get_Searches();
```

Description

This is Get_Searches, a member of class ISearchAssistantOC.

8.2.9.2.13 ISearchAssistantOC.Get_ShellFeaturesEnabled Method

Pascal

```
function Get_ShellFeaturesEnabled: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_ShellFeaturesEnabled();
```

Description

This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.

8.2.9.2.14 ISearchAssistantOC.Get_Version Method

Pascal

```
function Get_Version: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_Version();
```

Description

This is Get_Version, a member of class ISearchAssistantOC.

8.2.9.2.15 ISearchAssistantOC.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString; safecall;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC.

8.2.9.2.16 ISearchAssistantOC.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
safecall;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC.

8.2.9.2.17 ISearchAssistantOC.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool; safecall;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC.

8.2.9.2.18 ISearchAssistantOC.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch; safecall;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.

8.2.9.2.19 ISearchAssistantOC.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate; safecall;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC.

8.2.9.2.20 ISearchAssistantOC.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged; safecall;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.

8.2.9.2.21 ISearchAssistantOC.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString); safecall;
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC.

8.2.9.2.22 ISearchAssistantOC.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString); safecall;
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC.

8.2.9.2.23 ISearchAssistantOC.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu; safecall;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC.

8.2.9.2.24 ISearchAssistantOC.Set_ASProvider Method

Pascal

```
procedure Set_ASProvider(const pProvider: WideString); safecall;
```

C++

```
__fastcall Set_ASProvider(const BSTR pProvider);
```

Description

This is Set_ASProvider, a member of class ISearchAssistantOC.

8.2.9.2.25 ISearchAssistantOC.Set_ASSetting Method

Pascal

```
procedure Set_ASSetting(pSetting: SYSINT); safecall;
```

C++

```
__fastcall Set_ASSetting(SYSINT pSetting);
```

Description

This is Set_ASSetting, a member of class ISearchAssistantOC.

8.2.9.2.26 ISearchAssistantOC.Set_EventHandled Method

Pascal

```
procedure Set_EventHandled(Param1: WordBool); safecall;
```

C++

```
__fastcall Set_EventHandled(WordBool Param1);
```

Description

This is Set_EventHandled, a member of class ISearchAssistantOC.

8.2.9.2.27 ISearchAssistantOC.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString); safecall;
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

8.2.9.3 ISearchAssistantOC Properties

The properties of the ISearchAssistantOC class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 654)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 655)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 655)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 655)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 655)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 656)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 656)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	Version (see page 656)	This is Version, a member of class ISearchAssistantOC.

8.2.9.3.1 ISearchAssistantOC.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC.

8.2.9.3.2 ISearchAssistantOC.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC.

8.2.9.3.3 ISearchAssistantOC.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC.

8.2.9.3.4 ISearchAssistantOC.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC.

8.2.9.3.5 ISearchAssistantOC.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC.

8.2.9.3.6 ISearchAssistantOC.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC.

8.2.9.3.7 ISearchAssistantOC.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.

8.2.9.3.8 ISearchAssistantOC.Version Property

Pascal

```
property Version: SYSINT;
```

C++

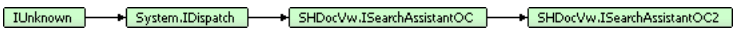
```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC.

8.2.10 ISearchAssistantOC2 Interface

Class Hierarchy



Pascal

```
ISearchAssistantOC2 = interface(ISearchAssistantOC);
```

C++

```
__interface ISearchAssistantOC2 : public ISearchAssistantOC;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC2.

8.2.10.1 ISearchAssistantOC2 Members

The following tables list the members exposed by ISearchAssistantOC2.

Public Methods

	Name	Description
	AddNextMenuItem (see page 648)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 648)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 649)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 649)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 649)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 649)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 649)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 650)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 650)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 650)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 650)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_Searches (see page 651)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 651)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_ShowFindPrinter (see page 658)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.
	Get_Version (see page 651)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetProperty (see page 651)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 651)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	IsRestricted (see page 652)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 652)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 652)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 652)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 653)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 653)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 653)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 653)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 653)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 654)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	SetDefaultSearchUrl (see page 654)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 654)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 655)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 655)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 655)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 655)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	Searches (see page 656)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 656)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	ShowFindPrinter (see page 658)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.
R	Version (see page 656)	This is Version, a member of class ISearchAssistantOC.

8.2.10.2 ISearchAssistantOC2 Methods

The methods of the ISearchAssistantOC2 class are listed here.

Public Methods

	Name	Description
	Get_ShowFindPrinter (see page 658)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.

8.2.10.2.1 ISearchAssistantOC2.Get_ShowFindPrinter Method

Pascal

```
function Get_ShowFindPrinter: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_ShowFindPrinter();
```

Description

This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.

8.2.10.3 ISearchAssistantOC2 Properties

The properties of the ISearchAssistantOC2 class are listed here.

Public Properties

	Name	Description
	ShowFindPrinter (see page 658)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.

8.2.10.3.1 ISearchAssistantOC2.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC2.

8.2.11 ISearchAssistantOC2Disp Interface

Class Hierarchy

SHDocVw.ISearchAssistantOC2Disp

Pascal

```
ISearchAssistantOC2Disp = dispinterface;
```

C++

```
class ISearchAssistantOC2Disp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC2Disp.

8.2.11.1 ISearchAssistantOC2Disp Members

The following tables list the members exposed by ISearchAssistantOC2Disp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 660)	This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.
	EncodeString (see page 660)	This is EncodeString, a member of class ISearchAssistantOC2Disp.
	FindComputer (see page 660)	This is FindComputer, a member of class ISearchAssistantOC2Disp.
	FindFilesOrFolders (see page 661)	This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.
	FindOnWeb (see page 661)	This is FindOnWeb, a member of class ISearchAssistantOC2Disp.
	FindPeople (see page 661)	This is FindPeople, a member of class ISearchAssistantOC2Disp.
	FindPrinter (see page 661)	This is FindPrinter, a member of class ISearchAssistantOC2Disp.
	GetProperty (see page 662)	This is GetProperty, a member of class ISearchAssistantOC2Disp.
	GetSearchAssistantURL (see page 662)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.
	IsRestricted (see page 662)	This is IsRestricted, a member of class ISearchAssistantOC2Disp.
	NavigateToDefaultSearch (see page 662)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.
	NETDetectNextNavigate (see page 662)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.
	NotifySearchSettingsChanged (see page 663)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.
	PutFindText (see page 663)	This is PutFindText, a member of class ISearchAssistantOC2Disp.
	PutProperty (see page 663)	This is PutProperty, a member of class ISearchAssistantOC2Disp.
	ResetNextMenu (see page 663)	This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.
	SetDefaultSearchUrl (see page 664)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

Public Properties

	Name	Description
	ASProvider (see page 664)	This is ASProvider, a member of class ISearchAssistantOC2Disp.
	ASSetting (see page 664)	This is ASSetting, a member of class ISearchAssistantOC2Disp.
	EventHandled (see page 665)	This is EventHandled, a member of class ISearchAssistantOC2Disp.
	InWebFolder (see page 665)	This is InWebFolder, a member of class ISearchAssistantOC2Disp.
	SearchAssistantDefault (see page 665)	This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.
	Searches (see page 665)	This is Searches, a member of class ISearchAssistantOC2Disp.
	ShellFeaturesEnabled (see page 665)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.
	ShowFindPrinter (see page 666)	This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.
	Version (see page 666)	This is Version, a member of class ISearchAssistantOC2Disp.

8.2.11.2 ISearchAssistantOC2Disp Methods

The methods of the ISearchAssistantOC2Disp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 660)	This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.
	EncodeString (see page 660)	This is EncodeString, a member of class ISearchAssistantOC2Disp.
	FindComputer (see page 660)	This is FindComputer, a member of class ISearchAssistantOC2Disp.
	FindFilesOrFolders (see page 661)	This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.
	FindOnWeb (see page 661)	This is FindOnWeb, a member of class ISearchAssistantOC2Disp.
	FindPeople (see page 661)	This is FindPeople, a member of class ISearchAssistantOC2Disp.
	FindPrinter (see page 661)	This is FindPrinter, a member of class ISearchAssistantOC2Disp.
	GetProperty (see page 662)	This is GetProperty, a member of class ISearchAssistantOC2Disp.
	GetSearchAssistantURL (see page 662)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.
	IsRestricted (see page 662)	This is IsRestricted, a member of class ISearchAssistantOC2Disp.
	NavigateToDefaultSearch (see page 662)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.
	NETDetectNextNavigate (see page 662)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.
	NotifySearchSettingsChanged (see page 663)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.
	PutFindText (see page 663)	This is PutFindText, a member of class ISearchAssistantOC2Disp.
	PutProperty (see page 663)	This is PutProperty, a member of class ISearchAssistantOC2Disp.
	ResetNextMenu (see page 663)	This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.
	SetDefaultSearchUrl (see page 664)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

8.2.11.2.1 ISearchAssistantOC2Disp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC2Disp.

8.2.11.2.2 ISearchAssistantOC2Disp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC2Disp.

8.2.11.2.3 ISearchAssistantOC2Disp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC2Disp.

8.2.11.2.4 ISearchAssistantOC2Disp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC2Disp.

8.2.11.2.5 ISearchAssistantOC2Disp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC2Disp.

8.2.11.2.6 ISearchAssistantOC2Disp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC2Disp.

8.2.11.2.7 ISearchAssistantOC2Disp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC2Disp.

8.2.11.2.8 ISearchAssistantOC2Disp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC2Disp.

8.2.11.2.9 ISearchAssistantOC2Disp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC2Disp.

8.2.11.2.10 ISearchAssistantOC2Disp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC2Disp.

8.2.11.2.11 ISearchAssistantOC2Disp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC2Disp.

8.2.11.2.12 ISearchAssistantOC2Disp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC2Disp.

8.2.11.2.13 ISearchAssistantOC2Disp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC2Disp.

8.2.11.2.14 ISearchAssistantOC2Disp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC2Disp.

8.2.11.2.15 ISearchAssistantOC2Disp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC2Disp.

8.2.11.2.16 ISearchAssistantOC2Disp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC2Disp.

8.2.11.2.17 ISearchAssistantOC2Disp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC2Disp.

8.2.11.3 ISearchAssistantOC2Disp Properties

The properties of the ISearchAssistantOC2Disp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 664)	This is ASProvider, a member of class ISearchAssistantOC2Disp.
	ASSetting (see page 664)	This is ASSetting, a member of class ISearchAssistantOC2Disp.
	EventHandled (see page 665)	This is EventHandled, a member of class ISearchAssistantOC2Disp.
	InWebFolder (see page 665)	This is InWebFolder, a member of class ISearchAssistantOC2Disp.
	SearchAssistantDefault (see page 665)	This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.
	Searches (see page 665)	This is Searches, a member of class ISearchAssistantOC2Disp.
	ShellFeaturesEnabled (see page 665)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.
	ShowFindPrinter (see page 666)	This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.
	Version (see page 666)	This is Version, a member of class ISearchAssistantOC2Disp.

8.2.11.3.1 ISearchAssistantOC2Disp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC2Disp.

8.2.11.3.2 ISearchAssistantOC2Disp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC2Disp.

8.2.11.3.3 ISearchAssistantOC2Disp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC2Disp.

8.2.11.3.4 ISearchAssistantOC2Disp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC2Disp.

8.2.11.3.5 ISearchAssistantOC2Disp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC2Disp.

8.2.11.3.6 ISearchAssistantOC2Disp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC2Disp.

8.2.11.3.7 ISearchAssistantOC2Disp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC2Disp.

8.2.11.3.8 ISearchAssistantOC2Disp.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC2Disp.

8.2.11.3.9 ISearchAssistantOC2Disp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

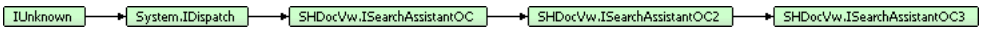
```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC2Disp.

8.2.12 ISearchAssistantOC3 Interface

Class Hierarchy



Pascal

```
ISearchAssistantOC3 = interface(ISearchAssistantOC2);
```

C++

```
__interface ISearchAssistantOC3 : public ISearchAssistantOC2;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC3.

8.2.12.1 ISearchAssistantOC3 Members

The following tables list the members exposed by ISearchAssistantOC3.

Public Methods

	Name	Description
	AddNextMenuItem (see page 648)	This is AddNextMenuItem, a member of class ISearchAssistantOC.
	EncodeString (see page 648)	This is EncodeString, a member of class ISearchAssistantOC.
	FindComputer (see page 649)	This is FindComputer, a member of class ISearchAssistantOC.
	FindFilesOrFolders (see page 649)	This is FindFilesOrFolders, a member of class ISearchAssistantOC.
	FindOnWeb (see page 649)	This is FindOnWeb, a member of class ISearchAssistantOC.
	FindPeople (see page 649)	This is FindPeople, a member of class ISearchAssistantOC.
	FindPrinter (see page 649)	This is FindPrinter, a member of class ISearchAssistantOC.
	Get_ASProvider (see page 650)	This is Get_ASProvider, a member of class ISearchAssistantOC.
	Get_ASSetting (see page 650)	This is Get_ASSetting, a member of class ISearchAssistantOC.
	Get_InWebFolder (see page 650)	This is Get_InWebFolder, a member of class ISearchAssistantOC.
	Get_SearchAssistantDefault (see page 650)	This is Get_SearchAssistantDefault, a member of class ISearchAssistantOC.
	Get_SearchCompanionAvailable (see page 668)	This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	Get_Searches (see page 651)	This is Get_Searches, a member of class ISearchAssistantOC.
	Get_ShellFeaturesEnabled (see page 651)	This is Get_ShellFeaturesEnabled, a member of class ISearchAssistantOC.
	Get_ShowFindPrinter (see page 658)	This is Get_ShowFindPrinter, a member of class ISearchAssistantOC2.
	Get_UseSearchCompanion (see page 668)	This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.
	Get_Version (see page 651)	This is Get_Version, a member of class ISearchAssistantOC.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetProperty (see page 651)	This is GetProperty, a member of class ISearchAssistantOC.
	GetSearchAssistantURL (see page 651)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	IsRestricted (see page 652)	This is IsRestricted, a member of class ISearchAssistantOC.
	NavigateToDefaultSearch (see page 652)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC.
	NETDetectNextNavigate (see page 652)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC.
	NotifySearchSettingsChanged (see page 652)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC.
	PutFindText (see page 653)	This is PutFindText, a member of class ISearchAssistantOC.
	PutProperty (see page 653)	This is PutProperty, a member of class ISearchAssistantOC.
	ResetNextMenu (see page 653)	This is ResetNextMenu, a member of class ISearchAssistantOC.
	Set_ASProvider (see page 653)	This is Set_ASProvider, a member of class ISearchAssistantOC.
	Set_ASSetting (see page 653)	This is Set_ASSetting, a member of class ISearchAssistantOC.
	Set_EventHandled (see page 654)	This is Set_EventHandled, a member of class ISearchAssistantOC.
	Set_UseSearchCompanion (see page 668)	This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.
	SetDefaultSearchUrl (see page 654)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC.

Public Properties

	Name	Description
	ASProvider (see page 654)	This is ASProvider, a member of class ISearchAssistantOC.
	ASSetting (see page 655)	This is ASSetting, a member of class ISearchAssistantOC.
W	EventHandled (see page 655)	This is EventHandled, a member of class ISearchAssistantOC.
R	InWebFolder (see page 655)	This is InWebFolder, a member of class ISearchAssistantOC.
R	SearchAssistantDefault (see page 655)	This is SearchAssistantDefault, a member of class ISearchAssistantOC.
R	SearchCompanionAvailable (see page 669)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.
R	Searches (see page 656)	This is Searches, a member of class ISearchAssistantOC.
R	ShellFeaturesEnabled (see page 656)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC.
R	ShowFindPrinter (see page 658)	This is ShowFindPrinter, a member of class ISearchAssistantOC2.
	UseSearchCompanion (see page 669)	This is UseSearchCompanion, a member of class ISearchAssistantOC3.
R	Version (see page 656)	This is Version, a member of class ISearchAssistantOC.

8.2.12.2 ISearchAssistantOC3 Methods

The methods of the ISearchAssistantOC3 class are listed here.

Public Methods

	Name	Description
	Get_SearchCompanionAvailable (see page 668)	This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	Get_UseSearchCompanion (see page 668)	This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.
	Set_UseSearchCompanion (see page 668)	This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.

8.2.12.2.1 ISearchAssistantOC3.Get_SearchCompanionAvailable Method

Pascal

```
function Get_SearchCompanionAvailable: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SearchCompanionAvailable();
```

Description

This is Get_SearchCompanionAvailable, a member of class ISearchAssistantOC3.

8.2.12.2.2 ISearchAssistantOC3.Get_UseSearchCompanion Method

Pascal

```
function Get_UseSearchCompanion: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_UseSearchCompanion();
```

Description

This is Get_UseSearchCompanion, a member of class ISearchAssistantOC3.

8.2.12.2.3 ISearchAssistantOC3.Set_UseSearchCompanion Method

Pascal

```
procedure Set_UseSearchCompanion(pbUseSC: WordBool); safecall;
```

C++

```
__fastcall Set_UseSearchCompanion(WordBool pbUseSC);
```

Description

This is Set_UseSearchCompanion, a member of class ISearchAssistantOC3.

8.2.12.3 ISearchAssistantOC3 Properties

The properties of the ISearchAssistantOC3 class are listed here.

Public Properties

	Name	Description
	SearchCompanionAvailable (see page 669)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.
	UseSearchCompanion (see page 669)	This is UseSearchCompanion, a member of class ISearchAssistantOC3.

8.2.12.3.1 ISearchAssistantOC3.SearchCompanionAvailable Property

Pascal

```
property SearchCompanionAvailable: WordBool;
```

C++

```
__property WordBool SearchCompanionAvailable;
```

Description

This is SearchCompanionAvailable, a member of class ISearchAssistantOC3.

8.2.12.3.2 ISearchAssistantOC3.UseSearchCompanion Property

Pascal

```
property UseSearchCompanion: WordBool;
```

C++

```
__property WordBool UseSearchCompanion;
```

Description

This is UseSearchCompanion, a member of class ISearchAssistantOC3.

8.2.13 ISearchAssistantOC3Disp Interface

Class Hierarchy

SHDocVw.ISearchAssistantOC3Disp

Pascal

```
ISearchAssistantOC3Disp = dispinterface;
```

C++

```
class ISearchAssistantOC3Disp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOC3Disp.

8.2.13.1 ISearchAssistantOC3Disp Members

The following tables list the members exposed by ISearchAssistantOC3Disp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 671)	This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.
	EncodeString (see page 671)	This is EncodeString, a member of class ISearchAssistantOC3Disp.
	FindComputer (see page 671)	This is FindComputer, a member of class ISearchAssistantOC3Disp.
	FindFilesOrFolders (see page 671)	This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.
	FindOnWeb (see page 672)	This is FindOnWeb, a member of class ISearchAssistantOC3Disp.
	FindPeople (see page 672)	This is FindPeople, a member of class ISearchAssistantOC3Disp.
	FindPrinter (see page 672)	This is FindPrinter, a member of class ISearchAssistantOC3Disp.
	GetProperty (see page 672)	This is GetProperty, a member of class ISearchAssistantOC3Disp.
	GetSearchAssistantURL (see page 672)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.
	IsRestricted (see page 673)	This is IsRestricted, a member of class ISearchAssistantOC3Disp.
	NavigateToDefaultSearch (see page 673)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.
	NETDetectNextNavigate (see page 673)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.
	NotifySearchSettingsChanged (see page 673)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.
	PutFindText (see page 674)	This is PutFindText, a member of class ISearchAssistantOC3Disp.
	PutProperty (see page 674)	This is PutProperty, a member of class ISearchAssistantOC3Disp.
	ResetNextMenu (see page 674)	This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.
	SetDefaultSearchUrl (see page 674)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

Public Properties

	Name	Description
	ASProvider (see page 675)	This is ASProvider, a member of class ISearchAssistantOC3Disp.
	ASSetting (see page 675)	This is ASSetting, a member of class ISearchAssistantOC3Disp.
	EventHandled (see page 675)	This is EventHandled, a member of class ISearchAssistantOC3Disp.
	InWebFolder (see page 676)	This is InWebFolder, a member of class ISearchAssistantOC3Disp.
	SearchAssistantDefault (see page 676)	This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.
	SearchCompanionAvailable (see page 676)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.
	Searches (see page 676)	This is Searches, a member of class ISearchAssistantOC3Disp.
	ShellFeaturesEnabled (see page 676)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.
	ShowFindPrinter (see page 677)	This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.
	UseSearchCompanion (see page 677)	This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.
	Version (see page 677)	This is Version, a member of class ISearchAssistantOC3Disp.

8.2.13.2 ISearchAssistantOC3Disp Methods

The methods of the ISearchAssistantOC3Disp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 671)	This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.
	EncodeString (see page 671)	This is EncodeString, a member of class ISearchAssistantOC3Disp.
	FindComputer (see page 671)	This is FindComputer, a member of class ISearchAssistantOC3Disp.
	FindFilesOrFolders (see page 671)	This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.
	FindOnWeb (see page 672)	This is FindOnWeb, a member of class ISearchAssistantOC3Disp.
	FindPeople (see page 672)	This is FindPeople, a member of class ISearchAssistantOC3Disp.
	FindPrinter (see page 672)	This is FindPrinter, a member of class ISearchAssistantOC3Disp.
	GetProperty (see page 672)	This is GetProperty, a member of class ISearchAssistantOC3Disp.
	GetSearchAssistantURL (see page 672)	This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.
	IsRestricted (see page 673)	This is IsRestricted, a member of class ISearchAssistantOC3Disp.
	NavigateToDefaultSearch (see page 673)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.

	NETDetectNextNavigate (see page 673)	This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.
	NotifySearchSettingsChanged (see page 673)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.
	PutFindText (see page 674)	This is PutFindText, a member of class ISearchAssistantOC3Disp.
	PutProperty (see page 674)	This is PutProperty, a member of class ISearchAssistantOC3Disp.
	ResetNextMenu (see page 674)	This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.
	SetDefaultSearchUrl (see page 674)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

8.2.13.2.1 ISearchAssistantOC3Disp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOC3Disp.

8.2.13.2.2 ISearchAssistantOC3Disp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOC3Disp.

8.2.13.2.3 ISearchAssistantOC3Disp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOC3Disp.

8.2.13.2.4 ISearchAssistantOC3Disp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOC3Disp.

8.2.13.2.5 ISearchAssistantOC3Disp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOC3Disp.

8.2.13.2.6 ISearchAssistantOC3Disp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOC3Disp.

8.2.13.2.7 ISearchAssistantOC3Disp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOC3Disp.

8.2.13.2.8 ISearchAssistantOC3Disp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOC3Disp.

8.2.13.2.9 ISearchAssistantOC3Disp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOC3Disp.

8.2.13.2.10 ISearchAssistantOC3Disp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOC3Disp.

8.2.13.2.11 ISearchAssistantOC3Disp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOC3Disp.

8.2.13.2.12 ISearchAssistantOC3Disp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOC3Disp.

8.2.13.2.13 ISearchAssistantOC3Disp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOC3Disp.

8.2.13.2.14 ISearchAssistantOC3Disp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOC3Disp.

8.2.13.2.15 ISearchAssistantOC3Disp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOC3Disp.

8.2.13.2.16 ISearchAssistantOC3Disp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOC3Disp.

8.2.13.2.17 ISearchAssistantOC3Disp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOC3Disp.

8.2.13.3 ISearchAssistantOC3Disp Properties

The properties of the ISearchAssistantOC3Disp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 675)	This is ASProvider, a member of class ISearchAssistantOC3Disp.
	ASSetting (see page 675)	This is ASSetting, a member of class ISearchAssistantOC3Disp.
	EventHandled (see page 675)	This is EventHandled, a member of class ISearchAssistantOC3Disp.
	InWebFolder (see page 676)	This is InWebFolder, a member of class ISearchAssistantOC3Disp.
	SearchAssistantDefault (see page 676)	This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.
	SearchCompanionAvailable (see page 676)	This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.
	Searches (see page 676)	This is Searches, a member of class ISearchAssistantOC3Disp.
	ShellFeaturesEnabled (see page 676)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.
	ShowFindPrinter (see page 677)	This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.
	UseSearchCompanion (see page 677)	This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.
	Version (see page 677)	This is Version, a member of class ISearchAssistantOC3Disp.

8.2.13.3.1 ISearchAssistantOC3Disp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOC3Disp.

8.2.13.3.2 ISearchAssistantOC3Disp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOC3Disp.

8.2.13.3.3 ISearchAssistantOC3Disp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOC3Disp.

8.2.13.3.4 ISearchAssistantOC3Disp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOC3Disp.

8.2.13.3.5 ISearchAssistantOC3Disp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOC3Disp.

8.2.13.3.6 ISearchAssistantOC3Disp.SearchCompanionAvailable Property

Pascal

```
property SearchCompanionAvailable: WordBool;
```

C++

```
__property WordBool SearchCompanionAvailable;
```

Description

This is SearchCompanionAvailable, a member of class ISearchAssistantOC3Disp.

8.2.13.3.7 ISearchAssistantOC3Disp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOC3Disp.

8.2.13.3.8 ISearchAssistantOC3Disp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOC3Disp.

8.2.13.3.9 ISearchAssistantOC3Disp.ShowFindPrinter Property

Pascal

```
property ShowFindPrinter: WordBool;
```

C++

```
__property WordBool ShowFindPrinter;
```

Description

This is ShowFindPrinter, a member of class ISearchAssistantOC3Disp.

8.2.13.3.10 ISearchAssistantOC3Disp.UseSearchCompanion Property

Pascal

```
property UseSearchCompanion: WordBool;
```

C++

```
__property WordBool UseSearchCompanion;
```

Description

This is UseSearchCompanion, a member of class ISearchAssistantOC3Disp.

8.2.13.3.11 ISearchAssistantOC3Disp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

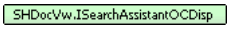
```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOC3Disp.

8.2.14 ISearchAssistantOCDisp Interface

Class Hierarchy



Pascal

```
ISearchAssistantOCDisp = dispinterface;
```

C++

```
class ISearchAssistantOCDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchAssistantOCDisp.

8.2.14.1 ISearchAssistantOCDisp Members

The following tables list the members exposed by ISearchAssistantOCDisp.

Public Methods

	Name	Description
	AddNextMenuItem (see page 679)	This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.
	EncodeString (see page 679)	This is EncodeString, a member of class ISearchAssistantOCDisp.
	FindComputer (see page 679)	This is FindComputer, a member of class ISearchAssistantOCDisp.
	FindFilesOrFolders (see page 679)	This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.
	FindOnWeb (see page 680)	This is FindOnWeb, a member of class ISearchAssistantOCDisp.
	FindPeople (see page 680)	This is FindPeople, a member of class ISearchAssistantOCDisp.
	FindPrinter (see page 680)	This is FindPrinter, a member of class ISearchAssistantOCDisp.
	GetProperty (see page 680)	This is GetProperty, a member of class ISearchAssistantOCDisp.
	GetSearchAssistantURL (see page 681)	This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.
	IsRestricted (see page 681)	This is IsRestricted, a member of class ISearchAssistantOCDisp.
	NavigateToDefaultSearch (see page 681)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.
	NETDetectNextNavigate (see page 681)	This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.
	NotifySearchSettingsChanged (see page 681)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.
	PutFindText (see page 682)	This is PutFindText, a member of class ISearchAssistantOCDisp.
	PutProperty (see page 682)	This is PutProperty, a member of class ISearchAssistantOCDisp.
	ResetNextMenu (see page 682)	This is ResetNextMenu, a member of class ISearchAssistantOCDisp.
	SetDefaultSearchUrl (see page 682)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

Public Properties

	Name	Description
	ASProvider (see page 683)	This is ASProvider, a member of class ISearchAssistantOCDisp.
	ASSetting (see page 683)	This is ASSetting, a member of class ISearchAssistantOCDisp.
	EventHandled (see page 683)	This is EventHandled, a member of class ISearchAssistantOCDisp.
	InWebFolder (see page 684)	This is InWebFolder, a member of class ISearchAssistantOCDisp.
	SearchAssistantDefault (see page 684)	This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.
	Searches (see page 684)	This is Searches, a member of class ISearchAssistantOCDisp.
	ShellFeaturesEnabled (see page 684)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.
	Version (see page 684)	This is Version, a member of class ISearchAssistantOCDisp.

8.2.14.2 ISearchAssistantOCDisp Methods

The methods of the ISearchAssistantOCDisp class are listed here.

Public Methods

	Name	Description
	AddNextMenuItem (see page 679)	This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.
	EncodeString (see page 679)	This is EncodeString, a member of class ISearchAssistantOCDisp.
	FindComputer (see page 679)	This is FindComputer, a member of class ISearchAssistantOCDisp.
	FindFilesOrFolders (see page 679)	This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.
	FindOnWeb (see page 680)	This is FindOnWeb, a member of class ISearchAssistantOCDisp.
	FindPeople (see page 680)	This is FindPeople, a member of class ISearchAssistantOCDisp.

	FindPrinter (see page 680)	This is FindPrinter, a member of class ISearchAssistantOCDisp.
	GetProperty (see page 680)	This is GetProperty, a member of class ISearchAssistantOCDisp.
	GetSearchAssistantURL (see page 681)	This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.
	IsRestricted (see page 681)	This is IsRestricted, a member of class ISearchAssistantOCDisp.
	NavigateToDefaultSearch (see page 681)	This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.
	NETDetectNextNavigate (see page 681)	This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.
	NotifySearchSettingsChanged (see page 681)	This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.
	PutFindText (see page 682)	This is PutFindText, a member of class ISearchAssistantOCDisp.
	PutProperty (see page 682)	This is PutProperty, a member of class ISearchAssistantOCDisp.
	ResetNextMenu (see page 682)	This is ResetNextMenu, a member of class ISearchAssistantOCDisp.
	SetDefaultSearchUrl (see page 682)	This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

8.2.14.2.1 ISearchAssistantOCDisp.AddNextMenuItem Method

Pascal

```
procedure AddNextMenuItem(const bstrText: WideString; idItem: Integer);
```

C++

```
__fastcall AddNextMenuItem(const BSTR bstrText, int idItem);
```

Description

This is AddNextMenuItem, a member of class ISearchAssistantOCDisp.

8.2.14.2.2 ISearchAssistantOCDisp.EncodeString Method

Pascal

```
function EncodeString(const bstrValue: WideString; const bstrCharSet: WideString; bUseUTF8:
WordBool): WideString;
```

C++

```
__fastcall BSTR EncodeString(const BSTR bstrValue, const BSTR bstrCharSet, WordBool bUseUTF8);
```

Description

This is EncodeString, a member of class ISearchAssistantOCDisp.

8.2.14.2.3 ISearchAssistantOCDisp.FindComputer Method

Pascal

```
procedure FindComputer;
```

C++

```
__fastcall FindComputer();
```

Description

This is FindComputer, a member of class ISearchAssistantOCDisp.

8.2.14.2.4 ISearchAssistantOCDisp.FindFilesOrFolders Method

Pascal

```
procedure FindFilesOrFolders;
```

C++

```
__fastcall FindFilesOrFolders();
```

Description

This is FindFilesOrFolders, a member of class ISearchAssistantOCDisp.

8.2.14.2.5 ISearchAssistantOCDisp.FindOnWeb Method

Pascal

```
procedure FindOnWeb;
```

C++

```
__fastcall FindOnWeb();
```

Description

This is FindOnWeb, a member of class ISearchAssistantOCDisp.

8.2.14.2.6 ISearchAssistantOCDisp.FindPeople Method

Pascal

```
procedure FindPeople;
```

C++

```
__fastcall FindPeople();
```

Description

This is FindPeople, a member of class ISearchAssistantOCDisp.

8.2.14.2.7 ISearchAssistantOCDisp.FindPrinter Method

Pascal

```
procedure FindPrinter;
```

C++

```
__fastcall FindPrinter();
```

Description

This is FindPrinter, a member of class ISearchAssistantOCDisp.

8.2.14.2.8 ISearchAssistantOCDisp.GetProperty Method

Pascal

```
function GetProperty(bPerLocale: WordBool; const bstrName: WideString): WideString;
```

C++

```
__fastcall BSTR GetProperty(WordBool bPerLocale, const BSTR bstrName);
```

Description

This is GetProperty, a member of class ISearchAssistantOCDisp.

8.2.14.2.9 ISearchAssistantOCDisp.GetSearchAssistantURL Method

Pascal

```
function GetSearchAssistantURL(bSubstitute: WordBool; bCustomize: WordBool): WideString;
```

C++

```
__fastcall BSTR GetSearchAssistantURL(WordBool bSubstitute, WordBool bCustomize);
```

Description

This is GetSearchAssistantURL, a member of class ISearchAssistantOCDisp.

8.2.14.2.10 ISearchAssistantOCDisp.IsRestricted Method

Pascal

```
function IsRestricted(const bstrGuid: WideString): WordBool;
```

C++

```
__fastcall WordBool IsRestricted(const BSTR bstrGuid);
```

Description

This is IsRestricted, a member of class ISearchAssistantOCDisp.

8.2.14.2.11 ISearchAssistantOCDisp.NavigateToDefaultSearch Method

Pascal

```
procedure NavigateToDefaultSearch;
```

C++

```
__fastcall NavigateToDefaultSearch();
```

Description

This is NavigateToDefaultSearch, a member of class ISearchAssistantOCDisp.

8.2.14.2.12 ISearchAssistantOCDisp.NETDetectNextNavigate Method

Pascal

```
procedure NETDetectNextNavigate;
```

C++

```
__fastcall NETDetectNextNavigate();
```

Description

This is NETDetectNextNavigate, a member of class ISearchAssistantOCDisp.

8.2.14.2.13 ISearchAssistantOCDisp.NotifySearchSettingsChanged Method

Pascal

```
procedure NotifySearchSettingsChanged;
```

C++

```
__fastcall NotifySearchSettingsChanged();
```

Description

This is NotifySearchSettingsChanged, a member of class ISearchAssistantOCDisp.

8.2.14.2.14 ISearchAssistantOCDisp.PutFindText Method

Pascal

```
procedure PutFindText(const FindText: WideString);
```

C++

```
__fastcall PutFindText(const BSTR FindText);
```

Description

This is PutFindText, a member of class ISearchAssistantOCDisp.

8.2.14.2.15 ISearchAssistantOCDisp.PutProperty Method

Pascal

```
procedure PutProperty(bPerLocale: WordBool; const bstrName: WideString; const bstrValue:
WideString);
```

C++

```
__fastcall PutProperty(WordBool bPerLocale, const BSTR bstrName, const BSTR bstrValue);
```

Description

This is PutProperty, a member of class ISearchAssistantOCDisp.

8.2.14.2.16 ISearchAssistantOCDisp.ResetNextMenu Method

Pascal

```
procedure ResetNextMenu;
```

C++

```
__fastcall ResetNextMenu();
```

Description

This is ResetNextMenu, a member of class ISearchAssistantOCDisp.

8.2.14.2.17 ISearchAssistantOCDisp.SetDefaultSearchUrl Method

Pascal

```
procedure SetDefaultSearchUrl(const bstrUrl: WideString);
```

C++

```
__fastcall SetDefaultSearchUrl(const BSTR bstrUrl);
```

Description

This is SetDefaultSearchUrl, a member of class ISearchAssistantOCDisp.

8.2.14.3 ISearchAssistantOCDisp Properties

The properties of the ISearchAssistantOCDisp class are listed here.

Public Properties

	Name	Description
	ASProvider (see page 683)	This is ASProvider, a member of class ISearchAssistantOCDisp.
	ASSetting (see page 683)	This is ASSetting, a member of class ISearchAssistantOCDisp.
	EventHandled (see page 683)	This is EventHandled, a member of class ISearchAssistantOCDisp.
	InWebFolder (see page 684)	This is InWebFolder, a member of class ISearchAssistantOCDisp.
	SearchAssistantDefault (see page 684)	This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.
	Searches (see page 684)	This is Searches, a member of class ISearchAssistantOCDisp.
	ShellFeaturesEnabled (see page 684)	This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.
	Version (see page 684)	This is Version, a member of class ISearchAssistantOCDisp.

8.2.14.3.1 ISearchAssistantOCDisp.ASProvider Property

Pascal

```
property ASProvider: WideString;
```

C++

```
__property BSTR ASProvider;
```

Description

This is ASProvider, a member of class ISearchAssistantOCDisp.

8.2.14.3.2 ISearchAssistantOCDisp.ASSetting Property

Pascal

```
property ASSetting: SYSINT;
```

C++

```
__property SYSINT ASSetting;
```

Description

This is ASSetting, a member of class ISearchAssistantOCDisp.

8.2.14.3.3 ISearchAssistantOCDisp.EventHandled Property

Pascal

```
property EventHandled: WordBool;
```

C++

```
__property WordBool EventHandled;
```

Description

This is EventHandled, a member of class ISearchAssistantOCDisp.

8.2.14.3.4 ISearchAssistantOCDisp.InWebFolder Property

Pascal

```
property InWebFolder: WordBool;
```

C++

```
__property WordBool InWebFolder;
```

Description

This is InWebFolder, a member of class ISearchAssistantOCDisp.

8.2.14.3.5 ISearchAssistantOCDisp.SearchAssistantDefault Property

Pascal

```
property SearchAssistantDefault: WordBool;
```

C++

```
__property WordBool SearchAssistantDefault;
```

Description

This is SearchAssistantDefault, a member of class ISearchAssistantOCDisp.

8.2.14.3.6 ISearchAssistantOCDisp.Searches Property

Pascal

```
property Searches: ISearches;
```

C++

```
__property ISearches Searches;
```

Description

This is Searches, a member of class ISearchAssistantOCDisp.

8.2.14.3.7 ISearchAssistantOCDisp.ShellFeaturesEnabled Property

Pascal

```
property ShellFeaturesEnabled: WordBool;
```

C++

```
__property WordBool ShellFeaturesEnabled;
```

Description

This is ShellFeaturesEnabled, a member of class ISearchAssistantOCDisp.

8.2.14.3.8 ISearchAssistantOCDisp.Version Property

Pascal

```
property Version: SYSINT;
```

C++

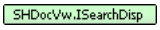
```
__property SYSINT Version;
```

Description

This is Version, a member of class ISearchAssistantOCDisp.

8.2.15 ISearchDisp Interface

Class Hierarchy



Pascal

```
ISearchDisp = dispinterface;
```

C++

```
class ISearchDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchDisp.

8.2.15.1 ISearchDisp Members

The following tables list the members exposed by ISearchDisp.

Public Properties

	Name	Description
	Id (see page 685)	This is Id, a member of class ISearchDisp.
	Title (see page 686)	This is Title, a member of class ISearchDisp.
	URL (see page 686)	This is URL, a member of class ISearchDisp.

8.2.15.2 ISearchDisp Properties

The properties of the ISearchDisp class are listed here.

Public Properties

	Name	Description
	Id (see page 685)	This is Id, a member of class ISearchDisp.
	Title (see page 686)	This is Title, a member of class ISearchDisp.
	URL (see page 686)	This is URL, a member of class ISearchDisp.

8.2.15.2.1 ISearchDisp.Id Property

Pascal

```
property Id: WideString;
```

C++

```
__property BSTR Id;
```

Description

This is Id, a member of class ISearchDisp.

8.2.15.2.2 ISearchDisp.Title Property

Pascal

```
property Title: WideString;
```

C++

```
__property BSTR Title;
```

Description

This is Title, a member of class ISearchDisp.

8.2.15.2.3 ISearchDisp.URL Property

Pascal

```
property URL: WideString;
```

C++

```
__property BSTR URL;
```

Description

This is URL, a member of class ISearchDisp.

8.2.16 ISearches Interface

Class Hierarchy

Pascal

```
ISearches = interface(IDispatch);
```

C++

```
__interface ISearches : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.ISearches.

8.2.16.1 ISearches Members

The following tables list the members exposed by ISearches.

Public Methods

	Name	Description
	_NewEnum (see page 687)	This is _NewEnum, a member of class ISearches.
	Get_Count (see page 687)	This is Get_Count, a member of class ISearches.
	Get_Default (see page 688)	This is Get_Default, a member of class ISearches.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	Item (see page 688)	This is Item, a member of class ISearches.

Public Properties

	Name	Description
R	Count (see page 688)	This is Count, a member of class ISearches.
R	Default (see page 688)	This is Default, a member of class ISearches.

8.2.16.2 ISearches Methods

The methods of the ISearches class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 687)	This is _NewEnum, a member of class ISearches.
	Get_Count (see page 687)	This is Get_Count, a member of class ISearches.
	Get_Default (see page 688)	This is Get_Default, a member of class ISearches.
	Item (see page 688)	This is Item, a member of class ISearches.

8.2.16.2.1 ISearches._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class ISearches.

8.2.16.2.2 ISearches.Get_Count Method

Pascal

```
function Get_Count: Integer; safecall;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class ISearches.

8.2.16.2.3 ISearches.Get_Default Method

Pascal

```
function Get_Default: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Default();
```

Description

This is Get_Default, a member of class ISearches.

8.2.16.2.4 ISearches.Item Method

Pascal

```
function Item(index: OleVariant): ISearch; safecall;
```

C++

```
__fastcall ISearch Item(OleVariant index);
```

Description

This is Item, a member of class ISearches.

8.2.16.3 ISearches Properties

The properties of the ISearches class are listed here.

Public Properties

	Name	Description
	Count (see page 688)	This is Count, a member of class ISearches.
	Default (see page 688)	This is Default, a member of class ISearches.

8.2.16.3.1 ISearches.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class ISearches.

8.2.16.3.2 ISearches.Default Property

Pascal

```
property Default: WideString;
```

C++

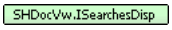
```
__property BSTR Default;
```

Description

This is Default, a member of class ISearches.

8.2.17 ISearchesDisp Interface

Class Hierarchy



Pascal

```
ISearchesDisp = dispinterface;
```

C++

```
class ISearchesDisp;
```

File

SHDocVw

Description

This is class SHDocVw.ISearchesDisp.

8.2.17.1 ISearchesDisp Members

The following tables list the members exposed by ISearchesDisp.

Public Methods

	Name	Description
	_NewEnum (see page 689)	This is _NewEnum, a member of class ISearchesDisp.
	Item (see page 690)	This is Item, a member of class ISearchesDisp.

Public Properties

	Name	Description
	Count (see page 690)	This is Count, a member of class ISearchesDisp.
	Default (see page 690)	This is Default, a member of class ISearchesDisp.

8.2.17.2 ISearchesDisp Methods

The methods of the ISearchesDisp class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 689)	This is _NewEnum, a member of class ISearchesDisp.
	Item (see page 690)	This is Item, a member of class ISearchesDisp.

8.2.17.2.1 ISearchesDisp._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class ISearchesDisp.

8.2.17.2.2 ISearchesDisp.Item Method

Pascal

```
function Item(index: OleVariant): ISearch;
```

C++

```
__fastcall ISearch Item(OleVariant index);
```

Description

This is Item, a member of class ISearchesDisp.

8.2.17.3 ISearchesDisp Properties

The properties of the ISearchesDisp class are listed here.

Public Properties

	Name	Description
	Count (see page 690)	This is Count, a member of class ISearchesDisp.
	Default (see page 690)	This is Default, a member of class ISearchesDisp.

8.2.17.3.1 ISearchesDisp.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class ISearchesDisp.

8.2.17.3.2 ISearchesDisp.Default Property

Pascal

```
property Default: WideString;
```

C++

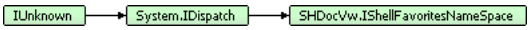
```
__property BSTR Default;
```

Description

This is Default, a member of class ISearchesDisp.

8.2.18 IShellFavoritesNameSpace Interface

Class Hierarchy



Pascal

```
IShellFavoritesNameSpace = interface(IDispatch);
```

C++

```
__interface IShellFavoritesNameSpace : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellFavoritesNameSpace.

8.2.18.1 IShellFavoritesNameSpace Members

The following tables list the members exposed by IShellFavoritesNameSpace.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 692)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 692)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Export (see page 692)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_SubscriptionsEnabled (see page 693)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Import (see page 693)	This is Import, a member of class IShellFavoritesNameSpace.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	InvokeContextMenuCommand (see page 693)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 693)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 694)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 694)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 694)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 694)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SetRoot (see page 694)	This is SetRoot, a member of class IShellFavoritesNameSpace.
	Synchronize (see page 695)	This is Synchronize, a member of class IShellFavoritesNameSpace.

Public Properties

	Name	Description
R	SubscriptionsEnabled (see page 695)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

8.2.18.2 IShellFavoritesNameSpace Methods

The methods of the IShellFavoritesNameSpace class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 692)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 692)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Export (see page 692)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_SubscriptionsEnabled (see page 693)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	Import (see page 693)	This is Import, a member of class IShellFavoritesNameSpace.
	InvokeContextMenuCommand (see page 693)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 693)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 694)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 694)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 694)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 694)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SetRoot (see page 694)	This is SetRoot, a member of class IShellFavoritesNameSpace.
	Synchronize (see page 695)	This is Synchronize, a member of class IShellFavoritesNameSpace.

8.2.18.2.1 IShellFavoritesNameSpace.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool; safecall;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.

8.2.18.2.2 IShellFavoritesNameSpace.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool; safecall;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.

8.2.18.2.3 IShellFavoritesNameSpace.Export Method

Pascal

```
procedure Export; safecall;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellFavoritesNameSpace.

8.2.18.2.4 IShellFavoritesNameSpace.Get_SubscriptionsEnabled Method

Pascal

```
function Get_SubscriptionsEnabled: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_SubscriptionsEnabled();
```

Description

This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

8.2.18.2.5 IShellFavoritesNameSpace.Import Method

Pascal

```
procedure Import; safecall;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellFavoritesNameSpace.

8.2.18.2.6 IShellFavoritesNameSpace.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString); safecall;
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.

8.2.18.2.7 IShellFavoritesNameSpace.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown; safecall;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.

8.2.18.2.8 IShellFavoritesNameSpace.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo; safecall;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.

8.2.18.2.9 IShellFavoritesNameSpace.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp; safecall;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.

8.2.18.2.10 IShellFavoritesNameSpace.NewFolder Method

Pascal

```
procedure NewFolder; safecall;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellFavoritesNameSpace.

8.2.18.2.11 IShellFavoritesNameSpace.ResetSort Method

Pascal

```
procedure ResetSort; safecall;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellFavoritesNameSpace.

8.2.18.2.12 IShellFavoritesNameSpace.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString); safecall;
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellFavoritesNameSpace.

8.2.18.2.13 IShellFavoritesNameSpace.Synchronize Method

Pascal

```
procedure Synchronize; safecall;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellFavoritesNameSpace.

8.2.18.3 IShellFavoritesNameSpace Properties

The properties of the IShellFavoritesNameSpace class are listed here.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 695)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

8.2.18.3.1 IShellFavoritesNameSpace.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

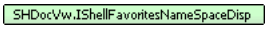
```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

8.2.19 IShellFavoritesNameSpaceDisp Interface

Class Hierarchy



Pascal

```
IShellFavoritesNameSpaceDisp = dispinterface;
```

C++

```
class IShellFavoritesNameSpaceDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellFavoritesNameSpaceDisp.

8.2.19.1 IShellFavoritesNameSpaceDisp Members

The following tables list the members exposed by IShellFavoritesNameSpaceDisp.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 697)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 697)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	Export (see page 697)	This is Export, a member of class IShellFavoritesNameSpaceDisp.
	Import (see page 697)	This is Import, a member of class IShellFavoritesNameSpaceDisp.
	InvokeContextMenuCommand (see page 698)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionDown (see page 698)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionTo (see page 698)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionUp (see page 698)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.
	NewFolder (see page 698)	This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.
	ResetSort (see page 699)	This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.
	SetRoot (see page 699)	This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.
	Synchronize (see page 699)	This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 700)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2 IShellFavoritesNameSpaceDisp Methods

The methods of the IShellFavoritesNameSpaceDisp class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 697)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 697)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.
	Export (see page 697)	This is Export, a member of class IShellFavoritesNameSpaceDisp.
	Import (see page 697)	This is Import, a member of class IShellFavoritesNameSpaceDisp.
	InvokeContextMenuCommand (see page 698)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionDown (see page 698)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionTo (see page 698)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.
	MoveSelectionUp (see page 698)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.
	NewFolder (see page 698)	This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.
	ResetSort (see page 699)	This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.
	SetRoot (see page 699)	This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.
	Synchronize (see page 699)	This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.1 IShellFavoritesNameSpaceDisp.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.2 IShellFavoritesNameSpaceDisp.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.3 IShellFavoritesNameSpaceDisp.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.4 IShellFavoritesNameSpaceDisp.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.5 IShellFavoritesNameSpaceDisp.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.6 IShellFavoritesNameSpaceDisp.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.7 IShellFavoritesNameSpaceDisp.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.8 IShellFavoritesNameSpaceDisp.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.9 IShellFavoritesNameSpaceDisp.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.10 IShellFavoritesNameSpaceDisp.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.11 IShellFavoritesNameSpaceDisp.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.2.12 IShellFavoritesNameSpaceDisp.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.3 IShellFavoritesNameSpaceDisp Properties

The properties of the IShellFavoritesNameSpaceDisp class are listed here.

Public Properties

	Name	Description
	SubscriptionsEnabled (see page 700)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

8.2.19.3.1 IShellFavoritesNameSpaceDisp.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

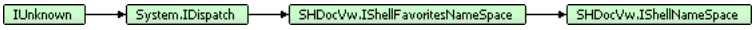
```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpaceDisp.

8.2.20 IShellNameSpace Interface

Class Hierarchy



Pascal

```
IShellNameSpace = interface(IShellFavoritesNameSpace);
```

C++

```
__interface IShellNameSpace : public IShellFavoritesNameSpace;
```

File

SHDocVw

Description

This is class SHDocVw.IShellNameSpace.

8.2.20.1 IShellNameSpace Members

The following tables list the members exposed by IShellNameSpace.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 692)	This is CreateSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	DeleteSubscriptionForSelection (see page 692)	This is DeleteSubscriptionForSelection, a member of class IShellFavoritesNameSpace.
	Expand (see page 702)	This is Expand, a member of class IShellNameSpace.
	Export (see page 692)	This is Export, a member of class IShellFavoritesNameSpace.
	Get_Columns (see page 702)	This is Get_Columns, a member of class IShellNameSpace.
	Get_CountViewTypes (see page 702)	This is Get_CountViewTypes, a member of class IShellNameSpace.
	Get_Depth (see page 702)	This is Get_Depth, a member of class IShellNameSpace.
	Get_EnumOptions (see page 703)	This is Get_EnumOptions, a member of class IShellNameSpace.
	Get_Flags (see page 703)	This is Get_Flags, a member of class IShellNameSpace.
	Get_Mode (see page 703)	This is Get_Mode, a member of class IShellNameSpace.
	Get_Root (see page 703)	This is Get_Root, a member of class IShellNameSpace.
	Get_SelectedItem (see page 704)	This is Get_SelectedItem, a member of class IShellNameSpace.
	Get_SubscriptionsEnabled (see page 693)	This is Get_SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.

	Get_TVFlags (see page 704)	This is Get_TVFlags, a member of class IShellNameSpace.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Import (see page 693)	This is Import, a member of class IShellFavoritesNameSpace.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	InvokeContextMenuCommand (see page 693)	This is InvokeContextMenuCommand, a member of class IShellFavoritesNameSpace.
	MoveSelectionDown (see page 693)	This is MoveSelectionDown, a member of class IShellFavoritesNameSpace.
	MoveSelectionTo (see page 694)	This is MoveSelectionTo, a member of class IShellFavoritesNameSpace.
	MoveSelectionUp (see page 694)	This is MoveSelectionUp, a member of class IShellFavoritesNameSpace.
	NewFolder (see page 694)	This is NewFolder, a member of class IShellFavoritesNameSpace.
	ResetSort (see page 694)	This is ResetSort, a member of class IShellFavoritesNameSpace.
	SelectedItems (see page 704)	This is SelectedItems, a member of class IShellNameSpace.
	Set_Columns (see page 704)	This is Set_Columns, a member of class IShellNameSpace.
	Set_Depth (see page 704)	This is Set_Depth, a member of class IShellNameSpace.
	Set_EnumOptions (see page 705)	This is Set_EnumOptions, a member of class IShellNameSpace.
	Set_Flags (see page 705)	This is Set_Flags, a member of class IShellNameSpace.
	Set_Mode (see page 705)	This is Set_Mode, a member of class IShellNameSpace.
	Set_Root (see page 705)	This is Set_Root, a member of class IShellNameSpace.
	Set_SelectedItem (see page 706)	This is Set_SelectedItem, a member of class IShellNameSpace.
	Set_TVFlags (see page 706)	This is Set_TVFlags, a member of class IShellNameSpace.
	SetRoot (see page 694)	This is SetRoot, a member of class IShellFavoritesNameSpace.
	SetViewType (see page 706)	This is SetViewType, a member of class IShellNameSpace.
	Synchronize (see page 695)	This is Synchronize, a member of class IShellFavoritesNameSpace.
	UnselectAll (see page 706)	This is UnselectAll, a member of class IShellNameSpace.

Public Properties

	Name	Description
	Columns (see page 707)	This is Columns, a member of class IShellNameSpace.
R	CountViewTypes (see page 707)	This is CountViewTypes, a member of class IShellNameSpace.
	Depth (see page 707)	This is Depth, a member of class IShellNameSpace.
	EnumOptions (see page 707)	This is EnumOptions, a member of class IShellNameSpace.
	Flags (see page 708)	This is Flags, a member of class IShellNameSpace.
	Mode (see page 708)	This is Mode, a member of class IShellNameSpace.
	Root (see page 708)	This is Root, a member of class IShellNameSpace.
	SelectedItem (see page 708)	This is SelectedItem, a member of class IShellNameSpace.
R	SubscriptionsEnabled (see page 695)	This is SubscriptionsEnabled, a member of class IShellFavoritesNameSpace.
	TVFlags (see page 709)	This is TVFlags, a member of class IShellNameSpace.

8.2.20.2 IShellNameSpace Methods

The methods of the IShellNameSpace class are listed here.

Public Methods

	Name	Description
	Expand (see page 702)	This is Expand, a member of class IShellNameSpace.
	Get_Columns (see page 702)	This is Get_Columns, a member of class IShellNameSpace.
	Get_CountViewTypes (see page 702)	This is Get_CountViewTypes, a member of class IShellNameSpace.
	Get_Depth (see page 702)	This is Get_Depth, a member of class IShellNameSpace.
	Get_EnumOptions (see page 703)	This is Get_EnumOptions, a member of class IShellNameSpace.
	Get_Flags (see page 703)	This is Get_Flags, a member of class IShellNameSpace.
	Get_Mode (see page 703)	This is Get_Mode, a member of class IShellNameSpace.

	Get_Root (see page 703)	This is Get_Root, a member of class IShellNameSpace.
	Get_SelectedItem (see page 704)	This is Get_SelectedItem, a member of class IShellNameSpace.
	Get_TVFlags (see page 704)	This is Get_TVFlags, a member of class IShellNameSpace.
	SelectedItems (see page 704)	This is SelectedItems, a member of class IShellNameSpace.
	Set_Columns (see page 704)	This is Set_Columns, a member of class IShellNameSpace.
	Set_Depth (see page 704)	This is Set_Depth, a member of class IShellNameSpace.
	Set_EnumOptions (see page 705)	This is Set_EnumOptions, a member of class IShellNameSpace.
	Set_Flags (see page 705)	This is Set_Flags, a member of class IShellNameSpace.
	Set_Mode (see page 705)	This is Set_Mode, a member of class IShellNameSpace.
	Set_Root (see page 705)	This is Set_Root, a member of class IShellNameSpace.
	Set_SelectedItem (see page 706)	This is Set_SelectedItem, a member of class IShellNameSpace.
	Set_TVFlags (see page 706)	This is Set_TVFlags, a member of class IShellNameSpace.
	SetViewType (see page 706)	This is SetViewType, a member of class IShellNameSpace.
	UnselectAll (see page 706)	This is UnselectAll, a member of class IShellNameSpace.

8.2.20.2.1 IShellNameSpace.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT); safecall;
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class IShellNameSpace.

8.2.20.2.2 IShellNameSpace.Get_Columns Method

Pascal

```
function Get_Columns: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Columns();
```

Description

This is Get_Columns, a member of class IShellNameSpace.

8.2.20.2.3 IShellNameSpace.Get_CountViewTypes Method

Pascal

```
function Get_CountViewTypes: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_CountViewTypes();
```

Description

This is Get_CountViewTypes, a member of class IShellNameSpace.

8.2.20.2.4 IShellNameSpace.Get_Depth Method

Pascal

```
function Get_Depth: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_Depth();
```

Description

This is Get_Depth, a member of class IShellNameSpace.

8.2.20.2.5 IShellNameSpace.Get_EnumOptions Method

Pascal

```
function Get_EnumOptions: Integer; safecall;
```

C++

```
__fastcall int Get_EnumOptions();
```

Description

This is Get_EnumOptions, a member of class IShellNameSpace.

8.2.20.2.6 IShellNameSpace.Get_Flags Method

Pascal

```
function Get_Flags: LongWord; safecall;
```

C++

```
__fastcall LongWord Get_Flags();
```

Description

This is Get_Flags, a member of class IShellNameSpace.

8.2.20.2.7 IShellNameSpace.Get_Mode Method

Pascal

```
function Get_Mode: SYSUINT; safecall;
```

C++

```
__fastcall SYSUINT Get_Mode();
```

Description

This is Get_Mode, a member of class IShellNameSpace.

8.2.20.2.8 IShellNameSpace.Get_Root Method

Pascal

```
function Get_Root: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_Root();
```

Description

This is Get_Root, a member of class IShellNameSpace.

8.2.20.2.9 IShellNameSpace.Get_SelectedItem Method

Pascal

```
function Get_SelectedItem: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_SelectedItem();
```

Description

This is Get_SelectedItem, a member of class IShellNameSpace.

8.2.20.2.10 IShellNameSpace.Get_TVFlags Method

Pascal

```
function Get_TVFlags: LongWord; safecall;
```

C++

```
__fastcall LongWord Get_TVFlags();
```

Description

This is Get_TVFlags, a member of class IShellNameSpace.

8.2.20.2.11 IShellNameSpace.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch; safecall;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class IShellNameSpace.

8.2.20.2.12 IShellNameSpace.Set_Columns Method

Pascal

```
procedure Set_Columns(const bstrColumns: WideString); safecall;
```

C++

```
__fastcall Set_Columns(const BSTR bstrColumns);
```

Description

This is Set_Columns, a member of class IShellNameSpace.

8.2.20.2.13 IShellNameSpace.Set_Depth Method

Pascal

```
procedure Set_Depth(piDepth: SYSINT); safecall;
```

C++

```
__fastcall Set_Depth(SYSINT piDepth);
```

Description

This is Set_Depth, a member of class IShellNameSpace.

8.2.20.2.14 IShellNameSpace.Set_EnumOptions Method

Pascal

```
procedure Set_EnumOptions(pgrfEnumFlags: Integer); safecall;
```

C++

```
__fastcall Set_EnumOptions(int pgrfEnumFlags);
```

Description

This is Set_EnumOptions, a member of class IShellNameSpace.

8.2.20.2.15 IShellNameSpace.Set_Flags Method

Pascal

```
procedure Set_Flags(pdwFlags: LongWord); safecall;
```

C++

```
__fastcall Set_Flags(LongWord pdwFlags);
```

Description

This is Set_Flags, a member of class IShellNameSpace.

8.2.20.2.16 IShellNameSpace.Set_Mode Method

Pascal

```
procedure Set_Mode(puMode: SYSUINT); safecall;
```

C++

```
__fastcall Set_Mode(SYSUINT puMode);
```

Description

This is Set_Mode, a member of class IShellNameSpace.

8.2.20.2.17 IShellNameSpace.Set_Root Method

Pascal

```
procedure Set_Root(pvar: OleVariant); safecall;
```

C++

```
__fastcall Set_Root(OleVariant pvar);
```

Description

This is Set_Root, a member of class IShellNameSpace.

8.2.20.2.18 IShellNameSpace.Set_SelectedItem Method

Pascal

```
procedure Set_SelectedItem(const pItem: IDispatch); safecall;
```

C++

```
__fastcall Set_SelectedItem(const IDispatch pItem);
```

Description

This is Set_SelectedItem, a member of class IShellNameSpace.

8.2.20.2.19 IShellNameSpace.Set_TVFlags Method

Pascal

```
procedure Set_TVFlags(dwFlags: LongWord); safecall;
```

C++

```
__fastcall Set_TVFlags(LongWord dwFlags);
```

Description

This is Set_TVFlags, a member of class IShellNameSpace.

8.2.20.2.20 IShellNameSpace.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT); safecall;
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class IShellNameSpace.

8.2.20.2.21 IShellNameSpace.UnselectAll Method

Pascal

```
procedure UnselectAll; safecall;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class IShellNameSpace.

8.2.20.3 IShellNameSpace Properties

The properties of the IShellNameSpace class are listed here.

Public Properties

	Name	Description
	Columns (see page 707)	This is Columns, a member of class IShellNameSpace.
R	CountViewTypes (see page 707)	This is CountViewTypes, a member of class IShellNameSpace.
	Depth (see page 707)	This is Depth, a member of class IShellNameSpace.
	EnumOptions (see page 707)	This is EnumOptions, a member of class IShellNameSpace.
	Flags (see page 708)	This is Flags, a member of class IShellNameSpace.
	Mode (see page 708)	This is Mode, a member of class IShellNameSpace.
	Root (see page 708)	This is Root, a member of class IShellNameSpace.
	SelectedItem (see page 708)	This is SelectedItem, a member of class IShellNameSpace.
	TVFlags (see page 709)	This is TVFlags, a member of class IShellNameSpace.

8.2.20.3.1 IShellNameSpace.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class IShellNameSpace.

8.2.20.3.2 IShellNameSpace.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class IShellNameSpace.

8.2.20.3.3 IShellNameSpace.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class IShellNameSpace.

8.2.20.3.4 IShellNameSpace.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class IShellNameSpace.

8.2.20.3.5 IShellNameSpace.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class IShellNameSpace.

8.2.20.3.6 IShellNameSpace.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class IShellNameSpace.

8.2.20.3.7 IShellNameSpace.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class IShellNameSpace.

8.2.20.3.8 IShellNameSpace.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class IShellNameSpace.

8.2.20.3.9 IShellNameSpace.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

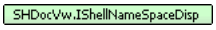
```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class IShellNameSpace.

8.2.21 IShellNameSpaceDisp Interface

Class Hierarchy



Pascal

```
IShellNameSpaceDisp = dispinterface;
```

C++

```
class IShellNameSpaceDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellNameSpaceDisp.

8.2.21.1 IShellNameSpaceDisp Members

The following tables list the members exposed by IShellNameSpaceDisp.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 710)	This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 711)	This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	Expand (see page 711)	This is Expand, a member of class IShellNameSpaceDisp.
	Export (see page 711)	This is Export, a member of class IShellNameSpaceDisp.
	Import (see page 711)	This is Import, a member of class IShellNameSpaceDisp.
	InvokeContextMenuCommand (see page 711)	This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.
	MoveSelectionDown (see page 712)	This is MoveSelectionDown, a member of class IShellNameSpaceDisp.
	MoveSelectionTo (see page 712)	This is MoveSelectionTo, a member of class IShellNameSpaceDisp.
	MoveSelectionUp (see page 712)	This is MoveSelectionUp, a member of class IShellNameSpaceDisp.
	NewFolder (see page 712)	This is NewFolder, a member of class IShellNameSpaceDisp.
	ResetSort (see page 713)	This is ResetSort, a member of class IShellNameSpaceDisp.
	SelectedItems (see page 713)	This is SelectedItems, a member of class IShellNameSpaceDisp.
	SetRoot (see page 713)	This is SetRoot, a member of class IShellNameSpaceDisp.
	SetViewType (see page 713)	This is SetViewType, a member of class IShellNameSpaceDisp.

	Synchronize (see page 713)	This is Synchronize, a member of class IShellNameSpaceDisp.
	UnselectAll (see page 714)	This is UnselectAll, a member of class IShellNameSpaceDisp.

Public Properties

	Name	Description
	Columns (see page 714)	This is Columns, a member of class IShellNameSpaceDisp.
	CountViewTypes (see page 714)	This is CountViewTypes, a member of class IShellNameSpaceDisp.
	Depth (see page 715)	This is Depth, a member of class IShellNameSpaceDisp.
	EnumOptions (see page 715)	This is EnumOptions, a member of class IShellNameSpaceDisp.
	Flags (see page 715)	This is Flags, a member of class IShellNameSpaceDisp.
	Mode (see page 715)	This is Mode, a member of class IShellNameSpaceDisp.
	Root (see page 716)	This is Root, a member of class IShellNameSpaceDisp.
	SelectedItem (see page 716)	This is SelectedItem, a member of class IShellNameSpaceDisp.
	SubscriptionsEnabled (see page 716)	This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.
	TVFlags (see page 716)	This is TVFlags, a member of class IShellNameSpaceDisp.

8.2.21.2 IShellNameSpaceDisp Methods

The methods of the IShellNameSpaceDisp class are listed here.

Public Methods

	Name	Description
	CreateSubscriptionForSelection (see page 710)	This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	DeleteSubscriptionForSelection (see page 711)	This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.
	Expand (see page 711)	This is Expand, a member of class IShellNameSpaceDisp.
	Export (see page 711)	This is Export, a member of class IShellNameSpaceDisp.
	Import (see page 711)	This is Import, a member of class IShellNameSpaceDisp.
	InvokeContextMenuCommand (see page 711)	This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.
	MoveSelectionDown (see page 712)	This is MoveSelectionDown, a member of class IShellNameSpaceDisp.
	MoveSelectionTo (see page 712)	This is MoveSelectionTo, a member of class IShellNameSpaceDisp.
	MoveSelectionUp (see page 712)	This is MoveSelectionUp, a member of class IShellNameSpaceDisp.
	NewFolder (see page 712)	This is NewFolder, a member of class IShellNameSpaceDisp.
	ResetSort (see page 713)	This is ResetSort, a member of class IShellNameSpaceDisp.
	SelectedItems (see page 713)	This is SelectedItems, a member of class IShellNameSpaceDisp.
	SetRoot (see page 713)	This is SetRoot, a member of class IShellNameSpaceDisp.
	SetViewType (see page 713)	This is SetViewType, a member of class IShellNameSpaceDisp.
	Synchronize (see page 713)	This is Synchronize, a member of class IShellNameSpaceDisp.
	UnselectAll (see page 714)	This is UnselectAll, a member of class IShellNameSpaceDisp.

8.2.21.2.1 IShellNameSpaceDisp.CreateSubscriptionForSelection Method

Pascal

```
function CreateSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool CreateSubscriptionForSelection();
```

Description

This is CreateSubscriptionForSelection, a member of class IShellNameSpaceDisp.

8.2.21.2.2 IShellNameSpaceDisp.DeleteSubscriptionForSelection Method

Pascal

```
function DeleteSubscriptionForSelection: WordBool;
```

C++

```
__fastcall WordBool DeleteSubscriptionForSelection();
```

Description

This is DeleteSubscriptionForSelection, a member of class IShellNameSpaceDisp.

8.2.21.2.3 IShellNameSpaceDisp.Expand Method

Pascal

```
procedure Expand(var_: OleVariant; iDepth: SYSINT);
```

C++

```
__fastcall Expand(OleVariant var_, SYSINT iDepth);
```

Description

This is Expand, a member of class IShellNameSpaceDisp.

8.2.21.2.4 IShellNameSpaceDisp.Export Method

Pascal

```
procedure Export;
```

C++

```
__fastcall Export();
```

Description

This is Export, a member of class IShellNameSpaceDisp.

8.2.21.2.5 IShellNameSpaceDisp.Import Method

Pascal

```
procedure Import;
```

C++

```
__fastcall Import();
```

Description

This is Import, a member of class IShellNameSpaceDisp.

8.2.21.2.6 IShellNameSpaceDisp.InvokeContextMenuCommand Method

Pascal

```
procedure InvokeContextMenuCommand(const strCommand: WideString);
```

C++

```
__fastcall InvokeContextMenuCommand(const BSTR strCommand);
```

Description

This is InvokeContextMenuCommand, a member of class IShellNameSpaceDisp.

8.2.21.2.7 IShellNameSpaceDisp.MoveSelectionDown Method

Pascal

```
procedure MoveSelectionDown;
```

C++

```
__fastcall MoveSelectionDown();
```

Description

This is MoveSelectionDown, a member of class IShellNameSpaceDisp.

8.2.21.2.8 IShellNameSpaceDisp.MoveSelectionTo Method

Pascal

```
procedure MoveSelectionTo;
```

C++

```
__fastcall MoveSelectionTo();
```

Description

This is MoveSelectionTo, a member of class IShellNameSpaceDisp.

8.2.21.2.9 IShellNameSpaceDisp.MoveSelectionUp Method

Pascal

```
procedure MoveSelectionUp;
```

C++

```
__fastcall MoveSelectionUp();
```

Description

This is MoveSelectionUp, a member of class IShellNameSpaceDisp.

8.2.21.2.10 IShellNameSpaceDisp.NewFolder Method

Pascal

```
procedure NewFolder;
```

C++

```
__fastcall NewFolder();
```

Description

This is NewFolder, a member of class IShellNameSpaceDisp.

8.2.21.2.11 IShellNameSpaceDisp.ResetSort Method

Pascal

```
procedure ResetSort;
```

C++

```
__fastcall ResetSort();
```

Description

This is ResetSort, a member of class IShellNameSpaceDisp.

8.2.21.2.12 IShellNameSpaceDisp.SelectedItems Method

Pascal

```
function SelectedItems: IDispatch;
```

C++

```
__fastcall IDispatch SelectedItems();
```

Description

This is SelectedItems, a member of class IShellNameSpaceDisp.

8.2.21.2.13 IShellNameSpaceDisp.SetRoot Method

Pascal

```
procedure SetRoot(const bstrFullPath: WideString);
```

C++

```
__fastcall SetRoot(const BSTR bstrFullPath);
```

Description

This is SetRoot, a member of class IShellNameSpaceDisp.

8.2.21.2.14 IShellNameSpaceDisp.SetViewType Method

Pascal

```
procedure SetViewType(iType: SYSINT);
```

C++

```
__fastcall SetViewType(SYSINT iType);
```

Description

This is SetViewType, a member of class IShellNameSpaceDisp.

8.2.21.2.15 IShellNameSpaceDisp.Synchronize Method

Pascal

```
procedure Synchronize;
```

C++

```
__fastcall Synchronize();
```

Description

This is Synchronize, a member of class IShellNameSpaceDisp.

8.2.21.2.16 IShellNameSpaceDisp.UnselectAll Method

Pascal

```
procedure UnselectAll;
```

C++

```
__fastcall UnselectAll();
```

Description

This is UnselectAll, a member of class IShellNameSpaceDisp.

8.2.21.3 IShellNameSpaceDisp Properties

The properties of the IShellNameSpaceDisp class are listed here.

Public Properties

	Name	Description
	Columns (see page 714)	This is Columns, a member of class IShellNameSpaceDisp.
	CountViewTypes (see page 714)	This is CountViewTypes, a member of class IShellNameSpaceDisp.
	Depth (see page 715)	This is Depth, a member of class IShellNameSpaceDisp.
	EnumOptions (see page 715)	This is EnumOptions, a member of class IShellNameSpaceDisp.
	Flags (see page 715)	This is Flags, a member of class IShellNameSpaceDisp.
	Mode (see page 715)	This is Mode, a member of class IShellNameSpaceDisp.
	Root (see page 716)	This is Root, a member of class IShellNameSpaceDisp.
	SelectedItem (see page 716)	This is SelectedItem, a member of class IShellNameSpaceDisp.
	SubscriptionsEnabled (see page 716)	This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.
	TVFlags (see page 716)	This is TVFlags, a member of class IShellNameSpaceDisp.

8.2.21.3.1 IShellNameSpaceDisp.Columns Property

Pascal

```
property Columns: WideString;
```

C++

```
__property BSTR Columns;
```

Description

This is Columns, a member of class IShellNameSpaceDisp.

8.2.21.3.2 IShellNameSpaceDisp.CountViewTypes Property

Pascal

```
property CountViewTypes: SYSINT;
```

C++

```
__property SYSINT CountViewTypes;
```

Description

This is CountViewTypes, a member of class IShellNameSpaceDisp.

8.2.21.3.3 IShellNameSpaceDisp.Depth Property

Pascal

```
property Depth: SYSINT;
```

C++

```
__property SYSINT Depth;
```

Description

This is Depth, a member of class IShellNameSpaceDisp.

8.2.21.3.4 IShellNameSpaceDisp.EnumOptions Property

Pascal

```
property EnumOptions: Integer;
```

C++

```
__property int EnumOptions;
```

Description

This is EnumOptions, a member of class IShellNameSpaceDisp.

8.2.21.3.5 IShellNameSpaceDisp.Flags Property

Pascal

```
property Flags: LongWord;
```

C++

```
__property LongWord Flags;
```

Description

This is Flags, a member of class IShellNameSpaceDisp.

8.2.21.3.6 IShellNameSpaceDisp.Mode Property

Pascal

```
property Mode: SYSUINT;
```

C++

```
__property SYSUINT Mode;
```

Description

This is Mode, a member of class IShellNameSpaceDisp.

8.2.21.3.7 IShellNameSpaceDisp.Root Property

Pascal

```
property Root: OleVariant;
```

C++

```
__property OleVariant Root;
```

Description

This is Root, a member of class IShellNameSpaceDisp.

8.2.21.3.8 IShellNameSpaceDisp.SelectedItem Property

Pascal

```
property SelectedItem: IDispatch;
```

C++

```
__property IDispatch SelectedItem;
```

Description

This is SelectedItem, a member of class IShellNameSpaceDisp.

8.2.21.3.9 IShellNameSpaceDisp.SubscriptionsEnabled Property

Pascal

```
property SubscriptionsEnabled: WordBool;
```

C++

```
__property WordBool SubscriptionsEnabled;
```

Description

This is SubscriptionsEnabled, a member of class IShellNameSpaceDisp.

8.2.21.3.10 IShellNameSpaceDisp.TVFlags Property

Pascal

```
property TVFlags: LongWord;
```

C++

```
__property LongWord TVFlags;
```

Description

This is TVFlags, a member of class IShellNameSpaceDisp.

8.2.22 IShellUIHelper Interface

Class Hierarchy

Pascal

```
IShellUIHelper = interface(IDispatch);
```

C++

```
__interface IShellUIHelper : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellUIHelper.

8.2.22.1 IShellUIHelper Members

The following tables list the members exposed by IShellUIHelper.

Public Methods

	Name	Description
	AddChannel (see page 718)	This is AddChannel, a member of class IShellUIHelper.
	AddDesktopComponent (see page 718)	This is AddDesktopComponent, a member of class IShellUIHelper.
	AddFavorite (see page 718)	This is AddFavorite, a member of class IShellUIHelper.
	AutoCompleteAttach (see page 718)	This is AutoCompleteAttach, a member of class IShellUIHelper.
	AutoCompleteSaveForm (see page 719)	This is AutoCompleteSaveForm, a member of class IShellUIHelper.
	AutoScan (see page 719)	This is AutoScan, a member of class IShellUIHelper.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	ImportExportFavorites (see page 719)	This is ImportExportFavorites, a member of class IShellUIHelper.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	IsSubscribed (see page 719)	This is IsSubscribed, a member of class IShellUIHelper.
	NavigateAndFind (see page 719)	This is NavigateAndFind, a member of class IShellUIHelper.
	RefreshOfflineDesktop (see page 720)	This is RefreshOfflineDesktop, a member of class IShellUIHelper.
	ResetFirstBootMode (see page 720)	This is ResetFirstBootMode, a member of class IShellUIHelper.
	ResetSafeMode (see page 720)	This is ResetSafeMode, a member of class IShellUIHelper.
	ShowBrowserUI (see page 720)	This is ShowBrowserUI, a member of class IShellUIHelper.

8.2.22.2 IShellUIHelper Methods

The methods of the IShellUIHelper class are listed here.

Public Methods

	Name	Description
	AddChannel (see page 718)	This is AddChannel, a member of class IShellUIHelper.
	AddDesktopComponent (see page 718)	This is AddDesktopComponent, a member of class IShellUIHelper.
	AddFavorite (see page 718)	This is AddFavorite, a member of class IShellUIHelper.
	AutoCompleteAttach (see page 718)	This is AutoCompleteAttach, a member of class IShellUIHelper.
	AutoCompleteSaveForm (see page 719)	This is AutoCompleteSaveForm, a member of class IShellUIHelper.
	AutoScan (see page 719)	This is AutoScan, a member of class IShellUIHelper.
	ImportExportFavorites (see page 719)	This is ImportExportFavorites, a member of class IShellUIHelper.
	IsSubscribed (see page 719)	This is IsSubscribed, a member of class IShellUIHelper.
	NavigateAndFind (see page 719)	This is NavigateAndFind, a member of class IShellUIHelper.

	RefreshOfflineDesktop (see page 720)	This is RefreshOfflineDesktop, a member of class IShellUIHelper.
	ResetFirstBootMode (see page 720)	This is ResetFirstBootMode, a member of class IShellUIHelper.
	ResetSafeMode (see page 720)	This is ResetSafeMode, a member of class IShellUIHelper.
	ShowBrowserUI (see page 720)	This is ShowBrowserUI, a member of class IShellUIHelper.

8.2.22.2.1 IShellUIHelper.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString); safecall;
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class IShellUIHelper.

8.2.22.2.2 IShellUIHelper.AddDesktopComponent Method

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant); safecall;
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class IShellUIHelper.

8.2.22.2.3 IShellUIHelper.AddFavorite Method

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant); safecall;
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class IShellUIHelper.

8.2.22.2.4 IShellUIHelper.AutoCompleteAttach Method

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant); safecall;
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class IShellUIHelper.

8.2.22.2.5 IShellUIHelper.AutoCompleteSaveForm Method

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant); safecall;
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class IShellUIHelper.

8.2.22.2.6 IShellUIHelper.AutoScan Method

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant); safecall;
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class IShellUIHelper.

8.2.22.2.7 IShellUIHelper.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString); safecall;
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class IShellUIHelper.

8.2.22.2.8 IShellUIHelper.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool; safecall;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class IShellUIHelper.

8.2.22.2.9 IShellUIHelper.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant); safecall;
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class IShellUIHelper.

8.2.22.2.10 IShellUIHelper.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop; safecall;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class IShellUIHelper.

8.2.22.2.11 IShellUIHelper.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode; safecall;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class IShellUIHelper.

8.2.22.2.12 IShellUIHelper.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode; safecall;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class IShellUIHelper.

8.2.22.2.13 IShellUIHelper.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
safecall;
```

C++

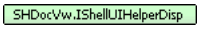
```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class IShellUIHelper.

8.2.23 IShellUIHelperDisp Interface

Class Hierarchy



Pascal

```
IShellUIHelperDisp = dispinterface;
```

C++

```
class IShellUIHelperDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellUIHelperDisp.

8.2.23.1 IShellUIHelperDisp Members

The following tables list the members exposed by IShellUIHelperDisp.

Public Methods

	Name	Description
	AddChannel (see page 722)	This is AddChannel, a member of class IShellUIHelperDisp.
	AddDesktopComponent (see page 722)	This is AddDesktopComponent, a member of class IShellUIHelperDisp.
	AddFavorite (see page 722)	This is AddFavorite, a member of class IShellUIHelperDisp.
	AutoCompleteAttach (see page 722)	This is AutoCompleteAttach, a member of class IShellUIHelperDisp.
	AutoCompleteSaveForm (see page 723)	This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.
	AutoScan (see page 723)	This is AutoScan, a member of class IShellUIHelperDisp.
	ImportExportFavorites (see page 723)	This is ImportExportFavorites, a member of class IShellUIHelperDisp.
	IsSubscribed (see page 723)	This is IsSubscribed, a member of class IShellUIHelperDisp.
	NavigateAndFind (see page 724)	This is NavigateAndFind, a member of class IShellUIHelperDisp.
	RefreshOfflineDesktop (see page 724)	This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.
	ResetFirstBootMode (see page 724)	This is ResetFirstBootMode, a member of class IShellUIHelperDisp.
	ResetSafeMode (see page 724)	This is ResetSafeMode, a member of class IShellUIHelperDisp.
	ShowBrowserUI (see page 724)	This is ShowBrowserUI, a member of class IShellUIHelperDisp.

8.2.23.2 IShellUIHelperDisp Methods

The methods of the IShellUIHelperDisp class are listed here.

Public Methods

	Name	Description
	AddChannel (see page 722)	This is AddChannel, a member of class IShellUIHelperDisp.
	AddDesktopComponent (see page 722)	This is AddDesktopComponent, a member of class IShellUIHelperDisp.
	AddFavorite (see page 722)	This is AddFavorite, a member of class IShellUIHelperDisp.
	AutoCompleteAttach (see page 722)	This is AutoCompleteAttach, a member of class IShellUIHelperDisp.
	AutoCompleteSaveForm (see page 723)	This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.

	AutoScan (see page 723)	This is AutoScan, a member of class IShellUIHelperDisp.
	ImportExportFavorites (see page 723)	This is ImportExportFavorites, a member of class IShellUIHelperDisp.
	IsSubscribed (see page 723)	This is IsSubscribed, a member of class IShellUIHelperDisp.
	NavigateAndFind (see page 724)	This is NavigateAndFind, a member of class IShellUIHelperDisp.
	RefreshOfflineDesktop (see page 724)	This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.
	ResetFirstBootMode (see page 724)	This is ResetFirstBootMode, a member of class IShellUIHelperDisp.
	ResetSafeMode (see page 724)	This is ResetSafeMode, a member of class IShellUIHelperDisp.
	ShowBrowserUI (see page 724)	This is ShowBrowserUI, a member of class IShellUIHelperDisp.

8.2.23.2.1 IShellUIHelperDisp.AddChannel Method

Pascal

```
procedure AddChannel(const URL: WideString);
```

C++

```
__fastcall AddChannel(const BSTR URL);
```

Description

This is AddChannel, a member of class IShellUIHelperDisp.

8.2.23.2.2 IShellUIHelperDisp.AddDesktopComponent Method

Pascal

```
procedure AddDesktopComponent(const URL: WideString; const Type_: WideString; var Left:
OleVariant; var Top: OleVariant; var Width: OleVariant; var Height: OleVariant);
```

C++

```
__fastcall AddDesktopComponent(const BSTR URL, const BSTR Type_, OleVariant Left, OleVariant
Top, OleVariant Width, OleVariant Height);
```

Description

This is AddDesktopComponent, a member of class IShellUIHelperDisp.

8.2.23.2.3 IShellUIHelperDisp.AddFavorite Method

Pascal

```
procedure AddFavorite(const URL: WideString; var Title: OleVariant);
```

C++

```
__fastcall AddFavorite(const BSTR URL, OleVariant Title);
```

Description

This is AddFavorite, a member of class IShellUIHelperDisp.

8.2.23.2.4 IShellUIHelperDisp.AutoCompleteAttach Method

Pascal

```
procedure AutoCompleteAttach(var Reserved: OleVariant);
```

C++

```
__fastcall AutoCompleteAttach(OleVariant Reserved);
```

Description

This is AutoCompleteAttach, a member of class IShellUIHelperDisp.

8.2.23.2.5 IShellUIHelperDisp.AutoCompleteSaveForm Method

Pascal

```
procedure AutoCompleteSaveForm(var Form: OleVariant);
```

C++

```
__fastcall AutoCompleteSaveForm(OleVariant Form);
```

Description

This is AutoCompleteSaveForm, a member of class IShellUIHelperDisp.

8.2.23.2.6 IShellUIHelperDisp.AutoScan Method

Pascal

```
procedure AutoScan(const strSearch: WideString; const strFailureUrl: WideString; var
pvarTargetFrame: OleVariant);
```

C++

```
__fastcall AutoScan(const BSTR strSearch, const BSTR strFailureUrl, OleVariant
pvarTargetFrame);
```

Description

This is AutoScan, a member of class IShellUIHelperDisp.

8.2.23.2.7 IShellUIHelperDisp.ImportExportFavorites Method

Pascal

```
procedure ImportExportFavorites(fImport: WordBool; const strImpExpPath: WideString);
```

C++

```
__fastcall ImportExportFavorites(WordBool fImport, const BSTR strImpExpPath);
```

Description

This is ImportExportFavorites, a member of class IShellUIHelperDisp.

8.2.23.2.8 IShellUIHelperDisp.IsSubscribed Method

Pascal

```
function IsSubscribed(const URL: WideString): WordBool;
```

C++

```
__fastcall WordBool IsSubscribed(const BSTR URL);
```

Description

This is IsSubscribed, a member of class IShellUIHelperDisp.

8.2.23.2.9 IShellUIHelperDisp.NavigateAndFind Method

Pascal

```
procedure NavigateAndFind(const URL: WideString; const strQuery: WideString; var
varTargetFrame: OleVariant);
```

C++

```
__fastcall NavigateAndFind(const BSTR URL, const BSTR strQuery, OleVariant varTargetFrame);
```

Description

This is NavigateAndFind, a member of class IShellUIHelperDisp.

8.2.23.2.10 IShellUIHelperDisp.RefreshOfflineDesktop Method

Pascal

```
procedure RefreshOfflineDesktop;
```

C++

```
__fastcall RefreshOfflineDesktop();
```

Description

This is RefreshOfflineDesktop, a member of class IShellUIHelperDisp.

8.2.23.2.11 IShellUIHelperDisp.ResetFirstBootMode Method

Pascal

```
procedure ResetFirstBootMode;
```

C++

```
__fastcall ResetFirstBootMode();
```

Description

This is ResetFirstBootMode, a member of class IShellUIHelperDisp.

8.2.23.2.12 IShellUIHelperDisp.ResetSafeMode Method

Pascal

```
procedure ResetSafeMode;
```

C++

```
__fastcall ResetSafeMode();
```

Description

This is ResetSafeMode, a member of class IShellUIHelperDisp.

8.2.23.2.13 IShellUIHelperDisp.ShowBrowserUI Method

Pascal

```
function ShowBrowserUI(const bstrName: WideString; var pvarIn: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ShowBrowserUI(const BSTR bstrName, OleVariant pvarIn);
```

Description

This is ShowBrowserUI, a member of class IShellUIHelperDisp.

8.2.24 IShellWindows Interface

Class Hierarchy

Pascal

```
IShellWindows = interface(IDispatch);
```

C++

```
__interface IShellWindows : public IDispatch;
```

File

SHDocVw

Description

This is class SHDocVw.IShellWindows.

8.2.24.1 IShellWindows Members

The following tables list the members exposed by IShellWindows.

Public Methods

	Name	Description
	_NewEnum (see page 726)	This is _NewEnum, a member of class IShellWindows.
	FindWindowSW (see page 726)	This is FindWindowSW, a member of class IShellWindows.
	Get_Count (see page 726)	This is Get_Count, a member of class IShellWindows.
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	Item (see page 727)	This is Item, a member of class IShellWindows.
	OnActivated (see page 727)	This is OnActivated, a member of class IShellWindows.
	OnCreated (see page 727)	This is OnCreated, a member of class IShellWindows.
	OnNavigate (see page 727)	This is OnNavigate, a member of class IShellWindows.
	ProcessAttachDetach (see page 727)	This is ProcessAttachDetach, a member of class IShellWindows.
	Register (see page 728)	This is Register, a member of class IShellWindows.
	RegisterPending (see page 728)	This is RegisterPending, a member of class IShellWindows.
	Revoke (see page 728)	This is Revoke, a member of class IShellWindows.

Public Properties

	Name	Description
R	Count (see page 729)	This is Count, a member of class IShellWindows.

8.2.24.2 IShellWindows Methods

The methods of the IShellWindows class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 726)	This is _NewEnum, a member of class IShellWindows.
	FindWindowSW (see page 726)	This is FindWindowSW, a member of class IShellWindows.
	Get_Count (see page 726)	This is Get_Count, a member of class IShellWindows.
	Item (see page 727)	This is Item, a member of class IShellWindows.
	OnActivated (see page 727)	This is OnActivated, a member of class IShellWindows.
	OnCreated (see page 727)	This is OnCreated, a member of class IShellWindows.
	OnNavigate (see page 727)	This is OnNavigate, a member of class IShellWindows.
	ProcessAttachDetach (see page 727)	This is ProcessAttachDetach, a member of class IShellWindows.
	Register (see page 728)	This is Register, a member of class IShellWindows.
	RegisterPending (see page 728)	This is RegisterPending, a member of class IShellWindows.
	Revoke (see page 728)	This is Revoke, a member of class IShellWindows.

8.2.24.2.1 IShellWindows._NewEnum Method

Pascal

```
function _NewEnum: IUnknown; safecall;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IShellWindows.

8.2.24.2.2 IShellWindows.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch; safecall;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class IShellWindows.

8.2.24.2.3 IShellWindows.Get_Count Method

Pascal

```
function Get_Count: Integer; safecall;
```

C++

```
__fastcall int Get_Count();
```

Description

This is Get_Count, a member of class IShellWindows.

8.2.24.2.4 IShellWindows.Item Method

Pascal

```
function Item(index: OleVariant): IDispatch; safecall;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class IShellWindows.

8.2.24.2.5 IShellWindows.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool); safecall;
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class IShellWindows.

8.2.24.2.6 IShellWindows.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown); safecall;
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class IShellWindows.

8.2.24.2.7 IShellWindows.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant); safecall;
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class IShellWindows.

8.2.24.2.8 IShellWindows.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool); safecall;
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class IShellWindows.

8.2.24.2.9 IShellWindows.Register Method

Pascal

```
procedure Register(const pid: IDispatch; hWnd: Integer; swClass: SYSINT; out plCookie:
Integer); safecall;
```

C++

```
__fastcall Register(const IDispatch pid, int hWnd, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class IShellWindows.

8.2.24.2.10 IShellWindows.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer); safecall;
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class IShellWindows.

8.2.24.2.11 IShellWindows.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer); safecall;
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class IShellWindows.

8.2.24.3 IShellWindows Properties

The properties of the IShellWindows class are listed here.

Public Properties

	Name	Description
	Count (see page 729)	This is Count, a member of class IShellWindows.

8.2.24.3.1 IShellWindows.Count Property

Pascal

```
property Count: Integer;
```

C++

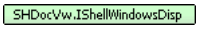
```
__property int Count;
```

Description

This is Count, a member of class IShellWindows.

8.2.25 IShellWindowsDisp Interface

Class Hierarchy



Pascal

```
IShellWindowsDisp = dispinterface;
```

C++

```
class IShellWindowsDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IShellWindowsDisp.

8.2.25.1 IShellWindowsDisp Members

The following tables list the members exposed by IShellWindowsDisp.

Public Methods

	Name	Description
	_NewEnum (see page 730)	This is _NewEnum, a member of class IShellWindowsDisp.
	FindWindowSW (see page 730)	This is FindWindowSW, a member of class IShellWindowsDisp.
	Item (see page 730)	This is Item, a member of class IShellWindowsDisp.
	OnActivated (see page 731)	This is OnActivated, a member of class IShellWindowsDisp.
	OnCreated (see page 731)	This is OnCreated, a member of class IShellWindowsDisp.
	OnNavigate (see page 731)	This is OnNavigate, a member of class IShellWindowsDisp.
	ProcessAttachDetach (see page 731)	This is ProcessAttachDetach, a member of class IShellWindowsDisp.
	Register (see page 731)	This is Register, a member of class IShellWindowsDisp.
	RegisterPending (see page 732)	This is RegisterPending, a member of class IShellWindowsDisp.
	Revoke (see page 732)	This is Revoke, a member of class IShellWindowsDisp.

Public Properties

	Name	Description
	Count (see page 732)	This is Count, a member of class IShellWindowsDisp.

8.2.25.2 IShellWindowsDisp Methods

The methods of the IShellWindowsDisp class are listed here.

Public Methods

	Name	Description
	_NewEnum (see page 730)	This is _NewEnum, a member of class IShellWindowsDisp.
	FindWindowSW (see page 730)	This is FindWindowSW, a member of class IShellWindowsDisp.
	Item (see page 730)	This is Item, a member of class IShellWindowsDisp.
	OnActivated (see page 731)	This is OnActivated, a member of class IShellWindowsDisp.
	OnCreated (see page 731)	This is OnCreated, a member of class IShellWindowsDisp.
	OnNavigate (see page 731)	This is OnNavigate, a member of class IShellWindowsDisp.
	ProcessAttachDetach (see page 731)	This is ProcessAttachDetach, a member of class IShellWindowsDisp.
	Register (see page 731)	This is Register, a member of class IShellWindowsDisp.
	RegisterPending (see page 732)	This is RegisterPending, a member of class IShellWindowsDisp.
	Revoke (see page 732)	This is Revoke, a member of class IShellWindowsDisp.

8.2.25.2.1 IShellWindowsDisp._NewEnum Method

Pascal

```
function _NewEnum: IUnknown;
```

C++

```
__fastcall IUnknown _NewEnum();
```

Description

This is _NewEnum, a member of class IShellWindowsDisp.

8.2.25.2.2 IShellWindowsDisp.FindWindowSW Method

Pascal

```
function FindWindowSW(var pvarloc: OleVariant; var pvarlocRoot: OleVariant; swClass: SYSINT;
out pHWND: Integer; swfwOptions: SYSINT): IDispatch;
```

C++

```
__fastcall IDispatch FindWindowSW(OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT swClass,
int pHWND, SYSINT swfwOptions);
```

Description

This is FindWindowSW, a member of class IShellWindowsDisp.

8.2.25.2.3 IShellWindowsDisp.Item Method

Pascal

```
function Item(index: OleVariant): IDispatch;
```

C++

```
__fastcall IDispatch Item(OleVariant index);
```

Description

This is Item, a member of class IShellWindowsDisp.

8.2.25.2.4 IShellWindowsDisp.OnActivated Method

Pascal

```
procedure OnActivated(lCookie: Integer; fActive: WordBool);
```

C++

```
__fastcall OnActivated(int lCookie, WordBool fActive);
```

Description

This is OnActivated, a member of class IShellWindowsDisp.

8.2.25.2.5 IShellWindowsDisp.OnCreated Method

Pascal

```
procedure OnCreated(lCookie: Integer; const punk: IUnknown);
```

C++

```
__fastcall OnCreated(int lCookie, const IUnknown punk);
```

Description

This is OnCreated, a member of class IShellWindowsDisp.

8.2.25.2.6 IShellWindowsDisp.OnNavigate Method

Pascal

```
procedure OnNavigate(lCookie: Integer; var pvarloc: OleVariant);
```

C++

```
__fastcall OnNavigate(int lCookie, OleVariant pvarloc);
```

Description

This is OnNavigate, a member of class IShellWindowsDisp.

8.2.25.2.7 IShellWindowsDisp.ProcessAttachDetach Method

Pascal

```
procedure ProcessAttachDetach(fAttach: WordBool);
```

C++

```
__fastcall ProcessAttachDetach(WordBool fAttach);
```

Description

This is ProcessAttachDetach, a member of class IShellWindowsDisp.

8.2.25.2.8 IShellWindowsDisp.Register Method

Pascal

```
procedure Register(const pid: IDispatch; hWnd: Integer; swClass: SYSINT; out plCookie:
Integer);
```

C++

```
__fastcall Register(const IDispatch pid, int hWnd, SYSINT swClass, int plCookie);
```

Description

This is Register, a member of class IShellWindowsDisp.

8.2.25.2.9 IShellWindowsDisp.RegisterPending Method

Pascal

```
procedure RegisterPending(lThreadId: Integer; var pvarloc: OleVariant; var pvarlocRoot:
OleVariant; swClass: SYSINT; out plCookie: Integer);
```

C++

```
__fastcall RegisterPending(int lThreadId, OleVariant pvarloc, OleVariant pvarlocRoot, SYSINT
swClass, int plCookie);
```

Description

This is RegisterPending, a member of class IShellWindowsDisp.

8.2.25.2.10 IShellWindowsDisp.Revoke Method

Pascal

```
procedure Revoke(lCookie: Integer);
```

C++

```
__fastcall Revoke(int lCookie);
```

Description

This is Revoke, a member of class IShellWindowsDisp.

8.2.25.3 IShellWindowsDisp Properties

The properties of the IShellWindowsDisp class are listed here.

Public Properties

	Name	Description
	Count (see page 732)	This is Count, a member of class IShellWindowsDisp.

8.2.25.3.1 IShellWindowsDisp.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IShellWindowsDisp.

8.2.26 IWebBrowser Interface

IWebBrowser provides access to the Web browser functionality of Microsoft's Shell Doc Object and Control Library (SHDOCVW.DLL).

Class Hierarchy

Pascal

```
IWebBrowser = interface(IDispatch);
```

C++

```
__interface IWebBrowser : public IDispatch;
```

File

SHDocVw

Description

IWebBrowser exposes methods to be implemented by TWebBrowser (see page 504) in order to allow you to create a customized Web browsing application or to add Internet, file and network browsing, document viewing, and data downloading capabilities to your applications.

Note: Because IWebBrowser resides in SHDOCVW.DLL, you must have SHDOCVW.DLL installed to use this component. This DLL ships with Internet Explorer 4 and later.

8.2.26.1 IWebBrowser Members

The following tables list the members exposed by IWebBrowser.

Public Methods

	Name	Description
	Get_Application (see page 735)	Returns the value of Application (see page 745).
	Get_Busy (see page 735)	Returns the value of Application (see page 745).
	Get_Container (see page 735)	Returns the value of Container (see page 746).
	Get_Document (see page 736)	Returns the value of Document (see page 746).
	Get_Height (see page 736)	Returns the value of Height (see page 746).
	Get_Left (see page 736)	Returns the value of Left (see page 747).
	Get_LocationName (see page 737)	Returns the value of LocationName (see page 747).
	Get_LocationURL (see page 737)	Returns the value of LocationURL (see page 747).
	Get_Parent (see page 737)	Returns the value of Parent (see page 748).
	Get_Top (see page 738)	Returns the value of Top (see page 748).
	Get_TopLevelContainer (see page 738)	Returns the value of TopLevelContainer (see page 748).
	Get_type_ (see page 738)	Returns the value of type_ (see page 749).
	Get_Width (see page 739)	Returns the value of Width (see page 749).
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	GoBack (see page 739)	Navigates to the previous item in the history list.
	GoForward (see page 739)	Navigates to the next item in the history list.

	GoHome (see page 740)	Navigates to the current start page.
	GoSearch (see page 740)	Navigates to the current search page.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	Navigate (see page 741)	Navigates to a resource identified by a path name or URL.
	Refresh (see page 742)	Reloads the current document.
	Refresh2 (see page 742)	Reloads the current document if necessary.
	Set_Height (see page 743)	Modifies the object height in pixels.
	Set_Left (see page 743)	Modifies the X axis coordinate of the left edge of the window.
	Set_Top (see page 743)	Modifies the Y axis coordinate of the upper edge of the window.
	Set_Width (see page 744)	Modifies the object width in pixels.
	Stop (see page 744)	Stops the current operation.

Public Properties

	Name	Description
R	Application (see page 745)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, for the interface for the WebBrowser (see page 837) control.
R	Busy (see page 745)	Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.
R	Container (see page 746)	Provides access to the automation interface of the Web browser's container.
R	Document (see page 746)	Provides access to the Automation interface of the active document.
	Height (see page 746)	Specifies the object height in pixels.
	Left (see page 747)	Specifies the X axis coordinate of the left edge of the window.
R	LocationName (see page 747)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 747)	Identifies the URL or path of the document or folder currently displayed.
R	Parent (see page 748)	Provides access to the Automation interface of the Web browser's container.
	Top (see page 748)	Specifies the Y axis coordinate of the upper edge of the window.
R	TopLevelContainer (see page 748)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 749)	Indicates the type name of the contained document object.
	Width (see page 749)	Specifies the object width in pixels.

8.2.26.2 IWebBrowser Methods

The methods of the IWebBrowser class are listed here.

Public Methods

	Name	Description
	Get_Application (see page 735)	Returns the value of Application (see page 745).
	Get_Busy (see page 735)	Returns the value of Application (see page 745).
	Get_Container (see page 735)	Returns the value of Container (see page 746).
	Get_Document (see page 736)	Returns the value of Document (see page 746).
	Get_Height (see page 736)	Returns the value of Height (see page 746).
	Get_Left (see page 736)	Returns the value of Left (see page 747).
	Get_LocationName (see page 737)	Returns the value of LocationName (see page 747).
	Get_LocationURL (see page 737)	Returns the value of LocationURL (see page 747).
	Get_Parent (see page 737)	Returns the value of Parent (see page 748).
	Get_Top (see page 738)	Returns the value of Top (see page 748).
	Get_TopLevelContainer (see page 738)	Returns the value of TopLevelContainer (see page 748).
	Get_type_ (see page 738)	Returns the value of type_ (see page 749).
	Get_Width (see page 739)	Returns the value of Width (see page 749).
	GoBack (see page 739)	Navigates to the previous item in the history list.
	GoForward (see page 739)	Navigates to the next item in the history list.
	GoHome (see page 740)	Navigates to the current start page.
	GoSearch (see page 740)	Navigates to the current search page.

	Navigate (see page 741)	Navigates to a resource identified by a path name or URL.
	Refresh (see page 742)	Reloads the current document.
	Refresh2 (see page 742)	Reloads the current document if necessary.
	Set_Height (see page 743)	Modifies the object height in pixels.
	Set_Left (see page 743)	Modifies the X axis coordinate of the left edge of the window.
	Set_Top (see page 743)	Modifies the Y axis coordinate of the upper edge of the window.
	Set_Width (see page 744)	Modifies the object width in pixels.
	Stop (see page 744)	Stops the current operation.

8.2.26.2.1 IWebBrowser.Get_Application Method

Returns the value of Application (see page 745).

Pascal

```
function Get_Application: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Application();
```

Description

Use Get_Application to access the interface for the application containing the WebBrowser (see page 837) control.

See Also

Application (see page 745)

8.2.26.2.2 IWebBrowser.Get_Busy Method

Returns the value of Application (see page 745).

Pascal

```
function Get_Busy: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Busy();
```

Description

Use Get_Busy to determine whether the Web browser component is currently navigating to a new URL or downloading a document.

See Also

Busy (see page 745)

8.2.26.2.3 IWebBrowser.Get_Container Method

Returns the value of Container (see page 746).

Pascal

```
function Get_Container: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Container();
```

Description

Use Get_Container to access the interface for the Web browser's parent or container. If the Web browser does not have a

container with an Automation interface, Get_Container returns nil.

See Also

Container (see page 746)

8.2.26.2.4 IWebBrowser.Get_Document Method

Returns the value of Document (see page 746).

Pascal

```
function Get_Document: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Document();
```

Description

Use Get_Document to access an IDispatch (see page 3494) interface for manipulating the contents of the HTML document through the COM object model.

See Also

Document (see page 746)

8.2.26.2.5 IWebBrowser.Get_Height Method

Returns the value of Height (see page 746).

Pascal

```
function Get_Height: Integer; safecall;
```

C++

```
__fastcall int Get_Height();
```

Description

Use Get_Height to determine the height of the object.

See Also

Set_Height (see page 743)

Height (see page 746)

8.2.26.2.6 IWebBrowser.Get_Left Method

Returns the value of Left (see page 747).

Pascal

```
function Get_Left: Integer; safecall;
```

C++

```
__fastcall int Get_Left();
```

Description

Use Get_Left to determine the left coordinate of the window.

See Also

Left (see page 747)

Set_Left (see page 743)

8.2.26.2.7 IWebBrowser.Get_LocationName Method

Returns the value of LocationName (see page 747).

Pascal

```
function Get_LocationName: WideString; safecall;
```

C++

```
__fastcall BSTR Get_LocationName();
```

Description

Use Get_LocationName to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, Get_LocationName returns the title of that page. If the resource is a folder or file on the local network or on a disk, Get_LocationName is the full UNC name of the folder or file.

See Also

LocationName (see page 747)

8.2.26.2.8 IWebBrowser.Get_LocationURL Method

Returns the value of LocationURL (see page 747).

Pascal

```
function Get_LocationURL: WideString; safecall;
```

C++

```
__fastcall BSTR Get_LocationURL();
```

Description

Use Get_LocationURL to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, Get_LocationName (see page 737) returns its URL. If the resource is a folder or file on the local network or on a disk, Get_LocationName (see page 737) is the full UNC name of the folder or file.

See Also

LocationURL (see page 747)

8.2.26.2.9 IWebBrowser.Get_Parent Method

Returns the value of Parent (see page 748).

Pascal

```
function Get_Parent: IDispatch; safecall;
```

C++

```
__fastcall IDispatch Get_Parent();
```

Description

Use Get_Parent to access the interface for the Web browser's parent or container.

See Also

Parent (see page 748)

8.2.26.2.10 IWebBrowser.Get_Top Method

Returns the value of Top (see page 748).

Pascal

```
function Get_Top: Integer; safecall;
```

C++

```
__fastcall int Get_Top();
```

Description

Use Get_Top to determine the upper coordinate of the window.

See Also

Top (see page 748)

Set_Top (see page 743)

8.2.26.2.11 IWebBrowser.Get_TopLevelContainer Method

Returns the value of TopLevelContainer (see page 748).

Pascal

```
function Get_TopLevelContainer: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_TopLevelContainer();
```

Description

Use Get_TopLevelContainer to determine whether the Web browser has a parent with an Automation interface.

See Also

TopLevelContainer (see page 748)

8.2.26.2.12 IWebBrowser.Get_type_ Method

Returns the value of type_ (see page 749).

Pascal

```
function Get_type_: WideString; safecall;
```

C++

```
__fastcall BSTR Get_type_();
```

Description

Use Get_type_ to determine the type of the Automation object whose interface is represented by the Document (see page 746) property.

See Also

type_ (see page 749)

8.2.26.2.13 IWebBrowser.Get_Width Method

Returns the value of Width (see page 749).

Pascal

```
function Get_Width: Integer; safecall;
```

C++

```
__fastcall int Get_Width();
```

Description

Use Get_Width to determine the width of the object.

See Also

Width (see page 749)

Set_Width (see page 744)

8.2.26.2.14 IWebBrowser.GoBack Method

Navigates to the previous item in the history list.

Pascal

```
procedure GoBack; safecall;
```

C++

```
__fastcall GoBack();
```

Description

Call GoBack to return to the last document in the history list. GoBack has no effect unless the history list contains additional documents.

Note: The Navigate (see page 741) or Navigate2 command can change the history list, depending on the specified flags.

See Also

GoForward (see page 739)

GoHome (see page 740)

GoSearch (see page 740)

8.2.26.2.15 IWebBrowser.GoForward Method

Navigates to the next item in the history list.

Pascal

```
procedure GoForward; safecall;
```

C++

```
__fastcall GoForward();
```

Description

Call GoForward to navigate to the next document in the history list. GoForward reverses the effect of the last GoBack (see page 739) command.

See Also

GoBack (see page 739)

GoHome (see page 740)

GoSearch (see page 740)

8.2.26.2.16 IWebBrowser.GoHome Method

Navigates to the current start page.

Pascal

```
procedure GoHome; safecall;
```

C++

```
__fastcall GoHome();
```

Description

Call GoHome to navigate to the registered home or start page. The URL for this page is specified in the Registry at `HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main` under the name Startpage. You can change this value using the TRegistry or TRegistryIniFile object, or using the Navigation tab of the Internet Explorer Options dialog.

See Also

GoBack (see page 739)

GoForward (see page 739)

GoSearch (see page 740)

TRegistry

TRegistryIniFile

8.2.26.2.17 IWebBrowser.GoSearch Method

Navigates to the current search page.

Pascal

```
procedure GoSearch; safecall;
```

C++

```
__fastcall GoSearch();
```

Description

Call GoSearch to navigate to the registered search page. The URL for this page is specified in the Registry at `HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main`, under the name SearchPage. You can change this value using the TRegistry or TRegistryIniFile object, or using the Navigation tab of the Internet Explorer Options dialog.

See Also

GoBack (see page 739)

GoForward (see page 739)

GoHome (see page 740)

TRegistry

TRegistryIniFile

8.2.26.2.18 IWebBrowser.Navigate Method

Navigates to a resource identified by a path name or URL.

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant); safecall;
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

Use Navigate to locate and download a specific resource. Navigate can send an HTTP message to a specified URL and display the results or simply display the results of a specified file.

URL specifies the UNC path name of a file or the Uniform Resource Locator of an Internet resource that the Web browser should display.

Flags is a set of values that specify whether to add the resource to the history list, to read from or write to the cache, or to display the resource in a new window. It can be a sum of zero or more of the following.

Constant	Value	Meaning
navOpenInNewWindow (see page 854)	1	Open the resource or file in a new window.
navNoHistory (see page 853)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 854)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 854)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 853)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate to generate an HTTP POST message. If PostData is NULL, Navigate generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

Note: To navigate to a location that cannot be expressed as an URL (for example, to an item identifier list), use the Navigate2 method instead.

See Also

Navigate2 (see page 755)

GoBack (see page 739)

GoForward (see page 739)

GoHome (see page 740)

GoSearch (see page 740)

8.2.26.2.19 IWebBrowser.Refresh Method

Reloads the current document.

Pascal

```
procedure Refresh; safecall;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to reload the current document. This is useful when the Web browser is displaying the resource at an URL that may have changed.

Refresh always sends a pragma:nocache header, telling the server not to return a cached copy. This can cause problems with some servers. To prevent the pragma:nocache header, or to refresh only expired pages, use the Refresh2 (see page 742) method instead.

See Also

Refresh2 (see page 742)

Document (see page 746)

8.2.26.2.20 IWebBrowser.Refresh2 Method

Reloads the current document if necessary.

Pascal

```
procedure Refresh2(var Level: OleVariant); safecall;
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

Call Refresh2 to reload the current document. Unlike the Refresh (see page 742) method, Refresh2 lets you specify what level of refresh to perform.

Level indicates what type of information is refreshed. The following table lists the possible values.

Constant	Value	Meaning
REFRESH_NORMAL (see page 878)	0	Perform a lightweight refresh that does not include the pragma:nocache header. The pragma:nocache header tells the server not to return a cached copy. This can cause problems with some servers.
REFRESH_IFEXPIRED (see page 878)	1	Only refresh if the page has expired. Do not include the pragma:nocache header.
REFRESH_CONTINUE (see page 877)	2	For internal use only.
REFRESH_COMPLETELY (see page 877)	3	Perform a full refresh, including the pragma:nocache header. Using this option is the same as calling the Refresh (see page 742) method.

When Level is omitted, a value of REFRESH_COMPLETELY (see page 877) is assumed.

See Also

Refresh (see page 742)

8.2.26.2.21 IWebBrowser.Set_Height Method

Modifies the object height in pixels.

Pascal

```
procedure Set_Height(pl: Integer); safecall;
```

C++

```
__fastcall Set_Height(int pl);
```

Description

Use Set_Height to modify the height of the object.

See Also

Get_Height (see page 736)

Height (see page 746)

8.2.26.2.22 IWebBrowser.Set_Left Method

Modifies the X axis coordinate of the left edge of the window.

Pascal

```
procedure Set_Left(pl: Integer); safecall;
```

C++

```
__fastcall Set_Left(int pl);
```

Description

Use Set_Left to modify the upper coordinate of the window and therefore the position of the window on the X axis.

See Also

Left (see page 747)

Get_Left (see page 736)

8.2.26.2.23 IWebBrowser.Set_Top Method

Modifies the Y axis coordinate of the upper edge of the window.

Pascal

```
procedure Set_Top(pl: Integer); safecall;
```

C++

```
__fastcall Set_Top(int pl);
```

Description

Use Top (see page 748) to modify the upper coordinate of the window and therefore the position of the window on the Y axis.

See Also

Top (see page 748)

Get_Top (see page 738)

8.2.26.2.24 IWebBrowser.Set_Width Method

Modifies the object width in pixels.

Pascal

```
procedure Set_Width(pl: Integer); safecall;
```

C++

```
__fastcall Set_Width(int pl);
```

Description

Use Width (see page 749) to modify the width of the object.

See Also

Width (see page 749)

Get_Width (see page 739)

8.2.26.2.25 IWebBrowser.Stop Method

Stops the current operation.

Pascal

```
procedure Stop; safecall;
```

C++

```
__fastcall Stop();
```

Description

Stop cancels any pending navigation or download operation and terminates any dynamic page elements, such as background sounds and animations.

See Also

GoForward (see page 739)

GoHome (see page 740)

GoSearch (see page 740)

GoBack (see page 739)

Navigate (see page 741)

8.2.26.3 IWebBrowser Properties

The properties of the IWebBrowser class are listed here.

Public Properties

	Name	Description
R	Application (see page 745)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, for the interface for the WebBrowser (see page 837) control.
R	Busy (see page 745)	Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.
R	Container (see page 746)	Provides access to the automation interface of the Web browser's container.
R	Document (see page 746)	Provides access to the Automation interface of the active document.
	Height (see page 746)	Specifies the object height in pixels.
	Left (see page 747)	Specifies the X axis coordinate of the left edge of the window.
R	LocationName (see page 747)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 747)	Identifies the URL or path of the document or folder currently displayed.
R	Parent (see page 748)	Provides access to the Automation interface of the Web browser's container.
	Top (see page 748)	Specifies the Y axis coordinate of the upper edge of the window.
R	TopLevelContainer (see page 748)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 749)	Indicates the type name of the contained document object.
	Width (see page 749)	Specifies the object width in pixels.

8.2.26.3.1 IWebBrowser.Application Property

Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, for the interface for the WebBrowser (see page 837) control.

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

Use Application to access the interface for the application containing the WebBrowser (see page 837) control. If the application does not have an Automation interface, or if the interface is not accessible, Application provides access to the Automation interface of the Web browser control.

See Also

Get_Application (see page 735)

8.2.26.3.2 IWebBrowser.Busy Property

Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

Read (see page 3634) Busy to determine whether the Web browser component is currently navigating to a new URL or downloading a document.

Note: The Web browser receives events at the beginning and end of each operation, as well as at discrete points during a download operation.

See Also

Get_Busy (see page 735)

8.2.26.3.3 IWebBrowser.Container Property

Provides access to the automation interface of the Web browser's container.

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

Use Container to access the interface for the Web browser's parent or container. If the Web browser does not have a container with an Automation interface, Container is nil.

See Also

Get_Container (see page 735)

8.2.26.3.4 IWebBrowser.Document Property

Provides access to the Automation interface of the active document.

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

Use Document to access an IDispatch (see page 3494) interface for manipulating the contents of the HTML document through the COM object model.

See Also

Get_Document (see page 736)

8.2.26.3.5 IWebBrowser.Height Property

Specifies the object height in pixels.

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

Read (see page 3634) Height to determine the height of the object. Write (see page 3707) Height to modify the height of the object.

See Also

Get_Height (see page 736)

Set_Height (see page 743)

8.2.26.3.6 IWebBrowser.Left Property

Specifies the X axis coordinate of the left edge of the window.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

Read (see page 3634) Left to determine the left coordinate of the window. Write (see page 3707) Left to modify the upper coordinate of the window and therefore the position of the window on the X axis.

See Also

Get_Left (see page 736)

Set_Left (see page 743)

8.2.26.3.7 IWebBrowser.LocationName Property

Identifies the short (UI-friendly) name of the URL or file currently displayed.

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

Read (see page 3634) LocationName to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName is the title of that page. If the resource is a folder or file on the local network or on a disk, LocationName is the full UNC name of the folder or file.

See Also

Get_LocationName (see page 737)

8.2.26.3.8 IWebBrowser.LocationURL Property

Identifies the URL or path of the document or folder currently displayed.

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

Read (see page 3634) LocationURL to get the name of the resource currently displayed in the Web browser control. If the resource is an HTML page from the Web, LocationName (see page 747) is its URL. If the resource is a folder or file on the local network or on a disk, LocationName (see page 747) is the full UNC name of the folder or file.

See Also

Get_LocationURL (see page 737)

8.2.26.3.9 IWebBrowser.Parent Property

Provides access to the Automation interface of the Web browser's container.

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

Use Parent to access the interface for the Web browser's parent or container.

See Also

Get_Parent (see page 737)

8.2.26.3.10 IWebBrowser.Top Property

Specifies the Y axis coordinate of the upper edge of the window.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

Read (see page 3634) Top to determine the upper coordinate of the window. Write (see page 3707) Top to modify the upper coordinate of the window and therefore the position of the window on the Y axis.

See Also

Get_Top (see page 738)

Set_Top (see page 743)

8.2.26.3.11 IWebBrowser.TopLevelContainer Property

Indicates whether the Web browser is a top-level Automation object.

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

Read (see page 3634) TopLevelContainer to determine whether the Web browser has a parent with an Automation interface.

See Also

Get_TopLevelContainer (see page 738)

8.2.26.3.12 IWebBrowser.type_ Property

Indicates the type name of the contained document object.

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

Read (see page 3634) type_ to determine the type of the Automation object whose interface is represented by the Document (see page 746) property.

See Also

Get_type_ (see page 738)

8.2.26.3.13 IWebBrowser.Width Property

Specifies the object width in pixels.

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

Read (see page 3634) Width to determine the width of the object. Write (see page 3707) Width to modify the width of the object.

See Also

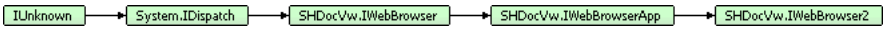
Get_Width (see page 739)

Set_Width (see page 744)

8.2.27 IWebBrowser2 Interface

IWebBrowser2 provides access to the Web browser functionality of Microsoft Shell Doc Object and Control Library (SHDOCVW.DLL).

Class Hierarchy



Pascal

```
IWebBrowser2 = interface(IWebBrowserApp);
```

C++

```
__interface IWebBrowser2 : public IWebBrowserApp;
```

File

SHDocVw

Description

IWebBrowser2 exposes methods to be implemented by TWebBrowser (see page 504) in order to allow you to create a customized Web browsing application or to add Internet, file and network browsing, document viewing, and data downloading capabilities to your applications.

Note: Because IWebBrowser2 resides in SHDOCVW.DLL, you must have SHDOCVW.DLL installed to use this component. This DLL ships with Internet Explorer 4 and later.

8.2.27.1 IWebBrowser2 Members

The following tables list the members exposed by IWebBrowser2.

Public Methods

	Name	Description
	ClientToWindow (see page 778)	This is ClientToWindow, a member of class IWebBrowserApp.
	ExecWB (see page 752)	Executes a command.
	Get_AddressBar (see page 753)	Returns the value of AddressBar (see page 760).
	Get_Application (see page 735)	Returns the value of Application (see page 745).
	Get_Busy (see page 735)	Returns the value of Application (see page 745).
	Get_Container (see page 735)	Returns the value of Container (see page 746).
	Get_Document (see page 736)	Returns the value of Document (see page 746).
	Get_FullName (see page 778)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 779)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_Height (see page 736)	Returns the value of Height (see page 746).
	Get_HWnd (see page 779)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_Left (see page 736)	Returns the value of Left (see page 747).
	Get_LocationName (see page 737)	Returns the value of LocationName (see page 747).
	Get_LocationURL (see page 737)	Returns the value of LocationURL (see page 747).
	Get_MenuBar (see page 779)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 779)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Offline (see page 753)	Returns the value of Offline (see page 760).
	Get_Parent (see page 737)	Returns the value of Parent (see page 748).
	Get_Path (see page 779)	This is Get_Path, a member of class IWebBrowserApp.
	Get_ReadyState (see page 754)	Returns the value of ReadyState (see page 760).
	Get_RegisterAsBrowser (see page 754)	Returns the value of RegisterAsBrowser (see page 761).
	Get_RegisterAsDropTarget (see page 754)	Returns the value of RegisterAsDropTarget (see page 761).
	Get_Resizable (see page 754)	Returns the value of Resizable (see page 762).
	Get_Silent (see page 755)	Returns the value of Silent (see page 762).
	Get_StatusBar (see page 780)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 780)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_TheaterMode (see page 755)	Returns the value of TheaterMode (see page 762).
	Get_ToolBar (see page 780)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Top (see page 738)	Returns the value of Top (see page 748).
	Get_TopLevelContainer (see page 738)	Returns the value of TopLevelContainer (see page 748).
	Get_type_ (see page 738)	Returns the value of type_ (see page 749).
	Get_Visible (see page 780)	This is Get_Visible, a member of class IWebBrowserApp.
	Get_Width (see page 739)	Returns the value of Width (see page 749).
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetProperty (see page 781)	This is GetProperty, a member of class IWebBrowserApp.

	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	GoBack (see page 739)	Navigates to the previous item in the history list.
	GoForward (see page 739)	Navigates to the next item in the history list.
	GoHome (see page 740)	Navigates to the current start page.
	GoSearch (see page 740)	Navigates to the current search page.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	Navigate (see page 741)	Navigates to a resource identified by a path name or URL.
	Navigate2 (see page 755)	Navigates to a specified resource.
	PutProperty (see page 781)	This is PutProperty, a member of class IWebBrowserApp.
	QueryStatusWB (see page 757)	Returns information about what commands the Web browser supports.
	Quit (see page 781)	This is Quit, a member of class IWebBrowserApp.
	Refresh (see page 742)	Reloads the current document.
	Refresh2 (see page 742)	Reloads the current document if necessary.
	Set_AddressBar (see page 757)	Sets the value of AddressBar (see page 760).
	Set_FullScreen (see page 781)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_Height (see page 743)	Modifies the object height in pixels.
	Set_Left (see page 743)	Modifies the X axis coordinate of the left edge of the window.
	Set_MenuBar (see page 781)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_Offline (see page 757)	Sets the value of Offline (see page 760).
	Set_RegisterAsBrowser (see page 758)	Sets the value of RegisterAsBrowser (see page 761).
	Set_RegisterAsDropTarget (see page 758)	Sets the value of RegisterAsDropTarget (see page 761).
	Set_Resizable (see page 758)	Sets the value of Resizable (see page 762).
	Set_Silent (see page 759)	Sets the value of Silent (see page 762).
	Set_StatusBar (see page 782)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 782)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_TheaterMode (see page 759)	Sets the value of TheaterMode (see page 762).
	Set_ToolBar (see page 782)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Top (see page 743)	Modifies the Y axis coordinate of the upper edge of the window.
	Set_Visible (see page 782)	This is Set_Visible, a member of class IWebBrowserApp.
	Set_Width (see page 744)	Modifies the object width in pixels.
	ShowBrowserBar (see page 759)	Used to hide or show a specific browser bar.
	Stop (see page 744)	Stops the current operation.

Public Properties

	Name	Description
	AddressBar (see page 760)	Specifies whether the address bar is visible.
R	Application (see page 745)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, for the interface for the WebBrowser (see page 837) control.
R	Busy (see page 745)	Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.
R	Container (see page 746)	Provides access to the automation interface of the Web browser's container.
R	Document (see page 746)	Provides access to the Automation interface of the active document.
R	FullName (see page 783)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 783)	This is FullScreen, a member of class IWebBrowserApp.
	Height (see page 746)	Specifies the object height in pixels.
R	HWnd (see page 783)	This is HWnd, a member of class IWebBrowserApp.
	Left (see page 747)	Specifies the X axis coordinate of the left edge of the window.
R	LocationName (see page 747)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 747)	Identifies the URL or path of the document or folder currently displayed.
	MenuBar (see page 783)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 784)	This is Name, a member of class IWebBrowserApp.
	Offline (see page 760)	Indicates whether the document is loaded from the local cache or not.
R	Parent (see page 748)	Provides access to the Automation interface of the Web browser's container.

R	Path (see page 784)	This is Path, a member of class IWebBrowserApp.
R	ReadyState (see page 760)	Indicates the current state.
	RegisterAsBrowser (see page 761)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 761)	Specifies whether users can load a document using the drag-and-drop operation.
	Resizable (see page 762)	Speficies whether the object can be resized.
	Silent (see page 762)	Specifies whether the dialog boxes can be displayed.
	StatusBar (see page 784)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 784)	This is StatusText, a member of class IWebBrowserApp.
	TheaterMode (see page 762)	Specifies if the object is in theater mode.
	ToolBar (see page 785)	This is ToolBar, a member of class IWebBrowserApp.
	Top (see page 748)	Specifies the Y axis coordinate of the upper edge of the window.
R	TopLevelContainer (see page 748)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 749)	Indicates the type name of the contained document object.
	Visible (see page 785)	This is Visible, a member of class IWebBrowserApp.
	Width (see page 749)	Specifies the object width in pixels.

8.2.27.2 IWebBrowser2 Methods

The methods of the IWebBrowser2 class are listed here.

Public Methods

	Name	Description
	ExecWB (see page 752)	Executes a command.
	Get_AddressBar (see page 753)	Returns the value of AddressBar (see page 760).
	Get_Offline (see page 753)	Returns the value of Offline (see page 760).
	Get_ReadyState (see page 754)	Returns the value of ReadyState (see page 760).
	Get_RegisterAsBrowser (see page 754)	Returns the value of RegisterAsBrowser (see page 761).
	Get_RegisterAsDropTarget (see page 754)	Returns the value of RegisterAsDropTarget (see page 761).
	Get_Resizable (see page 754)	Returns the value of Resizable (see page 762).
	Get_Silent (see page 755)	Returns the value of Silent (see page 762).
	Get_TheaterMode (see page 755)	Returns the value of TheaterMode (see page 762).
	Navigate2 (see page 755)	Navigates to a specified resource.
	QueryStatusWB (see page 757)	Returns information about what commands the Web browser supports.
	Set_AddressBar (see page 757)	Sets the value of AddressBar (see page 760).
	Set_Offline (see page 757)	Sets the value of Offline (see page 760).
	Set_RegisterAsBrowser (see page 758)	Sets the value of RegisterAsBrowser (see page 761).
	Set_RegisterAsDropTarget (see page 758)	Sets the value of RegisterAsDropTarget (see page 761).
	Set_Resizable (see page 758)	Sets the value of Resizable (see page 762).
	Set_Silent (see page 759)	Sets the value of Silent (see page 762).
	Set_TheaterMode (see page 759)	Sets the value of TheaterMode (see page 762).
	ShowBrowserBar (see page 759)	Used to hide or show a specific browser bar.

8.2.27.2.1 IWebBrowser2.ExecWB Method

Executes a command.

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant); safecall;
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

ExecWB uses the IOleCommandTarget interface to pass a command to the Web browser control. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command to execute. The Web Browser does not support all the commands for which Microsoft provides constants. Use the QueryStatusWB (see page 757) method to determine what commands can be used.

cmdexecopt modifies how the command is executed.

pvaIn specifies a Variant (see page 3421) array that contains any input arguments. This value can be omitted if there are no input arguments.

pvaOut specifies a Variant (see page 3421) that receives the results of the command, if any. This value can be omitted if there are no results.

See Also

QueryStatusWB (see page 757)

DoObjectVerb

GetObjectVerbs

8.2.27.2.2 IWebBrowser2.Get_AddressBar Method

Returns the value of AddressBar (see page 760).

Pascal

```
function Get_AddressBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_AddressBar();
```

Description

Get_AddressBar is used to return the value of AddressBar (see page 760).

Note: TWebBrowser (see page 504) does not have an address bar; this property is used for compatibility with Internet Explorer.

See Also

AddressBar (see page 760)

8.2.27.2.3 IWebBrowser2.Get_Offline Method

Returns the value of Offline (see page 760).

Pascal

```
function Get_Offline: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Offline();
```

Description

Get_Offline is used to return the value of Offline (see page 760).

See Also

Offline (see page 760)

8.2.27.2.4 IWebBrowser2.Get_ReadyState Method

Returns the value of ReadyState (see page 760).

Pascal

```
function Get_ReadyState: tagREADYSTATE; safecall;
```

C++

```
__fastcall tagREADYSTATE Get_ReadyState();
```

Description

Get_ReadyState is used to return the value of ReadyState (see page 760).

See Also

ReadyState (see page 760)

8.2.27.2.5 IWebBrowser2.Get_RegisterAsBrowser Method

Returns the value of RegisterAsBrowser (see page 761).

Pascal

```
function Get_RegisterAsBrowser: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_RegisterAsBrowser();
```

Description

Get_RegisterAsBrowser is used to return the value of RegisterAsBrowser (see page 761).

See Also

RegisterAsBrowser (see page 761)

8.2.27.2.6 IWebBrowser2.Get_RegisterAsDropTarget Method

Returns the value of RegisterAsDropTarget (see page 761).

Pascal

```
function Get_RegisterAsDropTarget: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_RegisterAsDropTarget();
```

Description

Get_RegisterAsDropTarget is used to return the value of RegisterAsDropTarget (see page 761).

See Also

RegisterAsDropTarget (see page 761)

8.2.27.2.7 IWebBrowser2.Get_Resizable Method

Returns the value of Resizable (see page 762).

Pascal

```
function Get_Resizable: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Resizable();
```

Description

Get_Resizable is used to return the value of Resizable (see page 762).

See Also

Resizable (see page 762)

8.2.27.2.8 IWebBrowser2.Get_Silent Method

Returns the value of Silent (see page 762).

Pascal

```
function Get_Silent: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Silent();
```

Description

Get_Silent is used to return the value of Silent (see page 762).

See Also

Silent (see page 762)

8.2.27.2.9 IWebBrowser2.Get_TheaterMode Method

Returns the value of TheaterMode (see page 762).

Pascal

```
function Get_TheaterMode: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_TheaterMode();
```

Description

Get_TheaterMode is used to return the value of TheaterMode (see page 762).

Note: TWebBrowser (see page 504) does not have theater mode; this property is used for compatibility with Internet Explorer.

See Also

TheaterMode (see page 762)

8.2.27.2.10 IWebBrowser2.Navigate2 Method

Navigates to a specified resource.

Pascal

```
procedure Navigate2(const URL: OleVariant; const Flags: OleVariant; const TargetFrameName:
OleVariant; const PostData: OleVariant; const Headers: OleVariant); safecall;
```

C++

```
__fastcall Navigate2(const OleVariant URL, const OleVariant Flags, const OleVariant
TargetFrameName, const OleVariant PostData, const OleVariant Headers);
```

Description

Use Navigate2 to locate and download a specific resource. Navigate (see page 741) can send an HTTP message to a specified URL and display the results, display the results of a specified file, or navigate to a resource that cannot be expressed as an URL, such as an item identifier list.

URL specifies the UNC path name of a file, the Uniform Resource Locator of an Internet resource, or a pointer to an item identifier list (PIDL).

Flags is a set of values that specify whether to add the resource to the history list, to read from or write to the cache, or to display the resource in a new window. It can be a sum of zero or more of the following.

Constant	Value	Meaning
navOpenInNewWindow (see page 854)	1	Open the resource or file in a new window.
navNoHistory (see page 853)	2	Do not add the resource or file to the history list. The new page replaces the current page in the list.
navNoReadFromCache (see page 854)	4	Do not read from the disk cache for this navigation.
navNoWriteToCache (see page 854)	8	Do not write the results of this navigation to the disk cache.
navAllowAutosearch (see page 853)	16	If the navigation fails, the Web browser attempts to navigate common root domains (.com, .edu, and so on). If this still fails, the URL is passed to a search engine.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame.

PostData contains the data sent to the server when using Navigate (see page 741) in order to generate an HTTP POST message. If PostData is NULL, Navigate (see page 741) generates an HTTP GET message. PostData is ignored if URL does not specify an HTTP URL.

Headers contains any headers sent to the servers when the URL represents an HTTP URL. HTTP headers specify the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers).

See Also

Navigate (see page 512)

GoBack (see page 510)

GoForward (see page 510)

GoHome (see page 511)

GoSearch (see page 511)

OnBeforeNavigate2 (see page 545)

OnNavigateComplete2 (see page 562)

TWebRequest

8.2.27.2.11 IWebBrowser2.QueryStatusWB Method

Returns information about what commands the Web browser supports.

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF; safecall;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

Use QueryStatusWB to determine what commands can be executed using the ExecWB (see page 752) method. QueryStatusWB uses the QueryStatus method of the IOleCommandTarget interface to obtain information about the status of commands. For more information, see the Microsoft documentation of IOleCommandTarget.

cmdID specifies the command about which you want information.

QueryStatusWB returns a value obtained by adding together flags that indicate the status of the command.

See Also

ExecWB (see page 752)

8.2.27.2.12 IWebBrowser2.Set_AddressBar Method

Sets the value of AddressBar (see page 760).

Pascal

```
procedure Set_AddressBar(Value: WordBool); safecall;
```

C++

```
__fastcall Set_AddressBar(WordBool Value);
```

Description

Set_AddressBar is used to set the value of AddressBar (see page 760).

Note: TWebBrowser (see page 504) does not have an address bar; this property is used for compatibility with Internet Explorer.

See Also

AddressBar (see page 760)

8.2.27.2.13 IWebBrowser2.Set_Offline Method

Sets the value of Offline (see page 760).

Pascal

```
procedure Set_Offline(pbOffline: WordBool); safecall;
```

C++

```
__fastcall Set_Offline(WordBool pbOffline);
```

Description

Set_Offline is used to set the value of Offline (see page 760).

See Also

Offline (see page 760)

8.2.27.2.14 IWebBrowser2.Set_RegisterAsBrowser Method

Sets the value of RegisterAsBrowser (see page 761).

Pascal

```
procedure Set_RegisterAsBrowser(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_RegisterAsBrowser(WordBool pbRegister);
```

Description

Set_RegisterAsBrowser is used to set the value of RegisterAsBrowser (see page 761).

See Also

RegisterAsBrowser (see page 761)

8.2.27.2.15 IWebBrowser2.Set_RegisterAsDropTarget Method

Sets the value of RegisterAsDropTarget (see page 761).

Pascal

```
procedure Set_RegisterAsDropTarget(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_RegisterAsDropTarget(WordBool pbRegister);
```

Description

Set_RegisterAsDropTargetis used to set the value of RegisterAsDropTarget (see page 761).

See Also

RegisterAsDropTarget (see page 761)

8.2.27.2.16 IWebBrowser2.Set_Resizable Method

Sets the value of Resizable (see page 762).

Pascal

```
procedure Set_Resizable(Value: WordBool); safecall;
```

C++

```
__fastcall Set_Resizable(WordBool Value);
```

Description

Set_Resizable is used to set the value of Resizable (see page 762).

See Also

Resizable (see page 762)

8.2.27.2.17 IWebBrowser2.Set_Silent Method

Sets the value of Silent (see page 762).

Pascal

```
procedure Set_Silent(pbSilent: WordBool); safecall;
```

C++

```
__fastcall Set_Silent(WordBool pbSilent);
```

Description

Set_Silent is used to set the value of Silent (see page 762).

See Also

Silent (see page 762)

8.2.27.2.18 IWebBrowser2.Set_TheaterMode Method

Sets the value of TheaterMode (see page 762).

Pascal

```
procedure Set_TheaterMode(pbRegister: WordBool); safecall;
```

C++

```
__fastcall Set_TheaterMode(WordBool pbRegister);
```

Description

Set_TheaterMode is used to set the value of TheaterMode (see page 762).

Note: TWebBrowser (see page 504) does not have theater mode; this property is used for compatibility with Internet Explorer.

See Also

TheaterMode (see page 762)

8.2.27.2.19 IWebBrowser2.ShowBrowserBar Method

Used to hide or show a specific browser bar.

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant); safecall;
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

Note: ShowBrowserBar is used for compatibility with Internet Explorer.

8.2.27.3 IWebBrowser2 Properties

The properties of the IWebBrowser2 class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 760)	Specifies whether the address bar is visible.
	Offline (see page 760)	Indicates whether the document is loaded from the local cache or not.
R	ReadyState (see page 760)	Indicates the current state.
	RegisterAsBrowser (see page 761)	Registers the control as a top-level browser for target name resolution.
	RegisterAsDropTarget (see page 761)	Specifies whether users can load a document using the drag-and-drop operation.
	Resizable (see page 762)	Speficies whether the object can be resized.
	Silent (see page 762)	Specifies whether the dialog boxes can be displayed.
	TheaterMode (see page 762)	Specifies if the object is in theater mode.

8.2.27.3.1 IWebBrowser2.AddressBar Property

Specifies whether the address bar is visible.

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

Has a boolean value that specifies the visibility of the address bar.

Note: TWebBrowser (see page 504) does not have an address bar; this property is used for compatibility with Internet Explorer.

8.2.27.3.2 IWebBrowser2.Offline Property

Indicates whether the document is loaded from the local cache or not.

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

Set Offline to True to work offline; the HTML documents are loaded from the local cache.

Note: This property is used to save HTML pages to a local cache so that they are available for offline work.

See Also

Navigate2 (see page 514)

Navigate2 (see page 755)

8.2.27.3.3 IWebBrowser2.ReadyState Property

Indicates the current state.

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

Read (see page 3634) ReadyState to obtain state information. Use this information to avoid attempting to use the Web browser control when it is not ready to perform an action.

The following table lists the possible values.

Constant	Value	Meaning
READYSTATE_UNINITIALIZED (see page 877)	0	Default initialization state. The Control has not begun loading any properties.
READYSTATE_LOADING (see page 876)	1	The control is loading its properties.
READYSTATE_LOADED (see page 876)	2	The control has initialized its properties, but is not yet interactive.
READYSTATE_INTERACTIVE (see page 876)	3	The control is interactive, but not all of its data is available. (It is downloading a document or frame).
READYSTATE_COMPLETE (see page 875)	4	The control is interactive and all of its data is available. (The current document is fully downloaded).

See Also

ComponentState

ControlState

Busy (see page 524)

8.2.27.3.4 IWebBrowser2.RegisterAsBrowser Property

Registers the control as a top-level browser for target name resolution.

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

Set RegisterAsBrowser to True when implementing IWebBrowser2 (see page 749) in order to create a top-level browser.

8.2.27.3.5 IWebBrowser2.RegisterAsDropTarget Property

Specifies whether users can load a document using the drag-and-drop operation.

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

Set RegisterAsDropTarget to True to allow users drop HTML documents onto the Web browser. When RegisterAsDropTarget is

True, users can load a document by dragging it onto the component, otherwise dragging and dropping has no effect.

See Also

Navigate2 (see page 755)

Navigate2 (see page 514)

8.2.27.3.6 IWebBrowser2.Resizable Property

Speficies whether the object can be resized.

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

Has a boolean value that specifies whether the object can be resized.

Note: The Resizable property is used for compatibility with Internet Explorer.

8.2.27.3.7 IWebBrowser2.Silent Property

Specifies whether the dialog boxes can be displayed.

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

Set Silent to True to prevent the Web browser control from displaying dialog boxes.

8.2.27.3.8 IWebBrowser2.TheaterMode Property

Specifies if the object is in theater mode.

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

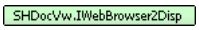
Description

TheaterMode is used to change the display of Internet Explorer to use theater or normal mode.

Note: TWebBrowser (see page 504) does not have theater mode; this property is used for compatibility with Internet Explorer.

8.2.28 IWebBrowser2Disp Interface

Class Hierarchy



Pascal

```
IWebBrowser2Disp = dispinterface;
```

C++

```
class IWebBrowser2Disp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowser2Disp.

8.2.28.1 IWebBrowser2Disp Members

The following tables list the members exposed by IWebBrowser2Disp.

Public Methods

	Name	Description
	ClientToWindow (see page 764)	This is ClientToWindow, a member of class IWebBrowser2Disp.
	ExecWB (see page 765)	This is ExecWB, a member of class IWebBrowser2Disp.
	GetProperty (see page 765)	This is GetProperty, a member of class IWebBrowser2Disp.
	GoBack (see page 765)	This is GoBack, a member of class IWebBrowser2Disp.
	GoForward (see page 765)	This is GoForward, a member of class IWebBrowser2Disp.
	GoHome (see page 766)	This is GoHome, a member of class IWebBrowser2Disp.
	GoSearch (see page 766)	This is GoSearch, a member of class IWebBrowser2Disp.
	Navigate (see page 766)	This is Navigate, a member of class IWebBrowser2Disp.
	Navigate2 (see page 766)	This is Navigate2, a member of class IWebBrowser2Disp.
	PutProperty (see page 767)	This is PutProperty, a member of class IWebBrowser2Disp.
	QueryStatusWB (see page 767)	This is QueryStatusWB, a member of class IWebBrowser2Disp.
	Quit (see page 767)	This is Quit, a member of class IWebBrowser2Disp.
	Refresh (see page 767)	This is Refresh, a member of class IWebBrowser2Disp.
	Refresh2 (see page 767)	This is Refresh2, a member of class IWebBrowser2Disp.
	ShowBrowserBar (see page 768)	This is ShowBrowserBar, a member of class IWebBrowser2Disp.
	Stop (see page 768)	This is Stop, a member of class IWebBrowser2Disp.

Public Properties

	Name	Description
	AddressBar (see page 769)	This is AddressBar, a member of class IWebBrowser2Disp.
	Application (see page 769)	This is Application, a member of class IWebBrowser2Disp.
	Busy (see page 769)	This is Busy, a member of class IWebBrowser2Disp.
	Container (see page 769)	This is Container, a member of class IWebBrowser2Disp.
	Document (see page 770)	This is Document, a member of class IWebBrowser2Disp.
	FullName (see page 770)	This is FullName, a member of class IWebBrowser2Disp.
	FullScreen (see page 770)	This is FullScreen, a member of class IWebBrowser2Disp.
	Height (see page 770)	This is Height, a member of class IWebBrowser2Disp.

	HWnd (see page 771)	This is HWnd, a member of class IWebBrowser2Disp.
	Left (see page 771)	This is Left, a member of class IWebBrowser2Disp.
	LocationName (see page 771)	This is LocationName, a member of class IWebBrowser2Disp.
	LocationURL (see page 771)	This is LocationURL, a member of class IWebBrowser2Disp.
	MenuBar (see page 771)	This is MenuBar, a member of class IWebBrowser2Disp.
	Name (see page 772)	This is Name, a member of class IWebBrowser2Disp.
	Offline (see page 772)	This is Offline, a member of class IWebBrowser2Disp.
	Parent (see page 772)	This is Parent, a member of class IWebBrowser2Disp.
	Path (see page 772)	This is Path, a member of class IWebBrowser2Disp.
	ReadyState (see page 773)	This is ReadyState, a member of class IWebBrowser2Disp.
	RegisterAsBrowser (see page 773)	This is RegisterAsBrowser, a member of class IWebBrowser2Disp.
	RegisterAsDropTarget (see page 773)	This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.
	Resizable (see page 773)	This is Resizable, a member of class IWebBrowser2Disp.
	Silent (see page 773)	This is Silent, a member of class IWebBrowser2Disp.
	StatusBar (see page 774)	This is StatusBar, a member of class IWebBrowser2Disp.
	StatusText (see page 774)	This is StatusText, a member of class IWebBrowser2Disp.
	TheaterMode (see page 774)	This is TheaterMode, a member of class IWebBrowser2Disp.
	ToolBar (see page 774)	This is ToolBar, a member of class IWebBrowser2Disp.
	Top (see page 775)	This is Top, a member of class IWebBrowser2Disp.
	TopLevelContainer (see page 775)	This is TopLevelContainer, a member of class IWebBrowser2Disp.
	type_ (see page 775)	This is type_, a member of class IWebBrowser2Disp.
	Visible (see page 775)	This is Visible, a member of class IWebBrowser2Disp.
	Width (see page 775)	This is Width, a member of class IWebBrowser2Disp.

8.2.28.2 IWebBrowser2Disp Methods

The methods of the IWebBrowser2Disp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 764)	This is ClientToWindow, a member of class IWebBrowser2Disp.
	ExecWB (see page 765)	This is ExecWB, a member of class IWebBrowser2Disp.
	GetProperty (see page 765)	This is GetProperty, a member of class IWebBrowser2Disp.
	GoBack (see page 765)	This is GoBack, a member of class IWebBrowser2Disp.
	GoForward (see page 765)	This is GoForward, a member of class IWebBrowser2Disp.
	GoHome (see page 766)	This is GoHome, a member of class IWebBrowser2Disp.
	GoSearch (see page 766)	This is GoSearch, a member of class IWebBrowser2Disp.
	Navigate (see page 766)	This is Navigate, a member of class IWebBrowser2Disp.
	Navigate2 (see page 766)	This is Navigate2, a member of class IWebBrowser2Disp.
	PutProperty (see page 767)	This is PutProperty, a member of class IWebBrowser2Disp.
	QueryStatusWB (see page 767)	This is QueryStatusWB, a member of class IWebBrowser2Disp.
	Quit (see page 767)	This is Quit, a member of class IWebBrowser2Disp.
	Refresh (see page 767)	This is Refresh, a member of class IWebBrowser2Disp.
	Refresh2 (see page 767)	This is Refresh2, a member of class IWebBrowser2Disp.
	ShowBrowserBar (see page 768)	This is ShowBrowserBar, a member of class IWebBrowser2Disp.
	Stop (see page 768)	This is Stop, a member of class IWebBrowser2Disp.

8.2.28.2.1 IWebBrowser2Disp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowser2Disp.

8.2.28.2.2 IWebBrowser2Disp.ExecWB Method

Pascal

```
procedure ExecWB(cmdID: OLECMDID; cmdexecopt: OLECMDEXECOPT; var pvaIn: OleVariant; var
pvaOut: OleVariant);
```

C++

```
__fastcall ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, OleVariant pvaIn, OleVariant
pvaOut);
```

Description

This is ExecWB, a member of class IWebBrowser2Disp.

8.2.28.2.3 IWebBrowser2Disp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowser2Disp.

8.2.28.2.4 IWebBrowser2Disp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowser2Disp.

8.2.28.2.5 IWebBrowser2Disp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowser2Disp.

8.2.28.2.6 IWebBrowser2Disp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowser2Disp.

8.2.28.2.7 IWebBrowser2Disp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowser2Disp.

8.2.28.2.8 IWebBrowser2Disp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowser2Disp.

8.2.28.2.9 IWebBrowser2Disp.Navigate2 Method

Pascal

```
procedure Navigate2(const URL: OleVariant; const Flags: OleVariant; const TargetFrameName:
OleVariant; const PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate2(const OleVariant URL, const OleVariant Flags, const OleVariant
TargetFrameName, const OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate2, a member of class IWebBrowser2Disp.

8.2.28.2.10 IWebBrowser2Disp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowser2Disp.

8.2.28.2.11 IWebBrowser2Disp.QueryStatusWB Method

Pascal

```
function QueryStatusWB(cmdID: OLECMDID): OLECMDF;
```

C++

```
__fastcall OLECMDF QueryStatusWB(OLECMDID cmdID);
```

Description

This is QueryStatusWB, a member of class IWebBrowser2Disp.

8.2.28.2.12 IWebBrowser2Disp.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowser2Disp.

8.2.28.2.13 IWebBrowser2Disp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowser2Disp.

8.2.28.2.14 IWebBrowser2Disp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowser2Disp.

8.2.28.2.15 IWebBrowser2Disp.ShowBrowserBar Method

Pascal

```
procedure ShowBrowserBar(var pvaClsid: OleVariant; var pvarShow: OleVariant; var pvarSize:
OleVariant);
```

C++

```
__fastcall ShowBrowserBar(OleVariant pvaClsid, OleVariant pvarShow, OleVariant pvarSize);
```

Description

This is ShowBrowserBar, a member of class IWebBrowser2Disp.

8.2.28.2.16 IWebBrowser2Disp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowser2Disp.

8.2.28.3 IWebBrowser2Disp Properties

The properties of the IWebBrowser2Disp class are listed here.

Public Properties

	Name	Description
	AddressBar (see page 769)	This is AddressBar, a member of class IWebBrowser2Disp.
	Application (see page 769)	This is Application, a member of class IWebBrowser2Disp.
	Busy (see page 769)	This is Busy, a member of class IWebBrowser2Disp.
	Container (see page 769)	This is Container, a member of class IWebBrowser2Disp.
	Document (see page 770)	This is Document, a member of class IWebBrowser2Disp.
	FullName (see page 770)	This is FullName, a member of class IWebBrowser2Disp.
	FullScreen (see page 770)	This is FullScreen, a member of class IWebBrowser2Disp.
	Height (see page 770)	This is Height, a member of class IWebBrowser2Disp.
	HWnd (see page 771)	This is HWnd, a member of class IWebBrowser2Disp.
	Left (see page 771)	This is Left, a member of class IWebBrowser2Disp.
	LocationName (see page 771)	This is LocationName, a member of class IWebBrowser2Disp.
	LocationURL (see page 771)	This is LocationURL, a member of class IWebBrowser2Disp.
	MenuBar (see page 771)	This is MenuBar, a member of class IWebBrowser2Disp.
	Name (see page 772)	This is Name, a member of class IWebBrowser2Disp.
	Offline (see page 772)	This is Offline, a member of class IWebBrowser2Disp.
	Parent (see page 772)	This is Parent, a member of class IWebBrowser2Disp.
	Path (see page 772)	This is Path, a member of class IWebBrowser2Disp.
	ReadyState (see page 773)	This is ReadyState, a member of class IWebBrowser2Disp.
	RegisterAsBrowser (see page 773)	This is RegisterAsBrowser, a member of class IWebBrowser2Disp.

	RegisterAsDropTarget (see page 773)	This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.
	Resizable (see page 773)	This is Resizable, a member of class IWebBrowser2Disp.
	Silent (see page 773)	This is Silent, a member of class IWebBrowser2Disp.
	StatusBar (see page 774)	This is StatusBar, a member of class IWebBrowser2Disp.
	StatusText (see page 774)	This is StatusText, a member of class IWebBrowser2Disp.
	TheaterMode (see page 774)	This is TheaterMode, a member of class IWebBrowser2Disp.
	ToolBar (see page 774)	This is ToolBar, a member of class IWebBrowser2Disp.
	Top (see page 775)	This is Top, a member of class IWebBrowser2Disp.
	TopLevelContainer (see page 775)	This is TopLevelContainer, a member of class IWebBrowser2Disp.
	type_ (see page 775)	This is type_, a member of class IWebBrowser2Disp.
	Visible (see page 775)	This is Visible, a member of class IWebBrowser2Disp.
	Width (see page 775)	This is Width, a member of class IWebBrowser2Disp.

8.2.28.3.1 IWebBrowser2Disp.AddressBar Property

Pascal

```
property AddressBar: WordBool;
```

C++

```
__property WordBool AddressBar;
```

Description

This is AddressBar, a member of class IWebBrowser2Disp.

8.2.28.3.2 IWebBrowser2Disp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowser2Disp.

8.2.28.3.3 IWebBrowser2Disp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowser2Disp.

8.2.28.3.4 IWebBrowser2Disp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowser2Disp.

8.2.28.3.5 IWebBrowser2Disp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowser2Disp.

8.2.28.3.6 IWebBrowser2Disp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowser2Disp.

8.2.28.3.7 IWebBrowser2Disp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowser2Disp.

8.2.28.3.8 IWebBrowser2Disp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowser2Disp.

8.2.28.3.9 IWebBrowser2Disp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowser2Disp.

8.2.28.3.10 IWebBrowser2Disp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowser2Disp.

8.2.28.3.11 IWebBrowser2Disp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowser2Disp.

8.2.28.3.12 IWebBrowser2Disp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowser2Disp.

8.2.28.3.13 IWebBrowser2Disp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowser2Disp.

8.2.28.3.14 IWebBrowser2Disp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowser2Disp.

8.2.28.3.15 IWebBrowser2Disp.Offline Property

Pascal

```
property Offline: WordBool;
```

C++

```
__property WordBool Offline;
```

Description

This is Offline, a member of class IWebBrowser2Disp.

8.2.28.3.16 IWebBrowser2Disp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowser2Disp.

8.2.28.3.17 IWebBrowser2Disp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowser2Disp.

8.2.28.3.18 IWebBrowser2Disp.ReadyState Property

Pascal

```
property ReadyState: tagREADYSTATE;
```

C++

```
__property tagREADYSTATE ReadyState;
```

Description

This is ReadyState, a member of class IWebBrowser2Disp.

8.2.28.3.19 IWebBrowser2Disp.RegisterAsBrowser Property

Pascal

```
property RegisterAsBrowser: WordBool;
```

C++

```
__property WordBool RegisterAsBrowser;
```

Description

This is RegisterAsBrowser, a member of class IWebBrowser2Disp.

8.2.28.3.20 IWebBrowser2Disp.RegisterAsDropTarget Property

Pascal

```
property RegisterAsDropTarget: WordBool;
```

C++

```
__property WordBool RegisterAsDropTarget;
```

Description

This is RegisterAsDropTarget, a member of class IWebBrowser2Disp.

8.2.28.3.21 IWebBrowser2Disp.Resizable Property

Pascal

```
property Resizable: WordBool;
```

C++

```
__property WordBool Resizable;
```

Description

This is Resizable, a member of class IWebBrowser2Disp.

8.2.28.3.22 IWebBrowser2Disp.Silent Property

Pascal

```
property Silent: WordBool;
```

C++

```
__property WordBool Silent;
```

Description

This is Silent, a member of class IWebBrowser2Disp.

8.2.28.3.23 IWebBrowser2Disp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowser2Disp.

8.2.28.3.24 IWebBrowser2Disp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowser2Disp.

8.2.28.3.25 IWebBrowser2Disp.TheaterMode Property

Pascal

```
property TheaterMode: WordBool;
```

C++

```
__property WordBool TheaterMode;
```

Description

This is TheaterMode, a member of class IWebBrowser2Disp.

8.2.28.3.26 IWebBrowser2Disp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowser2Disp.

8.2.28.3.27 IWebBrowser2Disp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowser2Disp.

8.2.28.3.28 IWebBrowser2Disp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowser2Disp.

8.2.28.3.29 IWebBrowser2Disp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowser2Disp.

8.2.28.3.30 IWebBrowser2Disp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowser2Disp.

8.2.28.3.31 IWebBrowser2Disp.Width Property

Pascal

```
property Width: Integer;
```

C++

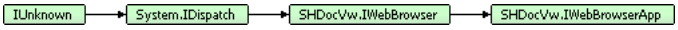
```
__property int Width;
```

Description

This is Width, a member of class IWebBrowser2Disp.

8.2.29 IWebBrowserApp Interface

Class Hierarchy



Pascal

```
IWebBrowserApp = interface(IWebBrowser);
```

C++

```
__interface IWebBrowserApp : public IWebBrowser;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserApp.

8.2.29.1 IWebBrowserApp Members

The following tables list the members exposed by IWebBrowserApp.

Public Methods

	Name	Description
	ClientToWindow (see page 778)	This is ClientToWindow, a member of class IWebBrowserApp.
	Get_Application (see page 735)	Returns the value of Application (see page 745).
	Get_Busy (see page 735)	Returns the value of Application (see page 745).
	Get_Container (see page 735)	Returns the value of Container (see page 746).
	Get_Document (see page 736)	Returns the value of Document (see page 746).
	Get_FullName (see page 778)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 779)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_Height (see page 736)	Returns the value of Height (see page 746).
	Get_HWnd (see page 779)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_Left (see page 736)	Returns the value of Left (see page 747).
	Get_LocationName (see page 737)	Returns the value of LocationName (see page 747).
	Get_LocationURL (see page 737)	Returns the value of LocationURL (see page 747).
	Get_MenuBar (see page 779)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 779)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Parent (see page 737)	Returns the value of Parent (see page 748).
	Get_Path (see page 779)	This is Get_Path, a member of class IWebBrowserApp.
	Get_StatusBar (see page 780)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 780)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_ToolBar (see page 780)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Top (see page 738)	Returns the value of Top (see page 748).
	Get_TopLevelContainer (see page 738)	Returns the value of TopLevelContainer (see page 748).
	Get_type_ (see page 738)	Returns the value of type_ (see page 749).
	Get_Visible (see page 780)	This is Get_Visible, a member of class IWebBrowserApp.

	Get_Width (see page 739)	Returns the value of Width (see page 749).
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetProperty (see page 781)	This is GetProperty, a member of class IWebBrowserApp.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	GoBack (see page 739)	Navigates to the previous item in the history list.
	GoForward (see page 739)	Navigates to the next item in the history list.
	GoHome (see page 740)	Navigates to the current start page.
	GoSearch (see page 740)	Navigates to the current search page.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.
	Navigate (see page 741)	Navigates to a resource identified by a path name or URL.
	PutProperty (see page 781)	This is PutProperty, a member of class IWebBrowserApp.
	Quit (see page 781)	This is Quit, a member of class IWebBrowserApp.
	Refresh (see page 742)	Reloads the current document.
	Refresh2 (see page 742)	Reloads the current document if necessary.
	Set_FullScreen (see page 781)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_Height (see page 743)	Modifies the object height in pixels.
	Set_Left (see page 743)	Modifies the X axis coordinate of the left edge of the window.
	Set_MenuBar (see page 781)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_StatusBar (see page 782)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 782)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_ToolBar (see page 782)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Top (see page 743)	Modifies the Y axis coordinate of the upper edge of the window.
	Set_Visible (see page 782)	This is Set_Visible, a member of class IWebBrowserApp.
	Set_Width (see page 744)	Modifies the object width in pixels.
	Stop (see page 744)	Stops the current operation.

Public Properties

	Name	Description
R	Application (see page 745)	Provides access to the Automation interface for the application that contains the WebBrowser (see page 837) control, or, if that is unavailable, for the interface for the WebBrowser (see page 837) control.
R	Busy (see page 745)	Indicates whether the WebBrowser (see page 837) control is in the middle of a navigation or downloading operation.
R	Container (see page 746)	Provides access to the automation interface of the Web browser's container.
R	Document (see page 746)	Provides access to the Automation interface of the active document.
R	FullName (see page 783)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 783)	This is FullScreen, a member of class IWebBrowserApp.
	Height (see page 746)	Specifies the object height in pixels.
R	HWnd (see page 783)	This is HWnd, a member of class IWebBrowserApp.
	Left (see page 747)	Specifies the X axis coordinate of the left edge of the window.
R	LocationName (see page 747)	Identifies the short (UI-friendly) name of the URL or file currently displayed.
R	LocationURL (see page 747)	Identifies the URL or path of the document or folder currently displayed.
	MenuBar (see page 783)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 784)	This is Name, a member of class IWebBrowserApp.
R	Parent (see page 748)	Provides access to the Automation interface of the Web browser's container.
R	Path (see page 784)	This is Path, a member of class IWebBrowserApp.
	StatusBar (see page 784)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 784)	This is StatusText, a member of class IWebBrowserApp.
	ToolBar (see page 785)	This is ToolBar, a member of class IWebBrowserApp.
	Top (see page 748)	Specifies the Y axis coordinate of the upper edge of the window.
R	TopLevelContainer (see page 748)	Indicates whether the Web browser is a top-level Automation object.
R	type_ (see page 749)	Indicates the type name of the contained document object.
	Visible (see page 785)	This is Visible, a member of class IWebBrowserApp.
	Width (see page 749)	Specifies the object width in pixels.

8.2.29.2 IWebBrowserApp Methods

The methods of the IWebBrowserApp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 778)	This is ClientToWindow, a member of class IWebBrowserApp.
	Get_FullName (see page 778)	This is Get_FullName, a member of class IWebBrowserApp.
	Get_FullScreen (see page 779)	This is Get_FullScreen, a member of class IWebBrowserApp.
	Get_HWnd (see page 779)	This is Get_HWnd, a member of class IWebBrowserApp.
	Get_MenuBar (see page 779)	This is Get_MenuBar, a member of class IWebBrowserApp.
	Get_Name (see page 779)	This is Get_Name, a member of class IWebBrowserApp.
	Get_Path (see page 779)	This is Get_Path, a member of class IWebBrowserApp.
	Get_StatusBar (see page 780)	This is Get_StatusBar, a member of class IWebBrowserApp.
	Get_StatusText (see page 780)	This is Get_StatusText, a member of class IWebBrowserApp.
	Get_ToolBar (see page 780)	This is Get_ToolBar, a member of class IWebBrowserApp.
	Get_Visible (see page 780)	This is Get_Visible, a member of class IWebBrowserApp.
	GetProperty (see page 781)	This is GetProperty, a member of class IWebBrowserApp.
	PutProperty (see page 781)	This is PutProperty, a member of class IWebBrowserApp.
	Quit (see page 781)	This is Quit, a member of class IWebBrowserApp.
	Set_FullScreen (see page 781)	This is Set_FullScreen, a member of class IWebBrowserApp.
	Set_MenuBar (see page 781)	This is Set_MenuBar, a member of class IWebBrowserApp.
	Set_StatusBar (see page 782)	This is Set_StatusBar, a member of class IWebBrowserApp.
	Set_StatusText (see page 782)	This is Set_StatusText, a member of class IWebBrowserApp.
	Set_ToolBar (see page 782)	This is Set_ToolBar, a member of class IWebBrowserApp.
	Set_Visible (see page 782)	This is Set_Visible, a member of class IWebBrowserApp.

8.2.29.2.1 IWebBrowserApp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT); safecall;
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowserApp.

8.2.29.2.2 IWebBrowserApp.Get_FullName Method

Pascal

```
function Get_FullName: WideString; safecall;
```

C++

```
__fastcall BSTR Get_FullName();
```

Description

This is Get_FullName, a member of class IWebBrowserApp.

8.2.29.2.3 IWebBrowserApp.Get_FullScreen Method

Pascal

```
function Get_FullScreen: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_FullScreen();
```

Description

This is Get_FullScreen, a member of class IWebBrowserApp.

8.2.29.2.4 IWebBrowserApp.Get_HWnd Method

Pascal

```
function Get_HWnd: Integer; safecall;
```

C++

```
__fastcall int Get_HWnd();
```

Description

This is Get_HWnd, a member of class IWebBrowserApp.

8.2.29.2.5 IWebBrowserApp.Get_MenuBar Method

Pascal

```
function Get_MenuBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_MenuBar();
```

Description

This is Get_MenuBar, a member of class IWebBrowserApp.

8.2.29.2.6 IWebBrowserApp.Get_Name Method

Pascal

```
function Get_Name: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Name();
```

Description

This is Get_Name, a member of class IWebBrowserApp.

8.2.29.2.7 IWebBrowserApp.Get_Path Method

Pascal

```
function Get_Path: WideString; safecall;
```

C++

```
__fastcall BSTR Get_Path();
```

Description

This is Get_Path, a member of class IWebBrowserApp.

8.2.29.2.8 IWebBrowserApp.Get_StatusBar Method

Pascal

```
function Get_StatusBar: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_StatusBar();
```

Description

This is Get_StatusBar, a member of class IWebBrowserApp.

8.2.29.2.9 IWebBrowserApp.Get_StatusText Method

Pascal

```
function Get_StatusText: WideString; safecall;
```

C++

```
__fastcall BSTR Get_StatusText();
```

Description

This is Get_StatusText, a member of class IWebBrowserApp.

8.2.29.2.10 IWebBrowserApp.Get_ToolBar Method

Pascal

```
function Get_ToolBar: SYSINT; safecall;
```

C++

```
__fastcall SYSINT Get_ToolBar();
```

Description

This is Get_ToolBar, a member of class IWebBrowserApp.

8.2.29.2.11 IWebBrowserApp.Get_Visible Method

Pascal

```
function Get_Visible: WordBool; safecall;
```

C++

```
__fastcall WordBool Get_Visible();
```

Description

This is Get_Visible, a member of class IWebBrowserApp.

8.2.29.2.12 IWebBrowserApp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant; safecall;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowserApp.

8.2.29.2.13 IWebBrowserApp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant); safecall;
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowserApp.

8.2.29.2.14 IWebBrowserApp.Quit Method

Pascal

```
procedure Quit; safecall;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowserApp.

8.2.29.2.15 IWebBrowserApp.Set_FullScreen Method

Pascal

```
procedure Set_FullScreen(pbFullScreen: WordBool); safecall;
```

C++

```
__fastcall Set_FullScreen(WordBool pbFullScreen);
```

Description

This is Set_FullScreen, a member of class IWebBrowserApp.

8.2.29.2.16 IWebBrowserApp.Set_MenuBar Method

Pascal

```
procedure Set_MenuBar(Value: WordBool); safecall;
```

C++

```
__fastcall Set_MenuBar(WordBool Value);
```

Description

This is Set_MenuBar, a member of class IWebBrowserApp.

8.2.29.2.17 IWebBrowserApp.Set_StatusBar Method

Pascal

```
procedure Set_StatusBar(pBool: WordBool); safecall;
```

C++

```
__fastcall Set_StatusBar(WordBool pBool);
```

Description

This is Set_StatusBar, a member of class IWebBrowserApp.

8.2.29.2.18 IWebBrowserApp.Set_StatusText Method

Pascal

```
procedure Set_StatusText(const StatusText: WideString); safecall;
```

C++

```
__fastcall Set_StatusText(const BSTR StatusText);
```

Description

This is Set_StatusText, a member of class IWebBrowserApp.

8.2.29.2.19 IWebBrowserApp.Set_ToolBar Method

Pascal

```
procedure Set_ToolBar(Value: SYSINT); safecall;
```

C++

```
__fastcall Set_ToolBar(SYSINT Value);
```

Description

This is Set_ToolBar, a member of class IWebBrowserApp.

8.2.29.2.20 IWebBrowserApp.Set_Visible Method

Pascal

```
procedure Set_Visible(pBool: WordBool); safecall;
```

C++

```
__fastcall Set_Visible(WordBool pBool);
```

Description

This is Set_Visible, a member of class IWebBrowserApp.

8.2.29.3 IWebBrowserApp Properties

The properties of the IWebBrowserApp class are listed here.

Public Properties

	Name	Description
R	FullName (see page 783)	This is FullName, a member of class IWebBrowserApp.
	FullScreen (see page 783)	This is FullScreen, a member of class IWebBrowserApp.
R	HWnd (see page 783)	This is HWnd, a member of class IWebBrowserApp.
	MenuBar (see page 783)	This is MenuBar, a member of class IWebBrowserApp.
R	Name (see page 784)	This is Name, a member of class IWebBrowserApp.
R	Path (see page 784)	This is Path, a member of class IWebBrowserApp.
	StatusBar (see page 784)	This is StatusBar, a member of class IWebBrowserApp.
	StatusText (see page 784)	This is StatusText, a member of class IWebBrowserApp.
	ToolBar (see page 785)	This is ToolBar, a member of class IWebBrowserApp.
	Visible (see page 785)	This is Visible, a member of class IWebBrowserApp.

8.2.29.3.1 IWebBrowserApp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowserApp.

8.2.29.3.2 IWebBrowserApp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowserApp.

8.2.29.3.3 IWebBrowserApp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowserApp.

8.2.29.3.4 IWebBrowserApp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowserApp.

8.2.29.3.5 IWebBrowserApp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowserApp.

8.2.29.3.6 IWebBrowserApp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowserApp.

8.2.29.3.7 IWebBrowserApp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowserApp.

8.2.29.3.8 IWebBrowserApp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowserApp.

8.2.29.3.9 IWebBrowserApp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowserApp.

8.2.29.3.10 IWebBrowserApp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

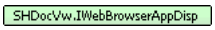
```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowserApp.

8.2.30 IWebBrowserAppDisp Interface

Class Hierarchy



Pascal

```
IWebBrowserAppDisp = dispinterface;
```

C++

```
class IWebBrowserAppDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserAppDisp.

8.2.30.1 IWebBrowserAppDisp Members

The following tables list the members exposed by IWebBrowserAppDisp.

Public Methods

	Name	Description
	ClientToWindow (see page 787)	This is ClientToWindow, a member of class IWebBrowserAppDisp.
	GetProperty (see page 787)	This is GetProperty, a member of class IWebBrowserAppDisp.
	GoBack (see page 787)	This is GoBack, a member of class IWebBrowserAppDisp.

	GoForward (see page 787)	This is GoForward, a member of class IWebBrowserAppDisp.
	GoHome (see page 787)	This is GoHome, a member of class IWebBrowserAppDisp.
	GoSearch (see page 788)	This is GoSearch, a member of class IWebBrowserAppDisp.
	Navigate (see page 788)	This is Navigate, a member of class IWebBrowserAppDisp.
	PutProperty (see page 788)	This is PutProperty, a member of class IWebBrowserAppDisp.
	Quit (see page 788)	This is Quit, a member of class IWebBrowserAppDisp.
	Refresh (see page 789)	This is Refresh, a member of class IWebBrowserAppDisp.
	Refresh2 (see page 789)	This is Refresh2, a member of class IWebBrowserAppDisp.
	Stop (see page 789)	This is Stop, a member of class IWebBrowserAppDisp.

Public Properties

	Name	Description
	Application (see page 790)	This is Application, a member of class IWebBrowserAppDisp.
	Busy (see page 790)	This is Busy, a member of class IWebBrowserAppDisp.
	Container (see page 790)	This is Container, a member of class IWebBrowserAppDisp.
	Document (see page 790)	This is Document, a member of class IWebBrowserAppDisp.
	FullName (see page 791)	This is FullName, a member of class IWebBrowserAppDisp.
	FullScreen (see page 791)	This is FullScreen, a member of class IWebBrowserAppDisp.
	Height (see page 791)	This is Height, a member of class IWebBrowserAppDisp.
	HWnd (see page 791)	This is HWnd, a member of class IWebBrowserAppDisp.
	Left (see page 792)	This is Left, a member of class IWebBrowserAppDisp.
	LocationName (see page 792)	This is LocationName, a member of class IWebBrowserAppDisp.
	LocationURL (see page 792)	This is LocationURL, a member of class IWebBrowserAppDisp.
	MenuBar (see page 792)	This is MenuBar, a member of class IWebBrowserAppDisp.
	Name (see page 792)	This is Name, a member of class IWebBrowserAppDisp.
	Parent (see page 793)	This is Parent, a member of class IWebBrowserAppDisp.
	Path (see page 793)	This is Path, a member of class IWebBrowserAppDisp.
	StatusBar (see page 793)	This is StatusBar, a member of class IWebBrowserAppDisp.
	StatusText (see page 793)	This is StatusText, a member of class IWebBrowserAppDisp.
	ToolBar (see page 794)	This is ToolBar, a member of class IWebBrowserAppDisp.
	Top (see page 794)	This is Top, a member of class IWebBrowserAppDisp.
	TopLevelContainer (see page 794)	This is TopLevelContainer, a member of class IWebBrowserAppDisp.
	type_ (see page 794)	This is type_, a member of class IWebBrowserAppDisp.
	Visible (see page 794)	This is Visible, a member of class IWebBrowserAppDisp.
	Width (see page 795)	This is Width, a member of class IWebBrowserAppDisp.

8.2.30.2 IWebBrowserAppDisp Methods

The methods of the IWebBrowserAppDisp class are listed here.

Public Methods

	Name	Description
	ClientToWindow (see page 787)	This is ClientToWindow, a member of class IWebBrowserAppDisp.
	GetProperty (see page 787)	This is GetProperty, a member of class IWebBrowserAppDisp.
	GoBack (see page 787)	This is GoBack, a member of class IWebBrowserAppDisp.
	GoForward (see page 787)	This is GoForward, a member of class IWebBrowserAppDisp.
	GoHome (see page 787)	This is GoHome, a member of class IWebBrowserAppDisp.
	GoSearch (see page 788)	This is GoSearch, a member of class IWebBrowserAppDisp.
	Navigate (see page 788)	This is Navigate, a member of class IWebBrowserAppDisp.
	PutProperty (see page 788)	This is PutProperty, a member of class IWebBrowserAppDisp.
	Quit (see page 788)	This is Quit, a member of class IWebBrowserAppDisp.
	Refresh (see page 789)	This is Refresh, a member of class IWebBrowserAppDisp.
	Refresh2 (see page 789)	This is Refresh2, a member of class IWebBrowserAppDisp.

	Stop (see page 789)	This is Stop, a member of class IWebBrowserAppDisp.

8.2.30.2.1 IWebBrowserAppDisp.ClientToWindow Method

Pascal

```
procedure ClientToWindow(var pcx: SYSINT; var pcy: SYSINT);
```

C++

```
__fastcall ClientToWindow(SYSINT pcx, SYSINT pcy);
```

Description

This is ClientToWindow, a member of class IWebBrowserAppDisp.

8.2.30.2.2 IWebBrowserAppDisp.GetProperty Method

Pascal

```
function GetProperty(const Property_: WideString): OleVariant;
```

C++

```
__fastcall OleVariant GetProperty(const BSTR Property_);
```

Description

This is GetProperty, a member of class IWebBrowserAppDisp.

8.2.30.2.3 IWebBrowserAppDisp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowserAppDisp.

8.2.30.2.4 IWebBrowserAppDisp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowserAppDisp.

8.2.30.2.5 IWebBrowserAppDisp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowserAppDisp.

8.2.30.2.6 IWebBrowserAppDisp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowserAppDisp.

8.2.30.2.7 IWebBrowserAppDisp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowserAppDisp.

8.2.30.2.8 IWebBrowserAppDisp.PutProperty Method

Pascal

```
procedure PutProperty(const Property_: WideString; vtValue: OleVariant);
```

C++

```
__fastcall PutProperty(const BSTR Property_, OleVariant vtValue);
```

Description

This is PutProperty, a member of class IWebBrowserAppDisp.

8.2.30.2.9 IWebBrowserAppDisp.Quit Method

Pascal

```
procedure Quit;
```

C++

```
__fastcall Quit();
```

Description

This is Quit, a member of class IWebBrowserAppDisp.

8.2.30.2.10 IWebBrowserAppDisp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowserAppDisp.

8.2.30.2.11 IWebBrowserAppDisp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowserAppDisp.

8.2.30.2.12 IWebBrowserAppDisp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowserAppDisp.

8.2.30.3 IWebBrowserAppDisp Properties

The properties of the IWebBrowserAppDisp class are listed here.

Public Properties

	Name	Description
	Application (see page 790)	This is Application, a member of class IWebBrowserAppDisp.
	Busy (see page 790)	This is Busy, a member of class IWebBrowserAppDisp.
	Container (see page 790)	This is Container, a member of class IWebBrowserAppDisp.
	Document (see page 790)	This is Document, a member of class IWebBrowserAppDisp.
	FullName (see page 791)	This is FullName, a member of class IWebBrowserAppDisp.
	FullScreen (see page 791)	This is FullScreen, a member of class IWebBrowserAppDisp.
	Height (see page 791)	This is Height, a member of class IWebBrowserAppDisp.
	HWnd (see page 791)	This is HWnd, a member of class IWebBrowserAppDisp.
	Left (see page 792)	This is Left, a member of class IWebBrowserAppDisp.
	LocationName (see page 792)	This is LocationName, a member of class IWebBrowserAppDisp.
	LocationURL (see page 792)	This is LocationURL, a member of class IWebBrowserAppDisp.
	MenuBar (see page 792)	This is MenuBar, a member of class IWebBrowserAppDisp.

	Name (see page 792)	This is Name, a member of class IWebBrowserAppDisp.
	Parent (see page 793)	This is Parent, a member of class IWebBrowserAppDisp.
	Path (see page 793)	This is Path, a member of class IWebBrowserAppDisp.
	StatusBar (see page 793)	This is StatusBar, a member of class IWebBrowserAppDisp.
	StatusText (see page 793)	This is StatusText, a member of class IWebBrowserAppDisp.
	ToolBar (see page 794)	This is ToolBar, a member of class IWebBrowserAppDisp.
	Top (see page 794)	This is Top, a member of class IWebBrowserAppDisp.
	TopLevelContainer (see page 794)	This is TopLevelContainer, a member of class IWebBrowserAppDisp.
	type_ (see page 794)	This is type_, a member of class IWebBrowserAppDisp.
	Visible (see page 794)	This is Visible, a member of class IWebBrowserAppDisp.
	Width (see page 795)	This is Width, a member of class IWebBrowserAppDisp.

8.2.30.3.1 IWebBrowserAppDisp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowserAppDisp.

8.2.30.3.2 IWebBrowserAppDisp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowserAppDisp.

8.2.30.3.3 IWebBrowserAppDisp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowserAppDisp.

8.2.30.3.4 IWebBrowserAppDisp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowserAppDisp.

8.2.30.3.5 IWebBrowserAppDisp.FullName Property

Pascal

```
property FullName: WideString;
```

C++

```
__property BSTR FullName;
```

Description

This is FullName, a member of class IWebBrowserAppDisp.

8.2.30.3.6 IWebBrowserAppDisp.FullScreen Property

Pascal

```
property FullScreen: WordBool;
```

C++

```
__property WordBool FullScreen;
```

Description

This is FullScreen, a member of class IWebBrowserAppDisp.

8.2.30.3.7 IWebBrowserAppDisp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowserAppDisp.

8.2.30.3.8 IWebBrowserAppDisp.HWnd Property

Pascal

```
property HWnd: Integer;
```

C++

```
__property int HWnd;
```

Description

This is HWnd, a member of class IWebBrowserAppDisp.

8.2.30.3.9 IWebBrowserAppDisp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowserAppDisp.

8.2.30.3.10 IWebBrowserAppDisp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowserAppDisp.

8.2.30.3.11 IWebBrowserAppDisp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowserAppDisp.

8.2.30.3.12 IWebBrowserAppDisp.MenuBar Property

Pascal

```
property MenuBar: WordBool;
```

C++

```
__property WordBool MenuBar;
```

Description

This is MenuBar, a member of class IWebBrowserAppDisp.

8.2.30.3.13 IWebBrowserAppDisp.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IWebBrowserAppDisp.

8.2.30.3.14 IWebBrowserAppDisp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowserAppDisp.

8.2.30.3.15 IWebBrowserAppDisp.Path Property

Pascal

```
property Path: WideString;
```

C++

```
__property BSTR Path;
```

Description

This is Path, a member of class IWebBrowserAppDisp.

8.2.30.3.16 IWebBrowserAppDisp.StatusBar Property

Pascal

```
property StatusBar: WordBool;
```

C++

```
__property WordBool StatusBar;
```

Description

This is StatusBar, a member of class IWebBrowserAppDisp.

8.2.30.3.17 IWebBrowserAppDisp.StatusText Property

Pascal

```
property StatusText: WideString;
```

C++

```
__property BSTR StatusText;
```

Description

This is StatusText, a member of class IWebBrowserAppDisp.

8.2.30.3.18 IWebBrowserAppDisp.ToolBar Property

Pascal

```
property ToolBar: SYSINT;
```

C++

```
__property SYSINT ToolBar;
```

Description

This is ToolBar, a member of class IWebBrowserAppDisp.

8.2.30.3.19 IWebBrowserAppDisp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowserAppDisp.

8.2.30.3.20 IWebBrowserAppDisp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowserAppDisp.

8.2.30.3.21 IWebBrowserAppDisp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowserAppDisp.

8.2.30.3.22 IWebBrowserAppDisp.Visible Property

Pascal

```
property Visible: WordBool;
```

C++

```
__property WordBool Visible;
```

Description

This is Visible, a member of class IWebBrowserAppDisp.

8.2.30.3.23 IWebBrowserAppDisp.Width Property

Pascal

```
property Width: Integer;
```

C++

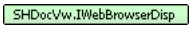
```
__property int Width;
```

Description

This is Width, a member of class IWebBrowserAppDisp.

8.2.31 IWebBrowserDisp Interface

Class Hierarchy



Pascal

```
IWebBrowserDisp = dispinterface;
```

C++

```
class IWebBrowserDisp;
```

File

SHDocVw

Description

This is class SHDocVw.IWebBrowserDisp.

8.2.31.1 IWebBrowserDisp Members

The following tables list the members exposed by IWebBrowserDisp.

Public Methods

	Name	Description
	GoBack (see page 796)	This is GoBack, a member of class IWebBrowserDisp.
	GoForward (see page 796)	This is GoForward, a member of class IWebBrowserDisp.
	GoHome (see page 797)	This is GoHome, a member of class IWebBrowserDisp.
	GoSearch (see page 797)	This is GoSearch, a member of class IWebBrowserDisp.
	Navigate (see page 797)	This is Navigate, a member of class IWebBrowserDisp.
	Refresh (see page 797)	This is Refresh, a member of class IWebBrowserDisp.
	Refresh2 (see page 797)	This is Refresh2, a member of class IWebBrowserDisp.
	Stop (see page 798)	This is Stop, a member of class IWebBrowserDisp.

Public Properties

	Name	Description
	Application (see page 798)	This is Application, a member of class IWebBrowserDisp.

	Busy (see page 799)	This is Busy, a member of class IWebBrowserDisp.
	Container (see page 799)	This is Container, a member of class IWebBrowserDisp.
	Document (see page 799)	This is Document, a member of class IWebBrowserDisp.
	Height (see page 799)	This is Height, a member of class IWebBrowserDisp.
	Left (see page 799)	This is Left, a member of class IWebBrowserDisp.
	LocationName (see page 800)	This is LocationName, a member of class IWebBrowserDisp.
	LocationURL (see page 800)	This is LocationURL, a member of class IWebBrowserDisp.
	Parent (see page 800)	This is Parent, a member of class IWebBrowserDisp.
	Top (see page 800)	This is Top, a member of class IWebBrowserDisp.
	TopLevelContainer (see page 801)	This is TopLevelContainer, a member of class IWebBrowserDisp.
	type_ (see page 801)	This is type_, a member of class IWebBrowserDisp.
	Width (see page 801)	This is Width, a member of class IWebBrowserDisp.

8.2.31.2 IWebBrowserDisp Methods

The methods of the IWebBrowserDisp class are listed here.

Public Methods

	Name	Description
	GoBack (see page 796)	This is GoBack, a member of class IWebBrowserDisp.
	GoForward (see page 796)	This is GoForward, a member of class IWebBrowserDisp.
	GoHome (see page 797)	This is GoHome, a member of class IWebBrowserDisp.
	GoSearch (see page 797)	This is GoSearch, a member of class IWebBrowserDisp.
	Navigate (see page 797)	This is Navigate, a member of class IWebBrowserDisp.
	Refresh (see page 797)	This is Refresh, a member of class IWebBrowserDisp.
	Refresh2 (see page 797)	This is Refresh2, a member of class IWebBrowserDisp.
	Stop (see page 798)	This is Stop, a member of class IWebBrowserDisp.

8.2.31.2.1 IWebBrowserDisp.GoBack Method

Pascal

```
procedure GoBack;
```

C++

```
__fastcall GoBack();
```

Description

This is GoBack, a member of class IWebBrowserDisp.

8.2.31.2.2 IWebBrowserDisp.GoForward Method

Pascal

```
procedure GoForward;
```

C++

```
__fastcall GoForward();
```

Description

This is GoForward, a member of class IWebBrowserDisp.

8.2.31.2.3 IWebBrowserDisp.GoHome Method

Pascal

```
procedure GoHome;
```

C++

```
__fastcall GoHome();
```

Description

This is GoHome, a member of class IWebBrowserDisp.

8.2.31.2.4 IWebBrowserDisp.GoSearch Method

Pascal

```
procedure GoSearch;
```

C++

```
__fastcall GoSearch();
```

Description

This is GoSearch, a member of class IWebBrowserDisp.

8.2.31.2.5 IWebBrowserDisp.Navigate Method

Pascal

```
procedure Navigate(const URL: WideString; const Flags: OleVariant; const TargetFrameName:
OleVariant; var PostData: OleVariant; const Headers: OleVariant);
```

C++

```
__fastcall Navigate(const BSTR URL, const OleVariant Flags, const OleVariant TargetFrameName,
OleVariant PostData, const OleVariant Headers);
```

Description

This is Navigate, a member of class IWebBrowserDisp.

8.2.31.2.6 IWebBrowserDisp.Refresh Method

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

This is Refresh, a member of class IWebBrowserDisp.

8.2.31.2.7 IWebBrowserDisp.Refresh2 Method

Pascal

```
procedure Refresh2(var Level: OleVariant);
```

C++

```
__fastcall Refresh2(OleVariant Level);
```

Description

This is Refresh2, a member of class IWebBrowserDisp.

8.2.31.2.8 IWebBrowserDisp.Stop Method

Pascal

```
procedure Stop;
```

C++

```
__fastcall Stop();
```

Description

This is Stop, a member of class IWebBrowserDisp.

8.2.31.3 IWebBrowserDisp Properties

The properties of the IWebBrowserDisp class are listed here.

Public Properties

	Name	Description
	Application (see page 798)	This is Application, a member of class IWebBrowserDisp.
	Busy (see page 799)	This is Busy, a member of class IWebBrowserDisp.
	Container (see page 799)	This is Container, a member of class IWebBrowserDisp.
	Document (see page 799)	This is Document, a member of class IWebBrowserDisp.
	Height (see page 799)	This is Height, a member of class IWebBrowserDisp.
	Left (see page 799)	This is Left, a member of class IWebBrowserDisp.
	LocationName (see page 800)	This is LocationName, a member of class IWebBrowserDisp.
	LocationURL (see page 800)	This is LocationURL, a member of class IWebBrowserDisp.
	Parent (see page 800)	This is Parent, a member of class IWebBrowserDisp.
	Top (see page 800)	This is Top, a member of class IWebBrowserDisp.
	TopLevelContainer (see page 801)	This is TopLevelContainer, a member of class IWebBrowserDisp.
	type_ (see page 801)	This is type_, a member of class IWebBrowserDisp.
	Width (see page 801)	This is Width, a member of class IWebBrowserDisp.

8.2.31.3.1 IWebBrowserDisp.Application Property

Pascal

```
property Application: IDispatch;
```

C++

```
__property IDispatch Application;
```

Description

This is Application, a member of class IWebBrowserDisp.

8.2.31.3.2 IWebBrowserDisp.Busy Property

Pascal

```
property Busy: WordBool;
```

C++

```
__property WordBool Busy;
```

Description

This is Busy, a member of class IWebBrowserDisp.

8.2.31.3.3 IWebBrowserDisp.Container Property

Pascal

```
property Container: IDispatch;
```

C++

```
__property IDispatch Container;
```

Description

This is Container, a member of class IWebBrowserDisp.

8.2.31.3.4 IWebBrowserDisp.Document Property

Pascal

```
property Document: IDispatch;
```

C++

```
__property IDispatch Document;
```

Description

This is Document, a member of class IWebBrowserDisp.

8.2.31.3.5 IWebBrowserDisp.Height Property

Pascal

```
property Height: Integer;
```

C++

```
__property int Height;
```

Description

This is Height, a member of class IWebBrowserDisp.

8.2.31.3.6 IWebBrowserDisp.Left Property

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

This is Left, a member of class IWebBrowserDisp.

8.2.31.3.7 IWebBrowserDisp.LocationName Property

Pascal

```
property LocationName: WideString;
```

C++

```
__property BSTR LocationName;
```

Description

This is LocationName, a member of class IWebBrowserDisp.

8.2.31.3.8 IWebBrowserDisp.LocationURL Property

Pascal

```
property LocationURL: WideString;
```

C++

```
__property BSTR LocationURL;
```

Description

This is LocationURL, a member of class IWebBrowserDisp.

8.2.31.3.9 IWebBrowserDisp.Parent Property

Pascal

```
property Parent: IDispatch;
```

C++

```
__property IDispatch Parent;
```

Description

This is Parent, a member of class IWebBrowserDisp.

8.2.31.3.10 IWebBrowserDisp.Top Property

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

This is Top, a member of class IWebBrowserDisp.

8.2.31.3.11 IWebBrowserDisp.TopLevelContainer Property

Pascal

```
property TopLevelContainer: WordBool;
```

C++

```
__property WordBool TopLevelContainer;
```

Description

This is TopLevelContainer, a member of class IWebBrowserDisp.

8.2.31.3.12 IWebBrowserDisp.type_ Property

Pascal

```
property type_: WideString;
```

C++

```
__property BSTR type_;
```

Description

This is type_, a member of class IWebBrowserDisp.

8.2.31.3.13 IWebBrowserDisp.Width Property

Pascal

```
property Width: Integer;
```

C++

```
__property int Width;
```

Description

This is Width, a member of class IWebBrowserDisp.

8.3 Types

The following table lists types in this documentation.

Types

Name	Description
BrowserNavConstants (see page 804)	This is type SHDocVw.BrowserNavConstants.
CommandStateChangeConstants (see page 804)	Represents an alias for TOleEnum.
CScriptErrorList (see page 804)	This is type SHDocVw.CScriptErrorList.
InternetExplorer (see page 805)	This is type SHDocVw.InternetExplorer.
OLECMDEXECOPT (see page 805)	OLECMDF (see page 806) influences the way in which an OLECMDID (see page 806) command is executed.
OLECMDF (see page 806)	OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.
OLECMDID (see page 806)	OLECMDID represents a command that can be executed by an Automation object.
POleVariant1 (see page 807)	POleVariant1 is a pointer to a System::OleVariant (see page 3274) variable.
RefreshConstants (see page 807)	This is type SHDocVw.RefreshConstants.
SearchAssistantOC (see page 808)	This is type SHDocVw.SearchAssistantOC.
SecureLockIconConstants (see page 808)	This is type SHDocVw.SecureLockIconConstants.
ShellBrowserWindow (see page 808)	This is type SHDocVw.ShellBrowserWindow.
ShellNameSpace (see page 809)	This is type SHDocVw.ShellNameSpace.
ShellUIHelper (see page 809)	This is type SHDocVw.ShellUIHelper.
ShellWindowFindWindowOptions (see page 809)	This is type SHDocVw.ShellWindowFindWindowOptions.
ShellWindows (see page 810)	This is type SHDocVw.ShellWindows.
ShellWindowTypeConstants (see page 810)	This is type SHDocVw.ShellWindowTypeConstants.
tagREADYSTATE (see page 810)	This is type SHDocVw.tagREADYSTATE.
TInternetExplorerBeforeNavigate2 (see page 811)	This is type SHDocVw.TInternetExplorerBeforeNavigate2.
TInternetExplorerClientToHostWindow (see page 811)	This is type SHDocVw.TInternetExplorerClientToHostWindow.
TInternetExplorerCommandStateChange (see page 811)	This is type SHDocVw.TInternetExplorerCommandStateChange.
TInternetExplorerDocumentComplete (see page 812)	This is type SHDocVw.TInternetExplorerDocumentComplete.
TInternetExplorerFileDownload (see page 812)	This is type SHDocVw.TInternetExplorerFileDownload.
TInternetExplorerNavigateComplete2 (see page 812)	This is type SHDocVw.TInternetExplorerNavigateComplete2.
TInternetExplorerNavigateError (see page 813)	This is type SHDocVw.TInternetExplorerNavigateError.
TInternetExplorerNewWindow2 (see page 813)	This is type SHDocVw.TInternetExplorerNewWindow2.
TInternetExplorerOnFullScreen (see page 813)	This is type SHDocVw.TInternetExplorerOnFullScreen.
TInternetExplorerOnMenuBar (see page 814)	This is type SHDocVw.TInternetExplorerOnMenuBar.
TInternetExplorerOnStatusBar (see page 814)	This is type SHDocVw.TInternetExplorerOnStatusBar.
TInternetExplorerOnTheaterMode (see page 814)	This is type SHDocVw.TInternetExplorerOnTheaterMode.
TInternetExplorerOnToolBar (see page 815)	This is type SHDocVw.TInternetExplorerOnToolBar.
TInternetExplorerOnVisible (see page 815)	This is type SHDocVw.TInternetExplorerOnVisible.
TInternetExplorerPrintTemplateInstantiation (see page 815)	This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.
TInternetExplorerPrintTemplateTeardown (see page 816)	This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.
TInternetExplorerPrivacyImpactedStateChange (see page 816)	This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.
TInternetExplorerProgressChange (see page 816)	This is type SHDocVw.TInternetExplorerProgressChange.
TInternetExplorerPropertyChange (see page 817)	This is type SHDocVw.TInternetExplorerPropertyChange.
TInternetExplorerSetSecureLockIcon (see page 817)	This is type SHDocVw.TInternetExplorerSetSecureLockIcon.
TInternetExplorerStatusTextChange (see page 817)	This is type SHDocVw.TInternetExplorerStatusTextChange.
TInternetExplorerTitleChange (see page 818)	This is type SHDocVw.TInternetExplorerTitleChange.
TInternetExplorerUpdatePageStatus (see page 818)	This is type SHDocVw.TInternetExplorerUpdatePageStatus.
TInternetExplorerWindowClosing (see page 818)	This is type SHDocVw.TInternetExplorerWindowClosing.

TInternetExplorerWindowSetHeight (see page 819)	This is type SHDocVw.TInternetExplorerWindowSetHeight.
TInternetExplorerWindowSetLeft (see page 819)	This is type SHDocVw.TInternetExplorerWindowSetLeft.
TInternetExplorerWindowSetResizable (see page 819)	This is type SHDocVw.TInternetExplorerWindowSetResizable.
TInternetExplorerWindowSetTop (see page 820)	This is type SHDocVw.TInternetExplorerWindowSetTop.
TInternetExplorerWindowSetWidth (see page 820)	This is type SHDocVw.TInternetExplorerWindowSetWidth.
TShellNameSpaceFavoritesSelectionChange (see page 820)	This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.
TShellWindowsWindowRegistered (see page 821)	This is type SHDocVw.TShellWindowsWindowRegistered.
TShellWindowsWindowRevoked (see page 821)	This is type SHDocVw.TShellWindowsWindowRevoked.
TWebBrowser_V1BeforeNavigate (see page 821)	This is type SHDocVw.TWebBrowser_V1BeforeNavigate.
TWebBrowser_V1CommandStateChange (see page 822)	This is type SHDocVw.TWebBrowser_V1CommandStateChange.
TWebBrowser_V1FrameBeforeNavigate (see page 822)	This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.
TWebBrowser_V1FrameNavigateComplete (see page 823)	This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.
TWebBrowser_V1FrameNewWindow (see page 823)	This is type SHDocVw.TWebBrowser_V1FrameNewWindow.
TWebBrowser_V1NavigateComplete (see page 823)	This is type SHDocVw.TWebBrowser_V1NavigateComplete.
TWebBrowser_V1NewWindow (see page 824)	This is type SHDocVw.TWebBrowser_V1NewWindow.
TWebBrowser_V1ProgressChange (see page 824)	This is type SHDocVw.TWebBrowser_V1ProgressChange.
TWebBrowser_V1PropertyChange (see page 824)	This is type SHDocVw.TWebBrowser_V1PropertyChange.
TWebBrowser_V1Quit (see page 825)	This is type SHDocVw.TWebBrowser_V1Quit.
TWebBrowser_V1StatusTextChange (see page 825)	This is type SHDocVw.TWebBrowser_V1StatusTextChange.
TWebBrowser_V1TitleChange (see page 825)	This is type SHDocVw.TWebBrowser_V1TitleChange.
TWebBrowserBeforeNavigate2 (see page 826)	TWebBrowserBeforeNavigate2 is the type for the BeforeNavigate2 event handler on TWebBrowser (see page 504).
TWebBrowserClientToHostWindow (see page 826)	This is type SHDocVw.TWebBrowserClientToHostWindow.
TWebBrowserCommandStateChange (see page 827)	TWebBrowserCommandStateChange is the type for the OnCommandStateChange event handler on TWebBrowser (see page 504).
TWebBrowserDocumentComplete (see page 827)	TWebBrowserDocumentComplete is the type for event handlers that respond when the Service Control manager stops a service.
TWebBrowserFileDownload (see page 828)	This is type SHDocVw.TWebBrowserFileDownload.
TWebBrowserNavigateComplete2 (see page 828)	TWebBrowserNavigateComplete2 is the type for the OnNavigateComplete2 event handler on TWebBrowser (see page 504).
TWebBrowserNavigateError (see page 829)	This is type SHDocVw.TWebBrowserNavigateError.
TWebBrowserNewWindow2 (see page 829)	TWebBrowserNewWindow2 is the type for the OnNewWindow2 event handler on TWebBrowser (see page 504).
TWebBrowserOnFullScreen (see page 830)	This is type SHDocVw.TWebBrowserOnFullScreen.
TWebBrowserOnMenuBar (see page 830)	This is type SHDocVw.TWebBrowserOnMenuBar.
TWebBrowserOnStatusBar (see page 830)	This is type SHDocVw.TWebBrowserOnStatusBar.
TWebBrowserOnTheaterMode (see page 831)	This is type SHDocVw.TWebBrowserOnTheaterMode.
TWebBrowserOnToolBar (see page 831)	This is type SHDocVw.TWebBrowserOnToolBar.
TWebBrowserOnVisible (see page 831)	TWebBrowserOnVisible is the type for the OnVisible event handler on TWebBrowser (see page 504).
TWebBrowserPrintTemplateInstantiation (see page 832)	This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.
TWebBrowserPrintTemplateTeardown (see page 832)	This is type SHDocVw.TWebBrowserPrintTemplateTeardown.
TWebBrowserPrivacyImpactedStateChange (see page 832)	This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.
TWebBrowserProgressChange (see page 833)	TWebBrowserProgressChange is the type for the OnProgressChange event handler on TWebBrowser (see page 504).
TWebBrowserPropertyChange (see page 833)	This is type SHDocVw.TWebBrowserPropertyChange.
TWebBrowserSetSecureLockIcon (see page 834)	This is type SHDocVw.TWebBrowserSetSecureLockIcon.
TWebBrowserStatusTextChange (see page 834)	This is type SHDocVw.TWebBrowserStatusTextChange.
TWebBrowserTitleChange (see page 834)	TWebBrowserTitleChange is the type for the OnTitleChange event handler on TWebBrowser (see page 504).
TWebBrowserUpdatePageStatus (see page 835)	This is type SHDocVw.TWebBrowserUpdatePageStatus.
TWebBrowserWindowClosing (see page 835)	This is type SHDocVw.TWebBrowserWindowClosing.
TWebBrowserWindowSetHeight (see page 835)	This is type SHDocVw.TWebBrowserWindowSetHeight.
TWebBrowserWindowSetLeft (see page 836)	This is type SHDocVw.TWebBrowserWindowSetLeft.
TWebBrowserWindowSetResizable (see page 836)	This is type SHDocVw.TWebBrowserWindowSetResizable.
TWebBrowserWindowSetTop (see page 836)	This is type SHDocVw.TWebBrowserWindowSetTop.
TWebBrowserWindowSetWidth (see page 837)	This is type SHDocVw.TWebBrowserWindowSetWidth.

WebBrowser (see page 837)	This is type SHDocVw.WebBrowser.
WebBrowser_V1 (see page 837)	This is type SHDocVw.WebBrowser_V1.

8.3.1 SHDocVw.BrowserNavConstants Type

Pascal

```
BrowserNavConstants = TOleEnum;
```

C++

```
TOleEnum BrowserNavConstants;
```

File

SHDocVw

Description

This is type SHDocVw.BrowserNavConstants.

8.3.2 SHDocVw.CommandStateChangeConstants Type

Represents an alias for TOleEnum.

Pascal

```
CommandStateChangeConstants = TOleEnum;
```

C++

```
TOleEnum CommandStateChangeConstants;
```

File

SHDocVw

Description

CommandStateChangeConstants is used by the IWebBrowser (see page 733) interface for sending update, forward and back commands.

See Also

TOleEnum

CSC_UPDATECOMMANDS (see page 844)

CSC_NAVIGATEFORWARD (see page 844)

CSC_NAVIGATEBACK (see page 844)

8.3.3 SHDocVw.CScriptErrorList Type

Pascal

```
CScriptErrorList = IScriptErrorList;
```

C++

```
IScriptErrorList CScriptErrorList;
```

File

SHDocVw

Description

This is type SHDocVw.CScriptErrorList.

8.3.4 SHDocVw.InternetExplorer Type

Pascal

```
InternetExplorer = IWebBrowser2;
```

C++

```
IWebBrowser2 InternetExplorer;
```

File

SHDocVw

Description

This is type SHDocVw.InternetExplorer.

8.3.5 SHDocVw.OLECMDEXECOPT Type

OLECMDF (see page 806) influences the way in which an OLECMDID (see page 806) command is executed.

Pascal

```
OLECMDEXECOPT = TOleEnum;
```

C++

```
TOleEnum OLECMDEXECOPT;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDEXECOPT indicates what an Automation object should do in response to an OLECMDID (see page 806) command. The following table lists the possible values:

Constant	Value	Meaning
OLECMDEXECOPT_DODEFAULT (see page 855)	0	Use the default behavior, whether prompting the user for input or not.
OLECMDEXECOPT_PROMPTUSER (see page 855)	1	Execute the command after obtaining user input.
OLECMDEXECOPT_DONTPROMPTUSER (see page 855)	2	Execute the command without prompting the user.

OLECMDEXECOPT_SHOWHELP (see page 856)	3	Show help for the command, but do not execute it.

See Also

OLECMDID (see page 806)

8.3.6 SHDocVw.OLECMDF Type

OLECMDF represents the type of support an Automation object provides for a specific command at a specific time.

Pascal

```
OLECMDF = TOleEnum;
```

C++

```
TOleEnum OLECMDF;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDF is a set of flags that together indicate how a command can be used. These values can be or'd together:

Value	Meaning	
OLECMDF_SUPPORTED (see page 858)	The Automation object supports the command (but it may not be enabled).	
OLECMDF_ENABLED (see page 856)	The command is currently enabled.	
OLECMDF_LATCHED (see page 857)	The command is an on/off toggle that is currently on.	
OLECMDF_NINCHED (see page 857)	Not currently used.	
OLECMDF_INVISIBLE (see page 857)	Not currently used.	
OLECMDF_DEFHIDEONCTXTMENU (see page 856)	Not currently used.	

See Also

OLECMDID (see page 806)

8.3.7 SHDocVw.OLECMDID Type

OLECMDID represents a command that can be executed by an Automation object.

Pascal

```
OLECMDID = TOleEnum;
```

C++

```
TOleEnum OLECMDID;
```

File

SHDocVw

Description

In C++ code imported from the shdocvw type library, OLECMDID represents a set of standard commands that can be executed by many Automation objects. Typically, each command represents a command in the user interface of the Automation object, such as a menu command. For specific details on how the Automation object interprets a particular command, see the documentation for the Automation object.

See Also

OLECMDEXECOPT (see page 805)

8.3.8 SHDocVw.POleVariant1 Type

POleVariant1 is a pointer to a System::OleVariant (see page 3274) variable.

Pascal

```
POleVariant1 = ^OleVariant;
```

C++

```
^OleVariant POleVariant1;
```

File

SHDocVw

Description

POleVariant1 is a pointer to a System::OleVariant (see page 3274) variable.

See Also

System::OleVariant (see page 3274)

POleVariant (see page 3770)

8.3.9 SHDocVw.RefreshConstants Type

Pascal

```
RefreshConstants = TOleEnum;
```

C++

```
TOleEnum RefreshConstants;
```

File

SHDocVw

Description

This is type SHDocVw.RefreshConstants.

8.3.10 SHDocVw.SearchAssistantOC Type

Pascal

```
SearchAssistantOC = ISearchAssistantOC3;
```

C++

```
ISearchAssistantOC3 SearchAssistantOC;
```

File

SHDocVw

Description

This is type SHDocVw.SearchAssistantOC.

8.3.11 SHDocVw.SecureLockIconConstants Type

Pascal

```
SecureLockIconConstants = TOleEnum;
```

C++

```
TOleEnum SecureLockIconConstants;
```

File

SHDocVw

Description

This is type SHDocVw.SecureLockIconConstants.

8.3.12 SHDocVw.ShellBrowserWindow Type

Pascal

```
ShellBrowserWindow = IWebBrowser2;
```

C++

```
IWebBrowser2 ShellBrowserWindow;
```

File

SHDocVw

Description

This is type SHDocVw.ShellBrowserWindow.

8.3.13 SHDocVw.ShellNameSpace Type

Pascal

```
ShellNameSpace = IShellNameSpace;
```

C++

```
IShellNameSpace ShellNameSpace;
```

File

SHDocVw

Description

This is type SHDocVw.ShellNameSpace.

8.3.14 SHDocVw.ShellUIHelper Type

Pascal

```
ShellUIHelper = IShellUIHelper;
```

C++

```
IShellUIHelper ShellUIHelper;
```

File

SHDocVw

Description

This is type SHDocVw.ShellUIHelper.

8.3.15 SHDocVw.ShellWindowFindWindowOptions Type

Pascal

```
ShellWindowFindWindowOptions = TOleEnum;
```

C++

```
TOleEnum ShellWindowFindWindowOptions;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindowFindWindowOptions.

8.3.16 SHDocVw.ShellWindows Type

Pascal

```
ShellWindows = IShellWindows;
```

C++

```
IShellWindows ShellWindows;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindows.

8.3.17 SHDocVw.ShellWindowTypeConstants Type

Pascal

```
ShellWindowTypeConstants = TOleEnum;
```

C++

```
TOleEnum ShellWindowTypeConstants;
```

File

SHDocVw

Description

This is type SHDocVw.ShellWindowTypeConstants.

8.3.18 SHDocVw.tagREADYSTATE Type

Pascal

```
tagREADYSTATE = TOleEnum;
```

C++

```
TOleEnum tagREADYSTATE;
```

File

SHDocVw

Description

This is type SHDocVw.tagREADYSTATE.

8.3.19 SHDocVw.TInternetExplorerBeforeNavigate2 Type

Pascal

```
TInternetExplorerBeforeNavigate2 = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant; var Flags: OleVariant; var TargetFrameName: OleVariant; var PostData:
OleVariant; var Headers: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Flags: OleVariant; var
TargetFrameName: OleVariant; var PostData: OleVariant; var Headers: OleVariant; var Cancel:
WordBool) ( TInternetExplorerBeforeNavigate2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerBeforeNavigate2.

8.3.20 SHDocVw.TInternetExplorerClientToHostWindow Type

Pascal

```
TInternetExplorerClientToHostWindow = procedure (ASender: TObject; var CX: Integer; var CY:
Integer) of object;
```

C++

```
(ASender: TObject; var CX: Integer; var CY: Integer) ( TInternetExplorerClientToHostWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerClientToHostWindow.

8.3.21 SHDocVw.TInternetExplorerCommandStateChange Type

Pascal

```
TInternetExplorerCommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) (
TInternetExplorerCommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerCommandStateChange.

8.3.22 SHDocVw.TInternetExplorerDocumentComplete Type

Pascal

```
TInternetExplorerDocumentComplete = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TInternetExplorerDocumentComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerDocumentComplete.

8.3.23 SHDocVw.TInternetExplorerFileDownload Type

Pascal

```
TInternetExplorerFileDownload = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TInternetExplorerFileDownload)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerFileDownload.

8.3.24 SHDocVw.TInternetExplorerNavigateComplete2 Type

Pascal

```
TInternetExplorerNavigateComplete2 = procedure (ASender: TObject; const pDisp: IDispatch; var
URL: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TInternetExplorerNavigateComplete2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNavigateComplete2.

8.3.25 SHDocVw.TInternetExplorerNavigateError Type

Pascal

```
TInternetExplorerNavigateError = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Frame: OleVariant; var StatusCode: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant; var
StatusCode: OleVariant; var Cancel: WordBool) ( TInternetExplorerNavigateError)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNavigateError.

8.3.26 SHDocVw.TInternetExplorerNewWindow2 Type

Pascal

```
TInternetExplorerNewWindow2 = procedure (ASender: TObject; var ppDisp: IDispatch; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; var ppDisp: IDispatch; var Cancel: WordBool) (
TInternetExplorerNewWindow2)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerNewWindow2.

8.3.27 SHDocVw.TInternetExplorerOnFullScreen Type

Pascal

```
TInternetExplorerOnFullScreen = procedure (ASender: TObject; FullScreen: WordBool) of object;
```

C++

```
(ASender: TObject; FullScreen: WordBool) ( TInternetExplorerOnFullScreen)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnFullScreen.

8.3.28 SHDocVw.TInternetExplorerOnMenuBar Type

Pascal

```
TInternetExplorerOnMenuBar = procedure (ASender: TObject; MenuBar: WordBool) of object;
```

C++

```
(ASender: TObject; MenuBar: WordBool) ( TInternetExplorerOnMenuBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnMenuBar.

8.3.29 SHDocVw.TInternetExplorerOnStatusBar Type

Pascal

```
TInternetExplorerOnStatusBar = procedure (ASender: TObject; StatusBar: WordBool) of object;
```

C++

```
(ASender: TObject; StatusBar: WordBool) ( TInternetExplorerOnStatusBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnStatusBar.

8.3.30 SHDocVw.TInternetExplorerOnTheaterMode Type

Pascal

```
TInternetExplorerOnTheaterMode = procedure (ASender: TObject; TheaterMode: WordBool) of object;
```

C++

```
(ASender: TObject; TheaterMode: WordBool) ( TInternetExplorerOnTheaterMode)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnTheaterMode.

8.3.31 SHDocVw.TInternetExplorerOnToolBar Type

Pascal

```
TInternetExplorerOnToolBar = procedure (ASender: TObject; ToolBar: WordBool) of object;
```

C++

```
(ASender: TObject; ToolBar: WordBool) ( TInternetExplorerOnToolBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnToolBar.

8.3.32 SHDocVw.TInternetExplorerOnVisible Type

Pascal

```
TInternetExplorerOnVisible = procedure (ASender: TObject; Visible: WordBool) of object;
```

C++

```
(ASender: TObject; Visible: WordBool) ( TInternetExplorerOnVisible)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerOnVisible.

8.3.33 SHDocVw.TInternetExplorerPrintTemplateInstantiation Type

Pascal

```
TInternetExplorerPrintTemplateInstantiation = procedure (ASender: TObject; const pDisp:
IDispatch) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TInternetExplorerPrintTemplateInstantiation)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrintTemplateInstantiation.

8.3.34 SHDocVw.TInternetExplorerPrintTemplateTeardown Type

Pascal

```
TInternetExplorerPrintTemplateTeardown = procedure (ASender: TObject; const pDisp: IDispatch)
of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TInternetExplorerPrintTemplateTeardown)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrintTemplateTeardown.

8.3.35 SHDocVw.TInternetExplorerPrivacyImpactedStateChange Type

Pascal

```
TInternetExplorerPrivacyImpactedStateChange = procedure (ASender: TObject; bImpacted:
WordBool) of object;
```

C++

```
(ASender: TObject; bImpacted: WordBool) ( TInternetExplorerPrivacyImpactedStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPrivacyImpactedStateChange.

8.3.36 SHDocVw.TInternetExplorerProgressChange Type

Pascal

```
TInternetExplorerProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) (
TInternetExplorerProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerProgressChange.

8.3.37 SHDocVw.TInternetExplorerPropertyChange Type

Pascal

```
TInternetExplorerPropertyChange = procedure (ASender: TObject; const szProperty: WideString)
of object;
```

C++

```
(ASender: TObject; const szProperty: WideString) ( TInternetExplorerPropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerPropertyChange.

8.3.38 SHDocVw.TInternetExplorerSetSecureLockIcon Type

Pascal

```
TInternetExplorerSetSecureLockIcon = procedure (ASender: TObject; SecureLockIcon: Integer) of
object;
```

C++

```
(ASender: TObject; SecureLockIcon: Integer) ( TInternetExplorerSetSecureLockIcon)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerSetSecureLockIcon.

8.3.39 SHDocVw.TInternetExplorerStatusTextChange Type

Pascal

```
TInternetExplorerStatusTextChange = procedure (ASender: TObject; const Text: WideString) of
object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TInternetExplorerStatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerStatusTextChange.

8.3.40 SHDocVw.TInternetExplorerTitleChange Type

Pascal

```
TInternetExplorerTitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TInternetExplorerTitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerTitleChange.

8.3.41 SHDocVw.TInternetExplorerUpdatePageStatus Type

Pascal

```
TInternetExplorerUpdatePageStatus = procedure (ASender: TObject; const pDisp: IDispatch; var
nPage: OleVariant; var fDone: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var nPage: OleVariant; var fDone: OleVariant) (
TInternetExplorerUpdatePageStatus)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerUpdatePageStatus.

8.3.42 SHDocVw.TInternetExplorerWindowClosing Type

Pascal

```
TInternetExplorerWindowClosing = procedure (ASender: TObject; IsChildWindow: WordBool; var
Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; IsChildWindow: WordBool; var Cancel: WordBool) (
TInternetExplorerWindowClosing)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowClosing.

8.3.43 SHDocVw.TInternetExplorerWindowSetHeight Type

Pascal

```
TInternetExplorerWindowSetHeight = procedure (ASender: TObject; Height: Integer) of object;
```

C++

```
(ASender: TObject; Height: Integer) ( TInternetExplorerWindowSetHeight)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetHeight.

8.3.44 SHDocVw.TInternetExplorerWindowSetLeft Type

Pascal

```
TInternetExplorerWindowSetLeft = procedure (ASender: TObject; Left: Integer) of object;
```

C++

```
(ASender: TObject; Left: Integer) ( TInternetExplorerWindowSetLeft)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetLeft.

8.3.45 SHDocVw.TInternetExplorerWindowSetResizable Type

Pascal

```
TInternetExplorerWindowSetResizable = procedure (ASender: TObject; Resizable: WordBool) of
object;
```

C++

```
(ASender: TObject; Resizable: WordBool) ( TInternetExplorerWindowSetResizable)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetResizable.

8.3.46 SHDocVw.TInternetExplorerWindowSetTop Type

Pascal

```
TInternetExplorerWindowSetTop = procedure (ASender: TObject; Top: Integer) of object;
```

C++

```
(ASender: TObject; Top: Integer) ( TInternetExplorerWindowSetTop)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetTop.

8.3.47 SHDocVw.TInternetExplorerWindowSetWidth Type

Pascal

```
TInternetExplorerWindowSetWidth = procedure (ASender: TObject; Width: Integer) of object;
```

C++

```
(ASender: TObject; Width: Integer) ( TInternetExplorerWindowSetWidth)();
```

File

SHDocVw

Description

This is type SHDocVw.TInternetExplorerWindowSetWidth.

8.3.48 SHDocVw.TShellNameSpaceFavoritesSelectionChange Type

Pascal

```
TShellNameSpaceFavoritesSelectionChange = procedure (ASender: TObject; cItems: Integer; hItem:
Integer; const strName: WideString; const strUrl: WideString; cVisits: Integer; const strDate:
WideString; fAvailableOffline: Integer) of object;
```

C++

```
(ASender: TObject; cItems: Integer; hItem: Integer; const strName: WideString; const strUrl:
WideString; cVisits: Integer; const strDate: WideString; fAvailableOffline: Integer) (
TShellNameSpaceFavoritesSelectionChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellNameSpaceFavoritesSelectionChange.

8.3.49 SHDocVw.TShellWindowsWindowRegistered Type

Pascal

```
TShellWindowsWindowRegistered = procedure (ASender: TObject; lCookie: Integer) of object;
```

C++

```
(ASender: TObject; lCookie: Integer) ( TShellWindowsWindowRegistered)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellWindowsWindowRegistered.

8.3.50 SHDocVw.TShellWindowsWindowRevoked Type

Pascal

```
TShellWindowsWindowRevoked = procedure (ASender: TObject; lCookie: Integer) of object;
```

C++

```
(ASender: TObject; lCookie: Integer) ( TShellWindowsWindowRevoked)();
```

File

SHDocVw

Description

This is type SHDocVw.TShellWindowsWindowRevoked.

8.3.51 SHDocVw.TWebBrowser_V1BeforeNavigate Type

Pascal

```
TWebBrowser_V1BeforeNavigate = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool) (
TWebBrowser_V1BeforeNavigate)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1BeforeNavigate.

8.3.52 SHDocVw.TWebBrowser_V1CommandStateChange Type

Pascal

```
TWebBrowser_V1CommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) ( TWebBrowser_V1CommandStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1CommandStateChange.

8.3.53 SHDocVw.TWebBrowser_V1FrameBeforeNavigate Type

Pascal

```
TWebBrowser_V1FrameBeforeNavigate = procedure (ASender: TObject; const URL: WideString; Flags:
Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers:
WideString; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Cancel: WordBool) (
TWebBrowser_V1FrameBeforeNavigate)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameBeforeNavigate.

8.3.54 SHDocVw.TWebBrowser_V1FrameNavigateComplete Type

Pascal

```
TWebBrowser_V1FrameNavigateComplete = procedure (ASender: TObject; const URL: WideString) of object;
```

C++

```
(ASender: TObject; const URL: WideString) ( TWebBrowser_V1FrameNavigateComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameNavigateComplete.

8.3.55 SHDocVw.TWebBrowser_V1FrameNewWindow Type

Pascal

```
TWebBrowser_V1FrameNewWindow = procedure (ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers: WideString; var Processed: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString; var PostData: OleVariant; const Headers: WideString; var Processed: WordBool) ( TWebBrowser_V1FrameNewWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1FrameNewWindow.

8.3.56 SHDocVw.TWebBrowser_V1NavigateComplete Type

Pascal

```
TWebBrowser_V1NavigateComplete = procedure (ASender: TObject; const URL: WideString) of object;
```

C++

```
(ASender: TObject; const URL: WideString) ( TWebBrowser_V1NavigateComplete)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1NavigateComplete.

8.3.57 SHDocVw.TWebBrowser_V1NewWindow Type

Pascal

```
TWebBrowser_V1NewWindow = procedure (ASender: TObject; const URL: WideString; Flags: Integer;
const TargetFrameName: WideString; var PostData: OleVariant; const Headers: WideString; var
Processed: WordBool) of object;
```

C++

```
(ASender: TObject; const URL: WideString; Flags: Integer; const TargetFrameName: WideString;
var PostData: OleVariant; const Headers: WideString; var Processed: WordBool) (
TWebBrowser_V1NewWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1NewWindow.

8.3.58 SHDocVw.TWebBrowser_V1ProgressChange Type

Pascal

```
TWebBrowser_V1ProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) ( TWebBrowser_V1ProgressChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1ProgressChange.

8.3.59 SHDocVw.TWebBrowser_V1PropertyChange Type

Pascal

```
TWebBrowser_V1PropertyChange = procedure (ASender: TObject; const Property_: WideString) of
object;
```

C++

```
(ASender: TObject; const Property_: WideString) ( TWebBrowser_V1PropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1PropertyChange.

8.3.60 SHDocVw.TWebBrowser_V1Quit Type

Pascal

```
TWebBrowser_V1Quit = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TWebBrowser_V1Quit)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1Quit.

8.3.61 SHDocVw.TWebBrowser_V1StatusTextChange Type

Pascal

```
TWebBrowser_V1StatusTextChange = procedure (ASender: TObject; const Text: WideString) of
object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowser_V1StatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1StatusTextChange.

8.3.62 SHDocVw.TWebBrowser_V1TitleChange Type

Pascal

```
TWebBrowser_V1TitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowser_V1TitleChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowser_V1TitleChange.

8.3.63 SHDocVw.TWebBrowserBeforeNavigate2 Type

TWebBrowserBeforeNavigate2 is the type for the BeforeNavigate2 event handler on TWebBrowser (see page 504).

Pascal

```
TWebBrowserBeforeNavigate2 = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Flags: OleVariant; var TargetFrameName: OleVariant; var PostData: OleVariant;
var Headers: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Flags: OleVariant; var
TargetFrameName: OleVariant; var PostData: OleVariant; var Headers: OleVariant; var Cancel:
WordBool) ( TWebBrowserBeforeNavigate2)();
```

File

SHDocVw

Description

Sender is the Web browser control that is about to look up a new URL.

pDisp is the Automation interface of the Web browser control specified by Sender.

URL is the Uniform Resource Locator of the resource the Web browser is looking up. Change this value to redirect the navigation operation to a different resource.

Flags is not currently used.

TargetFrameName is the name of the frame in which the resource will be displayed, or NULL if the resource should not be displayed in a named frame. Change this value to change where the resource is displayed. See the Navigate method for a list of possible values.

PostData contains the data sent to the server when the underlying operation is an HTTP post message. The event handler can change this value before it is sent to the target URL.

Headers contains any headers sent to the servers when the URL represents an HTTP message. HTTP headers specify such things as the intended action required of the server, the type of data, and so on. (See TWebRequest object, whose properties represent many of the more common headers). The event handler can change this value before it is sent to the target URL.

Cancel determines whether the Web browser looks up the specified resource after the event handler exits. Change Cancel to true to cancel the navigation operation.

8.3.64 SHDocVw.TWebBrowserClientToHostWindow Type

Pascal

```
TWebBrowserClientToHostWindow = procedure (ASender: TObject; var CX: Integer; var CY: Integer)
of object;
```

C++

```
(ASender: TObject; var CX: Integer; var CY: Integer) ( TWebBrowserClientToHostWindow)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserClientToHostWindow.

8.3.65 SHDocVw.TWebBrowserCommandStateChange Type

TWebBrowserCommandStateChange is the type for the OnCommandStateChange event handler on TWebBrowser (see page 504).

Pascal

```
TWebBrowserCommandStateChange = procedure (ASender: TObject; Command: Integer; Enable:
WordBool) of object;
```

C++

```
(ASender: TObject; Command: Integer; Enable: WordBool) ( TWebBrowserCommandStateChange)();
```

File

SHDocVw

Description

Sender is the Web browser control whose capabilities have changed.

Command indicates what has changed. The following table lists the possible values:

ConstantValueMeaning

Constant	Value	Meaning
CSC_UPDATECOMMANDS (see page 844)	-1	Any change not covered by the other constants. The application must check the properties of the Web browser to update its controls. For example, the event handler might check the Busy property to update a Stop button.
CSC_NAVIGATEFORWARD (see page 844)	1	The history list changed the ability of the GoForward method to navigate to a new URL. The Enable parameter indicates whether GoForward now navigates to a new URL (true), or not (false).
CSC_NAVIGATEBACK (see page 844)	2	The history list changed the ability of the GoBack method to navigate to a new URL. The Enable parameter indicates whether GoBack now navigates to a new URL (true), or not (false).

Enable indicates whether the CSC_NAVIGATEFORWARD (see page 844) or CSC_NAVIGATEBACK (see page 844) commands should now be enabled (true), or not (false).

8.3.66 SHDocVw.TWebBrowserDocumentComplete Type

TWebBrowserDocumentComplete is the type for event handlers that respond when the Service Control manager stops a service.

Pascal

```
TWebBrowserDocumentComplete = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
```

```
OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TWebBrowserDocumentComplete)();
```

File

SHDocVw

Description

Sender is the Web browser that is loading the document.

pDisp is the Automation interface of the top-level frame or browser. When loading a document without frames, pDisp is the interface of the Web browser. When loading a document with multiple frames, this is the interface of the containing frame, except for the very last time the event occurs, when it is the interface of the Web browser.

URL is the URL, UNC file name, or PIDL that to which the Web browser navigated. This URL can be different from the URL to which the browser was told to navigate. For example, the browser may have been redirected by the target resource or by an OnCommandStateChange event handler. In addition, the value of URL is the canonicalized and qualified URL: for example, if an application specified a URL of "www.borland.com" in a call to the Navigate or Navigate2 method, the URL in the OnDocumentComplete event handler is "http://www.borland.com/".

8.3.67 SHDocVw.TWebBrowserFileDownload Type

Pascal

```
TWebBrowserFileDownload = procedure (ASender: TObject; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; var Cancel: WordBool) ( TWebBrowserFileDownload)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserFileDownload.

8.3.68 SHDocVw.TWebBrowserNavigateComplete2 Type

TWebBrowserNavigateComplete2 is the type for the OnNavigateComplete2 event handler on TWebBrowser (see page 504).

Pascal

```
TWebBrowserNavigateComplete2 = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant) (
TWebBrowserNavigateComplete2)();
```

File

SHDocVw

Description

Sender is the Web browser that navigated to the new resource.

pDisp is the Automation interface of the browser.

URL is the URL, UNC file name, or PIDL that to which the Web browser navigated. This URL can be different from the URL to which the browser was told to navigate. For example, the browser may have been redirected by the target resource or by an OnBeforeNavigate2 event handler. In addition, the value of URL is the canonicalized and qualified URL: for example, if an application specified a URL of "www.borland.com" in a call to the Navigate or Navigate2 method, the URL in the OnDocumentComplete event handler is "http://www.borland.com/".

8.3.69 SHDocVw.TWebBrowserNavigateError Type

Pascal

```
TWebBrowserNavigateError = procedure (ASender: TObject; const pDisp: IDispatch; var URL:
OleVariant; var Frame: OleVariant; var StatusCode: OleVariant; var Cancel: WordBool) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var URL: OleVariant; var Frame: OleVariant; var
StatusCode: OleVariant; var Cancel: WordBool) ( TWebBrowserNavigateError)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserNavigateError.

8.3.70 SHDocVw.TWebBrowserNewWindow2 Type

TWebBrowserNewWindow2 is the type for the OnNewWindow2 event handler on TWebBrowser (see page 504).

Pascal

```
TWebBrowserNewWindow2 = procedure (ASender: TObject; var ppDisp: IDispatch; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; var ppDisp: IDispatch; var Cancel: WordBool) ( TWebBrowserNewWindow2)();
```

File

SHDocVw

Description

Sender is the WebBrowser (see page 837) that needs a new window to display its target resource.

ppDisp optionally returns the interface for a newly created, hidden, TWebBrowser (see page 504) component that can act as the new window. The Web browser configures this component and navigates to the target URL, starting with an OnBeforeNavigate2 event. If the event handler does not create a component and return its interface as the ppDisp parameter, the Web browser generates a top-level window as a separate, nonhosted process.

Cancel allows the event handler to block the creation of a new window. When the event handler sets Cancel to true, the Web

browser tries to display the target resource in its current window, starting with an OnBeforeNavigate2 event.

Note: The event handler should not return a value for ppDisp when setting Cancel to true.

8.3.71 SHDocVw.TWebBrowserOnFullScreen Type

Pascal

```
TWebBrowserOnFullScreen = procedure (ASender: TObject; FullScreen: WordBool) of object;
```

C++

```
(ASender: TObject; FullScreen: WordBool) ( TWebBrowserOnFullScreen)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnFullScreen.

8.3.72 SHDocVw.TWebBrowserOnMenuBar Type

Pascal

```
TWebBrowserOnMenuBar = procedure (ASender: TObject; MenuBar: WordBool) of object;
```

C++

```
(ASender: TObject; MenuBar: WordBool) ( TWebBrowserOnMenuBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnMenuBar.

8.3.73 SHDocVw.TWebBrowserOnStatusBar Type

Pascal

```
TWebBrowserOnStatusBar = procedure (ASender: TObject; StatusBar: WordBool) of object;
```

C++

```
(ASender: TObject; StatusBar: WordBool) ( TWebBrowserOnStatusBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnStatusBar.

8.3.74 SHDocVw.TWebBrowserOnTheaterMode Type

Pascal

```
TWebBrowserOnTheaterMode = procedure (ASender: TObject; TheaterMode: WordBool) of object;
```

C++

```
(ASender: TObject; TheaterMode: WordBool) ( TWebBrowserOnTheaterMode)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnTheaterMode.

8.3.75 SHDocVw.TWebBrowserOnToolBar Type

Pascal

```
TWebBrowserOnToolBar = procedure (ASender: TObject; ToolBar: WordBool) of object;
```

C++

```
(ASender: TObject; ToolBar: WordBool) ( TWebBrowserOnToolBar)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserOnToolBar.

8.3.76 SHDocVw.TWebBrowserOnVisible Type

TWebBrowserOnVisible is the type for the OnVisible event handler on TWebBrowser (see page 504).

Pascal

```
TWebBrowserOnVisible = procedure (ASender: TObject; Visible: WordBool) of object;
```

C++

```
(ASender: TObject; Visible: WordBool) ( TWebBrowserOnVisible)();
```

File

SHDocVw

Description

Sender is the Web browser whose Visible property is changing.

Visible is true if the Web browser is about to appear, false if the Web browser is about to be hidden.

8.3.77 SHDocVw.TWebBrowserPrintTemplateInstantiation Type

Pascal

```
TWebBrowserPrintTemplateInstantiation = procedure (ASender: TObject; const pDisp: IDispatch)
of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TWebBrowserPrintTemplateInstantiation)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrintTemplateInstantiation.

8.3.78 SHDocVw.TWebBrowserPrintTemplateTeardown Type

Pascal

```
TWebBrowserPrintTemplateTeardown = procedure (ASender: TObject; const pDisp: IDispatch) of
object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch) ( TWebBrowserPrintTemplateTeardown)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrintTemplateTeardown.

8.3.79 SHDocVw.TWebBrowserPrivacyImpactedStateChange Type

Pascal

```
TWebBrowserPrivacyImpactedStateChange = procedure (ASender: TObject; bImpacted: WordBool) of
object;
```

C++

```
(ASender: TObject; bImpacted: WordBool) ( TWebBrowserPrivacyImpactedStateChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPrivacyImpactedStateChange.

8.3.80 SHDocVw.TWebBrowserProgressChange Type

TWebBrowserProgressChange is the type for the OnProgressChange event handler on TWebBrowser (see page 504).

Pascal

```
TWebBrowserProgressChange = procedure (ASender: TObject; Progress: Integer; ProgressMax:
Integer) of object;
```

C++

```
(ASender: TObject; Progress: Integer; ProgressMax: Integer) ( TWebBrowserProgressChange)();
```

File

SHDocVw

Description

Sender is the Web browser control that is in the process of downloading a document.

Progress indicates how much of the document has already been downloaded, on a scale of 0 to ProgressMax. When Progress is –1, the operation is finished.

ProgressMax indicates the total size of the download operation.

Tip: To calculate the percentage of progress to show in a progress indicator (when Progress is not –1), multiply the value of Progress by 100 and divide by the value of ProgressMax.

8.3.81 SHDocVw.TWebBrowserPropertyChange Type

Pascal

```
TWebBrowserPropertyChange = procedure (ASender: TObject; const szProperty: WideString) of
object;
```

C++

```
(ASender: TObject; const szProperty: WideString) ( TWebBrowserPropertyChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserPropertyChange.

8.3.82 SHDocVw.TWebBrowserSetSecureLockIcon Type

Pascal

```
TWebBrowserSetSecureLockIcon = procedure (ASender: TObject; SecureLockIcon: Integer) of object;
```

C++

```
(ASender: TObject; SecureLockIcon: Integer) ( TWebBrowserSetSecureLockIcon)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserSetSecureLockIcon.

8.3.83 SHDocVw.TWebBrowserStatusTextChange Type

Pascal

```
TWebBrowserStatusTextChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowserStatusTextChange)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserStatusTextChange.

8.3.84 SHDocVw.TWebBrowserTitleChange Type

TWebBrowserTitleChange is the type for the OnTitleChange event handler on TWebBrowser (see page 504).

Pascal

```
TWebBrowserTitleChange = procedure (ASender: TObject; const Text: WideString) of object;
```

C++

```
(ASender: TObject; const Text: WideString) ( TWebBrowserTitleChange)();
```

File

SHDocVw

Description

Sender is the Web browser that is unloading a document.

Text is the newly-available title of the document.

8.3.85 SHDocVw.TWebBrowserUpdatePageStatus Type

Pascal

```
TWebBrowserUpdatePageStatus = procedure (ASender: TObject; const pDisp: IDispatch; var nPage:
OleVariant; var fDone: OleVariant) of object;
```

C++

```
(ASender: TObject; const pDisp: IDispatch; var nPage: OleVariant; var fDone: OleVariant) (
TWebBrowserUpdatePageStatus)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserUpdatePageStatus.

8.3.86 SHDocVw.TWebBrowserWindowClosing Type

Pascal

```
TWebBrowserWindowClosing = procedure (ASender: TObject; IsChildWindow: WordBool; var Cancel:
WordBool) of object;
```

C++

```
(ASender: TObject; IsChildWindow: WordBool; var Cancel: WordBool) (
TWebBrowserWindowClosing)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowClosing.

8.3.87 SHDocVw.TWebBrowserWindowSetHeight Type

Pascal

```
TWebBrowserWindowSetHeight = procedure (ASender: TObject; Height: Integer) of object;
```

C++

```
(ASender: TObject; Height: Integer) ( TWebBrowserWindowSetHeight)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetHeight.

8.3.88 SHDocVw.TWebBrowserWindowSetLeft Type

Pascal

```
TWebBrowserWindowSetLeft = procedure (ASender: TObject; Left: Integer) of object;
```

C++

```
(ASender: TObject; Left: Integer) ( TWebBrowserWindowSetLeft)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetLeft.

8.3.89 SHDocVw.TWebBrowserWindowSetResizable Type

Pascal

```
TWebBrowserWindowSetResizable = procedure (ASender: TObject; Resizable: WordBool) of object;
```

C++

```
(ASender: TObject; Resizable: WordBool) ( TWebBrowserWindowSetResizable)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetResizable.

8.3.90 SHDocVw.TWebBrowserWindowSetTop Type

Pascal

```
TWebBrowserWindowSetTop = procedure (ASender: TObject; Top: Integer) of object;
```

C++

```
(ASender: TObject; Top: Integer) ( TWebBrowserWindowSetTop)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetTop.

8.3.91 SHDocVw.TWebBrowserWindowSetWidth Type

Pascal

```
TWebBrowserWindowSetWidth = procedure (ASender: TObject; Width: Integer) of object;
```

C++

```
(ASender: TObject; Width: Integer) ( TWebBrowserWindowSetWidth)();
```

File

SHDocVw

Description

This is type SHDocVw.TWebBrowserWindowSetWidth.

8.3.92 SHDocVw.WebBrowser Type

Pascal

```
WebBrowser = IWebBrowser2;
```

C++

```
IWebBrowser2 WebBrowser;
```

File

SHDocVw

Description

This is type SHDocVw.WebBrowser.

8.3.93 SHDocVw.WebBrowser_V1 Type

Pascal

```
WebBrowser_V1 = IWebBrowser;
```

C++

```
IWebBrowser WebBrowser_V1;
```

File

SHDocVw

Description

This is type SHDocVw.WebBrowser_V1.

8.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CGID_IWebBrowser (see page 840)	This is constant SHDocVw.CGID_IWebBrowser.
CLASS_CScriptErrorList (see page 841)	This is constant SHDocVw.CLASS_CScriptErrorList.
CLASS_InternetExplorer (see page 841)	This is constant SHDocVw.CLASS_InternetExplorer.
CLASS_SearchAssistantOC (see page 841)	This is constant SHDocVw.CLASS_SearchAssistantOC.
CLASS_ShellBrowserWindow (see page 842)	This is constant SHDocVw.CLASS_ShellBrowserWindow.
CLASS_ShellNameSpace (see page 842)	This is constant SHDocVw.CLASS_ShellNameSpace.
CLASS_ShellUIHelper (see page 842)	This is constant SHDocVw.CLASS_ShellUIHelper.
CLASS_ShellWindows (see page 843)	This is constant SHDocVw.CLASS_ShellWindows.
CLASS_WebBrowser (see page 843)	This is constant SHDocVw.CLASS_WebBrowser.
CLASS_WebBrowser_V1 (see page 843)	This is constant SHDocVw.CLASS_WebBrowser_V1.
CSC_NAVIGATEBACK (see page 844)	This is constant SHDocVw.CSC_NAVIGATEBACK.
CSC_NAVIGATEFORWARD (see page 844)	This is constant SHDocVw.CSC_NAVIGATEFORWARD.
CSC_UPDATECOMMANDS (see page 844)	This is constant SHDocVw.CSC_UPDATECOMMANDS.
DIID__SearchAssistantEvents (see page 845)	This is constant SHDocVw.DIID__SearchAssistantEvents.
DIID_DShellNameSpaceEvents (see page 845)	This is constant SHDocVw.DIID_DShellNameSpaceEvents.
DIID_DShellWindowsEvents (see page 845)	This is constant SHDocVw.DIID_DShellWindowsEvents.
DIID_DWebBrowserEvents (see page 846)	This is constant SHDocVw.DIID_DWebBrowserEvents.
DIID_DWebBrowserEvents2 (see page 846)	This is constant SHDocVw.DIID_DWebBrowserEvents2.
dtlOcxPage (see page 846)	This is constant SHDocVw.dtlOcxPage.
dtlServerPage (see page 847)	This is constant SHDocVw.dtlServerPage.
HTMLID_FIND (see page 847)	This is constant SHDocVw.HTMLID_FIND.
HTMLID_OPTIONS (see page 847)	This is constant SHDocVw.HTMLID_OPTIONS.
HTMLID_VIEWSOURCE (see page 848)	This is constant SHDocVw.HTMLID_VIEWSOURCE.
IID_IScriptErrorList (see page 848)	This is constant SHDocVw.IID_IScriptErrorList.
IID_ISearch (see page 848)	This is constant SHDocVw.IID_ISearch.
IID_ISearchAssistantOC (see page 849)	This is constant SHDocVw.IID_ISearchAssistantOC.
IID_ISearchAssistantOC2 (see page 849)	This is constant SHDocVw.IID_ISearchAssistantOC2.
IID_ISearchAssistantOC3 (see page 849)	This is constant SHDocVw.IID_ISearchAssistantOC3.
IID_ISearches (see page 850)	This is constant SHDocVw.IID_ISearches.
IID_IShellFavoritesNameSpace (see page 850)	This is constant SHDocVw.IID_IShellFavoritesNameSpace.
IID_IShellNameSpace (see page 850)	This is constant SHDocVw.IID_IShellNameSpace.
IID_IShellUIHelper (see page 851)	This is constant SHDocVw.IID_IShellUIHelper.
IID_IShellWindows (see page 851)	This is constant SHDocVw.IID_IShellWindows.
IID_IWebBrowser (see page 851)	This is constant SHDocVw.IID_IWebBrowser.
IID_IWebBrowser2 (see page 852)	This is constant SHDocVw.IID_IWebBrowser2.
IID_IWebBrowserApp (see page 852)	This is constant SHDocVw.IID_IWebBrowserApp.
LIBID_SHDocVw (see page 852)	This is constant SHDocVw.LIBID_SHDocVw.
navAllowAutosearch (see page 853)	This is constant SHDocVw.navAllowAutosearch.
navBrowserBar (see page 853)	This is constant SHDocVw.navBrowserBar.
navNoHistory (see page 853)	This is constant SHDocVw.navNoHistory.
navNoReadFromCache (see page 854)	This is constant SHDocVw.navNoReadFromCache.
navNoWriteToCache (see page 854)	This is constant SHDocVw.navNoWriteToCache.
navOpenInNewWindow (see page 854)	This is constant SHDocVw.navOpenInNewWindow.
OLECMDEXECOPT_DODEFAULT (see page 855)	This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.

OLECMDEXECOPT_DONTPROMPTUSER (see page 855)	This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.
OLECMDEXECOPT_PROMPTUSER (see page 855)	This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.
OLECMDEXECOPT_SHOWHELP (see page 856)	This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.
OLECMDF_DEFHIDEONCTXTMENU (see page 856)	This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.
OLECMDF_ENABLED (see page 856)	This is constant SHDocVw.OLECMDF_ENABLED.
OLECMDF_INVISIBLE (see page 857)	This is constant SHDocVw.OLECMDF_INVISIBLE.
OLECMDF_LATCHED (see page 857)	This is constant SHDocVw.OLECMDF_LATCHED.
OLECMDF_NINCHED (see page 857)	This is constant SHDocVw.OLECMDF_NINCHED.
OLECMDF_SUPPORTED (see page 858)	This is constant SHDocVw.OLECMDF_SUPPORTED.
OLECMDID_ALLOWUILESSSAVEAS (see page 858)	This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.
OLECMDID_CLEARSELECTION (see page 858)	This is constant SHDocVw.OLECMDID_CLEARSELECTION.
OLECMDID_CLOSE (see page 859)	This is constant SHDocVw.OLECMDID_CLOSE.
OLECMDID_COPY (see page 859)	This is constant SHDocVw.OLECMDID_COPY.
OLECMDID_CUT (see page 859)	This is constant SHDocVw.OLECMDID_CUT.
OLECMDID_DELETE (see page 860)	This is constant SHDocVw.OLECMDID_DELETE.
OLECMDID_DONTDOWNLOADCSS (see page 860)	This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.
OLECMDID_ENABLE_INTERACTION (see page 860)	This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.
OLECMDID_FIND (see page 861)	This is constant SHDocVw.OLECMDID_FIND.
OLECMDID_GETPRINTTEMPLATE (see page 861)	This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.
OLECMDID_GETZOOMRANGE (see page 861)	This is constant SHDocVw.OLECMDID_GETZOOMRANGE.
OLECMDID_HIDETOOLBARS (see page 862)	This is constant SHDocVw.OLECMDID_HIDETOOLBARS.
OLECMDID_HTTPEQUIV (see page 862)	This is constant SHDocVw.OLECMDID_HTTPEQUIV.
OLECMDID_HTTPEQUIV_DONE (see page 862)	This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.
OLECMDID_NEW (see page 863)	This is constant SHDocVw.OLECMDID_NEW.
OLECMDID_ONTOOLBARACTIVATED (see page 863)	This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.
OLECMDID_ONUNLOAD (see page 863)	This is constant SHDocVw.OLECMDID_ONUNLOAD.
OLECMDID_OPEN (see page 864)	This is constant SHDocVw.OLECMDID_OPEN.
OLECMDID_PAGESETUP (see page 864)	This is constant SHDocVw.OLECMDID_PAGESETUP.
OLECMDID_PASTE (see page 864)	This is constant SHDocVw.OLECMDID_PASTE.
OLECMDID_PASTESPECIAL (see page 865)	This is constant SHDocVw.OLECMDID_PASTESPECIAL.
OLECMDID_PREREFRESH (see page 865)	This is constant SHDocVw.OLECMDID_PREREFRESH.
OLECMDID_PRINT (see page 865)	This is constant SHDocVw.OLECMDID_PRINT.
OLECMDID_PRINT2 (see page 866)	This is constant SHDocVw.OLECMDID_PRINT2.
OLECMDID_PRINTPREVIEW (see page 866)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW.
OLECMDID_PRINTPREVIEW2 (see page 866)	This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.
OLECMDID_PROPERTIES (see page 867)	This is constant SHDocVw.OLECMDID_PROPERTIES.
OLECMDID_PROPERTYBAG2 (see page 867)	This is constant SHDocVw.OLECMDID_PROPERTYBAG2.
OLECMDID_REDO (see page 867)	This is constant SHDocVw.OLECMDID_REDO.
OLECMDID_REFRESH (see page 868)	This is constant SHDocVw.OLECMDID_REFRESH.
OLECMDID_SAVE (see page 868)	This is constant SHDocVw.OLECMDID_SAVE.
OLECMDID_SAVEAS (see page 868)	This is constant SHDocVw.OLECMDID_SAVEAS.
OLECMDID_SAVECOPYAS (see page 869)	This is constant SHDocVw.OLECMDID_SAVECOPYAS.
OLECMDID_SELECTALL (see page 869)	This is constant SHDocVw.OLECMDID_SELECTALL.
OLECMDID_SETDOWNLOADSTATE (see page 869)	This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.
OLECMDID_SETPRINTTEMPLATE (see page 870)	This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.
OLECMDID_SETPROGRESSMAX (see page 870)	This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.
OLECMDID_SETPROGRESSPOS (see page 870)	This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.
OLECMDID_SETPROGRESSTEXT (see page 871)	This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.
OLECMDID_SETTITLE (see page 871)	This is constant SHDocVw.OLECMDID_SETTITLE.
OLECMDID_SHOWFIND (see page 871)	This is constant SHDocVw.OLECMDID_SHOWFIND.
OLECMDID_SHOWMESSAGE (see page 872)	This is constant SHDocVw.OLECMDID_SHOWMESSAGE.
OLECMDID_SHOWPAGESETUP (see page 872)	This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.
OLECMDID_SHOWPRINT (see page 872)	This is constant SHDocVw.OLECMDID_SHOWPRINT.
OLECMDID_SHOWSCRIPTERROR (see page 873)	This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.

OLECMDID_SPELL (see page 873)	This is constant SHDocVw.OLECMDID_SPELL.
OLECMDID_STOP (see page 873)	This is constant SHDocVw.OLECMDID_STOP.
OLECMDID_STOPDOWNLOAD (see page 874)	This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.
OLECMDID_UNDO (see page 874)	This is constant SHDocVw.OLECMDID_UNDO.
OLECMDID_UPDATECOMMANDS (see page 874)	This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.
OLECMDID_UPDATEPAGESTATUS (see page 875)	This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.
OLECMDID_ZOOM (see page 875)	This is constant SHDocVw.OLECMDID_ZOOM.
READYSTATE_COMPLETE (see page 875)	This is constant SHDocVw.READYSTATE_COMPLETE.
READYSTATE_INTERACTIVE (see page 876)	This is constant SHDocVw.READYSTATE_INTERACTIVE.
READYSTATE_LOADED (see page 876)	This is constant SHDocVw.READYSTATE_LOADED.
READYSTATE_LOADING (see page 876)	This is constant SHDocVw.READYSTATE_LOADING.
READYSTATE_UNINITIALIZED (see page 877)	This is constant SHDocVw.READYSTATE_UNINITIALIZED.
REFRESH_COMPLETELY (see page 877)	This is constant SHDocVw.REFRESH_COMPLETELY.
REFRESH_CONTINUE (see page 877)	This is constant SHDocVw.REFRESH_CONTINUE.
REFRESH_IFEXPIRED (see page 878)	This is constant SHDocVw.REFRESH_IFEXPIRED.
REFRESH_NORMAL (see page 878)	This is constant SHDocVw.REFRESH_NORMAL.
secureLockIconMixed (see page 878)	This is constant SHDocVw.secureLockIconMixed.
secureLockIconSecure128Bit (see page 879)	This is constant SHDocVw.secureLockIconSecure128Bit.
secureLockIconSecure40Bit (see page 879)	This is constant SHDocVw.secureLockIconSecure40Bit.
secureLockIconSecure56Bit (see page 879)	This is constant SHDocVw.secureLockIconSecure56Bit.
secureLockIconSecureFortezza (see page 880)	This is constant SHDocVw.secureLockIconSecureFortezza.
secureLockIconSecureUnknownBits (see page 880)	This is constant SHDocVw.secureLockIconSecureUnknownBits.
secureLockIconUnsecure (see page 880)	This is constant SHDocVw.secureLockIconUnsecure.
SHDocVwMajorVersion (see page 881)	This is constant SHDocVw.SHDocVwMajorVersion.
SHDocVwMinorVersion (see page 881)	This is constant SHDocVw.SHDocVwMinorVersion.
SWC_3RDPARTY (see page 881)	This is constant SHDocVw.SWC_3RDPARTY.
SWC_BROWSER (see page 882)	This is constant SHDocVw.SWC_BROWSER.
SWC_CALLBACK (see page 882)	This is constant SHDocVw.SWC_CALLBACK.
SWC_EXPLORER (see page 882)	This is constant SHDocVw.SWC_EXPLORER.
SWFO_COOKIEPASSED (see page 883)	This is constant SHDocVw.SWFO_COOKIEPASSED.
SWFO_INCLUDEPENDING (see page 883)	This is constant SHDocVw.SWFO_INCLUDEPENDING.
SWFO_NEEDDISPATCH (see page 883)	This is constant SHDocVw.SWFO_NEEDDISPATCH.

8.4.1 SHDocVw.CGID_IWebBrowser Constant

Pascal

```
CGID_IWebBrowser: TGUID = '{ED016940-BD5B-11CF-BA4E-00C04FD70816}';
```

C++

```
TGUID CGID_IWebBrowser = '{ED016940-BD5B-11CF-BA4E-00C04FD70816}';
```

File

SHDocVw

Description

This is constant SHDocVw.CGID_IWebBrowser.

8.4.2 SHDocVw.CLASS_CScriptErrorList Constant

Pascal

```
CLASS_CScriptErrorList: TGUID = '{EFD01300-160F-11D2-BB2E-00805FF7EFCA}';
```

C++

```
TGUID CLASS_CScriptErrorList = '{EFD01300-160F-11D2-BB2E-00805FF7EFCA}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_CScriptErrorList.

8.4.3 SHDocVw.CLASS_InternetExplorer Constant

Pascal

```
CLASS_InternetExplorer: TGUID = '{0002DF01-0000-0000-C000-000000000046}';
```

C++

```
TGUID CLASS_InternetExplorer = '{0002DF01-0000-0000-C000-000000000046}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_InternetExplorer.

8.4.4 SHDocVw.CLASS_SearchAssistantOC Constant

Pascal

```
CLASS_SearchAssistantOC: TGUID = '{B45FF030-4447-11D2-85DE-00C04FA35C89}';
```

C++

```
TGUID CLASS_SearchAssistantOC = '{B45FF030-4447-11D2-85DE-00C04FA35C89}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_SearchAssistantOC.

8.4.5 SHDocVw.CLASS_ShellBrowserWindow Constant

Pascal

```
CLASS_ShellBrowserWindow: TGUID = '{C08AFD90-F2A1-11D1-8455-00A0C91F3880}';
```

C++

```
TGUID CLASS_ShellBrowserWindow = '{C08AFD90-F2A1-11D1-8455-00A0C91F3880}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellBrowserWindow.

8.4.6 SHDocVw.CLASS_ShellNameSpace Constant

Pascal

```
CLASS_ShellNameSpace: TGUID = '{55136805-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID CLASS_ShellNameSpace = '{55136805-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellNameSpace.

8.4.7 SHDocVw.CLASS_ShellUIHelper Constant

Pascal

```
CLASS_ShellUIHelper: TGUID = '{64AB4BB7-111E-11D1-8F79-00C04FC2FBE1}';
```

C++

```
TGUID CLASS_ShellUIHelper = '{64AB4BB7-111E-11D1-8F79-00C04FC2FBE1}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellUIHelper.

8.4.8 SHDocVw.CLASS_ShellWindows Constant

Pascal

```
CLASS_ShellWindows: TGUID = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39}';
```

C++

```
TGUID CLASS_ShellWindows = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_ShellWindows.

8.4.9 SHDocVw.CLASS_WebBrowser Constant

Pascal

```
CLASS_WebBrowser: TGUID = '{8856F961-340A-11D0-A96B-00C04FD705A2}';
```

C++

```
TGUID CLASS_WebBrowser = '{8856F961-340A-11D0-A96B-00C04FD705A2}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_WebBrowser.

8.4.10 SHDocVw.CLASS_WebBrowser_V1 Constant

Pascal

```
CLASS_WebBrowser_V1: TGUID = '{EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID CLASS_WebBrowser_V1 = '{EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.CLASS_WebBrowser_V1.

8.4.11 SHDocVw.CSC_NAVIGATEBACK Constant

Pascal

```
CSC_NAVIGATEBACK = $00000002;
```

C++

```
CSC_NAVIGATEBACK = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_NAVIGATEBACK.

8.4.12 SHDocVw.CSC_NAVIGATEFORWARD Constant

Pascal

```
CSC_NAVIGATEFORWARD = $00000001;
```

C++

```
CSC_NAVIGATEFORWARD = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_NAVIGATEFORWARD.

8.4.13 SHDocVw.CSC_UPDATECOMMANDS Constant

Pascal

```
CSC_UPDATECOMMANDS = $FFFFFFFF;
```

C++

```
CSC_UPDATECOMMANDS = 0xFFFFFFFF;
```

File

SHDocVw

Description

This is constant SHDocVw.CSC_UPDATECOMMANDS.

8.4.14 SHDocVw.DIID__SearchAssistantEvents Constant

Pascal

```
DIID__SearchAssistantEvents: TGUID = '{1611FDDA-445B-11D2-85DE-00C04FA35C89}';
```

C++

```
TGUID DIID__SearchAssistantEvents = '{1611FDDA-445B-11D2-85DE-00C04FA35C89}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID__SearchAssistantEvents.

8.4.15 SHDocVw.DIID_DShellNameSpaceEvents Constant

Pascal

```
DIID_DShellNameSpaceEvents: TGUID = '{55136806-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID DIID_DShellNameSpaceEvents = '{55136806-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DShellNameSpaceEvents.

8.4.16 SHDocVw.DIID_DShellWindowsEvents Constant

Pascal

```
DIID_DShellWindowsEvents: TGUID = '{FE4106E0-399A-11D0-A48C-00A0C90A8F39}';
```

C++

```
TGUID DIID_DShellWindowsEvents = '{FE4106E0-399A-11D0-A48C-00A0C90A8F39}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DShellWindowsEvents.

8.4.17 SHDocVw.DIID_DWebBrowserEvents Constant

Pascal

```
DIID_DWebBrowserEvents: TGUID = '{EAB22AC2-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID DIID_DWebBrowserEvents = '{EAB22AC2-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DWebBrowserEvents.

8.4.18 SHDocVw.DIID_DWebBrowserEvents2 Constant

Pascal

```
DIID_DWebBrowserEvents2: TGUID = '{34A715A0-6587-11D0-924A-0020AFC7AC4D}';
```

C++

```
TGUID DIID_DWebBrowserEvents2 = '{34A715A0-6587-11D0-924A-0020AFC7AC4D}';
```

File

SHDocVw

Description

This is constant SHDocVw.DIID_DWebBrowserEvents2.

8.4.19 SHDocVw.dtlOcxPage Constant

Pascal

```
dtlOcxPage = 'ActiveX';
```

C++

```
dtlOcxPage = 'ActiveX';
```

File

SHDocVw

Description

This is constant SHDocVw.dtlOcxPage.

8.4.20 SHDocVw.dtlServerPage Constant

Pascal

```
dtlServerPage = 'Servers';
```

C++

```
dtlServerPage = 'Servers';
```

File

SHDocVw

Description

This is constant SHDocVw.dtlServerPage.

8.4.21 SHDocVw.HTMLID_FIND Constant

Pascal

```
HTMLID_FIND = 1;
```

C++

```
HTMLID_FIND = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_FIND.

8.4.22 SHDocVw.HTMLID_OPTIONS Constant

Pascal

```
HTMLID_OPTIONS = 3;
```

C++

```
HTMLID_OPTIONS = 3;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_OPTIONS.

8.4.23 SHDocVw.HTMLID_VIEWSOURCE Constant

Pascal

```
HTMLID_VIEWSOURCE = 2;
```

C++

```
HTMLID_VIEWSOURCE = 2;
```

File

SHDocVw

Description

This is constant SHDocVw.HTMLID_VIEWSOURCE.

8.4.24 SHDocVw.IID_IScriptErrorList Constant

Pascal

```
IID_IScriptErrorList: TGUID = '{F3470F24-15FD-11D2-BB2E-00805FF7EFCA}';
```

C++

```
TGUID IID_IScriptErrorList = '{F3470F24-15FD-11D2-BB2E-00805FF7EFCA}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IScriptErrorList.

8.4.25 SHDocVw.IID_ISearch Constant

Pascal

```
IID_ISearch: TGUID = '{BA9239A4-3DD5-11D2-BF8B-00C04FB93661}';
```

C++

```
TGUID IID_ISearch = '{BA9239A4-3DD5-11D2-BF8B-00C04FB93661}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearch.

8.4.26 SHDocVw.IID_ISearchAssistantOC Constant

Pascal

```
IID_ISearchAssistantOC: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA1}';
```

C++

```
TGUID IID_ISearchAssistantOC = '{72423E8F-8011-11D2-BE79-00A0C9A83DA1}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC.

8.4.27 SHDocVw.IID_ISearchAssistantOC2 Constant

Pascal

```
IID_ISearchAssistantOC2: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA2}';
```

C++

```
TGUID IID_ISearchAssistantOC2 = '{72423E8F-8011-11D2-BE79-00A0C9A83DA2}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC2.

8.4.28 SHDocVw.IID_ISearchAssistantOC3 Constant

Pascal

```
IID_ISearchAssistantOC3: TGUID = '{72423E8F-8011-11D2-BE79-00A0C9A83DA3}';
```

C++

```
TGUID IID_ISearchAssistantOC3 = '{72423E8F-8011-11D2-BE79-00A0C9A83DA3}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearchAssistantOC3.

8.4.29 SHDocVw.IID_ISearches Constant

Pascal

```
IID_ISearches: TGUID = '{47C922A2-3DD5-11D2-BF8B-00C04FB93661}';
```

C++

```
TGUID IID_ISearches = '{47C922A2-3DD5-11D2-BF8B-00C04FB93661}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_ISearches.

8.4.30 SHDocVw.IID_IShellFavoritesNameSpace Constant

Pascal

```
IID_IShellFavoritesNameSpace: TGUID = '{55136804-B2DE-11D1-B9F2-00A0C98BC547}';
```

C++

```
TGUID IID_IShellFavoritesNameSpace = '{55136804-B2DE-11D1-B9F2-00A0C98BC547}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellFavoritesNameSpace.

8.4.31 SHDocVw.IID_IShellNameSpace Constant

Pascal

```
IID_IShellNameSpace: TGUID = '{E572D3C9-37BE-4AE2-825D-D521763E3108}';
```

C++

```
TGUID IID_IShellNameSpace = '{E572D3C9-37BE-4AE2-825D-D521763E3108}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellNameSpace.

8.4.32 SHDocVw.IID_IShellUIHelper Constant

Pascal

```
IID_IShellUIHelper: TGUID = '{729FE2F8-1EA8-11D1-8F85-00C04FC2FBE1}';
```

C++

```
TGUID IID_IShellUIHelper = '{729FE2F8-1EA8-11D1-8F85-00C04FC2FBE1}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellUIHelper.

8.4.33 SHDocVw.IID_IShellWindows Constant

Pascal

```
IID_IShellWindows: TGUID = '{85CB6900-4D95-11CF-960C-0080C7F4EE85}';
```

C++

```
TGUID IID_IShellWindows = '{85CB6900-4D95-11CF-960C-0080C7F4EE85}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IShellWindows.

8.4.34 SHDocVw.IID_IWebBrowser Constant

Pascal

```
IID_IWebBrowser: TGUID = '{EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID IID_IWebBrowser = '{EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowser.

8.4.35 SHDocVw.IID_IWebBrowser2 Constant

Pascal

```
IID_IWebBrowser2: TGUID = '{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}';
```

C++

```
TGUID IID_IWebBrowser2 = '{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowser2.

8.4.36 SHDocVw.IID_IWebBrowserApp Constant

Pascal

```
IID_IWebBrowserApp: TGUID = '{0002DF05-0000-0000-C000-000000000046}';
```

C++

```
TGUID IID_IWebBrowserApp = '{0002DF05-0000-0000-C000-000000000046}';
```

File

SHDocVw

Description

This is constant SHDocVw.IID_IWebBrowserApp.

8.4.37 SHDocVw.LIBID_SHDocVw Constant

Pascal

```
LIBID_SHDocVw: TGUID = '{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}';
```

C++

```
TGUID LIBID_SHDocVw = '{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}';
```

File

SHDocVw

Description

This is constant SHDocVw.LIBID_SHDocVw.

8.4.38 SHDocVw.navAllowAutosearch Constant

Pascal

```
navAllowAutosearch = $00000010;
```

C++

```
navAllowAutosearch = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.navAllowAutosearch.

8.4.39 SHDocVw.navBrowserBar Constant

Pascal

```
navBrowserBar = $00000020;
```

C++

```
navBrowserBar = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.navBrowserBar.

8.4.40 SHDocVw.navNoHistory Constant

Pascal

```
navNoHistory = $00000002;
```

C++

```
navNoHistory = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoHistory.

8.4.41 SHDocVw.navNoReadFromCache Constant

Pascal

```
navNoReadFromCache = $00000004;
```

C++

```
navNoReadFromCache = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoReadFromCache.

8.4.42 SHDocVw.navNoWriteToCache Constant

Pascal

```
navNoWriteToCache = $00000008;
```

C++

```
navNoWriteToCache = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.navNoWriteToCache.

8.4.43 SHDocVw.navOpenInNewWindow Constant

Pascal

```
navOpenInNewWindow = $00000001;
```

C++

```
navOpenInNewWindow = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.navOpenInNewWindow.

8.4.44 SHDocVw.OLECMDEXECOPT_DODEFAULT Constant

Pascal

```
OLECMDEXECOPT_DODEFAULT = $00000000;
```

C++

```
OLECMDEXECOPT_DODEFAULT = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_DODEFAULT.

8.4.45 SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER Constant

Pascal

```
OLECMDEXECOPT_DONTPROMPTUSER = $00000002;
```

C++

```
OLECMDEXECOPT_DONTPROMPTUSER = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER.

8.4.46 SHDocVw.OLECMDEXECOPT_PROMPTUSER Constant

Pascal

```
OLECMDEXECOPT_PROMPTUSER = $00000001;
```

C++

```
OLECMDEXECOPT_PROMPTUSER = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_PROMPTUSER.

8.4.47 SHDocVw.OLECMDEXECOPT_SHOWHELP Constant

Pascal

```
OLECMDEXECOPT_SHOWHELP = $00000003;
```

C++

```
OLECMDEXECOPT_SHOWHELP = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDEXECOPT_SHOWHELP.

8.4.48 SHDocVw.OLECMDF_DEFHIDEONCTXTMENU Constant

Pascal

```
OLECMDF_DEFHIDEONCTXTMENU = $00000020;
```

C++

```
OLECMDF_DEFHIDEONCTXTMENU = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_DEFHIDEONCTXTMENU.

8.4.49 SHDocVw.OLECMDF_ENABLED Constant

Pascal

```
OLECMDF_ENABLED = $00000002;
```

C++

```
OLECMDF_ENABLED = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_ENABLED.

8.4.50 SHDocVw.OLECMDF_INVISIBLE Constant

Pascal

```
OLECMDF_INVISIBLE = $00000010;
```

C++

```
OLECMDF_INVISIBLE = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_INVISIBLE.

8.4.51 SHDocVw.OLECMDF_LATCHED Constant

Pascal

```
OLECMDF_LATCHED = $00000004;
```

C++

```
OLECMDF_LATCHED = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_LATCHED.

8.4.52 SHDocVw.OLECMDF_NINCHED Constant

Pascal

```
OLECMDF_NINCHED = $00000008;
```

C++

```
OLECMDF_NINCHED = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_NINCHED.

8.4.53 SHDocVw.OLECMDF_SUPPORTED Constant

Pascal

```
OLECMDF_SUPPORTED = $00000001;
```

C++

```
OLECMDF_SUPPORTED = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDF_SUPPORTED.

8.4.54 SHDocVw.OLECMDID_ALLOWUILESSSAVEAS Constant

Pascal

```
OLECMDID_ALLOWUILESSSAVEAS = $0000002E;
```

C++

```
OLECMDID_ALLOWUILESSSAVEAS = 0x0000002E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ALLOWUILESSSAVEAS.

8.4.55 SHDocVw.OLECMDID_CLEARSELECTION Constant

Pascal

```
OLECMDID_CLEARSELECTION = $00000012;
```

C++

```
OLECMDID_CLEARSELECTION = 0x00000012;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CLEARSELECTION.

8.4.56 SHDocVw.OLECMDID_CLOSE Constant

Pascal

```
OLECMDID_CLOSE = $0000002D;
```

C++

```
OLECMDID_CLOSE = 0x0000002D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CLOSE.

8.4.57 SHDocVw.OLECMDID_COPY Constant

Pascal

```
OLECMDID_COPY = $0000000C;
```

C++

```
OLECMDID_COPY = 0x0000000C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_COPY.

8.4.58 SHDocVw.OLECMDID_CUT Constant

Pascal

```
OLECMDID_CUT = $0000000B;
```

C++

```
OLECMDID_CUT = 0x0000000B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_CUT.

8.4.59 SHDocVw.OLECMDID_DELETE Constant

Pascal

```
OLECMDID_DELETE = $00000021;
```

C++

```
OLECMDID_DELETE = 0x00000021;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_DELETE.

8.4.60 SHDocVw.OLECMDID_DONTDOWNLOADCSS Constant

Pascal

```
OLECMDID_DONTDOWNLOADCSS = $0000002F;
```

C++

```
OLECMDID_DONTDOWNLOADCSS = 0x0000002F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_DONTDOWNLOADCSS.

8.4.61 SHDocVw.OLECMDID_ENABLE_INTERACTION Constant

Pascal

```
OLECMDID_ENABLE_INTERACTION = $00000024;
```

C++

```
OLECMDID_ENABLE_INTERACTION = 0x00000024;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ENABLE_INTERACTION.

8.4.62 SHDocVw.OLECMDID_FIND Constant

Pascal

```
OLECMDID_FIND = $00000020;
```

C++

```
OLECMDID_FIND = 0x00000020;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_FIND.

8.4.63 SHDocVw.OLECMDID_GETPRINTTEMPLATE Constant

Pascal

```
OLECMDID_GETPRINTTEMPLATE = $00000034;
```

C++

```
OLECMDID_GETPRINTTEMPLATE = 0x00000034;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_GETPRINTTEMPLATE.

8.4.64 SHDocVw.OLECMDID_GETZOOMRANGE Constant

Pascal

```
OLECMDID_GETZOOMRANGE = $00000014;
```

C++

```
OLECMDID_GETZOOMRANGE = 0x00000014;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_GETZOOMRANGE.

8.4.65 SHDocVw.OLECMDID_HIDETOOLBARS Constant

Pascal

```
OLECMDID_HIDETOOLBARS = $00000018;
```

C++

```
OLECMDID_HIDETOOLBARS = 0x00000018;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HIDETOOLBARS.

8.4.66 SHDocVw.OLECMDID_HTTPEQUIV Constant

Pascal

```
OLECMDID_HTTPEQUIV = $00000022;
```

C++

```
OLECMDID_HTTPEQUIV = 0x00000022;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HTTPEQUIV.

8.4.67 SHDocVw.OLECMDID_HTTPEQUIV_DONE Constant

Pascal

```
OLECMDID_HTTPEQUIV_DONE = $00000023;
```

C++

```
OLECMDID_HTTPEQUIV_DONE = 0x00000023;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_HTTPEQUIV_DONE.

8.4.68 SHDocVw.OLECMDID_NEW Constant

Pascal

```
OLECMDID_NEW = $00000002;
```

C++

```
OLECMDID_NEW = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_NEW.

8.4.69 SHDocVw.OLECMDID_ONTOOLBARACTIVATED Constant

Pascal

```
OLECMDID_ONTOOLBARACTIVATED = $0000001F;
```

C++

```
OLECMDID_ONTOOLBARACTIVATED = 0x0000001F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ONTOOLBARACTIVATED.

8.4.70 SHDocVw.OLECMDID_ONUNLOAD Constant

Pascal

```
OLECMDID_ONUNLOAD = $00000025;
```

C++

```
OLECMDID_ONUNLOAD = 0x00000025;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ONUNLOAD.

8.4.71 SHDocVw.OLECMDID_OPEN Constant

Pascal

```
OLECMDID_OPEN = $00000001;
```

C++

```
OLECMDID_OPEN = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_OPEN.

8.4.72 SHDocVw.OLECMDID_PAGESETUP Constant

Pascal

```
OLECMDID_PAGESETUP = $00000008;
```

C++

```
OLECMDID_PAGESETUP = 0x00000008;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PAGESETUP.

8.4.73 SHDocVw.OLECMDID_PASTE Constant

Pascal

```
OLECMDID_PASTE = $0000000D;
```

C++

```
OLECMDID_PASTE = 0x0000000D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PASTE.

8.4.74 SHDocVw.OLECMDID_PASTESPECIAL Constant

Pascal

```
OLECMDID_PASTESPECIAL = $0000000E;
```

C++

```
OLECMDID_PASTESPECIAL = 0x0000000E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PASTESPECIAL.

8.4.75 SHDocVw.OLECMDID_PREREFRESH Constant

Pascal

```
OLECMDID_PREREFRESH = $00000027;
```

C++

```
OLECMDID_PREREFRESH = 0x00000027;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PREREFRESH.

8.4.76 SHDocVw.OLECMDID_PRINT Constant

Pascal

```
OLECMDID_PRINT = $00000006;
```

C++

```
OLECMDID_PRINT = 0x00000006;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINT.

8.4.77 SHDocVw.OLECMDID_PRINT2 Constant

Pascal

```
OLECMDID_PRINT2 = $00000031;
```

C++

```
OLECMDID_PRINT2 = 0x00000031;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINT2.

8.4.78 SHDocVw.OLECMDID_PRINTPREVIEW Constant

Pascal

```
OLECMDID_PRINTPREVIEW = $00000007;
```

C++

```
OLECMDID_PRINTPREVIEW = 0x00000007;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINTPREVIEW.

8.4.79 SHDocVw.OLECMDID_PRINTPREVIEW2 Constant

Pascal

```
OLECMDID_PRINTPREVIEW2 = $00000032;
```

C++

```
OLECMDID_PRINTPREVIEW2 = 0x00000032;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PRINTPREVIEW2.

8.4.80 SHDocVw.OLECMDID_PROPERTIES Constant

Pascal

```
OLECMDID_PROPERTIES = $0000000A;
```

C++

```
OLECMDID_PROPERTIES = 0x0000000A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PROPERTIES.

8.4.81 SHDocVw.OLECMDID_PROPERTYBAG2 Constant

Pascal

```
OLECMDID_PROPERTYBAG2 = $00000026;
```

C++

```
OLECMDID_PROPERTYBAG2 = 0x00000026;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_PROPERTYBAG2.

8.4.82 SHDocVw.OLECMDID_REDO Constant

Pascal

```
OLECMDID_REDO = $00000010;
```

C++

```
OLECMDID_REDO = 0x00000010;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_REDO.

8.4.83 SHDocVw.OLECMDID_REFRESH Constant

Pascal

```
OLECMDID_REFRESH = $00000016;
```

C++

```
OLECMDID_REFRESH = 0x00000016;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_REFRESH.

8.4.84 SHDocVw.OLECMDID_SAVE Constant

Pascal

```
OLECMDID_SAVE = $00000003;
```

C++

```
OLECMDID_SAVE = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVE.

8.4.85 SHDocVw.OLECMDID_SAVEAS Constant

Pascal

```
OLECMDID_SAVEAS = $00000004;
```

C++

```
OLECMDID_SAVEAS = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVEAS.

8.4.86 SHDocVw.OLECMDID_SAVECOPYAS Constant

Pascal

```
OLECMDID_SAVECOPYAS = $00000005;
```

C++

```
OLECMDID_SAVECOPYAS = 0x00000005;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SAVECOPYAS.

8.4.87 SHDocVw.OLECMDID_SELECTALL Constant

Pascal

```
OLECMDID_SELECTALL = $00000011;
```

C++

```
OLECMDID_SELECTALL = 0x00000011;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SELECTALL.

8.4.88 SHDocVw.OLECMDID_SETDOWNLOADSTATE Constant

Pascal

```
OLECMDID_SETDOWNLOADSTATE = $0000001D;
```

C++

```
OLECMDID_SETDOWNLOADSTATE = 0x0000001D;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETDOWNLOADSTATE.

8.4.89 SHDocVw.OLECMDID_SETPRINTTEMPLATE Constant

Pascal

```
OLECMDID_SETPRINTTEMPLATE = $00000033;
```

C++

```
OLECMDID_SETPRINTTEMPLATE = 0x00000033;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPRINTTEMPLATE.

8.4.90 SHDocVw.OLECMDID_SETPROGRESSMAX Constant

Pascal

```
OLECMDID_SETPROGRESSMAX = $00000019;
```

C++

```
OLECMDID_SETPROGRESSMAX = 0x00000019;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSMAX.

8.4.91 SHDocVw.OLECMDID_SETPROGRESSPOS Constant

Pascal

```
OLECMDID_SETPROGRESSPOS = $0000001A;
```

C++

```
OLECMDID_SETPROGRESSPOS = 0x0000001A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSPOS.

8.4.92 SHDocVw.OLECMDID_SETPROGRESSTEXT Constant

Pascal

```
OLECMDID_SETPROGRESSTEXT = $0000001B;
```

C++

```
OLECMDID_SETPROGRESSTEXT = 0x0000001B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETPROGRESSTEXT.

8.4.93 SHDocVw.OLECMDID_SETTITLE Constant

Pascal

```
OLECMDID_SETTITLE = $0000001C;
```

C++

```
OLECMDID_SETTITLE = 0x0000001C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SETTITLE.

8.4.94 SHDocVw.OLECMDID_SHOWFIND Constant

Pascal

```
OLECMDID_SHOWFIND = $0000002A;
```

C++

```
OLECMDID_SHOWFIND = 0x0000002A;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWFIND.

8.4.95 SHDocVw.OLECMDID_SHOWMESSAGE Constant

Pascal

```
OLECMDID_SHOWMESSAGE = $00000029;
```

C++

```
OLECMDID_SHOWMESSAGE = 0x00000029;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWMESSAGE.

8.4.96 SHDocVw.OLECMDID_SHOWPAGESETUP Constant

Pascal

```
OLECMDID_SHOWPAGESETUP = $0000002B;
```

C++

```
OLECMDID_SHOWPAGESETUP = 0x0000002B;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWPAGESETUP.

8.4.97 SHDocVw.OLECMDID_SHOWPRINT Constant

Pascal

```
OLECMDID_SHOWPRINT = $0000002C;
```

C++

```
OLECMDID_SHOWPRINT = 0x0000002C;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWPRINT.

8.4.98 SHDocVw.OLECMDID_SHOWSCRIPTERROR Constant

Pascal

```
OLECMDID_SHOWSCRIPTERROR = $00000028;
```

C++

```
OLECMDID_SHOWSCRIPTERROR = 0x00000028;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SHOWSCRIPTERROR.

8.4.99 SHDocVw.OLECMDID_SPELL Constant

Pascal

```
OLECMDID_SPELL = $00000009;
```

C++

```
OLECMDID_SPELL = 0x00000009;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_SPELL.

8.4.100 SHDocVw.OLECMDID_STOP Constant

Pascal

```
OLECMDID_STOP = $00000017;
```

C++

```
OLECMDID_STOP = 0x00000017;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_STOP.

8.4.101 SHDocVw.OLECMDID_STOPDOWNLOAD Constant

Pascal

```
OLECMDID_STOPDOWNLOAD = $0000001E;
```

C++

```
OLECMDID_STOPDOWNLOAD = 0x0000001E;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_STOPDOWNLOAD.

8.4.102 SHDocVw.OLECMDID_UNDO Constant

Pascal

```
OLECMDID_UNDO = $0000000F;
```

C++

```
OLECMDID_UNDO = 0x0000000F;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UNDO.

8.4.103 SHDocVw.OLECMDID_UPDATECOMMANDS Constant

Pascal

```
OLECMDID_UPDATECOMMANDS = $00000015;
```

C++

```
OLECMDID_UPDATECOMMANDS = 0x00000015;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UPDATECOMMANDS.

8.4.104 SHDocVw.OLECMDID_UPDATEPAGESTATUS Constant

Pascal

```
OLECMDID_UPDATEPAGESTATUS = $00000030;
```

C++

```
OLECMDID_UPDATEPAGESTATUS = 0x00000030;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_UPDATEPAGESTATUS.

8.4.105 SHDocVw.OLECMDID_ZOOM Constant

Pascal

```
OLECMDID_ZOOM = $00000013;
```

C++

```
OLECMDID_ZOOM = 0x00000013;
```

File

SHDocVw

Description

This is constant SHDocVw.OLECMDID_ZOOM.

8.4.106 SHDocVw.READYSTATE_COMPLETE Constant

Pascal

```
READYSTATE_COMPLETE = $00000004;
```

C++

```
READYSTATE_COMPLETE = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_COMPLETE.

8.4.107 SHDocVw.READYSTATE_INTERACTIVE Constant

Pascal

```
READYSTATE_INTERACTIVE = $00000003;
```

C++

```
READYSTATE_INTERACTIVE = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_INTERACTIVE.

8.4.108 SHDocVw.READYSTATE_LOADED Constant

Pascal

```
READYSTATE_LOADED = $00000002;
```

C++

```
READYSTATE_LOADED = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_LOADED.

8.4.109 SHDocVw.READYSTATE_LOADING Constant

Pascal

```
READYSTATE_LOADING = $00000001;
```

C++

```
READYSTATE_LOADING = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_LOADING.

8.4.110 SHDocVw.READYSTATE_UNINITIALIZED Constant

Pascal

```
READYSTATE_UNINITIALIZED = $00000000;
```

C++

```
READYSTATE_UNINITIALIZED = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.READYSTATE_UNINITIALIZED.

8.4.111 SHDocVw.REFRESH_COMPLETELY Constant

Pascal

```
REFRESH_COMPLETELY = 3;
```

C++

```
REFRESH_COMPLETELY = 3;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_COMPLETELY.

8.4.112 SHDocVw.REFRESH_CONTINUE Constant

Pascal

```
REFRESH_CONTINUE = 2;
```

C++

```
REFRESH_CONTINUE = 2;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_CONTINUE.

8.4.113 SHDocVw.REFRESH_IFEXPIRED Constant

Pascal

```
REFRESH_IFEXPIRED = 1;
```

C++

```
REFRESH_IFEXPIRED = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_IFEXPIRED.

8.4.114 SHDocVw.REFRESH_NORMAL Constant

Pascal

```
REFRESH_NORMAL = 0;
```

C++

```
REFRESH_NORMAL = 0;
```

File

SHDocVw

Description

This is constant SHDocVw.REFRESH_NORMAL.

8.4.115 SHDocVw.secureLockIconMixed Constant

Pascal

```
secureLockIconMixed = $00000001;
```

C++

```
secureLockIconMixed = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconMixed.

8.4.116 SHDocVw.secureLockIconSecure128Bit Constant

Pascal

```
secureLockIconSecure128Bit = $00000006;
```

C++

```
secureLockIconSecure128Bit = 0x00000006;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure128Bit.

8.4.117 SHDocVw.secureLockIconSecure40Bit Constant

Pascal

```
secureLockIconSecure40Bit = $00000003;
```

C++

```
secureLockIconSecure40Bit = 0x00000003;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure40Bit.

8.4.118 SHDocVw.secureLockIconSecure56Bit Constant

Pascal

```
secureLockIconSecure56Bit = $00000004;
```

C++

```
secureLockIconSecure56Bit = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecure56Bit.

8.4.119 SHDocVw.secureLockIconSecureFortezza Constant

Pascal

```
secureLockIconSecureFortezza = $00000005;
```

C++

```
secureLockIconSecureFortezza = 0x00000005;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecureFortezza.

8.4.120 SHDocVw.secureLockIconSecureUnknownBits Constant

Pascal

```
secureLockIconSecureUnknownBits = $00000002;
```

C++

```
secureLockIconSecureUnknownBits = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconSecureUnknownBits.

8.4.121 SHDocVw.secureLockIconUnsecure Constant

Pascal

```
secureLockIconUnsecure = $00000000;
```

C++

```
secureLockIconUnsecure = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.secureLockIconUnsecure.

8.4.122 SHDocVw.SHDocVwMajorVersion Constant

Pascal

```
SHDocVwMajorVersion = 1;
```

C++

```
SHDocVwMajorVersion = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.SHDocVwMajorVersion.

8.4.123 SHDocVw.SHDocVwMinorVersion Constant

Pascal

```
SHDocVwMinorVersion = 1;
```

C++

```
SHDocVwMinorVersion = 1;
```

File

SHDocVw

Description

This is constant SHDocVw.SHDocVwMinorVersion.

8.4.124 SHDocVw.SWC_3RDPARTY Constant

Pascal

```
SWC_3RDPARTY = $00000002;
```

C++

```
SWC_3RDPARTY = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_3RDPARTY.

8.4.125 SHDocVw.SWC_BROWSER Constant

Pascal

```
SWC_BROWSER = $00000001;
```

C++

```
SWC_BROWSER = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_BROWSER.

8.4.126 SHDocVw.SWC_CALLBACK Constant

Pascal

```
SWC_CALLBACK = $00000004;
```

C++

```
SWC_CALLBACK = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_CALLBACK.

8.4.127 SHDocVw.SWC_EXPLORER Constant

Pascal

```
SWC_EXPLORER = $00000000;
```

C++

```
SWC_EXPLORER = 0x00000000;
```

File

SHDocVw

Description

This is constant SHDocVw.SWC_EXPLORER.

8.4.128 SHDocVw.SWFO_COOKIEPASSED Constant

Pascal

```
SWFO_COOKIEPASSED = $00000004;
```

C++

```
SWFO_COOKIEPASSED = 0x00000004;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_COOKIEPASSED.

8.4.129 SHDocVw.SWFO_INCLUDEPENDING Constant

Pascal

```
SWFO_INCLUDEPENDING = $00000002;
```

C++

```
SWFO_INCLUDEPENDING = 0x00000002;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_INCLUDEPENDING.

8.4.130 SHDocVw.SWFO_NEEDDISPATCH Constant

Pascal

```
SWFO_NEEDDISPATCH = $00000001;
```

C++

```
SWFO_NEEDDISPATCH = 0x00000001;
```

File

SHDocVw

Description

This is constant SHDocVw.SWFO_NEEDDISPATCH.

9 ShellAnimations Namespace

This is namespace ShellAnimations.

Topics

Name	Description
Classes (see page 886)	The following table lists classes in this documentation.

Classes

	Name	Description
	TShellResources (see page 886)	TShellResources provides forwards compability for programs that use system animations in Windows Vista.

9.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TShellResources (see page 886)	TShellResources provides forwards compability for programs that use system animations in Windows Vista.

9.1.1 TShellResources Class

TShellResources provides forwards compability for programs that use system animations in Windows Vista.

Class Hierarchy

Pascal

```
TShellResources = class(TComponent);
```

C++

```
class TShellResources : public TComponent;
```

File

ShellAnimations

Description

Windows Vista shell animations are no longer accessible outside of the system; programs must now store local copies of these animations instead of linking to them as was done in the past.

Placing TShellResources onto a form copies the following system animation resources into the program:

- CopyFile.res
- CopyFiles.res
- DeleteFile.res
- EmptyRecycle.res
- FindComputer.res
- FindFiles.res
- FindFolder.res
- RecycleFile.res

TShellResources should only be used in programs that are run on Vista and use animations from the common library, otherwise this function adds unneeded overhead.

See Also

TAnimate

10 SimpleDS Namespace

This is namespace SimpleDS.

Topics

Name	Description
Classes (see page 888)	The following table lists classes in this documentation.

Classes

	Name	Description
	TInternalSQLDataSet (see page 888)	This is class SimpleDS.TInternalSQLDataSet.
	TSimpleDataSet (see page 896)	TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

10.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TInternalSQLDataSet (see page 888)	This is class SimpleDS.TInternalSQLDataSet.
	TSimpleDataSet (see page 896)	TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

10.1.1 TInternalSQLDataSet Class

Class Hierarchy

Pascal

```
TInternalSQLDataSet = class(TCustomSQLDataSet);
```

C++

```
class TInternalSQLDataSet : public TCustomSQLDataSet;
```

File

SimpleDS

Description

This is class SimpleDS.TInternalSQLDataSet.

10.1.1.1 TInternalSQLDataSet Members

The following tables list the members exposed by TInternalSQLDataSet.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecSQL (see page 1529)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.

	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
	OpenCursor (see page 1539)	This is OpenCursor, a member of class TCustomSQLDataSet.
	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.
	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.

	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DbxCommandType (see page 1557)	This is DbxCommandType, a member of class TCustomSQLDataSet.
	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.
	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	RowsAffected (see page 1564)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.

Public Methods

	Name	Description
	Create (see page 1524)	This is the overview for the Create constructor overload.
	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	Destroy (see page 1525)	Destroys the TCustomSQLDataSet (see page 1520) instance.
	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
	PrepareStatement (see page 1540)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	SQLConnection (see page 1566)	Specifies the SQL connection component that connects the dataset to a database server.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.

	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Published Properties

	Name	Description
	Active (see page 1552)	Specifies whether or not a dataset is open.
	CommandText (see page 891)	Specifies the command the dataset executes.
	CommandType (see page 892)	Indicates the meaning of the CommandText property.
	DataSource (see page 893)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 894)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 894)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 895)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 895)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1565)	Specifies the name of the database schema.
	SortFieldNames (see page 896)	Indicates the sort order when CommandType is ctTable.

10.1.1.2 TInternalSQLDataSet Properties

The properties of the TInternalSQLDataSet class are listed here.

Published Properties

	Name	Description
	CommandText (see page 891)	Specifies the command the dataset executes.
	CommandType (see page 892)	Indicates the meaning of the CommandText property.
	DataSource (see page 893)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 894)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 894)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 895)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 895)	Represents the parameters for a query or stored procedure.
	SortFieldNames (see page 896)	Indicates the sort order when CommandType is ctTable.

10.1.1.2.1 TInternalSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: UnicodeString;
```

C++

```
__property UnicodeString CommandText;
```

Description

The value of CommandText depends on the CommandType property:

When CommandType is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL method is called.

When CommandType is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL method is called, depending on whether the stored procedure returns a set of records.

When CommandType is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params property to supply parameter values.

See Also

CommandType (see page 1555)

Params (see page 1562)

Open

ExecSQL (see page 1636)

SortFieldNames (see page 1566)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

10.1.1.2.2 TInternalSQLDataSet.CommandType Property

Indicates the meaning of the CommandText property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the type of command that is contained in the CommandText property. TCustomSQLDataSet descendants set this value to the way in which they specify a query or stored procedure.

The following table lists the possible values:

CommandType	Corresponding CommandText
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

See Also

CommandText (see page 1554)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

10.1.1.2.3 TInternalSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1554)

SQL (see page 1663)

Params (see page 1562)

MasterSource (see page 1692)

10.1.1.2.4 TInternalSQLDataSet.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataSet

TDataSetProvider

Fetching the Data using TSQLDataSet

10.1.1.2.5 TInternalSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1530)

Params (see page 1621)

SQLConnection (see page 1566)

10.1.1.2.6 TInternalSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 1562)

CommandText (see page 1554)

SQL (see page 1663)

10.1.1.2.7 TInternalSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by

the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1561)

SQL (see page 1663)

StoredProcName (see page 1680)

CommandText (see page 1554)

TParam

Value

DataType

DataSource (see page 1557)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

10.1.1.2.8 TInternalSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType is ctTable.

Pascal

```
property SortFieldNames: UnicodeString;
```

C++

```
__property UnicodeString SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 1555)

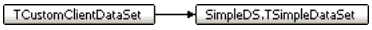
Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

10.1.2 TSimpleDataSet Class

TSimpleDataSet uses dbExpress to fetch data and cache it internally in memory.

Class Hierarchy



Pascal

```
TSimpleDataSet = class(TCustomClientDataSet);
```

C++

```
class TSimpleDataSet : public TCustomClientDataSet;
```

File

SimpleDS

Description

TSimpleDataSet is a client dataset that uses an internal TSQLDataSet (see page 1631) and TDataSetProvider for fetching data and applying updates. It combines the fast access and easy deployment of a unidirectional dataset with a client dataset's ability to edit and navigate data.

TSimpleDataSet uses dbExpress for fast access to database information. As a client dataset, TSimpleDataSet caches that information in memory and saves any updates the application makes. With its internal dataset provider, TSimpleDataSet can apply those updates back to the database server. Thus, TSimpleDataSet works like TClientDataSet when it is connected to a local TSQLDataSet (see page 1631) via a provider, except that the source dataset and provider are internal.

TSimpleDataSet surfaces some of the properties and events of TSQLDataSet (see page 1631) so that you can specify the database server from which to fetch data, indicate what data to fetch from that server, influence what information is included in data packets, and provide input on the updating process.

In addition to using the source TSQLDataSet (see page 1631), TSimpleDataSet can read from and write to a dedicated file on disk. The internal provider and source dataset are not needed for working with file-based data, making TClientDataSet a more suitable choice if you are writing a purely file-based application. However, the ability to work with a file on disk makes TSimpleDataSet well-suited to briefcase model applications.

See Also

TDataSetProvider

Choosing the Type of Dataset for Caching Updates

Types of Unidirectional Datasets

10.1.2.1 TSimpleDataSet Members

The following tables list the members exposed by TSimpleDataSet.

Protected Methods

	Name	Description
	AllocConnection (see page 900)	This is AllocConnection, a member of class TSimpleDataSet.
	AllocDataSet (see page 900)	This is AllocDataSet, a member of class TSimpleDataSet.
	AllocProvider (see page 901)	This is AllocProvider, a member of class TSimpleDataSet.
	Loaded (see page 901)	Initializes the component after the form file has been read into memory.
	Notification (see page 901)	Forwards notification messages to all owned components.
	OpenCursor (see page 902)	This is OpenCursor, a member of class TSimpleDataSet.
	PSGetCommandText (see page 902)	Gets the SQL command to be executed by PSExecute.
	SetConnection (see page 903)	This is SetConnection, a member of class TSimpleDataSet.

Public Methods

	Name	Description
	Create (see page 899)	Creates and initializes an instance of TSimpleDataSet (see page 896).
	Destroy (see page 900)	Destroys the instance of TSimpleDataSet (see page 896).

Published Events

	Name	Description
	AfterApplyUpdates (see page 931)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 931)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 933)	Occurs after an application closes a dataset.
	AfterDelete (see page 935)	Occurs after an application deletes a record.
	AfterEdit (see page 937)	Occurs after an application starts editing a record.
	AfterExecute (see page 938)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 939)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 940)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 940)	Occurs after an application inserts a new record.
	AfterOpen (see page 941)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 942)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 943)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 943)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 944)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 944)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 945)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 946)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 946)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 947)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 947)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 948)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 949)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 953)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 954)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 955)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 956)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 957)	Occurs when an application inserts or appends a new dataset record.
	OnPostError (see page 957)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 958)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Published Properties

	Name	Description
	Active (see page 904)	Specifies whether the client dataset contains data.
	Aggregates (see page 906)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 907)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 907)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	Connection (see page 908)	Identifies the TSQLConnection (see page 1580) component that connects the dataset to a database server.
	Constraints (see page 908)	Specifies record-level constraints that must be met when editing the data.
R	DataSet (see page 909)	Provides access to the internal dataset that fetches data from the server.
	DisableStringTrim (see page 909)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 910)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 911)	Points to the list of field definitions for the dataset.
	FileName (see page 914)	Specifies a file that stores the data of the client dataset.
	Filter (see page 915)	Specifies the text of the current filter.
	Filtered (see page 917)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 917)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 918)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 919)	Lists the fields to use as an index.
	IndexName (see page 920)	Identifies an index for the client dataset.
	MasterFields (see page 921)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 921)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	ObjectView (see page 922)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 923)	Indicates the number or type of records in a single data packet.
	Params (see page 924)	Contains parameter values that are sent to the provider.
	ReadOnly (see page 929)	Specifies whether the client dataset is read-only for this application.
	StoreDefs (see page 929)	Indicates whether field and index definitions are saved with the client dataset.

10.1.2.2 TSimpleDataSet.Create Constructor

Creates and initializes an instance of TSimpleDataSet (see page 896).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSimpleDataSet(TComponent * AOwner);
```

Description

Call Create to create TSimpleDataSet (see page 896) at runtime. Datasets added to forms or data modules at design time are created automatically.

AOwner is the component that is responsible for freeing the TSimpleDataSet (see page 896) instance. If AOwner is nil (Delphi) or NULL (C++), the application must explicitly free the dataset.

Create creates the internal source dataset and provider that fetch data from the database server and apply updates back to the database server.

See Also

Owner

10.1.2.3 TSimpleDataSet.Destroy Destructor

Destroys the instance of TSimpleDataSet (see page 896).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset without assigning an Owner, however, it should call Free, which verifies that the dataset component is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

10.1.2.4 TSimpleDataSet Methods

The methods of the TSimpleDataSet class are listed here.

Protected Methods

	Name	Description
	AllocConnection (see page 900)	This is AllocConnection, a member of class TSimpleDataSet.
	AllocDataSet (see page 900)	This is AllocDataSet, a member of class TSimpleDataSet.
	AllocProvider (see page 901)	This is AllocProvider, a member of class TSimpleDataSet.
	Loaded (see page 901)	Initializes the component after the form file has been read into memory.
	Notification (see page 901)	Forwards notification messages to all owned components.
	OpenCursor (see page 902)	This is OpenCursor, a member of class TSimpleDataSet.
	PSGetCommandText (see page 902)	Gets the SQL command to be executed by PSExecute.
	SetConnection (see page 903)	This is SetConnection, a member of class TSimpleDataSet.

10.1.2.4.1 TSimpleDataSet.AllocConnection Method

Pascal

```
procedure AllocConnection; virtual;
```

C++

```
virtual __fastcall AllocConnection();
```

Description

This is AllocConnection, a member of class TSimpleDataSet.

10.1.2.4.2 TSimpleDataSet.AllocDataSet Method

Pascal

```
procedure AllocDataSet; virtual;
```

C++

```
virtual __fastcall AllocDataSet();
```

Description

This is AllocDataSet, a member of class TSimpleDataSet.

10.1.2.4.3 TSimpleDataSet.AllocProvider Method

Pascal

```
procedure AllocProvider; virtual;
```

C++

```
virtual __fastcall AllocProvider();
```

Description

This is AllocProvider, a member of class TSimpleDataSet.

10.1.2.4.4 TSimpleDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

10.1.2.4.5 TSimpleDataSet.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

10.1.2.4.6 TSimpleDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TSimpleDataSet.

10.1.2.4.7 TSimpleDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText

PSExecute

PSSetCommandText

GetRecords

Execute

Options

10.1.2.4.8 TSimpleDataSet.SetConnection Method

Pascal

```
procedure SetConnection(Value: TSQLConnection); virtual;
```

C++

```
virtual __fastcall SetConnection(TSQLConnection Value);
```

Description

This is SetConnection, a member of class TSimpleDataSet.

10.1.2.5 TSimpleDataSet Properties

The properties of the TSimpleDataSet class are listed here.

Published Properties

	Name	Description
	Active (see page 904)	Specifies whether the client dataset contains data.
	Aggregates (see page 906)	Lists all aggregates that apply to the client dataset.
	AggregatesActive (see page 907)	Specifies whether the client dataset calculates and maintains aggregate values.
	AutoCalcFields (see page 907)	Determines when the OnCalcFields event is triggered and when lookup field values are calculated.
	Connection (see page 908)	Identifies the TSQLConnection (see page 1580) component that connects the dataset to a database server.
	Constraints (see page 908)	Specifies record-level constraints that must be met when editing the data.
R	DataSet (see page 909)	Provides access to the internal dataset that fetches data from the server.
	DisableStringTrim (see page 909)	Specifies whether trailing spaces are trimmed from field values when records are posted.
	FetchOnDemand (see page 910)	Indicates whether to fetch data packets from the provider on an as-needed basis.
	FieldDefs (see page 911)	Points to the list of field definitions for the dataset.
	FileName (see page 914)	Specifies a file that stores the data of the client dataset.
	Filter (see page 915)	Specifies the text of the current filter.
	Filtered (see page 917)	Specifies whether or not filtering is active for a dataset.
	FilterOptions (see page 917)	Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.
	IndexDefs (see page 918)	Contains information about the indexes for a client dataset.
	IndexFieldNames (see page 919)	Lists the fields to use as an index.
	IndexName (see page 920)	Identifies an index for the client dataset.
	MasterFields (see page 921)	Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.
	MasterSource (see page 921)	Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.
	ObjectView (see page 922)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PacketRecords (see page 923)	Indicates the number or type of records in a single data packet.
	Params (see page 924)	Contains parameter values that are sent to the provider.

	ReadOnly (see page 929)	Specifies whether the client dataset is read-only for this application.
	StoreDefs (see page 929)	Indicates whether field and index definitions are saved with the client dataset.

10.1.2.5.1 TSimpleDataSet.Active Property

Specifies whether the client dataset contains data.

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

Use Active to determine whether the client dataset makes its data available. When Active is false, the client dataset is closed; It can't manipulate the data it represents. When Active is true, data can be read or edited using the client dataset.

Setting Active to true:

1Fills the client dataset with data. Depending on the other properties of the client dataset, this data comes from:

The file specified by the FileName property.

The provider specified by the ProviderName property or SetProvider method. (For some TCustomClientDataSet descendants, this is an internal provider component)

The data that was active when the client dataset was deactivated (only if the application has run continuously since the dataset was deactivated).

2Triggers a BeforeOpen event.

3Sets the State property to dsBrowse.

4Opens a cursor into the dataset.

5Triggers an After Open event.

If the client dataset has never been active previously, setting Active to true defines the structure (metadata) of the client dataset. This structure is retrieved with the data packet from the provider or stored on disk, or, if there is no data packet, it is built using the current value of the FieldDefs property or from the persistent field components listed by the Fields property.

If an error occurs during the dataset open, the dataset state is set to dsInactive, and the cursor is closed.

Setting Active to false:

1Triggers a BeforeClose event.

2Sets the State property to dsInactive.

3Closes the cursor, saving the current data to disk if the FileName property is set, and saving the current data packet to a cache so that it can be restored when the client dataset is reopened.

4Triggers an AfterClose event.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
```

```
  information it needs for incremental data fetching. Before
  fetching the next data packet, the client dataset packages
  up the key value of the last record so that the provider
  knows where to begin the next data packet. It also sends
  some application-specific information, which is stored in
  Memo1.  This method is used to pass the sql statement and the
  value of the first field for the last record to the server so
  that the server can return the correct records. The complete
  project using this code example is in
  Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
  }
  procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
  var
    LastValue: OleVariant;
    CDSClone: TClientDataSet;
  begin
    if ClientDataSet1.Active then
    begin
      CDSClone := TClientDataSet.Create(Form1);
      try
        CDSClone.CloneCursor(ClientDataSet1, True);
        { turn off FetchOnDemand so that the clone only fetches
          the last LOCAL record }
        CDSClone.FetchOnDemand := False;
        CDSClone.Last;
        LastValue := CDSClone.Fields[0].AsString;
        CDSClone.Close;
      finally
        CDSClone.Free;
      end;
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
  {
  The following code fragment illustrates how DataSets and
  DataSetCount can be used to ensure that an action is taken
  for every open dataset.  Assign CheckButtonActive as the
  AfterClose and AfterOpen event handlers for TClientDataSet1.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I: Integer;
  begin
    if not RemoteServer1.Connected then
      RemoteServer1.Connected := True;
    ClientDataSet1.Close;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].EnableControls;
    end;
    ClientDataSet1.Open;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    I: Integer;
  begin
    ClientDataSet1.Close;
```

```
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

FileName

Open

Close

State

AfterOpen

BeforeOpen

FieldDefs

Fields

ProviderName

10.1.2.5.2 TSimpleDataSet.Aggregates Property

Lists all aggregates that apply to the client dataset.

Pascal

```
property Aggregates;
```

C++

```
__property Aggregates;
```

Description

Use Aggregates to define aggregates that summarize the data in the client dataset. Aggregates is a collection of TAggregate objects, each of which defines a formula for calculating an aggregate value from a group of records in the client dataset. The individual aggregates can summarize all the records in the client dataset or subgroups of records that have the same value on a set of fields. Aggregates that summarize a subgroup of records are associated with indexes, and can only be used when the associated index is current.

Do not use Aggregates to obtain summarized values maintained by the client dataset. Aggregates can contain TAggregate objects that are not currently active. Instead, use the ActiveAggs property, which represents only those TAggregate objects that are currently active.

Note: You do not need to add a TAggregate object to the Aggregates collection when you add an aggregate field using the Fields editor. The Fields editor automatically adds TAggregate objects to Aggregates for every aggregate field you define.

See Also

ActiveAggs

TAggregate

IndexName

TAggregateField

Obtaining Aggregate Values

Specifying Aggregates

Defining an Aggregate Field

10.1.2.5.3 TSimpleDataSet.AggregatesActive Property

Specifies whether the client dataset calculates and maintains aggregate values.

Pascal

```
property AggregatesActive;
```

C++

```
__property AggregatesActive;
```

Description

Use AggregatesActive to indicate whether the client dataset should maintain aggregate values. When AggregatesActive is false (the default), the client dataset does not support maintained aggregates. This allows the client dataset to avoid the overhead of calculating values that the application is not using. When AggregatesActive is true, the client dataset calculates and maintains all aggregate values specified by the Aggregates property that are compatible with the current index.

When aggregates are maintained, the Value method of every active aggregate object returns a value that reflects the current data in the client dataset. When users edit the data in the dataset, these values are recalculated to reflect the user's changes.

Note: To selectively enable and disable aggregates rather than turning them all on or off at once, use the Active property of individual TAggregate objects. These objects are available through the Aggregates property.

See Also

ActiveAggs

Aggregates

IndexName

Value

Active

10.1.2.5.4 TSimpleDataSet.AutoCalcFields Property

Determines when the OnCalcFields event is triggered and when lookup field values are calculated.

Pascal

```
property AutoCalcFields;
```

C++

```
__property AutoCalcFields;
```

Description

Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

The dataset is opened.

The dataset is put into dsEdit state.

Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

The dataset is opened.

The dataset is put into dsEdit state.

A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

See Also

OnCalcFields

Calculating Fields

10.1.2.5.5 TSimpleDataSet.Connection Property

Identifies the TSQLConnection (see page 1580) component that connects the dataset to a database server.

Pascal

```
property Connection: TSQLConnection;
```

C++

```
__property TSQLConnection Connection;
```

Description

Use Connection to specify the component that connects this dataset to a database server. TSimpleDataSet (see page 896) can use either an external TSQLConnection (see page 1580) component or, if you set the ConnectionName property instead, an internally-created component. By using an external TSQLConnection (see page 1580) component, you can let multiple datasets share a single database connection.

See Also

Setting Up a Simple Dataset

Connecting to a Data Store Using TADOConnection

10.1.2.5.6 TSimpleDataSet.Constraints Property

Specifies record-level constraints that must be met when editing the data.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to read or add record-level constraints to the dataset. Record-level constraints usually impose relationships between the fields in a single record. Constraints that limit the value of a single field should be added to the individual field components.

See Also

TCheckConstraint

CustomConstraint

ImportedConstraint

Constraining Data Values

10.1.2.5.7 TSimpleDataSet.DataSet Property

Provides access to the internal dataset that fetches data from the server.

Pascal

```
property DataSet: TInternalSQLDataSet;
```

C++

```
__property TInternalSQLDataSet DataSet;
```

Description

Read (see page 3634) DataSet to access the internal dataset that TSimpleDataSet (see page 896) uses to fetch data from the server. TSimpleDataSet (see page 896) populates its Data property by fetching data from this internal dataset using an internal provider component.

See Also

Data

10.1.2.5.8 TSimpleDataSet.DisableStringTrim Property

Specifies whether trailing spaces are trimmed from field values when records are posted.

Pascal

```
property DisableStringTrim;
```

C++

```
__property DisableStringTrim;
```

Description

When DisableStringTrim is false, trailing spaces are trimmed from field values when records are posted. When DisableStringTrim is true, field values are posted exactly as typed, including any trailing spaces.

10.1.2.5.9 TSimpleDataSet.FetchOnDemand Property

Indicates whether to fetch data packets from the provider on an as-needed basis.

Pascal

```
property FetchOnDemand;
```

C++

```
__property FetchOnDemand;
```

Description

By default FetchOnDemand is true, meaning that the client dataset fetches additional packets of data as needed (for example, as a user scrolls through data, or conducts a search). If the provider does not automatically include BLOB fields or nested detail sets in data packets, setting FetchOnDemand to true causes the client dataset to fetch these values when needed as well.

Set FetchOnDemand to false only if your application contains code that explicitly fetches data packets as needed using the GetNextPacket method. Similarly, if FetchOnDemand is false and the provider does not include BLOB data by default (Options includes poFetchBlobsOnDemand), the application must fetch BLOB fields explicitly using the FetchBlobs method. If the provider does not include nested detail sets (Options includes poFetchDetailsOnDemand) and FetchOnDemand is false, the application must fetch nested details explicitly using the FetchDetails method.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

GetNextPacket

Options

FetchBlobs

FetchDetails

Requesting Data from the Source Dataset or Document

10.1.2.5.10 TSimpleDataSet.FieldDefs Property

Points to the list of field definitions for the dataset.

Pascal

```
property FieldDefs;
```

C++

```
__property FieldDefs;
```

Description

FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table with CreateTable or CreateDataSet.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Note: If the dataset includes object field descendants, FieldDefs represents a hierarchical view of the data, meaning that the definitions include object field definitions. To determine the definitions in a flattened view, use FieldDefList instead.

C++ Examples:

```
/*
The following code creates and activates a client dataset in
the form's OnCreate event handler.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TFieldDefs *pDefs = CDS2->FieldDefs;
  TFieldDef *pDef = pDefs->AddFieldDef();
  pDef->DataType = ftInteger;
  pDef->Name = "Field1";

  pDef = pDefs->AddFieldDef();
  pDef->DataType = ftString;
  pDef->Size = 10;
  pDef->Name = "Field2";

  TIndexDef *pIDef = CDS2->IndexDefs->AddIndexDef();
  pIDef->Fields = "Field1";
  pIDef->Name = "IntIndex";

  CDS2->CreateDataSet();
}
```

```
/*
The following example shows how to create a table.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  if (Table1->Exists)
    MessageDlg("CustInfo table already exists.", mtWarning, TMsgDlgButtons() << mbOK, 0);
  else // Don't overwrite an existing table
  {
    // describe the fields in the table
    Table1->FieldDefs->Clear();
    TFieldDef *newDef = Table1->FieldDefs->AddFieldDef();
    newDef->Name = "Field1";
    newDef->DataType = ftInteger;
    newDef->Required = true;
    newDef = Table1->FieldDefs->AddFieldDef();
    newDef->Name = "Field2";
    newDef->DataType = ftString;
    newDef->Size = 30;
    // Next, describe any indexes
    Table1->IndexDefs->Clear();
    /* the 1st index has no name because it is a Paradox primary key */
    Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
    Table1->IndexDefs->Add("Fld2Index", "Field2", TIndexOptions() << ixCaseInsensitive);
    // Now that we have specified what we want, create the table
    Table1->CreateTable();
    Table1->Active = True;
    for (int i = 1; i <= 20; i++)
      Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  }
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->Active = True;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Close();
  Table1->DeleteTable();
}
```

Delphi Examples:

```
{
The following code creates and activates a client dataset in
the form's OnCreate event handler.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with CDS2 do
  begin
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftInteger;
      Name := 'Field1';
    end;
    with FieldDefs.AddFieldDef do
    begin
      DataType := ftString;
      Size := 10;
      Name := 'Field2';
```

```
    end;
    with IndexDefs.AddIndexDef do
    begin
      Fields := 'Field1';
      Name := 'IntIndex';
    end;
    CreateDataSet;
  end;
end;
{ The following example shows how to create a table. }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Close;
  Table1.DeleteTable;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
      MessageDlg('CustInfo table already exists.', mtWarning, [mbOK], 0)
    else
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
```

```
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.Active:= True;
end;
```

See Also

CreateTable

Fields

FieldValues

ClearFields

GetFieldList

GetFieldNames

FieldByName

FindField

SetFields

ObjectView

AggFields

CreateDataSet

Creating and Deleting Tables

10.1.2.5.11 TSimpleDataSet.FileName Property

Specifies a file that stores the data of the client dataset.

Pascal

```
property FileName;
```

C++

```
__property FileName;
```

Description

Set FileName if the client dataset always reads its data from and writes its data to a single file. When FileName represents a valid file name, the client dataset automatically writes its data to that file when it is closed. When FileName specifies the name of an existing file, the client dataset automatically reads its data from that file when it is opened.

Changes are not merged into the data when it is saved to the file. This allows an application that employs the "briefcase" model to set FileName without losing updates that are destined for a database server. Single-tiered applications should explicitly merge changes before closing the dataset and saving the data to disk.

Note: When client datasets write their data to a file, they overwrite the contents of the file. Any previous contents of the file are lost.

To read from or write to a different file, use the LoadFromFile and SaveToFile methods instead. These methods can also be used to load and save data at times other than when the Active property changes.

See Also

Open

Close

Active

LoadFromFile

SaveToFile

Loading Data from a File or Stream

Saving Data to a File or Stream

Using a Dedicated File on Disk

10.1.2.5.12 TSimpleDataSet.Filter Property

Specifies the text of the current filter.

Pascal

```
property Filter;
```

C++

```
__property Filter;
```

Description

Use Filter to specify a dataset filter. When filtering is applied to a client dataset, only the records that meet a filter's conditions are available to an application. Filter contains the string that describes the filter condition. The Filter property is effective only if the Filtered property is set to True. In this case, if you change the Filter property, the filter is automatically updated.

The following table lists all the operators that can be used when defining a filter, as well as usage examples.

Operator or function comparisons	Example	Comment
=	State = 'CA'	
<>	State <> 'CA'	
>=	DateEntered >= '1/1/1998'	
<=	Total <= 100,000	
>	Percentile > 50	
<	Field1 < Field2	
BLANK	State <> 'CA' or State = BLANK	Blank records do not appear unless explicitly included in the filter.
IS NULL	Field1 IS NULL	
IS NOT NULL	Field1 IS NOT NULL.	
Logical operators		

and	State = 'CA' and Country = 'US'	
or	State = 'CA' or State = 'MA'	
not	not (State = 'CA')	
Arithmetic operators		
+	Total + 5 > 100	Applies to numbers, strings, or date (time).
-	Field1 - 7 <> 10	Applies to numbers, dates, or date (time).
*	Discount * 100 > 20	Applies to numbers only.
/	Discount > Total / 5	Applies to numbers only.
String functions		
Upper	Upper(Field1) = 'ALWAYS'	
Lower	Lower(Field1 + Field2) = 'josp'	
Substring	Substring(DateString,8) = '1998' | Substring(DateString,1,3) = 'JAN'	Extracts a substring from the string named in the first argument, starting at the position specified in the second argument. The third argument represents the number of characters copied from the string to the substring. If the third argument is missing, the string is copied from the specified position until the end.
Trim	Trim(Field1 + Field2) | Trim(Field1, '-')	Removes spaces from the beginning and the end of a string.
TrimLeft	TrimLeft(StringField) | TrimLeft(Field1, '$') <> ''	Removes spaces from the beginning of a string.
TrimRight	TrimRight(StringField) | TrimRight(Field1, '.') <> ''	Removes spaces from the end of a string.
DateTime functions		
Year	Year(DateField) = 2000	
Month	Month(DateField) <> 12	
Day	Day(DateField) = 1	
Hour	Hour(DateField) < 16	
Minute	Minute(DateField) = 0	
Second	Second(DateField) = 30	
GetDate	GetDate - DateField > 7	Represents current date and time.
Date	DateField = Date(GetDate)	Returns the date portion of a date-time value.
Time	TimeField > Time(GetDate)	Returns the time portion of a date-time value.
Miscellaneous		
Like	Memo LIKE '%filters%'	Works like SQL-92 without the ESC clause. When applied to BLOB fields, FilterOptions determines whether case is considered.
In	Day(DateField) in (1,7)	Works like SQL-92. Second argument is a list of values all with the same type.
*	State = 'M*'	Wildcard for partial comparisons.

See Also

Filtered

FilterOptions

FindFirst

FindLast

FindNext

FindPrior

OnFilterRecord

Filter

10.1.2.5.13 TSimpleDataSet.Filtered Property

Specifies whether or not filtering is active for a dataset.

Pascal

```
property Filtered;
```

C++

```
__property Filtered;
```

Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is true, then filtering is active. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to true. Because unidirectional datasets do not support filters, when you set Filtered to true on a unidirectional dataset, it raises an exception.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time an attempt is made to the read the record from the dataset while the filter is in effect, the record is not visible. If that happens, the next record that passes the filter condition becomes the active record.

See Also

Filter

FilterOptions

OnFilterRecord

Locate

Lookup

10.1.2.5.14 TSimpleDataSet.FilterOptions Property

Specifies whether or not filtering is case insensitive, and whether or not partial comparisons are permitted when filtering records.

Pascal

```
property FilterOptions;
```

C++

```
__property FilterOptions;
```

Description

Set FilterOptions to specify whether or not filtering is case insensitive when filtering on string or character fields, and whether or not partial comparisons for matching filter conditions is allowed.

Note: Because unidirectional datasets do not support filters, when you set the FilterOptions property of a unidirectional dataset,

it raises an exception.

By default, FilterOptions is set to an empty set. For filters based on string fields, set FilterOptions to foCaseInsensitive to catch all variations on a string regardless of capitalization.

When a string in a filter ends with an asterisk (*), it can be used to match partial strings. To disable matching of partial strings and to treat the asterisk as a literal character in string comparisons, set FilterOptions to include foNoPartialCompare.

See Also

Filter

Filtered

OnFilterRecord

Locate

Lookup

10.1.2.5.15 TSimpleDataSet.IndexDefs Property

Contains information about the indexes for a client dataset.

Pascal

```
property IndexDefs;
```

C++

```
__property IndexDefs;
```

Description

Examine IndexDefs for index information. IndexDefs maintains an array of TIndexDef items, each of which describes an available index for the dataset. Set the value of IndexDefs before calling CreateDataSet to create a set of indexes with the client dataset.

Note: The index items in IndexDefs may not always reflect the current indexes available for a dataset. Before examining IndexDefs, call the IndexDefs.Update (Delphi) or IndexDefs->Update (C++) method to refresh the item list.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
  for I := 0 to CDS2.IndexDefs.Count - 1 do
    if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
    begin
      { set that index as the current index for the dataset}
      CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
    end;
  CDS2.Active := True;
end;
```

See Also

IndexFieldNames

TIndexDef

CreateDataSet

Adding a New Index

10.1.2.5.16 TSimpleDataSet.IndexFieldNames Property

Lists the fields to use as an index.

Pascal

```
property IndexFieldNames;
```

C++

```
__property IndexFieldNames;
```

Description

Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the dataset, separating names with semicolons. Ordering of field names is significant.

Note: Indexes added using IndexFieldNames do not support grouping or maintained aggregates.

Tip: Use IndexFieldNames to create sort orders on the fly at runtime.

Note: The IndexFieldNames and IndexName properties are mutually exclusive. Setting one clears the other.

See Also

IndexDefs

IndexName

Creating an Index with IndexFieldNames

Adding a New Index

10.1.2.5.17 TSimpleDataSet.IndexName Property

Identifies an index for the client dataset.

Pascal

```
property IndexName;
```

C++

```
__property IndexName;
```

Description

Use IndexName to specify an alternative index for a client dataset.If IndexName is empty, the dataset's sort order is based on the IndexFieldNames property or on its default ordering in the data packet. Default ordering is determined by the predefined index, DEFAULT_ORDER.

If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexFieldNames and IndexName are mutually exclusive. Setting one clears the other.

C++ Examples:

```
/*
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
*/
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2->Active = False;
  // Get the current available indices
  CDS2->IndexDefs->Update();
  // Find a field named "Field2"
  for (int I = 0; I < CDS2->IndexDefs->Count; I++)
    if (CDS2->IndexDefs->Items[I]->Fields == "Field2")
    {
      // set that index as the current index for the dataset}
      CDS2->IndexName = CDS2->IndexDefs->Items[I]->Name;
    }
  CDS2->Active = True;
}
```

Delphi Examples:

```
{
This example uses the IndexName property to sort the records
in a client dataset on the Field2 field.
}
procedure TForm1.Button3Click(Sender: TObject);
var
  I : Integer;
begin
  // If Active is True, Update will remove Fields added in FormCreate
  // and add the DEFAULT_ORDER and CHANGEINDEX Fields
  CDS2.Active := False;
  { Get the current available indices }
  CDS2.IndexDefs.Update;
  { Find a field named 'Field2' }
```

```
    for I := 0 to CDS2.IndexDefs.Count - 1 do
      if CDS2.IndexDefs.Items[I].Fields = 'Field2' then
      begin
        { set that index as the current index for the dataset}
        CDS2.IndexName := CDS2.IndexDefs.Items[I].Name;
      end;
    CDS2.Active := True;
  end;
```

See Also

IndexFieldNames

Deleting and Switching Indexes

10.1.2.5.18 TSimpleDataSet.MasterFields Property

Names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship.

Pascal

```
property MasterFields;
```

C++

```
__property MasterFields;
```

Description

Use MasterFields after setting the MasterSource property to specify the names of one or more fields to use in establishing a detail-master relationship between this dataset and the one specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

See Also

MasterSource

Making the Table a Detail of Another Dataset

10.1.2.5.19 TSimpleDataSet.MasterSource Property

Specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one.

Pascal

```
property MasterSource;
```

C++

```
__property MasterSource;
```

Description

Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this dataset and another one.

At design time choose an available data source from the MasterSource property's drop-down list in the Object Inspector.

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

Tip: MasterSource establishes a master/detail relationship using the traditional linked cursor approach. Alternately, client datasets can participate in master/detail relationships using nested datasets, where the detail dataset is stored with the master table's data. To assign a client dataset's data as the value of a nested detail, use the DataSetField property.

See Also

MasterFields

Making the Table a Detail of Another Dataset

10.1.2.5.20 TSimpleDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView;
```

C++

```
__property ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList

Fields

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

10.1.2.5.21 TSimpleDataSet.PacketRecords Property

Indicates the number or type of records in a single data packet.

Pascal

```
property PacketRecords;
```

C++

```
__property PacketRecords;
```

Description

Use PacketRecords to specify how many records should be contained in a single packet fetched from a provider, or to indicate that the packet returned should contain only metadata.

When a client dataset is instantiated, PacketRecords is automatically set to -1, meaning that a single packet should contain all records in the dataset.

If PacketRecords is greater than zero, then it specifies the number of records to return in a packet.

To retrieve the metadata for a dataset, set PacketRecords to 0. When PacketRecords is zero, the provider returns only information from its dataset that defines the database's structure, such as table, column, constraint, and domain definitions.

Warning: When PacketRecords is greater than 0, the client dataset must fetch its data incrementally. If the provider is on a stateless application server, the client dataset must inform the provider where to start fetching data. This communication can occur using a BeforeGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
knows where to begin the next data packet. It also sends
some application-specific information, which is stored in
Memo1.  This method is used to pass the sql statement and the
value of the first field for the last record to the server so
that the server can return the correct records. The complete
project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
var
  LastValue: OleVariant;
  CDSClone: TClientDataSet;
begin
  if ClientDataSet1.Active then
  begin
    CDSClone := TClientDataSet.Create(Form1);
    try
      CDSClone.CloneCursor(ClientDataSet1, True);
      { turn off FetchOnDemand so that the clone only fetches
        the last LOCAL record }
      CDSClone.FetchOnDemand := False;
      CDSClone.Last;
      LastValue := CDSClone.Fields[0].AsString;
      CDSClone.Close;
    finally
      CDSClone.Free;
```

```
    end;
  end else
    LastValue := NULL;
  OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
  ClientDataSet1.GetNextPacket;
end;
```

See Also

FetchOnDemand

GetNextPacket

BeforeGetRecords

Requesting Data from the Source Dataset or Document

Using the BDE to Cache Updates

Updating the Dataset

10.1.2.5.22 TSimpleDataSet.Params Property

Contains parameter values that are sent to the provider.

Pascal

```
property Params;
```

C++

```
__property Params;
```

Description

Use Params to specify parameter values that the provider should pass to a source dataset. If the client dataset uses an internal provider, Params represents the values for any parameters used by the SQL statement specified by the CommandText property. If the client dataset uses an external provider that is associated with a dataset that represents a query or stored procedure, these parameter values supply input parameter values for that query or stored procedure and return any output parameters of a stored procedure. If the provider is associated with a TTable or TSQLTable component, Params limits the records that are sent in data packets. After specifying the parameters (at design time or runtime), they are sent to the provider automatically when the client dataset fetches data or executes a query or stored procedure using its Execute method.

To send parameter values to a query or stored procedure, for every parameter on the query or stored procedure for which the client dataset should send a value, add a parameter object (TParam) to Params. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the provider when the client dataset fetches data.

To send parameter values to a TTable or TSQLTable, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the provider only contain records that match the assigned value on the corresponding fields. You can achieve the same affect by applying a filter to the field, but using Params is more efficient because the data need not be included in data packets.

Note: To initialize Params so that it reflects the parameters of a provider's associated query or stored procedure, call the FetchParams method.

Warning: The parameters are only applied to the provider when a query or stored procedure is run the first time the client dataset fetches records. To change parameter values and force a query or stored procedure to be rerun with new values, use the Execute method.

C++ Examples:

```
/*
  This example saves params from a ClientDataSet, alters them
  and restores them.
*/
TParams *SavedParams;

void __fastcall TForm1::SaveParamsClick(TObject *Sender)
{
  // Save the parameters for the TDataSet
  SavedParams->Assign(CDS->Params);
}

void __fastcall TForm1::AlterParamsClick(TObject *Sender)
{
  CDS->Params->CreateParam(ftString, "stringParam", ptInput);  // you can also edit the params
in the CDS dynamically
  CDS->Params->CreateParam(ftSmallint, "smallIntParam", ptInput);
  CDS->Params->CreateParam(ftInteger, "intParam", ptInput);
  CDS->Params->CreateParam(ftWord, "wordParam", ptInput);
  CDS->Params->CreateParam(ftBoolean, "boolParam", ptInput);
  CDS->Params->CreateParam(ftFloat, "floatParam", ptInput);
  CDS->Params->CreateParam(ftCurrency, "currencyParam", ptInput);
  CDS->Params->CreateParam(ftBCD, "bcdParam", ptInput);
  CDS->Params->CreateParam(ftDate, "dateParam", ptInput);
  CDS->Params->CreateParam(ftTime, "timeParam", ptInput);
  CDS->Params->CreateParam(ftDateTime, "dateTimeParam", ptInput);
  // Call FetchParams to ensure parameters reflect server metadata
  CDS->FetchParams();
//  with CDS->Params do
  for (int I = 0; I < CDS->Params->Count; I++)
  {
    AnsiString ListItem = ListBox1->Items->Strings[I];
    switch (CDS->Params->Items[I]->DataType)
    {
      case ftString:
        CDS->Params->Items[I]->AsString = ListItem; break;
      case ftSmallint:
        CDS->Params->Items[I]->AsSmallInt = StrToIntDef(ListItem, 0); break;
      case ftInteger:
        CDS->Params->Items[I]->AsInteger = StrToIntDef(ListItem, 0); break;
      case ftWord:
        CDS->Params->Items[I]->AsWord = StrToIntDef(ListItem, 0); break;
      case ftBoolean:
        CDS->Params->Items[I]->AsBoolean = (ListItem == "True"); break;
      case ftFloat:
        CDS->Params->Items[I]->AsFloat = StrToFloat(ListItem); break;
      case ftCurrency:
        CDS->Params->Items[I]->AsCurrency = double(StrToFloat(ListItem)); break;
      case ftBCD:
        CDS->Params->Items[I]->AsBCD = StrToCurr(ListItem); break;
      case ftDate:
        CDS->Params->Items[I]->AsDate = StrToDate(ListItem); break;
      case ftTime:
        CDS->Params->Items[I]->AsTime = StrToTime(ListItem); break;
      case ftDateTime:
        CDS->Params->Items[I]->AsDateTime = StrToDateTime(ListItem); break;
```

```
    }
  }
}

void __fastcall TForm1::RestoreParamsClick(TObject *Sender)
{
  // Restore the parameters to TDataSet
  CDS->Params->AssignValues(SavedParams);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Initialize SavedParams
  SavedParams = new TParams;
  ListBox1->Items->Add("Here is a string");
  ListBox1->Items->Add("4");
  ListBox1->Items->Add("3456789");
  ListBox1->Items->Add("34");
  ListBox1->Items->Add("0");
  ListBox1->Items->Add("12.56");
  ListBox1->Items->Add("4.23");
  ListBox1->Items->Add("99.5");
  ListBox1->Items->Add("4/12/53");
  ListBox1->Items->Add("7:45:00");
  ListBox1->Items->Add("4/12/53 7:45:00");
}
```

Delphi Examples:

```
{
  This example uses the FetchParams method to sync the params
  with the server.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
  ListItem: string;
begin
  CDS.Params.CreateParam(ftString, 'stringParam', ptInput);  // you can also edit the params
in the CDS dynamically
  CDS.Params.CreateParam(ftSmallInt, 'smallIntParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'intParam', ptInput);
  CDS.Params.CreateParam(ftWord, 'wordParam', ptInput);
  CDS.Params.CreateParam(ftBoolean, 'boolParam', ptInput);
  CDS.Params.CreateParam(ftFloat, 'floatParam', ptInput);
  CDS.Params.CreateParam(ftCurrency, 'currencyParam', ptInput);
  CDS.Params.CreateParam(ftBCD, 'bcdParam', ptInput);
  CDS.Params.CreateParam(ftDate, 'dateParam', ptInput);
  CDS.Params.CreateParam(ftTime, 'timeParam', ptInput);
  CDS.Params.CreateParam(ftDateTime, 'dateTimeParam', ptInput);
  { Call FetchParams to ensure parameters reflect server metadata }
  CDS.FetchParams;
  with CDS.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
```

```
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              Items[I].AsBoolean := True
            else
              Items[I].AsBoolean := False;
          end;
        ftFloat:
          Items[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          Items[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          Items[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          Items[I].AsDate := StrToDate(ListItem);
        ftTime:
          Items[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          Items[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;

procedure TForm1.RestoreParamsClick(Sender: TObject);
begin
  { Restore the parameters to TDataSet }
  CDS.Params.AssignValues(SavedParams);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Initialize SavedParams }
  SavedParams := TParams.Create;
  ListBox1.Items.Add('Here is a string');
  ListBox1.Items.Add('4');
  ListBox1.Items.Add('3456789');
  ListBox1.Items.Add('34');
  ListBox1.Items.Add('0');
  ListBox1.Items.Add('12.56');
  ListBox1.Items.Add('4.23');
  ListBox1.Items.Add('99.5');
  ListBox1.Items.Add('4/12/53');
  ListBox1.Items.Add('7:45:00');
  ListBox1.Items.Add('4/12/53 7:45:00');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SavedParams.Free;
end;
{
This example fills in the parameters of a client dataset
from the entries of a list box.
}
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
  ListItem: string;
begin
  { Call FetchParams to ensure parameters reflect server metadata }
```

```
    ClientDataSet1.FetchParams;
    with ClientDataSet1.Params do
    begin
      for I := 0 to Count - 1 do
      begin
        ListItem := ListBox1.Items[I];
        case Items[I].DataType of
          ftString:
            Items[I].AsString := ListItem;
          ftSmallInt:
            Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
          ftInteger:
            Items[I].AsInteger := StrToIntDef(ListItem, 0);
          ftWord:
            Items[I].AsWord := StrToIntDef(ListItem, 0);
          ftBoolean:
            begin
              if ListItem = 'True' then
                Items[I].AsBoolean := True
              else
                Items[I].AsBoolean := False;
            end;
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');

  end;
```

See Also

Params

Params

Active

FetchParams

Params (see page 1562)

Parameters

Getting Parameters from the Application Server

Limiting Records with Parameters

Overview of Using Cached Updates

Passing Parameters to the Source Dataset

Sending Query or Stored Procedure Parameters

Setting Up a Simple Dataset

10.1.2.5.23 TSimpleDataSet.ReadOnly Property

Specifies whether the client dataset is read-only for this application.

Pascal

```
property ReadOnly;
```

C++

```
__property ReadOnly;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is false, meaning users can potentially alter the dataset's data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to true.

When ReadOnly is true, the dataset's CanModify property is false.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a client dataset if it gets its data from a provider. The provider's Options property can limit whether data can be edited, and if so, what types of edits are allowed.

See Also

Active

CanModify

Options

10.1.2.5.24 TSimpleDataSet.StoreDefs Property

Indicates whether field and index definitions are saved with the client dataset.

Pascal

```
property StoreDefs;
```

C++

```
__property StoreDefs;
```

Description

If StoreDefs is true, the client dataset's index and field definitions are stored with the client dataset when the form or data module that contains it is saved. Setting StoreDefs to true makes the CreateDataSet method into a one-step procedure that creates fields and indexes at runtime.

See Also

FieldDefs

IndexDefs

CreateDataSet

10.1.2.6 TSimpleDataSet Events

The events of the TSimpleDataSet class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 931)	Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.
	AfterCancel (see page 931)	Occurs after an application completes a request to cancel modifications to the active record.
	AfterClose (see page 933)	Occurs after an application closes a dataset.
	AfterDelete (see page 935)	Occurs after an application deletes a record.
	AfterEdit (see page 937)	Occurs after an application starts editing a record.
	AfterExecute (see page 938)	Occurs after the client dataset receives the results from an execute command.
	AfterGetParams (see page 939)	Occurs after the client dataset receives parameter values from the provider.
	AfterGetRecords (see page 940)	Occurs after the client dataset fetches a data packet from the provider.
	AfterInsert (see page 940)	Occurs after an application inserts a new record.
	AfterOpen (see page 941)	Occurs after an application completes opening a dataset and before any data access occurs.
	AfterPost (see page 942)	Occurs after an application writes the active record to the database or change log and returns to browse state.
	AfterRefresh (see page 943)	Occurs after an application refreshes the data in the dataset.
	AfterRowRequest (see page 943)	Occurs after the client dataset fetches new information about the current record.
	AfterScroll (see page 944)	Occurs after an application scrolls from one record to another.
	BeforeApplyUpdates (see page 944)	Occurs before the client dataset applies updates to the remote server.
	BeforeCancel (see page 945)	Occurs before an application executes a request to cancel changes to the active record.
	BeforeClose (see page 946)	Occurs immediately before the dataset closes.
	BeforeDelete (see page 946)	Occurs before an application attempts to delete the active record.
	BeforeEdit (see page 947)	Occurs before an application enters edit mode for the active record.
	BeforeExecute (see page 947)	Occurs before the client dataset executes the provider's query or stored procedure.
	BeforeGetParams (see page 948)	Occurs before the client dataset fetches parameter values from the provider.
	BeforeGetRecords (see page 949)	Occurs before the client dataset fetches a data packet from the provider.
	BeforeInsert (see page 950)	Occurs before an application enters insert mode.
	BeforeOpen (see page 952)	Occurs before an application executes a request to open a dataset.
	BeforePost (see page 952)	Occurs before an application posts changes for the active record to the database or change log.
	BeforeRefresh (see page 953)	Occurs immediately before an application refreshes the data in the dataset.
	BeforeRowRequest (see page 953)	Occurs before the client dataset fetches new information about the current record.
	BeforeScroll (see page 954)	Occurs before an application scrolls from one record to another.
	OnCalcFields (see page 955)	Occurs when an application recalculates calculated fields.
	OnDeleteError (see page 955)	Occurs when an application attempts to delete a record and an exception is raised.
	OnEditError (see page 956)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnFilterRecord (see page 956)	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.
	OnNewRecord (see page 957)	Occurs when an application inserts or appends a new dataset record.

	OnPostError (see page 957)	Occurs when an application attempts to modify or insert a record and an exception is raised.
	OnReconcileError (see page 958)	Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

10.1.2.6.1 TSimpleDataSet.AfterApplyUpdates Event

Occurs after the client dataset applies updates to the provider but before it attempts to reconcile update errors.

Pascal

```
property AfterApplyUpdates;
```

C++

```
__property AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to respond to custom information returned to the client dataset from a provider's AfterApplyUpdates event handler. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

6.The client dataset generates an OnReconcileError for every record that could not be applied by the provider.

See Also

ApplyUpdates

AS_ApplyUpdates

ApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

OnReconcileError

Applying Updates

Communicating with the Client Dataset

10.1.2.6.2 TSimpleDataSet.AfterCancel Event

Occurs after an application completes a request to cancel modifications to the active record.

Pascal

```
property AfterCancel;
```

C++

```
__property AfterCancel;
```

Description

Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterCancel event occurs.  Set the StatusBar
SimplePanel property to True.
}
procedure TForm1.Table1AfterCancel(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := 'Record changes cancelled for ' + DataSet.Name;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Cancel;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Post
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
```

```
          Required := True;
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterCancel:= Table1AfterCancel;
  Table1.Active:= True;
end;
```

See Also

BeforeCancel

Cancel

State

10.1.2.6.3 TSimpleDataSet.AfterClose Event

Occurs after an application closes a dataset.

Pascal

```
property AfterClose;
```

C++

```
__property AfterClose;
```

Description

Write an AfterClose event handler to take specific action immediately after an application closes a dataset. For example, as a security measure, an application might clear a PASSWORD entry from the Params property of a TSQLConnection component when the dataset is closed.

AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  if not RemoteServer1.Connected then
    RemoteServer1.Connected := True;
  ClientDataSet1.Close;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].EnableControls;
  end;
  ClientDataSet1.Open;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  ClientDataSet1.Close;
  RemoteServer1.Connected := False;
  with RemoteServer1 do
  begin
    for I := 0 to DataSetCount - 1 do
      DataSets[I].DisableControls;
  end;
end;

procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
```

See Also

Active

State

BeforeClose

Close

Params (see page 1621)

Opening and Closing Datasets

10.1.2.6.4 TSimpleDataSet.AfterDelete Event

Occurs after an application deletes a record.

Pascal

```
property AfterDelete;
```

C++

```
__property AfterDelete;
```

Description

Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

C++ Examples:

```
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
```

```
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
}
procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
begin
  StatusBar1.SimpleText := Format('There are now %d records in the table',
[DataSet.RecordCount]);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Delete;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Table1:= TTable.Create(Form1);
  with Table1 do
  begin
    DatabaseName := 'DBDEMOS';
    TableType := ttParadox;
    TableName := 'CustInfo';
    Name := 'CustInfo';
    Table1.Active := False;
    { Don't overwrite an existing table }
    if Table1.Exists then
    begin
        Table1.Close;
        Table1.DeleteTable;
    end;
    begin
      { The Table component must not be active }
      { First, describe the type of table and give }
      { it a name }
      { Next, describe the fields in the table }
      with FieldDefs do
      begin
        Clear;
        with AddFieldDef do
        begin
          Name := 'Field1';
          DataType := ftInteger;
          Required := True;
```

```
        end;
        with AddFieldDef do
        begin
          Name := 'Field2';
          DataType := ftString;
          Size := 30;
        end;
      end;
      { Next, describe any indexes }
      with IndexDefs do
      begin
        Clear;
        { The 1st index has no name because it is
        { a Paradox primary key }
        with AddIndexDef do
        begin
          Name := '';
          Fields := 'Field1';
          Options := [ixPrimary];
        end;
        with AddIndexDef do
        begin
          Name := 'Fld2Indx';
          Fields := 'Field2';
          Options := [ixCaseInsensitive];
        end;
      end;
      { Call the CreateTable method to create the table }
      CreateTable;
      Table1.Active:= True;
      for i := 1 to 20 do
        Table1.AppendRecord([i*111, i*222]);
    end;
  end;
  DS2.DataSet:= Table1;
  DBGrid2.DataSource.DataSet:= Table1;
  Table1.AfterDelete:= Table1AfterDelete;
  Table1.Active:= True;
end;
```

See Also

State

BeforeDelete

Delete

10.1.2.6.5 TSimpleDataSet.AfterEdit Event

Occurs after an application starts editing a record.

Pascal

```
property AfterEdit;
```

C++

```
__property AfterEdit;
```

Description

Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

BeforeEdit

Edit

10.1.2.6.6 TSimpleDataSet.AfterExecute Event

Occurs after the client dataset receives the results from an execute command.

Pascal

```
property AfterExecute;
```

C++

```
__property AfterExecute;
```

Description

Write an AfterExecute event handler to respond to custom information returned to the client dataset from a provider's AfterExecute event handler. AfterExecute is part of the mechanism by which a client dataset and a provider communicate information when executing an SQL command from the client. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute

AS_Execute

Execute

AfterExecute

BeforeExecute

CommandText

Communicating with the Client Dataset

10.1.2.6.7 TSimpleDataSet.AfterGetParams Event

Occurs after the client dataset receives parameter values from the provider.

Pascal

```
property AfterGetParams;
```

C++

```
__property AfterGetParams;
```

Description

Write an AfterGetParams event handler to respond to custom information returned to the client dataset from a provider's AfterGetParams event handler. AfterGetParams is part of the mechanism by which a client dataset and a provider communicate information when the client application calls FetchParams. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams

AS_GetParams

GetParams

AfterGetParams

BeforeGetParams

Communicating with the Client Dataset

10.1.2.6.8 TSimpleDataSet.AfterGetRecords Event

Occurs after the client dataset fetches a data packet from the provider.

Pascal

```
property AfterGetRecords;
```

C++

```
__property AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to respond to custom information returned to the client dataset from a provider's AfterGetRecords event handler. AfterGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1. The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.
2. The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.
3. The provider generates an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.
4. The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.
5. The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

See Also

AS_GetRecords

GetRecords

AfterGetRecords

BeforeGetRecords

Communicating with the Client Dataset

10.1.2.6.9 TSimpleDataSet.AfterInsert Event

Occurs after an application inserts a new record.

Pascal

```
property AfterInsert;
```

C++

```
__property AfterInsert;
```

Description

Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
```

See Also

BeforeInsert

Append

Insert

10.1.2.6.10 TSimpleDataSet.AfterOpen Event

Occurs after an application completes opening a dataset and before any data access occurs.

Pascal

```
property AfterOpen;
```

C++

```
__property AfterOpen;
```

Description

Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

Delphi Examples:

```
{
This example adds an entry to a memo with a message when an
AfterOpen event or an AfterClose event occurs.  Populate the
ClientDataSet events with CheckButtonActive.
}
procedure TForm1.CheckButtonActive(DataSet: TDataSet);
begin
  if (Button1.Enabled) then
    Memo2.Lines.Add('Now that the dataset is open, record information is available.')
  else
    Memo2.Lines.Add('Now that the dataset is closed, record information is not available.');
  Button1.Enabled := not ClientDataSet1.Active;
  Button2.Enabled := ClientDataSet1.Active;
  Button3.Enabled := ClientDataSet1.Active;
end;
{
The following code fragment illustrates how DataSets and
DataSetCount can be used to ensure that an action is taken
for every open dataset.  Assign CheckButtonActive as the
AfterClose and AfterOpen event handlers for TClientDataSet1.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    I: Integer;
  begin
    if not RemoteServer1.Connected then
      RemoteServer1.Connected := True;
    ClientDataSet1.Close;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].EnableControls;
    end;
    ClientDataSet1.Open;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    I: Integer;
  begin
    ClientDataSet1.Close;
    RemoteServer1.Connected := False;
    with RemoteServer1 do
    begin
      for I := 0 to DataSetCount - 1 do
        DataSets[I].DisableControls;
    end;
  end;

  procedure TForm1.CheckButtonActive(DataSet: TDataSet);
  begin
    Button1.Enabled := not ClientDataSet1.Active;
    Button2.Enabled := ClientDataSet1.Active;
    Button3.Enabled := ClientDataSet1.Active;
  end;
```

See Also

Active

State

BeforeOpen

Open

Opening and Closing Datasets

10.1.2.6.11 TSimpleDataSet.AfterPost Event

Occurs after an application writes the active record to the database or change log and returns to browse state.

Pascal

```
property AfterPost;
```

C++

```
__property AfterPost;
```

Description

Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

Delphi Examples:

```
{
This example updates the form's status bar with a message
when an AfterPost event occurs.
```

```
}
procedure TForm1.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before posting a record');
end;
```

See Also

BeforePost

Post

10.1.2.6.12 TSimpleDataSet.AfterRefresh Event

Occurs after an application refreshes the data in the dataset.

Pascal

```
property AfterRefresh;
```

C++

```
__property AfterRefresh;
```

Description

Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

See Also

BeforeRefresh

Refresh

10.1.2.6.13 TSimpleDataSet.AfterRowRequest Event

Occurs after the client dataset fetches new information about the current record.

Pascal

```
property AfterRowRequest;
```

C++

```
__property AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to respond to custom information returned to the client dataset from a provider's AfterRowRequest event handler. AfterRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or

respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

AfterRowRequest

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

10.1.2.6.14 TSimpleDataSet.AfterScroll Event

Occurs after an application scrolls from one record to another.

Pascal

```
property AfterScroll;
```

C++

```
__property AfterScroll;
```

Description

Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

BeforeScroll

First

Last

MoveBy

Next

Prior

Navigating Datasets

10.1.2.6.15 TSimpleDataSet.BeforeApplyUpdates Event

Occurs before the client dataset applies updates to the remote server.

Pascal

```
property BeforeApplyUpdates;
```

C++

```
__property BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to send custom information to the client dataset's provider. BeforeApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's ApplyUpdates method, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeApplyUpdates event, where it can respond to or change that information before it applies the updates.

3.The provider applies the updates and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider's AfterApplyUpdates event handler.

See Also

ApplyUpdates

AS_ApplyUpdates

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

Applying Updates

Communicating with the Client Dataset

10.1.2.6.16 TSimpleDataSet.BeforeCancel Event

Occurs before an application executes a request to cancel changes to the active record.

Pascal

```
property BeforeCancel;
```

C++

```
__property BeforeCancel;
```

Description

Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

See Also

AfterCancel

Delete

Edit

Insert

10.1.2.6.17 TSimpleDataSet.BeforeClose Event

Occurs immediately before the dataset closes.

Pascal

```
property BeforeClose;
```

C++

```
__property BeforeClose;
```

Description

Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

See Also

Active

AfterClose

Close

Opening and Closing Datasets

Posting Data

10.1.2.6.18 TSimpleDataSet.BeforeDelete Event

Occurs before an application attempts to delete the active record.

Pascal

```
property BeforeDelete;
```

C++

```
__property BeforeDelete;
```

Description

Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

See Also

AfterDelete

Delete

Abort (see page 4047)

10.1.2.6.19 TSimpleDataSet.BeforeEdit Event

Occurs before an application enters edit mode for the active record.

Pascal

```
property BeforeEdit;
```

C++

```
__property BeforeEdit;
```

Description

Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

Delphi Examples:

```
{
This example adds an entry to a memo with a message
when an AfterEdit event occurs and when a BeforeEdit event
occurs.  Click on Run SQL to populate the DBGrid.  Then
select a cell to edit and click the Edit current cell
button, or just start editing the cell to invoke the
BeforeEdit and AfterEdit events.
}
procedure TForm1.ClientDataSet1AfterEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.ClientDataSet1BeforeEdit(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before editing record' + IntToStr(DataSet.RecNo));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  ClientDataSet1.Edit;
end;
```

See Also

State

AfterEdit

Edit

10.1.2.6.20 TSimpleDataSet.BeforeExecute Event

Occurs before the client dataset executes the provider's query or stored procedure.

Pascal

```
property BeforeExecute;
```

C++

```
__property BeforeExecute;
```

Description

Write a BeforeExecute event handler to send custom information to the client dataset's provider. BeforeExecute is part of the

mechanism by which a client dataset and a provider communicate information when explicitly executing an SQL command by the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's Execute method, the following events occur:

1.The client dataset receives a BeforeExecute event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeExecute event, where it can respond to or change that information before it executes the query or stored procedure.

3.The provider executes the appropriate command (either the command supplied by the CommandText property, or the command associated with a dataset such as a query or stored procedure).

4.The provider receives an AfterExecute event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeExecute event handler.

5.The client dataset unpacks any output parameters from the provider into the Params property and then receives an AfterExecute event, where it can respond to the custom information returned by the provider's AfterExecute event handler.

See Also

Execute

AS_Execute

Execute

BeforeExecute

AfterExecute

CommandText

Communicating with the Client Dataset

10.1.2.6.21 TSimpleDataSet.BeforeGetParams Event

Occurs before the client dataset fetches parameter values from the provider.

Pascal

```
property BeforeGetParams;
```

C++

```
__property BeforeGetParams;
```

Description

Write a BeforeGetParams event handler to send custom information to the client dataset's provider. BeforeGetparams is part of the mechanism by which a client dataset and a provider communicate information when passing parameter values from the provider's dataset. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the application calls the client dataset's FetchParams method, the following events occur:

1.The client dataset receives a BeforeGetParams event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetParams event, where it can respond to or change that information before it retrieves parameter values.

3.The provider encodes all of its dataset's current parameter values into an OleVariant.

4.The provider receives an AfterGetParams event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetParams event handler.

5.The client dataset unpacks the parameters from the provider into the Params property and then receives an AfterGetParams event, where it can respond to the custom information returned by the provider's AfterGetParams event handler.

See Also

FetchParams

AS_GetParams

GetParams

BeforeGetParams

AfterGetParams

Communicating with the Client Dataset

10.1.2.6.22 TSimpleDataSet.BeforeGetRecords Event

Occurs before the client dataset fetches a data packet from the provider.

Pascal

```
property BeforeGetRecords;
```

C++

```
__property BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to send custom information to the client dataset's provider. BeforeGetRecords is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches data from the application server, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant that is passed to the provider as the OwnerData parameter.

2.The provider receives a BeforeGetRecords event, where it can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider's AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows how to use the BeforeGetRecords event
handler (a TRemoteEventType value) to send the provider
information it needs for incremental data fetching. Before
fetching the next data packet, the client dataset packages
up the key value of the last record so that the provider
```

```
  knows where to begin the next data packet. It also sends
  some application-specific information, which is stored in
  Memo1.  This method is used to pass the sql statement and the
  value of the first field for the last record to the server so
  that the server can return the correct records. The complete
  project using this code example is in
  Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
  }
  procedure TForm1.ClientDataSet1BeforeGetRecords(Sender: TObject; var OwnerData: OleVariant);
  var
    LastValue: OleVariant;
    CDSClone: TClientDataSet;
  begin
    if ClientDataSet1.Active then
    begin
      CDSClone := TClientDataSet.Create(Form1);
      try
        CDSClone.CloneCursor(ClientDataSet1, True);
        { turn off FetchOnDemand so that the clone only fetches
          the last LOCAL record }
        CDSClone.FetchOnDemand := False;
        CDSClone.Last;
        LastValue := CDSClone.Fields[0].AsString;
        CDSClone.Close;
      finally
        CDSClone.Free;
      end;
    end else
      LastValue := NULL;
    OwnerData := VarArrayOf([Memo1.Lines.Text, LastValue]);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    ClientDataSet1.PacketRecords := StrToInt(Edit1.Text);
    ClientDataSet1.GetNextPacket;
  end;
```

See Also

AS_GetRecords

GetRecords

BeforeGetRecords

AfterGetRecords

Communicating with the Client Dataset

10.1.2.6.23 TSimpleDataSet.BeforeInsert Event

Occurs before an application enters insert mode.

Pascal

```
property BeforeInsert;
```

C++

```
__property BeforeInsert;
```

Description

Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

Delphi Examples:

```
{
This example updates a memo with a message when an
AfterInsert event or a BeforeInsert event occurs.
}
procedure TForm1.ClientDataSet1AfterInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('After inserting a record');
end;

procedure TForm1.ClientDataSet1BeforeInsert(DataSet: TDataSet);
begin
   Memo2.Lines.Add('Before inserting a record');
end;
{
This example uses the BeforeInsert event to do data
validation. If the StrToInt function raises an exception,
the edit control's contents are set to a valid value so
that the assignment to the INTEGER field in the dataset
will succeed.  Continuing after the exception causes a
'0' to be placed in the PORTA field.
DataSet design:
This example requires a ClientDataSet, a DBGrid, a
DataSource and a DBNavigator.  Name the ClientDataSet CDS.
Set the DataSet property in the DataSource to CDS and name
that DataSource DS.  Set the DataSource in the DBNavigator
to DS the DataSource in the DBGrid to DS.  Right click on CDS
and select the Fields Editor, then right click on the Field
Editor form and create two fields by selecting "new field".
In the field configuration window, set the name to HOST, the
Component value will automatically be set to CDSHOST.  Set
the filed type to String.  In the field configuration window
for the second field, set the name to PORTA, the Component
value will automatically be set to CDSPORTA.  Set the field
type to Integer.  The file used to load CDS must have an
attribute name of HOST with an attribute type of string and
an attribute name of PORTA with an attribute type of integer.
Add DB, DBCtrls, Grids, DBGrids and DBClient to the uses clause.
}
{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button3Click(Sender: TObject);
begin
  CDS.Insert;
  CDS.FieldByName('PORTA').AsInteger := StrToInt(Edit1.Text);
  CDS.Post;
end;

procedure TForm1.CDSBeforeInsert(DataSet: TDataSet);
begin
  try
  { Make sure edit field can be converted to integer --
   this will raise an exception if it can't }
    StrToInt(Edit1.Text);
  except
    Edit1.Text := '0';
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
```

```
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

State

AfterInsert

Append

Insert

10.1.2.6.24 TSimpleDataSet.BeforeOpen Event

Occurs before an application executes a request to open a dataset.

Pascal

```
property BeforeOpen;
```

C++

```
__property BeforeOpen;
```

Description

Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

See Also

Active

AfterOpen

Open

Opening and Closing Datasets

10.1.2.6.25 TSimpleDataSet.BeforePost Event

Occurs before an application posts changes for the active record to the database or change log.

Pascal

```
property BeforePost;
```

C++

```
__property BeforePost;
```

Description

Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation (Delphi) or throw an exception (C++).

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
```

```
  and calls the Abort procedure if the control is empty; Abort
  cancels the post before it happens.
  }
  procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
  begin
    if DBEdit1.Text = '' then
      Abort;
    DataSet.Fields[2].AsString := DBEdit1.Text;
  end;
```

See Also

AfterPost

Post

Abort (see page 4047)

Working with Field Component Methods at Runtime

10.1.2.6.26 TSimpleDataSet.BeforeRefresh Event

Occurs immediately before an application refreshes the data in the dataset.

Pascal

```
property BeforeRefresh;
```

C++

```
__property BeforeRefresh;
```

Description

Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

See Also

AfterRefresh

Refresh

10.1.2.6.27 TSimpleDataSet.BeforeRowRequest Event

Occurs before the client dataset fetches new information about the current record.

Pascal

```
property BeforeRowRequest;
```

C++

```
__property BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to send custom information to the client dataset's provider. BeforeRowRequest is part of the mechanism by which a client dataset and a provider communicate information about data fetching. When working with a provider on a stateless application server, this mechanism allows the client dataset and the provider to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant that is

passed to the provider as the OwnerData parameter.

2.The provider on the application server receives a BeforeRowRequest event, where it can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider's AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

BeforeRowRequest

AfterRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

10.1.2.6.28 TSimpleDataSet.BeforeScroll Event

Occurs before an application scrolls from one record to another.

Pascal

```
property BeforeScroll;
```

C++

```
__property BeforeScroll;
```

Description

Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

See Also

AfterScroll

First

Last

MoveBy

Next

Prior

Navigating Datasets

10.1.2.6.29 TSimpleDataSet.OnCalcFields Event

Occurs when an application recalculates calculated fields.

Pascal

```
property OnCalcFields;
```

C++

```
__property OnCalcFields;
```

Description

Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

A dataset is opened.

A dataset is put into dsEdit state.

A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

See Also

Active

AutoCalcFields

Open

Post

Calculating Fields

Using Internally Calculated Fields in Client Datasets

Defining a Calculated Field

10.1.2.6.30 TSimpleDataSet.OnDeleteError Event

Occurs when an application attempts to delete a record and an exception is raised.

Pascal

```
property OnDeleteError;
```

C++

```
__property OnDeleteError;
```

Description

Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

See Also

OnEditError

OnPostError

Delete

10.1.2.6.31 TSimpleDataSet.OnEditError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnEditError;
```

C++

```
__property OnEditError;
```

Description

Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

See Also

OnDeleteError

OnPostError

Edit

Insert

10.1.2.6.32 TSimpleDataSet.OnFilterRecord Event

Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

Pascal

```
property OnFilterRecord;
```

C++

```
__property OnFilterRecord;
```

Description

Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. Most TDataSet descendants (such as ADO datasets or BDE-based datasets) initialize Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can't be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables (Paradox, dBASE, Access, FoxPro), but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

See Also

Filter

Filtered

FilterOptions

State

Creating Filters

Navigating Records in a Filtered Dataset

Displaying and Editing a Subset of Data Using Filters

Writing an OnFilterRecord Event Handler

10.1.2.6.33 TSimpleDataSet.OnNewRecord Event

Occurs when an application inserts or appends a new dataset record.

Pascal

```
property OnNewRecord;
```

C++

```
__property OnNewRecord;
```

Description

Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

See Also

AfterInsert

BeforeInsert

10.1.2.6.34 TSimpleDataSet.OnPostError Event

Occurs when an application attempts to modify or insert a record and an exception is raised.

Pascal

```
property OnPostError;
```

C++

```
__property OnPostError;
```

Description

Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

See Also

OnDeleteError

OnEditError

Post

10.1.2.6.35 TSimpleDataSet.OnReconcileError Event

Occurs when a client dataset needs to reconcile an update to a record that could not be applied.

Pascal

```
property OnReconcileError;
```

C++

```
__property OnReconcileError;
```

Description

Write an OnReconcileError event handler to respond to error conditions that arise when the Reconcile method cycles through records that could not be applied to the database. This event handles only those update errors that were not handled in an OnUpdateError event handler.

The OnReconcileError event handler is called once for each record that could not be applied to the dataset or handled by the provider.

Warning: A provider can't detect when an update conflicts with another user's changes to a memo field. This situation does not generate an OnReconcileError event.

See Also

Reconcile

OldValue

NewValue

CurValue

ApplyUpdates

OnUpdateError

OnUpdateError

Applying Updates

Setting Up a Simple Dataset

Reconciling Update Errors

Using the BDE to Cache Updates

Updating the Dataset

11 SiteComp Namespace

This is namespace SiteComp.

Topics

Name	Description
Classes (see page 963)	The following table lists classes in this documentation.
Interfaces (see page 977)	The following table lists interfaces in this documentation.
Functions (see page 1120)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1121)	The following table lists structs, records, enums in this documentation.
Types (see page 1128)	The following table lists types in this documentation.
Variables (see page 1130)	The following table lists variables in this documentation.
Constants (see page 1133)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAbstractDesigntimeWarnings (see page 963)	TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.
	TAbstractInetFileType (see page 965)	This is class SiteComp.TAbstractInetFileType.
	TNotifyList (see page 968)	TNotifyList is a collection of objects that should be notified of changes.
	TScriptComponent (see page 973)	This is class SiteComp.TScriptComponent.
	TScriptObject (see page 976)	This is class SiteComp.TScriptObject.

Constants

Name	Description
AdapterActionHTMLElementTypeNames (see page 1133)	This is constant SiteComp.AdapterActionHTMLElementTypeNames.
AdapterDisplayHTMLElementTypeNames (see page 1133)	This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.
AdapterDisplayViewModeTypeNames (see page 1134)	This is constant SiteComp.AdapterDisplayViewModeTypeNames.
AdapterInputHTMLElementTypeNames (see page 1134)	This is constant SiteComp.AdapterInputHTMLElementTypeNames.

Functions

	Name	Description
	FullyQualifiedFieldName (see page 1120)	This is function SiteComp.FullyQualifiedFieldName.
	FullyQualifiedName (see page 1120)	This is function SiteComp.FullyQualifiedName.

Interfaces

	Name	Description
	IActionImageProducer (see page 978)	IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.
	IAdapterAccess (see page 981)	This is class SiteComp.IAdapterAccess.
	IAdapterActionAccess (see page 982)	This is class SiteComp.IAdapterActionAccess.
	IAdapterActionAttributes (see page 983)	This is class SiteComp.IAdapterActionAttributes.
	IAdapterDispatcher (see page 985)	IAdapterDispatcher is the interface for interacting with the adapter dispatcher.
	IAdapterDispatchParams (see page 987)	IAdapterDispatchParams describes adapter action parameters and adapter field values.
	IAdapterError (see page 992)	IAdapterError is the interface for retrieving adapter error information.
	IAdapterErrors (see page 995)	IAdapterErrors is implemented by adapter errors objects.
	IAdapterFieldAccess (see page 998)	This is class SiteComp.IAdapterFieldAccess.
	IAdapterFieldAttributes (see page 999)	This is class SiteComp.IAdapterFieldAttributes.
	IAdapterHiddenField (see page 1001)	This is class SiteComp.IAdapterHiddenField.
	IAdapterHiddenFields (see page 1003)	This is class SiteComp.IAdapterHiddenFields.
	IAdapterItemRequestParams (see page 1005)	IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.
	IAdapterMode (see page 1009)	This is class SiteComp.IAdapterMode.
	IAdapterRequestHandler (see page 1012)	IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.
	IAsFormatted (see page 1013)	This is class SiteComp.IAsFormatted.
	IDefaultPageFileName (see page 1014)	This is class SiteComp.IDefaultPageFileName.
	IFieldValuesAdapter (see page 1015)	This is class SiteComp.IFieldValuesAdapter.
	IFindIncludeFile (see page 1016)	This is class SiteComp.IFindIncludeFile.
	IGetAdapterActions (see page 1017)	This is class SiteComp.IGetAdapterActions.
	IGetAdapterDisplayCharacteristics (see page 1018)	This is class SiteComp.IGetAdapterDisplayCharacteristics.
	IGetAdapterErrors (see page 1019)	This is class SiteComp.IGetAdapterErrors.
	IGetAdapterFields (see page 1020)	This is class SiteComp.IGetAdapterFields.
	IGetAdapterHiddenFields (see page 1021)	This is class SiteComp.IGetAdapterHiddenFields.
	IGetAdapterImage (see page 1022)	This is class SiteComp.IGetAdapterImage.
	IGetAdapterItemRequestParams (see page 1023)	This is class SiteComp.IGetAdapterItemRequestParams.
	IGetAdapterRequestDefaultResponse (see page 1024)	This is class SiteComp.IGetAdapterRequestDefaultResponse.
	IGetAdapterValuesList (see page 1026)	This is class SiteComp.IGetAdapterValuesList.
	IGetDefaultAction (see page 1027)	This is class SiteComp.IGetDefaultAction.
	IGetDesigntimeWarnings (see page 1029)	This is class SiteComp.IGetDesigntimeWarnings.
	IGetFieldValuesAdapter (see page 1030)	This is class SiteComp.IGetFieldValuesAdapter.
	IGetHTMLStyle (see page 1031)	IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.
	IGetProducerComponent (see page 1034)	This is class SiteComp.IGetProducerComponent.
	IGetScriptObject (see page 1035)	This is class SiteComp.IGetScriptObject.
	IGetScriptObjectClassKey (see page 1036)	This is class SiteComp.IGetScriptObjectClassKey.
	IGetWebAppComponents (see page 1037)	This is class SiteComp.IGetWebAppComponents.
	IIDEFileManager (see page 1040)	This is class SiteComp.IIDEFileManager.
	IImageProducer (see page 1041)	IImageProducer is the interface for accessing the image that an adapter component supplies.
	IInternetEnvOptions (see page 1042)	This is class SiteComp.IInternetEnvOptions.
	IInternetExecuteScript (see page 1045)	This is class SiteComp.IInternetExecuteScript.
	IInternetFileTypes (see page 1047)	This is class SiteComp.IInternetFileTypes.
	IInternetScriptDebugger (see page 1048)	This is class SiteComp.IInternetScriptDebugger.
	IIsAdapterActionList (see page 1049)	This is class SiteComp.IIsAdapterActionList.
	IIterateIntfSupport (see page 1050)	This is class SiteComp.IIterateIntfSupport.
	IIterateObjectSupport (see page 1051)	This is class SiteComp.IIterateObjectSupport.
	IIteratorIndex (see page 1052)	IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.

	IIteratorSupport (see page 1054)	IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.
	INotifyAdapterChange (see page 1057)	INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.
	INotifyList (see page 1058)	INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.
	IPageDispatcher (see page 1060)	IPageDispatcher is the interface for interacting with the page dispatcher.
	IPageResult (see page 1064)	This is class SiteComp.IPageResult.
	ISessionsService (see page 1066)	ISessionsService is the interface for controlling a TSessionsService object.
	ISessionsServiceConnectID (see page 1068)	This is class SiteComp.ISessionsServiceConnectID.
	ISessionsServicePersist (see page 1070)	This is class SiteComp.ISessionsServicePersist.
	ISessionsServiceStoreID (see page 1071)	This is class SiteComp.ISessionsServiceStoreID.
	IValuesListAdapter (see page 1072)	IValuesListAdapter is the interface for supplying name/value pairs.
	IWebActionsList (see page 1076)	This is class SiteComp.IWebActionsList.
	IWebApplicationInfo (see page 1078)	IWebApplicationInfo is the interface for the application adapter.
	IWebContainerSubComponent (see page 1081)	This is class SiteComp.IWebContainerSubComponent.
	IWebDataFields (see page 1081)	This is class SiteComp.IWebDataFields.
	IWebEnabled (see page 1083)	This is class SiteComp.IWebEnabled.
	IWebFormComponentContainer (see page 1084)	This is class SiteComp.IWebFormComponentContainer.
	IWebGetActionName (see page 1086)	This is class SiteComp.IWebGetActionName.
	IWebGetDisplayLabel (see page 1088)	This is class SiteComp.IWebGetDisplayLabel.
	IWebGetDisplayWidth (see page 1089)	This is class SiteComp.IWebGetDisplayWidth.
	IWebGetFieldName (see page 1090)	This is class SiteComp.IWebGetFieldName.
	IWebGetFieldValue (see page 1091)	This is class SiteComp.IWebGetFieldValue.
	IWebGetFieldValues (see page 1092)	IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.
	IWebGetMaxLength (see page 1095)	This is class SiteComp.IWebGetMaxLength.
	IWebGetSessionIDs (see page 1096)	This is class SiteComp.IWebGetSessionIDs.
	IWebImageHREF (see page 1097)	IWebImageHREF is the interface for obtaining an HREF that addresses an image.
	IWebInputName (see page 1098)	This is class SiteComp.IWebInputName.
	IWebIsDefaultAction (see page 1100)	This is class SiteComp.IWebIsDefaultAction.
	IWebIsDefaultField (see page 1101)	This is class SiteComp.IWebIsDefaultField.
	IWebSession (see page 1102)	IWebSession allows an application to terminate a Web session.
	IWebSessionAttributes (see page 1103)	This is class SiteComp.IWebSessionAttributes.
	IWebSessionID (see page 1108)	This is class SiteComp.IWebSessionID.
	IWebSessionIDs (see page 1109)	IWebSessionIDs lists the session IDs (see page 1111) for all sessions that TSessionsService stores in memory.
	IWebSessionValues (see page 1112)	This is class SiteComp.IWebSessionValues.
	IWebSetActionName (see page 1114)	This is class SiteComp.IWebSetActionName.
	IWebSetFieldName (see page 1115)	This is class SiteComp.IWebSetFieldName.
	IWebUserList (see page 1116)	IWebUserList is the interface for validating user logins and access rights.
	IWebValueIsEqual (see page 1118)	This is class SiteComp.IWebValueIsEqual.

Structs, Records, Enums

	Name	Description
	TAdapterActionHTMLElementType (see page 1121)	TAdapterActionHTMLElementType identifies the HTML type that is generated to let a user execute an adapter action.
	TAdapterDisplayCharacteristic (see page 1122)	TAdapterDisplayCharacteristics (see page 1128) describes the characteristics of the HTML generated to display information from an adapter.
	TAdapterDisplayHTMLElementType (see page 1123)	TAdapterDisplayHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TAdapterDisplayViewModeType (see page 1123)	TAdapterDisplayViewModeType identifies how an HTML control for the associated adapter field should behave.
	TAdapterInputHTMLElementType (see page 1124)	TAdapterInputHTMLElementType identifies an HTML input type that can be used to edit the value(s) associated with an adapter field.
	TAdapterItemRequestParamOption (see page 1125)	TAdapterItemRequestParamOptions (see page 1128) identifies what parameter values to include when encoding the hidden fields and HREF values that represent calls to adapter fields and actions.

	TDispatchPageFlag (see page 1126)	TDiapatchPageFlags indicates the attributes that must be present on a Web page module for it to handle a request.
	TQualifyOption (see page 1126)	This is record SiteComp.TQualifyOption.

Types

Name	Description
TAdapterDisplayCharacteristics (see page 1128)	This is type SiteComp.TAdapterDisplayCharacteristics.
TAdapterItemRequestParamOptions (see page 1128)	This is type SiteComp.TAdapterItemRequestParamOptions.
TDispatchPageFlags (see page 1129)	This is type SiteComp.TDispatchPageFlags.
TSessionID (see page 1129)	TSessionID is the type of the session identifier in a WebSnap application.

Variables

Name	Description
ExecuteScript (see page 1130)	This is variable SiteComp.ExecuteScript.
GetHTMLSampleImage (see page 1130)	This is variable SiteComp.GetHTMLSampleImage.
IDEFileManager (see page 1131)	This is variable SiteComp.IDEFileManager.
InetFileTypes (see page 1131)	This is variable SiteComp.InetFileTypes.
InternetEnvOptions (see page 1131)	This is variable SiteComp.InternetEnvOptions.
ScriptDebugger (see page 1132)	This is variable SiteComp.ScriptDebugger.

11.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractDesigntimeWarnings (see page 963)	TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.
	TAbstractInetFileType (see page 965)	This is class SiteComp.TAbstractInetFileType.
	TNotifyList (see page 968)	TNotifyList is a collection of objects that should be notified of changes.
	TScriptComponent (see page 973)	This is class SiteComp.TScriptComponent.
	TScriptObject (see page 976)	This is class SiteComp.TScriptObject.

11.1.1 TAbstractDesigntimeWarnings Class

TAbstractDesigntimeWarnings assembles design-time warning messages that appear in the HTML view of the form your application builds.

Class Hierarchy

Pascal

```
TAbstractDesigntimeWarnings = class;
```

C++

```
class TAbstractDesigntimeWarnings;
```

File

SiteComp

Description

TAbstractDesigntimeWarnings is the base class for objects that collect the warning messages for an adapter page producer. At design time, as you add Web items to an adapter page producer, the page producer and the Web items you add to it write warnings to a TAbstractDesigntimeWarnings object describing any problems you can rectify (such as properties that need to be set or components that need to be added.) These warning messages appear in the preview of the emerging HTML form on the HTML view page of the editor.

See Also

TAdapterPageProducer (see page 1136)

GetDesigntimeWarnings

11.1.1.1 TAbstractDesigntimeWarnings Members

The following tables list the members exposed by TAbstractDesigntimeWarnings.

Public Methods

	Name	Description
	AddObject (see page 964)	Requests any design time warnings from a specified object.
	AddString (see page 964)	Adds a warning string to the set of warnings that appear at design time.

11.1.1.2 TAbstractDesigntimeWarnings Methods

The methods of the TAbstractDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	AddObject (see page 964)	Requests any design time warnings from a specified object.
	AddString (see page 964)	Adds a warning string to the set of warnings that appear at design time.

11.1.1.2.1 TAbstractDesigntimeWarnings.AddObject Method

Requests any design time warnings from a specified object.

Pascal

```
procedure AddObject(AObject: TObject); virtual; abstract;
```

C++

```
virtual __fastcall AddObject(TObject * AObject) = 0;
```

Description

AddObject collects the design time warnings from the specified object by calling its GetDesigntimeWarnings method (if it has one). If the object has already been added by a previous call to AddObject, AddObject detects this and does nothing.

AObject is the Web item from which to request design-time warnings.

See Also

GetDesigntimeWarnings (see page 6384)

AddString (see page 964)

11.1.1.2.2 TAbstractDesigntimeWarnings.AddString Method

Adds a warning string to the set of warnings that appear at design time.

Pascal

```
procedure AddString(AWarning: string); virtual; abstract;
```

C++

```
virtual __fastcall AddString(UnicodeString AWarning) = 0;
```

Description

AddString adds a string to the design-time warnings object. The adapter page producer displays all such warning strings in the HTML form that appears in the HTML view of the editor.

AWarning is a warning string that describes a problem that the developer can rectify (such as an invalid property value or missing component).

See Also

TAdapterPageProducer (see page 1136)

AddObject (see page 964)

11.1.2 TAbstractInetFileType Class

Class Hierarchy

Pascal

```
TAbstractInetFileType = class(TObject);
```

C++

```
class TAbstractInetFileType : public TObject;
```

File

SiteComp

Description

This is class SiteComp.TAbstractInetFileType.

11.1.2.1 TAbstractInetFileType Members

The following tables list the members exposed by TAbstractInetFileType.

Protected Methods

	Name	Description
	GetExtCount (see page 966)	This is GetExtCount, a member of class TAbstractInetFileType.
	GetFileExt (see page 966)	This is GetFileExt, a member of class TAbstractInetFileType.
	GetTypeID (see page 967)	This is GetTypeID, a member of class TAbstractInetFileType.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.

	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	FileExt (see page 967)	This is FileExt, a member of class TAbstractInetFileType.
R	FileExtCount (see page 967)	This is FileExtCount, a member of class TAbstractInetFileType.
R	TypeID (see page 968)	This is TypeID, a member of class TAbstractInetFileType.

11.1.2.2 TAbstractInetFileType Methods

The methods of the TAbstractInetFileType class are listed here.

Protected Methods

	Name	Description
A	GetExtCount (see page 966)	This is GetExtCount, a member of class TAbstractInetFileType.
A	GetFileExt (see page 966)	This is GetFileExt, a member of class TAbstractInetFileType.
A	GetTypeID (see page 967)	This is GetTypeID, a member of class TAbstractInetFileType.

11.1.2.2.1 TAbstractInetFileType.GetExtCount Method

Pascal

```
function GetExtCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetExtCount() = 0;
```

Description

This is GetExtCount, a member of class TAbstractInetFileType.

11.1.2.2.2 TAbstractInetFileType.GetFileExt Method

Pascal

```
function GetFileExt(I: Integer): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetFileExt(int I) = 0;
```

Description

This is GetFileExt, a member of class TAbstractInetFileType.

11.1.2.2.3 TAbstractInetFileType.GetTypeID Method

Pascal

```
function GetTypeID: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetTypeID() = 0;
```

Description

This is GetTypeID, a member of class TAbstractInetFileType.

11.1.2.3 TAbstractInetFileType Properties

The properties of the TAbstractInetFileType class are listed here.

Public Properties

	Name	Description
	FileExt (see page 967)	This is FileExt, a member of class TAbstractInetFileType.
	FileExtCount (see page 967)	This is FileExtCount, a member of class TAbstractInetFileType.
	TypeID (see page 968)	This is TypeID, a member of class TAbstractInetFileType.

11.1.2.3.1 TAbstractInetFileType.FileExt Property

Pascal

```
property FileExt [I: Integer]: string;
```

C++

```
__property UnicodeString FileExt[int I];
```

Description

This is FileExt, a member of class TAbstractInetFileType.

11.1.2.3.2 TAbstractInetFileType.FileExtCount Property

Pascal

```
property FileExtCount: Integer;
```

C++

```
__property int FileExtCount;
```

Description

This is FileExtCount, a member of class TAbstractInetFileType.

11.1.2.3.3 TAbstractInetFileType.TypeID Property

Pascal

```
property TypeID: string;
```

C++

```
__property UnicodeString TypeID;
```

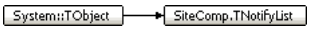
Description

This is TypeID, a member of class TAbstractInetFileType.

11.1.3 TNotifyList Class

TNotifyList is a collection of objects that should be notified of changes.

Class Hierarchy



Pascal

```
TNotifyList = class(TObject);
```

C++

```
class TNotifyList : public TObject;
```

File

SiteComp

Description

TNotifyList maintains a list of objects that have registered interest in change notifications. This class is used for a helper object on components that support the INotifyList (see page 1058) interface. When objects use the INotifyList (see page 1058) interface to register interest in change notifications, the component adds them to its TNotifyList instance. When those objects use the INotifyList (see page 1058) interface to unregister their interest, they are removed from the TNotifyList instance.

When the object that uses TNotifyList changes, it sends a notification to every object in the list. For example, when an adapter changes, it obtains an INotifyAdapterChange (see page 1057) interface from every object in the list. It uses this interface to call the object's NotifyAdapterChange method.

See Also

TCustomAdapter (see page 5531)

INotifyAdapterChange (see page 1057)

INotifyList (see page 1058)

11.1.3.1 TNotifyList Members

The following tables list the members exposed by TNotifyList.

Public Methods

	Name	Description
	AddNotify (see page 970)	Adds a new object to the list.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 969)	Creates an initializes an instance of TNotifyList (see page 968).
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 970)	Frees an instance of TNotifyList (see page 968).
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetCount (see page 971)	Returns the value of the Count (see page 972) property.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetItem (see page 971)	Returns the value of the Items (see page 972) property.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RemoveNotify (see page 971)	Removes an object from the list.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Count (see page 972)	Indicates the number of objects in the list.
R	Items (see page 972)	Lists the objects in this collection.

11.1.3.2 TNotifyList.Create Constructor

Creates an initializes an instance of TNotifyList (see page 968).

Pascal

```
constructor Create;
```

C++

```
__fastcall TNotifyList();
```

Description

WebSnap applications should not need to create instances of TNotifyList (see page 968). These objects are created by the component that uses them to keep track of objects that have registered an interest in change notifications.

See Also

NotifyList (see page 5574)

INotifyList (see page 1058)

11.1.3.3 TNotifyList.Destroy Destructor

Frees an instance of TNotifyList (see page 968).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNotifyList();
```

Description

WebSnap applications should not need to free instances of TNotifyList (see page 968), as this is handled by the components that use them. If you must free a TNotifyList (see page 968) instance, use the Free method.

See Also

System::TObject::Free (see page 3372)

11.1.3.4 TNotifyList Methods

The methods of the TNotifyList class are listed here.

Public Methods

	Name	Description
	AddNotify (see page 970)	Adds a new object to the list.
	GetCount (see page 971)	Returns the value of the Count (see page 972) property.
	GetItem (see page 971)	Returns the value of the Items (see page 972) property.
	RemoveNotify (see page 971)	Removes an object from the list.

11.1.3.4.1 TNotifyList.AddNotify Method

Adds a new object to the list.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to add an object to the end of the Items (see page 972) property array. This method is called to add objects that

want to be informed of changes to the component that uses this list.

ANotify is the object to add. This object should implement an interface that allows the associated component to inform it of changes. For example, objects in the TNotifyList (see page 968) collection maintained by an adapter support the INotifyAdapterChange (see page 1057) interface.

See Also

Items (see page 972)

RemoveNotify (see page 971)

INotifyAdapterChange (see page 1057)

11.1.3.4.2 TNotifyList.GetCount Method

Returns the value of the Count (see page 972) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 972) property. It returns the number of objects in the list.

See Also

Count (see page 972)

11.1.3.4.3 TNotifyList.GetItem Method

Returns the value of the Items (see page 972) property.

Pascal

```
function GetItem(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetItem(int Index);
```

Description

GetItem is the read implementation of the Items (see page 972) property.

Index is the zero-offset index of the item to return.

GetItem returns the specified item from the list.

See Also

Items (see page 972)

11.1.3.4.4 TNotifyList.RemoveNotify Method

Removes an object from the list.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to remove an object from the list. This method is called to remove objects that are no longer interested in changes to the associated component.

ANotify is the object to remove. This object must have been previously added by a call to the AddNotify (see page 970) method.

See Also

Items (see page 972)

AddNotify (see page 970)

11.1.3.5 TNotifyList Properties

The properties of the TNotifyList class are listed here.

Public Properties

	Name	Description
	Count (see page 972)	Indicates the number of objects in the list.
	Items (see page 972)	Lists the objects in this collection.

11.1.3.5.1 TNotifyList.Count Property

Indicates the number of objects in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Count indicates the number of objects in the Items (see page 972) property array. Use this value as an upper bound for the index when iterating over all the objects in the list.

See Also

Items (see page 972)

11.1.3.5.2 TNotifyList.Items Property

Lists the objects in this collection.

Pascal

```
property Items [Index: Integer]: TObject;
```

C++

```
__property TObject * Items[int Index];
```

Description

Use Items to reference an object in the list. Each item is an object that supports an interface that allows the associated

component to inform it of changes. For example, objects in the TNotifyList (see page 968) collection maintained by an adapter support the INotifyAdapterChange (see page 1057) interface.

Index is the index of the object, where 0 specifies the first item, 1 specifies the second item, and so on. The Count (see page 972) property gives an upper bound for this index.

Note: In Delphi, Items is the default property of TNotifyList (see page 968). That means the property name may be omitted in code.

See Also

Count (see page 972)

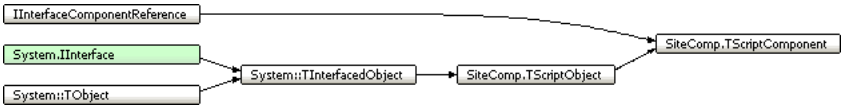
AddNotify (see page 970)

RemoveNotify (see page 971)

INotifyAdapterChange (see page 1057)

11.1.4 TScriptComponent Class

Class Hierarchy



Pascal

```
TScriptComponent = class(TScriptObject, IInterfaceComponentReference);
```

C++

```
class TScriptComponent : public TScriptObject, public IInterfaceComponentReference;
```

File

SiteComp

Description

This is class SiteComp.TScriptComponent.

11.1.4.1 TScriptComponent Members

The following tables list the members exposed by TScriptComponent.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetComponent (see page 975)	Returns a reference to the component associated by the implementing class.

	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 975)	This is Create, a member of class TScriptComponent.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Obj (see page 975)	This is Obj, a member of class TScriptComponent.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

11.1.4.2 TScriptComponent.Create Constructor

Pascal

```
constructor Create(AObject: TComponent); virtual;
```

C++

```
virtual __fastcall TScriptComponent(TComponent * AObject);
```

Description

This is Create, a member of class TScriptComponent.

11.1.4.3 TScriptComponent Methods

The methods of the TScriptComponent class are listed here.

Protected Methods

	Name	Description
	GetComponent (see page 975)	Returns a reference to the component associated by the implementing class.

11.1.4.3.1 TScriptComponent.GetComponent Method

Returns a reference to the component associated by the implementing class.

Pascal

```
function GetComponent: TComponent;
```

C++

```
__fastcall TComponent * GetComponent();
```

Description

GetComponent returns a TComponent value that is associated with the implementing class. For example, the TComponent class itself implements the GetComponent to return a pointer to itself.

See Also

TComponent

11.1.4.4 TScriptComponent Properties

The properties of the TScriptComponent class are listed here.

Public Properties

	Name	Description
R	Obj (see page 975)	This is Obj, a member of class TScriptComponent.

11.1.4.4.1 TScriptComponent.Obj Property

Pascal

```
property Obj: TComponent;
```

C++

```
__property TComponent * Obj;
```

Description

This is Obj, a member of class TScriptComponent.

11.1.5 TScriptObject Class

Class Hierarchy

Pascal

```
TScriptObject = class(TInterfacedObject);
```

C++

```
class TScriptObject : public TInterfacedObject;
```

File

SiteComp

Description

This is class SiteComp.TScriptObject.

11.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

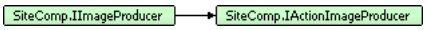
	Name	Description
	IActionImageProducer (see page 978)	IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.
	IAdapterAccess (see page 981)	This is class SiteComp.IAdapterAccess.
	IAdapterActionAccess (see page 982)	This is class SiteComp.IAdapterActionAccess.
	IAdapterActionAttributes (see page 983)	This is class SiteComp.IAdapterActionAttributes.
	IAdapterDispatcher (see page 985)	IAdapterDispatcher is the interface for interacting with the adapter dispatcher.
	IAdapterDispatchParams (see page 987)	IAdapterDispatchParams describes adapter action parameters and adapter field values.
	IAdapterError (see page 992)	IAdapterError is the interface for retrieving adapter error information.
	IAdapterErrors (see page 995)	IAdapterErrors is implemented by adapter errors objects.
	IAdapterFieldAccess (see page 998)	This is class SiteComp.IAdapterFieldAccess.
	IAdapterFieldAttributes (see page 999)	This is class SiteComp.IAdapterFieldAttributes.
	IAdapterHiddenField (see page 1001)	This is class SiteComp.IAdapterHiddenField.
	IAdapterHiddenFields (see page 1003)	This is class SiteComp.IAdapterHiddenFields.
	IAdapterItemRequestParams (see page 1005)	IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.
	IAdapterMode (see page 1009)	This is class SiteComp.IAdapterMode.
	IAdapterRequestHandler (see page 1012)	IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.
	IAsFormatted (see page 1013)	This is class SiteComp.IAsFormatted.
	IDefaultPageFileName (see page 1014)	This is class SiteComp.IDefaultPageFileName.
	IFieldValuesAdapter (see page 1015)	This is class SiteComp.IFieldValuesAdapter.
	IFindIncludeFile (see page 1016)	This is class SiteComp.IFindIncludeFile.
	IGetAdapterActions (see page 1017)	This is class SiteComp.IGetAdapterActions.
	IGetAdapterDisplayCharacteristics (see page 1018)	This is class SiteComp.IGetAdapterDisplayCharacteristics.
	IGetAdapterErrors (see page 1019)	This is class SiteComp.IGetAdapterErrors.
	IGetAdapterFields (see page 1020)	This is class SiteComp.IGetAdapterFields.
	IGetAdapterHiddenFields (see page 1021)	This is class SiteComp.IGetAdapterHiddenFields.
	IGetAdapterImage (see page 1022)	This is class SiteComp.IGetAdapterImage.
	IGetAdapterItemRequestParams (see page 1023)	This is class SiteComp.IGetAdapterItemRequestParams.
	IGetAdapterRequestDefaultResponse (see page 1024)	This is class SiteComp.IGetAdapterRequestDefaultResponse.
	IGetAdapterValuesList (see page 1026)	This is class SiteComp.IGetAdapterValuesList.
	IGetDefaultAction (see page 1027)	This is class SiteComp.IGetDefaultAction.
	IGetDesigntimeWarnings (see page 1029)	This is class SiteComp.IGetDesigntimeWarnings.
	IGetFieldValuesAdapter (see page 1030)	This is class SiteComp.IGetFieldValuesAdapter.
	IGetHTMLStyle (see page 1031)	IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.
	IGetProducerComponent (see page 1034)	This is class SiteComp.IGetProducerComponent.
	IGetScriptObject (see page 1035)	This is class SiteComp.IGetScriptObject.
	IGetScriptObjectClassKey (see page 1036)	This is class SiteComp.IGetScriptObjectClassKey.
	IGetWebAppComponents (see page 1037)	This is class SiteComp.IGetWebAppComponents.
	IIDEFileManager (see page 1040)	This is class SiteComp.IIDEFileManager.
	IImageProducer (see page 1041)	IImageProducer is the interface for accessing the image that an adapter component supplies.
	IInternetEnvOptions (see page 1042)	This is class SiteComp.IInternetEnvOptions.
	IInternetExecuteScript (see page 1045)	This is class SiteComp.IInternetExecuteScript.

	IInternetFileTypes (see page 1047)	This is class SiteComp.IInternetFileTypes.
	IInternetScriptDebugger (see page 1048)	This is class SiteComp.IInternetScriptDebugger.
	IIsAdapterActionList (see page 1049)	This is class SiteComp.IIsAdapterActionList.
	IIterateIntfSupport (see page 1050)	This is class SiteComp.IIterateIntfSupport.
	IIterateObjectSupport (see page 1051)	This is class SiteComp.IIterateObjectSupport.
	IIteratorIndex (see page 1052)	IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.
	IIteratorSupport (see page 1054)	IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.
	INotifyAdapterChange (see page 1057)	INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.
	INotifyList (see page 1058)	INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.
	IPageDispatcher (see page 1060)	IPageDispatcher is the interface for interacting with the page dispatcher.
	IPageResult (see page 1064)	This is class SiteComp.IPageResult.
	ISessionsService (see page 1066)	ISessionsService is the interface for controlling a TSessionsService object.
	ISessionsServiceConnectID (see page 1068)	This is class SiteComp.ISessionsServiceConnectID.
	ISessionsServicePersist (see page 1070)	This is class SiteComp.ISessionsServicePersist.
	ISessionsServiceStoreID (see page 1071)	This is class SiteComp.ISessionsServiceStoreID.
	IValuesListAdapter (see page 1072)	IValuesListAdapter is the interface for supplying name/value pairs.
	IWebActionsList (see page 1076)	This is class SiteComp.IWebActionsList.
	IWebApplicationInfo (see page 1078)	IWebApplicationInfo is the interface for the application adapter.
	IWebContainerSubComponent (see page 1081)	This is class SiteComp.IWebContainerSubComponent.
	IWebDataFields (see page 1081)	This is class SiteComp.IWebDataFields.
	IWebEnabled (see page 1083)	This is class SiteComp.IWebEnabled.
	IWebFormComponentContainer (see page 1084)	This is class SiteComp.IWebFormComponentContainer.
	IWebGetActionName (see page 1086)	This is class SiteComp.IWebGetActionName.
	IWebGetDisplayLabel (see page 1088)	This is class SiteComp.IWebGetDisplayLabel.
	IWebGetDisplayWidth (see page 1089)	This is class SiteComp.IWebGetDisplayWidth.
	IWebGetFieldName (see page 1090)	This is class SiteComp.IWebGetFieldName.
	IWebGetFieldValue (see page 1091)	This is class SiteComp.IWebGetFieldValue.
	IWebGetFieldValues (see page 1092)	IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.
	IWebGetMaxLength (see page 1095)	This is class SiteComp.IWebGetMaxLength.
	IWebGetSessionIDs (see page 1096)	This is class SiteComp.IWebGetSessionIDs.
	IWebImageHREF (see page 1097)	IWebImageHREF is the interface for obtaining an HREF that addresses an image.
	IWebInputName (see page 1098)	This is class SiteComp.IWebInputName.
	IWebIsDefaultAction (see page 1100)	This is class SiteComp.IWebIsDefaultAction.
	IWebIsDefaultField (see page 1101)	This is class SiteComp.IWebIsDefaultField.
	IWebSession (see page 1102)	IWebSession allows an application to terminate a Web session.
	IWebSessionAttributes (see page 1103)	This is class SiteComp.IWebSessionAttributes.
	IWebSessionID (see page 1108)	This is class SiteComp.IWebSessionID.
	IWebSessionIDs (see page 1109)	IWebSessionIDs lists the session IDs (see page 1111) for all sessions that TSessionsService stores in memory.
	IWebSessionValues (see page 1112)	This is class SiteComp.IWebSessionValues.
	IWebSetActionName (see page 1114)	This is class SiteComp.IWebSetActionName.
	IWebSetFieldName (see page 1115)	This is class SiteComp.IWebSetFieldName.
	IWebUserList (see page 1116)	IWebUserList is the interface for validating user logins and access rights.
	IWebValueIsEqual (see page 1118)	This is class SiteComp.IWebValueIsEqual.

11.2.1 IActionImageProducer Interface

IActionImageProducer is the interface for accessing the image or set of images that represent an adapter action.

Class Hierarchy



Pascal

```
IActionImageProducer = interface(IImageProducer);
```

C++

```
__interface IActionImageProducer : public IImageProducer;
```

File

SiteComp

Description

IActionImageProducer is the interface used to access the object that supplies images for an adapter action. This interface can supply a single image for the action, or a set of related images that can be used when an HTML control changes its appearance in response to user actions.

IActionImageProducer does not directly supply the HREF for the image or images it represents. Instead, it supplies the interface for another object. That object may generate the image, or it may simply generate an HREF that addresses the object that generates the image. This extra level of indirection allows the object that implements IActionImageProducer to work with multiple adapter fields that produce a variety of images.

Note: Method declarations that use IActionImageProducer use the _di_IActionImageProducer type instead. This type is a DelphiInterface (see page 3256) wrapper around the IActionImageProducer interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IActionImageProducer >
_di_IActionImageProducer;
```

See Also

TAdapterActionButton (see page 6316)

TCustomAdapterActionButton (see page 6430)

11.2.1.1 IActionImageProducer Members

The following tables list the members exposed by IActionImageProducer.

Public Methods

	Name	Description
	GetAdapterEventImage (see page 980)	Returns the object that is responsible for providing a specified image from a set of related images.
	GetAdapterImage (see page 1042)	Returns the object that is responsible for providing a specified image.
	GetDisplayStyle (see page 980)	Indicates what type of images this interface can be used to access.

11.2.1.2 IActionImageProducer Methods

The methods of the IActionImageProducer class are listed here.

Public Methods

	Name	Description
	GetAdapterEventImage (see page 980)	Returns the object that is responsible for providing a specified image from a set of related images.

	GetDisplayStyle (see page 980)	Indicates what type of images this interface can be used to access.

11.2.1.2.1 IActionImageProducer.GetAdapterEventImage Method

Returns the object that is responsible for providing a specified image from a set of related images.

Pascal

```
function GetAdapterEventImage(const AComponentName: WideString; const AEvent: WideString;
const ACaption: WideString): TObject;
```

C++

```
__fastcall TObject * GetAdapterEventImage(const BSTR AComponentName, const BSTR AEvent, const
BSTR ACaption);
```

Description

GetAdapterEventImage returns the object that supplies an HREF for the specified image. That object can supply the HREF in one of two ways:

It can implement the IWebImageHREF (see page 1097) interface.

It can be the object that produces the image in response to request messages. In this case, the adapter dispatcher's EncodeAdapterItemRequestAsHREF method generates the HREF to address this object.

AComponentName is the name of the component that is requesting the image. GetAdapterEventImage can use this parameter to determine which of several objects is appropriate for supplying the image.

AEvent identifies the user event that the image should reflect. This parameter is used when the GetDisplayStyle (see page 980) method returns 'EventImages'. If AEvent is an empty string, GetAdapterEventImage returns the object that supplies the base image. If AEvent is not empty, GetAdapterEventImage returns the object that supplies the version of the image to use in response to the associated user action. Possible values are 'onmouseover', 'onmouseout', and 'onmousedown'.

ACaption is a text label that describes the desired image. GetAdapterEventImage can use this parameter to determine which of several objects is appropriate for supplying the image.

See Also

IWebImageHREF (see page 1097)

EncodeAdapterItemRequestAsHREF (see page 6173)

GetAdapterImage (see page 1042)

GetDisplayStyle (see page 980)

11.2.1.2.2 IActionImageProducer.GetDisplayStyle Method

Indicates what type of images this interface can be used to access.

Pascal

```
function GetDisplayStyle(Sender: TComponent): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(TComponent * Sender);
```

Description

GetDisplayStyle returns either 'Image' or 'EventImages', depending on whether it can supply a single image for an adapter action and caption, or multiple images that reflect various user events.

If GetDisplayStyle returns 'Image', use the GetAdapterImage (see page 1042) method to obtain an interface for the image

object. If GetDisplayStyle returns 'EventImages', use the GetAdapterEventImage (see page 980) method to obtain an interface for the image object.

When a TAdapterActionButton component uses this interface for its ImageProducer property, its DisplayType property should be compatible with the display style this method returns. That is, if this method returns 'Image', the adapter action button should have a DisplayType of ctImage or ctDefault. If this method returns 'EventImages', the adapter action button can have a DisplayType of ctImage, ctEventImages, or ctDefault.

Sender is the component that is requesting the image.

See Also

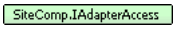
DisplayType (see page 6439)

GetAdapterEventImage (see page 980)

GetAdapterImage (see page 1042)

11.2.2 IAdapterAccess Interface

Class Hierarchy



Pascal

```
IAdapterAccess = interface;
```

C++

```
__interface IAdapterAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterAccess.

11.2.2.1 IAdapterAccess Members

The following tables list the members exposed by IAdapterAccess.

Public Methods

	Name	Description
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.

11.2.2.2 IAdapterAccess Methods

The methods of the IAdapterAccess class are listed here.

Public Methods

	Name	Description
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.

11.2.2.2.1 IAdapterAccess.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class IAdapterAccess.

11.2.2.2.2 IAdapterAccess.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class IAdapterAccess.

11.2.2.2.3 IAdapterAccess.HasViewAccess Method

Pascal

```
function HasViewAccess: Boolean;
```

C++

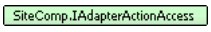
```
__fastcall Boolean HasViewAccess();
```

Description

This is HasViewAccess, a member of class IAdapterAccess.

11.2.3 IAdapterActionAccess Interface

Class Hierarchy



Pascal

```
IAdapterActionAccess = interface;
```

C++

```
__interface IAdapterActionAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterActionAccess.

11.2.3.1 IAdapterActionAccess Members

The following tables list the members exposed by IAdapterActionAccess.

Public Methods

	Name	Description
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

11.2.3.2 IAdapterActionAccess Methods

The methods of the IAdapterActionAccess class are listed here.

Public Methods

	Name	Description
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

11.2.3.2.1 IAdapterActionAccess.HasExecuteAccess Method

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

This is HasExecuteAccess, a member of class IAdapterActionAccess.

11.2.4 IAdapterActionAttributes Interface

Class Hierarchy

SiteComp.IAdapterActionAttributes

Pascal

```
IAdapterActionAttributes = interface;
```

C++

```
__interface IAdapterActionAttributes;
```

File

SiteComp

Description

This is class SiteComp.IAdapterActionAttributes.

11.2.4.1 IAdapterActionAttributes Members

The following tables list the members exposed by IAdapterActionAttributes.

Public Methods

	Name	Description
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.

Public Properties

	Name	Description
	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

11.2.4.2 IAdapterActionAttributes Methods

The methods of the IAdapterActionAttributes class are listed here.

Public Methods

	Name	Description
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.

11.2.4.2.1 IAdapterActionAttributes.GetVisible Method

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

This is GetVisible, a member of class IAdapterActionAttributes.

11.2.4.3 IAdapterActionAttributes Properties

The properties of the IAdapterActionAttributes class are listed here.

Public Properties

	Name	Description
	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

11.2.4.3.1 IAdapterActionAttributes.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

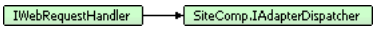
Description

This is Visible, a member of class IAdapterActionAttributes.

11.2.5 IAdapterDispatcher Interface

IAdapterDispatcher is the interface for interacting with the adapter dispatcher.

Class Hierarchy



Pascal

```
IAdapterDispatcher = interface(IWebRequestHandler);
```

C++

```
__interface IAdapterDispatcher : public IWebRequestHandler;
```

File

SiteComp

Description

The TAdapterDispatcher component implements IAdapterDispatcher. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to dispatch request messages that represent HTML form submissions and requests for dynamic images that can be resolved using adapter actions and fields. It also encodes the hidden fields and HREF values that are used to make those calls to adapter fields and actions.

Note: In C++, method declarations that use IAdapterDispatcher use the _di_IAdapterDispatcher type instead. This type is a DelphiInterface (see page 3256) wrapper around the IAdapterDispatcher interface:

Note: typedef System::DelphiInterface (see page 3256)< IAdapterDispatcher > _di_IAdapterDispatcher;

See Also

TAdapterDispatcher (see page 6148)

TCustomWebAppComponents (see page 6201)

TWebAppComponents (see page 6243)

11.2.5.1 IAdapterDispatcher Members

The following tables list the members exposed by IAdapterDispatcher.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 986)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 986)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 987)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

11.2.5.2 IAdapterDispatcher Methods

The methods of the IAdapterDispatcher class are listed here.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 986)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 986)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 987)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

11.2.5.2.1 IAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue Method

Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.

Pascal

```
function EncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Call EncodeAdapterItemRequestAsFieldValue to encode the adapter action or image field parameters as they should appear as field values in form submissions. That is, the resulting parameter string has a form such as '?param1=value1&param2=value2'.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 987)

EncodeAdapterItemRequestAsHREF (see page 986)

11.2.5.2.2 IAdapterDispatcher.EncodeAdapterItemRequestAsHREF Method

Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.

Pascal

```
function EncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Call EncodeAdapterItemRequestAsHREF to encode the adapter action or image field parameters as they should appear as parameters to an anchor tag in an HTML form.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 987)

EncodeAdapterItemRequestAsFieldValue (see page 986)

11.2.5.2.3 IAdapterDispatcher.GetAdapterItemRequestParams Method

Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Pascal

```
procedure GetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; AParams: IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(IUnknown AHandler, TAdapterItemRequestParamOptions
AOptions, IAdapterItemRequestParams AParams);
```

Description

Call GetAdapterItemRequestParams to retrieve the adapter action and image field parameters.

AHandler is the interface of the adapter whose action or field requires the parameters.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. GetAdapterItemRequestParams uses this interface to write the parameter values.

See Also

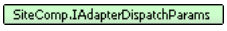
EncodeAdapterItemRequestAsHREF (see page 986)

EncodeAdapterItemRequestAsFieldValue (see page 986)

11.2.6 IAdapterDispatchParams Interface

IAdapterDispatchParams describes adapter action parameters and adapter field values.

Class Hierarchy



Pascal

```
IAdapterDispatchParams = interface;
```

C++

```
__interface IAdapterDispatchParams;
```

File

SiteComp

Description

Use IAdapterDispatchParams to obtain context information that is needed to execute an adapter action or to obtain the value of an image field from an adapter. When server-side script in a WebSnap application executes an adapter action or requests the value of an image field, an object that implements this interface is created to store information about the adapter that can perform the specified task and the parameters that must be passed to the adapter for it to do so. This interface is then passed to the adapter action's HandleRequest method so that it can look up the adapter and the parameters.

Note: In C++, method declarations that use IAdapterDispatchParams use the _di_IAdapterDispatchParams type instead. This type is a DelphiInterface (see page 3256) wrapper around the IAdapterDispatchParams interface:

Note: typedef System::DelphiInterface (see page 3256)< IAdapterDispatchParams > _di_IAdapterDispatchParams;

See Also

IAdapterRequestHandler (see page 1012)

TAdapterDispatchParams (see page 6153)

TBaseAdapterAction (see page 5447)

11.2.6.1 IAdapterDispatchParams Members

The following tables list the members exposed by IAdapterDispatchParams.

Public Methods

	Name	Description
	GetHandler (see page 989)	Returns the value of the Handler (see page 990) property.
	GetObjectIdentifier (see page 989)	Returns the value of the ObjectIdentifier (see page 990) property.
	GetParams (see page 989)	Returns the value of the Params (see page 991) property.
	GetRequestIdentifier (see page 989)	Returns the value of the RequestIdentifier (see page 991) property.

Public Properties

	Name	Description
R	Handler (see page 990)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 990)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 991)	Indicates the parameter values to use when executing the action or looking up the image.
R	RequestIdentifier (see page 991)	Identifies the type of the request identified by the ObjectIdentifier (see page 990) property.

11.2.6.2 IAdapterDispatchParams Methods

The methods of the IAdapterDispatchParams class are listed here.

Public Methods

	Name	Description
	GetHandler (see page 989)	Returns the value of the Handler (see page 990) property.
	GetObjectIdentifier (see page 989)	Returns the value of the ObjectIdentifier (see page 990) property.
	GetParams (see page 989)	Returns the value of the Params (see page 991) property.

	GetRequestIdentifier (see page 989)	Returns the value of the RequestIdentifier (see page 991) property.

11.2.6.2.1 IAdapterDispatchParams.GetHandler Method

Returns the value of the Handler (see page 990) property.

Pascal

```
function GetHandler: TObject;
```

C++

```
__fastcall TObject * GetHandler();
```

Description

GetHandler is the read implementation of the Handler (see page 990) property.

See Also

Handler (see page 990)

11.2.6.2.2 IAdapterDispatchParams.GetObjectIdentifier Method

Returns the value of the ObjectIdentifier (see page 990) property.

Pascal

```
function GetObjectIdentifier: string;
```

C++

```
__fastcall UnicodeString GetObjectIdentifier();
```

Description

GetObjectIdentifier is the read implementation of the ObjectIdentifier (see page 990) property.

See Also

ObjectIdentifier (see page 990)

11.2.6.2.3 IAdapterDispatchParams.GetParams Method

Returns the value of the Params (see page 991) property.

Pascal

```
function GetParams: TStrings;
```

C++

```
__fastcall TStrings GetParams();
```

Description

GetParams is the read implementation of the Params (see page 991) property.

See Also

Params (see page 991)

11.2.6.2.4 IAdapterDispatchParams.GetRequestIdentifier Method

Returns the value of the RequestIdentifier (see page 991) property.

Pascal

```
function GetRequestIdentifier: string;
```

C++

```
__fastcall UnicodeString GetRequestIdentifier();
```

Description

GetRequestIdentifier is the read implementation of the RequestIdentifier (see page 991) property.

See Also

RequestIdentifier (see page 991)

11.2.6.3 IAdapterDispatchParams Properties

The properties of the IAdapterDispatchParams class are listed here.

Public Properties

	Name	Description
R	Handler (see page 990)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 990)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 991)	Indicates the parameter values to use when executing the action or looking up the image.
R	RequestIdentifier (see page 991)	Identifies the type of the request identified by the ObjectIdentifier (see page 990) property.

11.2.6.3.1 IAdapterDispatchParams.Handler Property

Indicates the adapter object that can handle the current request.

Pascal

```
property Handler: TObject;
```

C++

```
__property TObject * Handler;
```

Description

Read (see page 3634) Handler to access the adapter that can execute the action or supply the image that is requested.

See Also

ObjectIdentifier (see page 990)

RequestIdentifier (see page 6160)

11.2.6.3.2 IAdapterDispatchParams.ObjectIdentifier Property

Specifies the name of the action to execute or the field to retrieve.

Pascal

```
property ObjectIdentifier: string;
```

C++

```
__property UnicodeString ObjectIdentifier;
```

Description

Read (see page 3634) ObjectIdentifier to determine the name of the requested action or field. The interpretation of this value is the responsibility of the object specified by the Handler (see page 990) parameter.

See Also

Handler (see page 990)

RequestIdentifier (see page 6160)

11.2.6.3.3 IAdapterDispatchParams.Params Property

Indicates the parameter values to use when executing the action or looking up the image.

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Read (see page 3634) Params to determine the parameters that the adapter action can use to perform its task. Each string in this string list has the form Name=Value, where Name is the name of a parameter and Value is its value.

The interpretation of the parameter values is up to the adapter specified by the Handler (see page 990) object or to the adapter action object that handles the request.

See Also

Handler (see page 990)

ObjectIdentifier (see page 6159)

11.2.6.3.4 IAdapterDispatchParams.RequestIdentifier Property

Identifies the type of the request identified by the ObjectIdentifier (see page 990) property.

Pascal

```
property RequestIdentifier: string;
```

C++

```
__property UnicodeString RequestIdentifier;
```

Description

Read (see page 3634) RequestIdentifier to determine the type of processing the current request requires. For example, the RequestIdentifier may identify the request as a request for the value of an image field or a request to execute an action.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 990) parameter.

See Also

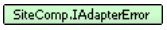
Handler (see page 990)

ObjectIdentifier (see page 6159)

11.2.7 IAdapterError Interface

IAdapterError is the interface for retrieving adapter error information.

Class Hierarchy



Pascal

```
IAdapterError = interface;
```

C++

```
__interface IAdapterError;
```

File

SiteComp

Description

The TAdapterError class implements IAdapterError, which is managed by TBaseAdapterErrorsList and its descendant classes. Applications uses IAdapterError to retrieve information relating to an adapter error.

See Also

TAdapterError (see page 5256)

TBaseAdapterErrorsList (see page 5479)

11.2.7.1 IAdapterError Members

The following tables list the members exposed by IAdapterError.

Public Methods

	Name	Description
	GetErrorText (see page 993)	Returns the value of the ErrorText (see page 994) property.
	GetField (see page 993)	Returns the value of the Field (see page 994) property.
	GetID (see page 993)	Returns the value of the ID (see page 994) property.

Public Properties

	Name	Description
R	ErrorText (see page 994)	Returns text describing the error.
R	Field (see page 994)	Returns text describing the error.
R	ID (see page 994)	Returns error ID.

11.2.7.2 IAdapterError Methods

The methods of the IAdapterError class are listed here.

Public Methods

	Name	Description
	GetErrorText (see page 993)	Returns the value of the ErrorText (see page 994) property.
	GetField (see page 993)	Returns the value of the Field (see page 994) property.

	GetID (see page 993)	Returns the value of the ID (see page 994) property.

11.2.7.2.1 IAdapterError.GetErrorText Method

Returns the value of the ErrorText (see page 994) property.

Pascal

```
function GetErrorText: string;
```

C++

```
__fastcall UnicodeString GetErrorText();
```

Description

GetErrorText is the read implementation of the ErrorText (see page 994) property.

See Also

ErrorText (see page 994)

11.2.7.2.2 IAdapterError.GetField Method

Returns the value of the Field (see page 994) property.

Pascal

```
function GetField: TComponent;
```

C++

```
__fastcall TComponent * GetField();
```

Description

GetField is the read implementation of the Field (see page 994) property.

See Also

Field (see page 994)

11.2.7.2.3 IAdapterError.GetID Method

Returns the value of the ID (see page 994) property.

Pascal

```
function GetID: integer;
```

C++

```
__fastcall int GetID();
```

Description

GetID is the read implementation of the ID (see page 994) property.

See Also

ID (see page 994)

11.2.7.3 IAdapterError Properties

The properties of the IAdapterError class are listed here.

Public Properties

	Name	Description
	ErrorText (see page 994)	Returns text describing the error.
	Field (see page 994)	Returns text describing the error.
	ID (see page 994)	Returns error ID.

11.2.7.3.1 IAdapterError.ErrorText Property

Returns text describing the error.

Pascal

```
property ErrorText: string;
```

C++

```
__property UnicodeString ErrorText;
```

Description

Examine ErrorText to obtain text describing the error.

See Also

GetErrorText (see page 993)

11.2.7.3.2 IAdapterError.Field Property

Returns text describing the error.

Pascal

```
property Field: TComponent;
```

C++

```
__property TComponent * Field;
```

Description

Examine Field to obtain a reference to the field component involved in the error.

See Also

GetField (see page 993)

11.2.7.3.3 IAdapterError.ID Property

Returns error ID.

Pascal

```
property ID: integer;
```

C++

```
__property int ID;
```

Description

Examine ID to obtain the error ID value.

See Also

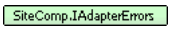
GetID (see page 993)

ID (see page 5178)

11.2.8 IAdapterErrors Interface

IAdapterErrors is implemented by adapter errors objects.

Class Hierarchy



Pascal

```
IAdapterErrors = interface;
```

C++

```
__interface IAdapterErrors;
```

File

SiteComp

Description

Adapter errors objects keep a list of errors detected while executing an adapter action or generating a page using an adapter. Adapter errors objects may be accessed with server-side script using the adapter's Errors (see page 998) property.

See Also

IAdapterError (see page 992)

TCustomAdapter (see page 5531)

11.2.8.1 IAdapterErrors Members

The following tables list the members exposed by IAdapterErrors.

Public Methods

	Name	Description
	ClearErrors (see page 996)	Clears the errors list.
	DefineLabel (see page 996)	Associates a label with all errors associated with a particular adapter field.
	GetError (see page 996)	Returns a particular error.
	GetErrorCount (see page 997)	Return the number of errors in the errors list.
	GetObjectErrors (see page 997)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 997)	Indicates whether the errors list contains objects associated with a particular adapter field.

Public Properties

	Name	Description
R	ErrorCount (see page 998)	Returns the number of errors in the errors list.
R	Errors (see page 998)	Returns an error from the list.

11.2.8.2 IAdapterErrors Methods

The methods of the IAdapterErrors class are listed here.

Public Methods

	Name	Description
	ClearErrors (see page 996)	Clears the errors list.
	DefineLabel (see page 996)	Associates a label with all errors associated with a particular adapter field.
	GetError (see page 996)	Returns a particular error.
	GetErrorCount (see page 997)	Return the number of errors in the errors list.
	GetObjectErrors (see page 997)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 997)	Indicates whether the errors list contains objects associated with a particular adapter field.

11.2.8.2.1 IAdapterErrors.ClearErrors Method

Clears the errors list.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

Clears the errors list.

11.2.8.2.2 IAdapterErrors.DefineLabel Method

Associates a label with all errors associated with a particular adapter field.

Pascal

```
procedure DefineLabel(const AName: string; const ALabel: string);
```

C++

```
__fastcall DefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

DefineLabel is responsible for replacing an label text in error messages associated with a particular adapter field with the given label. DefineLabel is used when the label used on the Web page is not the same as the label generated as part of the error message.

11.2.8.2.3 IAdapterErrors.GetError Method

Returns a particular error.

Pascal

```
function GetError(I: Integer): IAdapterError;
```

C++

```
__fastcall IAdapterError GetError(int I);
```

Description

I is a zero-based index.

See Also

IAdapterError (see page 992)

11.2.8.2.4 IAdapterErrors.GetErrorCount Method

Return the number of errors in the errors list.

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

Return the number of errors in the errors list.

11.2.8.2.5 IAdapterErrors.GetObjectErrors Method

Returns an object that is used to enumerate all errors associated with an adapter field.

Pascal

```
function GetObjectErrors(const AName: string): TObject;
```

C++

```
__fastcall TObject * GetObjectErrors(const UnicodeString AName);
```

Description

AName identifies an adapter field. nil (Delphi) or NULL (C++) is returned if there are no errors associated with the adapter field.

See Also

HasObjectErrors (see page 997)

11.2.8.2.6 IAdapterErrors.HasObjectErrors Method

Indicates whether the errors list contains objects associated with a particular adapter field.

Pascal

```
function HasObjectErrors(const AName: string): Boolean;
```

C++

```
__fastcall Boolean HasObjectErrors(const UnicodeString AName);
```

Description

HasObjectErrors returns true if any errors are associated with the given adapter field. AName identifies an adapter field.

See Also

GetObjectErrors (see page 997)

11.2.8.3 IAdapterErrors Properties

The properties of the IAdapterErrors class are listed here.

Public Properties

	Name	Description
	ErrorCount (see page 998)	Returns the number of errors in the errors list.

	Errors (see page 998)	Returns an error from the list.

11.2.8.3.1 IAdapterErrors.ErrorCount Property

Returns the number of errors in the errors list.

Pascal

```
property ErrorCount: Integer;
```

C++

```
__property int ErrorCount;
```

Description

ErrorCount returns the result of GetErrorCount (see page 997).

See Also

GetErrorCount (see page 997)

11.2.8.3.2 IAdapterErrors.Errors Property

Returns an error from the list.

Pascal

```
property Errors [I: Integer]: IAdapterError;
```

C++

```
__property IAdapterError Errors[int I];
```

Description

I is a zero-based index. Errors returns the result of GetError (see page 996).

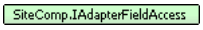
See Also

IAdapterError (see page 992)

GetError (see page 996)

11.2.9 IAdapterFieldAccess Interface

Class Hierarchy



Pascal

```
IAdapterFieldAccess = interface;
```

C++

```
__interface IAdapterFieldAccess;
```

File

SiteComp

Description

This is class SiteComp.IAdapterFieldAccess.

11.2.9.1 IAdapterFieldAccess Members

The following tables list the members exposed by IAdapterFieldAccess.

Public Methods

	Name	Description
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.

11.2.9.2 IAdapterFieldAccess Methods

The methods of the IAdapterFieldAccess class are listed here.

Public Methods

	Name	Description
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.

11.2.9.2.1 IAdapterFieldAccess.HasModifyAccess Method

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

This is HasModifyAccess, a member of class IAdapterFieldAccess.

11.2.9.2.2 IAdapterFieldAccess.HasViewAccess Method

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

This is HasViewAccess, a member of class IAdapterFieldAccess.

11.2.10 IAdapterFieldAttributes Interface

Class Hierarchy

SiteComp.IAdapterFieldAttributes

Pascal

```
IAdapterFieldAttributes = interface;
```

C++

```
__interface IAdapterFieldAttributes;
```

File

SiteComp

Description

This is class SiteComp.IAdapterFieldAttributes.

11.2.10.1 IAdapterFieldAttributes Members

The following tables list the members exposed by IAdapterFieldAttributes.

Public Methods

	Name	Description
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.

Public Properties

	Name	Description
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

11.2.10.2 IAdapterFieldAttributes Methods

The methods of the IAdapterFieldAttributes class are listed here.

Public Methods

	Name	Description
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.

11.2.10.2.1 IAdapterFieldAttributes.GetRequired Method

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This is GetRequired, a member of class IAdapterFieldAttributes.

11.2.10.2.2 IAdapterFieldAttributes.GetVisible Method

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

This is GetVisible, a member of class IAdapterFieldAttributes.

11.2.10.3 IAdapterFieldAttributes Properties

The properties of the IAdapterFieldAttributes class are listed here.

Public Properties

	Name	Description
	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

11.2.10.3.1 IAdapterFieldAttributes.Required Property

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

This is Required, a member of class IAdapterFieldAttributes.

11.2.10.3.2 IAdapterFieldAttributes.Visible Property

Pascal

```
property Visible: Boolean;
```

C++

```
__property Boolean Visible;
```

Description

This is Visible, a member of class IAdapterFieldAttributes.

11.2.11 IAdapterHiddenField Interface

Class Hierarchy

SiteComp.IAdapterHiddenField

Pascal

```
IAdapterHiddenField = interface;
```

C++

```
__interface IAdapterHiddenField;
```

File

SiteComp

Description

This is class SiteComp.IAdapterHiddenField.

11.2.11.1 IAdapterHiddenField Members

The following tables list the members exposed by IAdapterHiddenField.

Public Methods

	Name	Description
	GetName (see page 1002)	This is GetName, a member of class IAdapterHiddenField.
	GetValue (see page 1002)	This is GetValue, a member of class IAdapterHiddenField.

Public Properties

	Name	Description
R	Name (see page 1003)	This is Name, a member of class IAdapterHiddenField.
R	Value (see page 1003)	This is Value, a member of class IAdapterHiddenField.

11.2.11.2 IAdapterHiddenField Methods

The methods of the IAdapterHiddenField class are listed here.

Public Methods

	Name	Description
	GetName (see page 1002)	This is GetName, a member of class IAdapterHiddenField.
	GetValue (see page 1002)	This is GetValue, a member of class IAdapterHiddenField.

11.2.11.2.1 IAdapterHiddenField.GetName Method

Pascal

```
function GetName: string;
```

C++

```
__fastcall UnicodeString GetName();
```

Description

This is GetName, a member of class IAdapterHiddenField.

11.2.11.2.2 IAdapterHiddenField.GetValue Method

Pascal

```
function GetValue: string;
```

C++

```
__fastcall UnicodeString GetValue();
```

Description

This is GetValue, a member of class IAdapterHiddenField.

11.2.11.3 IAdapterHiddenField Properties

The properties of the IAdapterHiddenField class are listed here.

Public Properties

	Name	Description
	Name (see page 1003)	This is Name, a member of class IAdapterHiddenField.
	Value (see page 1003)	This is Value, a member of class IAdapterHiddenField.

11.2.11.3.1 IAdapterHiddenField.Name Property

Pascal

```
property Name: string;
```

C++

```
__property UnicodeString Name;
```

Description

This is Name, a member of class IAdapterHiddenField.

11.2.11.3.2 IAdapterHiddenField.Value Property

Pascal

```
property Value: string;
```

C++

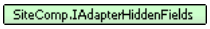
```
__property UnicodeString Value;
```

Description

This is Value, a member of class IAdapterHiddenField.

11.2.12 IAdapterHiddenFields Interface

Class Hierarchy



Pascal

```
IAdapterHiddenFields = interface;
```

C++

```
__interface IAdapterHiddenFields;
```

File

SiteComp

Description

This is class SiteComp.IAdapterHiddenFields.

11.2.12.1 IAdapterHiddenFields Members

The following tables list the members exposed by IAdapterHiddenFields.

Public Methods

	Name	Description
	GetField (see page 1004)	This is GetField, a member of class IAdapterHiddenFields.
	GetFieldCount (see page 1004)	This is GetFieldCount, a member of class IAdapterHiddenFields.

Public Properties

	Name	Description
	FieldCount (see page 1005)	This is FieldCount, a member of class IAdapterHiddenFields.
	Fields (see page 1005)	This is Fields, a member of class IAdapterHiddenFields.

11.2.12.2 IAdapterHiddenFields Methods

The methods of the IAdapterHiddenFields class are listed here.

Public Methods

	Name	Description
	GetField (see page 1004)	This is GetField, a member of class IAdapterHiddenFields.
	GetFieldCount (see page 1004)	This is GetFieldCount, a member of class IAdapterHiddenFields.

11.2.12.2.1 IAdapterHiddenFields.GetField Method

Pascal

```
function GetField(I: Integer): IAdapterHiddenField;
```

C++

```
__fastcall IAdapterHiddenField GetField(int I);
```

Description

This is GetField, a member of class IAdapterHiddenFields.

11.2.12.2.2 IAdapterHiddenFields.GetFieldCount Method

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

This is GetFieldCount, a member of class IAdapterHiddenFields.

11.2.12.3 IAdapterHiddenFields Properties

The properties of the IAdapterHiddenFields class are listed here.

Public Properties

	Name	Description
	FieldCount (see page 1005)	This is FieldCount, a member of class IAdapterHiddenFields.
	Fields (see page 1005)	This is Fields, a member of class IAdapterHiddenFields.

11.2.12.3.1 IAdapterHiddenFields.FieldCount Property

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

This is FieldCount, a member of class IAdapterHiddenFields.

11.2.12.3.2 IAdapterHiddenFields.Fields Property

Pascal

```
property Fields [I: Integer]: IAdapterHiddenField;
```

C++

```
__property IAdapterHiddenField Fields[int I];
```

Description

This is Fields, a member of class IAdapterHiddenFields.

11.2.13 IAdapterItemRequestParams Interface

IAdapterItemRequestParams is the interface for reading the parameters for an adapter field or action request.

Class Hierarchy

SiteComp.IAdapterItemRequestParams

Pascal

```
IAdapterItemRequestParams = interface;
```

C++

```
__interface IAdapterItemRequestParams;
```

File

SiteComp

Description

Use IAdapterItemRequestParams to read the parameters that accompany requests to an adapter field or action. The parameters, which can be accessed through the ParamValues (see page 1009) property, are a set of 'Name=Value' strings. In addition to providing access to the parameters, IAdapterItemRequestParams has two properties, SuccessPage (see page 1009) and FailurePage (see page 1008), which identify the Web pages that should be returned to the caller when the adapter request succeeds or fails.

Note: In C++, method declarations that use IAdapterItemRequestParams use the _di_IAdapterItemRequestParams type instead. This type is a DelphiInterface (see page 3256) wrapper around the IAdapterItemRequestParams interface:

Note: typedef System::DelphiInterface (see page 3256)< IAdapterItemRequestParams > _di_IAdapterItemRequestParams;

See Also

TPageDispatcher (see page 6236)

TCustomWebAppComponents (see page 6201)

TWebAppComponents (see page 6243)

11.2.13.1 IAdapterItemRequestParams Members

The following tables list the members exposed by IAdapterItemRequestParams.

Public Methods

	Name	Description
	GetFailurePage (see page 1006)	Returns the value of the FailurePage (see page 1008) property.
	GetParamValues (see page 1007)	Returns the value of the ParamValues (see page 1009) property.
	GetSuccessPage (see page 1007)	Returns the value of the SuccessPage (see page 1009) property.
	SetFailurePage (see page 1007)	Sets the value of the FailurePage (see page 1008) property.
	SetSuccessPage (see page 1008)	Sets the value of the SuccessPage (see page 1009) property.

Public Properties

	Name	Description
	FailurePage (see page 1008)	Indicates the Web page module that the application should return if the adapter request fails.
R	ParamValues (see page 1009)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 1009)	Indicates the Web page module that the application should return if the adapter request succeeds.

11.2.13.2 IAdapterItemRequestParams Methods

The methods of the IAdapterItemRequestParams class are listed here.

Public Methods

	Name	Description
	GetFailurePage (see page 1006)	Returns the value of the FailurePage (see page 1008) property.
	GetParamValues (see page 1007)	Returns the value of the ParamValues (see page 1009) property.
	GetSuccessPage (see page 1007)	Returns the value of the SuccessPage (see page 1009) property.
	SetFailurePage (see page 1007)	Sets the value of the FailurePage (see page 1008) property.
	SetSuccessPage (see page 1008)	Sets the value of the SuccessPage (see page 1009) property.

11.2.13.2.1 IAdapterItemRequestParams.GetFailurePage Method

Returns the value of the FailurePage (see page 1008) property.

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

GetFailurePage is the read implementation of the FailurePage (see page 1008) property.

See Also

FailurePage (see page 1008)

11.2.13.2.2 IAdapterItemRequestParams.GetParamValues Method

Returns the value of the ParamValues (see page 1009) property.

Pascal

```
function GetParamValues: TStrings;
```

C++

```
__fastcall TStrings GetParamValues();
```

Description

GetParamValues is the read implementation of the ParamValues (see page 1009) property.

See Also

ParamValues (see page 1009)

11.2.13.2.3 IAdapterItemRequestParams.GetSuccessPage Method

Returns the value of the SuccessPage (see page 1009) property.

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

GetSuccessPage is the read implementation of the SuccessPage (see page 1009) property.

See Also

SuccessPage (see page 1009)

11.2.13.2.4 IAdapterItemRequestParams.SetFailurePage Method

Sets the value of the FailurePage (see page 1008) property.

Pascal

```
procedure SetFailurePage(const AValue: string);
```

C++

```
__fastcall SetFailurePage(const UnicodeString AValue);
```

Description

SetFailurePage is the write implementation of the FailurePage (see page 1008) property.

AValue is the name of the Web page that should be returned if the adapter request fails.

See Also

FailurePage (see page 1008)

11.2.13.2.5 IAdapterItemRequestParams.SetSuccessPage Method

Sets the value of the SuccessPage (see page 1009) property.

Pascal

```
procedure SetSuccessPage(const AValue: string);
```

C++

```
__fastcall SetSuccessPage(const UnicodeString AValue);
```

Description

SetSuccessPage is the write implementation of the SuccessPage (see page 1009) property.

AValue is the name of the Web page that should be returned when the adapter request succeeds.

See Also

SuccessPage (see page 1009)

11.2.13.3 IAdapterItemRequestParams Properties

The properties of the IAdapterItemRequestParams class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 1008)	Indicates the Web page module that the application should return if the adapter request fails.
R	ParamValues (see page 1009)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 1009)	Indicates the Web page module that the application should return if the adapter request succeeds.

11.2.13.3.1 IAdapterItemRequestParams.FailurePage Property

Indicates the Web page module that the application should return if the adapter request fails.

Pascal

```
property FailurePage: string;
```

C++

```
__property UnicodeString FailurePage;
```

Description

FailurePage is the name of the registered Web page module that should generate the content of the response message if the request to the adapter fails.

See Also

SuccessPage (see page 1009)

11.2.13.3.2 IAdapterItemRequestParams.ParamValues Property

Lists the parameter values that are included in requests to the adapter action or field.

Pascal

```
property ParamValues: TStrings;
```

C++

```
__property TStrings ParamValues;
```

Description

Use ParamValues to access the list of parameter values. Use the Names property of ParamValues to obtain the names of the parameters and the Values property to access the corresponding values.

See Also

Names

Values

11.2.13.3.3 IAdapterItemRequestParams.SuccessPage Property

Indicates the Web page module that the application should return if the adapter request succeeds.

Pascal

```
property SuccessPage: string;
```

C++

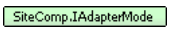
```
__property UnicodeString SuccessPage;
```

Description

SuccessPage is the name of the registered Web page module that should generate the content of the response message when the request to the adapter succeeds.

11.2.14 IAdapterMode Interface

Class Hierarchy



Pascal

```
IAdapterMode = interface;
```

C++

```
__interface IAdapterMode;
```

File

SiteComp

Description

This is class SiteComp.IAdapterMode.

11.2.14.1 IAdapterMode Members

The following tables list the members exposed by IAdapterMode.

Public Methods

	Name	Description
	GetAdapterMode (see page 1010)	This is GetAdapterMode, a member of class IAdapterMode.
	GetNeedAdapterMode (see page 1010)	This is GetNeedAdapterMode, a member of class IAdapterMode.
	RestoreDefaultAdapterMode (see page 1011)	This is RestoreDefaultAdapterMode, a member of class IAdapterMode.
	SetAdapterMode (see page 1011)	This is SetAdapterMode, a member of class IAdapterMode.

Public Properties

	Name	Description
	AdapterMode (see page 1011)	This is AdapterMode, a member of class IAdapterMode.
R	NeedAdapterMode (see page 1011)	This is NeedAdapterMode, a member of class IAdapterMode.

11.2.14.2 IAdapterMode Methods

The methods of the IAdapterMode class are listed here.

Public Methods

	Name	Description
	GetAdapterMode (see page 1010)	This is GetAdapterMode, a member of class IAdapterMode.
	GetNeedAdapterMode (see page 1010)	This is GetNeedAdapterMode, a member of class IAdapterMode.
	RestoreDefaultAdapterMode (see page 1011)	This is RestoreDefaultAdapterMode, a member of class IAdapterMode.
	SetAdapterMode (see page 1011)	This is SetAdapterMode, a member of class IAdapterMode.

11.2.14.2.1 IAdapterMode.GetAdapterMode Method

Pascal

```
function GetAdapterMode: string;
```

C++

```
__fastcall UnicodeString GetAdapterMode();
```

Description

This is GetAdapterMode, a member of class IAdapterMode.

11.2.14.2.2 IAdapterMode.GetNeedAdapterMode Method

Pascal

```
function GetNeedAdapterMode: Boolean;
```

C++

```
__fastcall Boolean GetNeedAdapterMode();
```

Description

This is GetNeedAdapterMode, a member of class IAdapterMode.

11.2.14.2.3 IAdapterMode.RestoreDefaultAdapterMode Method

Pascal

```
procedure RestoreDefaultAdapterMode;
```

C++

```
__fastcall RestoreDefaultAdapterMode();
```

Description

This is RestoreDefaultAdapterMode, a member of class IAdapterMode.

11.2.14.2.4 IAdapterMode.SetAdapterMode Method

Pascal

```
procedure SetAdapterMode(const AValue: string);
```

C++

```
__fastcall SetAdapterMode(const UnicodeString AValue);
```

Description

This is SetAdapterMode, a member of class IAdapterMode.

11.2.14.3 IAdapterMode Properties

The properties of the IAdapterMode class are listed here.

Public Properties

	Name	Description
	AdapterMode (see page 1011)	This is AdapterMode, a member of class IAdapterMode.
R	NeedAdapterMode (see page 1011)	This is NeedAdapterMode, a member of class IAdapterMode.

11.2.14.3.1 IAdapterMode.AdapterMode Property

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

This is AdapterMode, a member of class IAdapterMode.

11.2.14.3.2 IAdapterMode.NeedAdapterMode Property

Pascal

```
property NeedAdapterMode: Boolean;
```

C++

```
__property Boolean NeedAdapterMode;
```

Description

This is NeedAdapterMode, a member of class IAdapterMode.

11.2.15 IAdapterRequestHandler Interface

IAdapterRequestHandler handles requests to execute an adapter action or retrieve the value of an image field.

Class Hierarchy

Pascal

```
IAdapterRequestHandler = interface;
```

C++

```
__interface IAdapterRequestHandler;
```

File

SiteComp

Description

IAdapterRequestHandler is implemented by classes that execute an adapter action or look up the value of an image field.

See Also

TBaseAdapterAction (see page 5447)

TCustomAdapterImageField (see page 5623)

11.2.15.1 IAdapterRequestHandler Members

The following tables list the members exposed by IAdapterRequestHandler.

Public Methods

	Name	Description
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.

11.2.15.2 IAdapterRequestHandler Methods

The methods of the IAdapterRequestHandler class are listed here.

Public Methods

	Name	Description
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.

11.2.15.2.1 IAdapterRequestHandler.CreateRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext creates a new request context based on the specified parameters.

See Also

AdapterRequest (see page 5971)

TAdapterDispatcher (see page 6148)

CreateRequestContext (see page 5452)

CreateRequestContext (see page 5628)

11.2.15.2.2 IAdapterRequestHandler.HandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls HandleRequest to execute the adapter action and generate a response.

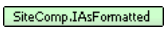
See Also

HandleRequest (see page 5463)

HandleRequest (see page 5630)

11.2.16 IAsFormatted Interface

Class Hierarchy



Pascal

```
IAsFormatted = interface;
```

C++

```
__interface IAsFormatted;
```

File

SiteComp

Description

This is class SiteComp.IAsFormatted.

11.2.16.1 IAsFormatted Members

The following tables list the members exposed by IAsFormatted.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.

11.2.16.2 IAsFormatted Methods

The methods of the IAsFormatted class are listed here.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.

11.2.16.2.1 IAsFormatted.AsFormatted Method

Pascal

```
function AsFormatted: string;
```

C++

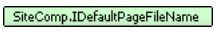
```
__fastcall UnicodeString AsFormatted();
```

Description

This is AsFormatted, a member of class IAsFormatted.

11.2.17 IDefaultPageFileName Interface

Class Hierarchy



Pascal

```
IDefaultPageFileName = interface;
```

C++

```
__interface IDefaultPageFileName;
```

File

SiteComp

Description

This is class SiteComp.IDefaultPageFileName.

11.2.17.1 IDefaultPageFileName Members

The following tables list the members exposed by IDefaultPageFileName.

Public Methods

	Name	Description
	SetDefaultPageFileName (see page 1015)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

11.2.17.2 IDefaultPageFileName Methods

The methods of the IDefaultPageFileName class are listed here.

Public Methods

	Name	Description
	SetDefaultPageFileName (see page 1015)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

11.2.17.2.1 IDefaultPageFileName.SetDefaultPageFileName Method

Pascal

```
procedure SetDefaultPageFileName(const APageFile: string);
```

C++

```
__fastcall SetDefaultPageFileName(const UnicodeString APageFile);
```

Description

This is SetDefaultPageFileName, a member of class IDefaultPageFileName.

11.2.18 IFieldValuesAdapter Interface

Class Hierarchy

SiteComp.IFieldValuesAdapter

Pascal

```
IFieldValuesAdapter = interface;
```

C++

```
__interface IFieldValuesAdapter;
```

File

SiteComp

Description

This is class SiteComp.IFieldValuesAdapter.

11.2.18.1 IFieldValuesAdapter Members

The following tables list the members exposed by IFieldValuesAdapter.

Public Methods

	Name	Description
	GetField (see page 1016)	This is GetField, a member of class IFieldValuesAdapter.
	GetValueIndex (see page 1016)	This is GetValueIndex, a member of class IFieldValuesAdapter.

11.2.18.2 IFieldValuesAdapter Methods

The methods of the IFieldValuesAdapter class are listed here.

Public Methods

	Name	Description
	GetField (see page 1016)	This is GetField, a member of class IFieldValuesAdapter.
	GetValueIndex (see page 1016)	This is GetValueIndex, a member of class IFieldValuesAdapter.

11.2.18.2.1 IFieldValuesAdapter.GetField Method

Pascal

```
function GetField: TObject;
```

C++

```
__fastcall TObject * GetField();
```

Description

This is GetField, a member of class IFieldValuesAdapter.

11.2.18.2.2 IFieldValuesAdapter.GetValueIndex Method

Pascal

```
function GetValueIndex: Integer;
```

C++

```
__fastcall int GetValueIndex();
```

Description

This is GetValueIndex, a member of class IFieldValuesAdapter.

11.2.19 IFindIncludeFile Interface

Class Hierarchy

SiteComp.IFindIncludeFile

Pascal

```
IFindIncludeFile = interface;
```

C++

```
__interface IFindIncludeFile;
```

File

SiteComp

Description

This is class SiteComp.IFindIncludeFile.

11.2.19.1 IFindIncludeFile Members

The following tables list the members exposed by IFindIncludeFile.

Public Methods

	Name	Description
	FindIncludeFile (see page 1017)	This is FindIncludeFile, a member of class IFindIncludeFile.

11.2.19.2 IFindIncludeFile Methods

The methods of the IFindIncludeFile class are listed here.

Public Methods

	Name	Description
	FindIncludeFile (see page 1017)	This is FindIncludeFile, a member of class IFindIncludeFile.

11.2.19.2.1 IFindIncludeFile.FindIncludeFile Method

Pascal

```
function FindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean;
```

C++

```
__fastcall Boolean FindIncludeFile(TComponent * AComponent, const UnicodeString AFileName,
UnicodeString AResult);
```

Description

This is FindIncludeFile, a member of class IFindIncludeFile.

11.2.20 IGetAdapterActions Interface

Class Hierarchy

SiteComp.IGetAdapterActions

Pascal

```
IGetAdapterActions = interface;
```

C++

```
__interface IGetAdapterActions;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterActions.

11.2.20.1 IGetAdapterActions Members

The following tables list the members exposed by IGetAdapterActions.

Public Methods

	Name	Description
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.

11.2.20.2 IGetAdapterActions Methods

The methods of the IGetAdapterActions class are listed here.

Public Methods

	Name	Description
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.

11.2.20.2.1 IGetAdapterActions.GetAdapterActions Method

Pascal

```
function GetAdapterActions: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterActions();
```

Description

This is GetAdapterActions, a member of class IGetAdapterActions.

11.2.21 IGetAdapterDisplayCharacteristics Interface

Class Hierarchy

SiteComp.IGetAdapterDisplayCharacteristics

Pascal

```
IGetAdapterDisplayCharacteristics = interface;
```

C++

```
__interface IGetAdapterDisplayCharacteristics;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterDisplayCharacteristics.

11.2.21.1 IGetAdapterDisplayCharacteristics Members

The following tables list the members exposed by IGetAdapterDisplayCharacteristics.

Public Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 1019)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

11.2.21.2 IGetAdapterDisplayCharacteristics Methods

The methods of the IGetAdapterDisplayCharacteristics class are listed here.

Public Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 1019)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

11.2.21.2.1 IGetAdapterDisplayCharacteristics.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.

11.2.22 IGetAdapterErrors Interface

Class Hierarchy

SiteComp.IGetAdapterErrors

Pascal

```
IGetAdapterErrors = interface;
```

C++

```
__interface IGetAdapterErrors;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterErrors.

11.2.22.1 IGetAdapterErrors Members

The following tables list the members exposed by IGetAdapterErrors.

Public Methods

	Name	Description
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.

11.2.22.2 IGetAdapterErrors Methods

The methods of the IGetAdapterErrors class are listed here.

Public Methods

	Name	Description
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.

11.2.22.2.1 IGetAdapterErrors.GetAdapterErrors Method

Pascal

```
function GetAdapterErrors: TObject;
```

C++

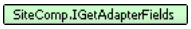
```
__fastcall TObject * GetAdapterErrors();
```

Description

This is GetAdapterErrors, a member of class IGetAdapterErrors.

11.2.23 IGetAdapterFields Interface

Class Hierarchy



Pascal

```
IGetAdapterFields = interface;
```

C++

```
__interface IGetAdapterFields;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterFields.

11.2.23.1 IGetAdapterFields Members

The following tables list the members exposed by IGetAdapterFields.

Public Methods

	Name	Description
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.

11.2.23.2 IGetAdapterFields Methods

The methods of the IGetAdapterFields class are listed here.

Public Methods

	Name	Description
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.

11.2.23.2.1 IGetAdapterFields.GetAdapterFields Method

Pascal

```
function GetAdapterFields: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterFields();
```

Description

This is GetAdapterFields, a member of class IGetAdapterFields.

11.2.24 IGetAdapterHiddenFields Interface

Class Hierarchy

SiteComp.IGetAdapterHiddenFields

Pascal

```
IGetAdapterHiddenFields = interface;
```

C++

```
__interface IGetAdapterHiddenFields;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterHiddenFields.

11.2.24.1 IGetAdapterHiddenFields Members

The following tables list the members exposed by IGetAdapterHiddenFields.

Public Methods

	Name	Description
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

11.2.24.2 IGetAdapterHiddenFields Methods

The methods of the IGetAdapterHiddenFields class are listed here.

Public Methods

	Name	Description
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

11.2.24.2.1 IGetAdapterHiddenFields.GetAdapterHiddenFields Method

Pascal

```
function GetAdapterHiddenFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenFields();
```

Description

This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.

11.2.24.2.2 IGetAdapterHiddenFields.GetAdapterHiddenRecordFields Method

Pascal

```
function GetAdapterHiddenRecordFields: TObject;
```

C++

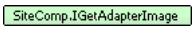
```
__fastcall TObject * GetAdapterHiddenRecordFields();
```

Description

This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.

11.2.25 IGetAdapterImage Interface

Class Hierarchy



Pascal

```
IGetAdapterImage = interface;
```

C++

```
__interface IGetAdapterImage;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterImage.

11.2.25.1 IGetAdapterImage Members

The following tables list the members exposed by IGetAdapterImage.

Public Methods

	Name	Description
	GetAdapterImage (see page 1023)	This is GetAdapterImage, a member of class IGetAdapterImage.

11.2.25.2 IGetAdapterImage Methods

The methods of the IGetAdapterImage class are listed here.

Public Methods

	Name	Description
	GetAdapterImage (see page 1023)	This is GetAdapterImage, a member of class IGetAdapterImage.

11.2.25.2.1 IGetAdapterImage.GetAdapterImage Method

Pascal

```
function GetAdapterImage: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterImage();
```

Description

This is GetAdapterImage, a member of class IGetAdapterImage.

11.2.26 IGetAdapterItemRequestParams Interface

Class Hierarchy

SiteComp.IGetAdapterItemRequestParams

Pascal

```
IGetAdapterItemRequestParams = interface;
```

C++

```
__interface IGetAdapterItemRequestParams;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterItemRequestParams.

11.2.26.1 IGetAdapterItemRequestParams Members

The following tables list the members exposed by IGetAdapterItemRequestParams.

Public Methods

	Name	Description
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

11.2.26.2 IGetAdapterItemRequestParams Methods

The methods of the IGetAdapterItemRequestParams class are listed here.

Public Methods

	Name	Description
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

11.2.26.2.1 IGetAdapterItemRequestParams.GetAdapterItemRequestParams Method

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(UnicodeString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.

11.2.27 IGetAdapterRequestDefaultResponse Interface

Class Hierarchy

SiteComp.IGetAdapterRequestDefaultResponse

Pascal

```
IGetAdapterRequestDefaultResponse = interface;
```

C++

```
__interface IGetAdapterRequestDefaultResponse;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterRequestDefaultResponse.

11.2.27.1 IGetAdapterRequestDefaultResponse Members

The following tables list the members exposed by IGetAdapterRequestDefaultResponse.

Public Methods

	Name	Description
	GetFailurePage (see page 1025)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetSuccessPage (see page 1025)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

Public Properties

	Name	Description
R	FailurePage (see page 1026)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	SuccessPage (see page 1026)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

11.2.27.2 IGetAdapterRequestDefaultResponse Methods

The methods of the IGetAdapterRequestDefaultResponse class are listed here.

Public Methods

	Name	Description
	GetFailurePage (see page 1025)	This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.
	GetSuccessPage (see page 1025)	This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

11.2.27.2.1 IGetAdapterRequestDefaultResponse.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

This is GetFailurePage, a member of class IGetAdapterRequestDefaultResponse.

11.2.27.2.2 IGetAdapterRequestDefaultResponse.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class IGetAdapterRequestDefaultResponse.

11.2.27.3 IGetAdapterRequestDefaultResponse Properties

The properties of the IGetAdapterRequestDefaultResponse class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 1026)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
	SuccessPage (see page 1026)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

11.2.27.3.1 IGetAdapterRequestDefaultResponse.FailurePage Property

Pascal

```
property FailurePage: string;
```

C++

```
__property UnicodeString FailurePage;
```

Description

This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.

11.2.27.3.2 IGetAdapterRequestDefaultResponse.SuccessPage Property

Pascal

```
property SuccessPage: string;
```

C++

```
__property UnicodeString SuccessPage;
```

Description

This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

11.2.28 IGetAdapterValuesList Interface

Class Hierarchy

SiteComp.IGetAdapterValuesList

Pascal

```
IGetAdapterValuesList = interface;
```

C++

```
__interface IGetAdapterValuesList;
```

File

SiteComp

Description

This is class SiteComp.IGetAdapterValuesList.

11.2.28.1 IGetAdapterValuesList Members

The following tables list the members exposed by IGetAdapterValuesList.

Public Methods

	Name	Description
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

11.2.28.2 IGetAdapterValuesList Methods

The methods of the IGetAdapterValuesList class are listed here.

Public Methods

	Name	Description
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

11.2.28.2.1 IGetAdapterValuesList.GetAdapterValuesList Method

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

This is GetAdapterValuesList, a member of class IGetAdapterValuesList.

11.2.29 IGetDefaultAction Interface

Class Hierarchy

SiteComp.IGetDefaultAction

Pascal

```
IGetDefaultAction = interface;
```

C++

```
__interface IGetDefaultAction;
```

File

SiteComp

Description

This is class SiteComp.IGetDefaultAction.

11.2.29.1 IGetDefaultAction Members

The following tables list the members exposed by IGetDefaultAction.

Public Methods

	Name	Description
	GetDefaultAction (see page 1028)	This is GetDefaultAction, a member of class IGetDefaultAction.

Public Properties

	Name	Description
	DefaultAction (see page 1028)	This is DefaultAction, a member of class IGetDefaultAction.

11.2.29.2 IGetDefaultAction Methods

The methods of the IGetDefaultAction class are listed here.

Public Methods

	Name	Description
	GetDefaultAction (see page 1028)	This is GetDefaultAction, a member of class IGetDefaultAction.

11.2.29.2.1 IGetDefaultAction.GetDefaultAction Method

Pascal

```
function GetDefaultAction: TComponent;
```

C++

```
__fastcall TComponent * GetDefaultAction();
```

Description

This is GetDefaultAction, a member of class IGetDefaultAction.

11.2.29.3 IGetDefaultAction Properties

The properties of the IGetDefaultAction class are listed here.

Public Properties

	Name	Description
	DefaultAction (see page 1028)	This is DefaultAction, a member of class IGetDefaultAction.

11.2.29.3.1 IGetDefaultAction.DefaultAction Property

Pascal

```
property DefaultAction: TComponent;
```

C++

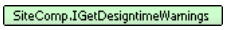
```
__property TComponent * DefaultAction;
```

Description

This is DefaultAction, a member of class IGetDefaultAction.

11.2.30 IGetDesigntimeWarnings Interface

Class Hierarchy



Pascal

```
IGetDesigntimeWarnings = interface;
```

C++

```
__interface IGetDesigntimeWarnings;
```

File

SiteComp

Description

This is class SiteComp.IGetDesigntimeWarnings.

11.2.30.1 IGetDesigntimeWarnings Members

The following tables list the members exposed by IGetDesigntimeWarnings.

Public Methods

	Name	Description
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.

11.2.30.2 IGetDesigntimeWarnings Methods

The methods of the IGetDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.

11.2.30.2.1 IGetDesigntimeWarnings.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the dataset values list needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the dataset values list an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DataSet property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. If DataSet is set, but the dataset is not active, GetDesigntimeWarnings adds a message that indicates this problem. Finally, if the ValueField property is not set, GetDesigntimeWarnings adds a message indicating this omission.

See Also

DataSet

Active

ValueField

11.2.31 IGetFieldValuesAdapter Interface

Class Hierarchy

SiteComp.IGetFieldValuesAdapter

Pascal

```
IGetFieldValuesAdapter = interface;
```

C++

```
__interface IGetFieldValuesAdapter;
```

File

SiteComp

Description

This is class SiteComp.IGetFieldValuesAdapter.

11.2.31.1 IGetFieldValuesAdapter Members

The following tables list the members exposed by IGetFieldValuesAdapter.

Public Methods

	Name	Description
	GetFieldValuesAdapter (see page 1031)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

11.2.31.2 IGetFieldValuesAdapter Methods

The methods of the IGetFieldValuesAdapter class are listed here.

Public Methods

	Name	Description
	GetFieldValuesAdapter (see page 1031)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

11.2.31.2.1 IGetFieldValuesAdapter.GetFieldValuesAdapter Method

Pascal

```
function GetFieldValuesAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetFieldValuesAdapter();
```

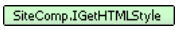
Description

This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.

11.2.32 IGetHTMLStyle Interface

IGetHTMLStyle is used to inquire about the preferred display representation of an adapter field or adapter action.

Class Hierarchy



Pascal

```
IGetHTMLStyle = interface;
```

C++

```
__interface IGetHTMLStyle;
```

File

SiteComp

Description

IGetHTMLStyle methods are typically called while generating HTML to display adapter fields and adapter actions. An adapter field implements this interface to indicate, for example, whether it's value should be displayed with an HTML input field or a select list. An adapter action implements this interface to indicate, for example, whether it's value should be displayed with an HTML button or hyperlink.

TAdapterPageProducer child components may call this interface when generating HTML. For example, TCustomAdapterDisplayField calls GetViewMode (see page 1033) when the TCustomAdapterDisplayField's ViewMode property is set to vmDefault. TCustomAdapterActionButton calls GetDisplayStyle (see page 1032) when its DisplayType property is ctDefault.

See Also

TAdapterNamedDisplayField (see page 5343)

TCustomAdapterDisplayColumn (see page 6461)

TCustomAdapterDisplayField (see page 6467)

11.2.32.1 IGetHTMLStyle Members

The following tables list the members exposed by IGetHTMLStyle.

Public Methods

	Name	Description
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.

11.2.32.2 IGetHTMLStyle Methods

The methods of the IGetHTMLStyle class are listed here.

Public Methods

	Name	Description
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.

11.2.32.2.1 IGetHTMLStyle.GetDisplayStyle Method

Returns the preferred display style for an adapter field or adapter action.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(const UnicodeString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Edit).

Adapter fields typically return one of the following values:

Value	Meaning
Text	Display the field as read-only text
Image	Display the field as an image
List	Display the field as a list of read only values
Button	Display the adapter action as a button
Image	Display the adapter action as an image
Anchor	Display the adapter action as a hyperlink
EventImages	Display the adapter action as an image with events that change the image as the mouse is moved.

Adapter actions typically return one of the following values:

Value	Meaning
Button	Display the adapter action as a button
Image	Display the adapter action as an image
Anchor	Display the adapter action as a hyperlink
EventImages	Display the adapter action as an image with events that change the image as the mouse is moved.

User the server-side script DisplayStyle property to get these values from adapter fields and adapter actions.

See Also

DisplayType (see page 6473)

DisplayType (see page 6439)

11.2.32.2.2 IGetHTMLStyle.GetInputStyle Method

Returns the preferred editing style for an adapter field.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetInputStyle(const UnicodeString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Browse).

Adapter fields typically return one of the following values:

Value	Meaning
TextInput	Edit the adapter field using a single line input field.
PasswordInput	Edit the adapter field as a password field.
Select	Edit the adapter field using a single selection list.
SelectMultiple	Edit the adapter field using a multiple selection list.
Radio	Edit the adapter field using radio buttons.
CheckBox	Edit the adapter field using check boxes.
TextArea	Edit the adapter field using a multi-line input field.
File	Edit the adapter field using a file upload input field.

Use the server-side script InputStyle property to get these values from adapter fields.

See Also

InputType (see page 6474)

11.2.32.2.3 IGetHTMLStyle.GetViewMode Method

Returns the preferred mode of an adapter field.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetViewMode(const UnicodeString AAdapterMode);
```

Description

The AAdapterMode parameter can be used to request the style to be used when the adapter is a particular mode (e.g.; Insert (see page 3601)).

Adapter fields typically return one of the following values:

Value	Meaning
Input (see page 3827)	Display the adapter field as an input field, by default use the style returned by the GetInputStyle (see page 1033) method.
Display	Display the adapter field as a read-only field, by default use the style returned by GetDisplayStyle (see page 1032).
ToggleOnAccess	Display as an input field in the end-user has modify rights, otherwise displays as a read-only field.

Use the server-side script ViewMode property to get these values from adapter fields.

See Also

ViewMode (see page 6475)

11.2.33 IGetProducerComponent Interface

Class Hierarchy

SiteComp.IGetProducerComponent

Pascal

```
IGetProducerComponent = interface;
```

C++

```
__interface IGetProducerComponent;
```

File

SiteComp

Description

This is class SiteComp.IGetProducerComponent.

11.2.33.1 IGetProducerComponent Members

The following tables list the members exposed by IGetProducerComponent.

Public Methods

	Name	Description
	GetProducerComponent (see page 1035)	This is GetProducerComponent, a member of class IGetProducerComponent.

11.2.33.2 IGetProducerComponent Methods

The methods of the IGetProducerComponent class are listed here.

Public Methods

	Name	Description
	GetProducerComponent (see page 1035)	This is GetProducerComponent, a member of class IGetProducerComponent.

11.2.33.2.1 IGetProducerComponent.GetProducerComponent Method

Pascal

```
function GetProducerComponent: TComponent;
```

C++

```
__fastcall TComponent * GetProducerComponent();
```

Description

This is GetProducerComponent, a member of class IGetProducerComponent.

11.2.34 IGetScriptObject Interface

Class Hierarchy

SiteComp.IGetScriptObject

Pascal

```
IGetScriptObject = interface;
```

C++

```
__interface IGetScriptObject;
```

File

SiteComp

Description

This is class SiteComp.IGetScriptObject.

11.2.34.1 IGetScriptObject Members

The following tables list the members exposed by IGetScriptObject.

Public Methods

	Name	Description
	GetScriptObject (see page 1036)	This is GetScriptObject, a member of class IGetScriptObject.

11.2.34.2 IGetScriptObject Methods

The methods of the IGetScriptObject class are listed here.

Public Methods

	Name	Description
	GetScriptObject (see page 1036)	This is GetScriptObject, a member of class IGetScriptObject.

11.2.34.2.1 IGetScriptObject.GetScriptObject Method

Pascal

```
function GetScriptObject: IInterface; deprecated;
```

C++

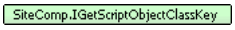
```
__fastcall IInterface GetScriptObject();
```

Description

This is GetScriptObject, a member of class IGetScriptObject.

11.2.35 IGetScriptObjectClassKey Interface

Class Hierarchy



Pascal

```
IGetScriptObjectClassKey = interface;
```

C++

```
__interface IGetScriptObjectClassKey;
```

File

SiteComp

Description

This is class SiteComp.IGetScriptObjectClassKey.

11.2.35.1 IGetScriptObjectClassKey Members

The following tables list the members exposed by IGetScriptObjectClassKey.

Public Methods

	Name	Description
	GetScriptObjectClassKey (see page 1037)	This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

11.2.35.2 IGetScriptObjectClassKey Methods

The methods of the IGetScriptObjectClassKey class are listed here.

Public Methods

	Name	Description
	GetScriptObjectClassKey (see page 1037)	This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

11.2.35.2.1 IGetScriptObjectClassKey.GetScriptObjectClassKey Method

Pascal

```
function GetScriptObjectClassKey: TClass; deprecated;
```

C++

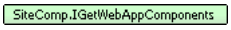
```
__fastcall TClass GetScriptObjectClassKey();
```

Description

This is GetScriptObjectClassKey, a member of class IGetScriptObjectClassKey.

11.2.36 IGetWebAppComponents Interface

Class Hierarchy



Pascal

```
IGetWebAppComponents = interface;
```

C++

```
__interface IGetWebAppComponents;
```

File

SiteComp

Description

This is class SiteComp.IGetWebAppComponents.

11.2.36.1 IGetWebAppComponents Members

The following tables list the members exposed by IGetWebAppComponents.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 1038)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 1038)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 1038)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 1038)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 1039)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 1039)	This is GetWebUserList, a member of class IGetWebAppComponents.

Public Properties

	Name	Description
R	WebApplicationInfo (see page 1039)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
R	WebEndUser (see page 1040)	This is WebEndUser, a member of class IGetWebAppComponents.

11.2.36.2 IGetWebAppComponents Methods

The methods of the IGetWebAppComponents class are listed here.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 1038)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 1038)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 1038)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 1038)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 1039)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 1039)	This is GetWebUserList, a member of class IGetWebAppComponents.

11.2.36.2.1 IGetWebAppComponents.GetAdapterDispatcher Method

Pascal

```
function GetAdapterDispatcher: IAdapterDispatcher;
```

C++

```
__fastcall IAdapterDispatcher GetAdapterDispatcher();
```

Description

This is GetAdapterDispatcher, a member of class IGetWebAppComponents.

11.2.36.2.2 IGetWebAppComponents.GetLocateFileService Method

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

This is GetLocateFileService, a member of class IGetWebAppComponents.

11.2.36.2.3 IGetWebAppComponents.GetPageDispatcher Method

Pascal

```
function GetPageDispatcher: IPageDispatcher;
```

C++

```
__fastcall IPageDispatcher GetPageDispatcher();
```

Description

This is GetPageDispatcher, a member of class IGetWebAppComponents.

11.2.36.2.4 IGetWebAppComponents.GetWebApplicationInfo Method

Pascal

```
function GetWebApplicationInfo: IWebApplicationInfo;
```

C++

```
__fastcall IWebApplicationInfo GetWebApplicationInfo();
```

Description

This is GetWebApplicationInfo, a member of class IGetWebAppComponents.

11.2.36.2.5 IGetWebAppComponents.GetWebEndUser Method

Pascal

```
function GetWebEndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser GetWebEndUser();
```

Description

This is GetWebEndUser, a member of class IGetWebAppComponents.

11.2.36.2.6 IGetWebAppComponents.GetWebUserList Method

Pascal

```
function GetWebUserList: IWebUserList;
```

C++

```
__fastcall IWebUserList GetWebUserList();
```

Description

This is GetWebUserList, a member of class IGetWebAppComponents.

11.2.36.3 IGetWebAppComponents Properties

The properties of the IGetWebAppComponents class are listed here.

Public Properties

	Name	Description
	WebApplicationInfo (see page 1039)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
	WebEndUser (see page 1040)	This is WebEndUser, a member of class IGetWebAppComponents.

11.2.36.3.1 IGetWebAppComponents.WebApplicationInfo Property

Pascal

```
property WebApplicationInfo: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo WebApplicationInfo;
```

Description

This is WebApplicationInfo, a member of class IGetWebAppComponents.

11.2.36.3.2 IGetWebAppComponents.WebEndUser Property

Pascal

```
property WebEndUser: IWebEndUser;
```

C++

```
__property IWebEndUser WebEndUser;
```

Description

This is WebEndUser, a member of class IGetWebAppComponents.

11.2.37 IIDEFileManager Interface

Class Hierarchy

SiteComp.IIDEFileManager

Pascal

```
IIDEFileManager = interface;
```

C++

```
__interface IIDEFileManager;
```

File

SiteComp

Description

This is class SiteComp.IIDEFileManager.

11.2.37.1 IIDEFileManager Members

The following tables list the members exposed by IIDEFileManager.

Public Methods

	Name	Description
	SearchFile (see page 1041)	This is SearchFile, a member of class IIDEFileManager.

11.2.37.2 IIDEFileManager Methods

The methods of the IIDEFileManager class are listed here.

Public Methods

	Name	Description
	SearchFile (see page 1041)	This is SearchFile, a member of class IIDEFileManager.

11.2.37.2.1 IIDEFileManager.SearchFile Method

Pascal

```
function SearchFile(const AFileName: string): string;
```

C++

```
__fastcall UnicodeString SearchFile(const UnicodeString AFileName);
```

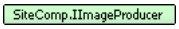
Description

This is SearchFile, a member of class IIDEFileManager.

11.2.38 IImageProducer Interface

IImageProducer is the interface for accessing the image that an adapter component supplies.

Class Hierarchy



Pascal

```
IImageProducer = interface;
```

C++

```
__interface IImageProducer;
```

File

SiteComp

Description

IImageProducer is the interface used to access the object that supplies an image to server-side script in a WebSnap application. It defines a single method, GetAdapterImage (see page 1042), which returns the interface of the object that is responsible for providing the image. That object may generate the image, or it may simply generate an HREF that addresses the object that generates the image.

Note that IImageProducer introduces an extra level of indirection when fetching images. IImageProducer is not necessarily the interface of the object that provides the image, it merely retrieves that interface. The reason for this indirection is because the object that implements IImageProducer may work with multiple adapter fields that produce a variety of images, depending on the supplied caption.

Note: In C++, method declarations that use IImageProducer use the _di_IImageProducer type instead. This type is a DelphiInterface (see page 3256) wrapper around the IImageProducer interface:

Note: typedef System::DelphiInterface (see page 3256)< IImageProducer > _di_IImageProducer;

See Also

IActionImageProducer (see page 978)

TDataSetAdapterImageField

TAdapterImageField (see page 5308)

11.2.38.1 IImageProducer Members

The following tables list the members exposed by IImageProducer.

Public Methods

	Name	Description
	GetAdapterImage (see page 1042)	Returns the object that is responsible for providing a specified image.

11.2.38.2 IImageProducer Methods

The methods of the IImageProducer class are listed here.

Public Methods

	Name	Description
	GetAdapterImage (see page 1042)	Returns the object that is responsible for providing a specified image.

11.2.38.2.1 IImageProducer.GetAdapterImage Method

Returns the object that is responsible for providing a specified image.

Pascal

```
function GetAdapterImage(const AComponentName: string; const ACaption: WideString): TObject;
```

C++

```
__fastcall TObject * GetAdapterImage(const UnicodeString AComponentName, const BSTR ACaption);
```

Description

GetAdapterImage returns the object that supplies an HREF for the specified image. That object can supply the HREF in one of two ways:

It can implement the IWebImageHREF (see page 1097) interface.

It can be the object that produces the image in response to request messages. In this case, the adapter dispatcher's EncodeAdapterItemRequestAsHREF method generates the HREF to address this object.

AComponentName is the name of the component that is requesting the image. GetAdapterImage can use this parameter to determine which of several objects is appropriate for supplying the image.

ACaption is a text label that describes the desired image. GetAdapterImage can use this parameter to determine which of several objects is appropriate for supplying the image.

See Also

IWebImageHREF (see page 1097)

EncodeAdapterItemRequestAsHREF (see page 6173)

11.2.39 IInternetEnvOptions Interface

Class Hierarchy

SiteComp.IInternetEnvOptions

Pascal

```
IInternetEnvOptions = interface;
```

C++

```
__interface IInternetEnvOptions;
```

File

SiteComp

Description

This is class SiteComp.IInternetEnvOptions.

11.2.39.1 IInternetEnvOptions Members

The following tables list the members exposed by IInternetEnvOptions.

Public Methods

	Name	Description
	GetDisableDebugging (see page 1043)	This is GetDisableDebugging, a member of class IInternetEnvOptions.
	GetHTMLExt (see page 1044)	This is GetHTMLExt, a member of class IInternetEnvOptions.
	GetSampleImageFile (see page 1044)	This is GetSampleImageFile, a member of class IInternetEnvOptions.
	GetScriptTimeoutSecs (see page 1044)	This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

Public Properties

	Name	Description
R	DisableDebugger (see page 1044)	This is DisableDebugger, a member of class IInternetEnvOptions.
R	HTMLExt (see page 1045)	This is HTMLExt, a member of class IInternetEnvOptions.
R	SampleImageFile (see page 1045)	This is SampleImageFile, a member of class IInternetEnvOptions.
R	ScriptTimeoutSecs (see page 1045)	This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

11.2.39.2 IInternetEnvOptions Methods

The methods of the IInternetEnvOptions class are listed here.

Public Methods

	Name	Description
	GetDisableDebugging (see page 1043)	This is GetDisableDebugging, a member of class IInternetEnvOptions.
	GetHTMLExt (see page 1044)	This is GetHTMLExt, a member of class IInternetEnvOptions.
	GetSampleImageFile (see page 1044)	This is GetSampleImageFile, a member of class IInternetEnvOptions.
	GetScriptTimeoutSecs (see page 1044)	This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

11.2.39.2.1 IInternetEnvOptions.GetDisableDebugging Method

Pascal

```
function GetDisableDebugging: Boolean;
```

C++

```
__fastcall Boolean GetDisableDebugging();
```

Description

This is GetDisableDebugging, a member of class IInternetEnvOptions.

11.2.39.2.2 IInternetEnvOptions.GetHTMLExt Method

Pascal

```
function GetHTMLExt: string;
```

C++

```
__fastcall UnicodeString GetHTMLExt();
```

Description

This is GetHTMLExt, a member of class IInternetEnvOptions.

11.2.39.2.3 IInternetEnvOptions.GetSampleImageFile Method

Pascal

```
function GetSampleImageFile: string;
```

C++

```
__fastcall UnicodeString GetSampleImageFile();
```

Description

This is GetSampleImageFile, a member of class IInternetEnvOptions.

11.2.39.2.4 IInternetEnvOptions.GetScriptTimeoutSecs Method

Pascal

```
function GetScriptTimeoutSecs: Integer;
```

C++

```
__fastcall int GetScriptTimeoutSecs();
```

Description

This is GetScriptTimeoutSecs, a member of class IInternetEnvOptions.

11.2.39.3 IInternetEnvOptions Properties

The properties of the IInternetEnvOptions class are listed here.

Public Properties

	Name	Description
	DisableDebugger (see page 1044)	This is DisableDebugger, a member of class IInternetEnvOptions.
	HTMLExt (see page 1045)	This is HTMLExt, a member of class IInternetEnvOptions.
	SampleImageFile (see page 1045)	This is SampleImageFile, a member of class IInternetEnvOptions.
	ScriptTimeoutSecs (see page 1045)	This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

11.2.39.3.1 IInternetEnvOptions.DisableDebugger Property

Pascal

```
property DisableDebugger: Boolean;
```

C++

```
__property Boolean DisableDebugger;
```

Description

This is DisableDebugger, a member of class IInternetEnvOptions.

11.2.39.3.2 IInternetEnvOptions.HTMLExt Property

Pascal

```
property HTMLExt: string;
```

C++

```
__property UnicodeString HTMLExt;
```

Description

This is HTMLExt, a member of class IInternetEnvOptions.

11.2.39.3.3 IInternetEnvOptions.SampleImageFile Property

Pascal

```
property SampleImageFile: string;
```

C++

```
__property UnicodeString SampleImageFile;
```

Description

This is SampleImageFile, a member of class IInternetEnvOptions.

11.2.39.3.4 IInternetEnvOptions.ScriptTimeoutSecs Property

Pascal

```
property ScriptTimeoutSecs: Integer;
```

C++

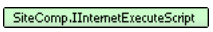
```
__property int ScriptTimeoutSecs;
```

Description

This is ScriptTimeoutSecs, a member of class IInternetEnvOptions.

11.2.40 IInternetExecuteScript Interface

Class Hierarchy



Pascal

```
IInternetExecuteScript = interface;
```

C++

```
__interface IInternetExecuteScript;
```

File

SiteComp

Description

This is class SiteComp.IInternetExecuteScript.

11.2.40.1 IInternetExecuteScript Members

The following tables list the members exposed by IInternetExecuteScript.

Public Methods

	Name	Description
	CanExecuteScript (see page 1046)	This is CanExecuteScript, a member of class IInternetExecuteScript.
	ExecuteScript (see page 1046)	This is ExecuteScript, a member of class IInternetExecuteScript.

11.2.40.2 IInternetExecuteScript Methods

The methods of the IInternetExecuteScript class are listed here.

Public Methods

	Name	Description
	CanExecuteScript (see page 1046)	This is CanExecuteScript, a member of class IInternetExecuteScript.
	ExecuteScript (see page 1046)	This is ExecuteScript, a member of class IInternetExecuteScript.

11.2.40.2.1 IInternetExecuteScript.CanExecuteScript Method

Pascal

```
function CanExecuteScript(AContext: IUnknown): Boolean;
```

C++

```
__fastcall Boolean CanExecuteScript(IUnknown AContext);
```

Description

This is CanExecuteScript, a member of class IInternetExecuteScript.

11.2.40.2.2 IInternetExecuteScript.ExecuteScript Method

Pascal

```
procedure ExecuteScript(AContext: IUnknown);
```

C++

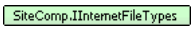
```
__fastcall ExecuteScript(IUnknown AContext);
```

Description

This is ExecuteScript, a member of class IInternetExecuteScript.

11.2.41 IInternetFileTypes Interface

Class Hierarchy



Pascal

```
IInternetFileTypes = interface;
```

C++

```
__interface IInternetFileTypes;
```

File

SiteComp

Description

This is class SiteComp.IInternetFileTypes.

11.2.41.1 IInternetFileTypes Members

The following tables list the members exposed by IInternetFileTypes.

Public Methods

	Name	Description
	FindFileExt (see page 1047)	This is FindFileExt, a member of class IInternetFileTypes.

11.2.41.2 IInternetFileTypes Methods

The methods of the IInternetFileTypes class are listed here.

Public Methods

	Name	Description
	FindFileExt (see page 1047)	This is FindFileExt, a member of class IInternetFileTypes.

11.2.41.2.1 IInternetFileTypes.FindFileExt Method

Pascal

```
function FindFileExt(const AFileExt: string): TAbstractInetFileType;
```

C++

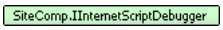
```
__fastcall TAbstractInetFileType FindFileExt(const UnicodeString AFileExt);
```

Description

This is FindFileExt, a member of class IInternetFileTypes.

11.2.42 IInternetScriptDebugger Interface

Class Hierarchy



Pascal

```
IInternetScriptDebugger = interface;
```

C++

```
__interface IInternetScriptDebugger;
```

File

SiteComp

Description

This is class SiteComp.IInternetScriptDebugger.

11.2.42.1 IInternetScriptDebugger Members

The following tables list the members exposed by IInternetScriptDebugger.

Public Methods

	Name	Description
	CanDebugScript (see page 1048)	This is CanDebugScript, a member of class IInternetScriptDebugger.
	DebugScript (see page 1049)	This is DebugScript, a member of class IInternetScriptDebugger.

11.2.42.2 IInternetScriptDebugger Methods

The methods of the IInternetScriptDebugger class are listed here.

Public Methods

	Name	Description
	CanDebugScript (see page 1048)	This is CanDebugScript, a member of class IInternetScriptDebugger.
	DebugScript (see page 1049)	This is DebugScript, a member of class IInternetScriptDebugger.

11.2.42.2.1 IInternetScriptDebugger.CanDebugScript Method

Pascal

```
function CanDebugScript(AContext: IUnknown): Boolean;
```

C++

```
__fastcall Boolean CanDebugScript(IUnknown AContext);
```

Description

This is CanDebugScript, a member of class IInternetScriptDebugger.

11.2.42.2.2 IInternetScriptDebugger.DebugScript Method

Pascal

```
procedure DebugScript(AContext: IUnknown);
```

C++

```
__fastcall DebugScript(IUnknown AContext);
```

Description

This is DebugScript, a member of class IInternetScriptDebugger.

11.2.43 IIsAdapterActionList Interface

Class Hierarchy

SiteComp.IIsAdapterActionList

Pascal

```
IIsAdapterActionList = interface;
```

C++

```
__interface IIsAdapterActionList;
```

File

SiteComp

Description

This is class SiteComp.IIsAdapterActionList.

11.2.43.1 IIsAdapterActionList Members

The following tables list the members exposed by IIsAdapterActionList.

Public Methods

	Name	Description
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

Public Properties

	Name	Description
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.

11.2.43.2 IIsAdapterActionList Methods

The methods of the IIsAdapterActionList class are listed here.

Public Methods

	Name	Description
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

11.2.43.2.1 IIsAdapterActionList.GetIsAdapterActionList Method

Pascal

```
function GetIsAdapterActionList: Boolean;
```

C++

```
__fastcall Boolean GetIsAdapterActionList();
```

Description

This is GetIsAdapterActionList, a member of class IIsAdapterActionList.

11.2.43.3 IIsAdapterActionList Properties

The properties of the IIsAdapterActionList class are listed here.

Public Properties

	Name	Description
	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.

11.2.43.3.1 IIsAdapterActionList.IsAdapterActionList Property

Pascal

```
property IsAdapterActionList: Boolean;
```

C++

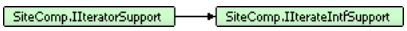
```
__property Boolean IsAdapterActionList;
```

Description

This is IsAdapterActionList, a member of class IIsAdapterActionList.

11.2.44 IIterateIntfSupport Interface

Class Hierarchy



Pascal

```
IIterateIntfSupport = interface(IIteratorSupport);
```

C++

```
__interface IIterateIntfSupport : public IIteratorSupport;
```

File

SiteComp

Description

This is class SiteComp.IIterateIntfSupport.

11.2.44.1 IIterateIntfSupport Members

The following tables list the members exposed by IIterateIntfSupport.

Public Methods

	Name	Description
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentIntf (see page 1051)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	NextIteration (see page 1056)	Advances to the objects next item.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

11.2.44.2 IIterateIntfSupport Methods

The methods of the IIterateIntfSupport class are listed here.

Public Methods

	Name	Description
	GetCurrentIntf (see page 1051)	This is GetCurrentIntf, a member of class IIterateIntfSupport.

11.2.44.2.1 IIterateIntfSupport.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

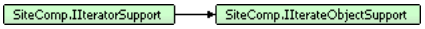
```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class IIterateIntfSupport.

11.2.45 IIterateObjectSupport Interface

Class Hierarchy



Pascal

```
IIterateObjectSupport = interface(IIteratorSupport);
```

C++

```
__interface IIterateObjectSupport : public IIteratorSupport;
```

File

SiteComp

Description

This is class SiteComp.IIterateObjectSupport.

11.2.45.1 IIterateObjectSupport Members

The following tables list the members exposed by IIterateObjectSupport.

Public Methods

	Name	Description
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 1056)	Advances to the objects next item.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

11.2.45.2 IIterateObjectSupport Methods

The methods of the IIterateObjectSupport class are listed here.

Public Methods

	Name	Description
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.

11.2.45.2.1 IIterateObjectSupport.GetCurrentObject Method

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

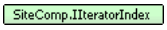
Description

This is GetCurrentObject, a member of class IIterateObjectSupport.

11.2.46 IIteratorIndex Interface

IIteratorIndex is the interface that indicates where an object is positioned when it is iterating over its subitems.

Class Hierarchy



Pascal

```
IIteratorIndex = interface;
```

C++

```
__interface IIteratorIndex;
```

File

SiteComp

Description

Server-side script uses the IIteratorIndex interface to detect when an object is in the middle of an enumeration, and if it is, to determine the item on which the object is positioned. This interface is used in conjunction with the IIteratorSupport (see page

1054) interface to enable server-side script to iterate over the items of an adapter or one of its sub-components.

Note: In C++, method declarations that use IIteratorIndex use the _di_IIteratorIndex type instead. This type is a DelphiInterface (see page 3256) wrapper around the IIteratorIndex interface:

Note: typedef System::DelphiInterface (see page 3256)< IIteratorIndex > _di_IIteratorIndex;

See Also

IIteratorSupport (see page 1054)

11.2.46.1 IIteratorIndex Members

The following tables list the members exposed by IIteratorIndex.

Public Methods

	Name	Description
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.

Public Properties

	Name	Description
R	IteratorIndex (see page 1054)	Indicates the index of the current item.

11.2.46.2 IIteratorIndex Methods

The methods of the IIteratorIndex class are listed here.

Public Methods

	Name	Description
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.

11.2.46.2.1 IIteratorIndex.GetIteratorIndex Method

Returns the value of the IteratorIndex (see page 1054) property.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

GetIteratorIndex is the read implementation for the IteratorIndex (see page 1054) property.

See Also

IteratorIndex (see page 1054)

11.2.46.2.2 IIteratorIndex.InIterator Method

Indicates whether the object is iterating over its items.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

InIterator returns true if the object is iterating over its items. It returns false if an iteration never started or if an iteration process finished. When InIterator returns false, the current position of the object is indeterminate. In this case, the IteratorIndex (see page 1054) property's value can't be used.

See Also

InIterator (see page 5562)

IIteratorSupport (see page 1054)

11.2.46.3 IIteratorIndex Properties

The properties of the IIteratorIndex class are listed here.

Public Properties

	Name	Description
	IteratorIndex (see page 1054)	Indicates the index of the current item.

11.2.46.3.1 IIteratorIndex.IteratorIndex Property

Indicates the index of the current item.

Pascal

```
property IteratorIndex: Integer;
```

C++

```
__property int IteratorIndex;
```

Description

IteratorIndex indicates the index of the current item when the object is iterating over its items. When IteratorIndex is 0, the object is positioned on its first item; When IteratorIndex is 1, the object is positioned on its second item, and so on.

When the object is not iterating over its items (when the InIterator (see page 1053) method returns false), the value of IteratorIndex is undefined.

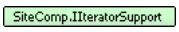
See Also

InIterator (see page 1053)

11.2.47 IIteratorSupport Interface

IIteratorSupport is the interface that allows server-side script to iterate over the items of an adapter or one of its child objects.

Class Hierarchy



Pascal

```
IIteratorSupport = interface;
```

C++

```
__interface IIteratorSupport;
```

File

SiteComp

Description

Server-side script uses the IIteratorSupport interface to enumerate the items of an adapter or one of its sub-components. When the server-side script includes an enumerator, the WebSnap application automatically generates calls to the IIteratorSupport methods of the object that contains the items to enumerate.

The WebSnap application first calls the StartIterator (see page 1056) method to position the object on its first item. To advance to the next item, the application calls the NextIteration (see page 1056) method. Finally, the application calls the EndIterator (see page 1056) method to signal the end of the iteration.

Each object that implements IIteratorSupport determines the items over which it iterates. For example, if the object is an adapter, these can be records or pages that the adapter uses to group information, or name/value pairs that the adapter represents. If the object is a list of fields or actions, these can be the fields or actions in the list. If the object is an adapter field, these can be the possible values of the field. If the object is an adapter action, these can be the commands the action can execute.

Note: In C++, method declarations that use IIteratorSupport use the _di_IIteratorSupport type instead. This type is a DelphiInterface (see page 3256) wrapper around the IIteratorSupport interface:

Note: typedef System::DelphiInterface (see page 3256)< IIteratorSupport > _di_IIteratorSupport;

See Also

IIteratorIndex (see page 1052)

11.2.47.1 IIteratorSupport Members

The following tables list the members exposed by IIteratorSupport.

Public Methods

	Name	Description
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	NextIteration (see page 1056)	Advances to the objects next item.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

11.2.47.2 IIteratorSupport Methods

The methods of the IIteratorSupport class are listed here.

Public Methods

	Name	Description
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	NextIteration (see page 1056)	Advances to the objects next item.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

11.2.47.2.1 IIteratorSupport.EndIterator Method

Signals the end of the iterating operation.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

Server-side script calls EndIterator when it stops iterating over the object's items. The object uses this method to perform any clean-up to reflect the fact that it is no longer iterating over its items.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator (see page 1056) method. It is passed to each NextIteration (see page 1056) call (which can modify the information), and finally supplied to the EndIterator method, which can free the memory used to store the information.

See Also

StartIterator (see page 1056)

NextIteration (see page 1056)

11.2.47.2.2 IIteratorSupport.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator (see page 1056) method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator (see page 1056) method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

StartIterator (see page 1056)

EndIterator (see page 1056)

11.2.47.2.3 IIteratorSupport.StartIterator Method

Positions the object on its first item and starts the iteration process.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

Server-side script calls StartIterator to begin iterating over the object's items.

OwnerData returns a pointer to information that the object can use to influence the iteration process. StartIterator initializes this information to reflect what ever information the object needs to communicate to itself as it advances over its items. This pointer is then passed to the next call to the NextIteration (see page 1056) or EndIterator (see page 1056) method.

StartIterator returns true if it successfully moves to the first item, false if there are no items to iterate.

See Also

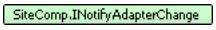
NextIteration (see page 1056)

EndIterator (see page 1056)

11.2.48 INotifyAdapterChange Interface

INotifyAdapterChange is the interface that allows objects to respond when an adapter changes.

Class Hierarchy



Pascal

```
INotifyAdapterChange = interface;
```

C++

```
__interface INotifyAdapterChange;
```

File

SiteComp

Description

Objects that depend on the fields, actions, or properties of an adapter implement the INotifyAdapterChange interface so that the adapter can notify them of changes. They register an interest in such notifications using the adapter's INotifyList (see page 1058) interface. Then, when the adapter changes, it uses this interface to inform them of the change. They can remove themselves from the adapter's notification list using the INotifyList (see page 1058) interface.

See Also

TCustomAdapter (see page 5531)

INotifyList (see page 1058)

11.2.48.1 INotifyAdapterChange Members

The following tables list the members exposed by INotifyAdapterChange.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.

11.2.48.2 INotifyAdapterChange Methods

The methods of the INotifyAdapterChange class are listed here.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.

11.2.48.2.1 INotifyAdapterChange.NotifyAdapterChange Method

Responds when the adapter changes.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

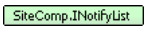
NotifyAdapterChange is the method objects implement to respond when the associated adapter changes.

Sender is the adapter component that changed.

11.2.49 INotifyList Interface

INotifyList is the interface that allows objects to register and unregister an interest in changes to a component.

Class Hierarchy



Pascal

```
INotifyList = interface;
```

C++

```
__interface INotifyList;
```

File

SiteComp

Description

WebSnap components that let other objects register interest in their changes implement INotifyList. When another object wants to be informed of changes, it calls the AddNotify (see page 1059) method. Then, when the component that implements INotifyList changes, it informs all of the objects that have called AddNotify (see page 1059) of the change. When an object wants to unregister its interest, it calls the RemoveNotify (see page 1059) method.

See Also

TNotifyList (see page 968)

11.2.49.1 INotifyList Members

The following tables list the members exposed by INotifyList.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.

11.2.49.2 INotifyList Methods

The methods of the INotifyList class are listed here.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.

11.2.49.2.1 INotifyList.AddNotify Method

Registers an interest in change notifications.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to register an interest in change notifications. After a call to AddNotify, the object is notified every time the component changes, until it unregisters its interest by a call to the RemoveNotify (see page 1059) method.

ANotify is the object to notify of changes. It should implement an interface that allows the notifying component to call it when changes occur. What interface this is depends on the component that implements INotifyList (see page 1058).

See Also

RemoveNotify (see page 1059)

11.2.49.2.2 INotifyList.RemoveNotify Method

Unregisters interest in change notifications.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to unregister interest in change notifications. After a call to RemoveNotify, the object is no longer notified when the component changes.

ANotify is the object to unregister. It must be an object that previously registered interest by calling the AddNotify (see page 1059) method.

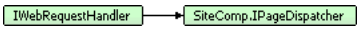
See Also

AddNotify (see page 1059)

11.2.50 IPageDispatcher Interface

IPageDispatcher is the interface for interacting with the page dispatcher.

Class Hierarchy



Pascal

```
IPageDispatcher = interface(IWebRequestHandler);
```

C++

```
__interface IPageDispatcher : public IWebRequestHandler;
```

File

SiteComp

Description

The TPageDispatcher component implements IPageDispatcher. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to dispatch request messages. The request messages identify a target Web page using the pathinfo portion of their target URL. If the request is not addressed to an URL with a pathinfo portion, the IPageDispatcher forwards the request to the Web page module identified by the DefaultPageName (see page 1064) property.

In addition to dispatching requests for Web pages, IPageDispatcher includes methods that check whether the current user needs to log in, check whether the current user has rights to view the requested page, and to redirect requests.

Note: In C++, method declarations that use IPageDispatcher use the _di_IPageDispatcher type instead. This type is a DelphiInterface (see page 3256) wrapper around the IPageDispatcher interface:

Note: typedef System::DelphiInterface (see page 3256)< IPageDispatcher > _di_IPageDispatcher;

See Also

TPageDispatcher (see page 6236)

TCustomWebAppComponents (see page 6201)

TWebAppComponents (see page 6243)

11.2.50.1 IPageDispatcher Members

The following tables list the members exposed by IPageDispatcher.

Public Methods

	Name	Description
	DispatchPageName (see page 1061)	Checks whether a specified user has a given access rights string.

	GetCanView (see page 1062)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 1062)	Returns the value of the DefaultPageName (see page 1064) property.
	GetLoginRequired (see page 1062)	Indicates whether a user must log in before accessing a specified Web page.
	PageNameOfRequest (see page 1063)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 1063)	Fills out a response message so that it redirects the caller to another page in the application.

Public Properties

	Name	Description
R	DefaultPageName (see page 1064)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

11.2.50.2 IPageDispatcher Methods

The methods of the IPageDispatcher class are listed here.

Public Methods

	Name	Description
	DispatchPageName (see page 1061)	Checks whether a specified user has a given access rights string.
	GetCanView (see page 1062)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 1062)	Returns the value of the DefaultPageName (see page 1064) property.
	GetLoginRequired (see page 1062)	Indicates whether a user must log in before accessing a specified Web page.
	PageNameOfRequest (see page 1063)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 1063)	Fills out a response message so that it redirects the caller to another page in the application.

11.2.50.2.1 IPageDispatcher.DispatchPageName Method

Checks whether a specified user has a given access rights string.

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean DispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

Call DispatchPageName to fill out a response message with the Web page that a specified Web page module provides. DispatchPageName locates the Web page module that produces the requested Web page and uses it to fill out the content of a response object.

APageName is the name of the Web page to fetch. If APageName is an empty string, DispatchPageName uses the first Web page module that is registered with the current Web context.

AResponse is the response object whose Content property receives the requested Web page.

AFlags indicates the attributes that must be present on the Web page module. For example, if AFlags includes dpPublished, APageName must specify a published Web page module or the page dispatcher won't fetch the Web page.

DispatchPageName returns true if it successfully fetches the requested page, false if it fails to obtain the requested page.

See Also

RedirectToPageName (see page 1063)

HandleRequest

Content

WebContext (see page 6003)

11.2.50.2.2 IPageDispatcher.GetCanView Method

Indicates whether the current user is permitted to view a specified Web page.

Pascal

```
function GetCanView(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetCanView(const UnicodeString APageName);
```

Description

Call GetCanView to check whether the current user should be allowed to view a specified Web page. Typically, this method is called after the user logs in.

APageName is the name of the Web page the user wants to view.

See Also

GetLoginRequired (see page 1062)

HasViewRights (see page 5992)

OnCanViewPage (see page 6198)

11.2.50.2.3 IPageDispatcher.GetDefaultPageName Method

Returns the value of the DefaultPageName (see page 1064) property.

Pascal

```
function GetDefaultPageName: string;
```

C++

```
__fastcall UnicodeString GetDefaultPageName();
```

Description

GetDefaultPageName is the read implementation of the DefaultPageName (see page 1064) property.

See Also

DefaultPageName (see page 1064)

11.2.50.2.4 IPageDispatcher.GetLoginRequired Method

Indicates whether a user must log in before accessing a specified Web page.

Pascal

```
function GetLoginRequired(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired(const UnicodeString APageName);
```

Description

Call GetLoginRequired to check whether users must log in before accessing a specified Web page. GetLoginRequired can check the Web page information object for the target Web page, but can also use other means (such as an event handler) to obtain this information.

APageName is the name of the Web page to check whether it requires users to first log in.

See Also

GetCanView (see page 1062)

IsLoginRequired

OnIsLoginRequired (see page 6199)

11.2.50.2.5 IPageDispatcher.PageNameOfRequest Method

Returns the name of the Web page to which an HTTP request is addressed.

Pascal

```
function PageNameOfRequest(ARequest: TWebRequest): string;
```

C++

```
__fastcall UnicodeString PageNameOfRequest(TWebRequest ARequest);
```

Description

Call PageNameOfRequest to obtain the name of the Web page that a request message is trying to fetch.

ARequest is the request object that describes the HTTP request message.

PageNameOfRequest returns the last string in the path info portion of the request message's target URL.

See Also

DefaultPageName (see page 1064)

11.2.50.2.6 IPageDispatcher.RedirectToPageName Method

Fills out a response message so that it redirects the caller to another page in the application.

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean RedirectToPageName(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

Call RedirectToPageName to generate a response that redirects the caller to another Web page in the WebSnap application.

APageName is the name of the Web page to which the request should be redirected. If APageName is an empty string, RedirectToPageName uses the first Web page module that is registered with the current Web context.

AParams is a string list containing application-defined name-value pairs. These name-value pairs supply the query fields of the redirected request message, and are added to the URL that is the target of the SendRedirect call.

AResponse is the response object that the WebSnap uses to make the SendRedirect call.

AFlags indicates the attributes that must be present on the Web page module to which the current request is redirected. For example, if AFlags includes dpPublished, APageName must specify a published Web page module.

RedirectToPageName returns true if it successfully redirects to the specified page.

See Also

DispatchPageName (see page 1061)

HandleRequest

Content

WebContext (see page 6003)

11.2.50.3 IPageDispatcher Properties

The properties of the IPageDispatcher class are listed here.

Public Properties

	Name	Description
	DefaultPageName (see page 1064)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

11.2.50.3.1 IPageDispatcher.DefaultPageName Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPageName: string;
```

C++

```
__property UnicodeString DefaultPageName;
```

Description

DefaultPageName is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPageName is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a TWebAppPageModule object.

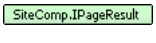
See Also

DispatchPageName (see page 1061)

TWebAppPageModule (see page 6588)

11.2.51 IPageResult Interface

Class Hierarchy



Pascal

```
IPageResult = interface;
```

C++

```
__interface IPageResult;
```

File

SiteComp

Description

This is class SiteComp.IPageResult.

11.2.51.1 IPageResult Members

The following tables list the members exposed by IPageResult.

Public Methods

	Name	Description
	DispatchPage (see page 1065)	This is DispatchPage, a member of class IPageResult.
	IncludePage (see page 1065)	This is IncludePage, a member of class IPageResult.
	RedirectToPage (see page 1066)	This is RedirectToPage, a member of class IPageResult.

11.2.51.2 IPageResult Methods

The methods of the IPageResult class are listed here.

Public Methods

	Name	Description
	DispatchPage (see page 1065)	This is DispatchPage, a member of class IPageResult.
	IncludePage (see page 1065)	This is IncludePage, a member of class IPageResult.
	RedirectToPage (see page 1066)	This is RedirectToPage, a member of class IPageResult.

11.2.51.2.1 IPageResult.DispatchPage Method

Pascal

```
function DispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchPage(const UnicodeString APageName, TWebResponse AWebResponse);
```

Description

This is DispatchPage, a member of class IPageResult.

11.2.51.2.2 IPageResult.IncludePage Method

Pascal

```
function IncludePage(const APageName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream IncludePage(const UnicodeString APageName, Boolean AOwned);
```

Description

This is IncludePage, a member of class IPageResult.

11.2.51.2.3 IPageResult.RedirectToPage Method

Pascal

```
function RedirectToPage(const APageName: string; AParams: TStrings; AWebResponse:
TWebResponse): Boolean;
```

C++

```
__fastcall Boolean RedirectToPage(const UnicodeString APageName, TStrings AParams,
TWebResponse AWebResponse);
```

Description

This is RedirectToPage, a member of class IPageResult.

11.2.52 ISessionsService Interface

ISessionsService is the interface for controlling a TSessionsService object.

Class Hierarchy

Pascal

```
ISessionsService = interface;
```

C++

```
__interface ISessionsService;
```

File

SiteComp

Description

The TSessionsService component implements ISessionsService. Most applications do not need to use this interface directly. Instead, the TWebAppComponents component uses this interface to drive the orderly management of sessions. In addition, Web session objects (TWebSession) use this interface to update the sessions service when they are terminated.

Note: In C++, method declarations that use ISessionsService use the _di_ISessionsService type instead. This type is a DelphiInterface (see page 3256) wrapper around the ISessionsService interface:

Note: typedef System::DelphiInterface (see page 3256)< ISessionsService > _di_ISessionsService;

See Also

TSessionsService (see page 6756)

TWebAppComponents (see page 6243)

TWebSession (see page 6251)

11.2.52.1 ISessionsService Members

The following tables list the members exposed by ISessionsService.

Public Methods

	Name	Description
	CheckStatusChange (see page 1067)	Checks whether any sessions have started or terminated.
	Clear (see page 1067)	Deletes all sessions from memory.
	NewSession (see page 1068)	Starts a new session.
	NotifyStatusChange (see page 1068)	Generates an OnEndSession or OnStartSession event, as apprORpriate.

11.2.52.2 ISessionsService Methods

The methods of the ISessionsService class are listed here.

Public Methods

	Name	Description
	CheckStatusChange (see page 1067)	Checks whether any sessions have started or terminated.
	Clear (see page 1067)	Deletes all sessions from memory.
	NewSession (see page 1068)	Starts a new session.
	NotifyStatusChange (see page 1068)	Generates an OnEndSession or OnStartSession event, as apprORpriate.

11.2.52.2.1 ISessionsService.CheckStatusChange Method

Checks whether any sessions have started or terminated.

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

CheckStatusChange examines the sessions currently in memory to see if any sessions have been started or terminated since the last call to NotifyStatusChange (see page 1068). If a session has started or terminated since the last call, CheckStatusChange returns true. Otherwise, it returns false.

See Also

NotifyStatusChange (see page 1068)

11.2.52.2.2 ISessionsService.Clear Method

Deletes all sessions from memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the sessions service of all the sessions it has stored in memory. Clear does not terminate active sessions in an orderly fashion (for example, it omits the call to NotifyStatusChange (see page 1068)), it merely frees all memory used to store sessions.

See Also

NotifyStatusChange (see page 1068)

11.2.52.2.3 ISessionsService.NewSession Method

Starts a new session.

Pascal

```
function NewSession: IWebSession;
```

C++

```
__fastcall IWebSession NewSession();
```

Description

Call NewSession to generate a new session. NewSession starts a new session and returns an interface that can be used to terminate that session.

See Also

NotifyStatusChange (see page 1068)

11.2.52.2.4 ISessionsService.NotifyStatusChange Method

Generates an OnEndSession or OnStartSession event, as approrpriate.

Pascal

```
procedure NotifyStatusChange;
```

C++

```
__fastcall NotifyStatusChange();
```

Description

Call NotifyStatusChange to generate the OnStartSession event for any new sessions that have been created since the last call and to generate the OnEndSession event for any sessions that terminated since the last call. For any terminated sessions, NotifyStatusChange frees the memory allocated to the session after the OnEndSession event handler exits.

See Also

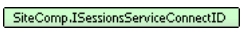
OnEndSession (see page 6755)

OnStartSession (see page 6755)

CheckStatusChange (see page 1067)

11.2.53 ISessionsServiceConnectID Interface

Class Hierarchy



Pascal

```
ISessionsServiceConnectID = interface;
```

C++

```
__interface ISessionsServiceConnectID;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServiceConnectID.

11.2.53.1 ISessionsServiceConnectID Members

The following tables list the members exposed by ISessionsServiceConnectID.

Public Methods

	Name	Description
	Connect (see page 1069)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 1069)	This is ConnectActive, a member of class ISessionsServiceConnectID.

11.2.53.2 ISessionsServiceConnectID Methods

The methods of the ISessionsServiceConnectID class are listed here.

Public Methods

	Name	Description
	Connect (see page 1069)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 1069)	This is ConnectActive, a member of class ISessionsServiceConnectID.

11.2.53.2.1 ISessionsServiceConnectID.Connect Method

Pascal

```
function Connect(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession Connect(TSessionID AID);
```

Description

This is Connect, a member of class ISessionsServiceConnectID.

11.2.53.2.2 ISessionsServiceConnectID.ConnectActive Method

Pascal

```
function ConnectActive(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession ConnectActive(TSessionID AID);
```

Description

This is ConnectActive, a member of class ISessionsServiceConnectID.

11.2.54 ISessionsServicePersist Interface

Class Hierarchy

SiteComp.ISessionsServicePersist

Pascal

```
ISessionsServicePersist = interface;
```

C++

```
__interface ISessionsServicePersist;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServicePersist.

11.2.54.1 ISessionsServicePersist Members

The following tables list the members exposed by ISessionsServicePersist.

Public Methods

	Name	Description
	LoadFromFile (see page 1070)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 1071)	This is LoadFromStream, a member of class ISessionsServicePersist.
	SaveToFile (see page 1071)	This is SaveToFile, a member of class ISessionsServicePersist.
	SaveToStream (see page 1071)	This is SaveToStream, a member of class ISessionsServicePersist.

11.2.54.2 ISessionsServicePersist Methods

The methods of the ISessionsServicePersist class are listed here.

Public Methods

	Name	Description
	LoadFromFile (see page 1070)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 1071)	This is LoadFromStream, a member of class ISessionsServicePersist.
	SaveToFile (see page 1071)	This is SaveToFile, a member of class ISessionsServicePersist.
	SaveToStream (see page 1071)	This is SaveToStream, a member of class ISessionsServicePersist.

11.2.54.2.1 ISessionsServicePersist.LoadFromFile Method

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString Filename);
```

Description

This is LoadFromFile, a member of class ISessionsServicePersist.

11.2.54.2.2 ISessionsServicePersist.LoadFromStream Method

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

This is LoadFromStream, a member of class ISessionsServicePersist.

11.2.54.2.3 ISessionsServicePersist.SaveToFile Method

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString Filename);
```

Description

This is SaveToFile, a member of class ISessionsServicePersist.

11.2.54.2.4 ISessionsServicePersist.SaveToStream Method

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

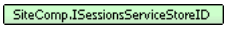
```
__fastcall SaveToStream(TStream S);
```

Description

This is SaveToStream, a member of class ISessionsServicePersist.

11.2.55 ISessionsServiceStoreID Interface

Class Hierarchy



Pascal

```
ISessionsServiceStoreID = interface;
```

C++

```
__interface ISessionsServiceStoreID;
```

File

SiteComp

Description

This is class SiteComp.ISessionsServiceStoreID.

11.2.55.1 ISessionsServiceStoreID Members

The following tables list the members exposed by ISessionsServiceStoreID.

Public Methods

	Name	Description
	ReadSessionID (see page 1072)	This is ReadSessionID, a member of class ISessionsServiceStoreID.
	WriteSessionID (see page 1072)	This is WriteSessionID, a member of class ISessionsServiceStoreID.

11.2.55.2 ISessionsServiceStoreID Methods

The methods of the ISessionsServiceStoreID class are listed here.

Public Methods

	Name	Description
	ReadSessionID (see page 1072)	This is ReadSessionID, a member of class ISessionsServiceStoreID.
	WriteSessionID (see page 1072)	This is WriteSessionID, a member of class ISessionsServiceStoreID.

11.2.55.2.1 ISessionsServiceStoreID.ReadSessionID Method

Pascal

```
function ReadSessionID(ARequest: TWebRequest; AResponse: TWebResponse; out AID: TSessionID):
Boolean;
```

C++

```
__fastcall Boolean ReadSessionID(TWebRequest ARequest, TWebResponse AResponse, TSessionID AID);
```

Description

This is ReadSessionID, a member of class ISessionsServiceStoreID.

11.2.55.2.2 ISessionsServiceStoreID.WriteSessionID Method

Pascal

```
procedure WriteSessionID(AResponse: TWebResponse; AID: TSessionID);
```

C++

```
__fastcall WriteSessionID(TWebResponse AResponse, TSessionID AID);
```

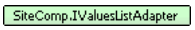
Description

This is WriteSessionID, a member of class ISessionsServiceStoreID.

11.2.56 IValuesListAdapter Interface

IValuesListAdapter is the interface for supplying name/value pairs.

Class Hierarchy



Pascal

```
IValuesListAdapter = interface;
```

C++

```
__interface IValuesListAdapter;
```

File

SiteComp

Description

Values list adapters such as TStringsValuesList and TDataSetValuesList implement the IValuesListAdapter interface. This allows adapter fields to use the values list adapter for supplying an associated list of name/value pairs. Typically, this list is used to enumerate the possible values for the adapter field so that they can be added to a selection control such as a list box in an HTML form.

The IValuesListAdapter interface introduces methods to read the current name or value, and to look up the name (or image) associated with a specified value. If the adapter that implements IValuesListAdapter does not also support iteration, server-side script can't change the current name/value pair. However, it can still look up the name or image for a specified value.

Note: In C++, method declarations that use IValuesListAdapter use the _di_IValuesListAdapter type instead. This type is a DelphiInterface (see page 3256) wrapper around the IValuesListAdapter interface:

Note: typedef System::DelphiInterface (see page 3256)< IValuesListAdapter > _di_IValuesListAdapter;

See Also

TAdapterUpdateField (see page 5419)

TCustomDataSetAdapterField

TStringsValuesList (see page 5857)

TDataSetValuesList

IIteratorSupport (see page 1054)

11.2.56.1 IValuesListAdapter Members

The following tables list the members exposed by IValuesListAdapter.

Public Methods

	Name	Description
	GetListImage (see page 1074)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1074)	Returns the image object associated with a specified value.
	GetListName (see page 1075)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1075)	Returns the name associated with a specified value.
	GetListValue (see page 1076)	Returns the value of the current name/value pair.

11.2.56.2 IValuesListAdapter Methods

The methods of the IValuesListAdapter class are listed here.

Public Methods

	Name	Description
	GetListImage (see page 1074)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1074)	Returns the image object associated with a specified value.
	GetListName (see page 1075)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1075)	Returns the name associated with a specified value.
	GetListValue (see page 1076)	Returns the value of the current name/value pair.

11.2.56.2.1 IValuesListAdapter.GetListImage Method

Returns the image object associated with the current name/value pair.

Pascal

```
function GetListImage: TComponent;
```

C++

```
__fastcall TComponent * GetListImage();
```

Description

GetListImage returns the object that represents an image associated with the current name/value pair. The image object is the component that responds to request messages which originate from an anchor tag in an HTML form that refers to the image.

If the current name/value pair does not have an associated image, GetListImage returns nil (Delphi) or NULL (C++).

Note: GetListImage provides the support for the Image property of the values list adapter, which can be addressed by server-side script.

See Also

GetListImageOfValue (see page 1074)

GetListName (see page 1075)

GetListValue (see page 1076)

TWebRequestFile

11.2.56.2.2 IValuesListAdapter.GetListImageOfValue Method

Returns the image object associated with a specified value.

Pascal

```
function GetListImageOfValue(Value: Variant): TComponent;
```

C++

```
__fastcall TComponent * GetListImageOfValue(Variant Value);
```

Description

GetListImageOfValue looks for a name/value pair with the specified value and returns the object that represents an image associated with that value. The image object is the component that responds to request messages which originate from an anchor tag in an HTML form that refers to the image.

Value is the value part of the name/value pair whose image you want.

If Value does not identify a name/value pair with an associated image, GetListImageOfValue returns nil (Delphi) or NULL (C++).

See Also

GetListImage (see page 1074)

GetListNameOfValue (see page 1075)

GetListValue (see page 1076)

TWebRequestFile

11.2.56.2.3 IValuesListAdapter.GetListName Method

Returns the name part of the current name/value pair.

Pascal

```
function GetListName: string;
```

C++

```
__fastcall UnicodeString GetListName();
```

Description

GetListName returns the name part of the current name/value pair. This method provides the support for the Name property of the values list adapter, which can be addressed by server-side script.

If the current name/value pair does not have a name, GetListName returns an empty string.

See Also

GetListNameOfValue (see page 1075)

GetListImage (see page 1074)

GetListValue (see page 1076)

11.2.56.2.4 IValuesListAdapter.GetListNameOfValue Method

Returns the name associated with a specified value.

Pascal

```
function GetListNameOfValue(Value: Variant): string;
```

C++

```
__fastcall UnicodeString GetListNameOfValue(Variant Value);
```

Description

GetListNameOfValue looks for a name/value pair with the specified value and returns the name part.

Value is the value part of the name/value pair whose name you want.

If Value does not identify a name/value pair with a name, GetListNameOfValue returns an empty string.

See Also

GetListName (see page 1075)

GetListImageOfValue (see page 1074)

GetListValue (see page 1076)

11.2.56.2.5 IValuesListAdapter.GetListValue Method

Returns the value of the current name/value pair.

Pascal

```
function GetListValue: Variant;
```

C++

```
__fastcall Variant GetListValue();
```

Description

GetListValue returns the value part of the current name/value pair. This method provides the support for the Value property of the values list adapter, which can be addressed by server-side script.

If the current name/value pair does not have a value, GetListValue returns either the name part, or an unassigned Variant (see page 3421), depending on the type of values list adapter.

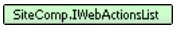
See Also

GetListName (see page 1075)

GetListImage (see page 1074)

11.2.57 IWebActionsList Interface

Class Hierarchy



Pascal

```
IWebActionsList = interface;
```

C++

```
__interface IWebActionsList;
```

File

SiteComp

Description

This is class SiteComp.IWebActionsList.

11.2.57.1 IWebActionsList Members

The following tables list the members exposed by IWebActionsList.

Public Methods

	Name	Description
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.

Public Properties

	Name	Description
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.

11.2.57.2 IWebActionsList Methods

The methods of the IWebActionsList class are listed here.

Public Methods

	Name	Description
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.

11.2.57.2.1 IWebActionsList.FindAction Method

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const UnicodeString AName);
```

Description

This is FindAction, a member of class IWebActionsList.

11.2.57.2.2 IWebActionsList.GetActionsList Method

Pascal

```
procedure GetActionsList(AList: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings AList);
```

Description

This is GetActionsList, a member of class IWebActionsList.

11.2.57.2.3 IWebActionsList.GetVisibleActions Method

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

This is GetVisibleActions, a member of class IWebActionsList.

11.2.57.2.4 IWebActionsList.IsActionInUse Method

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const UnicodeString AName);
```

Description

This is IsActionInUse, a member of class IWebActionsList.

11.2.57.3 IWebActionsList Properties

The properties of the IWebActionsList class are listed here.

Public Properties

	Name	Description
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.

11.2.57.3.1 IWebActionsList.VisibleActions Property

Pascal

```
property VisibleActions: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleActions;
```

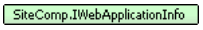
Description

This is VisibleActions, a member of class IWebActionsList.

11.2.58 IWebApplicationInfo Interface

IWebApplicationInfo is the interface for the application adapter.

Class Hierarchy



Pascal

```
IWebApplicationInfo = interface;
```

C++

```
__interface IWebApplicationInfo;
```

File

SiteComp

Description

The TApplicationAdapter component implements IWebApplicationInfo. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to obtain application. IWebApplicationInfo declares an ApplicationTitle (see page

1080) property, which allows server-side scripts to use the title of the application. It also provides access to the application adapter object, which may provide additional application-level information.

Note: In C++, method declarations that use IWebApplicationInfo use the _di_IWebApplicationInfo type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebApplicationInfo interface:

Note: typedef System::DelphiInterface (see page 3256)< IWebApplicationInfo > _di_IWebApplicationInfo;

See Also

TApplicationAdapter (see page 5439)

TCustomWebAppComponents (see page 6201)

TWebAppComponents (see page 6243)

11.2.58.1 IWebApplicationInfo Members

The following tables list the members exposed by IWebApplicationInfo.

Public Methods

	Name	Description
	GetApplicationAdapter (see page 1079)	Returns the value of the ApplicationAdapter (see page 1080) property.
	GetApplicationTitle (see page 1080)	Returns the value of the ApplicationTitle (see page 1080) property.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 1080)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
R	ApplicationTitle (see page 1080)	Indicates the title of the WebSnap application.

11.2.58.2 IWebApplicationInfo Methods

The methods of the IWebApplicationInfo class are listed here.

Public Methods

	Name	Description
	GetApplicationAdapter (see page 1079)	Returns the value of the ApplicationAdapter (see page 1080) property.
	GetApplicationTitle (see page 1080)	Returns the value of the ApplicationTitle (see page 1080) property.

11.2.58.2.1 IWebApplicationInfo.GetApplicationAdapter Method

Returns the value of the ApplicationAdapter (see page 1080) property.

Pascal

```
function GetApplicationAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetApplicationAdapter();
```

Description

GetApplicationAdapter is the read implementation of the ApplicationAdapter (see page 1080) property.

See Also

ApplicationAdapter (see page 1080)

11.2.58.2.2 IWebApplicationInfo.GetApplicationTitle Method

Returns the value of the ApplicationTitle (see page 1080) property.

Pascal

```
function GetApplicationTitle: string;
```

C++

```
__fastcall UnicodeString GetApplicationTitle();
```

Description

GetApplicationTitle is the read implementation of the ApplicationTitle (see page 1080) property.

See Also

ApplicationTitle (see page 1080)

11.2.58.3 IWebApplicationInfo Properties

The properties of the IWebApplicationInfo class are listed here.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 1080)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
R	ApplicationTitle (see page 1080)	Indicates the title of the WebSnap application.

11.2.58.3.1 IWebApplicationInfo.ApplicationAdapter Property

Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.

Pascal

```
property ApplicationAdapter: TComponent;
```

C++

```
__property TComponent * ApplicationAdapter;
```

Description

Read (see page 3634) ApplicationAdapter to access the application adapter component. This is the adapter that implements the fields and actions of the Application variable in server-side scripts. Typically, this is an instance of TApplicationAdapter.

See Also

TApplicationAdapter (see page 5439)

11.2.58.3.2 IWebApplicationInfo.ApplicationTitle Property

Indicates the title of the WebSnap application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property UnicodeString ApplicationTitle;
```

Description

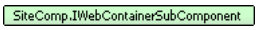
Read (see page 3634) ApplicationTitle to obtain the title of the WebSnap application. This property is the value of the Title field in the Application variable in server-side scripts.

See Also

Title (see page 5940)

11.2.59 IWebContainerSubComponent Interface

Class Hierarchy



Pascal

```
IWebContainerSubComponent = interface;
```

C++

```
__interface IWebContainerSubComponent;
```

File

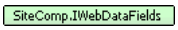
SiteComp

Description

This is class SiteComp.IWebContainerSubComponent.

11.2.60 IWebDataFields Interface

Class Hierarchy



Pascal

```
IWebDataFields = interface;
```

C++

```
__interface IWebDataFields;
```

File

SiteComp

Description

This is class SiteComp.IWebDataFields.

11.2.60.1 IWebDataFields Members

The following tables list the members exposed by IWebDataFields.

Public Methods

	Name	Description
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.

Public Properties

	Name	Description
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

11.2.60.2 IWebDataFields Methods

The methods of the IWebDataFields class are listed here.

Public Methods

	Name	Description
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.

11.2.60.2.1 IWebDataFields.FindField Method

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const UnicodeString AName);
```

Description

This is FindField, a member of class IWebDataFields.

11.2.60.2.2 IWebDataFields.GetFieldsList Method

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

This is GetFieldsList, a member of class IWebDataFields.

11.2.60.2.3 IWebDataFields.GetVisibleFields Method

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

This is GetVisibleFields, a member of class IWebDataFields.

11.2.60.2.4 IWebDataFields.IsFieldInUse Method

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const UnicodeString AName);
```

Description

This is IsFieldInUse, a member of class IWebDataFields.

11.2.60.3 IWebDataFields Properties

The properties of the IWebDataFields class are listed here.

Public Properties

	Name	Description
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

11.2.60.3.1 IWebDataFields.VisibleFields Property

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

This is VisibleFields, a member of class IWebDataFields.

11.2.61 IWebEnabled Interface

Class Hierarchy

SiteComp.IWebEnabled

Pascal

```
IWebEnabled = interface;
```

C++

```
__interface IWebEnabled;
```

File

SiteComp

Description

This is class SiteComp.IWebEnabled.

11.2.61.1 IWebEnabled Members

The following tables list the members exposed by IWebEnabled.

Public Methods

	Name	Description
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

11.2.61.2 IWebEnabled Methods

The methods of the IWebEnabled class are listed here.

Public Methods

	Name	Description
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

11.2.61.2.1 IWebEnabled.WebEnabled Method

Pascal

```
function WebEnabled: Boolean;
```

C++

```
__fastcall Boolean WebEnabled();
```

Description

This is WebEnabled, a member of class IWebEnabled.

11.2.62 IWebFormComponentContainer Interface

Class Hierarchy

SiteComp.IWebFormComponentContainer

Pascal

```
IWebFormComponentContainer = interface;
```

C++

```
__interface IWebFormComponentContainer;
```

File

SiteComp

Description

This is class SiteComp.IWebFormComponentContainer.

11.2.62.1 IWebFormComponentContainer Members

The following tables list the members exposed by IWebFormComponentContainer.

Public Methods

	Name	Description
	GetCount (see page 1085)	This is GetCount, a member of class IWebFormComponentContainer.
	GetItem (see page 1085)	This is GetItem, a member of class IWebFormComponentContainer.

Public Properties

	Name	Description
R	Count (see page 1086)	This is Count, a member of class IWebFormComponentContainer.
R	Items (see page 1086)	This is Items, a member of class IWebFormComponentContainer.

11.2.62.2 IWebFormComponentContainer Methods

The methods of the IWebFormComponentContainer class are listed here.

Public Methods

	Name	Description
	GetCount (see page 1085)	This is GetCount, a member of class IWebFormComponentContainer.
	GetItem (see page 1085)	This is GetItem, a member of class IWebFormComponentContainer.

11.2.62.2.1 IWebFormComponentContainer.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class IWebFormComponentContainer.

11.2.62.2.2 IWebFormComponentContainer.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetItem(int I);
```

Description

This is GetItem, a member of class IWebFormComponentContainer.

11.2.62.3 IWebFormComponentContainer Properties

The properties of the IWebFormComponentContainer class are listed here.

Public Properties

	Name	Description
	Count (see page 1086)	This is Count, a member of class IWebFormComponentContainer.
	Items (see page 1086)	This is Items, a member of class IWebFormComponentContainer.

11.2.62.3.1 IWebFormComponentContainer.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class IWebFormComponentContainer.

11.2.62.3.2 IWebFormComponentContainer.Items Property

Pascal

```
property Items [I: Integer]: TComponent;
```

C++

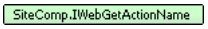
```
__property TComponent * Items[int I];
```

Description

This is Items, a member of class IWebFormComponentContainer.

11.2.63 IWebGetActionName Interface

Class Hierarchy



Pascal

```
IWebGetActionName = interface;
```

C++

```
__interface IWebGetActionName;
```

File

SiteComp

Description

This is class SiteComp.IWebGetActionName.

11.2.63.1 IWebGetActionName Members

The following tables list the members exposed by IWebGetActionName.

Public Methods

	Name	Description
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.

11.2.63.2 IWebGetActionName Methods

The methods of the IWebGetActionName class are listed here.

Public Methods

	Name	Description
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.

11.2.63.2.1 IWebGetActionName.GetActionName Method

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall UnicodeString GetActionName();
```

Description

This is GetActionName, a member of class IWebGetActionName.

11.2.63.3 IWebGetActionName Properties

The properties of the IWebGetActionName class are listed here.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.

11.2.63.3.1 IWebGetActionName.ActionName Property

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

This is ActionName, a member of class IWebGetActionName.

11.2.64 IWebGetDisplayLabel Interface

Class Hierarchy

SiteComp.IWebGetDisplayLabel

Pascal

```
IWebGetDisplayLabel = interface;
```

C++

```
__interface IWebGetDisplayLabel;
```

File

SiteComp

Description

This is class SiteComp.IWebGetDisplayLabel.

11.2.64.1 IWebGetDisplayLabel Members

The following tables list the members exposed by IWebGetDisplayLabel.

Public Methods

	Name	Description
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

11.2.64.2 IWebGetDisplayLabel Methods

The methods of the IWebGetDisplayLabel class are listed here.

Public Methods

	Name	Description
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

11.2.64.2.1 IWebGetDisplayLabel.GetDisplayLabel Method

Pascal

```
function GetDisplayLabel: string;
```

C++

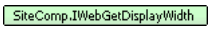
```
__fastcall UnicodeString GetDisplayLabel();
```

Description

This is GetDisplayLabel, a member of class IWebGetDisplayLabel.

11.2.65 IWebGetDisplayWidth Interface

Class Hierarchy



Pascal

```
IWebGetDisplayWidth = interface;
```

C++

```
__interface IWebGetDisplayWidth;
```

File

SiteComp

Description

This is class SiteComp.IWebGetDisplayWidth.

11.2.65.1 IWebGetDisplayWidth Members

The following tables list the members exposed by IWebGetDisplayWidth.

Public Methods

	Name	Description
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

11.2.65.2 IWebGetDisplayWidth Methods

The methods of the IWebGetDisplayWidth class are listed here.

Public Methods

	Name	Description
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

11.2.65.2.1 IWebGetDisplayWidth.GetDisplayWidth Method

Pascal

```
function GetDisplayWidth: Integer;
```

C++

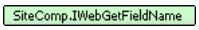
```
__fastcall int GetDisplayWidth();
```

Description

This is GetDisplayWidth, a member of class IWebGetDisplayWidth.

11.2.66 IWebGetFieldName Interface

Class Hierarchy



Pascal

```
IWebGetFieldName = interface;
```

C++

```
__interface IWebGetFieldName;
```

File

SiteComp

Description

This is class SiteComp.IWebGetFieldName.

11.2.66.1 IWebGetFieldName Members

The following tables list the members exposed by IWebGetFieldName.

Public Methods

	Name	Description
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.

Public Properties

	Name	Description
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.

11.2.66.2 IWebGetFieldName Methods

The methods of the IWebGetFieldName class are listed here.

Public Methods

	Name	Description
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.

11.2.66.2.1 IWebGetFieldName.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall UnicodeString GetFieldName();
```

Description

This is GetFieldName, a member of class IWebGetFieldName.

11.2.66.3 IWebGetFieldName Properties

The properties of the IWebGetFieldName class are listed here.

Public Properties

	Name	Description
	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.

11.2.66.3.1 IWebGetFieldName.FieldName Property

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

This is FieldName, a member of class IWebGetFieldName.

11.2.67 IWebGetFieldValue Interface

Class Hierarchy

SiteComp.IWebGetFieldValue

Pascal

```
IWebGetFieldValue = interface;
```

C++

```
__interface IWebGetFieldValue;
```

File

SiteComp

Description

This is class SiteComp.IWebGetFieldValue.

11.2.67.1 IWebGetFieldValue Members

The following tables list the members exposed by IWebGetFieldValue.

Public Methods

	Name	Description
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.

Public Properties

	Name	Description
	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.

11.2.67.2 IWebGetFieldValue Methods

The methods of the IWebGetFieldValue class are listed here.

Public Methods

	Name	Description
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.

11.2.67.2.1 IWebGetFieldValue.GetValue Method

Pascal

```
function GetValue: Variant;
```

C++

```
__fastcall Variant GetValue();
```

Description

This is GetValue, a member of class IWebGetFieldValue.

11.2.67.3 IWebGetFieldValue Properties

The properties of the IWebGetFieldValue class are listed here.

Public Properties

	Name	Description
	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.

11.2.67.3.1 IWebGetFieldValue.Value Property

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

This is Value, a member of class IWebGetFieldValue.

11.2.68 IWebGetFieldValues Interface

IWebGetFieldValues is used to iterate over multiple values associated with an adapter field.

Class Hierarchy

SiteComp.IWebGetFieldValues

Pascal

```
IWebGetFieldValues = interface;
```

C++

```
__interface IWebGetFieldValues;
```

File

SiteComp

Description

IWebGetFieldValues is implemented by adapter fields that may have zero or more associated values. Server-side script accesses these values using the adapter field's Values property.

11.2.68.1 IWebGetFieldValues Members

The following tables list the members exposed by IWebGetFieldValues.

Public Methods

	Name	Description
	EndValueIterator (see page 1093)	Indicate the iteration is finished.
	GetCurrentValue (see page 1094)	Retrieves the current value as a variant.
	HasValue (see page 1094)	Indicates whether the adapter field has a matching value.
	NextValueIteration (see page 1094)	Position the iteration pointer on the next value.
	StartValueIterator (see page 1095)	Reset (see page 3643) the iteration pointer to the first item.

11.2.68.2 IWebGetFieldValues Methods

The methods of the IWebGetFieldValues class are listed here.

Public Methods

	Name	Description
	EndValueIterator (see page 1093)	Indicate the iteration is finished.
	GetCurrentValue (see page 1094)	Retrieves the current value as a variant.
	HasValue (see page 1094)	Indicates whether the adapter field has a matching value.
	NextValueIteration (see page 1094)	Position the iteration pointer on the next value.
	StartValueIterator (see page 1095)	Reset (see page 3643) the iteration pointer to the first item.

11.2.68.2.1 IWebGetFieldValues.EndValueIterator Method

Indicate the iteration is finished.

Pascal

```
procedure EndValueIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndValueIterator(void * OwnerData);
```

Description

EndValueIterator is used with StartValueIterator (see page 1095) and NextValueIteration (see page 1094) to loop through the values associated with an adapter field. EndValueIterator should be called when iteration is complete. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

StartValueIterator (see page 1095)

NextValueIteration (see page 1094)

11.2.68.2.2 IWebGetFieldValues.GetCurrentValue Method

Retrieves the current value as a variant.

Pascal

```
function GetCurrentValue: Variant;
```

C++

```
__fastcall Variant GetCurrentValue();
```

Description

GetCurrentValue retrieves the value associated with internal iteration pointer. The FirstValueIterator and NextValueIteration (see page 1094) are called to position the internal iteration pointer. Server-side script accesses this value using the Values.Value property.

See Also

StartValueIterator (see page 1095)

NextValueIteration (see page 1094)

11.2.68.2.3 IWebGetFieldValues.HasValue Method

Indicates whether the adapter field has a matching value.

Pascal

```
function HasValue(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(const Variant AValue);
```

Description

Server-side script accesses this value using the Values.HasValue method. Server-side script to generate an HTML select list or check box group can use this method to determine whether an item should be selected or checked.

See Also

ViewMode (see page 6475)

11.2.68.2.4 IWebGetFieldValues.NextValueIteration Method

Position the iteration pointer on the next value.

Pascal

```
function NextValueIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextValueIteration(void * OwnerData);
```

Description

NextValueIteration is used with StartValueIterator (see page 1095) and EndValueIterator (see page 1093) to loop through the values associated with an adapter field. NextValueIteration returns False if there are no more values. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

StartValueIterator (see page 1095)

EndValueIterator (see page 1093)

GetCurrentValue (see page 1094)

11.2.68.2.5 IWebGetFieldValues.StartValueIterator Method

Reset (see page 3643) the iteration pointer to the first item.

Pascal

```
function StartValueIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartValueIterator(void * OwnerData);
```

Description

StartValueIterator is used with NextValueIteration (see page 1094) and EndValueIterator (see page 1093) to loop through the values associated with an adapter field. StartValueIterator returns false if there are no values. The OwnerData parameter can be used to pass a value between the three iteration functions.

See Also

NextValueIteration (see page 1094)

EndValueIterator (see page 1093)

GetCurrentValue (see page 1094)

11.2.69 IWebGetMaxLength Interface

Class Hierarchy

Pascal

```
IWebGetMaxLength = interface;
```

C++

```
__interface IWebGetMaxLength;
```

File

SiteComp

Description

This is class SiteComp.IWebGetMaxLength.

11.2.69.1 IWebGetMaxLength Members

The following tables list the members exposed by IWebGetMaxLength.

Public Methods

	Name	Description
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.

11.2.69.2 IWebGetMaxLength Methods

The methods of the IWebGetMaxLength class are listed here.

Public Methods

	Name	Description
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.

11.2.69.2.1 IWebGetMaxLength.GetMaxLength Method

Pascal

```
function GetMaxLength: integer;
```

C++

```
__fastcall int GetMaxLength();
```

Description

This is GetMaxLength, a member of class IWebGetMaxLength.

11.2.70 IWebGetSessionIDs Interface

Class Hierarchy

SiteComp.IWebGetSessionIDs

Pascal

```
IWebGetSessionIDs = interface;
```

C++

```
__interface IWebGetSessionIDs;
```

File

SiteComp

Description

This is class SiteComp.IWebGetSessionIDs.

11.2.70.1 IWebGetSessionIDs Members

The following tables list the members exposed by IWebGetSessionIDs.

Public Methods

	Name	Description
	GetSessionIDs (see page 1097)	This is GetSessionIDs, a member of class IWebGetSessionIDs.

11.2.70.2 IWebGetSessionIDs Methods

The methods of the IWebGetSessionIDs class are listed here.

Public Methods

	Name	Description
	GetSessionIDs (see page 1097)	This is GetSessionIDs, a member of class IWebGetSessionIDs.

11.2.70.2.1 IWebGetSessionIDs.GetSessionIDs Method

Pascal

```
function GetSessionIDs: IWebSessionIDs;
```

C++

```
__fastcall IWebSessionIDs GetSessionIDs();
```

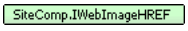
Description

This is GetSessionIDs, a member of class IWebGetSessionIDs.

11.2.71 IWebImageHREF Interface

IWebImageHREF is the interface for obtaining an HREF that addresses an image.

Class Hierarchy



Pascal

```
IWebImageHREF = interface;
```

C++

```
__interface IWebImageHREF;
```

File

SiteComp

Description

IWebImageHREF defines a single method, WebImageHREF (see page 1098), which supplies a URL for an image that should appear in HTML forms. This interface is used when server-side script addresses the Image.AsHREF property of an adapter.

Adapter field objects that represent images (that is, adapter fields with an Image property) implement IWebImageHREF so that the image reference can be obtained from server-side script. Such adapter fields include TAdapterImageField and TDataSetAdapterImageField.

Note: Method declarations that use IWebImageHREF use the _di_IWebImageHREF type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebImageHREF interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWebImageHREF >  _di_IWebImageHREF;
```

See Also

TAdapterImageField (see page 5308)

TDataSetAdapterImageField

11.2.71.1 IWebImageHREF Members

The following tables list the members exposed by IWebImageHREF.

Public Methods

	Name	Description
	WebImageHREF (see page 1098)	Retrieves a URL for an external image file.

11.2.71.2 IWebImageHREF Methods

The methods of the IWebImageHREF class are listed here.

Public Methods

	Name	Description
	WebImageHREF (see page 1098)	Retrieves a URL for an external image file.

11.2.71.2.1 IWebImageHREF.WebImageHREF Method

Retrieves a URL for an external image file.

Pascal

```
function WebImageHREF(var AHREF: string): Boolean;
```

C++

```
__fastcall Boolean WebImageHREF(UnicodeString AHREF);
```

Description

WebImageHREF is called when server-side script uses the Image.AsHREF property. WebImageHREF returns a URL that identifies a specific image that is available as a simple file reference. If images must be generated dynamically in response to requests, then WebImageHREF returns false and the adapter dispatcher generates an HREF based on the request parameters supplied by the object that implements this interface.

AHREF returns the URL that identifies the image of this field.

WebImageHREF returns true if it supplies a URL as the value of the AHREF parameter, false if the HTML form must request images using a set of request parameters instead.

See Also

EncodeAdapterItemRequestAsHREF (see page 6173)

11.2.72 IWebInputName Interface

Class Hierarchy

SiteComp.IWebInputName

Pascal

```
IWebInputName = interface;
```

C++

```
__interface IWebInputName;
```

File

SiteComp

Description

This is class SiteComp.IWebInputName.

11.2.72.1 IWebInputName Members

The following tables list the members exposed by IWebInputName.

Public Methods

	Name	Description
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.

Public Properties

	Name	Description
	InputName (see page 1100)	This is InputName, a member of class IWebInputName.

11.2.72.2 IWebInputName Methods

The methods of the IWebInputName class are listed here.

Public Methods

	Name	Description
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.

11.2.72.2.1 IWebInputName.GetInputName Method

Pascal

```
function GetInputName: string;
```

C++

```
__fastcall UnicodeString GetInputName();
```

Description

This is GetInputName, a member of class IWebInputName.

11.2.72.3 IWebInputName Properties

The properties of the IWebInputName class are listed here.

Public Properties

	Name	Description
	InputName (see page 1100)	This is InputName, a member of class IWebInputName.

11.2.72.3.1 IWebInputName.InputName Property

Pascal

```
property InputName: string;
```

C++

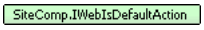
```
__property UnicodeString InputName;
```

Description

This is InputName, a member of class IWebInputName.

11.2.73 IWebIsDefaultAction Interface

Class Hierarchy



Pascal

```
IWebIsDefaultAction = interface;
```

C++

```
__interface IWebIsDefaultAction;
```

File

SiteComp

Description

This is class SiteComp.IWebIsDefaultAction.

11.2.73.1 IWebIsDefaultAction Members

The following tables list the members exposed by IWebIsDefaultAction.

Public Methods

	Name	Description
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.

11.2.73.2 IWebIsDefaultAction Methods

The methods of the IWebIsDefaultAction class are listed here.

Public Methods

	Name	Description
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.

11.2.73.2.1 IWebIsDefaultAction.IsDefaultAction Method

Pascal

```
function IsDefaultAction(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultAction(TObject * ADisplay);
```

Description

This is IsDefaultAction, a member of class IWebIsDefaultAction.

11.2.74 IWebIsDefaultField Interface

Class Hierarchy

SiteComp.IWebIsDefaultField

Pascal

```
IWebIsDefaultField = interface;
```

C++

```
__interface IWebIsDefaultField;
```

File

SiteComp

Description

This is class SiteComp.IWebIsDefaultField.

11.2.74.1 IWebIsDefaultField Members

The following tables list the members exposed by IWebIsDefaultField.

Public Methods

	Name	Description
	IsDefaultField (see page 1102)	This is IsDefaultField, a member of class IWebIsDefaultField.

11.2.74.2 IWebIsDefaultField Methods

The methods of the IWebIsDefaultField class are listed here.

Public Methods

	Name	Description
	IsDefaultField (see page 1102)	This is IsDefaultField, a member of class IWebIsDefaultField.

11.2.74.2.1 IWebIsDefaultField.IsDefaultField Method

Pascal

```
function IsDefaultField(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultField(TObject * ADisplay);
```

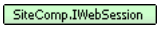
Description

This is IsDefaultField, a member of class IWebIsDefaultField.

11.2.75 IWebSession Interface

IWebSession allows an application to terminate a Web session.

Class Hierarchy



Pascal

```
IWebSession = interface;
```

C++

```
__interface IWebSession;
```

File

SiteComp

Description

Use IWebSession to terminate an active Web session. When you add a new session to a TSessionsService component using the ISessionsService (see page 1066) interface, it returns an IWebSession interface for the new session. You can use this interface to terminate the session. Once a session terminates, the application can no longer access its named values.

Note: In C++, method declarations that use IWebSession use the _di_IWebSession type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebSession interface:

Note: typedef System::DelphiInterface (see page 3256)< IWebSession > _di_IWebSession;

See Also

TWebSession (see page 6251)

TSessionsService (see page 6756)

ISessionsService (see page 1066)

11.2.75.1 IWebSession Members

The following tables list the members exposed by IWebSession.

Public Methods

	Name	Description
	Terminate (see page 1103)	Terminates the session.

11.2.75.2 IWebSession Methods

The methods of the IWebSession class are listed here.

Public Methods

	Name	Description
	Terminate (see page 1103)	Terminates the session.

11.2.75.2.1 IWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to terminate a session. Once a session terminates, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimeOutMinutes minutes.

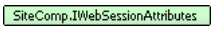
See Also

TimeoutMinutes

Values

11.2.76 IWebSessionAttributes Interface

Class Hierarchy



Pascal

```
IWebSessionAttributes = interface;
```

C++

```
__interface IWebSessionAttributes;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionAttributes.

11.2.76.1 IWebSessionAttributes Members

The following tables list the members exposed by IWebSessionAttributes.

Public Methods

	Name	Description
	GetExpirationTime (see page 1104)	This is GetExpirationTime, a member of class IWebSessionAttributes.
	GetExpired (see page 1105)	This is GetExpired, a member of class IWebSessionAttributes.
	GetIsActive (see page 1105)	This is GetIsActive, a member of class IWebSessionAttributes.
	GetStartTime (see page 1105)	This is GetStartTime, a member of class IWebSessionAttributes.
	GetTerminated (see page 1105)	This is GetTerminated, a member of class IWebSessionAttributes.
	GetTimeoutMinutes (see page 1105)	This is GetTimeoutMinutes, a member of class IWebSessionAttributes.
	GetTouchTime (see page 1106)	This is GetTouchTime, a member of class IWebSessionAttributes.
	SetTimeoutMinutes (see page 1106)	This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

Public Properties

	Name	Description
R	ExpirationTime (see page 1106)	This is ExpirationTime, a member of class IWebSessionAttributes.
R	Expired (see page 1107)	This is Expired, a member of class IWebSessionAttributes.
R	IsActive (see page 1107)	This is IsActive, a member of class IWebSessionAttributes.
R	StartTime (see page 1107)	This is StartTime, a member of class IWebSessionAttributes.
R	Terminated (see page 1107)	This is Terminated, a member of class IWebSessionAttributes.
	TimeoutMinutes (see page 1107)	This is TimeoutMinutes, a member of class IWebSessionAttributes.
R	TouchTime (see page 1108)	This is TouchTime, a member of class IWebSessionAttributes.

11.2.76.2 IWebSessionAttributes Methods

The methods of the IWebSessionAttributes class are listed here.

Public Methods

	Name	Description
	GetExpirationTime (see page 1104)	This is GetExpirationTime, a member of class IWebSessionAttributes.
	GetExpired (see page 1105)	This is GetExpired, a member of class IWebSessionAttributes.
	GetIsActive (see page 1105)	This is GetIsActive, a member of class IWebSessionAttributes.
	GetStartTime (see page 1105)	This is GetStartTime, a member of class IWebSessionAttributes.
	GetTerminated (see page 1105)	This is GetTerminated, a member of class IWebSessionAttributes.
	GetTimeoutMinutes (see page 1105)	This is GetTimeoutMinutes, a member of class IWebSessionAttributes.
	GetTouchTime (see page 1106)	This is GetTouchTime, a member of class IWebSessionAttributes.
	SetTimeoutMinutes (see page 1106)	This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

11.2.76.2.1 IWebSessionAttributes.GetExpirationTime Method

Pascal

```
function GetExpirationTime: TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime();
```

Description

This is GetExpirationTime, a member of class IWebSessionAttributes.

11.2.76.2.2 IWebSessionAttributes.GetExpired Method

Pascal

```
function GetExpired: Boolean;
```

C++

```
__fastcall Boolean GetExpired();
```

Description

This is GetExpired, a member of class IWebSessionAttributes.

11.2.76.2.3 IWebSessionAttributes.GetIsActive Method

Pascal

```
function GetIsActive: Boolean;
```

C++

```
__fastcall Boolean GetIsActive();
```

Description

This is GetIsActive, a member of class IWebSessionAttributes.

11.2.76.2.4 IWebSessionAttributes.GetStartTime Method

Pascal

```
function GetStartTime: TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime();
```

Description

This is GetStartTime, a member of class IWebSessionAttributes.

11.2.76.2.5 IWebSessionAttributes.GetTerminated Method

Pascal

```
function GetTerminated: Boolean;
```

C++

```
__fastcall Boolean GetTerminated();
```

Description

This is GetTerminated, a member of class IWebSessionAttributes.

11.2.76.2.6 IWebSessionAttributes.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes: Integer;
```

C++

```
__fastcall int GetTimeoutMinutes();
```

Description

This is GetTimeoutMinutes, a member of class IWebSessionAttributes.

11.2.76.2.7 IWebSessionAttributes.GetTouchTime Method

Pascal

```
function GetTouchTime: TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime();
```

Description

This is GetTouchTime, a member of class IWebSessionAttributes.

11.2.76.2.8 IWebSessionAttributes.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(int AValue);
```

Description

This is SetTimeoutMinutes, a member of class IWebSessionAttributes.

11.2.76.3 IWebSessionAttributes Properties

The properties of the IWebSessionAttributes class are listed here.

Public Properties

	Name	Description
R	ExpirationTime (see page 1106)	This is ExpirationTime, a member of class IWebSessionAttributes.
R	Expired (see page 1107)	This is Expired, a member of class IWebSessionAttributes.
R	IsActive (see page 1107)	This is IsActive, a member of class IWebSessionAttributes.
R	StartTime (see page 1107)	This is StartTime, a member of class IWebSessionAttributes.
R	Terminated (see page 1107)	This is Terminated, a member of class IWebSessionAttributes.
	TimeoutMinutes (see page 1107)	This is TimeoutMinutes, a member of class IWebSessionAttributes.
R	TouchTime (see page 1108)	This is TouchTime, a member of class IWebSessionAttributes.

11.2.76.3.1 IWebSessionAttributes.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class IWebSessionAttributes.

11.2.76.3.2 IWebSessionAttributes.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class IWebSessionAttributes.

11.2.76.3.3 IWebSessionAttributes.IsActive Property

Pascal

```
property IsActive: Boolean;
```

C++

```
__property Boolean IsActive;
```

Description

This is IsActive, a member of class IWebSessionAttributes.

11.2.76.3.4 IWebSessionAttributes.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class IWebSessionAttributes.

11.2.76.3.5 IWebSessionAttributes.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class IWebSessionAttributes.

11.2.76.3.6 IWebSessionAttributes.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class IWebSessionAttributes.

11.2.76.3.7 IWebSessionAttributes.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

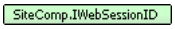
```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class IWebSessionAttributes.

11.2.77 IWebSessionID Interface

Class Hierarchy



Pascal

```
IWebSessionID = interface;
```

C++

```
__interface IWebSessionID;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionID.

11.2.77.1 IWebSessionID Members

The following tables list the members exposed by IWebSessionID.

Public Methods

	Name	Description
	GetID (see page 1109)	This is GetID, a member of class IWebSessionID.

Public Properties

	Name	Description
R	ID (see page 1109)	This is ID, a member of class IWebSessionID.

11.2.77.2 IWebSessionID Methods

The methods of the IWebSessionID class are listed here.

Public Methods

	Name	Description
	GetID (see page 1109)	This is GetID, a member of class IWebSessionID.

11.2.77.2.1 IWebSessionID.GetID Method

Pascal

```
function GetID: TSessionID;
```

C++

```
__fastcall TSessionID GetID();
```

Description

This is GetID, a member of class IWebSessionID.

11.2.77.3 IWebSessionID Properties

The properties of the IWebSessionID class are listed here.

Public Properties

	Name	Description
	ID (see page 1109)	This is ID, a member of class IWebSessionID.

11.2.77.3.1 IWebSessionID.ID Property

Pascal

```
property ID: TSessionID;
```

C++

```
__property TSessionID ID;
```

Description

This is ID, a member of class IWebSessionID.

11.2.78 IWebSessionIDs Interface

IWebSessionIDs lists the session IDs (see page 1111) for all sessions that TSessionsService stores in memory.

Class Hierarchy

SiteComp.IWebSessionIDs

Pascal

```
IWebSessionIDs = interface;
```

C++

```
__interface IWebSessionIDs;
```

File

SiteComp

Description

Use IWebSessionIDs to obtain the session IDs (see page 1111) for all the sessions in a sessions service.

Note: In C++, method declarations that use IWebSessionIDs use the _di_IWebSessionIDs type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebSessionIDs interface:

Note: typedef System::DelphiInterface (see page 3256)< IWebSessionIDs > _di_IWebSessionIDs;

See Also

TSessionsService (see page 6756)

11.2.78.1 IWebSessionIDs Members

The following tables list the members exposed by IWebSessionIDs.

Public Methods

	Name	Description
	GetID (see page 1110)	Returns the value of the IDs (see page 1111) property.
	GetIDCount (see page 1111)	Returns the value of the IDCount (see page 1111) property.

Public Properties

	Name	Description
R	IDCount (see page 1111)	Indicates the number of session IDs (see page 1111) available.
R	IDs (see page 1111)	Lists the session IDs for all the sessions stored by the sessions service.

11.2.78.2 IWebSessionIDs Methods

The methods of the IWebSessionIDs class are listed here.

Public Methods

	Name	Description
	GetID (see page 1110)	Returns the value of the IDs (see page 1111) property.
	GetIDCount (see page 1111)	Returns the value of the IDCount (see page 1111) property.

11.2.78.2.1 IWebSessionIDs.GetID Method

Returns the value of the IDs (see page 1111) property.

Pascal

```
function GetID(I: Integer): TSessionID;
```

C++

```
__fastcall TSessionID GetID(int I);
```

Description

GetID is the read implementation of the IDs (see page 1111) property.

I is the index of the session ID to return.

See Also

IDs (see page 1111)

11.2.78.2.2 IWebSessionIDs.GetIDCount Method

Returns the value of the IDCount (see page 1111) property.

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

GetIDCount is the read implementation of the IDCount (see page 1111) property.

See Also

IDCount (see page 1111)

11.2.78.3 IWebSessionIDs Properties

The properties of the IWebSessionIDs class are listed here.

Public Properties

	Name	Description
	IDCount (see page 1111)	Indicates the number of session IDs (see page 1111) available.
	IDs (see page 1111)	Lists the session IDs for all the sessions stored by the sessions service.

11.2.78.3.1 IWebSessionIDs.IDCount Property

Indicates the number of session IDs (see page 1111) available.

Pascal

```
property IDCount: Integer;
```

C++

```
__property int IDCount;
```

Description

Read (see page 3634) IDCount to obtain an upper bound for the Index parameter of the IDs (see page 1111) property. IDCount is the number of session IDs (see page 1111) (hence, the number of sessions) that the sessions service has stored in memory.

See Also

IDs (see page 1111)

11.2.78.3.2 IWebSessionIDs.IDs Property

Lists the session IDs for all the sessions stored by the sessions service.

Pascal

```
property IDs [I: Integer]: TSessionID;
```

C++

```
__property TSessionID IDs[int I];
```

Description

Use IDs to iterate over the session IDs of all the sessions that the sessions service stores.

I is the index of the session ID. It can be any value from 0 to IDCount (see page 1111) – 1.

See Also

IDCount (see page 1111)

11.2.79 IWebSessionValues Interface

Class Hierarchy

SiteComp.IWebSessionValues

Pascal

```
IWebSessionValues = interface;
```

C++

```
__interface IWebSessionValues;
```

File

SiteComp

Description

This is class SiteComp.IWebSessionValues.

11.2.79.1 IWebSessionValues Members

The following tables list the members exposed by IWebSessionValues.

Public Methods

	Name	Description
	GetItemCount (see page 1113)	This is GetItemCount, a member of class IWebSessionValues.
	GetItems (see page 1113)	This is GetItems, a member of class IWebSessionValues.
	GetValue (see page 1113)	This is GetValue, a member of class IWebSessionValues.
	SetValue (see page 1113)	This is SetValue, a member of class IWebSessionValues.

Public Properties

	Name	Description
	Values (see page 1114)	This is Values, a member of class IWebSessionValues.

11.2.79.2 IWebSessionValues Methods

The methods of the IWebSessionValues class are listed here.

Public Methods

	Name	Description
	GetItemCount (see page 1113)	This is GetItemCount, a member of class IWebSessionValues.
	GetItems (see page 1113)	This is GetItems, a member of class IWebSessionValues.

	GetValue (see page 1113)	This is GetValue, a member of class IWebSessionValues.
	SetValue (see page 1113)	This is SetValue, a member of class IWebSessionValues.

11.2.79.2.1 IWebSessionValues.GetItemCount Method

Pascal

```
function GetItemCount: Integer;
```

C++

```
__fastcall int GetItemCount();
```

Description

This is GetItemCount, a member of class IWebSessionValues.

11.2.79.2.2 IWebSessionValues.GetItems Method

Pascal

```
procedure GetItems(var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class IWebSessionValues.

11.2.79.2.3 IWebSessionValues.GetValue Method

Pascal

```
function GetValue(const AName: string): Variant;
```

C++

```
__fastcall Variant GetValue(const UnicodeString AName);
```

Description

This is GetValue, a member of class IWebSessionValues.

11.2.79.2.4 IWebSessionValues.SetValue Method

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

This is SetValue, a member of class IWebSessionValues.

11.2.79.3 IWebSessionValues Properties

The properties of the IWebSessionValues class are listed here.

Public Properties

	Name	Description
	Values (see page 1114)	This is Values, a member of class IWebSessionValues.

11.2.79.3.1 IWebSessionValues.Values Property

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[UnicodeString const AName];
```

Description

This is Values, a member of class IWebSessionValues.

11.2.80 IWebSetActionName Interface

Class Hierarchy

SiteComp.IWebSetActionName

Pascal

```
IWebSetActionName = interface;
```

C++

```
__interface IWebSetActionName;
```

File

SiteComp

Description

This is class SiteComp.IWebSetActionName.

11.2.80.1 IWebSetActionName Members

The following tables list the members exposed by IWebSetActionName.

Public Methods

	Name	Description
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.

11.2.80.2 IWebSetActionName Methods

The methods of the IWebSetActionName class are listed here.

Public Methods

	Name	Description
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.

11.2.80.2.1 IWebSetActionName.SetActionName Method

Pascal

```
procedure SetActionName(const AFieldName: string);
```

C++

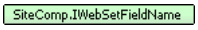
```
__fastcall SetActionName(const UnicodeString AFieldName);
```

Description

This is SetActionName, a member of class IWebSetActionName.

11.2.81 IWebSetFieldName Interface

Class Hierarchy



Pascal

```
IWebSetFieldName = interface;
```

C++

```
__interface IWebSetFieldName;
```

File

SiteComp

Description

This is class SiteComp.IWebSetFieldName.

11.2.81.1 IWebSetFieldName Members

The following tables list the members exposed by IWebSetFieldName.

Public Methods

	Name	Description
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

11.2.81.2 IWebSetFieldName Methods

The methods of the IWebSetFieldName class are listed here.

Public Methods

	Name	Description
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

11.2.81.2.1 IWebSetFieldName.SetFieldName Method

Pascal

```
procedure SetFieldName(const AFieldName: string);
```

C++

```
__fastcall SetFieldName(const UnicodeString AFieldName);
```

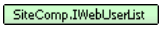
Description

This is SetFieldName, a member of class IWebSetFieldName.

11.2.82 IWebUserList Interface

IWebUserList is the interface for validating user logins and access rights.

Class Hierarchy



Pascal

```
IWebUserList = interface;
```

C++

```
__interface IWebUserList;
```

File

SiteComp

Description

The TWebUserList component implements IWebUserList. This interface, which the TWebAppComponents object publishes, allows the WebSnap application to check whether a user supplied the correct password when logging in and whether the user has rights to perform the operation requested in an HTTP request message.

Note: In C++, method declarations that use IWebUserList use the _di_IWebUserList type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebUserList interface:

Note: typedef System::DelphiInterface (see page 3256)< IWebUserList > _di_IWebUserList;

See Also

TWebUserList (see page 6796)

TCustomWebAppComponents (see page 6201)

TWebAppComponents (see page 6243)

11.2.82.1 IWebUserList Members

The following tables list the members exposed by IWebUserList.

Public Methods

	Name	Description
	CheckAccessRights (see page 1117)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 1117)	Validates a user login.

11.2.82.2 IWebUserList Methods

The methods of the IWebUserList class are listed here.

Public Methods

	Name	Description
	CheckAccessRights (see page 1117)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 1117)	Validates a user login.

11.2.82.2.1 IWebUserList.CheckAccessRights Method

Checks whether a specified user has a given access rights string.

Pascal

```
function CheckAccessRights(UserID: Variant; Rights: string): Boolean;
```

C++

```
__fastcall Boolean CheckAccessRights(Variant UserID, UnicodeString Rights);
```

Description

Call CheckAccessRights to determine whether the current end user is authorized to view Web pages or adapters whose access rights string is Rights. The end user must be logged in before CheckAccessRights can determine the user's access rights.

UserID is the user ID of the user to check.

Rights is the access rights string that CheckAccessRights must look for. Typically, this is the value of the ViewRights property on a Web page information object that describes the page the user wants to access or on an adapter that the user wants to call.

See Also

HasViewRights (see page 5510)

HasModifyRights (see page 5509)

HasExecuteRights (see page 5509)

ViewAccess

ViewAccess (see page 5574)

11.2.82.2.2 IWebUserList.ValidateUser Method

Validates a user login.

Pascal

```
function ValidateUser(Strings: TStrings): Variant;
```

C++

```
__fastcall Variant ValidateUser(TStrings Strings);
```

Description

Call ValidateUser to check whether the information a user supplies on login matches a user in the user list.

Strings is a string list that lists the information that the user supplied when logging in. Each entry in the string list is of the form Name=Value. Typically, the names of strings in the list are 'UserName' and 'Password'.

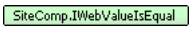
If the login information is valid, ValidateUser returns the UserID of the identified user. Otherwise, ValidateUser returns an unassigned Variant (see page 3421).

See Also

CheckAccessRights (see page 1117)

11.2.83 IWebValueIsEqual Interface

Class Hierarchy



Pascal

```
IWebValueIsEqual = interface;
```

C++

```
__interface IWebValueIsEqual;
```

File

SiteComp

Description

This is class SiteComp.IWebValueIsEqual.

11.2.83.1 IWebValueIsEqual Members

The following tables list the members exposed by IWebValueIsEqual.

Public Methods

	Name	Description
	IsEqual (see page 1119)	This is IsEqual, a member of class IWebValueIsEqual.

11.2.83.2 IWebValueIsEqual Methods

The methods of the IWebValueIsEqual class are listed here.

Public Methods

	Name	Description
	IsEqual (see page 1119)	This is IsEqual, a member of class IWebValueIsEqual.

11.2.83.2.1 IWebValueIsEqual.IsEqual Method

Pascal

```
function IsEqual(const Value: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(const Variant Value);
```

Description

This is IsEqual, a member of class IWebValueIsEqual.

11.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FullyQualifiedFieldName (see page 1120)	This is function SiteComp.FullyQualifiedFieldName.
	FullyQualifiedName (see page 1120)	This is function SiteComp.FullyQualifiedName.

11.3.1 SiteComp.FullyQualifiedFieldName Function

Pascal

```
function FullyQualifiedFieldName(AComponent: TComponent): string;
```

C++

```
UnicodeString FullyQualifiedFieldName(TComponent * AComponent);
```

File

SiteComp

Description

This is function SiteComp.FullyQualifiedFieldName.

11.3.2 SiteComp.FullyQualifiedName Function

Pascal

```
function FullyQualifiedName(AComponent: TComponent; AOption: TQualifyOption): string;
```

C++

```
UnicodeString FullyQualifiedName(TComponent * AComponent, TQualifyOption AOption);
```

File

SiteComp

Description

This is function SiteComp.FullyQualifiedName.

11.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAdapterActionHTMLElementType (see page 1121)	TAdapterActionHTMLElementType identifies the HTML type that is generated to let a user execute an adapter action.
	TAdapterDisplayCharacteristic (see page 1122)	TAdapterDisplayCharacteristics (see page 1128) describes the characteristics of the HTML generated to display information from an adapter.
	TAdapterDisplayHTMLElementType (see page 1123)	TAdapterDisplayHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TAdapterDisplayViewModeType (see page 1123)	TAdapterDisplayViewModeType identifies how an HTML control for the associated adapter field should behave.
	TAdapterInputHTMLElementType (see page 1124)	TAdapterInputHTMLElementType identifies an HTML input type that can be used to edit the value(s) associated with an adapter field.
	TAdapterItemRequestParamOption (see page 1125)	TAdapterItemRequestParamOptions (see page 1128) identifies what parameter values to include when encoding the hidden fields and HREF values that represent calls to adapter fields and actions.
	TDispatchPageFlag (see page 1126)	TDiapatchPageFlags indicates the attributes that must be present on a Web page module for it to handle a request.
	TQualifyOption (see page 1126)	This is record SiteComp.TQualifyOption.

11.4.1 SiteComp.TAdapterActionHTMLElementType Enumeration

TAdapterActionHTMLElementType identifies the HTML type that is generated to let a user execute an adapter action.

Pascal

```
TAdapterActionHTMLElementType = (
  htmlaButton,
  htmlaImage,
  htmlaAnchor,
  htmlaEventImages
);
```

C++

```
enum TAdapterActionHTMLElementType {
  htmlaButton,
  htmlaImage,
  htmlaAnchor,
  htmlaEventImages
};
```

File

SiteComp

Description

Each element type indicates a different HTML element. TAdapterActionHTMLElementType defines executable HTML elements.

The following table lists the possible values:

Value	Meaning
htmlaButton	Push button control
htmlaImage	Image control that user can click.
htmlaAnchor	Anchor tag (hypertext link).
htmlaEventImages	Multiple images that change based on associated script.

See Also

TAdapterInputHTMLElementType (see page 1124)

11.4.2 SiteComp.TAdapterDisplayCharacteristic Enumeration

TAdapterDisplayCharacteristics (see page 1128) describes the characteristics of the HTML generated to display information from an adapter.

Pascal

```
TAdapterDisplayCharacteristic = (
  dcMultipleRecordView,
  dcCurrentRecordView,
  dcChangeCurrentRecordView
);
```

C++

```
enum TAdapterDisplayCharacteristic {
  dcMultipleRecordView,
  dcCurrentRecordView,
  dcChangeCurrentRecordView
};
```

File

SiteComp

Description

The components that generate HTML to represent the fields or actions of an adapter return a TAdapterDisplayCharacteristics (see page 1128) value to indicate what types of actions are appropriate to accompany the generated HTML. TAdapterDisplayCharacteristics (see page 1128) is a set of TAdapterDisplayCharacteristic values.

The following table lists the TAdapterDisplayCharacteristic values:

Value	Appropriate Actions
dcMultipleRecordView	Actions that operate on or display multiple records are appropriate.
dcCurrentRecordView	Actions that operate on the current adapter record are appropriate.
dcChangeCurrentRecordView	Actions that change the current record are appropriate.

11.4.3 SiteComp.TAdapterDisplayHTMLElementType Enumeration

TAdapterDisplayHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.

Pascal

```
TAdapterDisplayHTMLElementType = (
  htmldText,
  htmldImage,
  htmldList,
  htmldNone
);
```

C++

```
enum TAdapterDisplayHTMLElementType {
  htmldText,
  htmldImage,
  htmldList,
  htmldNone
};
```

File

SiteComp

Description

Each element type indicates a different HTML element. TAdapterDisplayHTMLElementType defines read-only HTML elements. TAdapterInputHTMLElementType (see page 1124) defines editable HTML elements.

The following table lists the possible values:

Value	Meaning
htmldText	Display as a text block.
htmldImage	Display as an image.
htmldList	Display as a list of values.
htmldNone	No display type specified.

See Also

GetDisplayStyleType (see page 5347)

TAdapterInputHTMLElementType (see page 1124)

11.4.4 SiteComp.TAdapterDisplayViewModeType Enumeration

TAdapterDisplayViewModeType identifies how an HTML control for the associated adapter field should behave.

Pascal

```
TAdapterDisplayViewModeType = (
  htmlvmInput,
  htmlvmDisplay,
  htmlvmToggleOnAccess,
  htmlvmNone
);
```

C++

```
enum TAdapterDisplayViewModeType {
  htmlvmInput,
  htmlvmDisplay,
  htmlvmToggleOnAccess,
  htmlvmNone
};
```

File

SiteComp

Description

Each view mode type indicates how an HTML element is to be used. The following table lists the possible values:

Value	Meaning
htmlvmInput	The HTML control is used for editing values.
htmlvmDisplay	The HTML control is used for read-only display.
htmlvmToggleOnAccess	The HTML control is used for editing if the user has modify rights, otherwise for read-only display.
htmlvmNone	No view mode type specified.

See Also

TAdapterNamedDisplayField (see page 5343)

11.4.5 SiteComp.TAdapterInputHTMLElementType Enumeration

TAdapterInputHTMLElementType identifies an HTML input type that can be used to edit the value(s) associated with an adapter field.

Pascal

```
TAdapterInputHTMLElementType = (
  htmliNone,
  htmliTextInput,
  htmliPasswordInput,
  htmliSelect,
  htmliSelectMultiple,
  htmliRadio,
  htmliCheckBox,
  htmliTextArea,
  htmliFile
);
```

C++

```
enum TAdapterInputHTMLElementType {
  htmliNone,
  htmliTextInput,
  htmliPasswordInput,
  htmliSelect,
  htmliSelectMultiple,
  htmliRadio,
  htmliCheckBox,
  htmliTextArea,
  htmliFile
};
```

File

SiteComp

Description

Each element type indicates a different HTML element. TAdapterDisplayHTMLElementType (see page 1123) defines read-only HTML elements. TAdapterInputHTMLElementType defines editable HTML elements.

The following table lists the possible values:

Value	Meaning
htmliNone	No type specified
htmliTextInput	Single line input element
htmliPasswordInput	Single line password input element
htmliSelect	Single selection list
htmliSelectMultiple	Multiple select list
htmliRadio	Group of radio buttons
htmliCheckBox	Group of check boxes
htmliTextArea	Multi-line input element
htmliFile	File upload element

See Also

GetInputStyleType (see page 5347)

TAdapterDisplayHTMLElementType (see page 1123)

11.4.6 SiteComp.TAdapterItemRequestParamOption Enumeration

TAdapterItemRequestParamOptions (see page 1128) identifies what parameter values to include when encoding the hidden fields and HREF values that represent calls to adapter fields and actions.

Pascal

```
TAdapterItemRequestParamOption = (
  roExcludeID
);
```

C++

```
enum TAdapterItemRequestParamOption {
  roExcludeID
};
```

File

SiteComp

Description

TAdapterItemRequestParamOptions (see page 1128) is the set of flags that an adapter dispatcher uses when encoding parameter values for a call to an adapter action. It represents zero or more TAdapterItemRequestParamOption values.

The following table lists the possible values:

Value	Meaning
roExcludeID	Do not include a parameter that identifies the action or image field that is the target of the request.

11.4.7 SiteComp.TDispatchPageFlag Enumeration

TDiapatchPageFlags indicates the attributes that must be present on a Web page module for it to handle a request.

Pascal

```
TDispatchPageFlag = (
  dpPublished
);
```

C++

```
enum TDispatchPageFlag {
  dpPublished
};
```

File

SiteComp

Description

TDispatchPageFlags (see page 1129) is a set of TDispatchPageFlag values. This type is used to supply flags to methods that dispatch request messages to a page module for handling. The following table lists the possible values:

Option	Meaning
dpPublished	The request can only be dispatched to the Web page module if it is published.

See Also

IPageDispatcher (see page 1060)

11.4.8 SiteComp.TQualifyOption Enumeration

Pascal

```
TQualifyOption = (
```

```
    qaUseModulesVar,
    qaNoModulesVar,
    qaNoModuleName
  );
```

C++

```
  enum TQualifyOption {
    qaUseModulesVar,
    qaNoModulesVar,
    qaNoModuleName
  };
```

File

SiteComp

Description

This is record SiteComp.TQualifyOption.

11.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAdapterDisplayCharacteristics (see page 1128)	This is type SiteComp.TAdapterDisplayCharacteristics.
TAdapterItemRequestParamOptions (see page 1128)	This is type SiteComp.TAdapterItemRequestParamOptions.
TDispatchPageFlags (see page 1129)	This is type SiteComp.TDispatchPageFlags.
TSessionID (see page 1129)	TSessionID is the type of the session identifier in a WebSnap application.

11.5.1 SiteComp.TAdapterDisplayCharacteristics Type

Pascal

```
TAdapterDisplayCharacteristics = set of TAdapterDisplayCharacteristic;
```

C++

```
TAdapterDisplayCharacteristic TAdapterDisplayCharacteristics;
```

File

SiteComp

Description

This is type SiteComp.TAdapterDisplayCharacteristics.

11.5.2 SiteComp.TAdapterItemRequestParamOptions Type

Pascal

```
TAdapterItemRequestParamOptions = set of TAdapterItemRequestParamOption;
```

C++

```
TAdapterItemRequestParamOption TAdapterItemRequestParamOptions;
```

File

SiteComp

Description

This is type SiteComp.TAdapterItemRequestParamOptions.

11.5.3 SiteComp.TDispatchPageFlags Type

Pascal

```
TDispatchPageFlags = set of TDispatchPageFlag;
```

C++

```
TDispatchPageFlag TDispatchPageFlags;
```

File

SiteComp

Description

This is type SiteComp.TDispatchPageFlags.

11.5.4 SiteComp.TSessionID Type

TSessionID is the type of the session identifier in a WebSnap application.

Pascal

```
TSessionID = type string;
```

C++

```
type string TSessionID;
```

File

SiteComp

Description

TSessionID is the type that TSessionsService uses to uniquely identify the sessions in a WebSnap application.

See Also

TWebSession (see page 6251)

TInvalidWebSession (see page 6216)

TSessionsService (see page 6756)

11.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ExecuteScript (see page 1130)	This is variable SiteComp.ExecuteScript.
GetHTMLSampleImage (see page 1130)	This is variable SiteComp.GetHTMLSampleImage.
IDEFileManager (see page 1131)	This is variable SiteComp.IDEFileManager.
InetFileTypes (see page 1131)	This is variable SiteComp.InetFileTypes.
InternetEnvOptions (see page 1131)	This is variable SiteComp.InternetEnvOptions.
ScriptDebugger (see page 1132)	This is variable SiteComp.ScriptDebugger.

11.6.1 SiteComp.ExecuteScript Variable

Pascal

```
ExecuteScript: IInternetExecuteScript = nil;
```

C++

```
IInternetExecuteScript ExecuteScript = nil;
```

File

SiteComp

Description

This is variable SiteComp.ExecuteScript.

11.6.2 SiteComp.GetHTMLSampleImage Variable

Pascal

```
GetHTMLSampleImage: function: string = nil;
```

C++

```
function: string GetHTMLSampleImage = nil;
```

File

SiteComp

Description

This is variable SiteComp.GetHTMLSampleImage.

11.6.3 SiteComp.IDEFileManager Variable

Pascal

```
IDEFileManager: IIDEFileManager = nil;
```

C++

```
IIDEFileManager IDEFileManager = nil;
```

File

SiteComp

Description

This is variable SiteComp.IDEFileManager.

11.6.4 SiteComp.InetFileTypes Variable

Pascal

```
InetFileTypes: IInternetFileTypes = nil;
```

C++

```
IInternetFileTypes InetFileTypes = nil;
```

File

SiteComp

Description

This is variable SiteComp.InetFileTypes.

11.6.5 SiteComp.InternetEnvOptions Variable

Pascal

```
InternetEnvOptions: IInternetEnvOptions = nil;
```

C++

```
IInternetEnvOptions InternetEnvOptions = nil;
```

File

SiteComp

Description

This is variable SiteComp.InternetEnvOptions.

11.6.6 SiteComp.ScriptDebugger Variable

Pascal

```
ScriptDebugger: IInternetScriptDebugger = nil;
```

C++

```
IInternetScriptDebugger ScriptDebugger = nil;
```

File

SiteComp

Description

This is variable SiteComp.ScriptDebugger.

11.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
AdapterActionHTMLElementTypeNames (see page 1133)	This is constant SiteComp.AdapterActionHTMLElementTypeNames.
AdapterDisplayHTMLElementTypeNames (see page 1133)	This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.
AdapterDisplayViewModeTypeNames (see page 1134)	This is constant SiteComp.AdapterDisplayViewModeTypeNames.
AdapterInputHTMLElementTypeNames (see page 1134)	This is constant SiteComp.AdapterInputHTMLElementTypeNames.

11.7.1 SiteComp.AdapterActionHTMLElementTypeNames Constant

Pascal

```
AdapterActionHTMLElementTypeNames: array[TAdapterActionHTMLElementType] of string = ('Button',
'Image', 'Anchor', 'EventImages');
```

C++

```
array[TAdapterActionHTMLElementType] of string AdapterActionHTMLElementTypeNames = ('Button',
'Image', 'Anchor', 'EventImages');
```

File

SiteComp

Description

This is constant SiteComp.AdapterActionHTMLElementTypeNames.

11.7.2 SiteComp.AdapterDisplayHTMLElementTypeNames Constant

Pascal

```
AdapterDisplayHTMLElementTypeNames: array[TAdapterDisplayHTMLElementType] of string = ('Text',
'Image', 'List', '');
```

C++

```
array[TAdapterDisplayHTMLElementType] of string AdapterDisplayHTMLElementTypeNames = ('Text',
'Image', 'List', '');
```

File

SiteComp

Description

This is constant SiteComp.AdapterDisplayHTMLElementTypeNames.

11.7.3 SiteComp.AdapterDisplayViewModeTypeNames Constant

Pascal

```
AdapterDisplayViewModeTypeNames: array[TAdapterDisplayViewModeType] of string = ('Input',
'Display', 'ToggleOnAccess', '');
```

C++

```
array[TAdapterDisplayViewModeType] of string AdapterDisplayViewModeTypeNames = ('Input',
'Display', 'ToggleOnAccess', '');
```

File

SiteComp

Description

This is constant SiteComp.AdapterDisplayViewModeTypeNames.

11.7.4 SiteComp.AdapterInputHTMLElementTypeNames Constant

Pascal

```
AdapterInputHTMLElementTypeNames: array[TAdapterInputHTMLElementType] of string = ('',
'TextInput', 'PasswordInput', 'Select', 'SelectMultiple', 'Radio', 'CheckBox', 'TextArea',
'File');
```

C++

```
array[TAdapterInputHTMLElementType] of string AdapterInputHTMLElementTypeNames = ('',
'TextInput', 'PasswordInput', 'Select', 'SelectMultiple', 'Radio', 'CheckBox', 'TextArea',
'File');
```

File

SiteComp

Description

This is constant SiteComp.AdapterInputHTMLElementTypeNames.

12 SiteProd Namespace

This is namespace SiteProd.

Topics

Name	Description
Classes (see page 1136)	The following table lists classes in this documentation.

Classes

	Name	Description
	TAdapterPageProducer (see page 1136)	TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.
	TBaseAdapterPageProducer (see page 1143)	TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TCustomAdapterPageProducer (see page 1159)	TCustomAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TSitePageElements (see page 1167)	TSitePageElements stores part of the content that page producers generate.

12.1 Classes

The following table lists classes in this documentation.

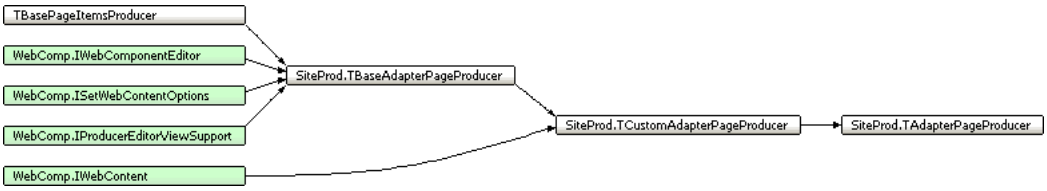
Classes

	Name	Description
	TAdapterPageProducer (see page 1136)	TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.
	TBaseAdapterPageProducer (see page 1143)	TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TCustomAdapterPageProducer (see page 1159)	TCustomAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.
	TSitePageElements (see page 1167)	TSitePageElements stores part of the content that page producers generate.

12.1.1 TAdapterPageProducer Class

TAdapterPageProducer generates the proper HTML and server-side script needed to display adapter fields and execute adapter actions.

Class Hierarchy



Pascal

```
TAdapterPageProducer = class(TCustomAdapterPageProducer);
```

C++

```
class TAdapterPageProducer : public TCustomAdapterPageProducer;
```

File

SiteProd

Description

TAdapterPageProducer creates an HTML page using content it obtains from a hierarchy of components called Web items. Add components to this hierarchy to define HTML forms and tables. For example, to create an HTML form with a grid, add a TAdapterForm (see page 6404) and TAdapterGrid (see page 6419) components. The Web items that appear in the TAdapterPageProducer's hierarchy use Adapter components to fetch the information that they display.

Use the Web Page Module wizard to create a web page that uses the TAdapterPageProducer. You can select TAdapterPageProducer from the drop-down list at the top of the Web Page Module wizard, or add a TAdapterPageProducer component to an existing web page modue.

Web page modules that use the TAdapterPageProducer provide an editor view called HTML Script. Use this view to see the HTML Script that TAdapterPageProducer generates. The script generated by TAdapterPageProducer is not editable. However, you can copy the generated script from the HTML Script view to a new Web Page Module. You can then edit the copy of the

script in the code editor or with an external HTML editor that supports embedded script tags (for example <% %>).

TAdapterPageProducer uses a template that contains the following special tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1140) or StylesFile (see page 1141) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with the generated HTML/JavaScript. The TAdapterPageProducer will not generate any content if this tag is missing.

See Also

TAdapter (see page 5190)

TAdapterForm (see page 6404)

TAdapterGrid (see page 6419)

Page Producers

Server-side Scripting in WebSnap

12.1.1.1 TAdapterPageProducer Members

The following tables list the members exposed by TAdapterPageProducer.

Protected Methods

	Name	Description
	CanAddClass (see page 1146)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1147)	Creates the object that implements the PageElements (see page 1157) property.
V	DeclareStyles (see page 1148)	Returns an HTML Styles (see page 1157) tag that declares the styles specified by the Styles (see page 1157) or StylesFile (see page 1158) property.
V	DeclareWarnings (see page 1148)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1148)	Generates an OnAfterGetContent (see page 1158) event.
	DoBeforeGetContent (see page 1149)	Generates an OnBeforeGetContent (see page 1159) event.
V	DoTagEvent (see page 1149)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1150)	Fills the PageElements (see page 1157) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1150)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1150)	Returns the default template that the page producer can use to generate content.
V	GetTemplateFileName (see page 1162)	Returns the value of the HTMLFile (see page 1166) property.
	GetTemplateFileType (see page 1152)	Returns the type of file that can store the templates this object uses to generate its content.
V	GetTemplateStream (see page 1163)	Returns a stream from which to read the HTML template that this producer uses.
	GetXMLData (see page 1152)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1152)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1153)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1153)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1154)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1154)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1154)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.

	ImplCanAddClass (see page 1155)	Provides the underlying implementation for the CanAddClass (see page 1146) method.
	SetHTMLDoc (see page 1163)	Sets the HTMLDoc (see page 1165) property.
	SetHTMLFile (see page 1164)	Sets the HTMLFile (see page 1166) property.
	SetWebContentOptions (see page 1156)	Sets the options that limit how the page producer creates its generated HTML.
	WebContent (see page 1164)	Returns the HTML document assembled by the Adapter page producer.

Protected Properties

	Name	Description
	PageElements (see page 1157)	Stores information that the Adapter page producer uses when generating its content.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.
	Content (see page 6104)	Returns the HTML that the Web item generates.
	ContentFromStream (see page 1147)	Returns the content produced using a template read from a stream.
	Create (see page 1145)	Creates and initializes an instance of TBaseAdapterPageProducer (see page 1143).
	Create (see page 1161)	Creates and initializes an instance of TCustomAdapterPageProducer (see page 1159).
	Destroy (see page 1145)	Frees an instance of TBaseAdapterPageProducer (see page 1143).
	Destroy (see page 1162)	Frees an instance of TCustomAdapterPageProducer (see page 1159).
	GetRequiredTags (see page 1151)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 1151)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	SetStyles (see page 1155)	Assigns a value to the Styles (see page 1157) property.
	SetStylesFile (see page 1156)	Assigns a value to the StylesFile (see page 1158) property.
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

Published Events

	Name	Description
	OnAfterGetContent (see page 1142)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1142)	Occurs before the page producer generates its content.

Published Properties

	Name	Description
	HTMLDoc (see page 1139)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1140)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Styles (see page 1140)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1141)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 1141)	Maintains a list of components that generate HTML content for the producer.

12.1.1.2 TAdapterPageProducer Properties

The properties of the TAdapterPageProducer class are listed here.

Published Properties

	Name	Description
	HTMLDoc (see page 1139)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1140)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Styles (see page 1140)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1141)	Specifies a file containing the individual styles for the page producer's style sheet.
	WebPageItems (see page 1141)	Maintains a list of components that generate HTML content for the producer.

12.1.1.2.1 TAdapterPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value that the Content method returns.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomAdapterPageProducer automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles or StylesFile property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer will not generate any content if this tag is missing.

If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile property.

See Also

Content

HTMLFile (see page 1166)

OnHTMLTag

12.1.1.2.2 TAdapterPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content

HTMLDoc (see page 1165)

OnHTMLTag

12.1.1.2.3 TAdapterPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseAdapterPageProducer. Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile property instead. If StylesFile is defined, it takes precedence over the Styles property.

See Also

StylesFile (see page 1158)

Content

12.1.1.2.4 TAdapterPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles property instead.

See Also

Styles (see page 1157)

Content

12.1.1.2.5 TAdapterPageProducer.WebPageItems Property

Maintains a list of components that generate HTML content for the producer.

Pascal

```
property WebPageItems;
```

C++

```
__property WebPageItems;
```

Description

WebPageItems lists the components that generate content for the page items producer. Each of these components supports the IWebContent interface, and can contain its own list of Web items that help it generate its content.

The type of items that can be added to the WebPageItems list are restricted by a protected method that is abstract or, in C++ terminology, pure virtual in TBasePageItemsProducer. Descendant classes implement this protected method to indicate what types of items can be added to the list.

At design time, use the Web page editor to add components to the WebPageItems property.

Note: Typically, WebPageItems lists all the components that generate content for the page items producer, in the order that the generated content appears in the final HTML document. However, the page items producer can transform COMPONENT tags in its template using components that are not in this list, as long as they have the same owner as the page items producer..

See Also

IWebContent (see page 6103)

Creating Web Pages with an InternetExpress Page Producer

12.1.1.3 TAdapterPageProducer Events

The events of the TAdapterPageProducer class are listed here.

Published Events

	Name	Description
	OnAfterGetContent (see page 1142)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1142)	Occurs before the page producer generates its content.

12.1.1.3.1 TAdapterPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its content.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write an OnAfterGetContent event handler to take specific action immediately after the producer generates its content.

See Also

ContentFromStream (see page 1147)

OnBeforeGetContent (see page 1159)

12.1.1.3.2 TAdapterPageProducer.OnBeforeGetContent Event

Occurs before the page producer generates its content.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write an OnBeforeGetContent event handler to take specific action immediately before the producer generates its content.

See Also

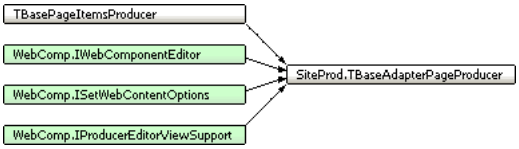
ContentFromStream (see page 1147)

OnAfterGetContent (see page 1158)

12.1.2 TBaseAdapterPageProducer Class

TBaseAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.

Class Hierarchy



Pascal

```
TBaseAdapterPageProducer = class(TBasePageItemsProducer, IWebComponentEditor,
ISetWebContentOptions, IProducerEditorViewSupport);
```

C++

```
class TBaseAdapterPageProducer : public TBasePageItemsProducer, public IWebComponentEditor,
public ISetWebContentOptions, public IProducerEditorViewSupport;
```

File

SiteProd

Description

TBaseAdapterPageProducer creates an HTML page using content it obtains from a hierarchy of components called Web items. The Web items that appear in the TBaseAdapterPageProducer's hierarchy use Adapter components to fetch the information that they display.

Use TBaseAdapterPageProducer as a base class for page producer components that assemble their content from Web items that use Adapter components. TBaseAdapterPageProducer implements the logic for converting an HTML template into content it assembles from its hierarchy of Web items.

TBaseAdapterPageProducer implements the logic to translate the following special HTML-transparent tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1157) or StylesFile (see page 1158) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items.

12.1.2.1 TBaseAdapterPageProducer Members

The following tables list the members exposed by TBaseAdapterPageProducer.

Protected Methods

	Name	Description
	CanAddClass (see page 1146)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1147)	Creates the object that implements the PageElements (see page 1157) property.
V	DeclareStyles (see page 1148)	Returns an HTML Styles (see page 1157) tag that declares the styles specified by the Styles (see page 1157) or StylesFile (see page 1158) property.
V	DeclareWarnings (see page 1148)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1148)	Generates an OnAfterGetContent (see page 1158) event.
	DoBeforeGetContent (see page 1149)	Generates an OnBeforeGetContent (see page 1159) event.
V	DoTagEvent (see page 1149)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1150)	Fills the PageElements (see page 1157) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1150)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1150)	Returns the default template that the page producer can use to generate content.
	GetTemplateFileType (see page 1152)	Returns the type of file that can store the templates this object uses to generate its content.
	GetXMLData (see page 1152)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1152)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1153)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1153)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1154)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1154)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1154)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 1155)	Provides the underlying implementation for the CanAddClass (see page 1146) method.
	SetWebContentOptions (see page 1156)	Sets the options that limit how the page producer creates its generated HTML.

Protected Properties

	Name	Description
R	PageElements (see page 1157)	Stores information that the Adapter page producer uses when generating its content.

Public Events

	Name	Description
	OnAfterGetContent (see page 1158)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1159)	Occurs before the page producer generates its content.

Public Methods

	Name	Description
V	ContentFromStream (see page 1147)	Returns the content produced using a template read from a stream.
V	Create (see page 1145)	Creates and initializes an instance of TBaseAdapterPageProducer (see page 1143).
V	Destroy (see page 1145)	Frees an instance of TBaseAdapterPageProducer (see page 1143).
	GetRequiredTags (see page 1151)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.

	GetScriptEngines (see page 1151)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	SetStyles (see page 1155)	Assigns a value to the Styles (see page 1157) property.
	SetStylesFile (see page 1156)	Assigns a value to the StylesFile (see page 1158) property.

Public Properties

	Name	Description
	Styles (see page 1157)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1158)	Specifies a file containing the individual styles for the page producer's style sheet.

12.1.2.2 TBaseAdapterPageProducer.Create Constructor

Creates and initializes an instance of TBaseAdapterPageProducer (see page 1143).

Pascal

```
constructor Create(AComponent: TComponent); override;
```

C++

```
virtual __fastcall TBaseAdapterPageProducer(TComponent * AComponent);
```

Description

Call Create to instantiate an adapter page producer at runtime. Adapter page producers placed in a Web page module at design time (or by the Web page module wizard) are instantiated automatically.

AOwner is a component, typically a Web page module, that becomes the value of the Owner property. The Owner is responsible for freeing the TBaseAdapterPageProducer (see page 1143) instance.

See Also

Owner

12.1.2.3 TBaseAdapterPageProducer.Destroy Destructor

Frees an instance of TBaseAdapterPageProducer (see page 1143).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseAdapterPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

12.1.2.4 TBaseAdapterPageProducer Methods

The methods of the TBaseAdapterPageProducer class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 1146)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1147)	Creates the object that implements the PageElements (see page 1157) property.
V	DeclareStyles (see page 1148)	Returns an HTML Styles (see page 1157) tag that declares the styles specified by the Styles (see page 1157) or StylesFile (see page 1158) property.
V	DeclareWarnings (see page 1148)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1148)	Generates an OnAfterGetContent (see page 1158) event.
	DoBeforeGetContent (see page 1149)	Generates an OnBeforeGetContent (see page 1159) event.
V	DoTagEvent (see page 1149)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1150)	Fills the PageElements (see page 1157) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1150)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1150)	Returns the default template that the page producer can use to generate content.
	GetTemplateFileType (see page 1152)	Returns the type of file that can store the templates this object uses to generate its content.
	GetXMLData (see page 1152)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1152)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1153)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1153)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1154)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1154)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1154)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 1155)	Provides the underlying implementation for the CanAddClass (see page 1146) method.
	SetWebContentOptions (see page 1156)	Sets the options that limit how the page producer creates its generated HTML.

Public Methods

	Name	Description
V	ContentFromStream (see page 1147)	Returns the content produced using a template read from a stream.
	GetRequiredTags (see page 1151)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 1151)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	SetStyles (see page 1155)	Assigns a value to the Styles (see page 1157) property.
	SetStylesFile (see page 1156)	Assigns a value to the StylesFile (see page 1158) property.

12.1.2.4.1 TBaseAdapterPageProducer.CanAddClass Method

Specifies whether a Web item of a specified type can be added to the WebPageItems property list.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The IDE calls CanAddClass to determine what Web items can be added to supply content for the TBaseAdapterPageProducer (see page 1143) descendant. CanAddClass calls the virtual ImplCanAddClass (see page 1155) method to determine whether a class of Web item is valid.

AParent is the TBaseAdapterPageProducer (see page 1143) descendant to whose WebPageItems property the Web item will be added.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the TBaseAdapterPageProducer (see page 1143) descendant can use content generated by a component of type AClass, false otherwise.

See Also

ImplCanAddClass (see page 1155)

WebPageItems

12.1.2.4.2 TBaseAdapterPageProducer.ContentFromStream Method

Returns the content produced using a template read from a stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; override;
```

C++

```
virtual __fastcall UnicodeString ContentFromStream(TStream InStream);
```

Description

Call ContentFromStream to retrieve the content produced by the adapter page producer based on the template provided by the Stream parameter. ContentFromStream:

1. Generates an OnBeforeGetContent (see page 1159) event.
2. Translates the HTML template that is the value of Stream, using information produced by the items listed in the WebPageItems property.
3. Generates an OnAfterGetContent (see page 1158) event.
4. Returns the HTML produced in step 2.

See Also

Content

OnBeforeGetContent (see page 1159)

WebPageItems

OnAfterGetContent (see page 1158)

12.1.2.4.3 TBaseAdapterPageProducer.CreatePageElements Method

Creates the object that implements the PageElements (see page 1157) property.

Pascal

```
function CreatePageElements: TSitePageElements; virtual;
```

C++

```
virtual __fastcall TSitePageElements CreatePageElements();
```

Description

CreatePageElements creates the object that is used as the value of the PageElements (see page 1157) property. This object holds temporary information that the page producer uses to assemble its content.

See Also

PageElements (see page 1157)

GeneratePageElements (see page 1150)

12.1.2.4.4 TBaseAdapterPageProducer.DeclareStyles Method

Returns an HTML Styles (see page 1157) tag that declares the styles specified by the Styles (see page 1157) or StylesFile (see page 1158) property.

Pascal

```
function DeclareStyles: string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareStyles();
```

Description

The GeneratePageElements (see page 1150) method calls DeclareStyles to generate the HTML that declares a stylesheet in the generated HTML page. DeclareStyles uses the StylesFile (see page 1158) property to determine the styles it declares. If StylesFile (see page 1158) is an empty string, DeclareStyles uses the Styles (see page 1157) property instead. If both StylesFile (see page 1158) and Styles (see page 1157) are set to empty strings, DeclareStyles returns an empty string.

See Also

Styles (see page 1157)

StylesFile (see page 1158)

GeneratePageElements (see page 1150)

12.1.2.4.5 TBaseAdapterPageProducer.DeclareWarnings Method

Returns the formatted warnings that appear at design time to indicate the status of content generation.

Pascal

```
function DeclareWarnings(Warnings: TStrings): string; virtual;
```

C++

```
virtual __fastcall UnicodeString DeclareWarnings(TStrings Warnings);
```

Description

The GeneratePageElements (see page 1150) method calls DeclareWarnings to generate the text that describes any design-time warnings. These warnings indicate possible problems with the hierarchy of web items.

See Also

GeneratePageElements (see page 1150)

WebPageItems

12.1.2.4.6 TBaseAdapterPageProducer.DoAfterGetContent Method

Generates an OnAfterGetContent (see page 1158) event.

Pascal

```
procedure DoAfterGetContent;
```

C++

```
__fastcall DoAfterGetContent();
```

Description

TBaseAdapterPageProducer (see page 1143) calls DoAfterGetContent internally to generate the OnAfterGetContent (see page 1158) event.

See Also

OnAfterGetContent (see page 1158)

12.1.2.4.7 TBaseAdapterPageProducer.DoBeforeGetContent Method

Generates an OnBeforeGetContent (see page 1159) event.

Pascal

```
procedure DoBeforeGetContent;
```

C++

```
__fastcall DoBeforeGetContent();
```

Description

TBaseAdapterPageProducer (see page 1143) calls DoBeforeGetContent internally to generate the OnBeforeGetContent (see page 1159) event.

See Also

OnBeforeGetContent (see page 1159)

12.1.2.4.8 TBaseAdapterPageProducer.DoTagEvent Method

Translates HTML-transparent tags in the template.

Pascal

```
procedure DoTagEvent(Tag: TTag; const TagString: string; TagParams: TStrings; var ReplaceText:
string); override;
```

C++

```
virtual __fastcall DoTagEvent(TTag Tag, const UnicodeString TagString, TStrings TagParams,
UnicodeString ReplaceText);
```

Description

When the page producer encounters an HTML-transparent tag in its template, it calls DoTagEvent to translate that tag into the appropriate value. TBaseAdapterPageProducer (see page 1143) overrides the inherited method to handle the <#STYLES>, <#WARNINGS>, and <#SERVERSCRIPT> tags. For any other tags, DoTagEvent generates an OnHTMLTag event.

Tag is the type of HTML-transparent tag that was encountered in the HTML template.

TagString is tag name of the HTML-transparent tag. This is the first string that follows the pound sign (#).

TagParams gives the parameter portion of the HTML-transparent tag. Each string in the TStrings object has the form ParamName=Value.

ReplaceText returns the translated string that should replace the HTML-transparent tag.

See Also

OnHTMLTag

12.1.2.4.9 TBaseAdapterPageProducer.GeneratePageElements Method

Fills the PageElements (see page 1157) property with information based on the other properties of the adapter Page Producer.

Pascal

```
procedure GeneratePageElements; virtual;
```

C++

```
virtual __fastcall GeneratePageElements();
```

Description

GeneratePageElements assembles information about the styles, warnings, server script, server functions, and server variables that the adapter page producer uses to generate its content. It writes this information into the members of the object that is the value of the PageElements (see page 1157) property.

See Also

PageElements (see page 1157)

12.1.2.4.10 TBaseAdapterPageProducer.GetContentOptions Method

Returns an object that influences how the page producer generates its content.

Pascal

```
function GetContentOptions(var Owned: Boolean): TWebContentOptions; override;
```

C++

```
virtual __fastcall TWebContentOptions GetContentOptions(Boolean Owned);
```

Description

TBaseAdapterPageProducer (see page 1143) calls GetContentOptions internally to look up the options that limit what can be included in the generated HTML.

Owned returns a value that indicates whether the return value should be freed by the caller. If Owned is true, the caller should free the TWebContentOptions (see page 6076) object after it has used it.

See Also

SetWebContentOptions (see page 1156)

12.1.2.4.11 TBaseAdapterPageProducer.GetDefaultTemplate Method

Returns the default template that the page producer can use to generate content.

Pascal

```
function GetDefaultTemplate: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultTemplate();
```

Description

GetDefaultTemplate returns the default template that the page producer can use to generate its content. This template has the form:

```
<HTML>
<HEAD>
```

```
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

TBaseAdapterPageProducer (see page 1143) automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1157) or StylesFile (see page 1158) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items.

See Also

Styles (see page 1157)

StylesFile (see page 1158)

12.1.2.4.12 TBaseAdapterPageProducer.GetRequiredTags Method

Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.

Pascal

```
class function GetRequiredTags: string;
```

C++

```
__fastcall UnicodeString GetRequiredTags();
```

Description

GetRequiredTags returns a string containing the special tags that the page producer expects to find for generating its content. This method returns the string '<#STYLES> <#WARNINGS> <#SERVERSCRIPT>'.

See Also

ContentFromStream (see page 1147)

12.1.2.4.13 TBaseAdapterPageProducer.GetScriptEngines Method

Returns a list of script engines that can work with server-side scripts that produce content for the page producer.

Pascal

```
class procedure GetScriptEngines(S: TStrings);
```

C++

```
__fastcall GetScriptEngines(TStrings S);
```

Description

GetScriptEngines fills an existing string list with the set of all values that can be assigned to the ScriptEngine property. These determine the script engine that works with associated server-side scripts.

This method adds a single string, 'JSCRIPT', to the string list passed as the S parameter.

See Also

ScriptEngine

12.1.2.4.14 TBaseAdapterPageProducer.GetTemplateFileType Method

Returns the type of file that can store the templates this object uses to generate its content.

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

GetTemplateFileType returns the string 'HTML', because Adapter page producers generate their contents based on an HTML template that includes HTML-transparent tags.

See Also

GetDefaultTemplate (see page 1150)

12.1.2.4.15 TBaseAdapterPageProducer.GetXMLData Method

Returns a stream for reading any XML data associated with the page producer.

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

The IDE calls GetXMLData for producers that indicate that they support an XML browser view (that is, if the HasXMLBrowserView (see page 1154) method returns true). In TBaseAdapterPageProducer (see page 1143), GetXMLData always returns nil (Delphi) or NULL (C++), because adapter page producers do not use associated XML data.

Owned returns an indication of whether the caller needs to free the returned stream. In this case, this parameter is irrelevant, because GetXMLData always returns nil (Delphi) or NULL (C++).

See Also

HasXMLBrowserView (see page 1154)

12.1.2.4.16 TBaseAdapterPageProducer.GetXSLData Method

Returns a stream for reading an XSL template associated with the page producer.

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

The IDE calls GetXSLData for producers that indicate that they support an XSL browser view (that is, if the HasXSLBrowserView (see page 1154) method returns true). In TBaseAdapterPageProducer (see page 1143), GetXSLData always returns nil (Delphi) or NULL (C++), because adapter page producers do not use an associated XSL template.

Owned returns an indication of whether the caller needs to free the returned stream. In this case, this parameter is irrelevant,

because GetXSLData always returns nil (Delphi) or NULL (C++).

See Also

HasXSLBrowserView (see page 1154)

12.1.2.4.17 TBaseAdapterPageProducer.HandleScriptTag Method

Translates HTML-transparent tags for an associated server-side script.

Pascal

```
function HandleScriptTag(const TagString: string; TagParams: TStrings; var ReplaceString:
string): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandleScriptTag(const UnicodeString TagString, TStrings TagParams,
UnicodeString ReplaceString);
```

Description

When a server-side script encounters an HTML-transparent tag, it calls HandleScriptTag to allow the page producer to translate that tag for it. HandleScriptTag translates the <#SERVERSCRIPT> tag, supplying the content produced by the hierarchy of Web items that is the value of the WebPageItems property.

TagString is tag name of the HTML-transparent tag. This is the first string that follows the pound sign (#).

TagParams gives the parameter portion of the HTML-transparent tag. Each string in the TStrings object has the form ParamName=Value.

ReplaceText returns the translated string that should replace the HTML-transparent tag.

HandleScriptTag returns true if it can translate the tag (that is, if TagString is 'SERVERSCRIPT'). It returns false if TagString identifies any other type of tag.

See Also

WebPageItems

12.1.2.4.18 TBaseAdapterPageProducer.HasHTMLBrowserView Method

Indicates whether the IDE should display an HTML browser view for the producer's content.

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

The IDE calls HasHTMLBrowserView to determine whether it should create an HTML tab page in the code editor for displaying the content of the page producer.

HasHTMLBrowserView returns true. As a result, at design time, the value that the Content method returns appears on the HTML tab page of the code editor when the adapter page producer is selected.

See Also

HasScriptView (see page 1154)

HasXMLBrowserView (see page 1154)

HasXSLBrowserView (see page 1154)

Content

12.1.2.4.19 TBaseAdapterPageProducer.HasScriptView Method

Indicates whether the IDE should display an associated server-side script.

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

The IDE calls HasScriptView to determine whether it should create a script tab page in the code editor for displaying the server-side script associated with the page producer.

HasScriptView returns true. As a result, at design time, the associated server-side script appears on a tab page of the code editor when the adapter page producer is selected.

See Also

HasHTMLBrowserView (see page 1153)

HasXMLBrowserView (see page 1154)

HasXSLBrowserView (see page 1154)

12.1.2.4.20 TBaseAdapterPageProducer.HasXMLBrowserView Method

Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

The IDE calls HasXMLBrowserView to determine whether it should create an XML tab page in the code editor for displaying the XML data that the page producer uses.

HasXMLBrowserView returns false because adapter page producers do not use associated XML data.

See Also

HasScriptView (see page 1154)

HasHTMLBrowserView (see page 1153)

HasXSLBrowserView (see page 1154)

GetXMLData (see page 1152)

12.1.2.4.21 TBaseAdapterPageProducer.HasXSLBrowserView Method

Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

The IDE calls HasXSLBrowserView to determine whether it should create an XSL tab page in the code editor for displaying the XSL template that the page producer uses.

HasXSLBrowserView returns false because adapter page producers do not use associated XSL templates.

See Also

HasScriptView (see page 1154)

HasHTMLBrowserView (see page 1153)

HasXMLBrowserView (see page 1154)

GetXSLData (see page 1152)

12.1.2.4.22 TBaseAdapterPageProducer.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 1146) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 1146) method calls ImpleCanAddClass to determine whether the TBaseAdapterPageProducer (see page 1143) descendant can add a component of type AClass to its WebPageItems property list. AParent is the TBaseAdapterPageProducer (see page 1143) instance to which the component will be added.

In TBaseAdapterPageProducer (see page 1143), ImplCanAddClass returns true if AClass descends from TCustomAdapterForm (see page 6490) or TCustomLayoutGroup.

See Also

CanAddClass (see page 1146)

WebPageItems

TCustomAdapterForm (see page 6490)

TCustomLayoutGroup

12.1.2.4.23 TBaseAdapterPageProducer.SetStyles Method

Assigns a value to the Styles (see page 1157) property.

Pascal

```
procedure SetStyles(Value: TStrings);
```

C++

```
__fastcall SetStyles(TStrings Value);
```

Description

SetStyles is the write implementation of the Styles (see page 1157) property.

See Also

Styles (see page 1157)

12.1.2.4.24 TBaseAdapterPageProducer.SetStylesFile Method

Assigns a value to the StylesFile (see page 1158) property.

Pascal

```
procedure SetStylesFile(const Value: TFileName);
```

C++

```
__fastcall SetStylesFile(const TFileName Value);
```

Description

SetStylesFile is the write implementation of the StylesFile (see page 1158) property.

See Also

StylesFile (see page 1158)

12.1.2.4.25 TBaseAdapterPageProducer.SetWebContentOptions Method

Sets the options that limit how the page producer creates its generated HTML.

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

SetWebContentOptions assigns a set of flags to the adapter page producer that limit what it can include in generated HTML.

AOptions is the TWebContentOptions (see page 6076) object that represents the set of options for this page producer. TBaseAdapterPageProducer (see page 1143) takes ownership of the object, so the caller should not free it.

See Also

GetContentOptions (see page 1150)

12.1.2.5 TBaseAdapterPageProducer Properties

The properties of the TBaseAdapterPageProducer class are listed here.

Protected Properties

	Name	Description
	PageElements (see page 1157)	Stores information that the Adapter page producer uses when generating its content.

Public Properties

	Name	Description
	Styles (see page 1157)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1158)	Specifies a file containing the individual styles for the page producer's style sheet.

12.1.2.5.1 TBaseAdapterPageProducer.PageElements Property

Stores information that the Adapter page producer uses when generating its content.

Pascal

```
property PageElements: TSitePageElements;
```

C++

```
__property TSitePageElements PageElements;
```

Description

TBaseAdapterPageProducer (see page 1143) uses PageElements internally to store styles, warnings, server script, server functions, and server variables. It uses the properties of PageElements to assemble its content.

See Also

Content

OnHTMLTag

12.1.2.5.2 TBaseAdapterPageProducer.Styles Property

Defines the styles that can be used by the page producer and its Web items.

Pascal

```
property Styles: TStrings;
```

C++

```
__property TStrings Styles;
```

Description

Use Styles to define a style sheet for the HTML document produced by TBaseAdapterPageProducer (see page 1143). Each string in the TStrings object specifies a specific style by giving a style selector followed by a set of attributes in curly braces. These definitions can define styles for standard HTML elements such as

```
H2 B  {color: red}
```

or they can define styles that you name, such as

```
.MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
```

Note that user-defined style names must begin with a dot.

Styles for standard elements are applied automatically to HTML that uses the tag or tags that form the style selector. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector.

Note: If you are sharing a set of styles between multiple page producers or multiple applications, you can define the styles in a file. To define the styles in a file, use the StylesFile (see page 1158) property instead. If StylesFile (see page 1158) is defined, it takes precedence over the Styles property.

See Also

StylesFile (see page 1158)

Content

12.1.2.5.3 TBaseAdapterPageProducer.StylesFile Property

Specifies a file containing the individual styles for the page producer's style sheet.

Pascal

```
property StylesFile: TFileName;
```

C++

```
__property TFileName StylesFile;
```

Description

Set StylesFile to the name of a file containing the style definitions used by the page producer. The page producer uses these style definitions to declare a style sheet in the HTML document it generates.

Styles (see page 1157) for standard elements are applied automatically to HTML in the document. User-defined styles can be applied by setting the StyleRule property of a Web item to the user-defined style name that forms the selector of a style definition.

Note: If the StylesFile property is not set, you can specify the style definitions using the Styles (see page 1157) property instead.

See Also

Styles (see page 1157)

Content

12.1.2.6 TBaseAdapterPageProducer Events

The events of the TBaseAdapterPageProducer class are listed here.

Public Events

	Name	Description
	OnAfterGetContent (see page 1158)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1159)	Occurs before the page producer generates its content.

12.1.2.6.1 TBaseAdapterPageProducer.OnAfterGetContent Event

Occurs after the page producer generates its content.

Pascal

```
property OnAfterGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAfterGetContent;
```

Description

Write (see page 3707) an OnAfterGetContent event handler to take specific action immediately after the producer generates its content.

See Also

ContentFromStream (see page 1147)

OnBeforeGetContent (see page 1159)

12.1.2.6.2 TBaseAdapterPageProducer.OnBeforeGetContent Event

Occurs before the page producer generates its content.

Pascal

```
property OnBeforeGetContent: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnBeforeGetContent;
```

Description

Write (see page 3707) an OnBeforeGetContent event handler to take specific action immediately before the producer generates its content.

See Also

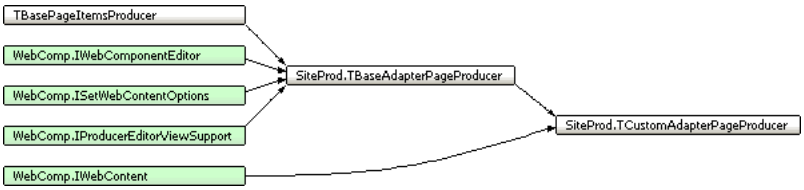
ContentFromStream (see page 1147)

OnAfterGetContent (see page 1158)

12.1.3 TCustomAdapterPageProducer Class

TCustomAdapterPageProducer generates the proper HTML and JavaScript needed to display adapter fields and execute adapter actions.

Class Hierarchy



Pascal

```
TCustomAdapterPageProducer = class(TBaseAdapterPageProducer, IWebContent);
```

C++

```
class TCustomAdapterPageProducer : public TBaseAdapterPageProducer, public IWebContent;
```

File

SiteProd

Description

TCustomAdapterPageProducer creates an HTML page using content it obtains from a hierarchy of components called Web items. The Web items that appear in the TCustomAdapterPageProducer's hierarchy use Adapter components to fetch the information that they display.

Use TCustomAdapterPageProducer as a base class for page producer components that assemble their content from Web items

that use Adapter components.

TCustomAdapterPageProducer introduces two properties, HTMLDoc (see page 1165) and HTMLFile (see page 1166), each of which can specify a template that the page producer uses to generate its content. By default, this template is set to include the following special tags:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1157) or StylesFile (see page 1158) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer will not generate any content if this tag is missing.

Applications can alter the template to further embellish the content that the page producer generates.

12.1.3.1 TCustomAdapterPageProducer Members

The following tables list the members exposed by TCustomAdapterPageProducer.

Protected Methods

	Name	Description
	CanAddClass (see page 1146)	Specifies whether a Web item of a specified type can be added to the WebPageItems property list.
V	CreatePageElements (see page 1147)	Creates the object that implements the PageElements (see page 1157) property.
V	DeclareStyles (see page 1148)	Returns an HTML Styles (see page 1157) tag that declares the styles specified by the Styles (see page 1157) or StylesFile (see page 1158) property.
V	DeclareWarnings (see page 1148)	Returns the formatted warnings that appear at design time to indicate the status of content generation.
	DoAfterGetContent (see page 1148)	Generates an OnAfterGetContent (see page 1158) event.
	DoBeforeGetContent (see page 1149)	Generates an OnBeforeGetContent (see page 1159) event.
V	DoTagEvent (see page 1149)	Translates HTML-transparent tags in the template.
V	GeneratePageElements (see page 1150)	Fills the PageElements (see page 1157) property with information based on the other properties of the adapter Page Producer.
V	GetContentOptions (see page 1150)	Returns an object that influences how the page producer generates its content.
V	GetDefaultTemplate (see page 1150)	Returns the default template that the page producer can use to generate content.
V	GetTemplateFileName (see page 1162)	Returns the value of the HTMLFile (see page 1166) property.
	GetTemplateFileType (see page 1152)	Returns the type of file that can store the templates this object uses to generate its content.
V	GetTemplateStream (see page 1163)	Returns a stream from which to read the HTML template that this producer uses.
	GetXMLData (see page 1152)	Returns a stream for reading any XML data associated with the page producer.
	GetXSLData (see page 1152)	Returns a stream for reading an XSL template associated with the page producer.
V	HandleScriptTag (see page 1153)	Translates HTML-transparent tags for an associated server-side script.
	HasHTMLBrowserView (see page 1153)	Indicates whether the IDE should display an HTML browser view for the producer's content.
	HasScriptView (see page 1154)	Indicates whether the IDE should display an associated server-side script.
	HasXMLBrowserView (see page 1154)	Indicates whether the IDE should display an XML tree view for any XML data associated with the page producer.
	HasXSLBrowserView (see page 1154)	Indicates whether the IDE should display an XSL tree view for an XSL template associated with the page producer.
V	ImplCanAddClass (see page 1155)	Provides the underlying implementation for the CanAddClass (see page 1146) method.
	SetHTMLDoc (see page 1163)	Sets the HTMLDoc (see page 1165) property.
	SetHTMLFile (see page 1164)	Sets the HTMLFile (see page 1166) property.
	SetWebContentOptions (see page 1156)	Sets the options that limit how the page producer creates its generated HTML.
	WebContent (see page 1164)	Returns the HTML document assembled by the Adapter page producer.

Protected Properties

	Name	Description
	PageElements (see page 1157)	Stores information that the Adapter page producer uses when generating its content.

Public Events

	Name	Description
	OnAfterGetContent (see page 1158)	Occurs after the page producer generates its content.
	OnBeforeGetContent (see page 1159)	Occurs before the page producer generates its content.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.
	Content (see page 6104)	Returns the HTML that the Web item generates.
	ContentFromStream (see page 1147)	Returns the content produced using a template read from a stream.
	Create (see page 1161)	Creates and initializes an instance of TCustomAdapterPageProducer (see page 1159).
	Destroy (see page 1162)	Frees an instance of TCustomAdapterPageProducer (see page 1159).
	GetRequiredTags (see page 1151)	Returns the HTML-transparent tags that the adapter page producer requires to have present in its template.
	GetScriptEngines (see page 1151)	Returns a list of script engines that can work with server-side scripts that produce content for the page producer.
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	SetStyles (see page 1155)	Assigns a value to the Styles (see page 1157) property.
	SetStylesFile (see page 1156)	Assigns a value to the StylesFile (see page 1158) property.
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

Public Properties

	Name	Description
	HTMLDoc (see page 1165)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1166)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.
	Styles (see page 1157)	Defines the styles that can be used by the page producer and its Web items.
	StylesFile (see page 1158)	Specifies a file containing the individual styles for the page producer's style sheet.

12.1.3.2 TCustomAdapterPageProducer.Create Constructor

Creates and initializes an instance of TCustomAdapterPageProducer (see page 1159).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterPageProducer(TComponent * AOwner);
```

Description

Call Create to instantiate an adapter page producer at runtime. Adapter page producers placed in a Web page module at design time (or by the Web page module wizard) are instantiated automatically.

AOwner is a component, typically a Web page module, that becomes the value of the Owner property. The Owner is responsible for freeing the TCustomAdapterPageProducer (see page 1159) instance.

See Also

Owner

12.1.3.3 TCustomAdapterPageProducer.Destroy Destructor

Frees an instance of TCustomAdapterPageProducer (see page 1159).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterPageProducer();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

12.1.3.4 TCustomAdapterPageProducer Methods

The methods of the TCustomAdapterPageProducer class are listed here.

Protected Methods

	Name	Description
	GetTemplateFileName (see page 1162)	Returns the value of the HTMLFile (see page 1166) property.
	GetTemplateStream (see page 1163)	Returns a stream from which to read the HTML template that this producer uses.
	SetHTMLDoc (see page 1163)	Sets the HTMLDoc (see page 1165) property.
	SetHTMLFile (see page 1164)	Sets the HTMLFile (see page 1166) property.
	WebContent (see page 1164)	Returns the HTML document assembled by the Adapter page producer.

12.1.3.4.1 TCustomAdapterPageProducer.GetTemplateFileName Method

Returns the value of the HTMLFile (see page 1166) property.

Pascal

```
function GetTemplateFileName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetTemplateFileName();
```

Description

TBasePageProducer introduces the protected GetTemplateFileName method to obtain the name of a file that contains a template from which to produce content. Each page producer component overrides GetTemplateFileName to supply this template in a class-specific way. TCustomAdapterPageProducer (see page 1159) returns the value of the HTMLFile (see page 1166) property.

See Also

GetTemplateStream (see page 1163)

HTMLFile (see page 1166)

12.1.3.4.2 TCustomAdapterPageProducer.GetTemplateStream Method

Returns a stream from which to read the HTML template that this producer uses.

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream; override;
```

C++

```
virtual __fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

TBasePageProducer introduces the protected GetTemplateStream method to obtain a stream from which to read the producer component's HTML template. Each page producer component overrides GetTemplateStream to supply this stream in a class-specific way.

AOwned returns a value that indicates whether the caller should free the returned stream after reading the template. When AOwned is true, the caller should free the stream object.

TCustomAdapterPageProducer (see page 1159) returns a stream from which the template specified by the HTMLFile (see page 1166) or HTMLDoc (see page 1165) property can be read.

See Also

GetTemplateFileName (see page 1162)

HTMLFile (see page 1166)

HTMLDoc (see page 1165)

12.1.3.4.3 TCustomAdapterPageProducer.SetHTMLDoc Method

Sets the HTMLDoc (see page 1165) property.

Pascal

```
procedure SetHTMLDoc(const Value: TStrings);
```

C++

```
__fastcall SetHTMLDoc(const TStrings Value);
```

Description

SetHTMLDoc is the protected write implementation of the HTMLDoc (see page 1165) property.

Value is a TStrings descendant that contains the new HTML template. SetHTMLDoc assigns the strings in this object to the TStrings object that implements the HTMLDoc (see page 1165) property. Note that this method does not take ownership of the specified object.

See Also

HTMLDoc (see page 1165)

Assign

12.1.3.4.4 TCustomAdapterPageProducer.SetHTMLFile Method

Sets the HTMLFile (see page 1166) property.

Pascal

```
procedure SetHTMLFile(const Value: TFileName);
```

C++

```
__fastcall SetHTMLFile(const TFileName Value);
```

Description

SetHTMLFile is the protected write implementation of the HTMLFile (see page 1166) property.

Value is the name of the template that the page producer should use to generate its content.

See Also

HTMLFile (see page 1166)

12.1.3.4.5 TCustomAdapterPageProducer.WebContent Method

Returns the HTML document assembled by the Adapter page producer.

Pascal

```
function WebContent(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall UnicodeString WebContent(TWebContentOptions Options, TLayout Layout);
```

Description

Call WebContent to get the HTML document produced by TCustomAdapterPageProducer (see page 1159). TCustomAdapterPageProducer (see page 1159) uses WebContent to implement the Content method of its IWebContent (see page 6103) interface.

Options contains a set of flags that limit what can be included in the generated HTML.

Layout is ignored by the WebContent method. It is included as a parameter for compatibility with other content producers that use a helper object to coordinate the layout of content produced by multiple Web items.

The Adapter page producer creates its content by replacing HTML-transparent tags in the template specified by HTMLDoc (see page 1165) or HTMLFile (see page 1166).

By default, this template has the form

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Although the template can be changed to include any combination of these tags with custom HTML or tags that the application translates in an OnHTMLTag event handler.

The WebContent method translates the tags listed above as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1157) or StylesFile (see page 1158) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items specified by the WebPageItems property.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer (see page 1159) will not generate any content if this tag is missing.

The string returned by WebContent can be used as the Content property of the response to an HTTP request message.

See Also

IWebContent (see page 6103)

Styles (see page 1157)

StylesFile (see page 1158)

OnHTMLTag

HTMLDoc (see page 1165)

HTMLFile (see page 1166)

WebPageItems

12.1.3.5 TCustomAdapterPageProducer Properties

The properties of the TCustomAdapterPageProducer class are listed here.

Public Properties

	Name	Description
	HTMLDoc (see page 1165)	Specifies the HTML template used to generate the value that the Content method returns.
	HTMLFile (see page 1166)	Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

12.1.3.5.1 TCustomAdapterPageProducer.HTMLDoc Property

Specifies the HTML template used to generate the value that the Content method returns.

Pascal

```
property HTMLDoc: TStrings;
```

C++

```
__property TStrings HTMLDoc;
```

Description

Set HTMLDoc to a set of strings that make up the HTML template. By default, HTMLDoc is initialized to the following template:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#STYLES> <#WARNINGS> <#SERVERSCRIPT>
</BODY>
</HTML>
```

Each string is a sequence of one or more HTML commands or HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The

pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. TCustomAdapterPageProducer (see page 1159) automatically handles the conversion of the HTML-transparent tags in the default template as follows:

<#STYLES> - This tag is replaced by the stylesheet defined by the Styles (see page 1157) or StylesFile (see page 1158) property.

<#WARNINGS> - This tag is replaced with design time warnings that indicate possible problems with the hierarchy of web items.

<#SERVERSCRIPT> - This tag is replaced with generated HTML/JavaScript based on the hierarchy of Web items. TCustomAdapterPageProducer (see page 1159) will not generate any content if this tag is missing.

If you set HTMLDoc to include other HTML-transparent tags, you must translate these in an OnHTMLTag event handler.

Note: Setting HTMLDoc clears the HTMLFile (see page 1166) property.

See Also

Content

HTMLFile (see page 1166)

OnHTMLTag

12.1.3.5.2 TCustomAdapterPageProducer.HTMLFile Property

Specifies the name of a file that contains the HTML template used to generate the value returned by the Content method.

Pascal

```
property HTMLFile: TFileName;
```

C++

```
__property TFileName HTMLFile;
```

Description

Set HTMLFile to the name of the file that contains an HTML template. The HTML template is a sequence of HTML commands and HTML-transparent tags. An HTML-transparent tag has the form

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

A pound sign (#) immediately follows the opening angle bracket (<) with no spaces separating it from the angle bracket. The pound sign identifies the string to the page producer as an HTML-transparent tag. The tag name immediately follows the pound sign with no spaces separating it from the pound sign. The page producer converts the entire HTML-transparent tag into true HTML commands based on the value of the tag name. Following the tag name, the HTML-transparent tag can optionally include parameters that specify details of the conversion to be performed. Each parameter is of the form ParamName=Value, where there is no space between the parameter name, the equals symbol (=) and the value. Each parameter is delimited by white space.

The angle brackets (< >) make the tag transparent to HTML browsers that do not recognize the #TagName construct.

Note: Setting HTMLFile clears the HTMLDoc (see page 1165) property.

Note: When distributing applications that contain page producers which use the HTMLFile property, be sure to distribute the file along with the application. Also, be sure that the value of HTMLFile does not include a hard-coded path name that will not be available on the user's disk.

See Also

Content

HTMLDoc (see page 1165)

OnHTMLTag

12.1.4 TSitePageElements Class

TSitePageElements stores part of the content that page producers generate.

Class Hierarchy

Pascal

```
TSitePageElements = class(TObject);
```

C++

```
class TSitePageElements : public TObject;
```

File

SiteProd

Description

TSitePageElements stores styles, warnings, server script, server functions, and server variables. These values are stored as class members of type string (not properties). The adapter page producer combines them to make up its generated content.

12.1.4.1 TSitePageElements Members

The following tables list the members exposed by TSitePageElements.

Public Fields

	Name	Description
	ServerFunctionsContent (see page 1168)	This is ServerFunctionsContent, a member of class TSitePageElements.
	ServerScriptContent (see page 1169)	This is ServerScriptContent, a member of class TSitePageElements.
	ServerVariablesContent (see page 1169)	This is ServerVariablesContent, a member of class TSitePageElements.
	StylesContent (see page 1169)	This is StylesContent, a member of class TSitePageElements.
	WarningsContent (see page 1169)	This is WarningsContent, a member of class TSitePageElements.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

12.1.4.2 TSitePageElements Fields

The fields of the TSitePageElements class are listed here.

Public Fields

	Name	Description
	ServerFunctionsContent (see page 1168)	This is ServerFunctionsContent, a member of class TSitePageElements.
	ServerScriptContent (see page 1169)	This is ServerScriptContent, a member of class TSitePageElements.
	ServerVariablesContent (see page 1169)	This is ServerVariablesContent, a member of class TSitePageElements.
	StylesContent (see page 1169)	This is StylesContent, a member of class TSitePageElements.
	WarningsContent (see page 1169)	This is WarningsContent, a member of class TSitePageElements.

12.1.4.2.1 TSitePageElements.ServerFunctionsContent Field

Pascal

```
ServerFunctionsContent: string;
```

C++

```
UnicodeString ServerFunctionsContent;
```

Description

This is ServerFunctionsContent, a member of class TSitePageElements.

12.1.4.2.2 TSitePageElements.ServerScriptContent Field

Pascal

```
ServerScriptContent: string;
```

C++

```
UnicodeString ServerScriptContent;
```

Description

This is ServerScriptContent, a member of class TSitePageElements.

12.1.4.2.3 TSitePageElements.ServerVariablesContent Field

Pascal

```
ServerVariablesContent: string;
```

C++

```
UnicodeString ServerVariablesContent;
```

Description

This is ServerVariablesContent, a member of class TSitePageElements.

12.1.4.2.4 TSitePageElements.StylesContent Field

Pascal

```
StylesContent: string;
```

C++

```
UnicodeString StylesContent;
```

Description

This is StylesContent, a member of class TSitePageElements.

12.1.4.2.5 TSitePageElements.WarningsContent Field

Pascal

```
WarningsContent: string;
```

C++

```
UnicodeString WarningsContent;
```

Description

This is WarningsContent, a member of class TSitePageElements.

13 SOAPAttachIntf Namespace

This is namespace SOAPAttachIntf.

Topics

Name	Description
Classes (see page 1172)	The following table lists classes in this documentation.
Interfaces (see page 1176)	The following table lists interfaces in this documentation.
Types (see page 1185)	The following table lists types in this documentation.

Classes

	Name	Description
	TSoapDataList (see page 1172)	TSoapDataList maintains a list of SOAP attachments.

Interfaces

	Name	Description
	IMimeAttachmentHandler (see page 1176)	IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

Types

Name	Description
TOnGetAttachmentEvent (see page 1185)	TOnGetAttachmentEvent is the type of event handlers that respond when a remote interfaced object receives an attachment in a response from a Web Service provider.
TOnSendAttachmentEvent (see page 1185)	TOnSendAttachmentEvent is the type of event handlers that respond when a remote interfaced object sends an attachment in a request to a Web Service provider.

13.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapDataList (see page 1172)	TSoapDataList maintains a list of SOAP attachments.

13.1.1 TSoapDataList Class

TSoapDataList maintains a list of SOAP attachments.

Class Hierarchy

Pascal

```
TSoapDataList = class(TList);
```

C++

```
class TSoapDataList : public TList;
```

File

SOAPAttachIntf

Description

Use TSoapDataList to store and maintain a list of SOAP attachments (TSOAPAttachment objects).

See Also

TList

TSOAPAttachment

13.1.1.1 TSoapDataList Members

The following tables list the members exposed by TSoapDataList.

Public Methods

	Name	Description
	Add (see page 1173)	Adds a new attachment to the end of the list.
	Assign (see page 1173)	Raises an exception.
	Clear (see page 1174)	Empties the list and frees the TSOAPAttachment objects.
	Insert (see page 1174)	Raises an exception.
	Remove (see page 1174)	Raises an exception.

13.1.1.2 TSoapDataList Methods

The methods of the TSoapDataList class are listed here.

Public Methods

	Name	Description
	Add (see page 1173)	Adds a new attachment to the end of the list.
	Assign (see page 1173)	Raises an exception.
V	Clear (see page 1174)	Empties the list and frees the TSOAPAttachment objects.
	Insert (see page 1174)	Raises an exception.
	Remove (see page 1174)	Raises an exception.

13.1.1.2.1 TSoapDataList.Add Method

Adds a new attachment to the end of the list.

Pascal

```
function Add(Item: Pointer): Integer; reintroduce;
```

C++

```
__fastcall int Add(void * Item);
```

Description

Do not call the Add method. This method is used internally to add attachments to the data list when the application first reads an incoming message that includes attachments or when marshaling a message to be sent that includes attachments. Applications should not change the list of attachments, or they will interfere with the marshaling and unmarshaling of attachment data.

Item is a pointer to the TSoapAttachment instance that is added to the list.

Add returns the index of the new attachment.

See Also

Capacity

Count

Insert (see page 1174)

Items

Clear (see page 1174)

13.1.1.2.2 TSoapDataList.Assign Method

Raises an exception.

Pascal

```
procedure Assign(ListA: TList; AOperator: TListAssignOp = laCopy; ListB: TList = Nil);
reintroduce;
```

C++

```
__fastcall Assign(TList ListA, TListAssignOp AOperator = laCopy, TList ListB = Nil);
```

Description

Do not call the Assign method. TSoapDataList (see page 1172) replaces the inherited method to raise an exception if you call

this method because applications should not change the items in a TSoapDataList (see page 1172) object.

See Also

Add (see page 1173)

13.1.1.2.3 TSoapDataList.Clear Method

Empties the list and frees the TSOAPAttachment objects.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Clear removes all of the objects from the list and frees each one. Do not call this method. It is used internally when marshaling and unmarshaling messages that contain attachments. Applications should not change the items in a TSoapDataList (see page 1172) object.

See Also

Delete

Items

Remove

13.1.1.2.4 TSoapDataList.Insert Method

Raises an exception.

Pascal

```
procedure Insert(Index: Integer; aClass: TClass); reintroduce;
```

C++

```
__fastcall Insert(int Index, TClass aClass);
```

Description

Do not call the Insert method. TSoapDataList (see page 1172) replaces the inherited method to raise an exception if you call this method because applications should not change the items in a TSoapDataList (see page 1172) object.

See Also

Add (see page 1173)

13.1.1.2.5 TSoapDataList.Remove Method

Raises an exception.

Pascal

```
function Remove(aClass: TClass): Integer; reintroduce;
```

C++

```
__fastcall int Remove(TClass aClass);
```

Description

Do not call the Remove method. TSoapDataList (see page 1172) replaces the inherited method to raise an exception if you call this method because applications should not change the items in a TSoapDataList (see page 1172) object.

See Also

Clear (see page 1174)

13.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IMimeAttachmentHandler (see page 1176)	IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

13.2.1 IMimeAttachmentHandler Interface

IMimeAttachmentHandler is the interface for interpreting or manipulating the multipart form that represents a Web Service call that includes an attachment.

Class Hierarchy

SOAPAttachIntf.IMimeAttachmentHandler

Pascal

```
IMimeAttachmentHandler = interface;
```

C++

```
__interface IMimeAttachmentHandler;
```

File

SOAPAttachIntf

Description

IMimeAttachmentHandler defines methods for working with multipart forms that represent SOAP requests which include attachements. It is used internally to process the headers of the SOAP envelope part of multipart forms, to generate events when the application sends or receives a multipart form that represents a Web Service call which includes attachments, and to parse or assemble such a multipart form.

Note: C++ method declarations that use IMimeAttachmentHandler use the _di_IMimeAttachmentHandler type instead. This type is a DelphiInterface (see page 3256) wrapper around the IMimeAttachmentHandler interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IMimeAttachmentHandler >
_di_IMimeAttachmentHandler;
```

See Also

TSOAPAttachment

13.2.1.1 IMimeAttachmentHandler Members

The following tables list the members exposed by IMimeAttachmentHandler.

Public Events

	Name	Description
	OnGetAttachment (see page 1183)	Occurs for each attachment as it is read.
	OnSendAttachment (see page 1183)	Occurs for each attachment that is added to a multipart form to be sent.

Public Methods

	Name	Description
	AddSoapHeader (see page 1177)	Adds a specified header to the SOAP envelope portion of the emerging .
	CreateMimeStream (see page 1178)	Creates and initializes a stream that will eventually contain an assembled multipart form.
	FinalizeStream (see page 1178)	Performs any necessary final steps in creating the emerging multipart form.
	GetMIMEBoundary (see page 1179)	Returns the value of the MIMEBoundary (see page 1182) property.
	GetMIMEStream (see page 1179)	Returns a stream from which to read the assembled multipart form.
	GetOnGetAttachmentEvent (see page 1180)	Returns the OnGetAttachment (see page 1183) event handler.
	GetOnSendAttachmentEvent (see page 1180)	Returns the OnSendAttachment (see page 1183) event handler.
	ProcessMultiPartForm (see page 1180)	Parses a multipart form into the SOAP envelope and a list of attachments.
	SetMIMEBoundary (see page 1181)	Sets the value of the MIMEBoundary (see page 1182) property.
	SetOnGetAttachmentEvent (see page 1181)	Assigns the OnGetAttachment (see page 1183) event handler.
	SetOnSendAttachmentEvent (see page 1182)	Assigns the OnSendAttachment (see page 1183) event handler.

Public Properties

	Name	Description
	MIMEBoundary (see page 1182)	Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

13.2.1.2 IMimeAttachmentHandler Methods

The methods of the IMimeAttachmentHandler class are listed here.

Public Methods

	Name	Description
	AddSoapHeader (see page 1177)	Adds a specified header to the SOAP envelope portion of the emerging .
	CreateMimeStream (see page 1178)	Creates and initializes a stream that will eventually contain an assembled multipart form.
	FinalizeStream (see page 1178)	Performs any necessary final steps in creating the emerging multipart form.
	GetMIMEBoundary (see page 1179)	Returns the value of the MIMEBoundary (see page 1182) property.
	GetMIMEStream (see page 1179)	Returns a stream from which to read the assembled multipart form.
	GetOnGetAttachmentEvent (see page 1180)	Returns the OnGetAttachment (see page 1183) event handler.
	GetOnSendAttachmentEvent (see page 1180)	Returns the OnSendAttachment (see page 1183) event handler.
	ProcessMultiPartForm (see page 1180)	Parses a multipart form into the SOAP envelope and a list of attachments.
	SetMIMEBoundary (see page 1181)	Sets the value of the MIMEBoundary (see page 1182) property.
	SetOnGetAttachmentEvent (see page 1181)	Assigns the OnGetAttachment (see page 1183) event handler.
	SetOnSendAttachmentEvent (see page 1182)	Assigns the OnSendAttachment (see page 1183) event handler.

13.2.1.2.1 IMimeAttachmentHandler.AddSoapHeader Method

Adds a specified header to the SOAP envelope portion of the emerging .

Pascal

```
procedure AddSoapHeader(Value: string);
```

C++

```
__fastcall AddSoapHeader(UnicodeString Value);
```

Description

Call AddSoapHeader to add an arbitrary header to the SOAP envelope portion of an emerging multipart form. This method should not be called until after a call to the CreateMimeStream (see page 1178) method initializes the emerging multipart form.

Value is the text of the header to add to the SOAP envelope.

Typically, this method is called to add the ContentType header to the SOAP envelope when its value is not known at the point when the application calls CreateMimeStream (see page 1178).

Note: After all calls to AddSoapHeader have been made to add headers to the SOAP envelope, the application must call the FinalizeStream (see page 1178) method to allow the MIME attachment handler to finish generating the multipart form. Only after the call to FinalizeStream (see page 1178) can the application read the content of the multipart form by calling the GetMIMEStream (see page 1179) method.

See Also

CreateMimeStream (see page 1178)

FinalizeStream (see page 1178)

GetMIMEStream (see page 1179)

13.2.1.1.2.2 IMimeAttachmentHandler.CreateMimeStream Method

Creates and initializes a stream that will eventually contain an assembled multipart form.

Pascal

```
procedure CreateMimeStream(Envelope: TStream; Attachments: TSoapDataList);
```

C++

```
__fastcall CreateMimeStream(TStream Envelope, TSoapDataList Attachments);
```

Description

Call CreateMimeStream as the first step when generating a multipart form to represent a SOAP packet and its associated attachments.

Envelope is a stream from which CreateMimeStream can read the SOAP envelope portion of the multipart form. This must be a complete encoding with the exception of headers, which can be added by subsequent calls to the AddSoapHeader (see page 1177) method.

Attachments is a list of TSoapAttachment objects that represent the attachments that should be included in the multipart form.

CreateMimeStream creates a new stream object and initializes it with the data from Envelope and Attachments. This new stream can then be completed by calling the AddSoapHeader (see page 1177) method to add any headers to the SOAP envelope portion and then calling the FinalizeStream (see page 1178) method. After calling FinalizeStream (see page 1178), the application can access the generated stream by calling the GetMIMEStream (see page 1179) method.

See Also

FinalizeStream (see page 1178)

AddSoapHeader (see page 1177)

GetMIMEStream (see page 1179)

13.2.1.1.2.3 IMimeAttachmentHandler.FinalizeStream Method

Performs any necessary final steps in creating the emerging multipart form.

Pascal

```
procedure FinalizeStream;
```

C++

```
__fastcall FinalizeStream();
```

Description

Call FinalizeStream as a final step when assembling a multipart form to represent a SOAP packet and its associated attachments. When an application assembles a multipart form to represent a SOAP packet and its associated attachments, it takes the following steps:

1. Call CreateMimeStream (see page 1178) to create the stream object that includes data from the SOAP envelope and the content of any attachments.
2. Call AddSoapHeader (see page 1177) to add any headers to the SOAP envelope portion of the stream.
3. Call FinalizeStream to indicate that all changes are complete and allow the MIME attachment handler to make any final changes.

Only after a call to FinalizeStream can the application read the assembled multipart form by calling GetMIMEStream (see page 1179).

See Also

CreateMimeStream (see page 1178)

AddSoapHeader (see page 1177)

GetMIMEStream (see page 1179)

13.2.1.2.4 IMimeAttachmentHandler.GetMIMEBoundary Method

Returns the value of the MIMEBoundary (see page 1182) property.

Pascal

```
function GetMIMEBoundary: String;
```

C++

```
__fastcall UnicodeString GetMIMEBoundary();
```

Description

GetMIMEBoundary is the read implementation for the MIMEBoundary (see page 1182) property.

See Also

MIMEBoundary (see page 1182)

13.2.1.2.5 IMimeAttachmentHandler.GetMIMEStream Method

Returns a stream from which to read the assembled multipart form.

Pascal

```
function GetMIMEStream(Release: Boolean = False): TStream;
```

C++

```
__fastcall TStream GetMIMEStream(Boolean Release = False);
```

Description

Call GetMIMEStream after a call to the FinalizeStream (see page 1178) method to access the stream from which to read a newly-assembled multipart form. This stream should not be used until after the multipart form is fully assembled.

Release (see page 3641) indicates whether the caller is responsible for freeing the stream object. When Release (see page 3641) is true, the caller takes ownership of the stream and is responsible for freeing it. When Release (see page 3641) is false, the caller should not free the stream that is returned, because the object that implements IMimeAttachmentHandler (see page 1176) will free it.

See Also

CreateMimeStream (see page 1178)

AddSoapHeader (see page 1177)

FinalizeStream (see page 1178)

13.2.1.2.6 IMimeAttachmentHandler.GetOnGetAttachmentEvent Method

Returns the OnGetAttachment (see page 1183) event handler.

Pascal

```
function GetOnGetAttachmentEvent: TOnGetAttachmentEvent;
```

C++

```
__fastcall TOnGetAttachmentEvent GetOnGetAttachmentEvent();
```

Description

GetOnGetAttachmentEvent is the read implementation for the OnGetAttachment (see page 1183) event.

See Also

OnGetAttachment (see page 1183)

13.2.1.2.7 IMimeAttachmentHandler.GetOnSendAttachmentEvent Method

Returns the OnSendAttachment (see page 1183) event handler.

Pascal

```
function GetOnSendAttachmentEvent: TOnSendAttachmentEvent;
```

C++

```
__fastcall TOnSendAttachmentEvent GetOnSendAttachmentEvent();
```

Description

GetOnSendAttachmentEvent is the read implementation for the OnSendAttachment (see page 1183) event handler.

See Also

OnSendAttachment (see page 1183)

13.2.1.2.8 IMimeAttachmentHandler.ProcessMultiPartForm Method

Parses a multipart form into the SOAP envelope and a list of attachments.

Pascal

```
procedure ProcessMultiPartForm(const ASourceStream: TStream; const ADestStream: TStream;
```

```
AMimeBoundary: string; SoapHeaders: TStrings; Attachments: TSoapDataList; TempDir: string);
```

C++

```
__fastcall ProcessMultiPartForm(const TStream ASourceStream, const TStream ADestStream,
UnicodeString AMimeBoundary, TStrings SoapHeaders, TSoapDataList Attachments, UnicodeString
TempDir);
```

Description

Call ProcessMultiPartForm to interpret a multipart form that is received by the client or server in a Web Service application. This method separates the SOAP envelope portion of the multipart form from the portions that represent attachments, and parses the headers of the SOAP envelope into a convenient string list.

ASourceStream is a stream from which ProcessMultiPartForm can read the incoming multipart form. This is the content of the incoming message.

ADestStream is a stream to which ProcessMultiPartForm writes the SOAP envelope portion of ASourceStream.

AMimeBoundary is the mime boundary string that identifies the boundaries between the parts of the multipart form. Typically, it is obtained from the headers of the incoming HTTP message whose content is the content of ASourceStream.

SoapHeaders is an existing TStrings descendant to which ProcessMultiPartForm writes the headers of the SOAP envelope portion of the multipart form. Each header is added as a string of the form HeaderName=Value.

Attachments is an empty TSoapDataList (see page 1172) object to which ProcessMultiPartForm adds the TSoapAttachment objects that represent the attachments in the multipart form.

TempDir is the name of the directory where ProcessMultiPartForm writes the temporary files it generates for each of the attachments.

13.2.1.2.9 IMimeAttachmentHandler.SetMIMEBoundary Method

Sets the value of the MIMEBoundary (see page 1182) property.

Pascal

```
procedure SetMIMEBoundary(const MimeBndry: String);
```

C++

```
__fastcall SetMIMEBoundary(const UnicodeString MimeBndry);
```

Description

SetMIMEBoundary is the write implementation for the MIMEBoundary (see page 1182) property.

MimeBndry is the string that should identify the boundary between parts of the multipart form.

See Also

MIMEBoundary (see page 1182)

13.2.1.2.10 IMimeAttachmentHandler.SetOnGetAttachmentEvent Method

Assigns the OnGetAttachment (see page 1183) event handler.

Pascal

```
procedure SetOnGetAttachmentEvent(OnGetAttachment: TOnGetAttachmentEvent);
```

C++

```
__fastcall SetOnGetAttachmentEvent(TOnGetAttachmentEvent OnGetAttachment);
```

Description

SetOnGetAttachmentEvent is the write implementation for the OnGetAttachment (see page 1183) event.

OnGetAttachment (see page 1183) is the event handler that should to respond to the event.

See Also

OnGetAttachment (see page 1183)

13.2.1.2.11 IMimeAttachmentHandler.SetOnSendAttachmentEvent Method

Assigns the OnSendAttachment (see page 1183) event handler.

Pascal

```
procedure SetOnSendAttachmentEvent(OnSendAttachment: TOnSendAttachmentEvent);
```

C++

```
__fastcall SetOnSendAttachmentEvent(TOnSendAttachmentEvent OnSendAttachment);
```

Description

SetOnSendAttachmentEvent is the write implementation for the OnSendAttachment (see page 1183) event.

OnSendAttachment (see page 1183) is the event handler that should to respond to the event.

See Also

OnSendAttachment (see page 1183)

13.2.1.3 IMimeAttachmentHandler Properties

The properties of the IMimeAttachmentHandler class are listed here.

Public Properties

	Name	Description
	MIMEBoundary (see page 1182)	Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

13.2.1.3.1 IMimeAttachmentHandler.MIMEBoundary Property

Indicates the MIME boundary string that signals the boundaries between the parts of the multipart form.

Pascal

```
property MIMEBoundary: string;
```

C++

```
__property UnicodeString MIMEBoundary;
```

Description

MIMEBoundary is the string that identifies the boundary between the parts of the multipart form. It is an attribute of the ContentType header for the entire form.

When the MIME attachment handler is processing a received form, this property is not available until after the application has called the ProcessMultiPartForm (see page 1180) method. When the MIME attachment handler is assembling a multipart form in order to send a message that includes attachments, the handler generates this string for the new form, and the CreateMimeStream (see page 1178) method uses it when assembling the form.

See Also

ProcessMultiPartForm (see page 1180)

CreateMimeStream (see page 1178)

13.2.1.4 IMimeAttachmentHandler Events

The events of the IMimeAttachmentHandler class are listed here.

Public Events

	Name	Description
	OnGetAttachment (see page 1183)	Occurs for each attachment as it is read.
	OnSendAttachment (see page 1183)	Occurs for each attachment that is added to a multipart form to be sent.

13.2.1.4.1 IMimeAttachmentHandler.OnGetAttachment Event

Occurs for each attachment as it is read.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

The MIME attachment handler generates this event as it reads each incoming attachment. Classes that use the MIME attachment handler, such as TRio or TSoapPascalInvoker, assign their own event handlers as the value of this event.

See Also

OnSendAttachment (see page 1183)

OnGetAttachment

OnGetAttachment (see page 1358)

13.2.1.4.2 IMimeAttachmentHandler.OnSendAttachment Event

Occurs for each attachment that is added to a multipart form to be sent.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

The MIME attachment handler generates this event as it adds each outgoing attachment to the multipart form. Classes that use the MIME attachment handler, such as TRio or TSoapPascalInvoker, assign their own event handlers as the value of this event.

See Also

OnGetAttachment (see page 1183)

OnSendAttachment

OnSendAttachment (see page 1358)

13.3 Types

The following table lists types in this documentation.

Types

Name	Description
TOnGetAttachmentEvent (see page 1185)	TOnGetAttachmentEvent is the type of event handlers that respond when a remote interfaced object receives an attachment in a response from a Web Service provider.
TOnSendAttachmentEvent (see page 1185)	TOnSendAttachmentEvent is the type of event handlers that respond when a remote interfaced object sends an attachment in a request to a Web Service provider.

13.3.1 SOAPAttachIntf.TOnGetAttachmentEvent Type

TOnGetAttachmentEvent is the type of event handlers that respond when a remote interfaced object receives an attachment in a response from a Web Service provider.

Pascal

```
TOnGetAttachmentEvent = procedure (AttachmentStream: TStream; Attachment: TSOAPAttachment) of object;
```

C++

```
(AttachmentStream: TStream; Attachment: TSOAPAttachment) ( TOnGetAttachmentEvent)();
```

File

SOAPAttachIntf

Description

TOnGetAttachmentEvent is the type of the OnGetAttachment event handler, which is called for every attachment that a client receives in the encoded results from executing a method on an invokable interface.

AttachmentStream is a stream from which you can read or change the content of the attachment. This is the section of the multipart form received from the server that represents the current attachment.

Attachment is the TSOAPAttachment instance that will represent the current attachment. Because the attachment has not yet been saved to a temporary file, this object can't be used to read the current attachment (use the AttachmentStream parameter instead). However, its ID and Headers properties are initialized when the event handler is called, so Attachment can be used to read the values of any headers in the attachment.

See Also

Procedural types

Creating events

13.3.2 SOAPAttachIntf.TOnSendAttachmentEvent Type

TOnSendAttachmentEvent is the type of event handlers that respond when a remote interfaced object sends an attachment in a

request to a Web Service provider.

Pascal

```
TOnSendAttachmentEvent = procedure (AttachmentStream: TStream; Attachment: TSOAPAttachment) of object;
```

C++

```
(AttachmentStream: TStream; Attachment: TSOAPAttachment) ( TOnSendAttachmentEvent)();
```

File

SOAPAttachIntf

Description

TOnSendAttachmentEvent is the type of the OnSendAttachment event handler, which is called for every attachment that a client sends in the request to execute a method on an invokable interface.

AttachmentStream is a stream from which you can read or change the content of the attachment. This becomes the section of the multipart form sent to the server that represents the current attachment.

Attachment is the TSOAPAttachment instance that represents the current attachment.

See Also

Procedural types

Creating events

14 SOAPConn Namespace

This is namespace SOAPConn.

Topics

Name	Description
Classes (see page 1188)	The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapConnection (see page 1188)	TSoapConnection connects to an application server that is implemented as a Web Service.

14.1 Classes

The following table lists classes in this documentation.

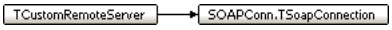
Classes

	Name	Description
	TSoapConnection (see page 1188)	TSoapConnection connects to an application server that is implemented as a Web Service.

14.1.1 TSoapConnection Class

TSoapConnection connects to an application server that is implemented as a Web Service.

Class Hierarchy



Pascal

```
TSoapConnection = class(TCustomRemoteServer);
```

C++

```
class TSoapConnection : public TCustomRemoteServer;
```

File

SOAPConn

Description

TSoapConnection is used in the client portion of a multi-tiered database application to establish and maintain the connection between the client and a remote application server that is implemented as a Web Service. The SOAP connection object can

Establish an initial connection to a remote application server.

Obtain an interface for the application server.

Obtain a list of providers on the application server.

Drop the connection to the remote application server.

TSoapConnection establishes the initial connection between the client application and a remote application server using SOAP.

Note: To use TSoapConnection on Windows, you must have wininet.dll installed on the client system. wininet.dll is found in the Windows system directory if you have IE3 or higher installed.

TSoapConnection uses an internal THTTPRio object to obtain an interface from the Web Service application. This interface is either IAppServer or IAppServerSOAP (see page 1340) (or one of their descendants), depending on the value of the UseSOAPAdapter (see page 1200) property. The recommended approach is to use the IAppServerSOAP (see page 1340) interface, which is better adapted to SOAP as a transport protocol. However, some application servers (those created using Delphi 6 or Kylix 2) only support an IAppServer interface. If you are using an application server that does not support IAppServerSOAP (see page 1340), set UseSOAPAdapter (see page 1200) to false.

Regardless of which interface TSoapConnection uses to communicate with the application server, it makes the information available to client datasets in the client application by providing an IAppServer interface. Client datasets use the IAppServer

interface from the SOAP connection component to communicate with providers on the application server, or to otherwise call the application server's data module interface.

Note: Unlike when using other DataSnap connection components, you can't use the AppServer property of TSoapConnection to call methods of the application server's interface that are not IAppServer methods. Instead, to communicate with a SOAP data module on the application interface, use a separate THTTPRIO (see page 1248) object or specify the interface using the SOAPServerIID (see page 1198) property and then call the GetSOAPServer (see page 1194) method

. To connect to remote application servers using DCOM, use the TDCOMConnection component instead. To connect to remote application servers using TCP/IP, use the TSocketConnection component instead. To connect to remote application servers using HTTP (but not SOAP), use TWebConnection.

See Also

TDataSetProvider

TClientDataSet

TDCOMConnection

TRemoteDataModule

IAppServer

TSocketConnection (see page 221)

TWebConnection (see page 254)

THTTPRIO (see page 1248)

IAppServerSOAP (see page 1340)

Extending the Interface of the Application Server

Specifying a Connection Using SOAP

14.1.1.1 TSoapConnection Members

The following tables list the members exposed by TSoapConnection.

Protected Methods

	Name	Description
	DoConnect (see page 1191)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 1192)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 1192)	Returns the value of the Connected property.
	GetSOAPServerIID (see page 1194)	This is GetSOAPServerIID, a member of class TSoapConnection.
	Notification (see page 1194)	Forwards notification messages to all owned components.
	SetSOAPServerIID (see page 1195)	This is SetSOAPServerIID, a member of class TSoapConnection.
	SetUseSoapAdapter (see page 1195)	This is SetUseSoapAdapter, a member of class TSoapConnection.

Public Methods

	Name	Description
	Create (see page 1190)	Creates and initializes an instance of TSoapConnection (see page 1188).
	Destroy (see page 1191)	Destroys an instance of TSoapConnection (see page 1188).
	GetProviderNames (see page 1192)	Executes a callback for all provider objects on the application server.
	GetServer (see page 1193)	Returns an IAppServer interface for communicating with the application server.
	GetServerList (see page 1193)	Returns a list of application servers (remote data modules) to which the component can connect.

	GetSOAPServer (see page 1194)	Returns the IAppServerSOAP (see page 1340) interface of the application server.

Public Properties

	Name	Description
	RIO (see page 1198)	Provides access to the internal remote interfaced object that communicates with the application server.

Published Events

	Name	Description
	AfterConnect (see page 1201)	Occurs after a connection is established.
	AfterDisconnect (see page 1201)	Occurs after the connection closes.
	BeforeConnect (see page 1202)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1202)	Occurs immediately before the connection closes.
	OnAfterExecute (see page 1203)	Occurs when TSoapConnection (see page 1188) receives the results from executing a method on the application server.
	OnBeforeExecute (see page 1203)	Occurs before TSoapConnection (see page 1188) sends a method call to the application server for execution.

Published Properties

	Name	Description
	Agent (see page 1196)	Provides the value of the User-Agent header for HTTP messages to the server.
	Connected (see page 1196)	Indicates whether the SOAP connection component is connected to the application server.
	Password (see page 1197)	Specifies the password that allows the client application to log on to a local host or proxy server.
	Proxy (see page 1197)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1197)	Lists the host names of locally registered servers that do not require a proxy.
	SOAPServerIID (see page 1198)	Identifies the invokable interface that TSoapConnection (see page 1188) uses to communicate with the application server.
	URL (see page 1199)	Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.
	UserName (see page 1200)	Specifies the user name that allows the client application to log on to a local host or proxy server.
	UseSOAPAdapter (see page 1200)	Specifies how the SOAP connection component communicates with the application server.

14.1.1.2 TSoapConnection.Create Constructor

Creates and initializes an instance of TSoapConnection (see page 1188).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSoapConnection(TComponent * AOwner);
```

Description

Call Create to instantiate a TSoapConnection (see page 1188) object at runtime. SOAP connection components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually a form or data module) that is responsible for freeing the SOAP connection component.

See Also

Owner

14.1.1.3 TSoapConnection.Destroy Destructor

Destroys an instance of TSoapConnection (see page 1188).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapConnection();
```

Description

Do not call Destroy directly in an application. Instead, use the Free method, which checks that the SOAP connection is not nil before calling Destroy.

14.1.1.4 TSoapConnection Methods

The methods of the TSoapConnection class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 1191)	Provides the interface for a method that opens a connection.
	DoDisconnect (see page 1192)	Provides the interface for a method that terminates the connection.
	GetConnected (see page 1192)	Returns the value of the Connected property.
	GetSOAPServerIID (see page 1194)	This is GetSOAPServerIID, a member of class TSoapConnection.
	Notification (see page 1194)	Forwards notification messages to all owned components.
	SetSOAPServerIID (see page 1195)	This is SetSOAPServerIID, a member of class TSoapConnection.
	SetUseSoapAdapter (see page 1195)	This is SetUseSoapAdapter, a member of class TSoapConnection.

Public Methods

	Name	Description
	GetProviderNames (see page 1192)	Executes a callback for all provider objects on the application server.
	GetServer (see page 1193)	Returns an IAppServer interface for communicating with the application server.
	GetServerList (see page 1193)	Returns a list of application servers (remote data modules) to which the component can connect.
	GetSOAPServer (see page 1194)	Returns the IAppServerSOAP (see page 1340) interface of the application server.

14.1.1.4.1 TSoapConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected

DoDisconnect

14.1.1.4.2 TSoapConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected

DoConnect

14.1.1.4.3 TSoapConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

14.1.1.4.4 TSoapConnection.GetProviderNames Method

Executes a callback for all provider objects on the application server.

Pascal

```
procedure GetProviderNames(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetProviderNames(TGetStrProc Proc);
```

Description

Call GetProviderNames to execute the procedure specified by the Proc parameter for every provider exposed by the application server. For every provider listed by the remote data module's AS_GetProviderNames method, Proc is called with the name of the provider.

See Also

AS_GetProviderNames

AppServer

ProviderName

14.1.1.4.5 TSoapConnection.GetServer Method

Returns an IAppServer interface for communicating with the application server.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

Client datasets call GetServer to set the value of their AppServer property. This allows them to access providers on the remote application server.

GetServer ensures that there is a connection and then returns an IAppServer interface that can be used to communicate with providers on the application server. To use other methods of the application server's interface, use a separate THTTPRIO (see page 1248) object or call the GetSOAPServer (see page 1194) method after setting the SOAPServerIID (see page 1198) property.

See Also

Connected (see page 1196)

AppServer

THTTPRIO (see page 1248)

UseSOAPAdapter (see page 1200)

SOAPServerIID (see page 1198)

14.1.1.4.6 TSoapConnection.GetServerList Method

Returns a list of application servers (remote data modules) to which the component can connect.

Pascal

```
function GetServerList: OleVariant; override;
```

C++

```
virtual __fastcall OleVariant GetServerList();
```

Description

Connection components use GetServerList internally to fill the drop down list in the Object Inspector for the ServerName property.

As implemented in TCustomRemoteServer, GetServerList always returns NULL. Descendant classes override this method to return a list of application servers that are on the local machine and registered in the system registry (TDispatchConnection), or fetched from the server machine (TStreamedConnection).

See Also

ServerName

14.1.1.4.7 TSoapConnection.GetSOAPServer Method

Returns the IAppServerSOAP (see page 1340) interface of the application server.

Pascal

```
function GetSOAPServer: IAppServerSOAP;
```

C++

```
__fastcall IAppServerSOAP GetSOAPServer();
```

Description

Call GetSOAPServer to access the IAppServerSOAP (see page 1340) interface that the SOAP connection component uses when UseSOAPAdapter (see page 1200) is true. The internal adapter uses this interface to implement the IAppServer interface that the GetServer (see page 1193) method returns.

Tip: If you have set the SOAPServerIID (see page 1198) property, you can use the as operator (Delphi) or the Supports (see page 4294) method (C++) to convert this interface into the application server's interface.

See Also

Connected (see page 1196)

GetServer (see page 1193)

UseSOAPAdapter (see page 1200)

THTTPRIO (see page 1248)

SOAPServerIID (see page 1198)

14.1.1.4.8 TSoapConnection.GetSOAPServerIID Method

Pascal

```
function GetSOAPServerIID: TGUID;
```

C++

```
__fastcall TGUID GetSOAPServerIID();
```

Description

This is GetSOAPServerIID, a member of class TSoapConnection.

14.1.1.4.9 TSoapConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

14.1.1.4.10 TSoapConnection.SetSOAPServerIID Method

Pascal

```
procedure SetSOAPServerIID(const IID: String);
```

C++

```
__fastcall SetSOAPServerIID(const UnicodeString IID);
```

Description

This is SetSOAPServerIID, a member of class TSoapConnection.

14.1.1.4.11 TSoapConnection.SetUseSoapAdapter Method

Pascal

```
procedure SetUseSoapAdapter(Value: Boolean);
```

C++

```
__fastcall SetUseSoapAdapter(Boolean Value);
```

Description

This is SetUseSoapAdapter, a member of class TSoapConnection.

14.1.1.5 TSoapConnection Properties

The properties of the TSoapConnection class are listed here.

Public Properties

	Name	Description
	RIO (see page 1198)	Provides access to the internal remote interfaced object that communicates with the application server.

Published Properties

	Name	Description
	Agent (see page 1196)	Provides the value of the User-Agent header for HTTP messages to the server.
	Connected (see page 1196)	Indicates whether the SOAP connection component is connected to the application server.
	Password (see page 1197)	Specifies the password that allows the client application to log on to a local host or proxy server.
	Proxy (see page 1197)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1197)	Lists the host names of locally registered servers that do not require a proxy.
	SOAPServerIID (see page 1198)	Identifies the invokable interface that TSoapConnection (see page 1188) uses to communicate with the application server.
	URL (see page 1199)	Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.
	UserName (see page 1200)	Specifies the user name that allows the client application to log on to a local host or proxy server.
	UseSOAPAdapter (see page 1200)	Specifies how the SOAP connection component communicates with the application server.

14.1.1.5.1 TSoapConnection.Agent Property

Provides the value of the User-Agent header for HTTP messages to the server.

Pascal

```
property Agent: string;
```

C++

```
__property UnicodeString Agent;
```

Description

Use Agent to supply HTTP messages generated by the SOAP connection component with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the HTTP message.

See Also

Title

14.1.1.5.2 TSoapConnection.Connected Property

Indicates whether the SOAP connection component is connected to the application server.

Pascal

```
property Connected;
```

C++

```
__property Connected;
```

Description

Set Connected to true to establish a connection to the application server. Set Connected to false to terminate the connection.

When changing the value of Connected, TSoapConnection (see page 1188) generates a BeforeConnect (see page 1202) and an AfterConnect (see page 1201) event, or a BeforeDisconnect (see page 1202) and an AfterDisconnect (see page 1201) event.

See Also

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

14.1.1.5.3 TSoapConnection.Password Property

Specifies the password that allows the client application to log on to a local host or proxy server.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

When the SOAP connection component connects to the application server, it supplies the values of the UserName (see page 1200) and Password properties so that it can log on to the local host or proxy. The value of Password can be an empty string if neither the local host nor proxy requires authentication.

Note: On Linux, if the proxy does not require authentication, you can use the UserName (see page 1200) and Password properties to log on to the Web Server.

See Also

UserName (see page 1200)

Specifying a Connection Using SOAP

14.1.1.5.4 TSoapConnection.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

Use Proxy if the SOAP connection component can't resolve the host portion of URL (see page 1199) locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 1197)

Specifying a Connection Using SOAP

14.1.1.5.5 TSoapConnection.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property UnicodeString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the SOAP connection component can make a direct connection. When the URL (see page 1199) property identifies a server in this list, the Web connection component makes a direct connection even if the Proxy (see page 1197) property is specified.

When ProxyByPass is an empty string, the Web connection component checks the registry for any locally known servers and bypasses the proxy server if the host portion of URL (see page 1199) refers to a registered local server.

See Also

Proxy (see page 1197)

URL (see page 1199)

14.1.1.5.6 TSoapConnection.RIO Property

Provides access to the internal remote interfaced object that communicates with the application server.

Pascal

```
property RIO: THTTPRIO;
```

C++

```
__property THTTPRIO RIO;
```

Description

TSoapConnection (see page 1188) uses an internal remote interfaced object (THTTPRIO (see page 1248)) to communicate with the application server. This internal object obtains a reference to an invokable interface that is specified by the SOAPServerIID (see page 1198) property.

Use RIO to access the internal THTTPRIO (see page 1248) instance.

See Also

SOAPServerIID (see page 1198)

14.1.1.5.7 TSoapConnection.SOAPServerIID Property

Identifies the invokable interface that TSoapConnection (see page 1188) uses to communicate with the application server.

Pascal

```
property SOAPServerIID: String;
```

C++

```
__property UnicodeString SOAPServerIID;
```

Description

Set SOAPServerIID to identify the interface of the application server. SOAPServerIID is the GUID of the application server's interface. This interface is a descendant of IAppServerSOAP (see page 1340). Once you have formed a connection, you can access this interface by first calling the GetSOAPServer (see page 1194) method and then using the global Supports (see page 4294) function or as operator (Delphi) or the Supports (see page 4294) method of the interface (C++).

Warning: Use the GUID for the interface that is generated by the WSDL importer when you import the application server's definitions. This may differ from the GUID that the server uses in its implementation.

Note: In Delphi, at runtime, you can use the interface name in code and the compiler automatically converts that to the GUID. At design time (in the Object Inspector) or in C++ code, you must specify the GUID itself, not the interface name.

Note: SOAPServerIID is only used if the UseSOAPAdapter (see page 1200) property is true. In addition, if you set SOAPServerIID to anything other than IAppServerSOAP (see page 1340), you must import the definition of the application server's interface so that it is registered with the invocation registry.

See Also

URL (see page 1199)

IAppServerSOAP (see page 1340)

UseSOAPAdapter (see page 1200)

GetSOAPServer (see page 1194)

Specifying a Connection Using SOAP

14.1.1.5.8 TSoapConnection.URL Property

Specifies the Uniform Resource Locator for the THTTPSoapDispatcher on the application server.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

Use URL to specify the Uniform Resource Locator that the SOAP connection component uses to locate the application server.

URL includes the protocol (http or https), host name, and scriptname for the application server, as well as a path portion that identifies the THTTPSoapDispatcher on the application server. Typically, it has the form

```
http://DataHost.org/scripts/AppServer.dll/SOAP/
http://DataHost.org/scripts/AppServer.so/SOAP/
```

Note: By default, URL does not include the interface name. The name of the interface is assumed to be IAppServerSOAP (see page 1340) (or IAppServer). If the server is written in the Delphi language and the target remote data module supports an IAppServerSOAP (see page 1340) descendant, Delphi clients can optionally include the target interface name at the end of the URL. This allows clients to identify a SOAP data module when the server contains multiple SOAP data modules, or allows you to access the server's interface by calling the GetSOAPServer (see page 1194) method. If the client or server is not written using Delphi, you can accomplish this same effect using the SOAPServerIID (see page 1198) property.

Note: URL is a standard Uniform Resource Locator. As such, you can also use it to override the port associated with http (80) or https (443) if the Web server is configured differently. The port number follows the host name, preceded by a colon.

See Also

THTTPSoapDispatcher (see page 5946)

SOAPServerIID (see page 1198)

GetSOAPServer (see page 1194)

Specifying a Connection Using SOAP

14.1.1.5.9 TSoapConnection.UserName Property

Specifies the user name that allows the client application to log on to a local host or proxy server.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

When the SOAP connection component connects to the application server on a local host or to a proxy server, it supplies the values of the UserName and Password (see page 1197) properties so that it can log in. If neither the host nor the proxy server requires authentication, UserName can be an empty string.

Note: On Linux, if the proxy does not require authentication, you can use the UserName and Password (see page 1197) properties to log on to the Web Server.

See Also

Password (see page 1197)

Proxy (see page 1197)

Specifying a Connection Using SOAP

14.1.1.5.10 TSoapConnection.UseSOAPAdapter Property

Specifies how the SOAP connection component communicates with the application server.

Pascal

```
property UseSOAPAdapter: Boolean;
```

C++

```
__property Boolean UseSOAPAdapter;
```

Description

When UseSOAPAdapter is true, TSoapConnection (see page 1188) communicates with the application server using an IAppServerSOAP (see page 1340) interface. It then uses an internal adapter which implements the IAppServer interface for client datasets. In this case, the GetServer (see page 1193) method returns the IAppServer interface that the internal adapter implements.

When UseSOAPAdpater is false, TSoapConnection (see page 1188) uses an IAppServer interface to communicate with the application server and exposes that interface with the GetServer (see page 1193) method. In this case, the GetSOAPServer (see page 1194) method returns nil (Delphi) or NULL (C++).

The recommended setting for UseSOAPAdapter is true. Unlike IAppServer, the IAppServerSOAP (see page 1340) interface does not use the safecall calling convention. This makes it easier to marshal interface calls when the client or server is written in C++. IAppServerSOAP (see page 1340) is not supported, however, when the application server is built using Delphi 6 (before update patch 2) or Kylix 2. If you are connecting to a server that does not support IAppServerSOAP (see page 1340), set UseSOAPAdapter to false.

See Also

IAppServer

IAppServerSOAP (see page 1340)

GetServer (see page 1193)

GetSOAPServer (see page 1194)

Specifying a Connection Using SOAP

14.1.1.6 TSoapConnection Events

The events of the TSoapConnection class are listed here.

Published Events

	Name	Description
	AfterConnect (see page 1201)	Occurs after a connection is established.
	AfterDisconnect (see page 1201)	Occurs after the connection closes.
	BeforeConnect (see page 1202)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1202)	Occurs immediately before the connection closes.
	OnAfterExecute (see page 1203)	Occurs when TSoapConnection (see page 1188) receives the results from executing a method on the application server.
	OnBeforeExecute (see page 1203)	Occurs before TSoapConnection (see page 1188) sends a method call to the application server for execution.

14.1.1.6.1 TSoapConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect;
```

C++

```
__property AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect

Connected

Connecting to a Database Server

14.1.1.6.2 TSoapConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect;
```

C++

```
__property AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect

Connected

Disconnecting from a Database Server

14.1.1.6.3 TSoapConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect;
```

C++

```
__property BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect

Connected

Connecting to a Database Server

14.1.1.6.4 TSoapConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect;
```

C++

```
__property BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
```

```
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect

Connected

Disconnecting from a Database Server

14.1.1.6.5 TSoapConnection.OnAfterExecute Event

Occurs when TSoapConnection (see page 1188) receives the results from executing a method on the application server.

Pascal

```
property OnAfterExecute: TAfterExecuteEvent;
```

C++

```
__property TAfterExecuteEvent OnAfterExecute;
```

Description

Write (see page 3707) an OnAfterExecute event handler to examine or alter the encoded results from executing a method on the application server. OnAfterExecute is assigned as the OnAfterExecute event handler of an internal remote interfaced object that the SOAP connection uses to communicate with the application server.

See Also

OnBeforeExecute (see page 1203)

OnAfterExecute

GetServer (see page 1193)

14.1.1.6.6 TSoapConnection.OnBeforeExecute Event

Occurs before TSoapConnection (see page 1188) sends a method call to the application server for execution.

Pascal

```
property OnBeforeExecute: TBeforeExecuteEvent;
```

C++

```
__property TBeforeExecuteEvent OnBeforeExecute;
```

Description

Write (see page 3707) an OnBeforeExecute event handler to examine or alter an encoded method call before it is sent to the server. OnBeforeExecute occurs after the call has been marshaled into a SOAP request packet, but before that SOAP request packet has been sent to the application server.

OnBeforeExecute is assigned as the OnBeforeExecute event handler of an internal remote interfaced object that the SOAP connection uses to communicate with the application server.

See Also

OnAfterExecute (see page 1203)

OnBeforeExecute

GetServer (see page 1193)

15 SOAPDm Namespace

This is namespace SOAPDm.

Topics

Name	Description
Classes (see page 1206)	The following table lists classes in this documentation.

Classes

	Name	Description
	TSoapDataModule (see page 1206)	TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

15.1 Classes

The following table lists classes in this documentation.

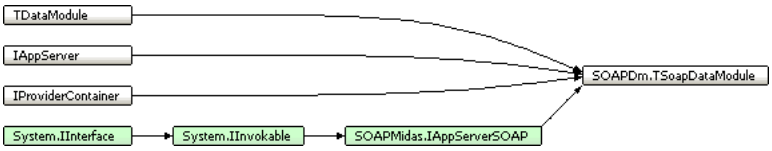
Classes

	Name	Description
	TSoapDataModule (see page 1206)	TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

15.1.1 TSoapDataModule Class

TSoapDataModule encapsulates the objects and interfaces of an application server in a SOAP-based multi-tiered database application.

Class Hierarchy



Pascal

```
TSoapDataModule = class(TDataModule, IAppServer, IAppServerSOAP, IProviderContainer);
```

C++

```
class TSoapDataModule : public TDataModule, public IAppServer, public IAppServerSOAP, public
IProviderContainer;
```

File

SOAPDm

Description

Use a TSoapDataModule object as the central repository for all objects in a SOAP-based application server. SOAP data modules can contain any nonvisual components. Typically these are data access components, such as datasets and the provider components that handle communication between these datasets and client applications. They can also contain other nonvisual components, such as TTimer, or TImageList.

At design time a TSoapDataModule object provides a visual container into which a developer can place nonvisual components, set their properties, and write event handlers for them. In the unit file for the SOAP data module a developer can also code any business rules that are to be centralized in the middle tier of a multi-tiered application.

TSoapDataModule implements the IAppServer and IAppServerSOAP (see page 1340) interfaces. These are both invokable interfaces, so if you add TSoapDataModule to a Web Service application, the invoker automatically forwards all IAppServer or IAppServerSOAP (see page 1340) calls to the SOAP data module. The client application uses a TSoapConnection component to generate these calls.

To create a SOAP data module at design time,

1. Choose File|New (see page 3619)|Other, and select Web Services application from the WebServices page of the new items dialog. This generates a Web module that contains a dispatcher and an invoker component. These components work together to interpret and execute incoming SOAP requests.
2. Choose File|New (see page 3619)|Other, and select SOAP DataModule from the Web Services page of the new items dialog. This adds a TSoapDataModule descendant to your application. The TSoapDataModule descendant supports an invokable interface that descends from IAppServerSOAP (see page 1340) as well as an IAppServer interface.

   Note: Web Service applications should not include more than one TSoapDataModule descendant. When the invoker component that appears in the primary Web module receives a request for the IAppServer or IAppServerSOAP (see page 1340) interface, it can't distinguish between multiple implementation classes.

See Also

IAppServer

TSoapConnection (see page 1188)

IAppServerSOAP (see page 1340)

15.1.1.1 TSoapDataModule Members

The following tables list the members exposed by TSoapDataModule.

Protected Methods

	Name	Description
	ApplyUpdates (see page 1211)	Applies updates received from a client using a specified provider.
	AS_ApplyUpdates (see page 1211)	Applies updates received from a client using a specified provider.
	AS_DataRequest (see page 1212)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 1213)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 1214)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 1215)	Returns a list of all the available providers on the SOAP data module.
	AS_GetRecords (see page 1215)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 1216)	Returns information from a specified record of the provider's dataset.
	DataRequest (see page 1217)	Generates an OnDataRequest event on the specified provider component.
	Execute (see page 1218)	Executes the query or stored procedure bound to a specified provider.
	GetParams (see page 1219)	Fetches current parameter values from the dataset bound to a specified provider.
V	GetProvider (see page 1219)	Returns a registered provider component given its name.
	GetProviderNames (see page 1220)	Returns a list of all the available providers on the SOAP data module.
	GetRecords (see page 1220)	Returns a data packet that contains the specified data.
	RowRequest (see page 1222)	Returns information from a specified record of the provider's dataset.
V	SAS_ApplyUpdates (see page 1223)	Applies updates received from a client using a specified provider.
V	SAS_DataRequest (see page 1224)	Generates an OnSAS_DataRequest event on the specified provider component.
V	SAS_Execute (see page 1225)	SAS_Executes the query or stored procedure bound to a specified provider.
V	SAS_GetParams (see page 1226)	Fetches current parameter values from the dataset bound to a specified provider.
V	SAS_GetProviderNames (see page 1226)	Returns a list of all the available providers on the SOAP data module.
V	SAS_GetRecords (see page 1227)	Returns a data packet that contains the specified data.
V	SAS_RowRequest (see page 1228)	Returns information from a specified record of the provider's dataset.

Public Methods

	Name	Description
	_AddRef (see page 1210)	Increments the reference count for this data module's interface.
	_Release (see page 1210)	Decrements the reference count for this data module's interface.
V	AfterConstruction (see page 1210)	Generates an OnCreate event.
V	BeforeDestruction (see page 1217)	Generates an OnDestroy event.
V	Create (see page 1208)	Creates a SOAP data module instance.

	Destroy (see page 1208)	Frees a SOAP data module instance.
	Lock (see page 1221)	Locks the SOAP data module so that it can't be locked by other threads.
	NewInstance (see page 1221)	Allocates memory for each instance of TSoapDataModule (see page 1206).
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterProvider (see page 1222)	Adds a provider to the list returned by AS_GetProviderNames (see page 1215).
	SafeCallException (see page 1223)	Handles exceptions in methods declared using the safecall calling convention.
	Unlock (see page 1229)	Unlocks the SOAP data module after it was locked by a call to the Lock (see page 1221) method.
	UnRegisterProvider (see page 1229)	Removes a provider from the list returned by AS_GetProviderNames (see page 1215).

Public Properties

	Name	Description
	ProviderCount (see page 1230)	Indicates the number of providers registered with the SOAP data module.
	Providers (see page 1230)	Lists the registered provider components that are contained in the SOAP data module.
	RefCount (see page 1231)	Reference counts the data module's interface.

15.1.1.2 TSoapDataModule.Create Constructor

Creates a SOAP data module instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSoapDataModule(TComponent * AOwner);
```

Description

Most applications do not need to call the constructor for TSoapDataModule (see page 1206). Instead, add a SOAP data module to a Web service application using the SOAP data module wizard on the SOAP page of the new items dialog.

AOwner specifies the component that is responsible for freeing the SOAP data module (typically the application). If you supply an owner, the SOAP data module acts as a component, and is freed by that owner. If you pass nil for the AOwner parameter, the SOAP data module acts like an interfaced object, and maintains a reference count for every use of its interface. When that reference count goes to zero, the SOAP data module automatically frees itself. Note that is you set AOwner to nil but never obtain a reference to this object's interface, you must explicitly free the data module.

See Also

Create

15.1.1.3 TSoapDataModule.Destroy Destructor

Frees a SOAP data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapDataModule();
```

Description

Most applications do not need to call the destructor for TSoapDataModule (see page 1206). The SOAP data module is freed

by its owner when the application shuts down.

If you must destroy a SOAP data module at runtime, call the Free method instead.

See Also

System::TObject::Free (see page 3372)

15.1.1.4 TSoapDataModule Methods

The methods of the TSoapDataModule class are listed here.

Protected Methods

	Name	Description
	ApplyUpdates (see page 1211)	Applies updates received from a client using a specified provider.
	AS_ApplyUpdates (see page 1211)	Applies updates received from a client using a specified provider.
	AS_DataRequest (see page 1212)	Generates an OnAS_DataRequest event on the specified provider component.
	AS_Execute (see page 1213)	AS_Executes the query or stored procedure bound to a specified provider.
	AS_GetParams (see page 1214)	Fetches current parameter values from the dataset bound to a specified provider.
	AS_GetProviderNames (see page 1215)	Returns a list of all the available providers on the SOAP data module.
	AS_GetRecords (see page 1215)	Returns a data packet that contains the specified data.
	AS_RowRequest (see page 1216)	Returns information from a specified record of the provider's dataset.
	DataRequest (see page 1217)	Generates an OnDataRequest event on the specified provider component.
	Execute (see page 1218)	Executes the query or stored procedure bound to a specified provider.
	GetParams (see page 1219)	Fetches current parameter values from the dataset bound to a specified provider.
V	GetProvider (see page 1219)	Returns a registered provider component given its name.
	GetProviderNames (see page 1220)	Returns a list of all the available providers on the SOAP data module.
	GetRecords (see page 1220)	Returns a data packet that contains the specified data.
	RowRequest (see page 1222)	Returns information from a specified record of the provider's dataset.
V	SAS_ApplyUpdates (see page 1223)	Applies updates received from a client using a specified provider.
V	SAS_DataRequest (see page 1224)	Generates an OnSAS_DataRequest event on the specified provider component.
V	SAS_Execute (see page 1225)	SAS_Executes the query or stored procedure bound to a specified provider.
V	SAS_GetParams (see page 1226)	Fetches current parameter values from the dataset bound to a specified provider.
V	SAS_GetProviderNames (see page 1226)	Returns a list of all the available providers on the SOAP data module.
V	SAS_GetRecords (see page 1227)	Returns a data packet that contains the specified data.
V	SAS_RowRequest (see page 1228)	Returns information from a specified record of the provider's dataset.

Public Methods

	Name	Description
	_AddRef (see page 1210)	Increments the reference count for this data module's interface.
	_Release (see page 1210)	Decrements the reference count for this data module's interface.
V	AfterConstruction (see page 1210)	Generates an OnCreate event.
V	BeforeDestruction (see page 1217)	Generates an OnDestroy event.
V	Lock (see page 1221)	Locks the SOAP data module so that it can't be locked by other threads.
V	NewInstance (see page 1221)	Allocates memory for each instance of TSoapDataModule (see page 1206).
	RegisterProvider (see page 1222)	Adds a provider to the list returned by AS_GetProviderNames (see page 1215).
V	SafeCallException (see page 1223)	Handles exceptions in methods declared using the safecall calling convention.
V	Unlock (see page 1229)	Unlocks the SOAP data module after it was locked by a call to the Lock (see page 1221) method.
	UnRegisterProvider (see page 1229)	Removes a provider from the list returned by AS_GetProviderNames (see page 1215).

15.1.1.4.1 TSoapDataModule._AddRef Method

Increments the reference count for this data module's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

Do not call the __AddRef method directly. This method is called when an application obtains a reference to the SOAP data module's interface pointer. __AddRef increments the RefCount (see page 1231) property.

When the caller is through with the interface, it calls _Release (see page 1210). This allows the data module to free itself when it is no longer used (when the reference count goes to zero).

See Also

_Release (see page 1210)

RefCount (see page 1231)

IAppServer

15.1.1.4.2 TSoapDataModule._Release Method

Decrements the reference count for this data module's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

Do not call the __Release method directly. This method is called when an application releases a reference to the SOAP data module's interface pointer. __Release decrements the RefCount (see page 1231) property.

If the SOAP data module does not have an Owner, that is, if the value of the Owner property is nil (Delphi) or NULL (C++), then if the call to __Release changes the value of RefCount (see page 1231) to zero, the SOAP data module automatically frees itself.

See Also

_AddRef (see page 1210)

RefCount (see page 1231)

IAppServer

Owner

15.1.1.4.3 TSoapDataModule.AfterConstruction Method

Generates an OnCreate event.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called after the SOAP data module's constructor has finished. Do not call it explicitly in applications.

AfterConstruction generates an OnCreate event when OldCreateOrder is false. Then, after the OnCreate event handler exits, it decrements the reference count, which is artificially incremented when the data module is first allocated to prevent the data module from accidentally freeing itself from within the constructor because of a zero reference count.

See Also

BeforeDestruction (see page 1217)

OnCreate

OldCreateOrder

RefCount (see page 1231)

15.1.1.4.4 TSoapDataModule.ApplyUpdates Method

Applies updates received from a client using a specified provider.

Pascal

```
function ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors: Integer;
out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int MaxErrors,
int ErrorCount, OleVariant OwnerData);
```

Description

ApplyUpdates provides the underlying implementation for the AS_ApplyUpates method (on the IAppServer interface) and the SAS_ApplyUpdates (see page 1223) method (on the IAppServerSoap interface). It writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant (see page 3421) containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before rolling back the update operation

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeApplyUpdates event handler and returned to the client application in an AfterApplyUpdates event handler. It is passed to the provider's BeforeApplyUpdates event handler and returned from the provider's AfterApplyUpdates event handler.

ApplyUpdates returns a Variant (see page 3421) that is a data packet containing all records that could not be applied.

See Also

AS_ApplyUpdates (see page 1211)

SAS_ApplyUpdates (see page 1223)

15.1.1.4.5 TSoapDataModule.AS_ApplyUpdates Method

Applies updates received from a client using a specified provider.

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

AS_AS_ApplyUpdates writes client updates to a dataset or database server. ProviderName specifies the name of the provider component that manages the update operation. Delta is a Variant (see page 3421) containing the updated, inserted, and deleted records to write to the database. MaxErrors indicates the maximum number of errors to permit before rolling back the update operation; ErrorCount returns the actual number of errors encountered during the update operation. OwnerData is information supplied by the client application in a BeforeAS_ApplyUpdates event handler and returned to the client application in an AfterAS_ApplyUpdates event handler. It is passed to the provider's BeforeAS_ApplyUpdates event handler and returned from the provider's AfterAS_ApplyUpdates event handler.

Note: In Delphi, AS_AS_ApplyUpdates returns a Variant (see page 3421) that is a data packet containing all records that could not be applied.

Note: In C++, the AS_ApplyUpdates_result parameter returns the Variant (see page 3421) containing the data packet.

Note: Applications can only call the protected AS_AS_ApplyUpdates method using the IAppServer interface. To apply updates directly in code from the application server, use the provider component's AS_ApplyUpdates method.

See Also

ApplyUpdates

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

Providers (see page 1230)

IAppServer

15.1.1.4.6 TSoapDataModule.AS_DataRequest Method

Generates an OnAS_DataRequest event on the specified provider component.

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use AS_AS_DataRequest to invoke an OnAS_DataRequest event handler of the provider named by ProviderName. This

method is provided to allow application developers to customize the communication between a client application and a provider component. There is no predefined meaning for the Data parameter or the return value (Delphi) or AS_DataRequest_result parameter (C++).

Note: Applications can only call the protected AS_AS_DataRequest method using the IAppServer interface.

See Also

OnDataRequest

Providers (see page 1230)

DataRequest

IAppServer

15.1.1.4.7 TSoapDataModule.AS_Execute Method

AS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Use AS_AS_Execute to pass an execute command to a remote provider, using a specified set of parameters. AS_AS_Execute calls the AS_Execute method of the specified provider after assigning any parameters. Output (see page 3833) parameters that result from executing a query or stored procedure are returned in the Params parameter.

Note: AS_AS_Execute should only be called for a provider whose dataset represents a query or stored procedure that does not return a cursor.

The ProviderName parameter specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

The Params parameter encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

The OwnerData parameter contains custom information that appears as an argument to the provider's BeforeAS_Execute and AfterAS_Execute event handlers. This information originates in a client dataset's BeforeAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterAS_Execute event handler.

Note: Applications can only call the protected AS_AS_Execute method using the IAppServer interface. To execute a dataset in code from the application server, use the dataset component directly.

See Also

DataSet

BeforeExecute

AfterExecute

Params

Execute

Execute

Providers (see page 1230)

IAppServer

BeforeExecute

AfterExecute

PackageParams

Options

15.1.1.4.8 TSoapDataModule.AS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use AS_AS_GetParams to retrieve the current parameter values from the dataset associated with a provider. The ProviderName parameter specifies the name of the provider component whose dataset has the parameters. The OwnerData represents custom information that originates in a client dataset's BeforeAS_GetParams event handler and returns information that is passed to the client dataset's AfterAS_GetParams event handler.

Note: In Delphi, AS_AS_GetParams returns the parameters, encoded as a Variant (see page 3421) array, or, if the provider does not support parameter fetching, a Null (see page 5056) Variant (see page 3421).

Note: In C++, the AS_GetParams_result parameter the Variant (see page 3421) array containing the parameters (or the Null (see page 5056) Variant (see page 3421)).

Use the UnpackParams procedure to convert parameter values returned by AS_AS_GetParams to a TParams object.

Warning: It is not a good idea to call AS_AS_GetParams on a SOAP data module. SOAP data modules are stateless, and any parameter values you fetch may be changed by other applications. The recommended way to obtain output parameter values is to call the AS_Execute (see page 1213) method instead.

Note: Applications can only call the protected AS_AS_GetParams method using the IAppServer interface.

See Also

DataSet

Params

FetchParams

GetParams

BeforeGetParams

AfterGetParams

Providers (see page 1230)

IAppServer

AS_Execute (see page 1213)

UnpackParams

15.1.1.4.9 TSoapDataModule.AS_GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

Call AS_AS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServer interface, pass one of these names to indicate the provider that is the target of the method call.

AS_AS_GetProviderNames returns the names of the providers in a Variant (see page 3421) array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected AS_AS_GetProviderNames method using the IAppServer interface.

See Also

ProviderName

Providers (see page 1230)

IAppServer

Exported

15.1.1.4.10 TSoapDataModule.AS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call AS_AS_GetRecords to retrieve a specified number of records. AS_AS_GetRecords returns the requested records, starting with the current record of the provider's dataset. Because SOAP data modules are stateless, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeAS_GetRecords event handler.

ProviderName gives the name of the provider component that provides the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant (see page 3421) containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_GetRecords event handler. This information is passed to the provider's BeforeAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterAS_GetRecords event handler.

Records are returned as a data packet in a Variant (see page 3421) (as the return value in Delphi or the AS_GetRecords_result parameter in C++).

Note: Applications can only call the protected AS_AS_GetRecords method using the IAppServer interface. To fetch records directly in code from the application server, use the provider component's AS_GetRecords method.

See Also

Data

Params

GetNextPacket

GetRecords

BeforeGetRecords

AfterGetRecords

Providers (see page 1230)

IAppServer

15.1.1.4.11 TSoapDataModule.AS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

Client datasets call AS_AS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

The ProviderName parameter indicates the provider associated with the dataset from which information should be fetched.

The Row parameter is an OleVariant (see page 3274) that describes the current record on the client dataset.

The RequestType parameter indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeAS_RowRequest event handler. This information is passed to the provider's BeforeAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterAS_RowRequest event handler.

The requested data is returned as a delta packet.

Note: Applications can only call the protected AS_AS_RowRequest method using the IAppServer interface.

See Also

RowRequest

BeforeRowRequest

AfterRowRequest

FetchBlobs

FetchDetails

RefreshRecord

Providers (see page 1230)

IAppServer

15.1.1.4.12 TSoapDataModule.BeforeDestruction Method

Generates an OnDestroy event.

Pascal

```
procedure BeforeDestruction; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

BeforeDestruction is called immediately before the data module's destructor. Do not call it explicitly in applications.

BeforeDestruction generates an OnDestroy event when OldCreateOrder is false.

See Also

AfterConstruction (see page 1210)

OnDestroy

OldCreateOrder

15.1.1.4.13 TSoapDataModule.DataRequest Method

Generates an OnDataRequest event on the specified provider component.

Pascal

```
function DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

DataRequest provides the underlying implementation for the AS_DataRequest (see page 1212) method (on the IAppServer interface) and the SAS_DataRequest (see page 1224) method (on the IAppServerSOAP (see page 1340) interface). This method calls any OnDataRequest event handler of the specified provider.

There is no predefined meaning for the Data parameter. It represents custom information supplied by the client and passed to the OnDataRequest event handler.

DataRequest returns the value that is assigned to the OwnerData parameter of the event handler on exit.

See Also

AS_DataRequest (see page 1212)

SAS_DataRequest (see page 1224)

OnDataRequest

15.1.1.4.14 TSoapDataModule.Execute Method

Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure Execute(const ProviderName: WideString; const CommandText: WideString; var Params:
OleVariant; var OwnerData: OleVariant);
```

C++

```
__fastcall Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

Execute provides the underlying implementation for the AS_Execute (see page 1213) method (on the IAppServer interface) and the SAS_Execute (see page 1225) method (on the IAppServerSOAP (see page 1340) interface). It calls the Execute method of the specified provider after assigning any parameters. Output (see page 3833) parameters that result from executing a query or stored procedure are returned in Params.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

OwnerData contains custom information that appears as an argument to the provider's BeforeExecute and AfterExecute event handlers. This information originates in a client dataset's BeforeExecute event handler. The value returned in OwnerData is passed to the client dataset's AfterExecute event handler.

See Also

AS_Execute (see page 1213)

SAS_Execute (see page 1225)

15.1.1.4.15 TSoapDataModule.GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

GetParams provides the underlying implementation of the AS_GetParams (see page 1214) method (on the IAppServer interface) and the SAS_GetParams (see page 1226) method (on the IAppServerSOAP (see page 1340) interface).

ProviderName specifies the name of the provider component whose dataset has the parameters to fetch.

OwnerData represents custom information that originates in a client dataset's BeforeGetParams event handler and returns information that is passed to the client dataset's AfterGetParams event handler.

GetParams returns the parameters, encoded as a Variant (see page 3421) array, or, if the provider does not support parameter fetching, GetParams returns a Null (see page 5056) Variant (see page 3421). Use the UnpackParams procedure to convert parameter values returned by GetParams to a TParams object.

See Also

SAS_GetParams (see page 1226)

AS_GetParams (see page 1214)

15.1.1.4.16 TSoapDataModule.GetProvider Method

Returns a registered provider component given its name.

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const UnicodeString ProviderName);
```

Description

GetProvider is the protected read implementation of the Providers (see page 1230) property.

ProviderName specifies the name of a provider that is returned. If ProviderName is not one of the names returned by AS_GetProviderNames (see page 1215) (for example, if the provider's Exported property is false), GetProvider raises an exception.

See Also

Providers (see page 1230)

Exported

AS_GetProviderNames (see page 1215)

15.1.1.4.17 TSoapDataModule.GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function GetProviderNames: OleVariant;
```

C++

```
__fastcall OleVariant GetProviderNames();
```

Description

GetProviderNames provides the underlying implementation for the AS_GetProviderNames (see page 1215) method (on the IAppServer interface) and the SAS_GetProviderNames (see page 1226) method (on the IAppServerSOAP (see page 1340) interface). It returns a list of all registered providers available from a SOAP data module.

GetProviderNames returns the names of the providers in a Variant (see page 3421) array.

See Also

AS_GetProviderNames (see page 1215)

SAS_GetProviderNames (see page 1226)

15.1.1.4.18 TSoapDataModule.GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetRecords(const BSTR ProviderName, int Count, int RecsOut, int Options,
const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

GetRecords provides the underlying implementation for the AS_GetRecords (see page 1215) method (on the IAppServer interface) and the SAS_GetRecords (see page 1227) method (on the IAppServerSOAP (see page 1340) interface). GetRecords returns the requested records, starting with the current record of the provider's dataset.

- ProviderName gives the name of the provider component that supplies the records.
- Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.
- RecsOut returns the actual number of records retrieved.
- Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants.:
- CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.
- Params is a Variant (see page 3421) containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.
- OwnerData contains custom information that is supplied by a client dataset's BeforeGetRecords event handler. This information is passed to the provider's BeforeGetRecords event handler. OwnerData returns information supplied by the provider's AfterGetRecords event handler.

- Records are returned as a data packet in a Variant (see page 3421).

See Also

AS_GetRecords (see page 1215)

SAS_GetRecords (see page 1227)

15.1.1.4.19 TSoapDataModule.Lock Method

Locks the SOAP data module so that it can't be locked by other threads.

Pascal

```
procedure Lock; virtual;
```

C++

```
virtual __fastcall Lock();
```

Description

SOAP data modules use Lock internally to ensure that all IAppServer calls are thread-safe. When implementing a SOAP data module descendant, call Lock before accessing any instance data, including the providers contained in the data module. When finished with the block of code that must be protected from thread conflicts, call the Unlock (see page 1229) method to release the lock.

Lock uses a critical section to guard against thread conflicts. This means that any code that accesses the SOAP data module's properties or contained objects must call Lock or it will introduce thread conflicts.

See Also

Providers (see page 1230)

Unlock (see page 1229)

IAppServer

TCriticalSection (see page 3125)

15.1.1.4.20 TSoapDataModule.NewInstance Method

Allocates memory for each instance of TSoapDataModule (see page 1206).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TSoapDataModule (see page 1206) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 1210) method decrements the reference count after all constructors have executed.

See Also

System::TObject::NewInstance (see page 3379)

System::TObject::InstanceSize (see page 3377)

AfterConstruction (see page 1210)

15.1.1.4.21 TSoapDataModule.RegisterProvider Method

Adds a provider to the list returned by AS_GetProviderNames (see page 1215).

Pascal

```
procedure RegisterProvider(Value: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Value);
```

Description

Call RegisterProvider to make a provider available through the SOAP data module's IAppServer interface. Pass a reference to the provider as the Value parameter. Once a provider is registered, its availability to client applications can be turned on or off using its Exported property.

There is no need to call RegisterProvider for providers that are added at design time or when the SOAP data module is assigned as an Owner in the provider's constructor. These are registered automatically.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider (see page 1229) method. Providers (see page 1230) that have the SOAP data module as an Owner are unregistered automatically from their destructors.

See Also

UnRegisterProvider (see page 1229)

AS_GetProviderNames (see page 1215)

IAppServer

Create

Exported

15.1.1.4.22 TSoapDataModule.RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer; var
OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

RowRequest provides the underlying implementation for the AS_RowRequest (see page 1216) method (on the IAppServer interface) and the SAS_RowRequest (see page 1228) method (on the IAppServerSOAP (see page 1340) interface).

ProviderName indicates the provider associated with the dataset from which information should be fetched.

Row is an OleVariant (see page 3274) that describes the current record on the client dataset.

RequestType indicates the type of information required. It is an integer version of the TFetchOptions type.

OwnerData contains custom information that is supplied by a client dataset's BeforeRowRequest event handler. This information

is passed to the provider's BeforeRowRequest event handler. OwnerData returns custom information supplied by the provider's AfterRowRequest event handler.

The requested data is returned as a delta packet.

See Also

AS_RowRequest (see page 1216)

SAS_RowRequest (see page 1228)

15.1.1.4.23 TSoapDataModule.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException is used for handling exceptions that are raised by methods such as the AS... methods, which use the safecall calling convention. This method ensures that the SOAP fault packet includes all the necessary error information so that clients can respond appropriately to exceptions in safecall methods.

ExceptObject (see page 3562) is the exception that was raised.

ExceptAddr (see page 3562) is the address where the exception occurred.

SafeCallException returns an error code that indicates what error occurred.

See Also

SafeCallException

15.1.1.4.24 TSoapDataModule.SAS_ApplyUpdates Method

Applies updates received from a client using a specified provider.

Pascal

```
function SAS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_ApplyUpdates(const BSTR ProviderName, OleVariant
Delta, int MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

SAS_SAS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant (see page 3421) containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before rolling back the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeSAS_ApplyUpdates event handler and returned to the

client application in an AfterSAS_ApplyUpdates event handler. It is passed to the provider's BeforeSAS_ApplyUpdates event handler and returned from the provider's AfterSAS_ApplyUpdates event handler.

SAS_SAS_ApplyUpdates returns a Variant (see page 3421) that is a data packet containing all records that could not be applied.

Note: Applications can only call the protected SAS_SAS_ApplyUpdates method using the IAppServerSOAP (see page 1340) interface. To apply updates directly in code from the application server, use the provider component's SAS_ApplyUpdates method.

See Also

ApplyUpdates

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

Providers (see page 1230)

IAppServerSOAP (see page 1340)

15.1.1.4.25 TSoapDataModule.SAS_DataRequest Method

Generates an OnSAS_DataRequest event on the specified provider component.

Pascal

```
function SAS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_DataRequest(const BSTR ProviderName, OleVariant
Data);
```

Description

Use SAS_SAS_DataRequest to invoke an OnSAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component.

ProviderName is the provider on which the data module generates an OnSAS_DataRequest event.

There is no predefined meaning for the Data parameter or the return value.

Note: Applications can only call the protected SAS_SAS_DataRequest method using the IAppServerSOAP (see page 1340) interface.

See Also

OnDataRequest

Providers (see page 1230)

DataRequest

IAppServerSOAP (see page 1340)

15.1.1.4.26 TSoapDataModule.SAS_Execute Method

SAS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure SAS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall SAS_Execute(const BSTR ProviderName, const BSTR CommandText,
OleVariant Params, OleVariant OwnerData);
```

Description

Use SAS_SAS_Execute to pass an execute command to a remote provider, using a specified set of parameters. SAS_SAS_Execute calls SAS_Execute, which calls the SAS_Execute method of the specified provider after assigning any parameters. Output (see page 3833) parameters that result from executing a query or stored procedure are returned in the Params parameter.

Note: SAS_SAS_Execute should only be called for a provider whose dataset represents a query or stored procedure that does not return a cursor.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a stored procedure that replaces the provider's underlying stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method.

OwnerData contains custom information that appears as an argument to the provider's BeforeSAS_Execute and AfterSAS_Execute event handlers. This information originates in a client dataset's BeforeSAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterSAS_Execute event handler.

Note: Applications can only call the protected SAS_SAS_Execute method using the IAppServerSOAP (see page 1340) interface. To execute a dataset in code from the application server, use the dataset component directly.

See Also

DataSet

BeforeExecute

AfterExecute

Params

Execute

Execute

Providers (see page 1230)

IAppServerSOAP (see page 1340)

BeforeExecute

AfterExecute

PackageParams

Options

15.1.1.4.27 TSoapDataModule.SAS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function SAS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_GetParams(const BSTR ProviderName, OleVariant
OwnerData);
```

Description

Use SAS_SAS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

ProviderName specifies the name of the provider component whose dataset has the parameters.

OwnerData represents custom information that originates in a client dataset's BeforeSAS_GetParams event handler and returns information that is passed to the client dataset's AfterSAS_GetParams event handler.

SAS_SAS_GetParams returns the parameters, encoded as a Variant (see page 3421) array, or, if the provider does not support parameter fetching, SAS_SAS_GetParams returns a Null (see page 5056) Variant (see page 3421). Use the UnpackParams procedure to convert parameter values returned by SAS_SAS_GetParams to a TParams object.

Warning: It is not a good idea to call SAS_SAS_GetParams on a SOAP data module. SOAP data modules are stateless, and any parameter values you fetch may be changed by other applications. The recommended way to obtain output parameter values is to call the SAS_Execute (see page 1225) method instead.

Note: Applications can only call the protected SAS_SAS_GetParams method using the IAppServerSOAP (see page 1340) interface.

See Also

DataSet

Params

FetchParams

GetParams

BeforeGetParams

AfterGetParams

Providers (see page 1230)

IAppServerSOAP (see page 1340)

SAS_Execute (see page 1225)

UnpackParams

15.1.1.4.28 TSoapDataModule.SAS_GetProviderNames Method

Returns a list of all the available providers on the SOAP data module.

Pascal

```
function SAS_GetProviderNames: TWideStringDynArray; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall TWideStringDynArray SAS_GetProviderNames();
```

Description

Call SAS_SAS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServerSOAP (see page 1340) interface, pass one of these names to indicate the provider that is the target of the method call.

SAS_SAS_GetProviderNames returns the names of the providers in a dynamic array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

Note: Applications can only call the protected SAS_SAS_GetProviderNames method using the IAppServerSOAP (see page 1340) interface.

See Also

ProviderName

Providers (see page 1230)

IAppServerSOAP (see page 1340)

Exported

15.1.1.4.29 TSoapDataModule.SAS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function SAS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_GetRecords(const BSTR ProviderName, int Count, int
RecsOut, int Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call SAS_SAS_GetRecords to retrieve a specified number of records. SAS_SAS_GetRecords returns the requested records, starting with the current record of the provider's dataset. Because SOAP data modules are stateless, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeSAS_GetRecords event handler.

ProviderName gives the name of the provider component that supplies the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or

stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant (see page 3421) containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeSAS_GetRecords event handler. This information is passed to the provider's BeforeSAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterSAS_GetRecords event handler.

Records are returned as a data packet in a Variant (see page 3421).

Note: Applications can only call the protected SAS_SAS_GetRecords method using the IAppServerSOAP (see page 1340) interface. To fetch records directly in code from the application server, use the provider component's SAS_GetRecords method.

See Also

Data

Params

GetNextPacket

GetRecords

BeforeGetRecords

AfterGetRecords

Providers (see page 1230)

IAppServerSOAP (see page 1340)

15.1.1.4.30 TSoapDataModule.SAS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function SAS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall OleVariant SAS_RowRequest(const BSTR ProviderName, OleVariant
Row, int RequestType, OleVariant OwnerData);
```

Description

Client datasets call SAS_SAS_RowRequest automatically to implement their FetchBlobs, FetchDetails , or RefreshRecord method.

ProviderName indicates the provider associated with the dataset from which information should be fetched.

Row is an OleVariant (see page 3274) that describes the current record on the client dataset.

RequestType indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeSAS_RowRequest event handler. This information is passed to the provider's BeforeSAS_RowRequest event handler. OwnerData returns custom information supplied

by the provider's AfterSAS_RowRequest event handler.

The requested data is returned as a delta packet.

Note: Applications can only call the protected SAS_SAS_RowRequest method using the IAppServerSOAP (see page 1340) interface.

See Also

RowRequest

BeforeRowRequest

AfterRowRequest

FetchBlobs

FetchDetails

RefreshRecord

Providers (see page 1230)

IAppServerSOAP (see page 1340)

15.1.1.4.31 TSoapDataModule.Unlock Method

Unlocks the SOAP data module after it was locked by a call to the Lock (see page 1221) method.

Pascal

```
procedure Unlock; virtual;
```

C++

```
virtual __fastcall Unlock();
```

Description

Use Unlock to release the critical section that was entered by a call to the Lock (see page 1221) method. Every call to the Lock (see page 1221) method must be paired with a corresponding call to Unlock.

See Also

Lock (see page 1221)

15.1.1.4.32 TSoapDataModule.UnRegisterProvider Method

Removes a provider from the list returned by AS_GetProviderNames (see page 1215).

Pascal

```
procedure UnRegisterProvider(Value: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Value);
```

Description

Call UnRegisterProvider to remove a provider from the SOAP data module's list of available providers. Pass a reference to the provider that should be removed as the Value parameter. When a provider is not registered, it is not returned by the AS_GetProviderNames (see page 1215) method, which can be accessed through the IAppServer interface.

To temporarily disable a provider, use the provider's Exported property. Unexported providers can't be accessed through any IAppServer method calls.

Before a registered provider is destroyed, it must be unregistered using the UnRegisterProvider method. Providers (see page 1230) that have the SOAP data module as an Owner are unregistered automatically from their destructors.

See Also

RegisterProvider (see page 1222)

AS_GetProviderNames (see page 1215)

IAppServer

Destroy

Exported

15.1.1.5 TSoapDataModule Properties

The properties of the TSoapDataModule class are listed here.

Public Properties

	Name	Description
	ProviderCount (see page 1230)	Indicates the number of providers registered with the SOAP data module.
	Providers (see page 1230)	Lists the registered provider components that are contained in the SOAP data module.
	RefCount (see page 1231)	Reference counts the data module's interface.

15.1.1.5.1 TSoapDataModule.ProviderCount Property

Indicates the number of providers registered with the SOAP data module.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

ProviderCount is the number of providers that can be accessed using the Providers (see page 1230) property.

See Also

Providers (see page 1230)

RegisterProvider (see page 1222)

15.1.1.5.2 TSoapDataModule.Providers Property

Lists the registered provider components that are contained in the SOAP data module.

Pascal

```
property Providers [const ProviderName: string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[UnicodeString const ProviderName];
```

Description

Use Providers to access a provider component from the SOAP data module given its name. The SOAP data module uses

Providers to locate the target of method calls made using the IAppServer interface.

If the value passed as is not one of the names returned by AS_GetProviderNames (see page 1215) (for example, if the provider's Exported property is false) reading Providers raises an exception.

See Also

AS_GetProviderNames (see page 1215)

Exported

RegisterProvider (see page 1222)

ProviderCount (see page 1230)

15.1.1.5.3 TSoapDataModule.RefCount Property

Reference counts the data module's interface.

Pascal

```
property RefCount: Integer;
```

C++

```
__property int RefCount;
```

Description

RefCount keeps track of the number of times the application takes a reference to the SOAP data module's interface. When application code obtains this data module's interface, a call is generated to the _AddRef (see page 1210) method, which increments RefCount. When application code releases the interface, a call is generated to the _Release (see page 1210) method, which decrements RefCount. When RefCount drops to zero, the SOAP data module frees itself if its Owner property is not nil (Delphi) or NULL (C++).

See Also

_AddRef (see page 1210)

_Release (see page 1210)

Owner

16 SOAPDomConv Namespace

This is namespace SOAPDomConv.

Topics

Name	Description
Classes (see page 1234)	The following table lists classes in this documentation.
Interfaces (see page 1241)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 1244)	The following table lists structs, records, enums in this documentation.
Types (see page 1245)	The following table lists types in this documentation.

Classes

	Name	Description
	TSOAPDOMProcessor (see page 1234)	TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

Interfaces

	Name	Description
	IDOMHeaderProcessor (see page 1241)	IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

Structs, Records, Enums

	Name	Description
	TDOMHeaderProcessorEntry (see page 1244)	This is record SOAPDomConv.TDOMHeaderProcessorEntry.

Types

Name	Description
TDOMHeaderProcessorArray (see page 1245)	This is type SOAPDomConv.TDOMHeaderProcessorArray.

16.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TSOAPDOMProcessor (see page 1234)	TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

16.1.1 TSOAPDOMProcessor Class

TSOAPDOMProcessor is the base class for components that process the SOAP headers from SOAP requests.

Class Hierarchy

Pascal

```
TSOAPDOMProcessor = class(TComponent, IInterface);
```

C++

```
class TSOAPDOMProcessor : public TComponent, public IInterface;
```

File

SOAPDomConv

Description

TSOAPDOMProcessor is the base class for DOM-based components that interpret and create SOAP method calls. It manages a list of interfaces that it uses to create or interpret specialized headers in the SOAP message. Such headers can include application-specific information such as authentication, compression/decompression, encryption/decryption, and so on.

Do not create instances of TSOAPDOMProcessor. It is intended only as a base class for components that use the information in SOAP headers when interpreting or creating SOAP method calls.

See Also

TOPToSoapDomConvert

16.1.1.1 TSOAPDOMProcessor Members

The following tables list the members exposed by TSOAPDOMProcessor.

Protected Fields

	Name	Description
	FHeaderProcessors (see page 1235)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

Protected Methods

	Name	Description
	_AddRef (see page 1236)	Implements the IInterface (see page 3502) __AddRef method.
	_Release (see page 1236)	Implements the IInterface (see page 3502) __Release method.
	FindHeaderProcessor (see page 1238)	Locates a processor to interpret or generate a specified SOAP header.
	MakeHeaderNodes (see page 1239)	Generates all the SOAP headers that this object knows how to create.

Public Methods

	Name	Description
	AddHeaderProcessor (see page 1237)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
	AfterConstruction (see page 1237)	Responds after the last constructor executes.
	DefaultProcessHeader (see page 1238)	Handles the interpretation of a SOAP header when no processor can be found.
	Destroy (see page 1235)	Frees the memory associated with the TSOAPDOMProcessor (see page 1234) object.
	NewInstance (see page 1239)	Allocates memory for each instance of TSOAPDOMProcessor (see page 1234).
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

16.1.1.2 TSOAPDOMProcessor.Destroy Destructor

Frees the memory associated with the TSOAPDOMProcessor (see page 1234) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSOAPDOMProcessor();
```

Description

Do not call Destroy to free an instance of TSOAPDOMProcessor (see page 1234). Instead, use the Free method, which first checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3372)

16.1.1.3 TSOAPDOMProcessor Fields

The fields of the TSOAPDOMProcessor class are listed here.

Protected Fields

	Name	Description
	FHeaderProcessors (see page 1235)	This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

16.1.1.3.1 TSOAPDOMProcessor.FHeaderProcessors Field

Pascal

```
FHeaderProcessors: TDOMHeaderProcessorArray;
```

C++

```
TDOMHeaderProcessorArray FHeaderProcessors;
```

Description

This is FHeaderProcessors, a member of class TSOAPDOMProcessor.

16.1.1.4 TSOAPDOMProcessor Methods

The methods of the TSOAPDOMProcessor class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 1236)	Implements the IInterface (see page 3502) __AddRef method.
	_Release (see page 1236)	Implements the IInterface (see page 3502) __Release method.
	FindHeaderProcessor (see page 1238)	Locates a processor to interpret or generate a specified SOAP header.
	MakeHeaderNodes (see page 1239)	Generates all the SOAP headers that this object knows how to create.

Public Methods

	Name	Description
	AddHeaderProcessor (see page 1237)	Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.
	AfterConstruction (see page 1237)	Responds after the last constructor executes.
	DefaultProcessHeader (see page 1238)	Handles the interpretation of a SOAP header when no processor can be found.
	NewInstance (see page 1239)	Allocates memory for each instance of TSOAPDOMProcessor (see page 1234).

16.1.1.4.1 TSOAPDOMProcessor._AddRef Method

Implements the IInterface (see page 3502) __AddRef method.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

Do not call the protected __AddRef method directly. __AddRef is called through the IInterface (see page 3502) interface to indicate that another object is using the TSOAPDOMProcessor (see page 1234) object's interface pointer. When the other object is through with the interface, it calls _Release (see page 1236) through the same interface. This allows the TSOAPDOMProcessor (see page 1234) object to free itself when it is no longer used (when the reference count goes to zero).

See Also

_Release (see page 1236)

16.1.1.4.2 TSOAPDOMProcessor._Release Method

Implements the IInterface (see page 3502) __Release method.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

Do not call the protected __Release method directly. __Release is called through the IInterface (see page 3502) interface to indicate that another object is through with the TSOAPDOMProcessor (see page 1234) object's interface pointer. This allows the TSOAPDOMProcessor (see page 1234) object to free itself when it is no longer used (when the reference count goes to zero).

See Also

_AddRef (see page 1236)

16.1.1.4.3 TSOAPDOMProcessor.AddHeaderProcessor Method

Adds a processor to the internal list of interfaces to use for interpreting and creating SOAP headers.

Pascal

```
procedure AddHeaderProcessor(Namespace: WideString; HeaderName: WideString; TypeName:
WideString; Processor: IDOMHeaderProcessor); virtual;
```

C++

```
virtual __fastcall AddHeaderProcessor(BSTR Namespace, BSTR HeaderName, BSTR TypeName,
IDOMHeaderProcessor Processor);
```

Description

AddHeaderProcessor stores a specified processor interface and the information necessary to identify the SOAP headers that it handles.

Namespace is the namespace URI in which the SOAP header identified by HeaderName is defined.

HeaderName is the name of the SOAP header that the processor handles.

TypeName is the name of the type that is used for the value of the header.

Processor is the interface for a helper object that creates or interprets SOAP headers that match Namespace, HeaderName, and TypeName.

Note: AddHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

FindHeaderProcessor (see page 1238)

16.1.1.4.4 TSOAPDOMProcessor.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of TSOAPDOMProcessor (see page 1234) is allocated, its reference count is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that reference count to re-establish the correct count after all constructors have executed.

See Also

System::TObject::BeforeDestruction (see page 3358)

NewInstance (see page 1239)

16.1.1.4.5 TSOAPDOMProcessor.DefaultProcessHeader Method

Handles the interpretation of a SOAP header when no processor can be found.

Pascal

```
procedure DefaultProcessHeader(HeaderNode: IXMLNode; var Handled: Boolean; var AbortRequest:
Boolean); virtual;
```

C++

```
virtual __fastcall DefaultProcessHeader(IXMLNode HeaderNode, Boolean Handled, Boolean
AbortRequest);
```

Description

DefaultProcessHeader is called to provide a default interpretation of a SOAP header when it is not processed by one of the processors that FindHeaderProcessor (see page 1238) returns.

HeaderNode is the interface of the DOM representation of the SOAP header to be processed.

Handled returns true if DefaultProcessHeader handles the header node.

AbortRequest returns true if DefaultProcessHeader determines that the SOAP header can't be handled and that this results in an inability to interpret the entire SOAP request.

Note: DefaultProcessHeader is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

FindHeaderProcessor (see page 1238)

16.1.1.4.6 TSOAPDOMProcessor.FindHeaderProcessor Method

Locates a processor to interpret or generate a specified SOAP header.

Pascal

```
function FindHeaderProcessor(Namespace: WideString; HeaderName: WideString; TypeName:
WideString): IDOMHeaderProcessor; virtual;
```

C++

```
virtual __fastcall IDOMHeaderProcessor FindHeaderProcessor(BSTR Namespace, BSTR HeaderName,
BSTR TypeName);
```

Description

FindHeaderProcessor returns the interface of the helper object that can generate or interpret a specified SOAP header. This processor must be previously added to the TSOAPDOMProcessor (see page 1234) object's internal list of processors by a call to the AddHeaderProcessor (see page 1237) method.

Namespace is the namespace URI in which the SOAP header identified by HeaderName is defined.

HeaderName is the name of the SOAP header that the processor handles.

TypeName is the name of the type that is used for the value of the header.

Note: FindHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

AddHeaderProcessor (see page 1237)

16.1.1.4.7 TSOAPDOMProcessor.MakeHeaderNodes Method

Generates all the SOAP headers that this object knows how to create.

Pascal

```
function MakeHeaderNodes(HeaderNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode MakeHeaderNodes(IXMLNode HeaderNode);
```

Description

MakeHeaderNodes uses the processors that have been added by the AddHeaderProcessor (see page 1237) method to generate a set of SOAP header nodes. These generated nodes are added as child nodes to the node specified by HeaderNode.

Note: MakeHeaderNodes is supplied for backward compatibility. Headers are handled differently, using lists of TSOAPHeader descendants that are supplied as parameters to methods in descendant classes.

See Also

AddHeaderProcessor (see page 1237)

16.1.1.4.8 TSOAPDOMProcessor.NewInstance Method

Allocates memory for each instance of TSOAPDOMProcessor (see page 1234).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

TSOAPDOMProcessor (see page 1234) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 1237) method decrements the reference count after all constructors have executed.

See Also

System::TObject::NewInstance (see page 3379)

System::TObject::InstanceSize (see page 3377)

AfterConstruction (see page 1237)

16.2 Interfaces

The following table lists interfaces in this documentation.

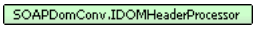
Interfaces

	Name	Description
	IDOMHeaderProcessor (see page 1241)	IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

16.2.1 IDOMHeaderProcessor Interface

IDOMHeaderProcessor is the interface for objects that interpret and create SOAP headers.

Class Hierarchy



Pascal

```
IDOMHeaderProcessor = interface;
```

C++

```
__interface IDOMHeaderProcessor;
```

File

SOAPDomConv

Description

IDOMHeaderProcessor provides an interface for working with specialized SOAP headers. Each object that implements IDOMHeaderProcessor handles a specific type of soap header. The TSOAPDOMProcessor (see page 1234) class calls on this interface when it needs to create or interpret the headers of a SOAP packet.

IDOMHeaderProcessor declares two methods, CreateHeader (see page 1242) and ProcessHeader (see page 1242), which create or respond to a given SOAP header, respectively.

Note: C++ method declarations that use IDOMHeaderProcessor use the _di_IDOMHeaderProcessor type instead. This type is a DelphiInterface (see page 3256) wrapper around the IDOMHeaderProcessor interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDOMHeaderProcessor >
_di_IDOMHeaderProcessor;
```

Note: IDOMHeaderProcessor is supplied for backward compatibility. Headers are handled differently, using the ISOAPHeaders interface.

See Also

TSOAPDOMProcessor (see page 1234)

ISOAPHeaders

16.2.1.1 IDOMHeaderProcessor Members

The following tables list the members exposed by IDOMHeaderProcessor.

Public Methods

	Name	Description
	CreateHeader (see page 1242)	Creates a new SOAP header node.
	ProcessHeader (see page 1242)	Interprets and responds to a SOAP header.

16.2.1.2 IDOMHeaderProcessor Methods

The methods of the IDOMHeaderProcessor class are listed here.

Public Methods

	Name	Description
	CreateHeader (see page 1242)	Creates a new SOAP header node.
	ProcessHeader (see page 1242)	Interprets and responds to a SOAP header.

16.2.1.2.1 IDOMHeaderProcessor.CreateHeader Method

Creates a new SOAP header node.

Pascal

```
function CreateHeader(HeaderNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateHeader(IXMLNode HeaderNode);
```

Description

CreateHeader creates a new header node.

HeaderNode is the node in the emerging SOAP packet that acts as a parent to the new header.

CreateHeader returns the new header node.

See Also

MakeHeaderNodes (see page 1239)

16.2.1.2.2 IDOMHeaderProcessor.ProcessHeader Method

Interprets and responds to a SOAP header.

Pascal

```
procedure ProcessHeader(HeaderNode: IXMLNode; var Handled: Boolean; var AbortRequest: Boolean);
```

C++

```
__fastcall ProcessHeader(IXMLNode HeaderNode, Boolean Handled, Boolean AbortRequest);
```

Description

ProcessHeader provides a response to a given SOAP header node.

HeaderNode is the interface of the DOM representation of the SOAP header to be processed.

Handled returns true if ProcessHeader handles the header node.

AbortRequest returns true if ProcessHeader determines that the SOAP header can't be handled and that this results in an inability to interpret the entire SOAP request. For example, if the header node represents authentication information, ProcessHeader could set AbortRequest to true if the header supplied an invalid password.

16.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TDOMHeaderProcessorEntry (see page 1244)	This is record SOAPDomConv.TDOMHeaderProcessorEntry.

16.3.1 SOAPDomConv.TDOMHeaderProcessorEntry Record

Pascal

```
TDOMHeaderProcessorEntry = record
  Processor: IDOMHeaderProcessor;
  NameSpace: WideString;
  HeaderName: WideString;
  TypeName: WideString;
end;
```

C++

```
struct TDOMHeaderProcessorEntry {
  IDOMHeaderProcessor Processor;
  BSTR NameSpace;
  BSTR HeaderName;
  BSTR TypeName;
};
```

File

SOAPDomConv

Description

This is record SOAPDomConv.TDOMHeaderProcessorEntry.

16.4 Types

The following table lists types in this documentation.

Types

Name	Description
TDOMHeaderProcessorArray (see page 1245)	This is type SOAPDomConv.TDOMHeaderProcessorArray.

16.4.1 SOAPDomConv.TDOMHeaderProcessorArray Type

Pascal

```
TDOMHeaderProcessorArray = array of TDOMHeaderProcessorEntry;
```

C++

```
array of TDOMHeaderProcessorEntry TDOMHeaderProcessorArray;
```

File

SOAPDomConv

Description

This is type SOAPDomConv.TDOMHeaderProcessorArray.

17 SOAPHTTPClient Namespace

This is namespace SOAPHTTPClient.

Topics

Name	Description
Classes (see page 1248)	The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPRIO (see page 1248)	THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

17.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPRIO (see page 1248)	THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

17.1.1 THTTPRIO Class

THTTPRIO uses HTTP messages to call remote interfaced objects using SOAP.

Class Hierarchy

Pascal

```
THTTPRIO = class(TRIO);
```

C++

```
class THTTPRIO : public TRIO;
```

File

SOAPHTTPClient

Description

Use THTTPRIO to generate statically-linked calls to invokable interfaces on a remote Web Service (see page 1253) application. When an application casts THTTPRIO to a registered invokable interface, it dynamically generates an in-memory method table, providing an implementation to that invokable interface. THTTPRIO executes the methods in this method table by encoding the method call as a SOAP request and sending an HTTP request message to the Web Service (see page 1253) application. It unpacks the resulting HTTP response message to obtain the return value and any output parameters, or to raise an exception if the request generated an exception on the server.

Use the published properties of THTTPRIO to indicate how connect to the Web Service (see page 1253) application. There are two ways to identify where the server application is located:

You can use the URL (see page 1254) property to specify the URL (see page 1254) where the server application is located,.

If you want to look up connection information dynamically at runtime from a WSDL document, you can set the WSDLLocation (see page 1255) property. After setting WSDLLocation (see page 1255), select values for the Service (see page 1253) and Port (see page 1253) properties from drop-down lists in the object inspector to fully specify the binding you want to use.

If you need to use a proxy server, or if the server requires authentication, use the properties of the THTTPReqResp (see page 1276) object that is the value of the HTTPWebNode (see page 1252) property to provide the necessary information.

Before THTTPRIO can generate a method table for an invokable interface, the interface must be registered with the invocation registry. To register an invokable interface, use the global InvRegistry function to access the registry, and use its RegisterInterface method.

See Also

InvRegistry

TInvokableClassRegistry

Calling Invokable Interfaces

Understanding Invokable Interfaces

17.1.1.1 THTTPRIO Members

The following tables list the members exposed by THTTPRIO.

Protected Methods

	Name	Description
	Notification (see page 1250)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Create (see page 1249)	Creates an instance of THTTPRIO (see page 1248).
	Destroy (see page 1250)	Destroys the instance of a THTTPRIO (see page 1248) object.
	QueryInterface (see page 1251)	Implements the IInterface (see page 3502) QueryInterface method.

Public Properties

	Name	Description
	WSDLItems (see page 1255)	Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 1255).

Published Properties

	Name	Description
	Converter (see page 1252)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	HTTPWebNode (see page 1252)	Handles the execution of method calls on the invokable interface.
	Port (see page 1253)	Specifies the port to use when sending the SOAP message to execute a method call.
	Service (see page 1253)	Specifies the Service to use when sending the SOAP message to execute a method call.
	URL (see page 1254)	Specifies the URL that identifies the target Web Service (see page 1253) application.
	WSDLLocation (see page 1255)	Specifies the location of the WSDL document that describes the target Web Service (see page 1253).

17.1.1.2 THTTPRIO.Create Constructor

Creates an instance of THTTPRIO (see page 1248).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THTTPRIO(TComponent * AOwner);
```

Description

Call Create to instantiate a THTTPRIO (see page 1248) object at runtime. THTTPRIO (see page 1248) components added to forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the THTTPRIO (see page 1248) instance. It becomes the value of the Owner property. If the THTTPRIO (see page 1248) instance is created with AOwner set to nil (Delphi) or NULL (C++), then it is automatically freed when its reference count drops to zero.

See Also

RefCount

Owner

17.1.1.3 THTTPRIO.Destroy Destructor

Destroys the instance of a THTTPRIO (see page 1248) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPRIO();
```

Description

There is no need to explicitly free a THTTPRIO (see page 1248) instance.

If the THTTPRIO (see page 1248) object is created with an Owner, then that Owner handles the freeing of the THTTPRIO (see page 1248) instance. If the THTTPRIO (see page 1248) object is created without an Owner, then it is automatically freed when the reference count on its interface drops to zero.

See Also

Create (see page 1249)

Owner

RefCount

17.1.1.4 THTTPRIO Methods

The methods of the THTTPRIO class are listed here.

Protected Methods

	Name	Description
	Notification (see page 1250)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	QueryInterface (see page 1251)	Implements the IInterface (see page 3502) QueryInterface method.

17.1.1.4.1 THTTPRIO.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

17.1.1.4.2 THTTPRIO.QueryInterface Method

Implements the IInterface (see page 3502) QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface (see page 3502) interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts THTTPRIO (see page 1248) to an invokable interface using the as operator.

THTTPRIO (see page 1248) can only return a single type of interface pointer. The first time an application calls QueryInterface, THTTPRIO (see page 1248) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this THTTPRIO (see page 1248) instance. If the THTTPRIO (see page 1248) object has already been typecast to an interface, then IID must identify that interface. If the THTTPRIO (see page 1248) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless THTTPRIO (see page 1248) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK (see page 3860)) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE (see page 3845) otherwise.

Note: In C++, a second, templatized syntax takes a variable that is a DelphiInterface (see page 3256) wrapper for the desired interface type.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry

RegisterInterface

RefCount

17.1.1.5 THTTPRIO Properties

The properties of the THTTPRIO class are listed here.

Public Properties

	Name	Description
	WSDLItems (see page 1255)	Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 1255).

Published Properties

	Name	Description
	Converter (see page 1252)	Handles the marshaling and unmarshaling of calls to an invokable interface.
	HTTPWebNode (see page 1252)	Handles the execution of method calls on the invokable interface.
	Port (see page 1253)	Specifies the port to use when sending the SOAP message to execute a method call.
	Service (see page 1253)	Specifies the Service to use when sending the SOAP message to execute a method call.
	URL (see page 1254)	Specifies the URL that identifies the target Web Service (see page 1253) application.
	WSDLLocation (see page 1255)	Specifies the location of the WSDL document that describes the target Web Service (see page 1253).

17.1.1.5.1 THTTPRIO.Converter Property

Handles the marshaling and unmarshaling of calls to an invokable interface.

Pascal

```
property Converter: TOPToSoapDomConvert;
```

C++

```
__property TOPToSoapDomConvert Converter;
```

Description

THTTPRIO (see page 1248) uses Converter internally to marshal and unmarshal calls to an invokable interface. That is, when the THTTPRIO (see page 1248) object has been typecast to an invokable interface, it implements each method call by using Converter to marshal the call from an internal format into an encoding protocol, and to unmarshal any return values after that call is executed.

See Also

HTTPWebNode (see page 1252)

17.1.1.5.2 THTTPRIO.HTTPWebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property HTTPWebNode: THTTPReqResp;
```

C++

```
__property THTTPReqResp HTTPWebNode;
```

Description

THTTPRIO (see page 1248) uses HTTPWebNode internally to execute method calls. It is the implementation of the WebNode (see page 6615) property.

THTTPRio executes a method call by sending a SOAP request as an HTTP message to the Web Service (see page 1253) identified by either the URL (see page 1254) property or the WSDLLocation (see page 1255), Service (see page 1253), and Port (see page 1253) properties. If you need to use a proxy server to access the Web Service (see page 1253) application, or if the server requires authentication, use the properties of the THTTPReqResp (see page 1276) object to provide the necessary information.

See Also

WebNode

URL (see page 1254)

WSDLLocation (see page 1255)

Service (see page 1253)

Port (see page 1253)

17.1.1.5.3 THTTPRIO.Port Property

Specifies the port to use when sending the SOAP message to execute a method call.

Pascal

```
property Port: string;
```

C++

```
__property UnicodeString Port;
```

Description

Set Port to indicate the desired port within the service specified by the Service (see page 1253) property. Port is the name of a port defined in the WSDL document specified by the WSDLLocation (see page 1255) property. Once you have set the WSDLLocation (see page 1255) property, you can select a value for Port from a drop-down list in the Object Inspector.

A port defines a binding (which indicates the interface to call and the encoding and transport protocols to use) and the address of the server machine.

See Also

WSDLLocation (see page 1255)

Service (see page 1253)

HTTPWebNode (see page 1252)

17.1.1.5.4 THTTPRIO.Service Property

Specifies the Service to use when sending the SOAP message to execute a method call.

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

Set Service to indicate the Service that you want to use by making a remote method call. Service is the name of a Service defined in the WSDL document specified by the WSDLLocation (see page 1255) property. Once you have set the WSDLLocation (see page 1255) property, you can select a value for Service from a drop-down list in the Object Inspector.

A Service is a set of related interfaces provided by a Web Service application. The interface that THTTPRIO (see page 1248) represents must be on one of the interfaces that make up the specified service.

Note: When defining the target Web Service application by referring to a WSDL document, you must set the Port (see page 1253) property as well as the Service and WSDLLocation (see page 1255) properties.

See Also

WSDLLocation (see page 1255)

Port (see page 1253)

HTTPWebNode (see page 1252)

17.1.1.5.5 THTTPRIO.URL Property

Specifies the URL that identifies the target Web Service (see page 1253) application.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

Use URL to identify the target application that implements the invokable interface when there is no need to look up connection information from a WSDL document dynamically at runtime. If the server was written using the VCL, the path portion of this URL must match the target path name of the THTTPSoapDispatcher component on the Web Service (see page 1253) application. The protocol portion must specify http or https. Thus, for servers written in the VCL, URL has the form

`http://MyHost.com/scripts/MyWebService.dllso/SOAP/IMyInterface`

When you use URL to identify the server, THTTPRIO (see page 1248) obtains a namespace for the target interface from the invocation registry. It is necessary, therefore, that the target interface be registered with the same namespace in both client and server applications

Note: In Delphi, this happens automatically if you

Use the same unit to declare and register the target interface in both client and server applications.

Use the same value for the AppNamespacePrefix global variable.

Use the automatically-generated namespace.

If you are using C++, or if one of the above criteria is not met, you can ensure that the target interface is registered with the same namespace by explicitly specifying a namespace in the code that registers the interface.

Note: If you want your application to look up connection information from a WSDL document dynamically at runtime, you must use the WSDLLocation (see page 1255), Service (see page 1253), and Port (see page 1253) properties instead. Using the URL property is mutually exclusive with the use of WSDLLocation (see page 1255), Service (see page 1253) and Port (see page 1253).

See Also

WSDLLocation (see page 1255)

WebDispatch (see page 5950)

HTTPWebNode (see page 1252)

AppNameSpacePrefix

17.1.1.5.6 THTTPRIO.WSDLItems Property

Provides access to the definitions in the WSDL file specified by WSDLLocation (see page 1255).

Pascal

```
property WSDLItems: TWSDLItems;
```

C++

```
__property TWSDLItems WSDLItems;
```

Description

WSDLItems is a helper object whose methods return the definitions included in the WSDL file specified by WSDLLocation (see page 1255). Applications should not need to use this object. Rather, THTTPRIO (see page 1248) uses this property to obtain information about how to connect to the Web Service (see page 1253) application and how to encode request messages.

See Also

WSDLLocation (see page 1255)

17.1.1.5.7 THTTPRIO.WSDLLocation Property

Specifies the location of the WSDL document that describes the target Web Service (see page 1253).

Pascal

```
property WSDLLocation: string;
```

C++

```
__property UnicodeString WSDLLocation;
```

Description

Use WSDLLocation to specify a WSDL document that defines the target interface (port type). WSDLLocation can be either a fully qualified path name, or a URL (see page 1254) that indicates where the WSDL document can be found.

When using a WSDL document to indicate the target Web Service (see page 1253), you must also specify values for the Service (see page 1253) and Port (see page 1253) properties. Once WSDLLocation is set, you can use a drop-down list in the Object Inspector to select values for Port (see page 1253) and Service (see page 1253).

Note: If you do not need to look up connection information from the WSDL document at runtime, you can use the URL (see page 1254) property instead. Using the WSDLLocation property is mutually exclusive with the use of the URL (see page 1254) property.

See Also

Port (see page 1253)

Service (see page 1253)

URL (see page 1254)

18 SOAPHTTPDisp Namespace

This is namespace SOAPHTTPDisp.

Topics

Name	Description
Classes (see page 1258)	The following table lists classes in this documentation.
Interfaces (see page 1262)	The following table lists interfaces in this documentation.

Classes

	Name	Description
	THTTPSoapDispatchNode (see page 1258)	THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

Interfaces

	Name	Description
	IHTTPSoapDispatch (see page 1262)	IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

18.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapDispatchNode (see page 1258)	THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

18.1.1 THTTPSoapDispatchNode Class

THTTPSoapDispatchNode forwards SOAP messages to an invoker for interpretation.

Class Hierarchy

Pascal

```
THTTPSoapDispatchNode = class(TComponent);
```

C++

```
class THTTPSoapDispatchNode : public TComponent;
```

File

SOAPHTTPDisp

Description

Use THTTPSoapDispatchNode as a base class for objects that use an invoker component to interpret and execute SOAP message calls. THTTPSoapDispatchNode maintains a reference to the interface of a component that can interpret and execute SOAP method calls. These calls are requests to execute a method on an invokable interface that is registered in the application. You can register an invokable interface by calling the RegisterInterface method of the TInvokableClassRegistry object that the global InvRegistry function returns.

See Also

THTTPSoapPascalInvoker (see page 1267)

THTTPSoapCppInvoker (see page 1266)

IHTTPSoapDispatch (see page 1262)

InvRegistry

TInvokableClassRegistry

18.1.1.1 THTTPSoapDispatchNode Members

The following tables list the members exposed by THTTPSoapDispatchNode.

Protected Fields

	Name	Description
	FSoapDispatcher (see page 1259)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

Protected Methods

	Name	Description
	Notification (see page 1260)	Responds when objects are added or removed from the Web module.

Public Methods

	Name	Description
	DispatchSOAP (see page 1259)	Forwards the SOAP message on to the invoker for interpretation and execution.

Published Properties

	Name	Description
	Dispatcher (see page 1261)	Specifies the interface of the component that interprets SOAP messages.

18.1.1.2 THTTPSoapDispatchNode Fields

The fields of the THTTPSoapDispatchNode class are listed here.

Protected Fields

	Name	Description
	FSoapDispatcher (see page 1259)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

18.1.1.2.1 THTTPSoapDispatchNode.FSoapDispatcher Field

Pascal

```
FSoapDispatcher: IHTTPSoapDispatch;
```

C++

```
IHTTPSoapDispatch FSoapDispatcher;
```

Description

This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

18.1.1.3 THTTPSoapDispatchNode Methods

The methods of the THTTPSoapDispatchNode class are listed here.

Protected Methods

	Name	Description
	Notification (see page 1260)	Responds when objects are added or removed from the Web module.

Public Methods

	Name	Description
	DispatchSOAP (see page 1259)	Forwards the SOAP message on to the invoker for interpretation and execution.

18.1.1.3.1 THTTPSoapDispatchNode.DispatchSOAP Method

Forwards the SOAP message on to the invoker for interpretation and execution.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream); virtual;
```

C++

```
virtual __fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request,
TStream Response);
```

Description

DispatchSOAP calls the DispatchSOAP method of the interface specified by the Dispatcher (see page 1261) property. The invoker that implements that interface interprets the message, executes it, and returns the results as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the content for the outgoing response message. DispatchSoap fills it out with the SOAP encoding of any return values.

See Also

DispatchSOAP (see page 1263)

Dispatcher (see page 1261)

18.1.1.3.2 THTTPSoapDispatchNode.Notification Method

Responds when objects are added or removed from the Web module.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. When Operation is opRemove, Notification checks whether AComponent is the implementer of the Dispatcher (see page 1261) property. If so, it sets Dispatcher (see page 1261) to nil (Delphi) or NULL (C++).

See Also

Dispatcher (see page 1261)

18.1.1.4 THTTPSoapDispatchNode Properties

The properties of the THTTPSoapDispatchNode class are listed here.

Published Properties

	Name	Description
	Dispatcher (see page 1261)	Specifies the interface of the component that interprets SOAP messages.

18.1.1.4.1 THTTPSoapDispatchNode.Dispatcher Property

Specifies the interface of the component that interprets SOAP messages.

Pascal

```
property Dispatcher: IHTTPSoapDispatch;
```

C++

```
__property IHTTPSoapDispatch Dispatcher;
```

Description

Set Dispatcher to the interface of a component that interprets SOAP messages and passes them on to the target invokable interface. Dispatcher is the interface of an invoker component, such as THTTPSoapPascalInvoker (Delphi) or THTTPSoapCppInvoker (C++). Invoker components interpret incoming SOAP calls, identify the invokable interface to which they are directed, execute the call, and assemble a response message.

See Also

THTTPSoapPascalInvoker (see page 1267)

THTTPSoapCppInvoker (see page 1266)

18.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHTTPSoapDispatch (see page 1262)	IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

18.2.1 IHTTPSoapDispatch Interface

IHTTPSoapDispatch is the interface for objects that interpret SOAP request messages and execute the corresponding invokable interface.

Class Hierarchy

SOAPHTTPDisp.IHTTPSoapDispatch

Pascal

```
IHTTPSoapDispatch = interface;
```

C++

```
__interface IHTTPSoapDispatch;
```

File

SOAPHTTPDisp

Description

The THTTPSoapDispatcher component uses this interface to call the object that interprets and responds to the incoming SOAP messages that it receives. By using an interface, THTTPSoapDispatcher can work with any component that interprets and executes SOAP messages.

This interface is implemented by THTTPSoapPascalInvoker and THTTPSoapCppInvoker.

Note: C++ method declarations that use IHTTPSoapDispatch use the _di_IHTTPSoapDispatch type instead. This type is a DelphiInterface (see page 3256) wrapper around the IHTTPSoapDispatch interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IHTTPSoapDispatch >
_di_IHTTPSoapDispatch;
```

See Also

THTTPSoapPascalInvoker (see page 1267)

THTTPSoapDispatchNode (see page 1258)

18.2.1.1 IHTTPSoapDispatch Members

The following tables list the members exposed by IHTTPSoapDispatch.

Public Methods

	Name	Description
	DispatchSOAP (see page 1263)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

18.2.1.2 IHTTPSoapDispatch Methods

The methods of the IHTTPSoapDispatch class are listed here.

Public Methods

	Name	Description
	DispatchSOAP (see page 1263)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

18.2.1.2.1 IHTTPSoapDispatch.DispatchSOAP Method

Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType);
```

C++

```
__fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request, TStream
Response, TWebServiceBindingType BindingType);
```

Description

DispatchSOAP interprets a SOAP message, executes the interface call it represents, and encodes any return values as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the content for the outgoing response message. DispatchSoap fills it out with the SOAP encoding of any return values.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. DispatchSOAP changes BindingType to the format of the response message.

See Also

THTTPSoapDispatcher (see page 5946)

19 SOAPHTTPPasInv Namespace

This is namespace SOAPHTTPPasInv.

Topics

Name	Description
Classes (see page 1266)	The following table lists classes in this documentation.

Classes

	Name	Description
	THTTPSoapCppInvoker (see page 1266)	THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.
	THTTPSoapPascalInvoker (see page 1267)	THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

19.1 Classes

The following table lists classes in this documentation.

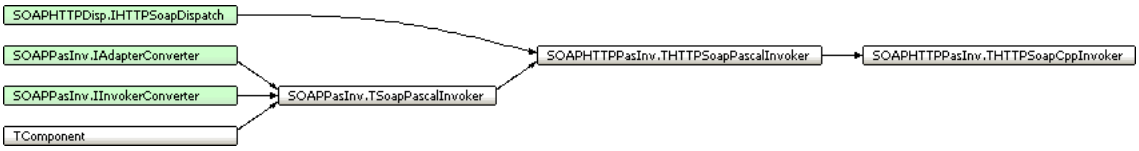
Classes

	Name	Description
	THTTPSoapCppInvoker (see page 1266)	THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.
	THTTPSoapPascalInvoker (see page 1267)	THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

19.1.1 THTTPSoapCppInvoker Class

THTTPSoapCppInvoker interprets a SOAP request message and executes the corresponding invokable interface.

Class Hierarchy



Pascal

```
THTTPSoapCppInvoker = class(THTTPSoapPascalInvoker);
```

C++

```
class THTTPSoapCppInvoker : public THTTPSoapPascalInvoker;
```

File

SOAPHTTPPasInv

Description

Use THTTPSoapCppInvoker in a C++ application that publishes a Web Service using SOAP. THTTPSoapCppInvoker receives an incoming SOAP request from a THTTPSoapDispatcher component, parses it and generates the appropriate call to a registered invokable interface, and encodes the results of that interface call.

THTTPSoapCppInvoker implements the IHTTPSoapDispatch (see page 1262) interface, which the dispatcher uses to pass it incoming SOAP requests and to receive the content of the response message. To enable the dispatcher to use this interface, assign it as the value of the THTTPSoapDispatcher component's Dispatcher property.

THTTPSoapCppInvoker uses the invocation registry (InvRegistry) to determine whether the interface call encoded in the incoming SOAP request matches a method on a registered invokable interface. Thus, before the invoker can handle a call, the target interface must be registered using the RegisterInterface method of the invocation registry. In addition, the implementation class for that interface must be registered with the invocation registry.

See Also

IInvokable (see page 3504)

InvRegistry

TInvokableClassRegistry

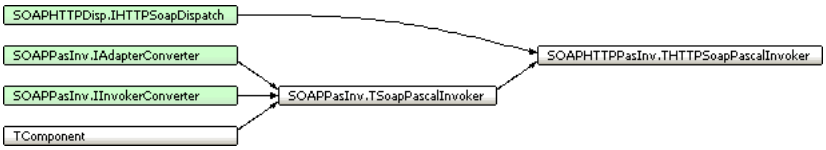
THTTPSoapDispatcher (see page 5946)

IHTTPSoapDispatch (see page 1262)

19.1.2 THTTPSoapPascalInvoker Class

THTTPSoapPascalInvoker interprets a SOAP request message and executes the corresponding invokable interface.

Class Hierarchy



Pascal

```
THTTPSoapPascalInvoker = class(TSoapPascalInvoker, IHTTPSoapDispatch);
```

C++

```
class THTTPSoapPascalInvoker : public TSoapPascalInvoker, public IHTTPSoapDispatch;
```

File

SOAPHTTPPasInv

Description

Use THTTPSoapPascalInvoker in an application that publishes a Web Service using SOAP. THTTPSoapPascalInvoker receives an incoming SOAP request from a THTTPSoapDispatcher component, parses it and generates the appropriate call to a registered invokable interface, and encodes the results of that interface call.

In Delphi, you use THTTPSoapPascalInvoker directly as the invoker in your Web Service application. In C++, you use THTTPSoapCppInvoker (see page 1266), a descendant class, instead.

THTTPSoapPascalInvoker implements the IHTTPSoapDispatch (see page 1262) interface, which the dispatcher uses to pass it incoming SOAP requests and to receive the content of the response message. To enable the dispatcher to use this interface, assign it as the value of the THTTPSoapDispatcher component's Dispatcher property.

THTTPSoapPascalInvoker uses the invocation registry (InvRegistry) to determine whether the interface call encoded in the incoming SOAP request matches a method on a registered invokable interface. Thus, before the invoker can handle a call, the target interface must be registered using the RegisterInterface method of the invocation registry. In addition, the implementation class for that interface must be registered with the invocation registry.

See Also

IInvokable (see page 3504)

InvRegistry

TInvokableClassRegistry

THTTPSoapDispatcher (see page 5946)

IHTTPSoapDispatch (see page 1262)

Understanding Invokable Interfaces

Using the SOAP Application Wizard

Writing Servers that Support Web Services

19.1.2.1 THTTPSoapPascalInvoker Members

The following tables list the members exposed by THTTPSoapPascalInvoker.

Protected Fields

	Name	Description
	FConverter (see page 1352)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 1352)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 1353)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 1353)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 1353)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

Protected Methods

	Name	Description
	GetDomConverter (see page 1354)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetInvoker (see page 1354)	This is GetInvoker, a member of class TSoapPascalInvoker.
	Notification (see page 1355)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Create (see page 1351)	Creates a new TSoapPascalInvoker (see page 1350) object.
	Destroy (see page 1352)	Frees the memory associated with the TSoapPascalInvoker (see page 1350) object.
	DispatchSOAP (see page 1269)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.
	GetDomConverter (see page 1360)	This is GetDomConverter, a member of class IAdapterConverter.
	GetInvoker (see page 1361)	This is GetInvoker, a member of class IInvokerConverter.
	Invoke (see page 1354)	Interprets a SOAP request message, calls the target interface, and encodes any return values.

Published Events

	Name	Description
	AfterDispatchEvent (see page 1356)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 1357)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 1357)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 1357)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 1358)	Occurs when TSoapPascalInvoker (see page 1350) reads each attachment included in an incoming request.
	OnSendAttachment (see page 1358)	Occurs when TSoapPascalInvoker (see page 1350) sends each attachment included in the response to the current request.

Published Properties

	Name	Description
	Converter (see page 1355)	Specifies the interface of the component that handles the marshaling of SOAP messages.

19.1.2.2 THTTPSoapPascalInvoker Methods

The methods of the THTTPSoapPascalInvoker class are listed here.

Public Methods

	Name	Description
	DispatchSOAP (see page 1269)	Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

19.1.2.2.1 THTTPSoapPascalInvoker.DispatchSOAP Method

Interprets an incoming SOAP message, calls the target interface, and encodes any return values.

Pascal

```
procedure DispatchSOAP(const Path: WideString; const SoapAction: WideString; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType); virtual;
```

C++

```
virtual __fastcall DispatchSOAP(const BSTR Path, const BSTR SoapAction, const TStream Request,
TStream Response, TWebServiceBindingType BindingType);
```

Description

The SOAP dispatcher component (THTTPSoapDispatcher) calls DispatchSOAP when it receives an incoming SOAP message. DispatchSOAP interprets the message, looks up the target interface in the invocation registry, and calls the Invoke method to execute the call and encode any return values as a SOAP string.

Path is the path portion of the URL to which the incoming request message was addressed.

SoapAction is the content of the SOAP Action header of the incoming HTTP request message.

Request is a stream from which DispatchSOAP can read the content of the incoming request message. It contains the SOAP encoding of an interface call.

Response is a stream to which DispatchSOAP writes the SOAP encoding of any return values. THTTPSoapPascalInvoker (see page 1267) uses this as the content for the outgoing response message.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. DispatchSOAP changes BindingType to the format of the response message.

See Also

THTTPSoapDispatcher (see page 5946)

IHTTPSoapDispatch (see page 1262)

InvRegistry

Invoke (see page 1354)

20 SOAPHTTPTrans Namespace

This is namespace SOAPHTTPTrans.

Topics

Name	Description
Classes (see page 1272)	The following table lists classes in this documentation.
Interfaces (see page 1296)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 1298)	The following table lists structs, records, enums in this documentation.
Types (see page 1299)	The following table lists types in this documentation.

Classes

	Name	Description
	ESOAPHTTPException (see page 1272)	ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.
	THTTPReqResp (see page 1276)	THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

Interfaces

	Name	Description
	IHTTPReqResp (see page 1296)	This is class SOAPHTTPTrans.IHTTPReqResp.

Structs, Records, Enums

	Name	Description
	SOAPInvokeOptions (see page 1298)	SOAPInvokeOptions and TSOAPInvokeOptions (see page 1300) represent flags that customize how to handle a SOAP message.

Types

Name	Description
TBeforePostEvent (see page 1299)	TBeforePostEvent is the type of the OnBeforePost event of THTTPReqResp (see page 1276).
TPostingDataEvent (see page 1299)	TPostingDataEvent is the type of the OnPostingData event of THTTPReqResp (see page 1276).
TReceivingDataEvent (see page 1300)	TReceivingDataEvent is the type of the OnReceivingData event of THTTPReqResp (see page 1276).
TSOAPInvokeOptions (see page 1300)	This is type SOAPHTTPTrans.TSOAPInvokeOptions.
TWinInetErrorEvent (see page 1301)	This is type SOAPHTTPTrans.TWinInetErrorEvent.

20.1 Classes

The following table lists classes in this documentation.

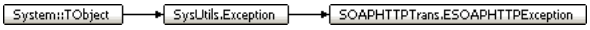
Classes

	Name	Description
	ESOAPHTTPException (see page 1272)	ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.
	THTTPReqResp (see page 1276)	THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

20.1.1 ESOAPHTTPException Class

ESOAPHTTPException is the class for exceptions that occur when sending or receiving an HTTP message that contains a SOAP packet.

Class Hierarchy



Pascal

```
ESOAPHTTPException = class(Exception);
```

C++

```
class ESOAPHTTPException : public Exception;
```

File

SOAPHTTPTrans

Description

ESOAPHTTPException is raised when there is a problem with the HTTP messages that communicate a SOAP request or response in a Web Services application. Such problems include

The application can't send or receive the HTTP message.

The URL that describes the server is missing or badly formed.

The Content-type header of an HTTP message is not consistent with SOAP.

See Also

THTTPReqResp (see page 1276)

20.1.1.1 ESOAPHTTPException Members

The following tables list the members exposed by ESOAPHTTPException.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.

	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1274)	Instantiates an instance of an ESOAPHTTPException (see page 1272) with a simple message string.
	CreateFmt (see page 1274)	Instantiates an instance of an ESOAPHTTPException (see page 1272) with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 3960)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 3380)	Returns a string containing the class name.
V	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.
	StatusCode (see page 1275)	Indicates the HTTP status code of the message that gave rise to this exception.

20.1.1.2 ESOAPHTTPException.Create Constructor

Instantiates an instance of an ESOAPHTTPException (see page 1272) with a simple message string.

Pascal

```
constructor Create(const Msg: string; SCode: Integer = 0; Dummy: Integer = 0);
```

C++

```
__fastcall ESOAPHTTPException(const UnicodeString Msg, int SCode = 0, int Dummy = 0);
```

Description

Call Create to construct an ESOAPHTTPException (see page 1272) object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

SCode is the HTTP status code, if any, that triggered this exception. It becomes the value of the StatusCode (see page 1275) property.

See Also

CreateFmt (see page 1274)

StatusCode (see page 1275)

Message (see page 3965)

20.1.1.3 ESOAPHTTPException.CreateFmt Constructor

Instantiates an instance of an ESOAPHTTPException (see page 1272) with a formatted message string.

Pascal

```
constructor CreateFmt(const Msg: string; const Args: array of const; SCode: Integer = 0;
Dummy: Integer = 0);
```

C++

```
__fastcall ESOAPHTTPException(const UnicodeString Msg, const array of const Args, int SCode =
```

```
0, int Dummy = 0);
```

Description

Call CreateFmt to construct an ESOAPHTTPException (see page 1272) with a formatted message string.

Msg is a string constant containing format specifiers to be replaced with values from Args.

Args is an array of constants containing values to format according to format specifiers in Msg. CreateFmt calls the Format (see page 4188) function to transform Msg with the values in Args.

SCode is the HTTP status code, if any, that triggered this exception. It becomes the value of the StatusCode (see page 1275) property.

See Also

Create (see page 1274)

StatusCode (see page 1275)

Message (see page 3965)

20.1.1.4 ESOAPHTTPException Properties

The properties of the ESOAPHTTPException class are listed here.

Public Properties

	Name	Description
	StatusCode (see page 1275)	Indicates the HTTP status code of the message that gave rise to this exception.

20.1.1.4.1 ESOAPHTTPException.StatusCode Property

Indicates the HTTP status code of the message that gave rise to this exception.

Pascal

```
property StatusCode: Integer;
```

C++

```
__property int StatusCode;
```

Description

StatusCode indicates the HTTP status code of the message that caused the application to raise an ESOAPHTTPException (see page 1272) exception. If the exception does not represent an HTTP response with a status code that indicates an error, StatusCode is 0.

The value of StatusCode is set by the ESOAPHTTPException (see page 1272) constructor.

Note: In Delphi code, this is set by the CreateFmt (see page 1274) or Create (see page 1274) constructor.

Note: In C++ code, this is set using one of the two constructor syntaxes that take a status code parameter.

See Also

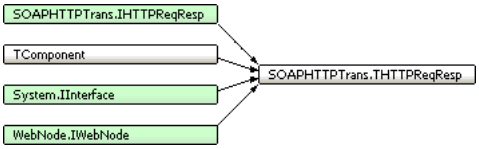
Create (see page 1274)

CreateFmt (see page 1274)

20.1.2 THTTPReqResp Class

THTTPReqResp executes a method call on an invokable interface by sending a SOAP message to the server.

Class Hierarchy



Pascal

```
THTTPReqResp = class(TComponent, IInterface, IWebNode, IHTTPReqResp);
```

C++

```
class THTTPReqResp : public TComponent, public IInterface, public IWebNode, public
IHTTPReqResp;
```

File

SOAPHTTPTrans

Description

THTTPReqResp handles HTTP-based communication with a Web Service provider on behalf of a THTTPRio instance. THTTPRio uses this component to establish a connection to the Web Service provider and perform two tasks:

It can issue a "Get (see page 1283)" request to obtain information from a WSDL document.

It can issue a "Post" request to pass a method call to the server for execution and retrieve the results.

The Web Service that is the target of these HTTP messages can be specified directly, using the SoapAction (see page 1291) and URL (see page 1292) properties, or it can be taken from the SoapAction (see page 1291) in a WSDL document, as specified by the WSDLView (see page 1293) property. Additional properties provide details that are included in the HTTP request message header.

Note: On Windows, THTTPReqResp uses WinInet to establish a connection to the server. Wininet.dll must be installed on the client system. wininet.dll is found in the Windows system directory if you have IE3 or higher installed. WinInet has the advantage that it provides support for secure connections (https). To use WinInet, compile your project without the USE_INDY symbol defined.

Note: On Linux, THTTPReqResp uses an internal instance of the Indy HTTP component (TIdHTTP) to communicate with the server. The Indy component has the advantage that it can be used from an application that acts as both client and server, whereas WinInet only works for pure client code. To use an internal TIdHTTP, compile your project with the USE_INDY symbol defined.

See Also

IWebNode (see page 6616)

THTTPRIO (see page 1248)

TRIO

20.1.2.1 THTTPReqResp Members

The following tables list the members exposed by THTTPReqResp.

Protected Methods

	Name	Description
	_AddRef (see page 1279)	Called when an application uses a component interface.
	_Release (see page 1280)	Called when an application releases a component interface.
	GetMimeBoundary (see page 1284)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 1287)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	AfterConstruction (see page 1280)	Responds after the last constructor executes.
	BeforeExecute (see page 1280)	Performs transport-specific setup before executing an encoded method call.
	Check (see page 1281)	Raises an exception when an error is detected.
	CheckContentType (see page 1282)	Raises an exception if the response to a SOAP method call has a common invalid content-type header.
	Connect (see page 1282)	Establishes or terminates a connection to the Web Service application.
	Create (see page 1278)	Creates a new THTTPReqResp (see page 1276) object.
	Destroy (see page 1278)	Frees the memory associated with the THTTPReqResp (see page 1276) object.
	Execute (see page 1282)	This is the overview for the Execute method overload.
	Execute (see page 6618)	This is the overview for the Execute method overload.
	Get (see page 1283)	Sends a "Get" request to the server and retrieves the results.
	GetHTTPReqResp (see page 1284)	Returns a reference to this component.
	HandleWinInetError (see page 1285)	This is HandleWinInetError, a member of class THTTPReqResp.
	NewInstance (see page 1285)	Allocates memory for each instance of THTTPReqResp (see page 1276).
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Receive (see page 1285)	Receives the contents of an HTTP response to the message sent by the Send (see page 1286) message.
	Send (see page 1286)	Sends the encoded method call as an HTTP-based SOAP message.
	SendGet (see page 1286)	SendGets sends a "GET" request to the server.

Public Properties

	Name	Description
	ConnectTimeout (see page 1288)	Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.
	MaxSinglePostSize (see page 1289)	This is MaxSinglePostSize, a member of class THTTPReqResp.
	MimeBoundary (see page 6619)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
	ReceiveTimeout (see page 1290)	Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.
	SendTimeout (see page 1291)	Indicates the number of milliseconds that can elapse before timing out the sending of a request.
	SoapAction (see page 1291)	Specifies the SoapAction header to include in the HTTP message that executes the method call.
	URL (see page 1292)	Specifies the URL that identifies the target interface on the target Web Service application.

Published Events

	Name	Description
	OnBeforePost (see page 1294)	Occurs before THTTPReqResp (see page 1276) sends its message.
	OnPostingData (see page 1294)	Occurs before THTTPReqResp (see page 1276) sends each chunk of data when the content is large.

	OnReceivingData (see page 1295)	Occurs after THTTPReqResp (see page 1276) receives each chunk of data when the content is large.
	OnWinInetError (see page 1295)	This is OnWinInetError, a member of class THTTPReqResp.

Published Properties

	Name	Description
	Agent (see page 1288)	Provides the value of the User-Agent header for HTTP messages to the Web Service provider.
	InvokeOptions (see page 1288)	Customizes the way THTTPReqResp (see page 1276) generates or handles headers in the messages it sends to the server.
	Password (see page 1289)	Specifies the password that allows the client application to log on when the Web Service provider requires authentication.
	Proxy (see page 1289)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1290)	Lists the host names of locally registered servers that do not require a proxy.
	UDDIBindingKey (see page 1291)	Indicates the binding key of a UDDI entry to use for fail-over support.
	UDDIOperator (see page 1292)	Indicates the URL (see page 1292) of the UDDI registry to use for fail-over support.
	UserName (see page 1293)	Specifies the user name that allows the client application to log on to the server.
	UseUTF8InHeader (see page 1293)	Specifies whether the Content-Type header should specify UTF-8 as a character set.
	WSDLView (see page 1293)	Indicates the target interface method using a WSDL document.

20.1.2.2 THTTPReqResp.Create Constructor

Creates a new THTTPReqResp (see page 1276) object.

Pascal

```
constructor Create(Owner: TComponent); override;
```

C++

```
virtual __fastcall THTTPReqResp(TComponent * Owner);
```

Description

Most applications do not create instances of THTTPReqResp (see page 1276). It is created and used by the THTTPRio class.

The Owner parameter specifies the component that is responsible for freeing the THTTPReqResp (see page 1276) instance.

See Also

Owner

THTTPRIO (see page 1248)

20.1.2.3 THTTPReqResp.Destroy Destructor

Frees the memory associated with the THTTPReqResp (see page 1276) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPReqResp();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

The THTTPReqResp (see page 1276) destructor ensures that the connection to the server is closed before the component is

freed.

See Also

System::TObject::Free (see page 3372)

20.1.2.4 THTTPReqResp Methods

The methods of the THTTPReqResp class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 1279)	Called when an application uses a component interface.
	_Release (see page 1280)	Called when an application releases a component interface.
	GetMimeBoundary (see page 1284)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 1287)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	AfterConstruction (see page 1280)	Responds after the last constructor executes.
	BeforeExecute (see page 1280)	Performs transport-specific setup before executing an encoded method call.
	Check (see page 1281)	Raises an exception when an error is detected.
	CheckContentType (see page 1282)	Raises an exception if the response to a SOAP method call has a common invalid content-type header.
	Connect (see page 1282)	Establishes or terminates a connection to the Web Service application.
	Execute (see page 1282)	This is the overview for the Execute method overload.
	Get (see page 1283)	Sends a "Get" request to the server and retrieves the results.
	GetHTTPReqResp (see page 1284)	Returns a reference to this component.
	HandleWinInetError (see page 1285)	This is HandleWinInetError, a member of class THTTPReqResp.
	NewInstance (see page 1285)	Allocates memory for each instance of THTTPReqResp (see page 1276).
	Receive (see page 1285)	Receives the contents of an HTTP response to the message sent by the Send (see page 1286) message.
	Send (see page 1286)	Sends the encoded method call as an HTTP-based SOAP message.
	SendGet (see page 1286)	SendGets sends a "GET" request to the server.

20.1.2.4.1 THTTPReqResp._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

QueryInterface

_Release

_AddRef (see page 3503)

20.1.2.4.2 THTTPReqResp._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

QueryInterface

_AddRef

20.1.2.4.3 THTTPReqResp.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of THTTPReqResp (see page 1276) is allocated, its reference count is incremented to prevent its destruction while processing the constructor. AfterConstruction decrements that reference count to re-establish the correct count after all constructors have executed.

See Also

System::TObject::BeforeDestruction (see page 3358)

NewInstance (see page 1285)

20.1.2.4.4 THTTPReqResp.BeforeExecute Method

Performs transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMD: TIntfMetaData; const MethMD: TIntfMethEntry;
MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMD, const TIntfMethEntry MethMD, int
MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute implements the IWebNode (see page 6616) method of the same name. The remote interface object (TRio) calls this method after its OnBeforeExecute event, but before calling the Execute (see page 1283) method to send a request to the server. BeforeExecute looks up the information needed to call the server from the invocation registry, based on either the URL (see page 1292) that was supplied to the remote interfaced object or on the WSDLLocation, Service, and Port that were supplied to the remote interfaced object. If the method call requires an attachment, BeforeExecute looks up the MIME boundary for the attachment and adds headers to indicate the MIME boundary and content type of the attachment.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method. If AttachHandler is nil (Delphi) or NULL (C++), the method call does not require passing any attachments.

See Also

Execute (see page 1283)

OnBeforeExecute

BeforeExecute (see page 6617)

URL (see page 1254)

WSDLLocation (see page 1255)

Service (see page 1253)

Port (see page 1253)

20.1.2.4.5 THTTPReqResp.Check Method

Raises an exception when an error is detected.

Pascal

```
procedure Check(Error: Boolean; ShowSOAPAction: Boolean = False);
```

C++

```
__fastcall Check(Boolean Error, Boolean ShowSOAPAction = False);
```

Description

Note: On Windows (when using WinInet), THTTPReqResp (see page 1276) calls Check internally to raise an exception when it encounters an error.

Note: Check is not available on Linux.

Error (see page 3560) indicates whether an error condition was detected.

ShowSOAPAction indicates whether the error message should include information about the SOAP Action header.

See Also

Exception (see page 3949)

20.1.2.4.6 THTTPReqResp.CheckContentType Method

Raises an exception if the response to a SOAP method call has a common invalid content-type header.

Pascal

```
procedure CheckContentType;
```

C++

```
__fastcall CheckContentType();
```

Description

CheckContentType checks the Content-type header of the response that THTTPReqResp (see page 1276) receives after sending a request to the Web Service application. If it recognizes the content-type as invalid, CheckContentType raises an ESOAPHTTPException (see page 1272).

CheckContentType does not rule out all invalid content types, only the most common.

See Also

ESOAPHTTPException (see page 1272)

20.1.2.4.7 THTTPReqResp.Connect Method

Establishes or terminates a connection to the Web Service application.

Pascal

```
procedure Connect(Value: Boolean);
```

C++

```
__fastcall Connect(Boolean Value);
```

Description

Note: On Windows, call Connect to open or close a connection to the Web Service application identified by the URL (see page 1292) property. Connect opens a connection using WinInet.

Note: Connect is not available on Linux.

Value indicates whether to open or close the connection. When Value is true, Connect establishes a connection. When Value is false, Connect terminates the connection.

See Also

URL (see page 1292)

20.1.2.4.8 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	THTTPReqResp.Execute (String, TStream) (see page 1283)	Executes a SOAP method call and returns the encoded results.
	THTTPReqResp.Execute (TStream) (see page 1283)	Executes a SOAP method call and returns the encoded results.
	THTTPReqResp.Execute (TStream, TStream) (see page 1283)	Executes a SOAP method call and returns the encoded results.

20.1.2.4.8.1 THTTPReqResp.Execute Method (String, TStream)

Executes a SOAP method call and returns the encoded results.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); virtual; overload;
procedure Execute(const Request: TStream; Response: TStream); virtual; overload;
function Execute(const Request: TStream): TStream; virtual; overload;
```

C++

```
virtual __fastcall Execute(const UnicodeString DataMsg, TStream Resp);
virtual __fastcall Execute(const TStream Request, TStream Response);
virtual __fastcall TStream Execute(const TStream Request);
```

Description

Execute implements the IWebNode (see page 6616) method of the same name. THTTPRio uses this method to execute a method on an invokable interface that has been encoded as a SOAP method call.

Execute

1. Connects to the Web Service application identified by the URL (see page 1292) property.
2. Creates an HTTP message with DataMsg or Request as its content and SoapAction (see page 1291) as its SoapAction (see page 1291) header.
3. Sends it to the Web Service application and retrieves the results.
4. Writes the results to the stream specified by Resp or returns a stream from which you can read the results. If using the third syntax, which returns a stream, the caller is responsible for freeing the stream object.

See Also

Execute (see page 6618)

THTTPRIO (see page 1248)

URL (see page 1292)

Send (see page 1286)

Receive (see page 1285)

SoapAction (see page 1291)

20.1.2.4.9 THTTPReqResp.Get Method

Sends a "Get" request to the server and retrieves the results.

Pascal

```
procedure Get(Resp: TStream); virtual;
```

C++

```
virtual __fastcall Get(TStream Resp);
```

Description

Get sends an HTTP request message to the server with a method type of GET. THTTPRio uses this method to fetch the WSDL document that underlies the WSDLView (see page 1293) property.

Resp is a stream that receives the contents of the response to the "Get" request message.

See Also

THTTPRIO (see page 1248)

URL (see page 1292)

Send (see page 1286)

Receive (see page 1285)

Execute (see page 1283)

WSDLView (see page 1293)

20.1.2.4.10 THTTPReqResp.GetHTTPReqResp Method

Returns a reference to this component.

Pascal

```
function GetHTTPReqResp: THTTPReqResp;
```

C++

```
__fastcall THTTPReqResp GetHTTPReqResp();
```

Description

GetHTTPReqResp returns this THTTPReqResp (see page 1276) instance (Delphi) or a pointer to this instance (C++).

20.1.2.4.11 THTTPReqResp.GetMimeBoundary Method

Returns the value of the MimeBoundary property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary property.

See Also

MimeBoundary (see page 6619)

20.1.2.4.12 THTTPReqResp.HandleWinInetError Method

Pascal

```
function HandleWinInetError(LastError: DWord; Request: HINTERNET): DWord;
```

C++

```
__fastcall DWord HandleWinInetError(DWord LastError, HINTERNET Request);
```

Description

This is HandleWinInetError, a member of class THTTPReqResp.

20.1.2.4.13 THTTPReqResp.NewInstance Method

Allocates memory for each instance of THTTPReqResp (see page 1276).

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call NewInstance automatically. NewInstance calls InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call NewInstance directly.

THTTPReqResp (see page 1276) overrides NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a reference count of zero. The AfterConstruction (see page 1280) method decrements the reference count after all constructors have executed.

See Also

System::TObject::NewInstance (see page 3379)

System::TObject::InstanceSize (see page 3377)

AfterConstruction (see page 1280)

20.1.2.4.14 THTTPReqResp.Receive Method

Receives the contents of an HTTP response to the message sent by the Send (see page 1286) message.

Pascal

```
procedure Receive(Context: Integer; Resp: TStream; IsGet: Boolean = False); virtual;
```

C++

```
virtual __fastcall Receive(int Context, TStream Resp, Boolean IsGet = False);
```

Description

Note: On Windows (when using WinInet), Receive fetches the HTTP response to the message sent by the SendMessage method, and writes its contents to the specified stream.

Note: Receive is not available on Linux.

Context is the context ID that identifies the original request message that the Send (see page 1286) message sent. It is the

return value from a previous call to the Send (see page 1286) method.

Resp is a stream to which Receive can write the contents of the response message.

IsGet indicates whether the response stream contains the results from a "Get (see page 1283)" request message.

See Also

Send (see page 1286)

Execute (see page 1283)

20.1.2.4.15 THTTPReqResp.Send Method

Sends the encoded method call as an HTTP-based SOAP message.

Pascal

```
function Send(const ASrc: TStream): Integer; virtual;
```

C++

```
virtual __fastcall int Send(const TStream ASrc);
```

Description

Note: On Windows (when using WinInet), Send sends an HTTP request message with the SoapAction (see page 1291) header specified by SoapAction (see page 1291) to the URL (see page 1292) specified by URL (see page 1292).

Note: Send is not available on Linux.

ASrc is a stream from which to read the contents of a SOAP message for the method call.

Send returns a context ID that identifies this request message. This context ID is passed to the Receive (see page 1285) method to retrieve the corresponding response message from the server.

See Also

Receive (see page 1285)

Execute (see page 1283)

SoapAction (see page 1291)

URL (see page 1292)

20.1.2.4.16 THTTPReqResp.SendGet Method

SendGets sends a "GET" request to the server.

Pascal

```
function SendGet: Integer; virtual;
```

C++

```
virtual __fastcall int SendGet();
```

Description

Note: On Windows (when using WinInet), the Get (see page 1283) method calls SendGet to send the GET request to the URL (see page 1292) specified by URL (see page 1292).

Note: SendGet is not available on Linux.

SendGet returns a context ID that identifies the GET request message. The Get (see page 1283) method uses this context ID to retrieve the corresponding response message from the server.

See Also

Get (see page 1283)

URL (see page 1292)

20.1.2.4.17 THTTPReqResp.SetMimeBoundary Method

Sets the value of the MimeBoundary property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary property.

Value is the new value for MimeBoundary.

See Also

MimeBoundary (see page 6619)

20.1.2.5 THTTPReqResp Properties

The properties of the THTTPReqResp class are listed here.

Public Properties

	Name	Description
	ConnectTimeout (see page 1288)	Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.
	MaxSinglePostSize (see page 1289)	This is MaxSinglePostSize, a member of class THTTPReqResp.
	ReceiveTimeout (see page 1290)	Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.
	SendTimeout (see page 1291)	Indicates the number of milliseconds that can elapse before timing out the sending of a request.
	SoapAction (see page 1291)	Specifies the SoapAction header to include in the HTTP message that executes the method call.
	URL (see page 1292)	Specifies the URL that identifies the target interface on the target Web Service application.

Published Properties

	Name	Description
	Agent (see page 1288)	Provides the value of the User-Agent header for HTTP messages to the Web Service provider.
	InvokeOptions (see page 1288)	Customizes the way THTTPReqResp (see page 1276) generates or handles headers in the messages it sends to the server.
	Password (see page 1289)	Specifies the password that allows the client application to log on when the Web Service provider requires authentication.
	Proxy (see page 1289)	Lists any proxy servers that are required to make the connection.
	ProxyByPass (see page 1290)	Lists the host names of locally registered servers that do not require a proxy.

	UDDIBindingKey (see page 1291)	Indicates the binding key of a UDDI entry to use for fail-over support.
	UDDIOperator (see page 1292)	Indicates the URL (see page 1292) of the UDDI registry to use for fail-over support.
	UserName (see page 1293)	Specifies the user name that allows the client application to log on to the server.
	UseUTF8InHeader (see page 1293)	Specifies whether the Content-Type header should specify UTF-8 as a character set.
	WSDLView (see page 1293)	Indicates the target interface method using a WSDL document.

20.1.2.5.1 THTTPReqResp.Agent Property

Provides the value of the User-Agent header for HTTP messages to the Web Service provider.

Pascal

```
property Agent: string;
```

C++

```
__property UnicodeString Agent;
```

Description

Use Agent to supply the SOAP request message with the name of your application. This name is included as a User-Agent header. The User-Agent header is an optional string that describes the application generating the request.

See Also

Title

20.1.2.5.2 THTTPReqResp.ConnectTimeout Property

Indicates the number of milliseconds that can elapse before aborting an attempt to connect to the server.

Pascal

```
property ConnectTimeout: Integer;
```

C++

```
__property int ConnectTimeout;
```

Description

Use ConnectTimeout to indicate the number of milliseconds that can elapse before THTTPReqResp (see page 1276) abandons the attempt to form a connection.

See Also

SendTimeout (see page 1291)

ReceiveTimeout (see page 1290)

20.1.2.5.3 THTTPReqResp.InvokeOptions Property

Customizes the way THTTPReqResp (see page 1276) generates or handles headers in the messages it sends to the server.

Pascal

```
property InvokeOptions: TSOAPInvokeOptions;
```

C++

```
__property TSOAPInvokeOptions InvokeOptions;
```

Description

Use InvokeOptions to customize the way THTTPReqResp (see page 1276) generates SOAP messages to the invokable interface on the server. InvokeOptions is a set, allowing you to specify that THTTPReqResp (see page 1276) include none of the options.

See Also

SoapAction (see page 1291)

20.1.2.5.4 THTTPReqResp.MaxSinglePostSize Property

Pascal

```
property MaxSinglePostSize: Integer;
```

C++

```
__property int MaxSinglePostSize;
```

Description

This is MaxSinglePostSize, a member of class THTTPReqResp.

20.1.2.5.5 THTTPReqResp.Password Property

Specifies the password that allows the client application to log on when the Web Service provider requires authentication.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

When THTTPReqResp (see page 1276) establishes a connection, it supplies the values of the UserName (see page 1293) and Password properties so that it can log on to the host or proxy. The value of Password can be an empty string if neither the host nor proxy requires authentication.

See Also

UserName (see page 1293)

Connect (see page 1282)

20.1.2.5.6 THTTPReqResp.Proxy Property

Lists any proxy servers that are required to make the connection.

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

Use Proxy if the client application can't resolve the value of host portion of URL (see page 1292) locally. If the server is registered locally, Proxy can be a null string. Otherwise, Proxy lists the host names of any proxy servers that can forward request messages on to the intended server machine.

See Also

ProxyByPass (see page 1290)

URL (see page 1292)

20.1.2.5.7 THTTPReqResp.ProxyByPass Property

Lists the host names of locally registered servers that do not require a proxy.

Pascal

```
property ProxyByPass: string;
```

C++

```
__property UnicodeString ProxyByPass;
```

Description

Use ProxyByPass to indicate any server systems to which the client application can make a direct connection. When the URL (see page 1292) property addresses a server in this list, THTTPReqResp (see page 1276) makes a direct connection even if the Proxy (see page 1289) property is specified.

Note: ProxyByPass is only used on Windows, when the connection is formed using WinInet. On Linux, where the connection is formed using Indy, this property has no effect.

Note: When using WinInet, if ProxyByPass is an empty string, THTTPReqResp (see page 1276) checks the registry for any locally known servers and bypasses the proxy server if URL (see page 1292) refers to a registered local server.

See Also

Proxy (see page 1289)

URL (see page 1292)

20.1.2.5.8 THTTPReqResp.ReceiveTimeout Property

Indicates the number of milliseconds that can elapse before canceling a request when no response is received from the server.

Pascal

```
property ReceiveTimeout: Integer;
```

C++

```
__property int ReceiveTimeout;
```

Description

Use ReceiveTimeout to indicate the number of milliseconds that can elapse after sending a request before THTTPReqResp (see page 1276) cancels a request on the assumption that there will be no response forthcoming.

See Also

SendTimeout (see page 1291)

ConnectTimeout (see page 1288)

20.1.2.5.9 THTTPReqResp.SendTimeout Property

Indicates the number of milliseconds that can elapse before timing out the sending of a request.

Pascal

```
property SendTimeout: Integer;
```

C++

```
__property int SendTimeout;
```

Description

Use SendTimeout to indicate the number of milliseconds that can elapse when sending a request before THTTPReqResp (see page 1276) cancels the request.

See Also

ReceiveTimeout (see page 1290)

ConnectTimeout (see page 1288)

20.1.2.5.10 THTTPReqResp.SoapAction Property

Specifies the SoapAction header to include in the HTTP message that executes the method call.

Pascal

```
property SoapAction: string;
```

C++

```
__property UnicodeString SoapAction;
```

Description

SoapAction specifies the SoapAction header that appears in the HTTP message that is sent by the Send (see page 1286) method.The value of this header is deduced from information in a WSDL document, as indicated by the WSDLView (see page 1293) property.

See Also

URL (see page 1292)

WSDLView (see page 1293)

Send (see page 1286)

RegisterInterface

20.1.2.5.11 THTTPReqResp.UDDIBindingKey Property

Indicates the binding key of a UDDI entry to use for fail-over support.

Pascal

```
property UDDIBindingKey: WideString;
```

C++

```
__property BSTR UDDIBindingKey;
```

Description

UDDIBindingKey identifies the binding key for the Web Service TModel the defines the target interface. This key is used to look

up the Web Service definition in the UDDI registry that the UDDIOperator (see page 1292) property identifies. UDDIBindingKey and UDDIOperator (see page 1292) are used to implement fail-over support for Web Services that are imported using the UDDI browser.

See Also

UDDIOperator (see page 1292)

20.1.2.5.12 THTTPReqResp.UDDIOperator Property

Indicates the URL (see page 1292) of the UDDI registry to use for fail-over support.

Pascal

```
property UDDIOperator: String;
```

C++

```
__property UnicodeString UDDIOperator;
```

Description

UDDIOperator identifies the UDDI registry that was used to import the Web Service definition for the target interface. THTTPReqResp (see page 1276) uses this registry to look up the Web Service definition that the UDDIBindingKey (see page 1291) property identifies. UDDIOperator and UDDIBindingKey (see page 1291) are used to implement fail-over support for Web Services that are imported using the UDDI browser.

See Also

UDDIBindingKey (see page 1291)

20.1.2.5.13 THTTPReqResp.URL Property

Specifies the URL that identifies the target interface on the target Web Service application.

Pascal

```
property URL: string;
```

C++

```
__property UnicodeString URL;
```

Description

URL specifies the URL to which to send the message containing the SOAP request.

When communicating with a Web Service based on an imported WSDL document, the value of URL is deduced from information in a WSDL document, as indicated by the WSDLView (see page 1293) property.

Note: In Delphi, you can construct this URL as follows: The path portion must match the target path name of the THTTPSoapDispatcher component on the Web Service application and the protocol portion must specify http or https.

See Also

WSDLView (see page 1293)

WebDispatch (see page 5950)

SoapAction (see page 1291)

Send (see page 1286)

20.1.2.5.14 THTTPReqResp.UserName Property

Specifies the user name that allows the client application to log on to the server.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

When THTTPReqResp (see page 1276) establishes a connection, it supplies the values of the UserName and Password (see page 1289) properties so that it can log on to the host or proxy. The value of Password (see page 1289) can be an empty string if neither the host nor proxy requires authentication.

See Also

Password (see page 1289)

Connect (see page 1282)

20.1.2.5.15 THTTPReqResp.UseUTF8InHeader Property

Specifies whether the Content-Type header should specify UTF-8 as a character set.

Pascal

```
property UseUTF8InHeader: Boolean;
```

C++

```
__property Boolean UseUTF8InHeader;
```

Description

Set UseUTF8InHeader to indicate how THTTPReqResp (see page 1276) should generate the Content-Type header of the HTTP request message it sends to the server. When UseUTF8InHeader is true, the Content-Type header indicates that the content of the request is encoded in XML using the UTF-8 character set. Otherwise, the Content-Type header simply indicates that the content is encoded in XML.

See Also

SoapAction (see page 1291)

20.1.2.5.16 THTTPReqResp.WSDLView Property

Indicates the target interface method using a WSDL document.

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

WSDLView is a helper object that represents the WSDL file used to define the interfaces and protocols in the target Web Service application, as well as properties that indicate which is the target method call. It is used to generate the URL (see page 1292) and SoapAction (see page 1291) properties when importing definitions from a WSDL document.

See Also

URL (see page 1292)

SoapAction (see page 1291)

20.1.2.6 THTTPReqResp Events

The events of the THTTPReqResp class are listed here.

Published Events

	Name	Description
	OnBeforePost (see page 1294)	Occurs before THTTPReqResp (see page 1276) sends its message.
	OnPostingData (see page 1294)	Occurs before THTTPReqResp (see page 1276) sends each chunk of data when the content is large.
	OnReceivingData (see page 1295)	Occurs after THTTPReqResp (see page 1276) receives each chunk of data when the content is large.
	OnWinInetError (see page 1295)	This is OnWinInetError, a member of class THTTPReqResp.

20.1.2.6.1 THTTPReqResp.OnBeforePost Event

Occurs before THTTPReqResp (see page 1276) sends its message.

Pascal

```
property OnBeforePost: TBeforePostEvent;
```

C++

```
__property TBeforePostEvent OnBeforePost;
```

Description

Write (see page 3707) an OnBeforePost event handler to make final changes to the HTTP request before it is sent. The Send (see page 1286) method generates OnBeforePost after setting up the content and headers of a request message but before sending it.

See Also

OnPostingData (see page 1294)

Send (see page 1286)

20.1.2.6.2 THTTPReqResp.OnPostingData Event

Occurs before THTTPReqResp (see page 1276) sends each chunk of data when the content is large.

Pascal

```
property OnPostingData: TPostingDataEvent;
```

C++

```
__property TPostingDataEvent OnPostingData;
```

Description

Write (see page 3707) an OnPostingData event handler to provide the end-user with feedback of the progress when sending a request that contains a large amount of data. When the content of a request is very large (for example, when it contains large attachments), the Send (see page 1286) method breaks the message into blocks, which it sends separately. After sending each of these blocks, it generates an OnPostingData event. You can use the event handler to update a progress bar or similar

control to indicate how much of the message has been sent.

See Also

OnBeforePost (see page 1294)

Send (see page 1286)

OnReceivingData (see page 1295)

20.1.2.6.3 THTTPReqResp.OnReceivingData Event

Occurs after THTTPReqResp (see page 1276) receives each chunk of data when the content is large.

Pascal

```
property OnReceivingData: TReceivingDataEvent;
```

C++

```
__property TReceivingDataEvent OnReceivingData;
```

Description

Write (see page 3707) an OnReceivingData event handler to provide the end-user with feedback of the progress when receiving a request that contains a large amount of data. When the content of a response is very large (for example, when it contains large attachments), it is broken into blocks, which are sent separately. After receiving each of these blocks, the Receive (see page 1285) method generates an OnReceivingData event. You can use the event handler to update a progress bar or similar control to indicate how much of the response has been received.

See Also

OnPostingData (see page 1294)

Receive (see page 1285)

20.1.2.6.4 THTTPReqResp.OnWinInetError Event

Pascal

```
property OnWinInetError: TWinInetErrorEvent;
```

C++

```
__property TWinInetErrorEvent OnWinInetError;
```

Description

This is OnWinInetError, a member of class THTTPReqResp.

20.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHTTPReqResp (see page 1296)	This is class SOAPHTTPTrans.IHTTPReqResp.

20.2.1 IHTTPReqResp Interface

Class Hierarchy

SOAPHTTPTrans.IHTTPReqResp

Pascal

```
IHTTPReqResp = interface;
```

C++

```
__interface IHTTPReqResp;
```

File

SOAPHTTPTrans

Description

This is class SOAPHTTPTrans.IHTTPReqResp.

20.2.1.1 IHTTPReqResp Members

The following tables list the members exposed by IHTTPReqResp.

Public Methods

	Name	Description
	GetHTTPReqResp (see page 1297)	This is GetHTTPReqResp, a member of class IHTTPReqResp.

20.2.1.2 IHTTPReqResp Methods

The methods of the IHTTPReqResp class are listed here.

Public Methods

	Name	Description
	GetHTTPReqResp (see page 1297)	This is GetHTTPReqResp, a member of class IHTTPReqResp.

20.2.1.2.1 IHTTPReqResp.GetHTTPReqResp Method

Pascal

```
function GetHTTPReqResp: THTTPReqResp;
```

C++

```
__fastcall THTTPReqResp GetHTTPReqResp();
```

Description

This is GetHTTPReqResp, a member of class IHTTPReqResp.

20.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	SOAPInvokeOptions (see page 1298)	SOAPInvokeOptions and TSOAPInvokeOptions (see page 1300) represent flags that customize how to handle a SOAP message.

20.3.1 SOAPHTTPTrans.SOAPInvokeOptions Enumeration

SOAPInvokeOptions and TSOAPInvokeOptions (see page 1300) represent flags that customize how to handle a SOAP message.

Pascal

```
SOAPInvokeOptions = (
  soNoValueForEmptySOAPAction,
  soIgnoreInvalidCerts,
  soNoSOAPActionHeader,
  soAutoCheckAccessPointViaUDDI
);
```

C++

```
enum SOAPInvokeOptions {
  soNoValueForEmptySOAPAction,
  soIgnoreInvalidCerts,
  soNoSOAPActionHeader,
  soAutoCheckAccessPointViaUDDI
};
```

File

SOAPHTTPTrans

Description

TSOAPInvokeOptions (see page 1300) is a set of TSoapInvokeOption values. TSoapInvokeOption values are flags that specify how THTTPReqResp (see page 1276) manages the sending and receiving of SOAP messages. The following table lists the possible values:

Value	Meaning
soNoValueForEmptySOAPAction	When calling Web services that specify empty SOAP actions, omit the SOAPAction header rather then generating a SOAPAction header whose value is an empty string.
soIgnoreInvalidCerts	When the server uses SSL but has an invalid certificate, treat it as if the certificate had been valid. This is useful when debugging applications that use a secure server.
soNoSOAPActionHeader	Always omit the SOAP action header when calling Web services. It is advised that you only use this option with extreme caution.
soAutoCheckAccessPointViaUDDI	Provide fail-over support for messages that result in a status code of 404, 405, or 410. This option causes THTTPReqResp (see page 1276) to look up the UDDI entry that was used to import the Web Service description and attempt to locate a new binding.

20.4 Types

The following table lists types in this documentation.

Types

Name	Description
TBeforePostEvent (see page 1299)	TBeforePostEvent is the type of the OnBeforePost event of THTTPReqResp (see page 1276).
TPostingDataEvent (see page 1299)	TPostingDataEvent is the type of the OnPostingData event of THTTPReqResp (see page 1276).
TReceivingDataEvent (see page 1300)	TReceivingDataEvent is the type of the OnReceivingData event of THTTPReqResp (see page 1276).
TSOAPInvokeOptions (see page 1300)	This is type SOAPHTTPTrans.TSOAPInvokeOptions.
TWinInetErrorEvent (see page 1301)	This is type SOAPHTTPTrans.TWinInetErrorEvent.

20.4.1 SOAPHTTPTrans.TBeforePostEvent Type

TBeforePostEvent is the type of the OnBeforePost event of THTTPReqResp (see page 1276).

Pascal

```
TBeforePostEvent = procedure (const HTTPReqResp: THTTPReqResp; Data: Pointer) of object;
```

C++

```
(const HTTPReqResp: THTTPReqResp; Data: Pointer) ( TBeforePostEvent)();
```

File

SOAPHTTPTrans

Description

TBeforePostEvent is the type of the OnBeforePost event handler, which is called before an HTTP request message containing SOAP content is sent to a Web Service provider.

HTTPReqResp is the component that generates and sends the HTTP request message.

Data is a handle to the wininet request object.

See Also

Procedural types

Creating events

20.4.2 SOAPHTTPTrans.TPostingDataEvent Type

TPostingDataEvent is the type of the OnPostingData event of THTTPReqResp (see page 1276).

Pascal

```
TPostingDataEvent = procedure (Sent: Integer; Total: Integer) of object;
```

C++

```
(Sent: Integer; Total: Integer) ( TPostingDataEvent)();
```

File

SOAPHTTPTrans

Description

TPostingDataEvent is the type of the OnPostingData event handler, which is called after each chunk of data that is sent when a request is very large.

Sent indicates the (cumulative) number of bytes that have already been sent.

Total indicates the total number of bytes that need to be sent.

See Also

Procedural types

Creating events

20.4.3 SOAPHTTPTrans.TReceivingDataEvent Type

TReceivingDataEvent is the type of the OnReceivingData event of THTTPReqResp (see page 1276).

Pascal

```
TReceivingDataEvent = procedure (Read: Integer; Total: Integer) of object;
```

C++

```
(Read: Integer; Total: Integer) ( TReceivingDataEvent)();
```

File

SOAPHTTPTrans

Description

TReceivingDataEvent is the type of the OnReceivingData event handler, which is called after each chunk of data that is received when a large response has been broken into smaller chunks.

Read (see page 3634) indicates the (cumulative) number of bytes that have been received.

Total indicates the total number of bytes that are expected.

See Also

Procedural types

Creating events

20.4.4 SOAPHTTPTrans.TSOAPInvokeOptions Type

Pascal

```
TSOAPInvokeOptions = set of SOAPInvokeOptions;
```

C++

```
SOAPInvokeOptions TSOAPInvokeOptions;
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TSOAPInvokeOptions.

20.4.5 SOAPHTTPTrans.TWinInetErrorEvent Type

Pascal

```
TWinInetErrorEvent = function (LastError: DWord; Request: Pointer): DWord of object;
```

C++

```
(LastError: DWord; Request: Pointer): DWord ( TWinInetErrorEvent)();
```

File

SOAPHTTPTrans

Description

This is type SOAPHTTPTrans.TWinInetErrorEvent.

21 SOAPLinked Namespace

This is namespace SOAPLinked.

Topics

Name	Description
Classes (see page 1305)	The following table lists classes in this documentation.
Interfaces (see page 1332)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 1336)	The following table lists structs, records, enums in this documentation.
Types (see page 1337)	The following table lists types in this documentation.

Classes

	Name	Description
	ECustomException (see page 1305)	This is class SOAPLinked.ECustomException.
	TFTCompT (see page 1311)	This is class SOAPLinked.TFTCompT.
	TFTCompTI (see page 1313)	This is class SOAPLinked.TFTCompTI.
	TLinkedRIO (see page 1316)	TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.
	TLinkedWebNode (see page 1321)	TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.
	TLogLinkedWebNode (see page 1326)	This is class SOAPLinked.TLogLinkedWebNode.
	TWinHandleUser (see page 1329)	This is class SOAPLinked.TWinHandleUser.

Interfaces

	Name	Description
	IIntegerTest (see page 1332)	This is class SOAPLinked.IIntegerTest.
	IObjectGetter (see page 1333)	This is class SOAPLinked.IObjectGetter.
	IStringTest (see page 1334)	This is class SOAPLinked.IStringTest.

Structs, Records, Enums

	Name	Description
	EventKind (see page 1336)	This is record SOAPLinked.EventKind.

Types

Name	Description
TCStringEvent (see page 1337)	This is type SOAPLinked.TCStringEvent.
TCustExCallback (see page 1337)	This is type SOAPLinked.TCustExCallback.
TStringEvent (see page 1338)	This is type SOAPLinked.TStringEvent.

TVStringEvent (see page 1338)	This is type SOAPLinked.TVStringEvent.

21.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ECustomException (see page 1305)	This is class SOAPLinked.ECustomException.
	TFTCompT (see page 1311)	This is class SOAPLinked.TFTCompT.
	TFTCompTI (see page 1313)	This is class SOAPLinked.TFTCompTI.
	TLinkedRIO (see page 1316)	TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.
	TLinkedWebNode (see page 1321)	TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.
	TLogLinkedWebNode (see page 1326)	This is class SOAPLinked.TLogLinkedWebNode.
	TWinHandleUser (see page 1329)	This is class SOAPLinked.TWinHandleUser.

21.1.1 ECustomException Class

Class Hierarchy

Pascal

```
ECustomException = class(Exception);
```

C++

```
class ECustomException : public Exception;
```

File

SOAPLinked

Description

This is class SOAPLinked.ECustomException.

21.1.1.1 ECustomException Members

The following tables list the members exposed by ECustomException.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.

	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CatchException (see page 1310)	This is CatchException, a member of class ECustomException.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 1307)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 1309)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceCount (see page 1310)	This is InstanceCount, a member of class ECustomException.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowExcept (see page 1310)	This is ThrowExcept, a member of class ECustomException.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3380)	Returns a string containing the class name.
	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

21.1.1.2 ECustomException.Create Constructor

Instantiates an instance of an exception with a simple message string.

Pascal

```
constructor Create(const msg: string);
```

C++

```
__fastcall ECustomException(const UnicodeString msg);
```

Description

Call Create to construct an exception object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
```

```
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  In addition to displaying the exception message, which
  happens by default, the following code shuts down the
  application when an exception is not caught and handled.
  AppException should be declared a method of TForm1.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Application.OnException := AppException;
  end;

  procedure TForm1.AppException(Sender: TObject; E: Exception);
  begin
    Application.ShowException(E);
    Application.Terminate;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    raise EPasswordInvalid.Create('Incorrect password entered');
  end;
```

See Also

CreateFmt (see page 3954)

CreateRes (see page 3956)

CreateResFmt (see page 3957)

Message (see page 3965)

21.1.1.3 ECustomException.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~ECustomException();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

21.1.1.4 ECustomException Methods

The methods of the ECustomException class are listed here.

Public Methods

	Name	Description
	CatchException (see page 1310)	This is CatchException, a member of class ECustomException.
	InstanceCount (see page 1310)	This is InstanceCount, a member of class ECustomException.
	ThrowExcept (see page 1310)	This is ThrowExcept, a member of class ECustomException.

21.1.1.4.1 ECustomException.CatchException Method

Pascal

```
class function CatchException(var ExceptTypeName: string; var Address: Pointer; TypedHandler:
boolean; Callback: TCustExCallback): boolean; static;
```

C++

```
static __fastcall boolean CatchException(UnicodeString ExceptTypeName, void * Address, boolean
TypedHandler, TCustExCallback Callback);
```

Description

This is CatchException, a member of class ECustomException.

21.1.1.4.2 ECustomException.InstanceCount Method

Pascal

```
class function InstanceCount: Integer; static;
```

C++

```
static __fastcall int InstanceCount();
```

Description

This is InstanceCount, a member of class ECustomException.

21.1.1.4.3 ECustomException.ThrowExcept Method

Pascal

```
class procedure ThrowExcept(const msg: string); static;
```

C++

```
static __fastcall ThrowExcept(const UnicodeString msg);
```

Description

This is ThrowExcept, a member of class ECustomException.

21.1.2 TFTCompT Class

Class Hierarchy

Pascal

```
TFTCompT = class(TComponent);
```

C++

```
class TFTCompT : public TComponent;
```

File

SOAPLinked

Description

This is class SOAPLinked.TFTCompT.

21.1.2.1 TFTCompT Members

The following tables list the members exposed by TFTCompT.

Public Methods

	Name	Description
	fireEvent (see page 1312)	This is fireEvent, a member of class TFTCompT.
	fireEventEx (see page 1312)	This is fireEventEx, a member of class TFTCompT.

Published Events

	Name	Description
	CStringEvent (see page 1312)	This is CStringEvent, a member of class TFTCompT.
	StringEvent (see page 1312)	This is StringEvent, a member of class TFTCompT.
	VStringEvent (see page 1313)	This is VStringEvent, a member of class TFTCompT.

21.1.2.2 TFTCompT Methods

The methods of the TFTCompT class are listed here.

Public Methods

	Name	Description
	fireEvent (see page 1312)	This is fireEvent, a member of class TFTCompT.
	fireEventEx (see page 1312)	This is fireEventEx, a member of class TFTCompT.

21.1.2.2.1 TFTCompT.fireEvent Method

Pascal

```
function fireEvent(S: string; ek: EventKind): string;
```

C++

```
__fastcall UnicodeString fireEvent(UnicodeString S, EventKind ek);
```

Description

This is fireEvent, a member of class TFTCompT.

21.1.2.2.2 TFTCompT.fireEventEx Method

Pascal

```
procedure fireEventEx(var S: string; ek: EventKind);
```

C++

```
__fastcall fireEventEx(UnicodeString S, EventKind ek);
```

Description

This is fireEventEx, a member of class TFTCompT.

21.1.2.3 TFTCompT Events

The events of the TFTCompT class are listed here.

Published Events

	Name	Description
	CStringEvent (see page 1312)	This is CStringEvent, a member of class TFTCompT.
	StringEvent (see page 1312)	This is StringEvent, a member of class TFTCompT.
	VStringEvent (see page 1313)	This is VStringEvent, a member of class TFTCompT.

21.1.2.3.1 TFTCompT.CStringEvent Event

Pascal

```
property CStringEvent: TCStringEvent;
```

C++

```
__property TCStringEvent CStringEvent;
```

Description

This is CStringEvent, a member of class TFTCompT.

21.1.2.3.2 TFTCompT.StringEvent Event

Pascal

```
property StringEvent: TStringEvent;
```

C++

```
__property TStringEvent StringEvent;
```

Description

This is StringEvent, a member of class TFTCompT.

21.1.2.3.3 TFTCompT.VStringEvent Event

Pascal

```
property VStringEvent: TVStringEvent;
```

C++

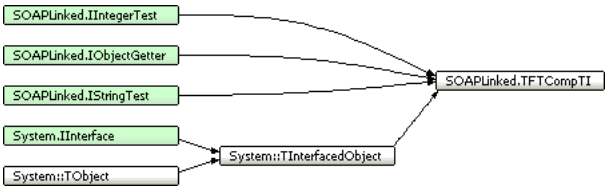
```
__property TVStringEvent VStringEvent;
```

Description

This is VStringEvent, a member of class TFTCompT.

21.1.3 TFTCompTI Class

Class Hierarchy



Pascal

```
TFTCompTI = class(TInterfacedObject, IStringTest, IIntegerTest, IObjectGetter);
```

C++

```
class TFTCompTI : public TInterfacedObject, public IStringTest, public IIntegerTest, public
IObjectGetter;
```

File

SOAPLinked

Description

This is class SOAPLinked.TFTCompTI.

21.1.3.1 TFTCompTI Members

The following tables list the members exposed by TFTCompTI.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.

	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	Add (see page 1315)	This is Add, a member of class TFTCompTI.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	Append (see page 1315)	This is Append, a member of class TFTCompTI.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Doubler (see page 1315)	This is Doubler, a member of class TFTCompTI.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAsObject (see page 1316)	This is GetAsObject, a member of class TFTCompTI.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
	UpCase (see page 1316)	This is UpCase, a member of class TFTCompTI.

Public Properties

	Name	Description
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

21.1.3.2 TFTCompTI Methods

The methods of the TFTCompTI class are listed here.

Public Methods

	Name	Description
	Add (see page 1315)	This is Add, a member of class TFTCompTI.
	Append (see page 1315)	This is Append, a member of class TFTCompTI.
	Doubler (see page 1315)	This is Doubler, a member of class TFTCompTI.
	GetAsObject (see page 1316)	This is GetAsObject, a member of class TFTCompTI.
	UpCase (see page 1316)	This is UpCase, a member of class TFTCompTI.

21.1.3.2.1 TFTCompTI.Add Method

Pascal

```
function Add(const I1: Integer; const I2: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Add(const int I1, const int I2);
```

Description

This is Add, a member of class TFTCompTI.

21.1.3.2.2 TFTCompTI.Append Method

Pascal

```
function Append(const Str1: string; const Str2: string): string; cdecl;
```

C++

```
__cdecl __fastcall UnicodeString Append(const UnicodeString Str1, const UnicodeString Str2);
```

Description

This is Append, a member of class TFTCompTI.

21.1.3.2.3 TFTCompTI.Doubler Method

Pascal

```
procedure Doubler(var I: Integer); cdecl;
```

C++

```
__cdecl __fastcall Doubler(int I);
```

Description

This is Doubler, a member of class TFTCompTI.

21.1.3.2.4 TFTCompTI.GetAsObject Method

Pascal

```
function GetAsObject: TObject;
```

C++

```
__fastcall TObject * GetAsObject();
```

Description

This is GetAsObject, a member of class TFTCompTI.

21.1.3.2.5 TFTCompTI.UpCase Method

Pascal

```
procedure UpCase(var Str1: string); stdcall;
```

C++

```
__fastcall __stdcall UpCase(UnicodeString Str1);
```

Description

This is UpCase, a member of class TFTCompTI.

21.1.4 TLinkedRIO Class

TLinkedRIO makes calls to a registered invokable interface that is implemented in the same application.

Class Hierarchy

Pascal

```
TLinkedRIO = class(TRIO);
```

C++

```
class TLinkedRIO : public TRIO;
```

File

SOAPLinked

Description

Use TLinkedRIO to generate calls to registered invokable interfaces within a Web Service application. Unlike other remote interfaced objects (TRIO descendants), TLinkedRIO provides no mechanism for connecting to a remote server. Rather, it passes calls directly to a TLinkedWebNode (see page 1321) object, which uses an internal invoker component to look up the target interface and forward calls to the implementation object.

TLinkedRIO is intended primarily for debugging Web Service applications during development. Instead of directly calling the implementation objects in the application, TLinkedRIO marshals the call into its SOAP representation and uses the invocation registry to locate the implementation object and forward the encoded request. If you use the CreateFile (see page 1318) constructor (Delphi) or supply a pair of file names to the constructor (C++), TLinkedRIO saves the encoded value of every method call and the encoded results so that you can track the "messages" that your application receives and sends.

Before TLinkedRIO can call a method table on an invokable interface, the interface must be registered with the invocation

registry. To register an invokable interface, use the global InvRegistry function to access the registry, and use its RegisterInterface method.

See Also

InvRegistry

TInvokableClassRegistry

TSoapPascalInvoker (see page 1350)

TLinkedWebNode (see page 1321)

21.1.4.1 TLinkedRIO Members

The following tables list the members exposed by TLinkedRIO.

Protected Methods

	Name	Description
	GetResponseStream (see page 1319)	Returns a newly created stream that receives the results of executing a method call.

Public Methods

	Name	Description
	Create (see page 1317)	Creates an instance of TLinkedRIO (see page 1316).
	CreateFile (see page 1318)	Creates an instance of TLinkedRIO (see page 1316).
	Destroy (see page 1318)	Destroys the instance of a TLinkedRIO (see page 1316) object.
	QueryInterface (see page 1319)	Implements the IInterface (see page 3502) QueryInterface method.

Public Properties

	Name	Description
	WebNode (see page 1320)	Handles the execution of method calls on the invokable interface.

21.1.4.2 TLinkedRIO.Create Constructor

Creates an instance of TLinkedRIO (see page 1316).

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
```

C++

```
virtual __fastcall TLinkedRIO(TComponent * AOwner);
```

Description

Call Create to instantiate a TLinkedRIO (see page 1316) object at runtime.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TLinkedRIO (see page 1316) instance. It becomes the value of the Owner property. If the TLinkedRIO (see page 1316) instance is created with AOwner set to nil, then it is automatically freed when its reference count drops to zero.

Tip: To save the encoded method calls that TLinkedRIO (see page 1316) generates and the encoded results that it receives, use the CreateFile (see page 1318) constructor instead.

See Also

RefCount

Owner

CreateFile (see page 1318)

21.1.4.3 TLinkedRIO.CreateFile Constructor

Creates an instance of TLinkedRIO (see page 1316).

Pascal

```
constructor CreateFile(AOwner: TComponent; ReqFile: string; RespFile: string); overload;
```

C++

```
__fastcall TLinkedRIO(TComponent * AOwner, UnicodeString ReqFile, UnicodeString RespFile);
```

Description

Call Create (see page 1317) to instantiate a TLinkedRIO (see page 1316) object when you want to save all of the encoded requests that it generates and the encoded results it receives to a file.

The AOwner parameter specifies another component (usually a form or data module) that is responsible for freeing the TLinkedRIO (see page 1316) instance. It becomes the value of the Owner property. If the TLinkedRIO (see page 1316) instance is created with AOwner set to nil, then it is automatically freed when its reference count drops to zero.

ReqFile is a file to which TLinkedRIO (see page 1316) writes every encoded request before passing it on to WebNode (see page 1320) for execution. If ReqFile is an empty string, TLinkedRIO (see page 1316) automatically generates a file with the same name as the invoked method. This lets you generate separate log files for each method of the interface.

RespFile is a file to which TLinkedRIO (see page 1316) writes every encoded response it receives from WebNode (see page 1320) after executing a method call. If RespFile is an empty string, TLinkedRIO (see page 1316) automatically generates a file with the name of the method and the string _resp appended. This lets you generate separate log files for responses to each method of the interface.

See Also

RefCount

Owner

Create (see page 1317)

WebNode (see page 1320)

21.1.4.4 TLinkedRIO.Destroy Destructor

Destroys the instance of a TLinkedRIO (see page 1316) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLinkedRIO();
```

Description

There is no need to explicitly free a TLinkedRIO (see page 1316) instance.

If the TLinkedRIO (see page 1316) object is created with an Owner, then that Owner handles the freeing of the TLinkedRIO (see page 1316) instance. If the TLinkedRIO (see page 1316) object is created without an Owner, then it is automatically freed

when the reference count on its interface drops to zero.

See Also

Create (see page 1317)

Owner

RefCount

21.1.4.5 TLinkedRIO Methods

The methods of the TLinkedRIO class are listed here.

Protected Methods

	Name	Description
	GetResponseStream (see page 1319)	Returns a newly created stream that receives the results of executing a method call.

Public Methods

	Name	Description
	QueryInterface (see page 1319)	Implements the IInterface (see page 3502) QueryInterface method.

21.1.4.5.1 TLinkedRIO.GetResponseStream Method

Returns a newly created stream that receives the results of executing a method call.

Pascal

```
function GetResponseStream(BindingType: TWebServiceBindingType): TStream; override;
```

C++

```
virtual __fastcall TStream GetResponseStream(TWebServiceBindingType BindingType);
```

Description

TRIO calls GetResponseStream internally when it executes a method call. It passes the returned stream as the Response parameter of the Execute method on the interface that the WebNode property specifies. After this stream receives the encoded response of a method call, the RIO component decodes the response and frees the stream.

BindingType indicates the expected coding of the response that the stream will hold. GetResponseStream can use this information to determine what type of stream to create.

See Also

Execute (see page 6618)

WebNode

21.1.4.5.2 TLinkedRIO.QueryInterface Method

Implements the IInterface (see page 3502) QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface is called through the IInterface (see page 3502) interface to obtain an interface pointer for the interface identified by the IID parameter. In Delphi, this call is generated when an application casts TLinkedRIO (see page 1316) to an invokable interface using the as operator.

TLinkedRIO (see page 1316) can only return a single type of interface pointer. The first time an application calls QueryInterface, TLinkedRIO (see page 1316) generates an in-memory method table to support calls to the methods on the requested interface.

IID specifies the invokable interface that the application wants to call using this TLinkedRIO (see page 1316) instance. If the TLinkedRIO (see page 1316) object has already been typecast to an interface, then IID must identify that interface. If the TLinkedRIO (see page 1316) instance has not yet been typecast to an interface, then IID must be a registered invokable interface.

Obj returns an interface pointer for the specified interface, unless TLinkedRIO (see page 1316) was already implementing a different interface or IID identified an interface that was not registered.

QueryInterface returns 0 (S_OK (see page 3860)) if Obj successfully returns an interface pointer. It returns E_NOINTERFACE (see page 3845) otherwise.

Note: In C++, a second, templatized syntax takes a variable that is a DelphiInterface (see page 3256) wrapper for the desired interface type.

If QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

InvRegistry

RegisterInterface

RefCount

21.1.4.6 TLinkedRIO Properties

The properties of the TLinkedRIO class are listed here.

Public Properties

	Name	Description
R	WebNode (see page 1320)	Handles the execution of method calls on the invokable interface.

21.1.4.6.1 TLinkedRIO.WebNode Property

Handles the execution of method calls on the invokable interface.

Pascal

```
property WebNode: TLinkedWebNode;
```

C++

```
__property TLinkedWebNode WebNode;
```

Description

TLinkedRIO (see page 1316) uses WebNode internally to execute method calls. It is the implementation of the IWebNode (see page 6616) interface that all TRIO descendants use to send method calls to the Web Service application. In the case of TLinkedRIO (see page 1316), this object can only execute calls on invokable interfaces that are implemented in the same application.

See Also

WebNode

IWebNode (see page 6616)

21.1.5 TLinkedWebNode Class

TLinkedWebNode executes a method call on an invokable interface that is implemented in the same application.

Class Hierarchy

Pascal

```
TLinkedWebNode = class(TComponent, IWebNode);
```

C++

```
class TLinkedWebNode : public TComponent, public IWebNode;
```

File

SOAPLinked

Description

TLinkedWebNode handles the communication between a TLinkedRIO (see page 1316) component and the implementation classes of registered invokable interfaces in the same application. To this end, it implements the IWebNode (see page 6616) interface that all TRIO descendants use for executing encoded method calls.

TLinkedRIO (see page 1316) uses an internal invoker (TSoapPascalInvoker (see page 1350)) to look up the invokable interface in the invocation registry and forward encoded requests for execution.

See Also

IWebNode (see page 6616)

TLinkedRIO (see page 1316)

TSoapPascalInvoker (see page 1350)

21.1.5.1 TLinkedWebNode Members

The following tables list the members exposed by TLinkedWebNode.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 1324)	Returns the value of the MimeBoundary (see page 1326) property.
V	GetResponseStream (see page 1325)	Returns a stream to which the application can write the response to the method call.
	SetMimeBoundary (see page 1325)	Sets the value of the MimeBoundary (see page 1326) property.

Public Methods

	Name	Description
V	BeforeExecute (see page 1323)	Caches the RTTI for the method that is about to be executed.

	Create (see page 1322)	Creates a new TLinkedWebNode (see page 1321) object.
	Destroy (see page 1322)	Frees the memory associated with the TLinkedWebNode (see page 1321) object.
	Execute (see page 1323)	This is the overview for the Execute method overload.
	Execute (see page 6618)	This is the overview for the Execute method overload.

Public Properties

	Name	Description
	Invoker (see page 1326)	Executes encoded method calls received from the associated TLinkedRIO (see page 1316) object.
	MimeBoundary (see page 1326)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

21.1.5.2 TLinkedWebNode.Create Constructor

Creates a new TLinkedWebNode (see page 1321) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLinkedWebNode(TComponent * AOwner);
```

Description

Most applications do not create instances of TLinkedWebNode (see page 1321). It is created and used by the TLinkedRIO (see page 1316) class.

The Owner parameter specifies the component that is responsible for freeing the TLinkedWebNode (see page 1321) instance.

See Also

Owner

TLinkedRIO (see page 1316)

21.1.5.3 TLinkedWebNode.Destroy Destructor

Frees the memory associated with the TLinkedWebNode (see page 1321) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLinkedWebNode();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3372)

21.1.5.4 TLinkedWebNode Methods

The methods of the TLinkedWebNode class are listed here.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 1324)	Returns the value of the MimeBoundary (see page 1326) property.
	GetResponseStream (see page 1325)	Returns a stream to which the application can write the response to the method call.
	SetMimeBoundary (see page 1325)	Sets the value of the MimeBoundary (see page 1326) property.

Public Methods

	Name	Description
	BeforeExecute (see page 1323)	Caches the RTTI for the method that is about to be executed.
	Execute (see page 1323)	This is the overview for the Execute method overload.

21.1.5.4.1 TLinkedWebNode.BeforeExecute Method

Caches the RTTI for the method that is about to be executed.

Pascal

```
procedure BeforeExecute(const IntfMetaData: TIntfMetaData; const MethodMetaData:
TIntfMethEntry; MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler); virtual;
```

C++

```
virtual __fastcall BeforeExecute(const TIntfMetaData IntfMetaData, const TIntfMethEntry
MethodMetaData, int MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute implements the IWebNode (see page 6616) method of the same name. The remote interface object (TRio) calls this method after its OnBeforeExecute event, but before calling the Execute (see page 1324) method to send a request to the Web Service application. BeforeExecute allows an IWebNode (see page 6616) implementation class to perform any transport-specific tasks prior to establishing a connection. Because TLinkedWebNode (see page 1321) only works with interfaces implemented in the same executable, there is no transport protocol at all, and no transport-specific tasks to perform. BeforeExecute merely caches the RTTI for the method that is about to be executed.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method. If AttachHandler is nil (Delphi) or NULL (C++), the method call does not require passing any attachments.

See Also

Execute (see page 1324)

OnBeforeExecute

BeforeExecute (see page 6617)

21.1.5.4.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TLinkedWebNode.Execute (String, TStream) (see page 1324)	Executes a SOAP method call and returns the encoded results.
	TLinkedWebNode.Execute (TStream) (see page 1324)	Executes a SOAP method call and returns the encoded results.
	TLinkedWebNode.Execute (TStream, TStream) (see page 1324)	Executes a SOAP method call and returns the encoded results.

21.1.5.4.2.1 TLinkedWebNode.Execute Method (String, TStream)

Executes a SOAP method call and returns the encoded results.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); virtual; overload;
procedure Execute(const Request: TStream; Response: TStream); virtual; overload;
function Execute(const Request: TStream): TStream; overload;
```

C++

```
virtual __fastcall Execute(const UnicodeString DataMsg, TStream Resp);
virtual __fastcall Execute(const TStream Request, TStream Response);
__fastcall TStream Execute(const TStream Request);
```

Description

Execute implements the IWebNode (see page 6616) method of the same name. TLinkedRio uses this method to execute a method on an invokable interface that has been encoded as a SOAP method call.

Execute

1. Uses the invocation registry to look up the implementation class for the invokable interface.
2. Uses its internal invoker (the Invoker (see page 1326) property) to unmarshal the request specified by Request or DataMsg and pass it to the implementation class for execution.
3. Returns the encoded results from the invoker in the stream specified by Response or, using the third syntax, in a stream that is the return value of the method. When using the third syntax, the caller is responsible for freeing the returned stream.

See Also

Execute (see page 6618)

TLinkedRIO (see page 1316)

Invoker (see page 1326)

21.1.5.4.3 TLinkedWebNode.GetMimeBoundary Method

Returns the value of the MimeBoundary (see page 1326) property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read implementation of the MimeBoundary (see page 1326) property.

See Also

MimeBoundary (see page 1326)

21.1.5.4.4 TLinkedWebNode.GetResponseStream Method

Returns a stream to which the application can write the response to the method call.

Pascal

```
function GetResponseStream: TStream; virtual;
```

C++

```
virtual __fastcall TStream GetResponseStream();
```

Description

GetResponseStream creates and returns a memory stream. The linked RIO object uses this stream as its response stream, which receives the response that results from the method call.

See Also

GetResponseStream

21.1.5.4.5 TLinkedWebNode.SetMimeBoundary Method

Sets the value of the MimeBoundary (see page 1326) property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary (see page 1326) property.

Value is the new MIME boundary string.

See Also

MimeBoundary (see page 1326)

21.1.5.5 TLinkedWebNode Properties

The properties of the TLinkedWebNode class are listed here.

Public Properties

	Name	Description
	Invoker (see page 1326)	Executes encoded method calls received from the associated TLinkedRIO (see page 1316) object.
	MimeBoundary (see page 1326)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

21.1.5.5.1 TLinkedWebNode.Invoker Property

Executes encoded method calls received from the associated TLinkedRIO (see page 1316) object.

Pascal

```
property Invoker: TSoapPascalInvoker;
```

C++

```
__property TSoapPascalInvoker Invoker;
```

Description

Invoker is the internal invoker component that looks up method calls in the invocation registry, forwards encoded requests to the registered implementation class, and returns the encoded results for the associated TLinkedRIO (see page 1316) object.

See Also

TLinkedRIO (see page 1316)

21.1.5.5.2 TLinkedWebNode.MimeBoundary Property

Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Pascal

```
property MimeBoundary: string;
```

C++

```
__property UnicodeString MimeBoundary;
```

Description

When the encoded method call is included in a multipart form, MimeBoundary is the value of the MIME boundary header that uniquely identifies the part of the form that contains the method call. If the method call is not included in a multipart form, MimeBoundary is an empty string.

See Also

MimeBoundary (see page 6619)

21.1.6 TLogLinkedWebNode Class

Class Hierarchy

Pascal

```
TLogLinkedWebNode = class(TLinkedWebNode);
```

C++

```
class TLogLinkedWebNode : public TLinkedWebNode;
```

File

SOAPLinked

Description

This is class SOAPLinked.TLogLinkedWebNode.

21.1.6.1 TLogLinkedWebNode Members

The following tables list the members exposed by TLogLinkedWebNode.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 1324)	Returns the value of the MimeBoundary (see page 1326) property.
V	GetResponseStream (see page 1328)	Returns a stream to which the application can write the response to the method call.
	SetMimeBoundary (see page 1325)	Sets the value of the MimeBoundary (see page 1326) property.

Public Methods

	Name	Description
V	BeforeExecute (see page 1323)	Caches the RTTI for the method that is about to be executed.
	BeforeExecute (see page 6617)	Performs any transport-specific setup before executing an encoded method call.
V	Create (see page 1327)	Creates a new TLinkedWebNode object.
V	Destroy (see page 1328)	Frees the memory associated with the TLinkedWebNode object.
V	Execute (see page 1328)	This is Execute, a member of class TLogLinkedWebNode.
V	Execute (see page 1323)	This is the overview for the Execute method overload.
	Execute (see page 6618)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 6619)	Returns the value of the MimeBoundary (see page 6619) property.
	SetMimeBoundary (see page 6619)	Sets the value of the MimeBoundary (see page 6619) property.

Public Properties

	Name	Description
R	Invoker (see page 1326)	Executes encoded method calls received from the associated TLinkedRIO (see page 1316) object.
	MimeBoundary (see page 1326)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.
	MimeBoundary (see page 6619)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Published Properties

	Name	Description
	ReqFile (see page 1329)	This is ReqFile, a member of class TLogLinkedWebNode.
	RespFile (see page 1329)	This is RespFile, a member of class TLogLinkedWebNode.

21.1.6.2 TLogLinkedWebNode.Create Constructor

Creates a new TLinkedWebNode object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLogLinkedWebNode(TComponent * AOwner);
```

Description

Most applications do not create instances of TLinkedWebNode. It is created and used by the TLinkedRIO class.

The Owner parameter specifies the component that is responsible for freeing the TLinkedWebNode instance.

See Also

Owner

TLinkedRIO (see page 1316)

21.1.6.3 TLogLinkedWebNode.Destroy Destructor

Frees the memory associated with the TLinkedWebNode object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLogLinkedWebNode();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3372)

21.1.6.4 TLogLinkedWebNode Methods

The methods of the TLogLinkedWebNode class are listed here.

Protected Methods

	Name	Description
	GetResponseStream (see page 1328)	Returns a stream to which the application can write the response to the method call.

Public Methods

	Name	Description
	Execute (see page 1328)	This is Execute, a member of class TLogLinkedWebNode.

21.1.6.4.1 TLogLinkedWebNode.Execute Method

Pascal

```
procedure Execute(const Request: TStream; Response: TStream); override;
```

C++

```
virtual __fastcall Execute(const TStream Request, TStream Response);
```

Description

This is Execute, a member of class TLogLinkedWebNode.

21.1.6.4.2 TLogLinkedWebNode.GetResponseStream Method

Returns a stream to which the application can write the response to the method call.

Pascal

```
function GetResponseStream: TStream; override;
```

C++

```
virtual __fastcall TStream GetResponseStream();
```

Description

GetResponseStream creates and returns a memory stream. The linked RIO object uses this stream as its response stream, which receives the response that results from the method call.

See Also

GetResponseStream

21.1.6.5 TLogLinkedWebNode Properties

The properties of the TLogLinkedWebNode class are listed here.

Published Properties

	Name	Description
	ReqFile (see page 1329)	This is ReqFile, a member of class TLogLinkedWebNode.
	RespFile (see page 1329)	This is RespFile, a member of class TLogLinkedWebNode.

21.1.6.5.1 TLogLinkedWebNode.ReqFile Property

Pascal

```
property ReqFile: string;
```

C++

```
__property UnicodeString ReqFile;
```

Description

This is ReqFile, a member of class TLogLinkedWebNode.

21.1.6.5.2 TLogLinkedWebNode.RespFile Property

Pascal

```
property RespFile: string;
```

C++

```
__property UnicodeString RespFile;
```

Description

This is RespFile, a member of class TLogLinkedWebNode.

21.1.7 TWinHandleUser Class

Class Hierarchy

Pascal

```
TWinHandleUser = class(TObject);
```

C++

```
class TWinHandleUser : public TObject;
```

File

SOAPLinked

Description

This is class SOAPLinked.TWinHandleUser.

21.1.7.1 TWinHandleUser Members

The following tables list the members exposed by TWinHandleUser.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
	UseHandle1 (see page 1331)	This is UseHandle1, a member of class TWinHandleUser.
	UseHandle2 (see page 1331)	This is UseHandle2, a member of class TWinHandleUser.

21.1.7.2 TWinHandleUser Methods

The methods of the TWinHandleUser class are listed here.

Public Methods

	Name	Description
	UseHandle1 (see page 1331)	This is UseHandle1, a member of class TWinHandleUser.
	UseHandle2 (see page 1331)	This is UseHandle2, a member of class TWinHandleUser.

21.1.7.2.1 TWinHandleUser.UseHandle1 Method

Pascal

```
procedure UseHandle1(ahdesk: HDESK; aenhMf: HENHMETAFILE; amf: HMETAFILE; apalette: HPALETTE;
apen: HPEN; adc: HDC; abmp: HBITMAP; awnd: HWND; ahkl: HKL; aicon: HICON; afont: HFONT);
virtual;
```

C++

```
virtual __fastcall UseHandle1(HDESK ahdesk, HENHMETAFILE aenhMf, HMETAFILE amf, HPALETTE
apalette, HPEN apen, HDC adc, HBITMAP abmp, HWND awnd, HKL ahkl, HICON aicon, HFONT afont);
```

Description

This is UseHandle1, a member of class TWinHandleUser.

21.1.7.2.2 TWinHandleUser.UseHandle2 Method

Pascal

```
procedure UseHandle2(abrush: HBRUSH; amenu: HMENU; akey: HKEY; atask: HTASK; ainst: HINST;
ahook: HHOOK; agdiobj: HGDIOBJ; aAccel: HACCEL; aClrSpc: HCOLORSPACE; aglrc: HGLRC; acrsr:
HCURSOR; ahstr: HSTR; ahwinsta: HWINSTA); virtual;
```

C++

```
virtual __fastcall UseHandle2(HBRUSH abrush, HMENU amenu, HKEY akey, HTASK atask, HINST ainst,
HHOOK ahook, HGDIOBJ agdiobj, HACCEL aAccel, HCOLORSPACE aClrSpc, HGLRC aglrc, HCURSOR acrsr,
HSTR ahstr, HWINSTA ahwinsta);
```

Description

This is UseHandle2, a member of class TWinHandleUser.

21.2 Interfaces

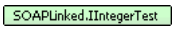
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IIntegerTest (see page 1332)	This is class SOAPLinked.IIntegerTest.
	IObjectGetter (see page 1333)	This is class SOAPLinked.IObjectGetter.
	IStringTest (see page 1334)	This is class SOAPLinked.IStringTest.

21.2.1 IIntegerTest Interface

Class Hierarchy



Pascal

```
IIntegerTest = interface;
```

C++

```
__interface IIntegerTest;
```

File

SOAPLinked

Description

This is class SOAPLinked.IIntegerTest.

21.2.1.1 IIntegerTest Members

The following tables list the members exposed by IIntegerTest.

Public Methods

	Name	Description
	Add (see page 1333)	This is Add, a member of class IIntegerTest.
	Doubler (see page 1333)	This is Doubler, a member of class IIntegerTest.

21.2.1.2 IIntegerTest Methods

The methods of the IIntegerTest class are listed here.

Public Methods

	Name	Description
	Add (see page 1333)	This is Add, a member of class IIntegerTest.
	Doubler (see page 1333)	This is Doubler, a member of class IIntegerTest.

21.2.1.2.1 IIntegerTest.Add Method

Pascal

```
function Add(const I1: Integer; const I2: Integer): Integer; stdcall;
```

C++

```
__fastcall __stdcall int Add(const int I1, const int I2);
```

Description

This is Add, a member of class IIntegerTest.

21.2.1.2.2 IIntegerTest.Doubler Method

Pascal

```
procedure Doubler(var I: Integer); cdecl;
```

C++

```
__cdecl __fastcall Doubler(int I);
```

Description

This is Doubler, a member of class IIntegerTest.

21.2.2 IObjectGetter Interface

Class Hierarchy

SOAPLinked.IObjectGetter

Pascal

```
IObjectGetter = interface;
```

C++

```
__interface IObjectGetter;
```

File

SOAPLinked

Description

This is class SOAPLinked.IObjectGetter.

21.2.2.1 IObjectGetter Members

The following tables list the members exposed by IObjectGetter.

Public Methods

	Name	Description
	GetAsObject (see page 1334)	This is GetAsObject, a member of class IObjectGetter.

21.2.2.2 IObjectGetter Methods

The methods of the IObjectGetter class are listed here.

Public Methods

	Name	Description
	GetAsObject (see page 1334)	This is GetAsObject, a member of class IObjectGetter.

21.2.2.2.1 IObjectGetter.GetAsObject Method

Pascal

```
function GetAsObject: TObject;
```

C++

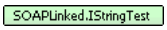
```
__fastcall TObject * GetAsObject();
```

Description

This is GetAsObject, a member of class IObjectGetter.

21.2.3 IStringTest Interface

Class Hierarchy



Pascal

```
IStringTest = interface;
```

C++

```
__interface IStringTest;
```

File

SOAPLinked

Description

This is class SOAPLinked.IStringTest.

21.2.3.1 IStringTest Members

The following tables list the members exposed by IStringTest.

Public Methods

	Name	Description
	Append (see page 1335)	This is Append, a member of class IStringTest.
	UpCase (see page 1335)	This is UpCase, a member of class IStringTest.

21.2.3.2 IStringTest Methods

The methods of the IStringTest class are listed here.

Public Methods

	Name	Description
	Append (see page 1335)	This is Append, a member of class IStringTest.
	UpCase (see page 1335)	This is UpCase, a member of class IStringTest.

21.2.3.2.1 IStringTest.Append Method

Pascal

```
function Append(const Str1: string; const Str2: string): string; cdecl;
```

C++

```
__cdecl __fastcall UnicodeString Append(const UnicodeString Str1, const UnicodeString Str2);
```

Description

This is Append, a member of class IStringTest.

21.2.3.2.2 IStringTest.UpCase Method

Pascal

```
procedure UpCase(var Str1: string); stdcall;
```

C++

```
__fastcall __stdcall UpCase(UnicodeString Str1);
```

Description

This is UpCase, a member of class IStringTest.

21.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	EventKind (see page 1336)	This is record SOAPLinked.EventKind.

21.3.1 SOAPLinked.EventKind Enumeration

Pascal

```
EventKind = (
  evNormal,
  evConst,
  evVar
);
```

C++

```
enum EventKind {
  evNormal,
  evConst,
  evVar
};
```

File

SOAPLinked

Description

This is record SOAPLinked.EventKind.

21.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCStringEvent (see page 1337)	This is type SOAPLinked.TCStringEvent.
TCustExCallback (see page 1337)	This is type SOAPLinked.TCustExCallback.
TStringEvent (see page 1338)	This is type SOAPLinked.TStringEvent.
TVStringEvent (see page 1338)	This is type SOAPLinked.TVStringEvent.

21.4.1 SOAPLinked.TCStringEvent Type

Pascal

```
TCStringEvent = procedure (Sender: TObject; const S: string) of object;
```

C++

```
(Sender: TObject; const S: string) ( TCStringEvent)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TCStringEvent.

21.4.2 SOAPLinked.TCustExCallback Type

Pascal

```
TCustExCallback = procedure;
```

C++

```
( TCustExCallback)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TCustExCallback.

21.4.3 SOAPLinked.TStringEvent Type

Pascal

```
TStringEvent = procedure (Sender: TObject; S: string) of object;
```

C++

```
(Sender: TObject; S: string) ( TStringEvent)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TStringEvent.

21.4.4 SOAPLinked.TVStringEvent Type

Pascal

```
TVStringEvent = procedure (Sender: TObject; var S: string) of object;
```

C++

```
(Sender: TObject; var S: string) ( TVStringEvent)();
```

File

SOAPLinked

Description

This is type SOAPLinked.TVStringEvent.

22 SOAPMidas Namespace

This is namespace SOAPMidas.

Topics

Name	Description
Interfaces (see page 1340)	The following table lists interfaces in this documentation.
Functions (see page 1347)	The following table lists functions in this documentation.

Functions

	Name	Description
	RegDefIAppServerInvClass (see page 1347)	Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

Interfaces

	Name	Description
	IAppServerSOAP (see page 1340)	IAppServerSOAP is the interface for communication with a SOAP data module.

22.1 Interfaces

The following table lists interfaces in this documentation.

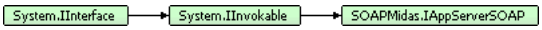
Interfaces

	Name	Description
	IAppServerSOAP (see page 1340)	IAppServerSOAP is the interface for communication with a SOAP data module.

22.1.1 IAppServerSOAP Interface

IAppServerSOAP is the interface for communication with a SOAP data module.

Class Hierarchy



Pascal

```
IAppServerSOAP = interface(IInvokable);
```

C++

```
__interface IAppServerSOAP : public IInvokable;
```

File

SOAPMidas

Description

TSoapConnection and TSoapDataModule use the IAppServerSOAP interface to communicate information in a SOAP-based multi-tiered database application. This is different from other DataSnap components, which use the IAppServer interface. IAppServerSOAP differs from IAppServer primarily in that it uses the stdcall calling convention rather than safecall. This makes it easier for marshalling interface calls when the client or server is written in C++.

Note: C++ method declarations that use IAppServerSOAP use the _di_IAppServerSOAP type instead. This type is a DelphiInterface (see page 3256) wrapper around the IAppServerSOAP interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IAppServerSOAP >  _di_IAppServerSOAP;
```

See Also

TSoapConnection (see page 1188)

TSoapDataModule (see page 1206)

IAppServer

22.1.1.1 IAppServerSOAP Members

The following tables list the members exposed by IAppServerSOAP.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SAS_ApplyUpdates (see page 1341)	Applies updates received from a client dataset using a specified provider.
	SAS_DataRequest (see page 1342)	Generates an OnSAS_DataRequest event on the specified provider component.
	SAS_Execute (see page 1342)	SAS_Executes the query or stored procedure bound to a specified provider.
	SAS_GetParams (see page 1343)	Fetches current parameter values from the dataset bound to a specified provider.
	SAS_GetProviderNames (see page 1344)	Returns a list of all the available providers on a remote data module.
	SAS_GetRecords (see page 1344)	Returns a data packet that contains the specified data.
	SAS_RowRequest (see page 1345)	Returns information from a specified record of the provider's dataset.

22.1.1.2 IAppServerSOAP Methods

The methods of the IAppServerSOAP class are listed here.

Public Methods

	Name	Description
	SAS_ApplyUpdates (see page 1341)	Applies updates received from a client dataset using a specified provider.
	SAS_DataRequest (see page 1342)	Generates an OnSAS_DataRequest event on the specified provider component.
	SAS_Execute (see page 1342)	SAS_Executes the query or stored procedure bound to a specified provider.
	SAS_GetParams (see page 1343)	Fetches current parameter values from the dataset bound to a specified provider.
	SAS_GetProviderNames (see page 1344)	Returns a list of all the available providers on a remote data module.
	SAS_GetRecords (see page 1344)	Returns a data packet that contains the specified data.
	SAS_RowRequest (see page 1345)	Returns information from a specified record of the provider's dataset.

22.1.1.2.1 IAppServerSOAP.SAS_ApplyUpdates Method

Applies updates received from a client dataset using a specified provider.

Pascal

```
function SAS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta,
int MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

SAS_SAS_ApplyUpdates writes client updates to a dataset or database server.

ProviderName specifies the name of the provider component that manages the update operation.

Delta is a Variant (see page 3421) containing the updated, inserted, and deleted records to write to the database.

MaxErrors indicates the maximum number of errors to permit before rolling back the update operation.

ErrorCount returns the actual number of errors encountered during the update operation.

OwnerData is information supplied by the client application in a BeforeSAS_ApplyUpdates event handler and returned to the client application in an AfterSAS_ApplyUpdates event handler. It is passed to the provider's BeforeSAS_ApplyUpdates event handler and returned from the provider's AfterSAS_ApplyUpdates event handler.

SAS_SAS_ApplyUpdates returns a Variant (see page 3421) that is a data packet containing all records that could not be

applied.

See Also

ApplyUpdates

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

SAS_ApplyUpdates (see page 1223)

22.1.1.2.2 IAppServerSOAP.SAS_DataRequest Method

Generates an OnSAS_DataRequest event on the specified provider component.

Pascal

```
function SAS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

Use SAS_SAS_DataRequest to invoke an OnSAS_DataRequest event handler of the provider named by ProviderName. This method is provided to allow application developers to customize the communication between a client application and a provider component.

ProviderName is the provider on which the data module generates an OnSAS_DataRequest event.

There is no predefined meaning for the Data parameter or the return value.

See Also

OnDataRequest

DataRequest

22.1.1.2.3 IAppServerSOAP.SAS_Execute Method

SAS_Executes the query or stored procedure bound to a specified provider.

Pascal

```
procedure SAS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); stdcall;
```

C++

```
__fastcall __stdcall SAS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant
Params, OleVariant OwnerData);
```

Description

Use SAS_SAS_Execute to pass an execute command to a remote provider, using a specified set of parameters. SAS_SAS_Execute calls the SAS_Execute method of the specified provider after assigning any parameters. Output (see page 3833) parameters that result from executing a query or stored procedure are returned in the Params parameter.

The provider's SAS_Execute method generates a BeforeSAS_Execute event, passes the command on to its associated dataset, and generates an AfterSAS_Execute event. The dataset must represent a query or stored procedure that does not return a cursor.

ProviderName specifies the name of the provider component associated with the query or stored procedure.

CommandText specifies an optional SQL statement that replaces the SQL of the query, or the name of a stored procedure that replaces the dataset's stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params encodes any parameters expected by the query or stored procedure and returns any output parameters. To create a properly encoded value for Params from a TParams object, use the PackageParams method. To decode Params into a TParams object, use the Unpackparams routine.

OwnerData contains custom information that appears as an argument to the provider's BeforeSAS_Execute and AfterSAS_Execute event handlers. This information originates in a client dataset's BeforeSAS_Execute event handler. The value returned in OwnerData is passed to the client dataset's AfterSAS_Execute event handler.

See Also

DataSet

BeforeExecute

AfterExecute

Params

Execute

Execute

BeforeExecute

AfterExecute

Options

PackageParams

UnpackParams

22.1.1.2.4 IAppServerSOAP.SAS_GetParams Method

Fetches current parameter values from the dataset bound to a specified provider.

Pascal

```
function SAS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

Use SAS_SAS_GetParams to retrieve the current parameter values from the dataset associated with a provider.

ProviderName specifies the name of the provider component whose dataset has the parameters.

OwnerData represents custom information that originates in a client dataset's BeforeSAS_GetParams event handler and returns information that is passed to the client dataset's AfterSAS_GetParams event handler.

SAS_SAS_GetParams returns the parameters, encoded as a Variant (see page 3421) array, or, if the provider does not

support parameter fetching, SAS_SAS_GetParams returns a Null Variant (see page 3421). Use the UnpackParams procedure to convert parameter values returned by SAS_SAS_GetParams to a TParams object.

Warning: Calling SAS_SAS_GetParams is not recommended because SOAP data modules are stateless, and any parameter values you fetch may be changed by other applications. The recommended way to obtain output parameter values is to call the SAS_Execute (see page 1342) method instead.

See Also

DataSet

Params

FetchParams

GetParams

BeforeGetParams

AfterGetParams

SAS_Execute (see page 1342)

UnpackParams

22.1.1.2.5 IAppServerSOAP.SAS_GetProviderNames Method

Returns a list of all the available providers on a remote data module.

Pascal

```
function SAS_GetProviderNames: TWideStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall TWideStringDynArray SAS_GetProviderNames();
```

Description

Call SAS_SAS_GetProviderNames for a list of all registered providers available from a SOAP data module. These values indicate the possible values for the ProviderName property of a client dataset that connects to the SOAP data module. When calling any of the SOAP data modules through the IAppServerSOAP (see page 1340) interface, pass one of these names to indicate the provider that is the target of the method call.

SAS_SAS_GetProviderNames returns the names of the providers in a dynamic array. Client datasets are associated with a provider by setting their ProviderName property to one of these names.

See Also

ProviderName

Providers (see page 1230)

22.1.1.2.6 IAppServerSOAP.SAS_GetRecords Method

Returns a data packet that contains the specified data.

Pascal

```
function SAS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_GetRecords(const BSTR ProviderName, int Count, int
RecsOut, int Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

Call SAS_SAS_GetRecords to retrieve a specified number of records. SAS_SAS_GetRecords returns the requested records, starting with the current record of the provider's dataset. Because SOAP data modules are stateless, you may need to reposition the cursor or re-execute a query or stored procedure in the provider's BeforeSAS_GetRecords event handler.

ProviderName gives the name of the provider component that supplies the records.

Count indicates the number or type of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only Count records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It is a combination of the GetRecordOption constants. For example, the following value requests metadata along with a data packet encoded in XML:

```
MetaDataOption + XMLOption
MetaDataOption || XMLOption
```

CommandText specifies an optional SQL statement that replaces the SQL of the provider's query, or the name of a table or stored procedure that replaces the provider's underlying database table or stored procedure. This parameter is ignored if the provider's Options property does not include poAllowCommandText.

Params is a Variant (see page 3421) containing any parameters that should be passed to the provider's dataset before it executes to generate the requested data. It returns any output parameters.

OwnerData contains custom information that is supplied by a client dataset's BeforeSAS_GetRecords event handler. This information is passed to the provider's BeforeSAS_GetRecords event handler. OwnerData returns information supplied by the provider's AfterSAS_GetRecords event handler.

Records are returned as a data packet in a Variant (see page 3421).

See Also

Data

Params

GetNextPacket

GetRecords

BeforeGetRecords

BeforeGetRecords

AfterGetRecords

Options

PackageParams

UnpackParams

22.1.1.2.7 IAppServerSOAP.SAS_RowRequest Method

Returns information from a specified record of the provider's dataset.

Pascal

```
function SAS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; stdcall;
```

C++

```
__fastcall __stdcall OleVariant SAS_RowRequest(const BSTR ProviderName, OleVariant Row, int
RequestType, OleVariant OwnerData);
```

Description

SAS_SAS_RowRequest provides the implementation for a client dataset's FetchBlobs, FetchDetails , or RefreshRecord method.

ProviderName indicates the provider associated with the dataset from which information should be fetched.

Row is an OleVariant (see page 3274) that describes the current record on the client dataset.

RequestType indicates the type of information required. It is an integer version of the TFetchOptions type. (An integer because the value may be marshaled to a remote application server). To create a value for RequestType, take the corresponding TFetchOptions value, cast it to a Byte, and cast the result to an Integer:

```
Integer(Byte([foBlobs, foDetails]));
(int) (Byte) (TFetchOptions() << foBlobs << foDetails);
```

OwnerData contains custom information that is supplied by a client dataset's BeforeSAS_RowRequest event handler. This information is passed to the provider's BeforeSAS_RowRequest event handler. OwnerData returns custom information supplied by the provider's AfterSAS_RowRequest event handler.

The requested data is returned as a delta packet.

See Also

RowRequest

BeforeRowRequest

AfterRowRequest

FetchBlobs

FetchDetails

RefreshRecord

22.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegDefIAppServerInvClass (see page 1347)	Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

22.2.1 SOAPMidas.RegDefIAppServerInvClass Function

Registers a SOAP data module for use by clients built using Kylix 2 or Delphi 5 prior to update patch 2.

Pascal

```
procedure RegDefIAppServerInvClass;
```

C++

```
RegDefIAppServerInvClass();
```

File

SOAPMidas

Description

RegDefIAppServerInvClass registers TSoapDataModule as an invokable class that implements IAppServer as a Web Service. C++ applications must add a call to this procedure after the registration of IAppServerSOAP (see page 1340) in the program startup code if they are to be used by clients written using Kylix 2 or Delphi 5 prior to update patch 2. This call is not necessary for other clients of a Web Service-based application server, because they look for the IAppServerSOAP (see page 1340) interface.

Note: Delphi applications do not need to call this procedure.

The following example shows where to place the call to RegDefIAppServerInvClass:

```
void static RegTypes()
{
  InvRegistry()->RegisterInterface(__interfaceTypeinfo(IMyAppServerService));
  InvRegistry()->RegisterInvokableClass(__classid(TMyAppServerService),
MyAppServerServiceFactory);
  RegDefIAppServerInvClass();
}
#pragma statrtup RegTypes 32
```

See Also

TSoapDataModule (see page 1206)

23 SOAPPasInv Namespace

This is namespace SOAPPasInv.

Topics

Name	Description
Classes (see page 1350)	The following table lists classes in this documentation.
Interfaces (see page 1359)	The following table lists interfaces in this documentation.
Functions (see page 1362)	The following table lists functions in this documentation.
Types (see page 1363)	The following table lists types in this documentation.

Classes

	Name	Description
	TSoapPascalInvoker (see page 1350)	TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

Functions

	Name	Description
	GetSOAPHeaders (see page 1362)	Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

Interfaces

	Name	Description
	IAdapterConverter (see page 1359)	This is class SOAPPasInv.IAdapterConverter.
	IInvokerConverter (see page 1360)	This is class SOAPPasInv.IInvokerConverter.

Types

Name	Description
TAfterDispatchEvent (see page 1363)	TAfterDispatchEvent is the type of the AfterDispatchEvent event on an invoker component.
TBeforeDispatchEvent (see page 1363)	TBeforeDispatchEvent is the type of the BeforeDispatchEvent event on an invoker component.
TBeforeDispatchEvent2 (see page 1364)	TBeforeDispatchEvent2 is the type of the BeforeDispatchEvent2 event on an invoker component.
TOnExceptionEvent (see page 1364)	TOnExceptionEvent is the type of the OnExceptionEvent event on an invoker component.

23.1 Classes

The following table lists classes in this documentation.

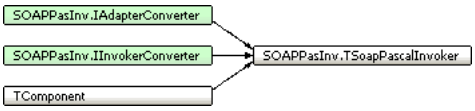
Classes

	Name	Description
	TSoapPascalInvoker (see page 1350)	TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

23.1.1 TSoapPascalInvoker Class

TSoapPascalInvoker is the base class for components that interpret SOAP requests and execute the corresponding invokable interface.

Class Hierarchy



Pascal

```
TSoapPascalInvoker = class(TComponent, IAdapterConverter, IInvokerConverter);
```

C++

```
class TSoapPascalInvoker : public TComponent, public IAdapterConverter, public
IInvokerConverter;
```

File

SOAPPasInv

Description

Use TSoapPascalInvoker as a base class for components that interpret SOAP requests. TSoapPascalInvoker implements the logic that parses a SOAP request, given its target invokable interface, executes the interface call, and encodes the results of that code.

Each TSoapPascalInvoker descendant receives SOAP requests that arrive via a specific transport protocol. The TSoapPascalInvoker descendant must implement the logic to take the incoming SOAP request, identify the target invokable interface, and look up its implementation class and runtime type information (RTTI).

See Also

IInvokable (see page 3504)

THTTPSoapPascalInvoker (see page 1267)

23.1.1.1 TSoapPascalInvoker Members

The following tables list the members exposed by TSoapPascalInvoker.

Protected Fields

	Name	Description
	FConverter (see page 1352)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 1352)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 1353)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 1353)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 1353)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

Protected Methods

	Name	Description
	GetDomConverter (see page 1354)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetInvoker (see page 1354)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	Notification (see page 1355)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
V	Create (see page 1351)	Creates a new TSoapPascalInvoker (see page 1350) object.
V	Destroy (see page 1352)	Frees the memory associated with the TSoapPascalInvoker (see page 1350) object.
V	Invoke (see page 1354)	Interprets a SOAP request message, calls the target interface, and encodes any return values.

Published Events

	Name	Description
	AfterDispatchEvent (see page 1356)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 1357)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 1357)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 1357)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 1358)	Occurs when TSoapPascalInvoker (see page 1350) reads each attachment included in an incoming request.
	OnSendAttachment (see page 1358)	Occurs when TSoapPascalInvoker (see page 1350) sends each attachment included in the response to the current request.

Published Properties

	Name	Description
	Converter (see page 1355)	Specifies the interface of the component that handles the marshaling of SOAP messages.

23.1.1.2 TSoapPascalInvoker.Create Constructor

Creates a new TSoapPascalInvoker (see page 1350) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSoapPascalInvoker(TComponent * AOwner);
```

Description

Call Create to instantiate an invoker at runtime. Invoker components added to Web modules at design time (including the one that the Soap Server wizard adds automatically) are created automatically.

The AOwner parameter specifies the component that is responsible for freeing the TSoapPascalInvoker (see page 1350) instance.

See Also

Owner

23.1.1.3 TSoapPascalInvoker.Destroy Destructor

Frees the memory associated with the TSoapPascalInvoker (see page 1350) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSoapPascalInvoker();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the invoker component is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

23.1.1.4 TSoapPascalInvoker Fields

The fields of the TSoapPascalInvoker class are listed here.

Protected Fields

	Name	Description
	FConverter (see page 1352)	This is FConverter, a member of class TSoapPascalInvoker.
	FDomConverter (see page 1352)	This is FDomConverter, a member of class TSoapPascalInvoker.
	FHeadersIn (see page 1353)	This is FHeadersIn, a member of class TSoapPascalInvoker.
	FHeadersOut (see page 1353)	This is FHeadersOut, a member of class TSoapPascalInvoker.
	FSOAPHeaders (see page 1353)	This is FSOAPHeaders, a member of class TSoapPascalInvoker.

23.1.1.4.1 TSoapPascalInvoker.FConverter Field

Pascal

```
FConverter: IOPConvert;
```

C++

```
IOPConvert FConverter;
```

Description

This is FConverter, a member of class TSoapPascalInvoker.

23.1.1.4.2 TSoapPascalInvoker.FDomConverter Field

Pascal

```
FDomConverter: TOPToSoapDomConvert;
```

C++

```
TOPToSoapDomConvert FDomConverter;
```

Description

This is FDomConverter, a member of class TSoapPascalInvoker.

23.1.1.4.3 TSoapPascalInvoker.FHeadersIn Field

Pascal

```
FHeadersIn: THeaderList;
```

C++

```
THeaderList FHeadersIn;
```

Description

This is FHeadersIn, a member of class TSoapPascalInvoker.

23.1.1.4.4 TSoapPascalInvoker.FHeadersOut Field

Pascal

```
FHeadersOut: THeaderList;
```

C++

```
THeaderList FHeadersOut;
```

Description

This is FHeadersOut, a member of class TSoapPascalInvoker.

23.1.1.4.5 TSoapPascalInvoker.FSOAPHeaders Field

Pascal

```
FSOAPHeaders: TSOAPHeaders;
```

C++

```
TSOAPHeaders FSOAPHeaders;
```

Description

This is FSOAPHeaders, a member of class TSoapPascalInvoker.

23.1.1.5 TSoapPascalInvoker Methods

The methods of the TSoapPascalInvoker class are listed here.

Protected Methods

	Name	Description
	GetDomConverter (see page 1354)	This is GetDomConverter, a member of class TSoapPascalInvoker.
	GetInvoker (see page 1354)	This is GetInvoker, a member of class TSoapPascalInvoker.
V	Notification (see page 1355)	Forwards notification messages to all owned components.

Public Methods

	Name	Description
	Invoke (see page 1354)	Interprets a SOAP request message, calls the target interface, and encodes any return values.

23.1.1.5.1 TSoapPascalInvoker.GetDomConverter Method

Pascal

```
function GetDomConverter: TOPToSoapDomConvert;
```

C++

```
__fastcall TOPToSoapDomConvert GetDomConverter();
```

Description

This is GetDomConverter, a member of class TSoapPascalInvoker.

23.1.1.5.2 TSoapPascalInvoker.GetInvoker Method

Pascal

```
function GetInvoker: TSoapPascalInvoker;
```

C++

```
__fastcall TSoapPascalInvoker GetInvoker();
```

Description

This is GetInvoker, a member of class TSoapPascalInvoker.

23.1.1.5.3 TSoapPascalInvoker.Invoke Method

Interprets a SOAP request message, calls the target interface, and encodes any return values.

Pascal

```
procedure Invoke(AClass: TClass; IntfInfo: PTypeInfo; MethName: string; const Request:
TStream; Response: TStream; var BindingType: TWebServiceBindingType); virtual;
```

C++

```
virtual __fastcall Invoke(TClass AClass, PTypeInfo IntfInfo, UnicodeString MethName, const
TStream Request, TStream Response, TWebServiceBindingType BindingType);
```

Description

Call Invoke to convert a SOAP request to an interface call on an invokable interface.

AClass is the class that implements the invokable interface. The invoker passes this to the invocation registry so that it can obtain an instance of the implementation class, which it uses to execute the SOAP request.

IntfInfo is the runtime type information (RTTI) from the invokable interface that is the target of the SOAP request. The invoker uses this to interpret the SOAP request.

MethName is the name of the method to call on the invokable interface.

Request is a stream from which Invoke can read the SOAP request. It contains the SOAP encoding of an interface call.

Response is a stream to which Invoke writes the SOAP encoding of any return values.

BindingType indicates the format of the Request and Response. It is either btSOAP (to indicate a SOAP packet) or btMime (for a

multipart form when the request or response includes attachments). On entry, BindingType indicates the format of the encoded request message. Invoke changes BindingType to the format of the response message.

See Also

IInvokable (see page 3504)

InvRegistry

GetInvokableObjectFromClass

23.1.1.5.4 TSoapPascalInvoker.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

23.1.1.6 TSoapPascalInvoker Properties

The properties of the TSoapPascalInvoker class are listed here.

Published Properties

	Name	Description
	Converter (see page 1355)	Specifies the interface of the component that handles the marshaling of SOAP messages.

23.1.1.6.1 TSoapPascalInvoker.Converter Property

Specifies the interface of the component that handles the marshaling of SOAP messages.

Pascal

```
property Converter: TOPToSoapDomConvert;
```

C++

```
__property TOPToSoapDomConvert Converter;
```

Description

Converter is the component that marshals and unmarshals method calls for the invoker. The invoker uses this object to interpret incoming method calls and to generate the results it returns.

See Also

TOPToSoapDomConvert

23.1.1.7 TSoapPascalInvoker Events

The events of the TSoapPascalInvoker class are listed here.

Published Events

	Name	Description
	AfterDispatchEvent (see page 1356)	Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent (see page 1357)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	BeforeDispatchEvent2 (see page 1357)	Occurs before the invoker calls an invokable interface to execute a SOAP request.
	OnExceptionEvent (see page 1357)	Occurs when the invoker encounters an exception while executing a SOAP request.
	OnGetAttachment (see page 1358)	Occurs when TSoapPascalInvoker (see page 1350) reads each attachment included in an incoming request.
	OnSendAttachment (see page 1358)	Occurs when TSoapPascalInvoker (see page 1350) sends each attachment included in the response to the current request.

23.1.1.7.1 TSoapPascalInvoker.AfterDispatchEvent Event

Occurs after the invoker successfully calls an invokable interface to execute a SOAP request.

Pascal

```
property AfterDispatchEvent: TAfterDispatchEvent;
```

C++

```
__property TAfterDispatchEvent AfterDispatchEvent;
```

Description

Write (see page 3707) a AfterDispatchEvent event handler to examine a SOAP response after the invoker executes a request by calling the target invokable interface. The Invoke (see page 1354) method generates this event after it has executed a requested method and generated a response packet.

Note: AfterDispatchEvent only occurs if the method call is successful. If the invoker encounters an exception when trying to execute the specified method, it generates an OnExceptionEvent (see page 1357) instead.

See Also

BeforeDispatchEvent (see page 1357)

Invoke (see page 1354)

OnExceptionEvent (see page 1357)

23.1.1.7.2 TSoapPascalInvoker.BeforeDispatchEvent Event

Occurs before the invoker calls an invokable interface to execute a SOAP request.

Pascal

```
property BeforeDispatchEvent: TBeforeDispatchEvent;
```

C++

```
__property TBeforeDispatchEvent BeforeDispatchEvent;
```

Description

Write (see page 3707) a BeforeDispatchEvent event handler to examine a SOAP request before the invoker calls on the target invokable interface to execute it. The Invoke (see page 1354) method generates this event after it has obtained an interface pointer for the target invokable interface, but before it calls the specified method on that interface.

See Also

AfterDispatchEvent (see page 1356)

Invoke (see page 1354)

BeforeDispatchEvent2 (see page 1357)

23.1.1.7.3 TSoapPascalInvoker.BeforeDispatchEvent2 Event

Occurs before the invoker calls an invokable interface to execute a SOAP request.

Pascal

```
property BeforeDispatchEvent2: TBeforeDispatchEvent2;
```

C++

```
__property TBeforeDispatchEvent2 BeforeDispatchEvent2;
```

Description

Write (see page 3707) a BeforeDispatchEvent2 event handler to override the default handling of an incoming request. The Invoke (see page 1354) method generates this event after it has obtained an interface pointer for the target invokable interface, but before it calls the specified method on that interface.

See Also

AfterDispatchEvent (see page 1356)

Invoke (see page 1354)

BeforeDispatchEvent (see page 1357)

23.1.1.7.4 TSoapPascalInvoker.OnExceptionEvent Event

Occurs when the invoker encounters an exception while executing a SOAP request.

Pascal

```
property OnExceptionEvent: TOnExceptionEvent;
```

C++

```
__property TOnExceptionEvent OnExceptionEvent;
```

Description

Write (see page 3707) a OnExceptionEvent event handler to examine a SOAP fault packet after the invoker encounters an exception when executing a SOAP request. The Invoke (see page 1354) method generates this event if it encounters an exception, after it generates the SOAP fault packet that describes the problem.

See Also

AfterDispatchEvent (see page 1356)

23.1.1.7.5 TSoapPascalInvoker.OnGetAttachment Event

Occurs when TSoapPascalInvoker (see page 1350) reads each attachment included in an incoming request.

Pascal

```
property OnGetAttachment: TOnGetAttachmentEvent;
```

C++

```
__property TOnGetAttachmentEvent OnGetAttachment;
```

Description

Write (see page 3707) an OnGetAttachment event handler to examine the encoded attachments included with an incoming request or to redirect them so that they are saved to a specific file. OnGetAttachment occurs once for each attachment received.

Tip: Use the FileName property of the attachment object to redirect the attachment to a different file.

See Also

OnSendAttachment (see page 1358)

BeforeDispatchEvent (see page 1357)

23.1.1.7.6 TSoapPascalInvoker.OnSendAttachment Event

Occurs when TSoapPascalInvoker (see page 1350) sends each attachment included in the response to the current request.

Pascal

```
property OnSendAttachment: TOnSendAttachmentEvent;
```

C++

```
__property TOnSendAttachmentEvent OnSendAttachment;
```

Description

Write (see page 3707) an OnSendAttachment event handler to examine or alter the attachments that are to be included in a message that represents the results of a method call. OnSendAttachment occurs once for each attachment sent with the response message.

See Also

AfterDispatchEvent (see page 1356)

OnGetAttachment (see page 1358)

23.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterConverter (see page 1359)	This is class SOAPPasInv.IAdapterConverter.
	IInvokerConverter (see page 1360)	This is class SOAPPasInv.IInvokerConverter.

23.2.1 IAdapterConverter Interface

Class Hierarchy

SOAPPasInv.IAdapterConverter

Pascal

```
IAdapterConverter = interface;
```

C++

```
__interface IAdapterConverter;
```

File

SOAPPasInv

Description

This is class SOAPPasInv.IAdapterConverter.

23.2.1.1 IAdapterConverter Members

The following tables list the members exposed by IAdapterConverter.

Public Methods

	Name	Description
	GetDomConverter (see page 1360)	This is GetDomConverter, a member of class IAdapterConverter.

23.2.1.2 IAdapterConverter Methods

The methods of the IAdapterConverter class are listed here.

Public Methods

	Name	Description
	GetDomConverter (see page 1360)	This is GetDomConverter, a member of class IAdapterConverter.

23.2.1.2.1 IAdapterConverter.GetDomConverter Method

Pascal

```
function GetDomConverter: TOPToSoapDomConvert;
```

C++

```
__fastcall TOPToSoapDomConvert GetDomConverter();
```

Description

This is GetDomConverter, a member of class IAdapterConverter.

23.2.2 IInvokerConverter Interface

Class Hierarchy

SOAPPasInv.IInvokerConverter

Pascal

```
IInvokerConverter = interface;
```

C++

```
__interface IInvokerConverter;
```

File

SOAPPasInv

Description

This is class SOAPPasInv.IInvokerConverter.

23.2.2.1 IInvokerConverter Members

The following tables list the members exposed by IInvokerConverter.

Public Methods

	Name	Description
	GetInvoker (see page 1361)	This is GetInvoker, a member of class IInvokerConverter.

23.2.2.2 IInvokerConverter Methods

The methods of the IInvokerConverter class are listed here.

Public Methods

	Name	Description
	GetInvoker (see page 1361)	This is GetInvoker, a member of class IInvokerConverter.

23.2.2.2.1 IInvokerConverter.GetInvoker Method

Pascal

```
function GetInvoker: TSoapPascalInvoker;
```

C++

```
__fastcall TSoapPascalInvoker GetInvoker();
```

Description

This is GetInvoker, a member of class IInvokerConverter.

23.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetSOAPHeaders (see page 1362)	Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

23.3.1 SOAPPasInv.GetSOAPHeaders Function

Returns the interface for accessing the SOAP headers that were received with a SOAP request and supplying the headers for the outgoing response.

Pascal

```
function GetSOAPHeaders: ISOAPHeaders;
```

C++

```
ISOAPHeaders GetSOAPHeaders();
```

File

SOAPPasInv

Description

Call GetSOAPHeaders from an application that implements a Web Service when you do not use TInvokableClass as a base class for the implementation of an invokable interface. GetSOAPHeaders returns an ISOAPHeaders interface, which lets your application examine the SOAP headers in an incoming request message and to add SOAP headers to be sent with the outgoing response.

See Also

TInvokableClass

Defining and Using SOAP Headers

23.4 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterDispatchEvent (see page 1363)	TAfterDispatchEvent is the type of the AfterDispatchEvent event on an invoker component.
TBeforeDispatchEvent (see page 1363)	TBeforeDispatchEvent is the type of the BeforeDispatchEvent event on an invoker component.
TBeforeDispatchEvent2 (see page 1364)	TBeforeDispatchEvent2 is the type of the BeforeDispatchEvent2 event on an invoker component.
TOnExceptionEvent (see page 1364)	TOnExceptionEvent is the type of the OnExceptionEvent event on an invoker component.

23.4.1 SOAPPasInv.TAfterDispatchEvent Type

TAfterDispatchEvent is the type of the AfterDispatchEvent event on an invoker component.

Pascal

```
TAfterDispatchEvent = procedure (const MethodName: string; SOAPResponse: TStream) of object;
```

C++

```
(const MethodName: string; SOAPResponse: TStream) ( TAfterDispatchEvent)();
```

File

SOAPPasInv

Description

TAfterDispatchEvent is the type of the AfterDispatchEvent event handler, which is called after an invoker executes a request by calling the target invokable interface.

MethodName is the name of the method that was just called.

SOAPResponse is a stream from which the event handler can read or alter the response packet that represents the results of the method call.

See Also

Procedural types

Creating events

23.4.2 SOAPPasInv.TBeforeDispatchEvent Type

TBeforeDispatchEvent is the type of the BeforeDispatchEvent event on an invoker component.

Pascal

```
TBeforeDispatchEvent = procedure (const MethodName: string; const Request: TStream) of object;
```

C++

```
(const MethodName: string; const Request: TStream) ( TBeforeDispatchEvent)();
```

File

SOAPPasInv

Description

TBeforeDispatchEvent is the type of the BeforeDispatchEvent event handler, which is called before an invoker executes a request by calling the target invokable interface.

MethodName is the name of the method that is about to be called.

Request is a stream from which the event handler can read the SOAP request that is about to be executed.

See Also

Procedural types

Creating events

23.4.3 SOAPPasInv.TBeforeDispatchEvent2 Type

TBeforeDispatchEvent2 is the type of the BeforeDispatchEvent2 event on an invoker component.

Pascal

```
TBeforeDispatchEvent2 = procedure (const MethodName: string; const Request: TStream; Response:
TStream; var BindingType: TWebServiceBindingType; var Handled: Boolean) of object;
```

C++

```
(const MethodName: string; const Request: TStream; Response: TStream; var BindingType:
TWebServiceBindingType; var Handled: Boolean) ( TBeforeDispatchEvent2)();
```

File

SOAPPasInv

Description

TBeforeDispatchEvent2 is the type of the BeforeDispatchEvent2 event handler, which is called before an invoker executes a request by calling the target invokable interface.

MethodName is the name of the method that is about to be called.

Request is a stream from which the event handler can read the SOAP request that is about to be executed.

Response is a stream to which the event handler can write a response for the request.

BindingType indicates the encoding that is used by the request.

Handled returns true if the event handler supplies a response using the Response parameter. It returns false if the invoker should forward the request to a registered implementation class after the event handler exits.

23.4.4 SOAPPasInv.TOnExceptionEvent Type

TOnExceptionEvent is the type of the OnExceptionEvent event on an invoker component.

Pascal

```
TOnExceptionEvent = procedure (const MethodName: string; const Request: TStream; const
Response: TStream) of object;
```

C++

```
(const MethodName: string; const Request: TStream; const Response: TStream) (
TOnExceptionEvent)();
```

File

SOAPPasInv

Description

TOnExceptionEvent is the type of the OnExceptionEventEvent event handler, which is called when an invoker encounters an exception while executing a request by calling a target invokable interface.

MethodName is the name of the method that was just called.

Request is a stream from which the event handler can read the SOAP request that contained the method call.

Response is a stream from which the event handler can read the SOAP fault packet that was created to describe the exception.

See Also

Procedural types

Creating events

24 Sockets Namespace

This is namespace Sockets.

Topics

Name	Description
Classes (see page 1369)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 1499)	The following table lists structs, records, enums in this documentation.
Types (see page 1504)	The following table lists types in this documentation.
Constants (see page 1508)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESocketError (see page 1369)	Exception (see page 3949) class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TBaseSocket (see page 1370)	TBaseSocket encapsulates local operations on file descriptors.
	TClientSocketThread (see page 1396)	Enables creation of separate threads of execution for the client socket in an application.
	TCustomIpClient (see page 1400)	Encapsulates the client network functions.
	TCustomTcpServer (see page 1406)	Ancestor object of all the server socket descendants.
	TIpSocket (see page 1416)	Encapsulates the network support functions.
	TRawSocket (see page 1431)	Encapsulates a raw socket.
	TServerSocketThread (see page 1433)	Enables creation of separate threads of execution for the server socket in an application.
	TTcpClient (see page 1439)	TCP client component.
	TTcpServer (see page 1459)	TTcpServer is the TCP server component.
	TUdpSocket (see page 1477)	UDP component.

Constants

Name	Description
CRLF (see page 1508)	This is constant Sockets.CRLF.

Structs, Records, Enums

	Name	Description
	TIPHeader (see page 1499)	This is record Sockets.TIPHeader.
	TServerSocketBlockMode (see page 1500)	Defines the set of possible socket flags for block mode.
	TSocketDomain (see page 1500)	Set of possible protocol family types used by a socket.
	TSocketType (see page 1502)	The TSocketType type defines the set of possible socket types.

Types

Name	Description
TGetThreadEvent (see page 1504)	This is type Sockets.TGetThreadEvent.
TSocketAcceptEvent (see page 1504)	This is type Sockets.TSocketAcceptEvent.
TSocketBlockMode (see page 1505)	Defines the set of possible flags for blocking mode of the socket.
TSocketDataEvent (see page 1505)	This is type Sockets.TSocketDataEvent.
TSocketErrorEvent (see page 1505)	This is type Sockets.TSocketErrorEvent.
TSocketHost (see page 1506)	This is type Sockets.TSocketHost.
TSocketNotifyEvent (see page 1506)	This is type Sockets.TSocketNotifyEvent.
TSocketPort (see page 1506)	This is type Sockets.TSocketPort.
TSocketProtocol (see page 1507)	TSocketProtocol is the code for a socket protocol.

24.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESocketError (see page 1369)	Exception (see page 3949) class for failures that occur when trying to create, use, or shut down Windows socket objects.
	TBaseSocket (see page 1370)	TBaseSocket encapsulates local operations on file descriptors.
	TClientSocketThread (see page 1396)	Enables creation of separate threads of execution for the client socket in an application.
	TCustomIpClient (see page 1400)	Encapsulates the client network functions.
	TCustomTcpServer (see page 1406)	Ancestor object of all the server socket descendants.
	TIpSocket (see page 1416)	Encapsulates the network support functions.
	TRawSocket (see page 1431)	Encapsulates a raw socket.
	TServerSocketThread (see page 1433)	Enables creation of separate threads of execution for the server socket in an application.
	TTcpClient (see page 1439)	TCP client component.
	TTcpServer (see page 1459)	TTcpServer is the TCP server component.
	TUdpSocket (see page 1477)	UDP component.

24.1.1 ESocketError Class

Exception (see page 3949) class for failures that occur when trying to create, use, or shut down Windows socket objects.

Class Hierarchy

Pascal

```
ESocketError = class(Exception);
```

C++

```
class ESocketError : public Exception;
```

File

Sockets

Description

ESocketError is the exception class for failures that occur when trying to create, use, or shut down Windows socket objects.

ESocketError is thrown if:

A socket cannot be created.

A socket cannot be initialized.

A socket cannot be opened.

An application tries to change the properties of an already open socket.

A failure occurs when reading from or writing to a socket.

A socket cannot be properly closed.

There are two ESocketError classes defined, one in the ScktComp (see page 11) unit and one in the Sockets (see page 1367) unit. The two exception classes are identically defined and serve the same function for different categories of socket objects. The two definitions occur simply to ensure that ESocketError is defined for all socket objects.

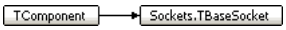
See Also

TCustomWinSocket (see page 64)

24.1.2 TBaseSocket Class

TBaseSocket encapsulates local operations on file descriptors.

Class Hierarchy



Pascal

```
TBaseSocket = class(TComponent);
```

C++

```
class TBaseSocket : public TComponent;
```

File

Sockets

Description

The TBaseSocket component encapsulates local operations on files descriptors, which are objects that a program uses to read from or write to an open file, or open a network socket or a variety of other devices.

Do not create instances of TBaseSocket. Use TBaseSocket as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TBaseSocket provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TIpSocket (see page 1416)

24.1.2.1 TBaseSocket Members

The following tables list the members exposed by TBaseSocket.

Protected Methods

	Name	Description
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnCreateHandle (see page 1388)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1390)	Occurs when a socket is deactivated.
	OnError (see page 1393)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1393)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1394)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
V	Close (see page 1373)	Shuts down the socket connection.
V	Create (see page 1371)	Creates an instance of TBaseSocket (see page 1370).
V	Destroy (see page 1372)	Frees the memory associated with the TBaseSocket (see page 1370) object.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
V	Open (see page 1375)	Creates the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveIn (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1383)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1385)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	SockType (see page 1387)	Sets or returns the socket type.

24.1.2.2 TBaseSocket.Create Constructor

Creates an instance of TBaseSocket (see page 1370).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBaseSocket(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TBaseSocket (see page 1370) object.

See Also

TBaseSocket (see page 1370)

24.1.2.3 TBaseSocket.Destroy Destructor

Frees the memory associated with the TBaseSocket (see page 1370) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseSocket();
```

Description

For C++, do not call ~TBaseSocket (see page 1370) directly. Instead, use the **delete** keyword on the object, which causes ~TBaseSocket (see page 1370) to be invoked automatically.

See Also

delete

TBaseSocket (see page 1370)

24.1.2.4 TBaseSocket Methods

The methods of the TBaseSocket class are listed here.

Protected Methods

	Name	Description
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Methods

	Name	Description
	Close (see page 1373)	Shuts down the socket connection.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
	Open (see page 1375)	Creates the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveIn (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

24.1.2.4.1 TBaseSocket.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; virtual;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 1383) property to **false**.

For client sockets, Close terminates the connection to the server. For server sockets, Close shuts down the connection so that the server socket is no longer listening for client requests.

See Also

Active (see page 1383)

Open (see page 1375)

24.1.2.4.2 TBaseSocket.DoCreateHandle Method

Pascal

```
procedure DoCreateHandle; dynamic;
```

C++

```
__fastcall DoCreateHandle();
```

Description

This is DoCreateHandle, a member of class TBaseSocket.

24.1.2.4.3 TBaseSocket.DoDestroyHandle Method

Pascal

```
procedure DoDestroyHandle; dynamic;
```

C++

```
__fastcall DoDestroyHandle();
```

Description

This is DoDestroyHandle, a member of class TBaseSocket.

24.1.2.4.4 TBaseSocket.DoHandleError Method

Pascal

```
procedure DoHandleError; dynamic;
```

C++

```
__fastcall DoHandleError();
```

Description

This is DoHandleError, a member of class TBaseSocket.

24.1.2.4.5 TBaseSocket.DoReceive Method

Pascal

```
procedure DoReceive(Buf: pchar; var DataLen: Integer); virtual;
```

C++

```
virtual __fastcall DoReceive(const char * Buf, int DataLen);
```

Description

This is DoReceive, a member of class TBaseSocket.

24.1.2.4.6 TBaseSocket.DoSend Method

Pascal

```
procedure DoSend(Buf: pchar; var DataLen: Integer); virtual;
```

C++

```
virtual __fastcall DoSend(const char * Buf, int DataLen);
```

Description

This is DoSend, a member of class TBaseSocket.

24.1.2.4.7 TBaseSocket.ErrorCheck Method

Pascal

```
function ErrorCheck(rc: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ErrorCheck(int rc);
```

Description

This is ErrorCheck, a member of class TBaseSocket.

24.1.2.4.8 TBaseSocket.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling

pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

24.1.2.4.9 TBaseSocket.MapDomain Method

Maps the domain.

Pascal

```
function MapDomain(sd: TSocketDomain): Integer;
```

C++

```
__fastcall int MapDomain(TSocketDomain sd);
```

Description

MapDomain maps the socket domain to an integer based on the value supplied by the Domain (see page 1386) property.

See Also

Domain (see page 1386)

MapSockType (see page 1375)

24.1.2.4.10 TBaseSocket.MapSockType Method

Maps the socket type.

Pascal

```
function MapSockType(st: TSocketType): Integer;
```

C++

```
__fastcall int MapSockType(TSocketType st);
```

Description

MapSockType maps the socket type to an integer based on the value supplied by the SockType (see page 1387) property.

See Also

SockType (see page 1387)

24.1.2.4.11 TBaseSocket.Open Method

Creates the socket connection.

Pascal

```
procedure Open; virtual;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to create the socket connection. Descendants of TBaseSocket (see page 1370) use Open to initiate the socket connection.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

TBaseSocket (see page 1370)

Active (see page 1383)

Close (see page 1373)

24.1.2.4.12 TBaseSocket.PeekBuf Method

Reads the buffer.

Pascal

```
function PeekBuf(var Buf; BufSize: Integer): Integer;
```

C++

```
__fastcall int PeekBuf( Buf, int BufSize);
```

Description

Use PeekBuf to make a copy of data waiting in the socket buffer. Use the **Buf** parameter to assign a variable to the buffer, and use **Bufsize** to set the size of the buffer to be copied. PeekBuf only copies, but does not otherwise affect the contents of the buffer.

See Also

ReceiveBuf (see page 1376)

SendBuf (see page 1380)

24.1.2.4.13 TBaseSocket.ReceiveBuf Method

Reads data buffers to the socket.

Pascal

```
function ReceiveBuf(var Buf; BufSize: Integer; Flags: Integer = 0): Integer;
```

C++

```
__fastcall int ReceiveBuf( Buf, int BufSize, int Flags = 0);
```

Description

ReceiveBuf reads data buffers of size **bufsize** to the socket. Use **buf** to assign a variable to the buffer, and use **bufsize** to set the size of the buffer to be read. The **flags** argument can be any of the flags used by the Linux **recv** function. Refer to recv(2) in the Linux manual pages for more information.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
```

```
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;
```

```
procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

PeekBuf (see page 1376)

SendBuf (see page 1380)

24.1.2.4.14 TBaseSocket.ReceiveIn Method

Reads delimited lines of data from the socket.

Pascal

```
function Receiveln(const eol: string = CRLF): string;
```

C++

```
__fastcall UnicodeString Receiveln(const UnicodeString eol = CRLF);
```

Description

Receiveln reads a delimited chunk of data from the socket. The **delimeter** cannot be part of the character set. Receiveln fills the buffer until it reaches the **delimeter**.

See Also

Sendln (see page 1380)

OnReceive (see page 1393)

24.1.2.4.15 TBaseSocket.Select Method

Tests for socket activity.

Pascal

```
function Select(ReadReady: PBoolean; WriteReady: PBoolean; ExceptFlag: PBoolean; TimeOut:
Integer = 0): Boolean;
```

C++

```
__fastcall Boolean Select(PBoolean ReadReady, PBoolean WriteReady, PBoolean ExceptFlag, int
TimeOut = 0);
```

Description

Use Select to test for socket activity. Select indicates whether there is any data on the socket waiting to be sent or whether an error condition exists.

The parameter **TimeOut** is the time-out value. The socket waits for activity up to **TimeOut** milliseconds before returning. If **TimeOut** is not explicitly specified, the default value is zero. If **TimeOut** is any negative number, the socket is blocked. It waits for activity indefinitely, which can be useful for servers.

See Also

WaitForData (see page 1383)

24.1.2.4.16 TBaseSocket.SendBuf Method

Writes data buffers of size bufsize from the socket.

Pascal

```
function SendBuf(var Buf; BufSize: Integer; Flags: Integer = 0): Integer;
```

C++

```
__fastcall int SendBuf( Buf, int BufSize, int Flags = 0);
```

Description

SendBuf sends data buffers of size **bufsize** to the socket. Use **buf** to assign a variable to the buffer, and use **bufsize** to set the size of the buffer to be sent. The flags argument can be any of the flags used by the Linux **send** function. Refer to the send Linux manual page for more information.

See Also

PeekBuf (see page 1376)

ReceiveBuf (see page 1376)

24.1.2.4.17 TBaseSocket.SendIn Method

Writes delimited lines of data to the socket.

Pascal

```
function Sendln(s: string; const eol: string = CRLF): Integer;
```

C++

```
__fastcall int Sendln(UnicodeString s, const UnicodeString eol = CRLF);
```

Description

Sendln sends a delimited chunk of data to the socket. The **delimeter** cannot be part of the character set. Sendln fills the buffer until it reaches the **delimeter**.

See Also

Receiveln (see page 1379)

OnReceive (see page 1393)

24.1.2.4.18 TBaseSocket.SendStream Method

Writes a stream of data to the socket.

Pascal

```
function SendStream(AStream: TStream): Integer;
```

C++

```
__fastcall int SendStream(TStream AStream);
```

Description

SendStream sends a stream of data to the socket.

C++ Examples:

```
/*
```

```
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
```

```
    myStream := TMemoryStream.Create();
    for I := 0 to 8 do
    begin
      myStream.WriteBuffer(I,1);
    end;
    userValue := StrToIntDef(Edit1.Text, 0);
    myStream.WriteBuffer(userValue,1);

    //resetting the stream position
    myStream.Seek(0,0);

    //sending the stream
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

OnReceive (see page 1393)

24.1.2.4.19 TBaseSocket.SetBytesReceived Method

Pascal

```
procedure SetBytesReceived(Value: Cardinal);
```

C++

```
__fastcall SetBytesReceived(unsigned Value);
```

Description

This is SetBytesReceived, a member of class TBaseSocket.

24.1.2.4.20 TBaseSocket.SetBytesSent Method

Pascal

```
procedure SetBytesSent(Value: Cardinal);
```

C++

```
__fastcall SetBytesSent(unsigned Value);
```

Description

This is SetBytesSent, a member of class TBaseSocket.

24.1.2.4.21 TBaseSocket.WaitForData Method

Waits for the socket connection to be ready to transfer data.

Pascal

```
function WaitForData(TimeOut: Integer = 0): Boolean;
```

C++

```
__fastcall Boolean WaitForData(int TimeOut = 0);
```

Description

Call WaitForData to ensure that the socket connection is ready to read or write information. WaitForData returns **true** if the socket connection is ready. WaitForData returns **false** if the socket connection is not ready after **TimeOut** milliseconds elapse.

Call WaitForData before reading or writing information over the socket connection. Otherwise, ReceiveBuf (see page 1376), ReceiveIn (see page 1379), SendBuf (see page 1380), and SendIn (see page 1380) method calls may time out before any data is transferred.

See Also

ReceiveBuf (see page 1376)

ReceiveIn (see page 1379)

SendBuf (see page 1380)

SendIn (see page 1380)

24.1.2.5 TBaseSocket Properties

The properties of the TBaseSocket class are listed here.

Public Properties

	Name	Description
	Active (see page 1383)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1385)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	SockType (see page 1387)	Sets or returns the socket type.

24.1.2.5.1 TBaseSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open (see page 1375) or Close (see page 1373) method to open or close the connection.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
```

```
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

Close (see page 1373)

Open (see page 1375)

24.1.2.5.2 TBaseSocket.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode (see page 1505) type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 1505)

OnReceive (see page 1393)

OnSend (see page 1394)

24.1.2.5.3 TBaseSocket.BytesReceived Property

Returns the number of bytes received by the socket.

Pascal

```
property BytesReceived: Cardinal;
```

C++

```
__property unsigned BytesReceived;
```

Description

BytesReceived returns the number of bytes received by the socket.

See Also

BytesSent (see page 1386)

24.1.2.5.4 TBaseSocket.BytesSent Property

Returns the number of bytes sent from the socket.

Pascal

```
property BytesSent: Cardinal;
```

C++

```
__property unsigned BytesSent;
```

Description

BytesSent returns the number of bytes sent from the socket.

See Also

BytesReceived (see page 1386)

24.1.2.5.5 TBaseSocket.Domain Property

Sets or returns the protocol family used by the socket.

Pascal

```
property Domain: TSocketDomain;
```

C++

```
__property TSocketDomain Domain;
```

Description

Use Domain to set or return the protocol family used by the socket. The TSocketDomain (see page 1500) type defines the set of possible protocol family types for the Domain property. The default value of this property is pfUnspec.

See Also

MapDomain (see page 1375)

TSocketDomain (see page 1500)

24.1.2.5.6 TBaseSocket.Handle Property

Provides access to the unique-4 byte identifier that represents the instance of the socket object.

Pascal

```
property Handle: TSocket;
```

C++

```
__property TSocket Handle;
```

Description

Use Handle to return a typed pointer to the instance of the socket class. This Handle can be used when making low level function calls into the QT shared libraries, where the function call requires an instance pointer to the object.

24.1.2.5.7 TBaseSocket.Protocol Property

Sets or returns the protocol used by the socket.

Pascal

```
property Protocol: TSocketProtocol;
```

C++

```
__property TSocketProtocol Protocol;
```

Description

Protocol sets or returns the protocol used by the socket. The socket protocol can be IP, TCP, UDP, or other network protocols.

See Also

TTcpServer (see page 1459)

24.1.2.5.8 TBaseSocket.SockType Property

Sets or returns the socket type.

Pascal

```
property SockType: TSocketType;
```

C++

```
__property TSocketType SockType;
```

Description

SockType sets or returns the type of socket connection.

The TSocketType (see page 1502) type defines the set of possible socket types for the SockType property.

See Also

TSocketType (see page 1502)

24.1.2.6 TBaseSocket Events

The events of the TBaseSocket class are listed here.

Public Events

	Name	Description
	OnCreateHandle (see page 1388)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1390)	Occurs when a socket is deactivated.
	OnError (see page 1393)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1393)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1394)	Occurs just before a client socket writes information to the socket connection.

24.1.2.6.1 TBaseSocket.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write (see page 3707) an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
```

```
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;
```

```
procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active (see page 1383)

Handle (see page 1387)

24.1.2.6.2 TBaseSocket.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write (see page 3707) an OnDestroyHandle event handler to take specific action when the socket is deactivated.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
```

```
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
```

```
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

24.1.2.6.3 TBaseSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write (see page 3707) an OnError event handler to respond to errors that arise with the socket connection. Set the **ErrorCode** parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError (see page 1369) exception from being raised.

See Also

ESocketError (see page 1369)

24.1.2.6.4 TBaseSocket.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write (see page 3707) an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 138)

BytesReceived (see page 1386)

ReceiveBuf (see page 1376)

ReceiveIn (see page 1379)

24.1.2.6.5 TBaseSocket.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write (see page 3707) an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------
```

```
void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

TWinSocketStream (see page 138)

SendBuf (see page 1380)

SendIn (see page 1380)

24.1.3 TClientSocketThread Class

Enables creation of separate threads of execution for the client socket in an application.

Class Hierarchy

Pascal

```
TClientSocketThread = class(TThread);
```

C++

```
class TClientSocketThread : public TThread;
```

File

Sockets

Description

Create (see page 1397) a TClientSocketThread (or a descendant of the class) to represent an execution thread for the client socket in a multi-threaded application. Each new instance of TClientSocketThread, a TThread descendant, is a new thread of execution. Using a separate TClientSocketThread allows the client socket to block without blocking the whole application.

See Also

Writing Multi-threaded Applications

TServerSocketThread (see page 1433)

TThread

24.1.3.1 TClientSocketThread Members

The following tables list the members exposed by TClientSocketThread.

Protected Methods

	Name	Description
	SyncProc (see page 1398)	Method where data synchronization is actually performed.

Public Methods

	Name	Description
	Create (see page 1397)	Creates an instance of a TClientSocketThread (see page 1396) object.
	Destroy (see page 1397)	Destroys the thread object and releases the memory allocated to it.
	Execute (see page 1398)	Executes when the thread is run.
	ExecuteSyncProc (see page 1398)	Synchronizes data between threads.

Public Properties

	Name	Description
	ClientSocket (see page 1399)	Refers to the client socket handling the connection in that thread.
	ServerSocketThread (see page 1399)	Specifies the server socket's thread.

24.1.3.2 TClientSocketThread.Create Constructor

Creates an instance of a TClientSocketThread (see page 1396) object.

Pascal

```
constructor Create(ServerSocketThread: TServerSocketThread);
```

C++

```
__fastcall TClientSocketThread(TServerSocketThread ServerSocketThread);
```

Description

For C++, call Create indirectly, using the **new** keyword, to create a thread in an application. If the **CreateSuspended** parameter is false, Execute (see page 1398) is called immediately. If **CreateSuspended** is true, Execute (see page 1398) is not called until after Resume is called.

See Also

TClientSocketThread (see page 1396)

Destroy (see page 1397)

Execute (see page 1398)

Resume

24.1.3.3 TClientSocketThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TClientSocketThread();
```

Description

For C++, do not call Destroy in an application. Instead, use the **delete** keyword to destroy a thread when FreeOnTerminate is false.

See Also

delete

Create

FreeOnTerminate

24.1.3.4 TClientSocketThread Methods

The methods of the TClientSocketThread class are listed here.

Protected Methods

	Name	Description
	SyncProc (see page 1398)	Method where data synchronization is actually performed.

Public Methods

	Name	Description
	Execute (see page 1398)	Executes when the thread is run.
	ExecuteSyncProc (see page 1398)	Synchronizes data between threads.

24.1.3.4.1 TClientSocketThread.Execute Method

Executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Use Execute to execute the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Terminated

Synchronize

24.1.3.4.2 TClientSocketThread.ExecuteSyncProc Method

Synchronizes data between threads.

Pascal

```
procedure ExecuteSyncProc;
```

C++

```
__fastcall ExecuteSyncProc();
```

Description

Objects in the client socket thread should not directly manipulate data in the main thread. Attempting to do so will cause synchronization problems and, most likely, the application will crash. To avoid synchronization errors, override the SyncProc (see page 1398) method to include instructions that must be executed by the main thread.

See Also

Synchronize

SyncProc (see page 1398)

24.1.3.4.3 TClientSocketThread.SyncProc Method

Method where data synchronization is actually performed.

Pascal

```
procedure SyncProc; virtual;
```

C++

```
virtual __fastcall SyncProc();
```

Description

In subclasses of TClientSocketThread (see page 1396), override SyncProc to perform synchronization between threads. Put code in it to be executed by the main thread.

Warning: Do not call this protected method directly; instead, call ExecuteSyncProc (see page 1398).

See Also

TClientSocketThread (see page 1396)

ExecuteSyncProc (see page 1398)

Synchronize

24.1.3.5 TClientSocketThread Properties

The properties of the TClientSocketThread class are listed here.

Public Properties

	Name	Description
R	ClientSocket (see page 1399)	Refers to the client socket handling the connection in that thread.
R	ServerSocketThread (see page 1399)	Specifies the server socket's thread.

24.1.3.5.1 TClientSocketThread.ClientSocket Property

Refers to the client socket handling the connection in that thread.

Pascal

```
property ClientSocket: TCustomIpClient;
```

C++

```
__property TCustomIpClient ClientSocket;
```

Description

Use ClientSocket to obtain access to the client socket object handling the connection in the current thread of execution.

See Also

TCustomIpClient (see page 1400)

24.1.3.5.2 TClientSocketThread.ServerSocketThread Property

Specifies the server socket's thread.

Pascal

```
property ServerSocketThread: TServerSocketThread;
```

C++

```
__property TServerSocketThread ServerSocketThread;
```

Description

Use ServerSocketThread get access to the thread object of the server socket.

See Also

TServerSocketThread (see page 1433)

24.1.4 TCustomIpClient Class

Encapsulates the client network functions.

Class Hierarchy

Pascal

```
TCustomIpClient = class(TIpSocket);
```

C++

```
class TCustomIpClient : public TIpSocket;
```

File

Sockets

Description

TCustomIpClient encapsulates the client network functions, linking sockets to network protocols such as TCP/IP and UDP, as well as raw sockets and custom protocols. Do not create instances of TCustomIpClient. Use TCustomIpClient as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TCustomIpClient provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TBaseSocket (see page 1370)

24.1.4.1 TCustomIpClient Members

The following tables list the members exposed by TCustomIpClient.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.
V	DoConnect (see page 1403)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
V	DoDisconnect (see page 1404)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
V	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnConnect (see page 1405)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1388)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1390)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1406)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1393)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1393)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1394)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
V	Close (see page 1402)	Shuts down the socket connection.
	Connect (see page 1403)	Connects the local socket to the remote socket.
V	Create (see page 1402)	Creates an instance of TCustomIpClient (see page 1400).
V	Destroy (see page 1372)	Frees the memory associated with the TBaseSocket (see page 1370) object.
	Disconnect (see page 1403)	Disconnects the local socket from the remote socket.
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	GetThreadObject (see page 1404)	Returns a reference to the TThread object for the currently executing thread.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.
	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
V	Open (see page 1404)	Creates the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveFrom (see page 1422)	Receives a packet of information.
	ReceiveIn (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	SendTo (see page 1422)	Sends a packet of information.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1383)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1385)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
R	Connected (see page 1405)	Indicates whether the socket connection is open and available for communication with other machines.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1423)	Specifies the name of the local system.

	LocalPort (see page 1424)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1426)	Specifies the name of the remote system.
	RemotePort (see page 1428)	Specifies the ID number of the port used by the RemoteHost (see page 1426) to form the socket connection.
	SockType (see page 1387)	Sets or returns the socket type.

24.1.4.2 TCustomIpClient.Create Constructor

Creates an instance of TCustomIpClient (see page 1400).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomIpClient(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TCustomIpClient (see page 1400) object.

See Also

Destroy (see page 1372)

24.1.4.3 TCustomIpClient Methods

The methods of the TCustomIpClient class are listed here.

Protected Methods

	Name	Description
	DoConnect (see page 1403)	This is DoConnect, a member of class TCustomIpClient.
	DoDisconnect (see page 1404)	This is DoDisconnect, a member of class TCustomIpClient.

Public Methods

	Name	Description
	Close (see page 1402)	Shuts down the socket connection.
	Connect (see page 1403)	Connects the local socket to the remote socket.
	Disconnect (see page 1403)	Disconnects the local socket from the remote socket.
	GetThreadObject (see page 1404)	Returns a reference to the TThread object for the currently executing thread.
	Open (see page 1404)	Creates the socket connection.

24.1.4.3.1 TCustomIpClient.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. This method performs the same functions as Disconnect (see page 1403).

See Also

Disconnect (see page 1403)

Open (see page 1404)

Active (see page 1383)

24.1.4.3.2 TCustomIpClient.Connect Method

Connects the local socket to the remote socket.

Pascal

```
function Connect: Boolean;
```

C++

```
__fastcall Boolean Connect();
```

Description

Call Connect to connect the local socket to the remote socket.

See Also

Connected (see page 1405)

Close (see page 1402)

Disconnect (see page 1403)

24.1.4.3.3 TCustomIpClient.Disconnect Method

Disconnects the local socket from the remote socket.

Pascal

```
procedure Disconnect;
```

C++

```
__fastcall Disconnect();
```

Description

Call Disconnect to break the connection between the local and remote sockets. This method performs the same function as Close (see page 1402).

See Also

Connected (see page 1405)

Close (see page 1402)

Connect (see page 1403)

24.1.4.3.4 TCustomIpClient.DoConnect Method

Pascal

```
procedure DoConnect; virtual;
```

C++

```
virtual __fastcall DoConnect();
```

Description

This is DoConnect, a member of class TCustomIpClient.

24.1.4.3.5 TCustomIpClient.DoDisconnect Method

Pascal

```
procedure DoDisconnect; virtual;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

This is DoDisconnect, a member of class TCustomIpClient.

24.1.4.3.6 TCustomIpClient.GetThreadObject Method

Returns a reference to the TThread object for the currently executing thread.

Pascal

```
function GetThreadObject: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread GetThreadObject();
```

Description

Call GetThreadObject to obtain a reference to the currently executing thread. GetThreadObject is only useful when BlockMode (see page 1385) is set to **bmBlockingMode**, which means that the client is executing in its own thread. If this method is called from the main application thread, the value NULL is returned.

See Also

TThread

BlockMode (see page 1385)

24.1.4.3.7 TCustomIpClient.Open Method

Creates the socket connection.

Pascal

```
procedure Open; override;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to create and initiate the socket connection.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

Active (see page 1383)

Close (see page 1402)

24.1.4.4 TCustomIpClient Properties

The properties of the TCustomIpClient class are listed here.

Public Properties

	Name	Description
	Connected (see page 1405)	Indicates whether the socket connection is open and available for communication with other machines.

24.1.4.4.1 TCustomIpClient.Connected Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read (see page 3634) Connected before trying to use or change the socket connection. Connected indicates whether the socket connection has been established. When Connected is **true**, the socket is open and available for use. When Connected is **false**, the socket is closed and can be changed.

To open a socket, use the Open (see page 1404) method. To close a socket, use the Closemethod.

See Also

Close (see page 1402)

Open (see page 1404)

OnConnect (see page 1405)

24.1.4.5 TCustomIpClient Events

The events of the TCustomIpClient class are listed here.

Public Events

	Name	Description
	OnConnect (see page 1405)	Occurs on client sockets just after the connection to the server is opened.
	OnDisconnect (see page 1406)	Occurs on client sockets just after the connection to the server is closed.

24.1.4.5.1 TCustomIpClient.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write (see page 3707) an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

See Also

Connected (see page 1405)

OnDisconnect (see page 1406)

24.1.4.5.2 TCustomIpClient.OnDisconnect Event

Occurs on client sockets just after the connection to the server is closed.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write (see page 3707) an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 1405)

OnConnect (see page 1405)

24.1.5 TCustomTcpServer Class

Ancestor object of all the server socket descendants.

Class Hierarchy

Pascal

```
TCustomTcpServer = class(TIpSocket);
```

C++

```
class TCustomTcpServer : public TIpSocket;
```

File

Sockets

Description

Do not create instances of TCustomTcpServer. Use TCustomTcpServer as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TCustomTcpServer provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

24.1.5.1 TCustomTcpServer Members

The following tables list the members exposed by TCustomTcpServer.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.
V	DoAccept (see page 1410)	This is DoAccept, a member of class TCustomTcpServer.
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
	Listen (see page 1410)	This is Listen, a member of class TCustomTcpServer.
V	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnAccept (see page 1413)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 1388)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1390)	Occurs when a socket is deactivated.
	OnError (see page 1393)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnGetThread (see page 1415)	A wrapper for the TServerSocketThread (see page 1433) event of the same name.
	OnListening (see page 1416)	Occurs when a server socket begins listening for connections.
	OnReceive (see page 1393)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1394)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Accept (see page 1409)	This is the overview for the Accept method overload.
V	Close (see page 1410)	Shuts down the socket connection.
V	Create (see page 1408)	Creates an instance of TCustomTcpServer (see page 1406).
V	Destroy (see page 1408)	Frees the memory associated with the TCustomTcpServer (see page 1406) object.
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.
	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
V	Open (see page 1411)	Opens the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveFrom (see page 1422)	Receives a packet of information.
	ReceiveIn (see page 1379)	Reads delimited lines of data from the socket.

	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendLn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	SendTo (see page 1422)	Sends a packet of information.
	WaitForConnection (see page 1411)	Listens for TCP connections.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1383)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1411)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Listening (see page 1412)	Sets the server application to listen for client requests.
	LocalHost (see page 1423)	Specifies the name of the local system.
	LocalPort (see page 1424)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1426)	Specifies the name of the remote system.
	RemotePort (see page 1428)	Specifies the ID number of the port used by the RemoteHost (see page 1426) to form the socket connection.
	ServerSocketThread (see page 1412)	Specifies the TServerSocketThread (see page 1433) in use.
	SockType (see page 1387)	Sets or returns the socket type.

24.1.5.2 TCustomTcpServer.Create Constructor

Creates an instance of TCustomTcpServer (see page 1406).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomTcpServer(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TCustomTcpServer (see page 1406) object.

See Also

TCustomTcpServer (see page 1406)

Destroy (see page 1408)

24.1.5.3 TCustomTcpServer.Destroy Destructor

Frees the memory associated with the TCustomTcpServer (see page 1406) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomTcpServer();
```

Description

For C++, do not call Destroy directly. Instead, use the **delete** keyword on the object, which causes Destroy to be invoked automatically.

See Also

delete

24.1.5.4 TCustomTcpServer Methods

The methods of the TCustomTcpServer class are listed here.

Protected Methods

	Name	Description
	DoAccept (see page 1410)	This is DoAccept, a member of class TCustomTcpServer.
	Listen (see page 1410)	This is Listen, a member of class TCustomTcpServer.

Public Methods

	Name	Description
	Accept (see page 1409)	This is the overview for the Accept method overload.
	Close (see page 1410)	Shuts down the socket connection.
	Open (see page 1411)	Opens the socket connection.
	WaitForConnection (see page 1411)	Listens for TCP connections.

24.1.5.4.1 Accept Method

This is the overview for the Accept method overload.

Overload List

	Name	Description
	TCustomTcpServer.Accept () (see page 1409)	Accepts a connection to a client socket.
	TCustomTcpServer.Accept (TCustomIpClient) (see page 1409)	Accepts a connection to a client socket.

24.1.5.4.1.1 TCustomTcpServer.Accept Method ()

Accepts a connection to a client socket.

Pascal

```
function Accept: Boolean; overload;
function Accept(var ClientSocket: TCustomIpClient): Boolean; overload;
```

C++

```
__fastcall Boolean Accept();
__fastcall Boolean Accept(TCustomIpClient ClientSocket);
```

Description

Accept is called automatically in response to notifications that a client socket is requesting a connection.

The Accept method for TCustomTcpServerdoes nothing with the notification. Descendants of TCustomTcpServer (see page

1406) that represent listening server sockets override this method to accept the connection to the client socket. In the second declaration of the method, a TCustomIpClient (see page 1400) object is created (as the **ClientSocket** parameter) to handle the connection.

See Also

TCustomIpClient (see page 1400)

TCustomTcpServer (see page 1406)

OnAccept (see page 1413)

24.1.5.4.2 TCustomTcpServer.Close Method

Shuts down the socket connection.

Pascal

```
procedure Close; override;
```

C++

```
virtual __fastcall Close();
```

Description

Call Close to shut down the socket connection. Close sets the Active (see page 1383) property to false.

For client sockets, Close terminates the connection to the server. For server sockets, Close shuts down the connection so that the server socket is no longer listening for client requests.

See Also

Active (see page 1383)

Open (see page 1411)

24.1.5.4.3 TCustomTcpServer.DoAccept Method

Pascal

```
procedure DoAccept(ClientSocket: TCustomIpClient); virtual;
```

C++

```
virtual __fastcall DoAccept(TCustomIpClient ClientSocket);
```

Description

This is DoAccept, a member of class TCustomTcpServer.

24.1.5.4.4 TCustomTcpServer.Listen Method

Pascal

```
function Listen(backlog: Integer = SOMAXCONN): Boolean;
```

C++

```
__fastcall Boolean Listen(int backlog = SOMAXCONN);
```

Description

This is Listen, a member of class TCustomTcpServer.

24.1.5.4.5 TCustomTcpServer.Open Method

Opens the socket connection.

Pascal

```
procedure Open; override;
```

C++

```
virtual __fastcall Open();
```

Description

Call Open to initiate the socket connection. Open sets the Active (see page 1383) property to true.

For client sockets, Open locates and connects to a server. For server sockets, Open opens the socket connection in a listening mode, but does not complete the connection to a client socket.

See Also

Active (see page 1383)

Close (see page 1410)

24.1.5.4.6 TCustomTcpServer.WaitForConnection Method

Listens for TCP connections.

Pascal

```
function WaitForConnection: Boolean;
```

C++

```
__fastcall Boolean WaitForConnection();
```

Description

Call WaitForConnection to listen for TCP connections on the port specified by the LocalPort (see page 1424) property. The thread will block until a connection request is received.

See Also

LocalPort (see page 1424)

Accept (see page 1409)

24.1.5.5 TCustomTcpServer Properties

The properties of the TCustomTcpServer class are listed here.

Public Properties

	Name	Description
	BlockMode (see page 1411)	Sets or returns the blocking mode of the socket.
R	Listening (see page 1412)	Sets the server application to listen for client requests.
	ServerSocketThread (see page 1412)	Specifies the TServerSocketThread (see page 1433) in use.

24.1.5.5.1 TCustomTcpServer.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TServerSocketBlockMode;
```

C++

```
__property TServerSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmThreadBlocking.

The TServerSocketBlockMode (see page 1500) type defines the set of possible socket flags for the BlockMode property.

See Also

OnReceive (see page 1393)

OnSend (see page 1394)

TServerSocketBlockMode (see page 1500)

24.1.5.5.2 TCustomTcpServer.Listening Property

Sets the server application to listen for client requests.

Pascal

```
property Listening: Boolean;
```

C++

```
__property Boolean Listening;
```

Description

Listening sets the server application to listen for client requests.

24.1.5.5.3 TCustomTcpServer.ServerSocketThread Property

Specifies the TServerSocketThread (see page 1433) in use.

Pascal

```
property ServerSocketThread: TServerSocketThread;
```

C++

```
__property TServerSocketThread ServerSocketThread;
```

Description

ServerSocketThread is a reference to the server socket thread object.

See Also

TServerSocketThread (see page 1433)

Listening (see page 1412)

24.1.5.6 TCustomTcpServer Events

The events of the TCustomTcpServer class are listed here.

Public Events

	Name	Description
	OnAccept (see page 1413)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnGetThread (see page 1415)	A wrapper for the TServerSocketThread (see page 1433) event of the same name.
	OnListening (see page 1416)	Occurs when a server socket begins listening for connections.

24.1.5.6.1 TCustomTcpServer.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketAcceptEvent;
```

C++

```
__property TSocketAcceptEvent OnAccept;
```

Description

Write (see page 3707) an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that is used in the server endpoint of the connection. The port number, and possibly the IP address, differs from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect event occurs for the client socket.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");
```

```
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;
```

```
procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active (see page 1383)

OnConnect (see page 1405)

OnListening (see page 1416)

24.1.5.6.2 TCustomTcpServer.OnGetThread Event

A wrapper for the TServerSocketThread (see page 1433) event of the same name.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

This event is exactly the same as the OnGetThread event of the TServerSocketThread (see page 1433) object referenced in ServerSocketThread (see page 1412).

See Also

TServerSocketThread (see page 1433)

ServerSocketThread (see page 1399)

OnGetThread (see page 1439)

24.1.5.6.3 TCustomTcpServer.OnListening Event

Occurs when a server socket begins listening for connections.

Pascal

```
property OnListening: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnListening;
```

Description

Write (see page 3707) an OnListening event to take specific action when a server begins listening for a connection.

See Also

Listening (see page 1412)

TNotifyEvent

24.1.6 TIpSocket Class

Encapsulates the network support functions.

Class Hierarchy

Pascal

```
TIpSocket = class(TBaseSocket);
```

C++

```
class TIpSocket : public TBaseSocket;
```

File

Sockets

Description

TIpSocket encapsulates the network support functions, linking sockets to network protocols such as TCP/IP and UDP, as well as raw sockets and custom protocols.

Do not create instances of TIpSocket. Use TIpSocket as a base class when declaring non-visual components that can appear on the component palette and be used in the form designer. Properties and methods of TIpSocket provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TBaseSocket (see page 1370)

24.1.6.1 TIpSocket Members

The following tables list the members exposed by TIpSocket.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnCreateHandle (see page 1388)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1390)	Occurs when a socket is deactivated.
	OnError (see page 1393)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1393)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1394)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Close (see page 1373)	Shuts down the socket connection.
	Create (see page 1418)	Creates an instance of TIpSocket (see page 1416).
	Destroy (see page 1372)	Frees the memory associated with the TBaseSocket (see page 1370) object.
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.
	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
	Open (see page 1375)	Creates the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveFrom (see page 1422)	Receives a packet of information.
	ReceiveLn (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendLn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	SendTo (see page 1422)	Sends a packet of information.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1383)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1385)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1423)	Specifies the name of the local system.
	LocalPort (see page 1424)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1426)	Specifies the name of the remote system.
	RemotePort (see page 1428)	Specifies the ID number of the port used by the RemoteHost (see page 1426) to form the socket connection.
	SockType (see page 1387)	Sets or returns the socket type.

24.1.6.2 TIpSocket.Create Constructor

Creates an instance of TIpSocket (see page 1416).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TIpSocket(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TIpSocket (see page 1416) object.

See Also

Destroy (see page 1372)

24.1.6.3 TIpSocket Methods

The methods of the TIpSocket class are listed here.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.

Public Methods

	Name	Description
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.
	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	ReceiveFrom (see page 1422)	Receives a packet of information.

	SendTo (see page 1422)	Sends a packet of information.

24.1.6.3.1 TIpSocket.Bind Method

Pascal

```
function Bind: Boolean;
```

C++

```
__fastcall Boolean Bind();
```

Description

This is Bind, a member of class TIpSocket.

24.1.6.3.2 TIpSocket.GetSocketAddr Method

Builds a valid network socket address.

Pascal

```
function GetSocketAddr(h: TSocketHost; p: TSocketPort): TSockAddr;
```

C++

```
__fastcall TSockAddr GetSocketAddr(TSocketHost h, TSocketPort p);
```

Description

Use GetSocketAddr to build a valid socket address based on the host name and port number. The **TSocketHost (see page 1506)** and **TSocketPort (see page 1506)** arguments are used to return a TSockAddr structure, which can then be passed to Connect or Bind (see page 1419).

See Also

Bind (see page 1419)

Connect (see page 1403)

LocalHostAddr (see page 1420)

LookupHostAddr (see page 1420)

RemoteHost (see page 1426)

RemotePort (see page 1428)

24.1.6.3.3 TIpSocket.LocalDomainName Method

Returns the domain name for the local host.

Pascal

```
function LocalDomainName: string;
```

C++

```
__fastcall UnicodeString LocalDomainName();
```

Description

Use LocalDomainName name to return the domain name for the local host.

See Also

LocalHostAddr (see page 1420)

LocalHostName (see page 1420)

24.1.6.3.4 TIpSocket.LocalHostAddr Method

Returns the local IP address for the host machine.

Pascal

```
function LocalHostAddr: TSocketHost;
```

C++

```
__fastcall TSocketHost LocalHostAddr();
```

Description

Use LocalHostAddr to return the local IP address in string format.

See Also

LocalHost (see page 1423)

LocalDomainName (see page 1419)

LocalHostName (see page 1420)

24.1.6.3.5 TIpSocket.LocalHostName Method

Returns the local host name.

Pascal

```
function LocalHostName: TSocketHost;
```

C++

```
__fastcall TSocketHost LocalHostName();
```

Description

Use LocalHostName to return the host name for the local host. For example:

```
ShowMessage(TcpClient1.LocalHostName);
```

See Also

LocalHost (see page 1423)

LocalDomainName (see page 1419)

LocalHostAddr (see page 1420)

24.1.6.3.6 TIpSocket.LookupHostAddr Method

Looks up the host address based on the host name.

Pascal

```
function LookupHostAddr(const hn: AnsiString): TSocketHost;
```

C++

```
__fastcall TSocketHost LookupHostAddr(const AnsiString hn);
```

Description

Use LookupHostAddr to look up the host address based on the machine's host name. For example:

```
StrToInt(TcpClient1.LookupHostAddr("borland.com"));
```

See Also

RemoteHost (see page 1426)

LookupHostName (see page 1421)

LookupPort (see page 1421)

24.1.6.3.7 TIpSocket.LookupHostName Method

Looks up the host name based on the IP address.

Pascal

```
function LookupHostName(const ipaddr: AnsiString): TSocketHost;
```

C++

```
__fastcall TSocketHost LookupHostName(const AnsiString ipaddr);
```

Description

Use LookupHostName to look up the host name based on the machine's IP address. For example:

```
TcpClient1.LookupHostName("123.60.192.10");
```

See Also

RemoteHost (see page 1426)

LookupHostAddr (see page 1420)

LookupPort (see page 1421)

24.1.6.3.8 TIpSocket.LookupPort Method

Returns the port number.

Pascal

```
function LookupPort(const sn: AnsiString; pn: PAnsiChar = nil): word;
```

C++

```
__fastcall word LookupPort(const AnsiString sn, PAnsiChar pn = nil);
```

Description

Use LookupPort to return the port number of the IP socket. The port number is related to the service name and protocol number.

For more information, refer to the home page of the Internet Assigned (see page 3524) Numbers Authority (www.iana.org), which maintains lists of port numbers and their associated protocols. Also, refer to the documentation for the application's target operating system for information on how port numbers are stored and changed in that OS.

See Also

RemotePort (see page 1428)

LookupProtocol (see page 1421)

24.1.6.3.9 TIpSocket.LookupProtocol Method

Returns the protocol number.

Pascal

```
function LookupProtocol(const pn: AnsiString): TSocketProtocol;
```

C++

```
__fastcall TSocketProtocol LookupProtocol(const AnsiString pn);
```

Description

Use LookupProtocol to return the protocol number being used by the IP socket. The protocol number is based on the protocol name.

For more information, refer to the home page of the Internet Assigned (see page 3524) Numbers Authority (www.iana.org), which maintains lists of protocol numbers and their associated protocols.

See Also

RemotePort (see page 1428)

LookupPort (see page 1421)

24.1.6.3.10 TIpSocket.ReceiveFrom Method

Receives a packet of information.

Pascal

```
function ReceiveFrom(var buf; bufsize: Integer; ToAddr: TSockAddr; var len: Integer; flags:
Integer = 0): Integer;
```

C++

```
__fastcall int ReceiveFrom( buf, int bufsize, TSockAddr ToAddr, int len, int flags = 0);
```

Description

Use ReceiveFrom to receive a single packet of information. Note that information in the packet is not checked for length, completeness, and such, which occurs at a higher level.

See Also

SendTo (see page 1422)

OnReceive (see page 1393)

24.1.6.3.11 TIpSocket.SendTo Method

Sends a packet of information.

Pascal

```
function SendTo(var buf; bufsize: Integer; ToAddr: TSockAddr; flags: Integer = 0): Integer;
```

C++

```
__fastcall int SendTo( buf, int bufsize, TSockAddr ToAddr, int flags = 0);
```

Description

Use SendTo to send a single packet of information over the network. Note that information in the packet is not checked for length, completeness, and such, which occurs at a higher level.

See Also

ReceiveFrom (see page 1422)

OnSend (see page 1394)

24.1.6.4 TIpSocket Properties

The properties of the TIpSocket class are listed here.

Public Properties

	Name	Description
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
	LocalHost (see page 1423)	Specifies the name of the local system.
	LocalPort (see page 1424)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 1426)	Specifies the name of the remote system.
	RemotePort (see page 1428)	Specifies the ID number of the port used by the RemoteHost (see page 1426) to form the socket connection.

24.1.6.4.1 TIpSocket.Domain Property

Sets or returns the protocol family used by the socket.

Pascal

```
property Domain: TSocketDomain;
```

C++

```
__property TSocketDomain Domain;
```

Description

Use Domain to set or return the protocol family used by the socket. The TSocketDomain type defines the set of possible protocol family types for the Domain property. The default value of this property is pfUnspec.

See Also

MapDomain (see page 1375)

TSocketDomain (see page 1500)

24.1.6.4.2 TIpSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr (see page 1420) function, to host multiple connections simultaneously. Each combination of LocalPort (see page 1424) and LocalHostAddr (see page 1420) can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr (see page 1420) function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given

system, contact your network administrator.

See Also

LocalPort (see page 1424)

RemoteHost (see page 1426)

LocalHostAddr (see page 1420)

LookupHostName (see page 1421)

LookupPort (see page 1421)

24.1.6.4.3 TIpSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
```

```
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
```

```
    Memo1.Lines.Clear;
  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Memo1.Lines.Clear;

    //server initialization
    TcpServer1 := TTcpServer.Create(Self);
    TcpServer1.OnAccept := TCpServer1Accept;
    TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
    TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
    TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
  end;

  procedure TForm3.TcpServer1Accept(Sender: TObject;
    ClientSocket: TCustomIpClient);
  var
    a : array[0..9] of byte;
    I : Cardinal;
  begin

    ClientSocket.ReceiveBuf(a,10,0);
    Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                    IntToStr(a[9]));
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

LocalHost (see page 1423)

RemotePort (see page 1428)

LookupPort (see page 1421)

24.1.6.4.4 TIpSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the

domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
```

```
    myStream : TMemoryStream;
    I : Cardinal;
    userValue : Integer;
  begin
    //creating a stream
    myStream := TMemoryStream.Create();
    for I := 0 to 8 do
    begin
      myStream.WriteBuffer(I,1);
    end;
    userValue := StrToIntDef(Edit1.Text, 0);
    myStream.WriteBuffer(userValue,1);

    //resetting the stream position
    myStream.Seek(0,0);

    //sending the stream
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

RemotePort (see page 1428)

LocalHostAddr (see page 1420)

LocalHostName (see page 1420)

LookupHostAddr (see page 1420)

LookupHostName (see page 1421)

24.1.6.4.5 TIpSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost (see page 1426) to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost (see page 1426) property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost (see page 1426) can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol (see page 1421) method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)
```

```
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

RemoteHost (see page 1426)

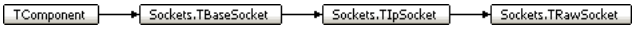
LookupPort (see page 1421)

LookupProtocol (see page 1421)

24.1.7 TRawSocket Class

Encapsulates a raw socket.

Class Hierarchy



Pascal

```
TRawSocket = class(TIpSocket);
```

C++

```
class TRawSocket : public TIpSocket;
```

File

Sockets

Description

TRawSocket encapsulates a raw socket.

See Also

TBaseSocket (see page 1370)

24.1.7.1 TRawSocket Members

The following tables list the members exposed by TRawSocket.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
V	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnCreateHandle (see page 1388)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1390)	Occurs when a socket is deactivated.
	OnError (see page 1393)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1393)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1394)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Close (see page 1373)	Shuts down the socket connection.
	Create (see page 1432)	Creates an instance of a TRawSocket (see page 1431) object.
	Destroy (see page 1372)	Frees the memory associated with the TBaseSocket (see page 1370) object.
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.
	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
	Open (see page 1375)	Creates the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveFrom (see page 1422)	Receives a packet of information.
	ReceiveIn (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	SendTo (see page 1422)	Sends a packet of information.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
	Active (see page 1383)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1385)	Sets or returns the blocking mode of the socket.
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1423)	Specifies the name of the local system.
	LocalPort (see page 1424)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1426)	Specifies the name of the remote system.
	RemotePort (see page 1428)	Specifies the ID number of the port used by the RemoteHost (see page 1426) to form the socket connection.
	SockType (see page 1433)	Sets or returns the socket type.

24.1.7.2 TRawSocket.Create Constructor

Creates an instance of a TRawSocket (see page 1431) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRawSocket(TComponent * AOwner);
```

Description

Call Create indirectly, using the **new** keyword, to instantiate a TRawSocket (see page 1431) object.

See Also

TRawSocket (see page 1431)

Destroy (see page 1372)

24.1.7.3 TRawSocket Properties

The properties of the TRawSocket class are listed here.

Public Properties

	Name	Description
	SockType (see page 1433)	Sets or returns the socket type.

24.1.7.3.1 TRawSocket.SockType Property

Sets or returns the socket type.

Pascal

```
property SockType: TSocketType;
```

C++

```
__property TSocketType SockType;
```

Description

SockType sets or returns the type of socket connection.

The TSocketType (see page 1502) type defines the set of possible socket types for the SockType property.

See Also

TSocketType (see page 1502)

24.1.8 TServerSocketThread Class

Enables creation of separate threads of execution for the server socket in an application.

Class Hierarchy

Pascal

```
TServerSocketThread = class(TThread);
```

C++

```
class TServerSocketThread : public TThread;
```

File

Sockets

Description

Create (see page 1434) TServerSocketThread (or a descendant of the class) to represent an execution thread for the server socket in a multi-threaded application. Each new instance of a TServerSocketThread, a TThread descendant, is a new thread of execution. Using a separate TServerSocketThread allows the server socket to block without blocking the whole application.

See Also

Writing Multi-threaded Applications

TClientSocketThread (see page 1396)

TThread

24.1.8.1 TServerSocketThread Members

The following tables list the members exposed by TServerSocketThread.

Protected Methods

	Name	Description
	AddClientSocketThread (see page 1436)	This is AddClientSocketThread, a member of class TServerSocketThread.
V	CreateThread (see page 1436)	This is CreateThread, a member of class TServerSocketThread.
	FetchClientSocketThread (see page 1437)	This is FetchClientSocketThread, a member of class TServerSocketThread.
	RemoveClientSocketThread (see page 1437)	This is RemoveClientSocketThread, a member of class TServerSocketThread.

Public Events

	Name	Description
	OnGetThread (see page 1439)	Creates and initializes a client socket thread when a connection is requested.

Public Methods

	Name	Description
	ClearThreadPool (see page 1436)	Terminates all of the threads in the thread pool.
	Create (see page 1434)	Creates an instance of a TServerSocketThread (see page 1433) object.
V	Destroy (see page 1435)	Destroys the thread object and releases the memory allocated to it.
V	Execute (see page 1436)	Provides the code which executes when the thread is run.

Public Properties

	Name	Description
R	ServerSocket (see page 1438)	Specifies the TCustomTcpServer (see page 1406) object that manages all the TClientSocketThread (see page 1396) objects for its client connections.
	ThreadCacheSize (see page 1438)	Indicates the number of TServerSocketThread (see page 1433) objects cached for reuse by new connections to client sockets.
R	ThreadPool (see page 1438)	A list of all the client socket threads created by the TServerSocketThread (see page 1433) object.

24.1.8.2 TServerSocketThread.Create Constructor

Creates an instance of a TServerSocketThread (see page 1433) object.

Pascal

```
constructor Create(AServerSocket: TCustomTcpServer);
```

C++

```
__fastcall TServerSocketThread(TCustomTcpServer AServerSocket);
```

Description

For C++, call Create indirectly, using the **new** keyword, to create a thread in an application.

See Also

Destroy (see page 1435)

Execute (see page 1436)

24.1.8.3 TServerSocketThread.Destroy Destructor

Destroys the thread object and releases the memory allocated to it.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServerSocketThread();
```

Description

For C++, do not call Destroy in an application. Instead, use the **delete** keyword to destroy a thread when FreeOnTerminate is false.

Destroy signals the thread to terminate and then waits for the thread to return before calling the parent class's destructor.

See Also

delete

FreeOnTerminate

Create (see page 1434)

24.1.8.4 TServerSocketThread Methods

The methods of the TServerSocketThread class are listed here.

Protected Methods

	Name	Description
	AddClientSocketThread (see page 1436)	This is AddClientSocketThread, a member of class TServerSocketThread.
V	CreateThread (see page 1436)	This is CreateThread, a member of class TServerSocketThread.
	FetchClientSocketThread (see page 1437)	This is FetchClientSocketThread, a member of class TServerSocketThread.
	RemoveClientSocketThread (see page 1437)	This is RemoveClientSocketThread, a member of class TServerSocketThread.

Public Methods

	Name	Description
	ClearThreadPool (see page 1436)	Terminates all of the threads in the thread pool.
V	Execute (see page 1436)	Provides the code which executes when the thread is run.

24.1.8.4.1 TServerSocketThread.AddClientSocketThread Method

Pascal

```
function AddClientSocketThread: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread AddClientSocketThread();
```

Description

This is AddClientSocketThread, a member of class TServerSocketThread.

24.1.8.4.2 TServerSocketThread.ClearThreadPool Method

Terminates all of the threads in the thread pool.

Pascal

```
procedure ClearThreadPool;
```

C++

```
__fastcall ClearThreadPool();
```

Description

The thread pool is a list of client socket threads that are available to handle connection requests. This list is kept to allow reuse of old threads and improve performance. It is sometimes useful to destroy all of the thread object instances in this list.

Use ClearThreadPool to terminate all of the threads in the thread pool list. Any suspended thread in the pool is resumed so that termination can proceed.

See Also

ThreadPool (see page 1438)

Terminate

24.1.8.4.3 TServerSocketThread.CreateThread Method

Pascal

```
function CreateThread: TClientSocketThread; virtual;
```

C++

```
virtual __fastcall TClientSocketThread CreateThread();
```

Description

This is CreateThread, a member of class TServerSocketThread.

24.1.8.4.4 TServerSocketThread.Execute Method

Provides the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Use Execute to execute the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

See Also

Terminated

Synchronize

Create (see page 1434)

24.1.8.4.5 TServerSocketThread.FetchClientSocketThread Method

Pascal

```
function FetchClientSocketThread: TClientSocketThread;
```

C++

```
__fastcall TClientSocketThread FetchClientSocketThread();
```

Description

This is FetchClientSocketThread, a member of class TServerSocketThread.

24.1.8.4.6 TServerSocketThread.RemoveClientSocketThread Method

Pascal

```
procedure RemoveClientSocketThread(ClientSocketThread: TClientSocketThread);
```

C++

```
__fastcall RemoveClientSocketThread(TClientSocketThread ClientSocketThread);
```

Description

This is RemoveClientSocketThread, a member of class TServerSocketThread.

24.1.8.5 TServerSocketThread Properties

The properties of the TServerSocketThread class are listed here.

Public Properties

	Name	Description
	ServerSocket (see page 1438)	Specifies the TCustomTcpServer (see page 1406) object that manages all the TClientSocketThread (see page 1396) objects for its client connections.
	ThreadCacheSize (see page 1438)	Indicates the number of TServerSocketThread (see page 1433) objects cached for reuse by new connections to client sockets.
	ThreadPool (see page 1438)	A list of all the client socket threads created by the TServerSocketThread (see page 1433) object.

24.1.8.5.1 TServerSocketThread.ServerSocket Property

Specifies the TCustomTcpServer (see page 1406) object that manages all the TClientSocketThread (see page 1396) objects for its client connections.

Pascal

```
property ServerSocket: TCustomTcpServer;
```

C++

```
__property TCustomTcpServer ServerSocket;
```

Description

Use ServerSocket to get direct access to the listening socket object that spawned the thread for a client connection it accepted. TServerSocketThread (see page 1433) passes events and errors to the appropriate event handlers assigned to ServerSocket. TServerSocketThread (see page 1433) adds itself to the cache of threads maintained by ServerSocket from its constructor, and removes itself when the thread terminates.

See Also

TClientSocketThread (see page 1396)

TServerSocketThread (see page 1433)

24.1.8.5.2 TServerSocketThread.ThreadCacheSize Property

Indicates the number of TServerSocketThread (see page 1433) objects cached for reuse by new connections to client sockets.

Pascal

```
property ThreadCacheSize: Integer;
```

C++

```
__property int ThreadCacheSize;
```

Description

Set ThreadCacheSize to specify the number of threads that are cached for reuse. The ideal value for ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket. If ThreadCacheSize is too low, the server socket will spend more time freeing and creating threads when client connections are accepted. If ThreadCacheSize is too high, the server socket may unnecessarily lock up the memory for threads that are never reused.

24.1.8.5.3 TServerSocketThread.ThreadPool Property

A list of all the client socket threads created by the TServerSocketThread (see page 1433) object.

Pascal

```
property ThreadPool: TList;
```

C++

```
__property TList ThreadPool;
```

Description

TServerSocketThread (see page 1433) objects listen for requests on the network and delegate them to client socket threads. Each client socket thread is added to the list ThreadPool after it is created. Creating and destroying client socket threads can be quite time consuming, so whenever one completes its work on a given connection it becomes dormant. When a new connection request is received, the ThreadPool is checked for dormant client socket threads. If one is available, it is assigned the

connection. If not, a new client socket thread created. By reusing previously-created threads, the application's performance is improved.

See Also

TClientSocketThread (see page 1396)

ClearThreadPool (see page 1436)

24.1.8.6 TServerSocketThread Events

The events of the TServerSocketThread class are listed here.

Public Events

	Name	Description
	OnGetThread (see page 1439)	Creates and initializes a client socket thread when a connection is requested.

24.1.8.6.1 TServerSocketThread.OnGetThread Event

Creates and initializes a client socket thread when a connection is requested.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

An OnGetThread event occurs when a connection is requested. The event handler creates an instance of TClientSocketThread (see page 1396) and initializes it. Write (see page 3707) an event handler on TCustomTcpServer (see page 1406) to create and initialize instances of your own client socket thread class.

See Also

TClientSocketThread (see page 1396)

TCustomTcpServer (see page 1406)

24.1.9 TTcpClient Class

TCP client component.

Class Hierarchy

Pascal

```
TTcpClient = class(TCustomIpClient);
```

C++

```
class TTcpClient : public TCustomIpClient;
```

File

Sockets

Description

Use TTcpClient to create TCP client applications.

TTcpClient publishes properties and events from its ancestor socket components.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
```

```
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

TCustomIpClient (see page 1400)

TTcpServer (see page 1459)

24.1.9.1 TTcpClient Members

The following tables list the members exposed by TTcpClient.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.
V	DoConnect (see page 1403)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
V	DoDisconnect (see page 1404)	This is DoDisconnect, a member of class TCustomIpClient.

	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
V	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Methods

	Name	Description
V	Close (see page 1373)	Shuts down the socket connection.
V	Close (see page 1402)	Shuts down the socket connection.
	Connect (see page 1403)	Connects the local socket to the remote socket.
V	Create (see page 1371)	Creates an instance of TBaseSocket (see page 1370).
V	Create (see page 1402)	Creates an instance of TCustomIpClient (see page 1400).
V	Create (see page 1418)	Creates an instance of TIpSocket (see page 1416).
V	Destroy (see page 1372)	Frees the memory associated with the TBaseSocket (see page 1370) object.
	Disconnect (see page 1403)	Disconnects the local socket from the remote socket.
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	GetThreadObject (see page 1404)	Returns a reference to the TThread object for the currently executing thread.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.
	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
V	Open (see page 1375)	Creates the socket connection.
V	Open (see page 1404)	Creates the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveFrom (see page 1422)	Receives a packet of information.
	Receiveln (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	Sendln (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	SendTo (see page 1422)	Sends a packet of information.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	LocalHost (see page 1423)	Specifies the name of the local system.
	LocalPort (see page 1424)	Specifies the ID number of the port used by the socket connection.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	SockType (see page 1387)	Sets or returns the socket type.

Published Events

	Name	Description
	OnConnect (see page 1450)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1451)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1453)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1456)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1456)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1456)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1457)	Occurs just before a client socket writes information to the socket connection.

Published Properties

	Name	Description
	Active (see page 1443)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1445)	Sets or returns the blocking mode of the socket.
	Connected (see page 1446)	Indicates whether the socket connection is open and available for communication with other machines.
	RemoteHost (see page 1446)	Specifies the name of the remote system.
	RemotePort (see page 1448)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

24.1.9.2 TTcpClient Properties

The properties of the TTcpClient class are listed here.

Published Properties

	Name	Description
	Active (see page 1443)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1445)	Sets or returns the blocking mode of the socket.
	Connected (see page 1446)	Indicates whether the socket connection is open and available for communication with other machines.
	RemoteHost (see page 1446)	Specifies the name of the remote system.
	RemotePort (see page 1448)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

24.1.9.2.1 TTcpClient.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
```

```
    userValue : Integer;
  begin
    //creating a stream
    myStream := TMemoryStream.Create();
    for I := 0 to 8 do
    begin
      myStream.WriteBuffer(I,1);
    end;
    userValue := StrToIntDef(Edit1.Text, 0);
    myStream.WriteBuffer(userValue,1);

    //resetting the stream position
    myStream.Seek(0,0);

    //sending the stream
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

Close (see page 1373)

Open (see page 1375)

24.1.9.2.2 TTcpClient.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 1505)

OnReceive (see page 1393)

OnSend (see page 1394)

24.1.9.2.3 TTcpClient.Connected Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Read Connected before trying to use or change the socket connection. Connected indicates whether the socket connection has been established. When Connected is **true**, the socket is open and available for use. When Connected is **false**, the socket is closed and can be changed.

To open a socket, use the Open method. To close a socket, use the Closemethod.

See Also

Close (see page 1402)

Open (see page 1404)

OnConnect (see page 1405)

24.1.9.2.4 TTcpClient.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
```

```
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
```

```
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

RemotePort (see page 1428)

LocalHostAddr (see page 1420)

LocalHostName (see page 1420)

LookupHostAddr (see page 1420)

LookupHostName (see page 1421)

24.1.9.2.5 TTcpClient.RemotePort Property

Specifies the ID number of the port used by the RemoteHost to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
```

```
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
```

```
    begin
      myStream.WriteBuffer(I,1);
    end;
    userValue := StrToIntDef(Edit1.Text, 0);
    myStream.WriteBuffer(userValue,1);

    //resetting the stream position
    myStream.Seek(0,0);

    //sending the stream
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

RemoteHost (see page 1426)

LookupPort (see page 1421)

LookupProtocol (see page 1421)

24.1.9.3 TTcpClient Events

The events of the TTcpClient class are listed here.

Published Events

	Name	Description
	OnConnect (see page 1450)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1451)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1453)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1456)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1456)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1456)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1457)	Occurs just before a client socket writes information to the socket connection.

24.1.9.3.1 TTcpClient.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

See Also

Connected (see page 1405)

OnDisconnect (see page 1406)

24.1.9.3.2 TTcpClient.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
```

```
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
```

```
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active (see page 1383)

Handle (see page 1387)

24.1.9.3.3 TTcpClient.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
```

```
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
```

```
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

24.1.9.3.4 TTcpClient.OnDisconnect Event

Occurs on client sockets just after the connection to the server is closed.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 1405)

OnConnect (see page 1405)

24.1.9.3.5 TTcpClient.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the **ErrorCode** parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError exception from being raised.

See Also

ESocketError (see page 1369)

24.1.9.3.6 TTcpClient.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 138)

BytesReceived (see page 1386)

ReceiveBuf (see page 1376)

ReceiveIn (see page 1379)

24.1.9.3.7 TTcpClient.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
```

```
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
```

```
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

TWinSocketStream (see page 138)

SendBuf (see page 1380)

SendIn (see page 1380)

24.1.10 TTcpServer Class

TTcpServer is the TCP server component.

Class Hierarchy

Pascal

```
TTcpServer = class(TCustomTcpServer);
```

C++

```
class TTcpServer : public TCustomTcpServer;
```

File

Sockets

Description

Use TTcpServer to create TCP server applications.

TTcpServer publishes properties and events from its ancestor socket components.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
```

```
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

TCustomTcpServer (see page 1406)

TTcpClient (see page 1439)

24.1.10.1 TTcpServer Members

The following tables list the members exposed by TTcpServer.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.
	DoAccept (see page 1410)	This is DoAccept, a member of class TCustomTcpServer.
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
	Listen (see page 1410)	This is Listen, a member of class TCustomTcpServer.
	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Events

	Name	Description
	OnError (see page 1393)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1393)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1394)	Occurs just before a client socket writes information to the socket connection.

Public Methods

	Name	Description
	Accept (see page 1409)	This is the overview for the Accept method overload.
	Close (see page 1373)	Shuts down the socket connection.
	Close (see page 1410)	Shuts down the socket connection.
	Create (see page 1371)	Creates an instance of TBaseSocket (see page 1370).
	Create (see page 1408)	Creates an instance of TCustomTcpServer (see page 1406).
	Create (see page 1418)	Creates an instance of TIpSocket (see page 1416).
	Destroy (see page 1372)	Frees the memory associated with the TBaseSocket (see page 1370) object.
	Destroy (see page 1408)	Frees the memory associated with the TCustomTcpServer (see page 1406) object.
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.
	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
	Open (see page 1375)	Creates the socket connection.
	Open (see page 1411)	Opens the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveFrom (see page 1422)	Receives a packet of information.
	ReceiveIn (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	SendTo (see page 1422)	Sends a packet of information.

	WaitForConnection (see page 1411)	Listens for TCP connections.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
R	Listening (see page 1412)	Sets the server application to listen for client requests.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	RemoteHost (see page 1426)	Specifies the name of the remote system.
	RemotePort (see page 1428)	Specifies the ID number of the port used by the RemoteHost (see page 1426) to form the socket connection.
	ServerSocketThread (see page 1412)	Specifies the TServerSocketThread (see page 1433) in use.
	SockType (see page 1387)	Sets or returns the socket type.

Published Events

	Name	Description
	OnAccept (see page 1469)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 1472)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1474)	Occurs when a socket is deactivated.
	OnGetThread (see page 1477)	A wrapper for the TServerSocketThread event of the same name.
	OnListening (see page 1477)	Occurs when a server socket begins listening for connections.

Published Properties

	Name	Description
	Active (see page 1463)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1465)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1466)	Specifies the name of the local system.
	LocalPort (see page 1466)	Specifies the ID number of the port used by the socket connection.

24.1.10.2 TTcpServer Properties

The properties of the TTcpServer class are listed here.

Published Properties

	Name	Description
	Active (see page 1463)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1465)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1466)	Specifies the name of the local system.
	LocalPort (see page 1466)	Specifies the ID number of the port used by the socket connection.

24.1.10.2.1 TTcpServer.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
```

```
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

Close (see page 1373)

Open (see page 1375)

24.1.10.2.2 TTcpServer.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TServerSocketBlockMode BlockMode;__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmThreadBlocking.

The TServerSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

OnReceive (see page 1393)

OnSend (see page 1394)

TServerSocketBlockMode (see page 1500)

24.1.10.2.3 TTcpServer.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr function, to host multiple connections simultaneously. Each combination of LocalPort and LocalHostAddr can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 1424)

RemoteHost (see page 1426)

LocalHostAddr (see page 1420)

LookupHostName (see page 1421)

LookupPort (see page 1421)

24.1.10.2.4 TTcpServer.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin
```

```
    ClientSocket.ReceiveBuf(a,10,0);
    Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                    IntToStr(a[9]));
    for I := 0 to 8 do
    begin
      Memo1.Lines.Add(IntToStr(a[I] + a[9]));
    end;
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

LocalHost (see page 1423)

RemotePort (see page 1428)

LookupPort (see page 1421)

24.1.10.3 TTcpServer Events

The events of the TTcpServer class are listed here.

Published Events

	Name	Description
	OnAccept (see page 1469)	Occurs on server sockets just after the connection to a client socket is accepted.
	OnCreateHandle (see page 1472)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1474)	Occurs when a socket is deactivated.
	OnGetThread (see page 1477)	A wrapper for the TServerSocketThread event of the same name.
	OnListening (see page 1477)	Occurs when a server socket begins listening for connections.

24.1.10.3.1 TTcpServer.OnAccept Event

Occurs on server sockets just after the connection to a client socket is accepted.

Pascal

```
property OnAccept: TSocketAcceptEvent;
```

C++

```
__property TSocketAcceptEvent OnAccept;
```

Description

Write an OnAccept event handler for a server socket to take specific action after the connection to a client socket has been accepted. This is the first point when information is available about the local port and IP address that is used in the server endpoint of the connection. The port number, and possibly the IP address, differs from those of the listening socket.

Server sockets open a socket connection for listening to establish a queue to hold client requests. After a request from the queue

is accepted by the server socket, an OnAccept event occurs. After the server socket accepts the connection, the client socket completes the connection, and an OnConnect event occurs for the client socket.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');
```

```
end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active (see page 1383)

OnConnect (see page 1405)

OnListening (see page 1416)

24.1.10.3.2 TTcpServer.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
```

```
                        IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;
```

```
procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

Active (see page 1383)

Handle (see page 1387)

24.1.10.3.3 TTcpServer.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
```

```
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

24.1.10.3.4 TTcpServer.OnGetThread Event

A wrapper for the TServerSocketThread event of the same name.

Pascal

```
property OnGetThread: TGetThreadEvent;
```

C++

```
__property TGetThreadEvent OnGetThread;
```

Description

This event is exactly the same as the OnGetThread event of the TServerSocketThread object referenced in ServerSocketThread.

See Also

TServerSocketThread (see page 1433)

ServerSocketThread (see page 1399)

OnGetThread (see page 1439)

24.1.10.3.5 TTcpServer.OnListening Event

Occurs when a server socket begins listening for connections.

Pascal

```
property OnListening: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnListening;
```

Description

Write an OnListening event to take specific action when a server begins listening for a connection.

See Also

Listening (see page 1412)

TNotifyEvent

24.1.11 TUdpSocket Class

UDP component.

Class Hierarchy

Pascal

```
TUdpSocket = class(TCustomIpClient);
```

C++

```
class TUdpSocket : public TCustomIpClient;
```

File

Sockets

Description

Use TUdpSocket to create UDP-based applications. Add a TUdpSocket object to a form or data module to turn an application into a UDP/IP client and server. TUdpSocket specifies a desired connection to a UDP/IP server, manages the connection when it is open, and terminates the connection when the application is through. It also listens for requests for UDP/IP connections from other machines and establishes connections when requests are received.

24.1.11.1 TUdpSocket Members

The following tables list the members exposed by TUdpSocket.

Protected Methods

	Name	Description
	Bind (see page 1419)	This is Bind, a member of class TIpSocket.
V	DoConnect (see page 1403)	This is DoConnect, a member of class TCustomIpClient.
	DoCreateHandle (see page 1373)	This is DoCreateHandle, a member of class TBaseSocket.
	DoDestroyHandle (see page 1373)	This is DoDestroyHandle, a member of class TBaseSocket.
V	DoDisconnect (see page 1404)	This is DoDisconnect, a member of class TCustomIpClient.
	DoHandleError (see page 1373)	This is DoHandleError, a member of class TBaseSocket.
V	DoReceive (see page 1374)	This is DoReceive, a member of class TBaseSocket.
V	DoSend (see page 1374)	This is DoSend, a member of class TBaseSocket.
V	ErrorCheck (see page 1374)	This is ErrorCheck, a member of class TBaseSocket.
V	Loaded (see page 1374)	Initializes the component after the form file has been read into memory.
	SetBytesReceived (see page 1382)	This is SetBytesReceived, a member of class TBaseSocket.
	SetBytesSent (see page 1382)	This is SetBytesSent, a member of class TBaseSocket.

Public Methods

	Name	Description
V	Close (see page 1373)	Shuts down the socket connection.
V	Close (see page 1402)	Shuts down the socket connection.
	Connect (see page 1403)	Connects the local socket to the remote socket.
V	Create (see page 1479)	Creates an instance of TUdpSocket (see page 1477).
V	Destroy (see page 1372)	Frees the memory associated with the TBaseSocket (see page 1370) object.
	Disconnect (see page 1403)	Disconnects the local socket from the remote socket.
	GetSocketAddr (see page 1419)	Builds a valid network socket address.
	GetThreadObject (see page 1404)	Returns a reference to the TThread object for the currently executing thread.
	LocalDomainName (see page 1419)	Returns the domain name for the local host.
	LocalHostAddr (see page 1420)	Returns the local IP address for the host machine.
	LocalHostName (see page 1420)	Returns the local host name.
	LookupHostAddr (see page 1420)	Looks up the host address based on the host name.
	LookupHostName (see page 1421)	Looks up the host name based on the IP address.

	LookupPort (see page 1421)	Returns the port number.
	LookupProtocol (see page 1421)	Returns the protocol number.
	MapDomain (see page 1375)	Maps the domain.
	MapSockType (see page 1375)	Maps the socket type.
V	Open (see page 1375)	Creates the socket connection.
V	Open (see page 1404)	Creates the socket connection.
	PeekBuf (see page 1376)	Reads the buffer.
	ReceiveBuf (see page 1376)	Reads data buffers to the socket.
	ReceiveFrom (see page 1422)	Receives a packet of information.
	ReceiveIn (see page 1379)	Reads delimited lines of data from the socket.
	Select (see page 1379)	Tests for socket activity.
	SendBuf (see page 1380)	Writes data buffers of size bufsize from the socket.
	SendIn (see page 1380)	Writes delimited lines of data to the socket.
	SendStream (see page 1380)	Writes a stream of data to the socket.
	SendTo (see page 1422)	Sends a packet of information.
	WaitForData (see page 1383)	Waits for the socket connection to be ready to transfer data.

Public Properties

	Name	Description
R	BytesReceived (see page 1386)	Returns the number of bytes received by the socket.
R	BytesSent (see page 1386)	Returns the number of bytes sent from the socket.
R	Connected (see page 1405)	Indicates whether the socket connection is open and available for communication with other machines.
	Domain (see page 1386)	Sets or returns the protocol family used by the socket.
	Domain (see page 1423)	Sets or returns the protocol family used by the socket.
R	Handle (see page 1387)	Provides access to the unique-4 byte identifier that represents the instance of the socket object.
	Protocol (see page 1387)	Sets or returns the protocol used by the socket.
	SockType (see page 1387)	Sets or returns the socket type.

Published Events

	Name	Description
	OnConnect (see page 1490)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1490)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1493)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1495)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1496)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1496)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1496)	Occurs just before a client socket writes information to the socket connection.

Published Properties

	Name	Description
	Active (see page 1480)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1482)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1482)	Specifies the name of the local system.
	LocalPort (see page 1483)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 1486)	Specifies the name of the remote system.
	RemotePort (see page 1488)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

24.1.11.2 TUdpSocket.Create Constructor

Creates an instance of TUdpSocket (see page 1477).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TUdpSocket(TComponent * AOwner);
```

Description

For C++, call Create indirectly, using the **new** keyword, to instantiate a TUdpSocket (see page 1477) object.

24.1.11.3 TUdpSocket Properties

The properties of the TUdpSocket class are listed here.

Published Properties

	Name	Description
	Active (see page 1480)	Indicates whether the socket connection is open and available for communication with other machines.
	BlockMode (see page 1482)	Sets or returns the blocking mode of the socket.
	LocalHost (see page 1482)	Specifies the name of the local system.
	LocalPort (see page 1483)	Specifies the ID number of the port used by the socket connection.
	RemoteHost (see page 1486)	Specifies the name of the remote system.
	RemotePort (see page 1488)	Specifies the ID number of the port used by the RemoteHost to form the socket connection.

24.1.11.3.1 TUdpSocket.Active Property

Indicates whether the socket connection is open and available for communication with other machines.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before attempting to use or change the socket connection, read Active to determine whether the connection is open and ready. For client sockets, setting Active opens or shuts down a socket connection to another machine. For server sockets, setting Active opens or shuts down a listening connection that makes the socket available for client requests.

At design time, set Active to **true** to make the socket open a connection when the application starts running (the default value is false). At runtime, use the Open or Close method to open or close the connection.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
```

```
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
```

```
    TcpClient1.Active := true;
    TcpClient1.SendStream(myStream);
    TcpClient1.Active := false;

    //free the stream
    myStream.Free;

  end;

  procedure TForm3.FormCreate(Sender: TObject);
  begin
    Edit1.Text := '0';
    Memo1.Lines.Clear;

    TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
    TcpClient1.RemoteHost := 'localhost';
    TcpClient1.Active := true;
  end;

  procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
    var DataLen: Integer);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
  end;
```

See Also

Close (see page 1373)

Open (see page 1375)

24.1.11.3.2 TUdpSocket.BlockMode Property

Sets or returns the blocking mode of the socket.

Pascal

```
property BlockMode: TSocketBlockMode;
```

C++

```
__property TSocketBlockMode BlockMode;
```

Description

Use BlockMode to set or return the blocking mode for the socket. By default, the socket is set to bmBlocking.

The TSocketBlockMode type defines the set of possible socket flags for the BlockMode property.

See Also

TSocketBlockMode (see page 1505)

OnReceive (see page 1393)

OnSend (see page 1394)

24.1.11.3.3 TUdpSocket.LocalHost Property

Specifies the name of the local system.

Pascal

```
property LocalHost: TSocketHost;
```

C++

```
__property TSocketHost LocalHost;
```

Description

Use LocalHost to get the name of the system that is running the application that uses the socket. LocalHost is a string containing the domain name and service of the local socket endpoint, such as http://www.wSite.Com.

Port numbers allow a single system, identified by the LocalHostAddr function, to host multiple connections simultaneously. Each combination of LocalPort and LocalHostAddr can only be bound to a single socket connection.

LocalHost provides an alias for the local IP address given in the LocalHostAddr function. Most Intranets provide host names for the IP addresses of systems on the net. For more information on defining a host name for a given system, contact your network administrator.

See Also

LocalPort (see page 1424)

RemoteHost (see page 1426)

LocalHostAddr (see page 1420)

LookupHostName (see page 1421)

LookupPort (see page 1421)

24.1.11.3.4 TUdpSocket.LocalPort Property

Specifies the ID number of the port used by the socket connection.

Pascal

```
property LocalPort: TSocketPort;
```

C++

```
__property TSocketPort LocalPort;
```

Description

Use LocalPort to determine the port number that is bound to the socket connection. For server endpoints of listening connections, LocalPort is the ID associated with the service the server socket provides. For client or server endpoints of connections to another socket, LocalPort is usually an arbitrary value picked when the connection was made.

Many values of LocalPort are associated by convention with a particular service such as FTP or HTTP.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
```

```
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}
```

```
procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

See Also

LocalHost (see page 1423)

RemotePort (see page 1428)

LookupPort (see page 1421)

24.1.11.3.5 TUdpSocket.RemoteHost Property

Specifies the name of the remote system.

Pascal

```
property RemoteHost: TSocketHost;
```

C++

```
__property TSocketHost RemoteHost;
```

Description

Use RemoteHost to get the name of the system on the other end of the socket connection. RemoteHost is a string containing the domain name and service of the remote endpoint of the socket connection, such as http://www.wSite.Com.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)
```

```
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

RemotePort (see page 1428)

LocalHostAddr (see page 1420)

LocalHostName (see page 1420)

LookupHostAddr (see page 1420)

LookupHostName (see page 1421)

24.1.11.3.6 TUdpSocket.RemotePort Property

Specifies the ID number of the port used by the RemoteHost to form the socket connection.

Pascal

```
property RemotePort: TSocketPort;
```

C++

```
__property TSocketPort RemotePort;
```

Description

Use RemotePort to determine the port number used by the socket on the other end of the socket connection.

Port numbers allow a single system, identified by the RemoteHost property, to host multiple connections simultaneously. Each combination of RemotePort and RemoteHost can only be bound to a single socket connection.

Many port numbers are associated by convention with a particular service such as ftp or http. To determine whether the value of RemotePort is associated with a specific service, use the LookupProtocol method.

Note: If a client socket requests a specific port to indicate a desired service, the value of RemotePort for the actual connection may differ from the requested port number. Most server sockets listen on the port associated with a service, but switch to an arbitrary available port number for forming connections.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);
```

```
    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
```

```
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

RemoteHost (see page 1426)

LookupPort (see page 1421)

LookupProtocol (see page 1421)

24.1.11.4 TUdpSocket Events

The events of the TUdpSocket class are listed here.

Published Events

	Name	Description
	OnConnect (see page 1490)	Occurs on client sockets just after the connection to the server is opened.
	OnCreateHandle (see page 1490)	Occurs when the handle to a socket becomes active.
	OnDestroyHandle (see page 1493)	Occurs when a socket is deactivated.
	OnDisconnect (see page 1495)	Occurs on client sockets just after the connection to the server is closed.
	OnError (see page 1496)	Occurs when the socket fails in making, using, or shutting down a connection.
	OnReceive (see page 1496)	Occurs after a socket receives data, but before it passes the data to the application.
	OnSend (see page 1496)	Occurs just before a client socket writes information to the socket connection.

24.1.11.4.1 TUdpSocket.OnConnect Event

Occurs on client sockets just after the connection to the server is opened.

Pascal

```
property OnConnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnConnect;
```

Description

Write an OnConnect event handler for a client socket to take specific action after the connection to a server socket has been established. Depending on the service, this may be the point when the socket should start reading or writing over the connection.

See Also

Connected (see page 1405)

OnDisconnect (see page 1406)

24.1.11.4.2 TUdpSocket.OnCreateHandle Event

Occurs when the handle to a socket becomes active.

Pascal

```
property OnCreateHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnCreateHandle;
```

Description

Write an OnCreateHandle event handler to take specific action when the handle to a socket becomes active.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
```

```
    Memo1.Lines.Add('--------------------');

  end;

  procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
  begin
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
  end;

  procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
  begin
    if Self.Memo1 <> nil then
      Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
  end;
```

See Also

Active (see page 1383)

Handle (see page 1387)

24.1.11.4.3 TUdpSocket.OnDestroyHandle Event

Occurs when a socket is deactivated.

Pascal

```
property OnDestroyHandle: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDestroyHandle;
```

Description

Write an OnDestroyHandle event handler to take specific action when the socket is deactivated.

C++ Examples:

```
/*
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
This example uses three buttons, one for activating the server, one
for deactivating it and the third for clearing the Memo.
*/
__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Memo1->Lines->Clear();

    //server initialization
    TcpServer1 = new TTcpServer(this);
    TcpServer1->OnAccept = TcpServer1Accept;
    TcpServer1->OnCreateHandle = TcpServer1CreateHandle;
    TcpServer1->OnDestroyHandle = TcpServer1DestroyHandle;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::TcpServer1Accept(TObject *Sender,
                                         TCustomIpClient *ClientSocket)
{
    byte a[10];
    ClientSocket->ReceiveBuf(a,10,0);
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Data added width " +
                      IntToStr(a[9]));
```

```
    for (int i=0; i < 9; i++)
    {
        Memo1->Lines->Add(IntToStr(a[i] + a[9]));
    }
    Memo1->Lines->Add("--------------------");

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    TcpServer1->LocalPort = DEFAULT_PORT;
    TcpServer1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    TcpServer1->Active = false;
}

//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
    Memo1->Lines->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1CreateHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Server started.");
}
//---------------------------------------------------------------------------

void __fastcall TForm3::TcpServer1DestroyHandle(TObject *Sender)
{
    TDateTime now = TDateTime::CurrentDateTime();
    if(this->Memo1)
    {
        Memo1->Lines->Add(DateTimeToStr(now) + " Server stopped.");
    }
}
//---------------------------------------------------------------------------
```

Delphi Examples:

```
{
This example demonstrated the use of TTcpServer in order to receive
a buffer (stream) and process it. It is to be used with the example
from TTcpClient section.
}

const DEFAULT_PORT = 2501;

{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject);
begin
  //connectiong the server
  TcpServer1.Active := true;
end;

procedure TForm3.Button2Click(Sender: TObject);
begin
  //disconnecting the server
  TcpServer1.Active := false;
end;
```

```
procedure TForm3.Button3Click(Sender: TObject);
begin
  //clearing the memo
  Memo1.Lines.Clear;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;

  //server initialization
  TcpServer1 := TTcpServer.Create(Self);
  TcpServer1.OnAccept := TCpServer1Accept;
  TcpServer1.OnCreateHandle := TcpServer1CreateHandle;
  TcpServer1.OnDestroyHandle := TcpServer1DestroyHandle;
  TcpServer1.LocalPort := IntToStr(DEFAULT_PORT);
end;

procedure TForm3.TcpServer1Accept(Sender: TObject;
  ClientSocket: TCustomIpClient);
var
  a : array[0..9] of byte;
  I : Cardinal;
begin

  ClientSocket.ReceiveBuf(a,10,0);
  Memo1.Lines.Add(DateTimeToStr(now) + ' Data added with ' +
                  IntToStr(a[9]));
  for I := 0 to 8 do
  begin
    Memo1.Lines.Add(IntToStr(a[I] + a[9]));
  end;
  Memo1.Lines.Add('--------------------');

end;

procedure TForm3.TcpServer1CreateHandle(Sender: TObject);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Server started.');
end;

procedure TForm3.TcpServer1DestroyHandle(Sender: TObject);
begin
  if Self.Memo1 <> nil then
    Memo1.Lines.Add(DateTimeToStr(now) + ' Server stopped.');
end;
```

24.1.11.4.4 TUdpSocket.OnDisconnect Event

Occurs on client sockets just after the connection to the server is closed.

Pascal

```
property OnDisconnect: TSocketNotifyEvent;
```

C++

```
__property TSocketNotifyEvent OnDisconnect;
```

Description

Write an OnDisconnect event handler for a client socket to take specific action just after the connection to a server socket is terminated.

See Also

Connected (see page 1405)

OnConnect (see page 1405)

24.1.11.4.5 TUdpSocket.OnError Event

Occurs when the socket fails in making, using, or shutting down a connection.

Pascal

```
property OnError: TSocketErrorEvent;
```

C++

```
__property TSocketErrorEvent OnError;
```

Description

Write an OnError event handler to respond to errors that arise with the socket connection. Set the **ErrorCode** parameter to 0 if the OnError event handler successfully deals with the error condition, to prevent an ESocketError exception from being raised.

See Also

ESocketError (see page 1369)

24.1.11.4.6 TUdpSocket.OnReceive Event

Occurs after a socket receives data, but before it passes the data to the application.

Pascal

```
property OnReceive: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnReceive;
```

Description

Write an OnReceive event handler to read from the socket connection. Use the OnReceive event to filter data before passing it back to the client application. If the socket is a blocking socket, use a TWinSocketStream object to read from the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual reading.

See Also

TWinSocketStream (see page 138)

BytesReceived (see page 1386)

ReceiveBuf (see page 1376)

ReceiveIn (see page 1379)

24.1.11.4.7 TUdpSocket.OnSend Event

Occurs just before a client socket writes information to the socket connection.

Pascal

```
property OnSend: TSocketDataEvent;
```

C++

```
__property TSocketDataEvent OnSend;
```

Description

Write an OnSend event handler to perform actions just prior to sending data. Use OnSend to monitor the data stream and

change it (for example, by removing linefeeds). If the socket is a blocking socket, use a TWinSocketStream object to write to the connection. Otherwise, use the methods of the **Socket** parameter to perform the actual writing.

C++ Examples:

```
/*
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
*/
#define DEFAULT_PORT 2501

__fastcall TForm3::TForm3(TComponent* Owner)
    : TForm(Owner)
{
    Edit1->Text = "0";
    Memo1->Lines->Clear();

    TcpClient1 = new TTcpClient(this);
    TcpClient1->OnSend = TcpClient1Send;
    TcpClient1->RemotePort = DEFAULT_PORT;
    TcpClient1->RemoteHost = "localhost";
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm3::Button1Click(TObject *Sender)
{
    //creating a stream
//  TMemoryStream* myStream = new TMemoryStream();
    std::auto_ptr<TMemoryStream> myStream(new TMemoryStream);
    for (byte i = 0; i < 9; i++)
    {
        myStream->WriteBuffer(&i,1);
    }
    int userValue = StrToIntDef(Edit1->Text, 0);
    myStream->WriteBuffer(&userValue, 1);

    //resetting the stream position
    myStream->Seek(0, 0);

    //sending the stream
    TcpClient1->Active = true;
    TcpClient1->SendStream(myStream.get());
    TcpClient1->Active = false;

    //free the stream
//  delete myStream;
}

//---------------------------------------------------------------------------

void __fastcall TForm3::TcpClient1Send(TObject *Sender, PWideChar Buf, int &DataLen)

{
    TDateTime now = TDateTime::CurrentDateTime();
    Memo1->Lines->Add(DateTimeToStr(now) + " Sent data with " + Edit1->Text);
}
```

Delphi Examples:

```
{
    This example demontrates the use of SendStream method. It is to be
    used with the example from TTcpServer section.
}
const DEFAULT_PORT = 2501;

procedure TForm3.Button1Click(Sender: TObject);
```

```
var
  myStream : TMemoryStream;
  I : Cardinal;
  userValue : Integer;
begin
  //creating a stream
  myStream := TMemoryStream.Create();
  for I := 0 to 8 do
  begin
    myStream.WriteBuffer(I,1);
  end;
  userValue := StrToIntDef(Edit1.Text, 0);
  myStream.WriteBuffer(userValue,1);

  //resetting the stream position
  myStream.Seek(0,0);

  //sending the stream
  TcpClient1.Active := true;
  TcpClient1.SendStream(myStream);
  TcpClient1.Active := false;

  //free the stream
  myStream.Free;

end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Edit1.Text := '0';
  Memo1.Lines.Clear;

  TcpClient1.RemotePort := IntToStr(DEFAULT_PORT);
  TcpClient1.RemoteHost := 'localhost';
  TcpClient1.Active := true;
end;

procedure TForm3.TcpClient1Send(Sender: TObject; Buf: PWideChar;
  var DataLen: Integer);
begin
  Memo1.Lines.Add(DateTimeToStr(now) + ' Sent data with ' + Edit1.Text);
end;
```

See Also

TWinSocketStream (see page 138)

SendBuf (see page 1380)

SendIn (see page 1380)

24.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TServerSocketBlockMode (see page 1500)	Defines the set of possible socket flags for block mode.
	TSocketDomain (see page 1500)	Set of possible protocol family types used by a socket.
	TSocketType (see page 1502)	The TSocketType type defines the set of possible socket types.

Records

	Name	Description
	TIPHeader (see page 1499)	This is record Sockets.TIPHeader.

24.2.1 Sockets.TIPHeader Record

Pascal

```
TIPHeader = packed record
  iph_verlen: byte;
  iph_tos: byte;
  iph_length: word;
  iph_id: word;
  iph_offset: word;
  iph_ttl: byte;
  iph_protocol: byte;
  iph_xsum: word;
  iph_src: longword;
  iph_dest: longword;
end;
```

C++

```
struct TIPHeader {
  byte iph_verlen;
  byte iph_tos;
  word iph_length;
  word iph_id;
  word iph_offset;
  byte iph_ttl;
  byte iph_protocol;
  word iph_xsum;
  longword iph_src;
  longword iph_dest;
};
```

File

Sockets

Description

This is record Sockets.TIPHeader.

24.2.2 Sockets.TServerSocketBlockMode Enumeration

Defines the set of possible socket flags for block mode.

Pascal

```
TServerSocketBlockMode = (
  bmBlocking,
  bmNonBlocking,
  bmThreadBlocking
);
```

C++

```
enum TServerSocketBlockMode {
  bmBlocking,
  bmNonBlocking,
  bmThreadBlocking
};
```

File

Sockets

Description

The following table lists the possible values for the TServerSocketBlockMode type and the meaning corresponding to each value:

Value	Socket state
bmBlocking	The socket is set to block mode.
bmNonBlocking	The socket is set to non-blocking mode.
bmThreadBlocking	A separate thread is created so that listening can be performed in the background without blocking the entire server application.

24.2.3 Sockets.TSocketDomain Enumeration

Set of possible protocol family types used by a socket.

Pascal

```
TSocketDomain = (
  pfUnspec,
  pfUnix,
  pfInet,
  pfImpLink,
  pfPup,
  pfChaos,
  pfIpx,
  pfNs,
  pfIso,
  pfOsi,
  pfEcma,
  pfDataKit,
  pfCcitt,
  pfSna,
  pfDecNet,
```

```
    pfDli,
    pfLat,
    pfHylink,
    pfAppleTalk,
    pfVoiceView,
    pfFireFox,
    pfUnknown1,
    pfBan,
    pfMax
  );
```

C++

```
  enum TSocketDomain {
    pfUnspec,
    pfUnix,
    pfInet,
    pfImpLink,
    pfPup,
    pfChaos,
    pfIpx,
    pfNs,
    pfIso,
    pfOsi,
    pfEcma,
    pfDataKit,
    pfCcitt,
    pfSna,
    pfDecNet,
    pfDli,
    pfLat,
    pfHylink,
    pfAppleTalk,
    pfVoiceView,
    pfFireFox,
    pfUnknown1,
    pfBan,
    pfMax
  };
```

File

Sockets

Description

The following table lists the possible values for the TSocketDomain type and the meaning corresponding to each value:

Value	Description
pfAppleTalk	AppleTalk DDP
pfBan	Banyan Vines
pfCcitt	CCIT (International Telegraph and Telephone Consultative)
pfChaos	
pfDataKit	DataKit
pfDecNet	Reserved for DECnet project
pfDli	
pfEcma	EcmaScript
pfFireFox	FireFox
pfHylink	Hyperlink
pfImpLink	Imported link

pfInet	IP protocol family
pfIpx	Novell Interner Protocol
pfIso	International Standards Organization (ISO)
pfLat	
pfMax	Maximum protocol family number
pfNs	Network Services Protocol (used with DECnet)
pfOsi	Open Systems Interconnection (OSI)
pfPup	PUP protocol
pfSna	Linux SNA project
pfUnix	Local to host (old BSD name for local)
pfUnknown1	Unknown
pfUnspec	Unspecified
pfVoiceView	VoiceView

24.2.4 Sockets.TSocketType Enumeration

The TSocketType type defines the set of possible socket types.

Pascal

```
TSocketType = (
  stStream,
  stDgram,
  stRaw,
  stRdm,
  stSeqPacket
);
```

C++

```
enum TSocketType {
  stStream,
  stDgram,
  stRaw,
  stRdm,
  stSeqPacket
};
```

File

Sockets

Description

The following table lists the possible values for the TSocketType type and the meaning corresponding to each value:

Value	Description
stDgram	Single datagram packets, not used by IP. Non-reliable.
stRaw	The socket sends and receives raw datagram packets. Used by IP.
stRdm	Reliably delivered message.
stSeqPacket	Sequenced packet stream sends and receives any type of IP packet.

stStream	Reliable streaming data packets. Used by TCP.

24.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetThreadEvent (see page 1504)	This is type Sockets.TGetThreadEvent.
TSocketAcceptEvent (see page 1504)	This is type Sockets.TSocketAcceptEvent.
TSocketBlockMode (see page 1505)	Defines the set of possible flags for blocking mode of the socket.
TSocketDataEvent (see page 1505)	This is type Sockets.TSocketDataEvent.
TSocketErrorEvent (see page 1505)	This is type Sockets.TSocketErrorEvent.
TSocketHost (see page 1506)	This is type Sockets.TSocketHost.
TSocketNotifyEvent (see page 1506)	This is type Sockets.TSocketNotifyEvent.
TSocketPort (see page 1506)	This is type Sockets.TSocketPort.
TSocketProtocol (see page 1507)	TSocketProtocol is the code for a socket protocol.

24.3.1 Sockets.TGetThreadEvent Type

Pascal

```
TGetThreadEvent = procedure (Sender: TObject; var ClientSocketThread: TClientSocketThread) of object;
```

C++

```
(Sender: TObject; var ClientSocketThread: TClientSocketThread) ( TGetThreadEvent)();
```

File

Sockets

Description

This is type Sockets.TGetThreadEvent.

24.3.2 Sockets.TSocketAcceptEvent Type

Pascal

```
TSocketAcceptEvent = procedure (Sender: TObject; ClientSocket: TCustomIpClient) of object;
```

C++

```
(Sender: TObject; ClientSocket: TCustomIpClient) ( TSocketAcceptEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketAcceptEvent.

24.3.3 Sockets.TSocketBlockMode Type

Defines the set of possible flags for blocking mode of the socket.

Pascal

```
TSocketBlockMode = bmBlocking..bmNonBlocking;
```

C++

```
bmBlocking..bmNonBlocking TSocketBlockMode;
```

File

Sockets

Description

The following table describes TSocketBlockMode values.

Value	Socket state
bmBlocking	The socket is set to block mode.
bmNonBlocking	The socket is set to non-blocking mode.

24.3.4 Sockets.TSocketDataEvent Type

Pascal

```
TSocketDataEvent = procedure (Sender: TObject; Buf: pchar; var DataLen: Integer) of object;
```

C++

```
(Sender: TObject; Buf: pchar; var DataLen: Integer) ( TSocketDataEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketDataEvent.

24.3.5 Sockets.TSocketErrorEvent Type

Pascal

```
TSocketErrorEvent = procedure (Sender: TObject; SocketError: Integer) of object;
```

C++

```
(Sender: TObject; SocketError: Integer) ( TSocketErrorEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketErrorEvent.

24.3.6 Sockets.TSocketHost Type

Pascal

```
TSocketHost = type string;
```

C++

```
type string TSocketHost;
```

File

Sockets

Description

This is type Sockets.TSocketHost.

24.3.7 Sockets.TSocketNotifyEvent Type

Pascal

```
TSocketNotifyEvent = procedure (Sender: TObject) of object;
```

C++

```
(Sender: TObject) ( TSocketNotifyEvent)();
```

File

Sockets

Description

This is type Sockets.TSocketNotifyEvent.

24.3.8 Sockets.TSocketPort Type

Pascal

```
TSocketPort = type string;
```

C++

```
type string TSocketPort;
```

File

Sockets

Description

This is type Sockets.TSocketPort.

24.3.9 Sockets.TSocketProtocol Type

TSocketProtocol is the code for a socket protocol.

Pascal

```
TSocketProtocol = Word;
```

C++

```
Word TSocketProtocol;
```

File

Sockets

Description

TSocketProtocol is a number that represents a network protocol such as IP, TCP, or UDP.

24.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CRLF (see page 1508)	This is constant Sockets.CRLF.

24.4.1 Sockets.CRLF Constant

Pascal

```
CRLF = #13#10;
```

C++

```
CRLF = #13#10;
```

File

Sockets

Description

This is constant Sockets.CRLF.

25 SqlExpr Namespace

This is namespace SqlExpr.

Topics

Name	Description
Classes (see page 1512)	The following table lists classes in this documentation.
Functions (see page 1696)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1699)	The following table lists structs, records, enums in this documentation.
Types (see page 1706)	The following table lists types in this documentation.
Constants (see page 1711)	The following table lists constants in this documentation.

Classes

	Name	Description
	SPParamDesc (see page 1512)	SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.
	TConnectionData (see page 1515)	This is class SqlExpr.TConnectionData.
	TCustomSQLDataSet (see page 1520)	TCustomSQLDataSet is the base class for all dbExpress dataset components.
	TFLDDesc (see page 1570)	This is class SqlExpr.TFLDDesc.
	TSQLBlobStream (see page 1577)	TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TSQLConnection (see page 1580)	TSQLConnection encapsulates a dbExpress connection to a database server.
	TSQLDataLink (see page 1628)	TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.
	TSQLDataSet (see page 1631)	TSQLDataSet represents data retrieved using dbExpress.
	TSQLMonitor (see page 1643)	TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.
	TSQLQuery (see page 1654)	TSQLQuery represents a query that is executed using dbExpress.
	TSqlServerMethod (see page 1665)	TDataSet used to call DataSnap server methods.
	TSQLStoredProc (see page 1671)	TSQLStoredProc represents a stored procedure that is executed using dbExpress.
	TSQLTable (see page 1680)	TSQLTable represents a database table that is accessed using dbExpress.
	TTransactionItem (see page 1694)	This is class SqlExpr.TTransactionItem.

Constants

Name	Description
BlobTypeMap (see page 1711)	This is constant SqlExpr.BlobTypeMap.
DataTypeMap (see page 1712)	This is constant SqlExpr.DataTypeMap.

DefaultCursor (see page 1712)	This is constant SqlExpr.DefaultCursor.
DefaultMaxBlobSize (see page 1713)	This is constant SqlExpr.DefaultMaxBlobSize.
DefaultRowsetSize (see page 1713)	DefaultRowsetSize represents the default row buffering size.
eSQLAutoIncr (see page 1713)	This is constant SqlExpr.eSQLAutoIncr.
eSQLDefault (see page 1714)	This is constant SqlExpr.eSQLDefault.
eSQLFunction (see page 1714)	This is constant SqlExpr.eSQLFunction.
eSQLLocal (see page 1714)	This is constant SqlExpr.eSQLLocal.
eSQLNonUnique (see page 1715)	This is constant SqlExpr.eSQLNonUnique.
eSQLPackage (see page 1715)	This is constant SqlExpr.eSQLPackage.
eSQLPrimaryKey (see page 1715)	This is constant SqlExpr.eSQLPrimaryKey.
eSQLProcedure (see page 1716)	This is constant SqlExpr.eSQLProcedure.
eSQLRowId (see page 1716)	This is constant SqlExpr.eSQLRowId.
eSQLRowVersion (see page 1716)	This is constant SqlExpr.eSQLRowVersion.
eSQLSynonym (see page 1717)	This is constant SqlExpr.eSQLSynonym.
eSQLSysProcedure (see page 1717)	This is constant SqlExpr.eSQLSysProcedure.
eSQLSystemTable (see page 1717)	This is constant SqlExpr.eSQLSystemTable.
eSQLTable (see page 1718)	This is constant SqlExpr.eSQLTable.
eSQLTempTable (see page 1718)	This is constant SqlExpr.eSQLTempTable.
eSQLUnique (see page 1718)	This is constant SqlExpr.eSQLUnique.
eSQLView (see page 1719)	This is constant SqlExpr.eSQLView.
FldSubTypeMap (see page 1719)	This is constant SqlExpr.FldSubTypeMap.
FldTypeMap (see page 1719)	This is constant SqlExpr.FldTypeMap.
HourGlassCursor (see page 1720)	This is constant SqlExpr.HourGlassCursor.
SAnd (see page 1721)	This is constant SqlExpr.SAnd.
SOrderBy (see page 1721)	This is constant SqlExpr.SOrderBy.
SParam (see page 1721)	This is constant SqlExpr.SParam.
SSelect (see page 1722)	This is constant SqlExpr.SSelect.
SSelectSpaces (see page 1722)	This is constant SqlExpr.SSelectSpaces.
SSelectStar (see page 1722)	This is constant SqlExpr.SSelectStar.
SSelectStarFrom (see page 1723)	This is constant SqlExpr.SSelectStarFrom.
SUB_TYPE_MEMO (see page 1723)	This is constant SqlExpr.SUB_TYPE_MEMO.
SWhere (see page 1723)	This is constant SqlExpr.SWhere.
TErrorMessageSize (see page 1724)	This is constant SqlExpr.TErrorMessageSize.

Functions

	Name	Description
	FreeProcParams (see page 1696)	Frees a list of procedure parameters that is allocated by GetProcedureParams.
	GetConnectionNames (see page 1696)	This is function SqlExpr.GetConnectionNames.
	GetConnectionRegistryFile (see page 1697)	This is function SqlExpr.GetConnectionRegistryFile.
	GetDelegateDriverNames (see page 1697)	This is function SqlExpr.GetDelegateDriverNames.
	GetDriverNames (see page 1697)	Populates a string list with the names of all available dbExpress drivers.
	GetDriverRegistryFile (see page 1698)	This is function SqlExpr.GetDriverRegistryFile.
	LoadParamListItems (see page 1698)	Fills a TParams object with information from a list of parameter descriptors.

Structs, Records, Enums

	Name	Description
	EConnectFlag (see page 1699)	This is record SqlExpr.EConnectFlag.
	TConnectionState (see page 1700)	TConnectionState describes the state of a TSQLConnection (see page 1580) component.
	TConnectionUserType (see page 1700)	This is record SqlExpr.TConnectionUserType.
	TSchemaType (see page 1701)	TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.
	TSQLExceptionType (see page 1702)	TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.
	TSQLSchemaInfo (see page 1703)	This is record SqlExpr.TSQLSchemaInfo.

	TTableScope (see page 1703)	TTableScopes (see page 1710) and TTableScopess indicate the types of tables for which to fetch metadata.
	TTransactionDesc (see page 1704)	This is record SqlExpr.TTransactionDesc.
	TTransIsolationLevel (see page 1704)	This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Types

Name	Description
SQLResult (see page 1706)	SQLResult is the return type for dbExpress function calls.
TFieldDescList (see page 1707)	This is type SqlExpr.TFieldDescList.
TFieldList (see page 1707)	This is type SqlExpr.TFieldList.
TGetDriverFunc (see page 1708)	This is type SqlExpr.TGetDriverFunc.
TLocale (see page 1708)	This is type SqlExpr.TLocale.
TLocaleCode (see page 1708)	TLocaleCode identifies a locale, which imposes a sort order on data.
TParseInsertSqlEvent (see page 1709)	This is type SqlExpr.TParseInsertSqlEvent.
TParseSqlEvent (see page 1709)	This is type SqlExpr.TParseSqlEvent.
TSQLConnectionLoginEvent (see page 1709)	This is type SqlExpr.TSQLConnectionLoginEvent.
TTableScopes (see page 1710)	This is type SqlExpr.TTableScopes.
TTraceEvent (see page 1710)	This is type SqlExpr.TTraceEvent.
TTraceLogEvent (see page 1710)	This is type SqlExpr.TTraceLogEvent.

25.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	SPParamDesc (see page 1512)	SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.
	TConnectionData (see page 1515)	This is class SqlExpr.TConnectionData.
	TCustomSQLDataSet (see page 1520)	TCustomSQLDataSet is the base class for all dbExpress dataset components.
	TFLDDesc (see page 1570)	This is class SqlExpr.TFLDDesc.
	TSQLBlobStream (see page 1577)	TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.
	TSQLConnection (see page 1580)	TSQLConnection encapsulates a dbExpress connection to a database server.
	TSQLDataLink (see page 1628)	TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.
	TSQLDataSet (see page 1631)	TSQLDataSet represents data retrieved using dbExpress.
	TSQLMonitor (see page 1643)	TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.
	TSQLQuery (see page 1654)	TSQLQuery represents a query that is executed using dbExpress.
	TSqlServerMethod (see page 1665)	TDataSet used to call DataSnap server methods.
	TSQLStoredProc (see page 1671)	TSQLStoredProc represents a stored procedure that is executed using dbExpress.
	TSQLTable (see page 1680)	TSQLTable represents a database table that is accessed using dbExpress.
	TTransactionItem (see page 1694)	This is class SqlExpr.TTransactionItem.

25.1.1 SPParamDesc Class

SPParamDesc describes a parameter to a stored procedure that is accessed using dbExpress.

Class Hierarchy

SqlExpr.SPParamDesc

Pascal

```
SPParamDesc = class;
```

C++

```
class SPParamDesc;
```

File

SqlExpr

Description

PSPParamDesc is a pointer to a parameter descriptor. SPParamDesc is a parameter descriptor that describes a parameter for a stored procedure. SPParamDesc includes the following members:

Member	Meaning
iParamNum (see page 1514)	The ordinal position of the parameter.

szName (see page 1515)	The name of the parameter
iArgType (see page 1513)	The type of the parameter (input, output, and so on)
iDataType (see page 1514)	The data type of the parameter
iUnits1 (see page 1514)	The size for the datatype (number of characters, digits, etc)
iUnits2 (see page 1515)	Number of decimal places if datatype is BCD
iLen (see page 1514)	The number of bytes needed to store a parameter value

See Also

TSQLConnection (see page 1580)

25.1.1.1 SPParamDesc Members

The following tables list the members exposed by SPParamDesc.

Public Fields

	Name	Description
	iArgType (see page 1513)	This is iArgType, a member of class SPParamDesc.
	iDataType (see page 1514)	This is iDataType, a member of class SPParamDesc.
	iLen (see page 1514)	This is iLen, a member of class SPParamDesc.
	iParamNum (see page 1514)	This is iParamNum, a member of class SPParamDesc.
	iUnits1 (see page 1514)	This is iUnits1, a member of class SPParamDesc.
	iUnits2 (see page 1515)	This is iUnits2, a member of class SPParamDesc.
	szName (see page 1515)	This is szName, a member of class SPParamDesc.

25.1.1.2 SPParamDesc Fields

The fields of the SPParamDesc class are listed here.

Public Fields

	Name	Description
	iArgType (see page 1513)	This is iArgType, a member of class SPParamDesc.
	iDataType (see page 1514)	This is iDataType, a member of class SPParamDesc.
	iLen (see page 1514)	This is iLen, a member of class SPParamDesc.
	iParamNum (see page 1514)	This is iParamNum, a member of class SPParamDesc.
	iUnits1 (see page 1514)	This is iUnits1, a member of class SPParamDesc.
	iUnits2 (see page 1515)	This is iUnits2, a member of class SPParamDesc.
	szName (see page 1515)	This is szName, a member of class SPParamDesc.

25.1.1.2.1 SPParamDesc.iArgType Field

Pascal

```
iArgType: TParamType;
```

C++

```
TParamType iArgType;
```

Description

This is iArgType, a member of class SPParamDesc.

25.1.1.2.2 SPParamDesc.iDataType Field

Pascal

```
iDataType: TFieldType;
```

C++

```
TFieldType iDataType;
```

Description

This is iDataType, a member of class SPParamDesc.

25.1.1.2.3 SPParamDesc.iLen Field

Pascal

```
iLen: LongWord;
```

C++

```
LongWord iLen;
```

Description

This is iLen, a member of class SPParamDesc.

25.1.1.2.4 SPParamDesc.iParamNum Field

Pascal

```
iParamNum: Word;
```

C++

```
Word iParamNum;
```

Description

This is iParamNum, a member of class SPParamDesc.

25.1.1.2.5 SPParamDesc.iUnits1 Field

Pascal

```
iUnits1: SmallInt;
```

C++

```
SmallInt iUnits1;
```

Description

This is iUnits1, a member of class SPParamDesc.

25.1.1.2.6 SPParamDesc.iUnits2 Field

Pascal

```
iUnits2: SmallInt;
```

C++

```
SmallInt iUnits2;
```

Description

This is iUnits2, a member of class SPParamDesc.

25.1.1.2.7 SPParamDesc.szName Field

Pascal

```
szName: string;
```

C++

```
UnicodeString szName;
```

Description

This is szName, a member of class SPParamDesc.

25.1.2 TConnectionData Class

Class Hierarchy

Pascal

```
TConnectionData = class(TPersistent);
```

C++

```
class TConnectionData : public TPersistent;
```

File

SqlExpr

Description

This is class SqlExpr.TConnectionData.

25.1.2.1 TConnectionData Members

The following tables list the members exposed by TConnectionData.

Protected Methods

	Name	Description
	GetFullParams (see page 1518)	This is GetFullParams, a member of class TConnectionData.

Public Methods

	Name	Description
	AddProperties (see page 1517)	This is AddProperties, a member of class TConnectionData.
	BeginUpdate (see page 1518)	This is BeginUpdate, a member of class TConnectionData.
	Create (see page 1516)	This is the overview for the Create constructor overload.
V	Destroy (see page 1517)	Destroys the TPersistent instance and frees its memory.
	EndUpdate (see page 1518)	This is EndUpdate, a member of class TConnectionData.
	RefreshProperties (see page 1518)	This is RefreshProperties, a member of class TConnectionData.
	ReloadProperties (see page 1518)	This is ReloadProperties, a member of class TConnectionData.
	UpdateProperties (see page 1519)	This is UpdateProperties, a member of class TConnectionData.

Published Properties

	Name	Description
	ConnectionName (see page 1519)	This is ConnectionName, a member of class TConnectionData.
	DelegateConnection (see page 1519)	This is DelegateConnection, a member of class TConnectionData.
	DriverName (see page 1520)	This is DriverName, a member of class TConnectionData.
R	IsModified (see page 1520)	This is IsModified, a member of class TConnectionData.
	Properties (see page 1520)	This is Properties, a member of class TConnectionData.

25.1.2.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TConnectionData.Create (TConnectionData) (see page 1516)	This is Create, a member of class TConnectionData.
	TConnectionData.Create (TSQLConnection) (see page 1516)	This is Create, a member of class TConnectionData.

25.1.2.2.1 TConnectionData.Create Constructor (TConnectionData)

Pascal

```
constructor Create(AParentData: TConnectionData); overload;
```

C++

```
__fastcall TConnectionData(TConnectionData AParentData);
```

Description

This is Create, a member of class TConnectionData.

25.1.2.2.2 TConnectionData.Create Constructor (TSQLConnection)

Pascal

```
constructor Create(AConnection: TSQLConnection); overload;
```

C++

```
__fastcall TConnectionData(TSQLConnection AConnection);
```

Description

This is Create, a member of class TConnectionData.

25.1.2.3 TConnectionData.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConnectionData();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

System::TObject::Destroy (see page 3356)

25.1.2.4 TConnectionData Methods

The methods of the TConnectionData class are listed here.

Protected Methods

	Name	Description
	GetFullParams (see page 1518)	This is GetFullParams, a member of class TConnectionData.

Public Methods

	Name	Description
	AddProperties (see page 1517)	This is AddProperties, a member of class TConnectionData.
	BeginUpdate (see page 1518)	This is BeginUpdate, a member of class TConnectionData.
	EndUpdate (see page 1518)	This is EndUpdate, a member of class TConnectionData.
	RefreshProperties (see page 1518)	This is RefreshProperties, a member of class TConnectionData.
	ReloadProperties (see page 1518)	This is ReloadProperties, a member of class TConnectionData.
	UpdateProperties (see page 1519)	This is UpdateProperties, a member of class TConnectionData.

25.1.2.4.1 TConnectionData.AddProperties Method

Pascal

```
procedure AddProperties(NewProperties: TStrings);
```

C++

```
__fastcall AddProperties(TStrings NewProperties);
```

Description

This is AddProperties, a member of class TConnectionData.

25.1.2.4.2 TConnectionData.BeginUpdate Method

Pascal

```
function BeginUpdate: Boolean;
```

C++

```
__fastcall Boolean BeginUpdate();
```

Description

This is BeginUpdate, a member of class TConnectionData.

25.1.2.4.3 TConnectionData.EndUpdate Method

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

This is EndUpdate, a member of class TConnectionData.

25.1.2.4.4 TConnectionData.GetFullParams Method

Pascal

```
procedure GetFullParams(Params: TStrings);
```

C++

```
__fastcall GetFullParams(TStrings Params);
```

Description

This is GetFullParams, a member of class TConnectionData.

25.1.2.4.5 TConnectionData.RefreshProperties Method

Pascal

```
procedure RefreshProperties;
```

C++

```
__fastcall RefreshProperties();
```

Description

This is RefreshProperties, a member of class TConnectionData.

25.1.2.4.6 TConnectionData.ReloadProperties Method

Pascal

```
procedure ReloadProperties;
```

C++

```
__fastcall ReloadProperties();
```

Description

This is ReloadProperties, a member of class TConnectionData.

25.1.2.4.7 TConnectionData.UpdateProperties Method

Pascal

```
procedure UpdateProperties(NewProperties: TStrings);
```

C++

```
__fastcall UpdateProperties(TStrings NewProperties);
```

Description

This is UpdateProperties, a member of class TConnectionData.

25.1.2.5 TConnectionData Properties

The properties of the TConnectionData class are listed here.

Published Properties

	Name	Description
	ConnectionName (see page 1519)	This is ConnectionName, a member of class TConnectionData.
	DelegateConnection (see page 1519)	This is DelegateConnection, a member of class TConnectionData.
	DriverName (see page 1520)	This is DriverName, a member of class TConnectionData.
R	IsModified (see page 1520)	This is IsModified, a member of class TConnectionData.
	Properties (see page 1520)	This is Properties, a member of class TConnectionData.

25.1.2.5.1 TConnectionData.ConnectionName Property

Pascal

```
property ConnectionName: string;
```

C++

```
__property UnicodeString ConnectionName;
```

Description

This is ConnectionName, a member of class TConnectionData.

25.1.2.5.2 TConnectionData.DelegateConnection Property

Pascal

```
property DelegateConnection: TConnectionData;
```

C++

```
__property TConnectionData DelegateConnection;
```

Description

This is DelegateConnection, a member of class TConnectionData.

25.1.2.5.3 TConnectionData.DriverName Property

Pascal

```
property DriverName: string;
```

C++

```
__property UnicodeString DriverName;
```

Description

This is DriverName, a member of class TConnectionData.

25.1.2.5.4 TConnectionData.IsModified Property

Pascal

```
property IsModified: Boolean;
```

C++

```
__property Boolean IsModified;
```

Description

This is IsModified, a member of class TConnectionData.

25.1.2.5.5 TConnectionData.Properties Property

Pascal

```
property Properties: TDBXProperties;
```

C++

```
__property TDBXProperties Properties;
```

Description

This is Properties, a member of class TConnectionData.

25.1.3 TCustomSQLDataSet Class

TCustomSQLDataSet is the base class for all dbExpress dataset components.

Class Hierarchy

Pascal

```
TCustomSQLDataSet = class(TWideDataSet);
```

C++

```
class TCustomSQLDataSet : public TWideDataSet;
```

File

SqlExpr

Description

TCustomSQLDataSet encapsulates the properties, events, and methods for working with data accessed through dbExpress. Applications should not instantiate TCustomSQLDataSet objects. Instead, instantiate one of the descendant classes: TSQLDataSet (see page 1631), TSQLTable (see page 1680), TSQLQuery (see page 1654), or TSQLStoredProc (see page 1671).

TCustomSQLDataSet and its descendants represent unidirectional datasets. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, they have the following limitations:

There is no searching or navigation support except for moving to the first record or moving to the next record.

There is no built-in editing support. The data underlying an SQL dataset can only be edited by explicitly creating an SQL UPDATE command or by connecting the SQL dataset to a client dataset using a provider.

There is no support for filtering records or for lookup fields.

See Also

TSQLDataSet (see page 1631)

TSQLQuery (see page 1654)

TSQLStoredProc (see page 1671)

TSQLTable (see page 1680)

TSQLConnection (see page 1580)

TClientDataSet

TDataSetProvider

25.1.3.1 TCustomSQLDataSet Members

The following tables list the members exposed by TCustomSQLDataSet.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1529)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.

	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1539)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
V	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
V	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1555)	Indicates the meaning of the CommandText (see page 1554) property.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1557)	Links the SQL dataset to another (master) dataset.
	DbxCommandType (see page 1557)	This is DbxCommandType, a member of class TCustomSQLDataSet.

R	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	MaxBlobSize (see page 1559)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 1561)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	Params (see page 1562)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1564)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1555) is ctTable.

Public Methods

	Name	Description
V	Create (see page 1524)	This is the overview for the Create constructor overload.
V	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
V	Destroy (see page 1525)	Destroys the TCustomSQLDataSet (see page 1520) instance.
V	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
V	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
V	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.
V	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
V	PrepareStatement (see page 1540)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1554)	Specifies the command the dataset executes.
	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	SQLConnection (see page 1566)	Specifies the SQL connection component that connects the dataset to a database server.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.

	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Published Properties

	Name	Description
	Active (see page 1552)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1558)	Specifies whether the SQL dataset fetches metadata information along with the data.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 1565)	Specifies the name of the database schema.

25.1.3.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TCustomSQLDataSet.Create (TComponent) (see page 1524)	Creates an instance of a TCustomSQLDataSet (see page 1520) component.
	TCustomSQLDataSet.Create (TComponent, TDBXReader, Boolean) (see page 1524)	Creates an instance of a TCustomSQLDataSet (see page 1520) component.

25.1.3.2.1 TCustomSQLDataSet.Create Constructor (TComponent)

Creates an instance of a TCustomSQLDataSet (see page 1520) component.

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
constructor Create(AOwner: TComponent; DBXReader: TDBXReader; AOwnsInstance: Boolean);
overload; reintroduce;
```

C++

```
virtual __fastcall TCustomSQLDataSet(TComponent * AOwner);
__fastcall TCustomSQLDataSet(TComponent * AOwner, TDBXReader DBXReader, Boolean AOwnsInstance);
```

Description

Call Create to instantiate an SQL dataset component at runtime. SQL datasets placed in forms or data modules at design time are created automatically.

TCustomSQLDataSet (see page 1520) is rarely used directly in applications because many of its properties are protected. Instead, applications instantiate TCustomSQLDataSet (see page 1520) descendants such as TSQLDataSet (see page 1631), TSQLTable (see page 1680), TSQLQuery (see page 1654), or TSQLStoredProc (see page 1671).

AOwner is the component that is responsible for freeing the dataset. It becomes the value of the Owner property.

See Also

Owner

Create

TSQLDataSet (see page 1631)

TSQLStoredProc (see page 1671)

TSQLTable (see page 1680)

TSQLQuery (see page 1654)

25.1.3.3 TCustomSQLDataSet.Destroy Destructor

Destroys the TCustomSQLDataSet (see page 1520) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomSQLDataSet();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner

Create (see page 1524)

System::TObject::Free (see page 3372)

25.1.3.4 TCustomSQLDataSet Methods

The methods of the TCustomSQLDataSet class are listed here.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1529)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.

	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1539)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
V	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
V	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Public Methods

	Name	Description
	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
	PrepareStatement (see page 1540)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

25.1.3.4.1 TCustomSQLDataSet.AddFieldDesc Method

Pascal

```
procedure AddFieldDesc(FieldDescs: TFieldDescList; DescNo: Integer; var FieldID: Integer;
RequiredFields: TBits; FieldDefs: TFieldDefs);
```

C++

```
__fastcall AddFieldDesc(TFieldDescList FieldDescs, int DescNo, int FieldID, TBits
RequiredFields, TFieldDefs FieldDefs);
```

Description

This is AddFieldDesc, a member of class TCustomSQLDataSet.

25.1.3.4.2 TCustomSQLDataSet.AddIndexDefs Method

Pascal

```
procedure AddIndexDefs(SourceDS: TCustomSQLDataSet; IndexName: string = '');
```

C++

```
__fastcall AddIndexDefs(TCustomSQLDataSet SourceDS, UnicodeString IndexName = '');
```

Description

This is AddIndexDefs, a member of class TCustomSQLDataSet.

25.1.3.4.3 TCustomSQLDataSet.CheckPrepareError Method

Pascal

```
procedure CheckPrepareError;
```

C++

```
__fastcall CheckPrepareError();
```

Description

This is CheckPrepareError, a member of class TCustomSQLDataSet.

25.1.3.4.4 TCustomSQLDataSet.ClearIndexDefs Method

Pascal

```
procedure ClearIndexDefs;
```

C++

```
__fastcall ClearIndexDefs();
```

Description

This is ClearIndexDefs, a member of class TCustomSQLDataSet.

25.1.3.4.5 TCustomSQLDataSet.CloseCursor Method

Pascal

```
procedure CloseCursor; override;
```

C++

```
virtual __fastcall CloseCursor();
```

Description

This is CloseCursor, a member of class TCustomSQLDataSet.

25.1.3.4.6 TCustomSQLDataSet.CloseStatement Method

Pascal

```
procedure CloseStatement;
```

C++

```
__fastcall CloseStatement();
```

Description

This is CloseStatement, a member of class TCustomSQLDataSet.

25.1.3.4.7 TCustomSQLDataSet.CreateBlobStream Method

Creates a blob stream for a Binary large object (BLOB) field in the dataset.

Pascal

```
function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override;
```

C++

```
virtual __fastcall TStream CreateBlobStream(TField Field, TBlobStreamMode Mode);
```

Description

Call CreateBlobStream to obtain a stream for reading and writing the value of the field specified by the Field parameter.

Field is a string containing the name of the BLOB field in the dataset. It must be an instance of TBlobField or one of its descendants.

Mode indicates whether the stream will be used for reading the field's value (bmRead), writing the field's value (bmWrite), or both (bmReadWrite).

CreateBlobStream returns the stream you can use for reading or writing data.

See Also

TSQLBlobStream (see page 1577)

TBlobField

25.1.3.4.8 TCustomSQLDataSet.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor

DefineBinaryProperty

DefineProperties

DefineProperty

25.1.3.4.9 TCustomSQLDataSet.ExecSQL Method

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; virtual;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

This is ExecSQL, a member of class TCustomSQLDataSet.

25.1.3.4.10 TCustomSQLDataSet.ExecuteStatement Method

Pascal

```
procedure ExecuteStatement;
```

C++

```
__fastcall ExecuteStatement();
```

Description

This is ExecuteStatement, a member of class TCustomSQLDataSet.

25.1.3.4.11 TCustomSQLDataSet.FreeBuffers Method

Pascal

```
procedure FreeBuffers;
```

C++

```
__fastcall FreeBuffers();
```

Description

This is FreeBuffers, a member of class TCustomSQLDataSet.

25.1.3.4.12 TCustomSQLDataSet.FreeCommand Method

Pascal

```
procedure FreeCommand;
```

C++

```
__fastcall FreeCommand();
```

Description

This is FreeCommand, a member of class TCustomSQLDataSet.

25.1.3.4.13 TCustomSQLDataSet.FreeReader Method

Pascal

```
procedure FreeReader;
```

C++

```
__fastcall FreeReader();
```

Description

This is FreeReader, a member of class TCustomSQLDataSet.

25.1.3.4.14 TCustomSQLDataSet.GetBlobFieldData Method

Retrieves the current value of a BLOB field into a buffer.

Pascal

```
function GetBlobFieldData(FieldNo: Integer; var Buffer: TBlobByteData): Integer; override;
```

C++

```
virtual __fastcall int GetBlobFieldData(int FieldNo, TBlobByteData Buffer);
```

Description

This method is used internally to read BLOB data from the field specified by FieldNo into a Buffer. The buffer is a dynamic array of bytes, so that it can grow to accommodate the size of the BLOB data. GetBlobFieldData returns the size of the buffer.

GetBlobFieldData is intended primarily for internal use. Applications should use the BLOB stream returned by the CreateBlobStream (see page 1528) method to access BLOB data.

See Also

GetFieldData (see page 1532)

IsBlob

CreateBlobStream (see page 1528)

GetBlobSize

25.1.3.4.15 TCustomSQLDataSet.GetCanModify Method

Pascal

```
function GetCanModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetCanModify();
```

Description

This is GetCanModify, a member of class TCustomSQLDataSet.

25.1.3.4.16 TCustomSQLDataSet.GetCommandNames Method

Pascal

```
procedure GetCommandNames(List: TWideStrings);
```

C++

```
__fastcall GetCommandNames(TWideStrings List);
```

Description

This is GetCommandNames, a member of class TCustomSQLDataSet.

25.1.3.4.17 TCustomSQLDataSet.GetDataSource Method

Pascal

```
function GetDataSource: TDataSource; override;
```

C++

```
virtual __fastcall TDataSource GetDataSource();
```

Description

This is GetDataSource, a member of class TCustomSQLDataSet.

25.1.3.4.18 TCustomSQLDataSet.GetDetailLinkFields Method

Lists the field components that link this dataset as a detail of a master dataset.

Pascal

```
procedure GetDetailLinkFields(MasterFields: TList; DetailFields: TList); override;
```

C++

```
virtual __fastcall GetDetailLinkFields(TList MasterFields, TList DetailFields);
```

Description

GetDetailLInkFIelds fills the two lists with field components that define a master-detail relationship between this dataset and another (master) dataset. The MasterFields list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list. The DetailFields list is filled with fields from this dataset.

Applications seldom call GetDetailLinkFields. It is used internally by the Field Link designer when setting up master/detail relationships.

See Also

DataSource (see page 1557)

MasterFields (see page 1691)

MasterSource (see page 1692)

25.1.3.4.19 GetFieldData Method

This is the overview for the GetFieldData method overload.

Overload List

	Name	Description
	TCustomSQLDataSet.GetFieldData (Integer, Pointer) (see page 1532)	Retrieves the current value of a field into a buffer.
	TCustomSQLDataSet.GetFieldData (TField, Pointer) (see page 1532)	Retrieves the current value of a field into a buffer.

25.1.3.4.19.1 TCustomSQLDataSet.GetFieldData Method (Integer, Pointer)

Retrieves the current value of a field into a buffer.

Pascal

```
function GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean; override; overload;
function GetFieldData(Field: TField; Buffer: Pointer): Boolean; override; overload;
```

C++

```
virtual __fastcall Boolean GetFieldData(int FieldNo, void * Buffer);
virtual __fastcall Boolean GetFieldData(TField Field, void * Buffer);
```

Description

Most applications do not need to call GetFieldData. TField objects call this method to implement their GetData method.

The Field or FieldNo parameter indicates the field whose data should be fetched. Field specifies the component itself, while FieldNo indicates its field number.

Buffer is a memory buffer with sufficient space to accept the value of the field as it exists in the database (unformatted and untranslated). GetFieldData returns a value that indicates whether the data was successfully fetched.

GetFieldData returns a value that indicates whether the data was successfully fetched.

See Also

GetBlobFieldData (see page 1530)

IsBlob

GetData

25.1.3.4.20 GetKeyFieldNames Method

This is the overview for the GetKeyFieldNames method overload.

Overload List

	Name	Description
	TCustomSQLDataSet.GetKeyFieldNames (TStrings) (see page 1533)	Fills a list with the names of all fields in all indexes defined for the dataset.
	TCustomSQLDataSet.GetKeyFieldNames (TWideStrings) (see page 1533)	Fills a list with the names of all fields in all indexes defined for the dataset.

25.1.3.4.20.1 TCustomSQLDataSet.GetKeyFieldNames Method (TStrings)

Fills a list with the names of all fields in all indexes defined for the dataset.

Pascal

```
function GetKeyFieldNames(List: TStrings): Integer; overload;
function GetKeyFieldNames(List: TWideStrings): Integer; overload;
```

C++

```
__fastcall int GetKeyFieldNames(TStrings List);
__fastcall int GetKeyFieldNames(TWideStrings List);
```

Description

Call GetKeyFieldNames to fill a string list with information about the indexes that are available on the database server for this dataset.

For every index defined on the server for the dataset, GetKeyFieldNames adds a string to the string list specified by List. Each string is a semicolon-delimited set of field names that lists the fields in a single index.

TCustomSQLDataSet (see page 1520) does not explicitly use these indexes. GetKeyFieldNames can be useful for composing SQL commands dynamically that need to use an index to access fields.

See Also

IndexDefs (see page 1558)

25.1.3.4.21 TCustomSQLDataSet.GetObjectTypeNames Method

Pascal

```
procedure GetObjectTypeNames(Fields: TFields);
```

C++

```
__fastcall GetObjectTypeNames(TFields Fields);
```

Description

This is GetObjectTypeNames, a member of class TCustomSQLDataSet.

25.1.3.4.22 TCustomSQLDataSet.GetOutputParams Method

Pascal

```
procedure GetOutputParams(AProcParams: TList = nil);
```

C++

```
__fastcall GetOutputParams(TList AProcParams = nil);
```

Description

This is GetOutputParams, a member of class TCustomSQLDataSet.

25.1.3.4.23 TCustomSQLDataSet.GetQuoteChar Method

Returns the character or characters to use in generated SQL statements to enclose quoted strings.

Pascal

```
function GetQuoteChar: UnicodeString; virtual;
```

C++

```
virtual __fastcall UnicodeString GetQuoteChar();
```

Description

GetQuoteChar returns the character or characters that delimit quoted strings in automatically generated SQL statements. This string appears both before and after quoted strings.

25.1.3.4.24 TCustomSQLDataSet.GetRecord Method

Pascal

```
function GetRecord(Buffer: TRecordBuffer; GetMode: TGetMode; DoCheck: Boolean): TGetResult;
override;
```

C++

```
virtual __fastcall TGetResult GetRecord(TRecordBuffer Buffer, TGetMode GetMode, Boolean
DoCheck);
```

Description

This is GetRecord, a member of class TCustomSQLDataSet.

25.1.3.4.25 TCustomSQLDataSet.GetRecordCount Method

Pascal

```
function GetRecordCount: Integer; override;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

This is GetRecordCount, a member of class TCustomSQLDataSet.

25.1.3.4.26 TCustomSQLDataSet.GetSortFieldNames Method

Pascal

```
function GetSortFieldNames: UnicodeString;
```

C++

```
__fastcall UnicodeString GetSortFieldNames();
```

Description

This is GetSortFieldNames, a member of class TCustomSQLDataSet.

25.1.3.4.27 TCustomSQLDataSet.InitRecord Method

Pascal

```
procedure InitRecord(Buffer: TRecordBuffer); override;
```

C++

```
virtual __fastcall InitRecord(TRecordBuffer Buffer);
```

Description

This is InitRecord, a member of class TCustomSQLDataSet.

25.1.3.4.28 TCustomSQLDataSet.InternalClose Method

Pascal

```
procedure InternalClose; override;
```

C++

```
virtual __fastcall InternalClose();
```

Description

This is InternalClose, a member of class TCustomSQLDataSet.

25.1.3.4.29 TCustomSQLDataSet.InternalFreeCommand Method

Pascal

```
procedure InternalFreeCommand;
```

C++

```
__fastcall InternalFreeCommand();
```

Description

This is InternalFreeCommand, a member of class TCustomSQLDataSet.

25.1.3.4.30 TCustomSQLDataSet.InternalHandleException Method

Pascal

```
procedure InternalHandleException; override;
```

C++

```
virtual __fastcall InternalHandleException();
```

Description

This is InternalHandleException, a member of class TCustomSQLDataSet.

25.1.3.4.31 TCustomSQLDataSet.InternalInitFieldDefs Method

Pascal

```
procedure InternalInitFieldDefs; override;
```

C++

```
virtual __fastcall InternalInitFieldDefs();
```

Description

This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.

25.1.3.4.32 TCustomSQLDataSet.InternalOpen Method

Pascal

```
procedure InternalOpen; override;
```

C++

```
virtual __fastcall InternalOpen();
```

Description

This is InternalOpen, a member of class TCustomSQLDataSet.

25.1.3.4.33 TCustomSQLDataSet.InternalRefresh Method

Pascal

```
procedure InternalRefresh; override;
```

C++

```
virtual __fastcall InternalRefresh();
```

Description

This is InternalRefresh, a member of class TCustomSQLDataSet.

25.1.3.4.34 TCustomSQLDataSet.IsCursorOpen Method

Pascal

```
function IsCursorOpen: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCursorOpen();
```

Description

This is IsCursorOpen, a member of class TCustomSQLDataSet.

25.1.3.4.35 TCustomSQLDataSet.IsSequenced Method

Indicates whether the dataset can use record numbers to indicate the order of records.

Pascal

```
function IsSequenced: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsSequenced();
```

Description

Datasets use IsSequenced to indicate whether the RecNo property corresponds to the order of records. When IsSequenced returns true, applications can safely use the RecNo property to navigate to records in the dataset.

Note: For TCustomSQLDataSet (see page 1520), IsSequenced always returns false because TCustomSQLDataSet (see page 1520) descendants do not support record numbers at all.

See Also

RecNo

RecordCount (see page 1564)

25.1.3.4.36 TCustomSQLDataSet.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

25.1.3.4.37 TCustomSQLDataSet.Locate Method

Implements a virtual method for searching a dataset for a specified record and making it the active record.

Pascal

```
function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):
Boolean; override;
```

C++

```
virtual __fastcall Boolean Locate(const UnicodeString KeyFields, const Variant KeyValues,
TLocateOptions Options);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError exception.

Returns false, indicating that a matching record was not found and the active record was not changed.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the semicolon-separated list of fields in KeyFields have the values specified by the Variant (see page 3421) or Variant (see page 3421) array KeyValues. Options indicates whether the search is case insensitive and whether partial matches are supported. Locate returns true if a record is found that matches the specified criteria and that record is now active.

See Also

FindField

Lookup

25.1.3.4.38 TCustomSQLDataSet.LocateRecord Method

Pascal

```
function LocateRecord(const KeyFields: string; const KeyValues: Variant; Options:
TLocateOptions; SyncCursor: Boolean): Boolean;
```

C++

```
__fastcall Boolean LocateRecord(const UnicodeString KeyFields, const Variant KeyValues,
TLocateOptions Options, Boolean SyncCursor);
```

Description

This is LocateRecord, a member of class TCustomSQLDataSet.

25.1.3.4.39 TCustomSQLDataSet.Lookup Method

Implements a virtual method to retrieve field values from a record that matches specified search values.

Pascal

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields:
string): Variant; override;
```

C++

```
virtual __fastcall Variant Lookup(const UnicodeString KeyFields, const Variant KeyValues,
const UnicodeString ResultFields);
```

Description

This function:

Checks whether the dataset is unidirectional, and if so, raises an EDatabaseError exception.

Returns a Variant (see page 3421) with the value false, indicating that a matching record was not found.

Descendant classes that are not unidirectional override this method so that it locates the record where the fields identified by the comma-delimited string KeyFields have the values specified by the Variant (see page 3421) or Variant (see page 3421) array KeyValues. In classes that implement Lookup, it returns a Variant (see page 3421) or Variant (see page 3421) array that contains the value or values of the fields specified by the comma-delimited string ResultFields on the specified record.

See Also

Locate (see page 1538)

25.1.3.4.40 TCustomSQLDataSet.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TCustomSQLDataSet.

25.1.3.4.41 TCustomSQLDataSet.OpenSchema Method

Pascal

```
procedure OpenSchema; virtual;
```

C++

```
virtual __fastcall OpenSchema();
```

Description

This is OpenSchema, a member of class TCustomSQLDataSet.

25.1.3.4.42 TCustomSQLDataSet.ParamByName Method

Locates a parameter in the Params (see page 1562) property collection.

Pascal

```
function ParamByName(const Value: string): TParam;
```

C++

```
__fastcall TParam ParamByName(const UnicodeString Value);
```

Description

Call ParamByName to access a specific parameter object, given its name. Value is the name of the parameter.

ParamByName is primarily used to set a parameter's value at runtime.

See Also

Params (see page 1562)

Name

25.1.3.4.43 TCustomSQLDataSet.PrepareStatement Method

Performs a prepare on the SQL statement.

Pascal

```
procedure PrepareStatement; virtual;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

The prepare operation is a type of pre-compile that checks the performance of the SQL statement. Some databases support a prepare statement that creates an execution plan that speeds processing.

25.1.3.4.44 TCustomSQLDataSet.PropertyChanged Method

Pascal

```
procedure PropertyChanged;
```

C++

```
__fastcall PropertyChanged();
```

Description

This is PropertyChanged, a member of class TCustomSQLDataSet.

25.1.3.4.45 TCustomSQLDataSet.PSEndTransaction Method

Ends a transaction that was started by a call to PSStartTransaction.

Pascal

```
procedure PSEndTransaction(Commit: Boolean); override;
```

C++

```
virtual __fastcall PSEndTransaction(Boolean Commit);
```

Description

The provider component applies updates within a transaction, if possible. To do so, it uses PSEndTransaction in conjunction with the PSStartTransaction method.

The Commit parameter indicates whether the dataset should commit the current transaction (Commit is true), or roll it back (Commit is false).

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there is no transaction already established. That is, it does not assume nested transaction support is available. Thus, there is no need to ensure that PSEndTransaction does not terminate a transaction that was not started by the provider.

See Also

PSStartTransaction

PSInTransaction

ApplyUpdates

25.1.3.4.46 TCustomSQLDataSet.PSExecute Method

Executes the SQL command associated with the dataset.

Pascal

```
procedure PSExecute; override;
```

C++

```
virtual __fastcall PSExecute();
```

Description

The provider component calls PSExecute to implement its Execute method. If the dataset represents a query or stored procedure that does not return a cursor, this method executes the query or stored procedure. If the provider Options includes poAllowCommandText and the dataset supports the PSSetCommandText method, the statement specified by PSSetCommandText may be executed instead.

If there is not statement to execute, PSExecute raises an EDatabaseError exception.

See Also

PSSetCommandText

Execute

Options

25.1.3.4.47 TCustomSQLDataSet.PSExecuteStatement Method

Pascal

```
function PSExecuteStatement(const ASQL: WideString; AParams: TParams; ResultSet: TPSResult =
nil): Integer; override;
```

C++

```
virtual __fastcall int PSExecuteStatement(const BSTR ASQL, TParams AParams, TPSResult
ResultSet = nil);
```

Description

This is PSExecuteStatement, a member of class TCustomSQLDataSet.

25.1.3.4.48 TCustomSQLDataSet.PSGetAttributes Method

Returns information that is stored in data packets as Name/Value pairs.

Pascal

```
procedure PSGetAttributes(List: TList); override;
```

C++

```
virtual __fastcall PSGetAttributes(TList List);
```

Description

The provider component calls PSGetAttributes to add information about the dataset to the metadata stored with generated data packets. This information is stored as Name/Value pairs that are added to the list specified by the List parameter.

See Also

DoGetProviderAttributes

25.1.3.4.49 TCustomSQLDataSet.PSGetCommandText Method

Gets the SQL command to be executed by PSExecute.

Pascal

```
function PSGetCommandText: string; override;
```

C++

```
virtual __fastcall UnicodeString PSGetCommandText();
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

See Also

CommandText

PSExecute

PSSetCommandText

GetRecords

Execute

Options

25.1.3.4.50 TCustomSQLDataSet.PSGetCommandType Method

Gets the command type to be executed by PSExecute.

Pascal

```
function PSGetCommandType: TPSCommandType; override;
```

C++

```
virtual __fastcall TPSCommandType PSGetCommandType();
```

Description

PSGetCommandType gets the command type executed by PSExecute. The type may be a type of SQL command or other command types, such as stored procedures.

See Also

CommandText

PSExecute

PSGetCommandText

Execute

Options

25.1.3.4.51 TCustomSQLDataSet.PSGetDefaultOrder Method

Returns the definition of an index that imposes the default order on the data included in data packets.

Pascal

```
function PSGetDefaultOrder: TIndexDef; override;
```

C++

```
virtual __fastcall TIndexDef PSGetDefaultOrder();
```

Description

The provider component calls PSGetDefaultOrder to generate a default index, called DEFAULT_ORDER, for the records that appear in data packets. This index defines the default sort order for records when they appear in the client dataset on the client.

PSGetDefaultOrder creates a TIndexDef object to describe the default index. The provider that calls PSGetDefaultOrder subsequently frees this TIndexDef object.

See Also

CreateDataPacket

25.1.3.4.52 TCustomSQLDataSet.PSGetIndexDefs Method

Returns the definitions of all specified indexes defined for the dataset.

Pascal

```
function PSGetIndexDefs(IndexTypes: TIndexOptions): TIndexDefs; override;
```

C++

```
virtual __fastcall TIndexDefs PSGetIndexDefs(TIndexOptions IndexTypes);
```

Description

The provider component calls PSGetIndexDefs to locate an index that contains information it needs. For example, it uses PSGetIndexDefs to locate an index that contains the key fields returned by PSGetKeyFields. This information allows the provider to locate the records that need to change when it applies updates. The provider also calls this method to locate indexes that it adds to the metadata of data packets.

IndexTypes indicates the options required of the returned indexes. PSGetIndexDefs returns all indexes whose Options property includes the specified flags.

PSGetIndexDefs returns a TIndexDefs object that contains all the indexes of the dataset that match the IndexTypes parameter.

See Also

Options

PSGetKeyFields

PSGetDefaultOrder

25.1.3.4.53 TCustomSQLDataSet.PSGetKeyFieldsW Method

Pascal

```
function PSGetKeyFieldsW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetKeyFieldsW();
```

Description

This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.

25.1.3.4.54 TCustomSQLDataSet.PSGetParams Method

Returns the current parameter values of the dataset.

Pascal

```
function PSGetParams: TParams; override;
```

C++

```
virtual __fastcall TParams PSGetParams();
```

Description

The provider component calls PSGetParams to implement its GetParams method.

PSGetParams returns a TParams object that describes the current values of all parameters of the dataset. If the dataset does not have any parameters, PSGetParams returns nil (Delphi) or NULL (C++).

Note: The PSGetParams method should not create a new instance of TParams for the return value. The provider that calls PSGetParams does not free the returned value.

See Also

GetParams

25.1.3.4.55 TCustomSQLDataSet.PSGetQuoteCharW Method

Pascal

```
function PSGetQuoteCharW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetQuoteCharW();
```

Description

This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.

25.1.3.4.56 TCustomSQLDataSet.PSGetTableNameW Method

Pascal

```
function PSGetTableNameW: WideString; override;
```

C++

```
virtual __fastcall BSTR PSGetTableNameW();
```

Description

This is PSGetTableNameW, a member of class TCustomSQLDataSet.

25.1.3.4.57 TCustomSQLDataSet.PSGetUpdateException Method

Generates an EUpdateError object based on another exception object.

Pascal

```
function PSGetUpdateException(E: Exception; Prev: EUpdateError): EUpdateError; override;
```

C++

```
virtual __fastcall EUpdateError PSGetUpdateException(Exception E, EUpdateError Prev);
```

Description

The provider component calls PSGetUpdateException when the exception E occurs while applying updates. This method allows the provider to create a single EUpdateError object that encapsulates all the exceptions that occur in an update operation.

PSGetUpdateError determines the error message, context string, and error code from the exception specified by E and creates a new EUpdateError exception that includes that information as well as the information about a previous error as specified by Prev. The context string and error code for E may be specific to the Exception class associated with the dataset. By default, TDataSet generates an EUpdateError with an error code of 1 and an empty context string.

See Also

Create

25.1.3.4.58 TCustomSQLDataSet.PSInTransaction Method

Indicates whether actions against the dataset occur in the context of a transaction.

Pascal

```
function PSInTransaction: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSInTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it first checks whether a transaction is already underway, and if not, generates one using PSStartTransaction. To determine whether a transaction is already underway, the provider calls PSInTransaction.

PSInTransaction returns true if there is a transaction underway, and false otherwise.

See Also

PSStartTransaction

PSEndTransaction

25.1.3.4.59 TCustomSQLDataSet.PSIsSQLBased Method

Indicates whether the dataset includes native SQL support.

Pascal

```
function PSIsSQLBased: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLBased();
```

Description

The provider calls PSIsSQLBased to determine whether the dataset supports some form of native SQL. This information is used when generating SQL commands that are executed by PSExecuteStatement.

Even if the dataset is not SQL-based, it may still allow the provider to execute SQL statements for applying updates. For example, BDE-enabled local databases such as Paradox and DBASE provide SQL support via the BDE's local SQL engine, but are not SQL-based. However, table and field names are not specified in the same way for the local SQL engine as they are for native SQL. The provider uses PSIsSQLBased to determine how these names should appear in generated SQL.

Note: The PSIsSQLSupported method indicates whether the dataset can execute SQL commands at all, regardless of whether

the dataset is SQL-based.

See Also

PSIsSQLSupported

PSExecuteStatement

25.1.3.4.60 TCustomSQLDataSet.PSIsSQLSupported Method

Indicates whether the dataset can execute SQL statements.

Pascal

```
function PSIsSQLSupported: Boolean; override;
```

C++

```
virtual __fastcall Boolean PSIsSQLSupported();
```

Description

PSIsSQLSupported indicates whether the provider can call PSExecuteStatement to execute an SQL statement. When PSIsSQLSupported returns false, calling PSExecuteStatement generates an exception. The provider can still work with a dataset that returns false from PSIsSQLSupported, but it must set the ResolveToDataSet property to true, or provide data on a read-only basis.

See Also

PSIsSQLBased

PSExecuteStatement

ResolveToDataSet

25.1.3.4.61 TCustomSQLDataSet.PSReset Method

Resets the dataset so that it is positioned on the first record.

Pascal

```
procedure PSReset; override;
```

C++

```
virtual __fastcall PSReset();
```

Description

The provider calls PSReset to reposition the dataset on the first record before fetching records.

See Also

GetRecords

AS_GetRecords

25.1.3.4.62 TCustomSQLDataSet.PSSetCommandText Method

Assigns a specified SQL command to be executed by PSExecute.

Pascal

```
procedure PSSetCommandText(const ACommandText: WideString); override;
```

C++

```
virtual __fastcall PSSetCommandText(const BSTR ACommandText);
```

Description

If the provider's Options property includes poAllowCommandText, clients can supply an SQL command along with a call to GetRecords or Execute. This command is passed to the dataset by a call to PSSetCommandText, and subsequently executed by a call to PSExecute.

The value of the CommandText parameter replaces the dataset's SQL statement (if it has one), or replaces the name of an underlying database table or stored procedure.

See Also

CommandText

PSExecute

GetRecords

Execute

Options

25.1.3.4.63 TCustomSQLDataSet.PSSetParams Method

Assigns the specified parameter values to the dataset.

Pascal

```
procedure PSSetParams(AParams: TParams); override;
```

C++

```
virtual __fastcall PSSetParams(TParams AParams);
```

Description

PSSetParams assigns the parameter values specified by AParams to the parameters of the dataset.

See Also

Params

PSGetParams

25.1.3.4.64 TCustomSQLDataSet.PSStartTransaction Method

Starts a transaction that encapsulates all activity until a call to PSEndTransaction.

Pascal

```
procedure PSStartTransaction; override;
```

C++

```
virtual __fastcall PSStartTransaction();
```

Description

The provider component applies updates within a transaction, if possible. To do so, it calls PSStartTransaction to start a transaction before applying updates, and calls PSEndTransaction when all updates are applied or too many errors are encountered.

Note: TDataSetProvider only starts a transaction (Using PSStartTransaction) if the PSInTransaction method indicates that there

is no transaction already established. That is, it does not assume nested transaction support is available.

See Also

PSEndTransaction

PSInTransaction

ApplyUpdates

25.1.3.4.65 TCustomSQLDataSet.PSUpdateRecord Method

Applies a single update to the underlying database table or tables.

Pascal

```
function PSUpdateRecord(UpdateKind: TUpdateKind; Delta: TDataSet): Boolean; override;
```

C++

```
virtual __fastcall Boolean PSUpdateRecord(TUpdateKind UpdateKind, TDataSet Delta);
```

Description

Before the provider generates SQL statements to apply an update, it calls PSUpdateRecord to let the dataset apply the update in some other fashion. For example, BDE-enabled datasets generate an OnUpdateRecord to allow the application to apply the update and, if the OnUpdateRecord event handler does not apply the update, uses any associated TUpdateSQL object to apply the update.

UpdateKind indicates whether the update is an insertion (ukInsert), deletion (ukDelete), or modification (ukModify).

Delta is the delta datapacket. The current record represents the record to be updated.

PSUpdateRecord returns true if it updates the record, false otherwise. If PSUpdateRecord returns false, the provider generates dynamic SQL to perform the update and calls PSExecuteStatement.

Note: PSUpdateRecord is only called if the provider's ResolveToDataSet property is false.

See Also

OnUpdateRecord

TUpdateSQL

ApplyUpdates

ResolveToDataSet

PSExecuteStatement

25.1.3.4.66 TCustomSQLDataSet.SetBufListSize Method

Pascal

```
procedure SetBufListSize(Value: Integer); override;
```

C++

```
virtual __fastcall SetBufListSize(int Value);
```

Description

This is SetBufListSize, a member of class TCustomSQLDataSet.

25.1.3.4.67 TCustomSQLDataSet.SetCommandText Method

Pascal

```
procedure SetCommandText(const Value: UnicodeString); virtual;
```

C++

```
virtual __fastcall SetCommandText(const UnicodeString Value);
```

Description

This is SetCommandText, a member of class TCustomSQLDataSet.

25.1.3.4.68 TCustomSQLDataSet.SetFCommandText Method

Pascal

```
procedure SetFCommandText(const Value: string);
```

C++

```
__fastcall SetFCommandText(const UnicodeString Value);
```

Description

This is SetFCommandText, a member of class TCustomSQLDataSet.

25.1.3.4.69 TCustomSQLDataSet.SetFieldData Method

Pascal

```
procedure SetFieldData(Field: TField; Buffer: Pointer); override;
```

C++

```
virtual __fastcall SetFieldData(TField Field, void * Buffer);
```

Description

This is SetFieldData, a member of class TCustomSQLDataSet.

25.1.3.4.70 TCustomSQLDataSet.SetMaxBlobSize Method

Pascal

```
procedure SetMaxBlobSize(MaxSize: Integer);
```

C++

```
__fastcall SetMaxBlobSize(int MaxSize);
```

Description

This is SetMaxBlobSize, a member of class TCustomSQLDataSet.

25.1.3.4.71 TCustomSQLDataSet.SetParamsFromCursor Method

Pascal

```
procedure SetParamsFromCursor;
```

C++

```
__fastcall SetParamsFromCursor();
```

Description

This is SetParamsFromCursor, a member of class TCustomSQLDataSet.

25.1.3.4.72 TCustomSQLDataSet.SetSchemaInfo Method

Indicates whether the dataset represents metadata from the server, and if so, what kind.

Pascal

```
procedure SetSchemaInfo(SchemaType: TSchemaType; SchemaObjectName: UnicodeString;
SchemaPattern: UnicodeString; PackageName: UnicodeString = '');
```

C++

```
__fastcall SetSchemaInfo(TSchemaType SchemaType, UnicodeString SchemaObjectName, UnicodeString
SchemaPattern, UnicodeString PackageName = '');
```

Description

Call SetSchemaInfo to indicate whether the SQL dataset represents the results of an SQL command (specified by CommandText (see page 1554)) or whether it represents a table of schema information (metadata).

The SchemaType parameter identifies the type of information you want. It can have any of the following values:

Value	Description
stNoSchema	No schema information. The SQL dataset is populated with the results of its query or stored procedure rather than metadata from the server.
stTables	Information about all the data tables on the database server that match the criteria specified by the SQL connection's TableScope property.
stSysTables	Information about all of the system tables on the database server. Not all servers use system tables to store metadata. Requesting a list of system tables from a server that does not use them results in an empty dataset.
stProcedures	Information about all of the stored procedures on the database server.
stColumns	Information about all of the columns (fields) in a specified table.
stProcedureParams	Information about all of the parameters of a specified stored procedure.
stIndexes	Information about all of the indexes defined for a specified table.

SchemaObjectName is the name of the table or stored procedure about whose columns, indexes, or parameters you want information. When SchemaType is stColumns or stIndexes, SchemaObjectName is the name of a table. When SchemaType is stProcedureParams, SchemaObjectName is the name of a stored procedure. When SchemaType is stNoSchema, stTables, stSysTables, or stProcedures, or stPackages, SchemaObjectName is ignored.

SchemaPattern is an SQL pattern mask that filters the resulting dataset. The name of each item (table, stored procedure, field, parameter, or index) must match this pattern mask or it does not appear as a record when the SQL dataset is opened. SchemaPattern uses the wildcards '%' (to match a string of arbitrary characters of any length) and '_' (to match a single arbitrary character). To use a literal percent or underscore in a pattern, the character is doubled (%% or __). If you do not want to use a pattern, set Pattern to nil (Delphi) or NULL (C++).

PackageName is the name of an Oracle package in which to find the stored procedure when SchemaType is stProcedures or stProcedureParams. If the server is not an Oracle server, or if the stored procedure is not defined in a package, PackageName should be an empty string.

After a call to SetSchemaInfo, opening the dataset results in a dataset populated with the specified information. If SchemaType is stNoSchema, the dataset reverts to the command specified by CommandText (see page 1554).

Note: When you change the CommandText (see page 1554) property, the dataset automatically reverts to a SchemaType of stNoSchema.

See Also

Open

CommandText (see page 1554)

TableScope (see page 1623)

Fetching Metadata into a Unidirectional Dataset

Accessing Schema Information

25.1.3.4.73 TCustomSQLDataSet.SetSortFieldNames Method

Pascal

```
procedure SetSortFieldNames(Value: UnicodeString);
```

C++

```
__fastcall SetSortFieldNames(UnicodeString Value);
```

Description

This is SetSortFieldNames, a member of class TCustomSQLDataSet.

25.1.3.4.74 TCustomSQLDataSet.UpdateIndexDefs Method

Pascal

```
procedure UpdateIndexDefs; override;
```

C++

```
virtual __fastcall UpdateIndexDefs();
```

Description

This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

25.1.3.5 TCustomSQLDataSet Properties

The properties of the TCustomSQLDataSet class are listed here.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1555)	Indicates the meaning of the CommandText (see page 1554) property.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.
R	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1557)	Links the SQL dataset to another (master) dataset.
	DbxCommandType (see page 1557)	This is DbxCommandType, a member of class TCustomSQLDataSet.
R	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	MaxBlobSize (see page 1559)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.

	ParamCheck (see page 1561)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	Params (see page 1562)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1564)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1555) is ctTable.

Public Properties

	Name	Description
	CommandText (see page 1554)	Specifies the command the dataset executes.
	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	SQLConnection (see page 1566)	Specifies the SQL connection component that connects the dataset to a database server.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Properties

	Name	Description
	Active (see page 1552)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1558)	Specifies whether the SQL dataset fetches metadata information along with the data.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 1565)	Specifies the name of the database schema.

25.1.3.5.1 TCustomSQLDataSet.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.
2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      // This path depends on the Demos directory of your RAD Studio installation.
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
  end;
end;
```

See Also

DataSource

State

AfterOpen

BeforeOpen

Close

Open

CanModify

BeforeClose

AfterClose

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

25.1.3.5.2 TCustomSQLDataSet.BlobBuffer Property

Reserves a buffer for storing BLOB values.

Pascal

```
property BlobBuffer: TBlobByteData;
```

C++

```
__property TBlobByteData BlobBuffer;
```

Description

SQL datasets use the BlobBuffer property to allocate a special buffer for holding BLOB data. Applications do not access this protected property.

See Also

MaxBlobSize (see page 1559)

GetBlobFieldData

25.1.3.5.3 TCustomSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: UnicodeString;
```

C++

```
__property UnicodeString CommandText;
```

Description

The value of CommandText depends on the CommandType (see page 1555) property:

When CommandType (see page 1555) is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL (see page 1529) method is called.

When CommandType (see page 1555) is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL (see page 1529) method is called, depending on whether the stored procedure returns a set of records.

When CommandType (see page 1555) is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames (see page 1566) property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params (see page 1562) property to supply parameter values.

See Also

CommandType (see page 1555)

Params (see page 1562)

Open

ExecSQL (see page 1636)

SortFieldNames (see page 1566)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

25.1.3.5.4 TCustomSQLDataSet.CommandType Property

Indicates the meaning of the CommandText (see page 1554) property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the type of command that is contained in the CommandText (see page 1554) property. TCustomSQLDataSet (see page 1520) descendants set this value to the way in which they specify a query or stored procedure.

The following table lists the possible values:

CommandType	Corresponding CommandText (see page 1554)
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

See Also

CommandText (see page 1554)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

25.1.3.5.5 TCustomSQLDataSet.CurrentBlobSize Property

Indicates the size of the most recently read BLOB field.

Pascal

```
property CurrentBlobSize: Int64;
```

C++

```
__property Int64 CurrentBlobSize;
```

Description

Applications can't use this protected property. TCustomSQLDataSet (see page 1520) uses CurrentBlobSize internally to determine the size needed for BlobBuffer (see page 1554). When the dataset reads a BLOB field, it sets CurrentBlobSize to the number of bytes in the BLOB field, as reported by the database driver.

See Also

MaxBlobSize (see page 1559)

BlobBuffer (see page 1554)

25.1.3.5.6 TCustomSQLDataSet.DataLink Property

Identifies the datalink that manages communication between the SQL dataset and another, master dataset.

Pascal

```
property DataLink: TDataLink;
```

C++

```
__property TDataLink DataLink;
```

Description

The protected DataLink property is used internally when the SQL dataset acts as the detail dataset in a master/detail relationship that is not based on nested datasets. The datalink object manages communication between another (master) dataset and this SQL dataset. For example, it is the datalink that tells this SQL dataset to refresh its data when the active record of the master dataset changes.

Do not use DataLink to establish the relationship between an SQL dataset and another (master) dataset from which it obtains parameter values. Instead, use the DataSource (see page 1557) property, or, if the dataset is TSQLTable (see page 1680), the MasterSource property.

See Also

DataSource (see page 1557)

TSQLDataLink (see page 1628)

MasterSource (see page 1692)

25.1.3.5.7 TCustomSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (see page 1554) (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params (see page 1562) property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1554)

SQL (see page 1663)

Params (see page 1562)

MasterSource (see page 1692)

25.1.3.5.8 TCustomSQLDataSet.DbxCommandType Property

Pascal

```
property DbxCommandType: UnicodeString;
```

C++

```
__property UnicodeString DbxCommandType;
```

Description

This is DbxCommandType, a member of class TCustomSQLDataSet.

25.1.3.5.9 TCustomSQLDataSet.DesignerData Property

Stores custom data.

Pascal

```
property DesignerData: string;
```

C++

```
__property UnicodeString DesignerData;
```

Description

DesignerData has no predefined meaning. Application developers can use the DesignerData property to associate application-specific information with an SQL dataset instance.

See Also

Tag

25.1.3.5.10 TCustomSQLDataSet.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataSet

TDataSetProvider

Fetching the Data using TSQLDataSet

25.1.3.5.11 TCustomSQLDataSet.IndexDefs Property

Contains the definitions of all indexes defined on the server for the dataset.

Pascal

```
property IndexDefs: TIndexDefs;
```

C++

```
__property TIndexDefs IndexDefs;
```

Description

Read (see page 3634) IndexDefs to obtain information about what indexes are available on the database server for this dataset.

TCustomSQLDataSet (see page 1520) does not explicitly use the information in IndexDefs. The order of its records is determined by the ORDER BY clause of its SQL statement.

If the SQL dataset is an instance of TSQLTable (see page 1680), you can use IndexDefs to identify the index to use for the IndexName property. Even with other TCustomSQLDataSet (see page 1520) descendants, IndexDefs can be useful for composing SQL commands dynamically that need to use an index on the server. For example, provider components use the index information to uniquely identify records when composing update commands.

See Also

TIndexDef

SortFieldNames (see page 1566)

CommandText (see page 1554)

SQL (see page 1663)

TDataSetProvider

IndexName (see page 1690)

GetKeyFieldNames (see page 1533)

25.1.3.5.12 TCustomSQLDataSet.InternalConnection Property

Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.

Pascal

```
property InternalConnection: TSqlConnection;
```

C++

```
__property TSqlConnection InternalConnection;
```

Description

Applications can't use this protected property. Instead, SQL datasets use InternalConnection internally when they need to access the dbExpress interfaces for the connection to the database server. Unlike the published SQLConnection (see page 1566) property, InternalConnection handles the case when the dataset must use a clone of the primary connection because the server limits the number of active statements that the connection supports.

It is not necessary to use InternalConnection except when using the connection component to reference its dbExpress interfaces.

See Also

CloneConnection (see page 1587)

AutoClone (see page 1608)

SQLConnection (see page 1566)

25.1.3.5.13 TCustomSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1530)

Params (see page 1621)

SQLConnection (see page 1566)

25.1.3.5.14 TCustomSQLDataSet.NativeCommand Property

Represents the SQL command that is sent to the server.

Pascal

```
property NativeCommand: UnicodeString;
```

C++

```
__property UnicodeString NativeCommand;
```

Description

SQL datasets sometimes need to alter a query before it can be sent to the dbExpress driver. For example, dbExpress does not support named parameters, so the SQL dataset converts named parameters to unnamed parameters before sending off a command.

NativeCommand is the adjusted command that is sent to the dbExpress driver. It is based on the value of CommandText (see page 1554).

See Also

CommandText (see page 1554)

25.1.3.5.15 TCustomSQLDataSet.NumericMapping Property

Maps fldBCD to either ftFMTBCD or ftBCD field types.

Pascal

```
property NumericMapping: Boolean;
```

C++

```
__property Boolean NumericMapping;
```

Description

Use NumericMapping to control how to map fldBCD (dbExpress Core logical type) to eitheir ftFMTBCD or ftBCD field types. The default is False. If True, all dbExpress fldBCD types will map to ftFMTBCD across all dbExpress drivers for field type compatibility.

See Also

TFieldType

25.1.3.5.16 TCustomSQLDataSet.ObjectView Property

Specifies whether fields are stored hierarchically or flattened out in the Fields property.

Pascal

```
property ObjectView;
```

C++

```
__property ObjectView;
```

Description

ObjectView affects the way the Fields property stores object fields and the way data-aware grids display ADT and array fields.

When ObjectView is true, the fields are stored hierarchically in the Fields property, meaning any child fields of an object field are referenced by the object field and don't appear sequentially after the object field in the TFields:: .Fields array. When ObjectView is false, the fields are stored sequentially in the Fields property, meaning any child fields of an object field are stored after the object field as siblings in the Fields array.

When ObjectView is false, object field types, such as TADTField, are not created. This switch is provided for increased compatibility with older data-aware controls, which may not handle object field types properly. The default is false for TBDEDataSet, and true for TClientDataSet.

Note: If the ObjectView property of the dataset provided to a TClientDataSet is false, then the TClientDataSet's ObjectView property always acts as false, since the data received is already flattened.

See Also

FieldDefList

Fields

Displaying ADT and Array Fields

Using Nested Detail Tables

Using Object Fields

Working with ADT Fields

Working with Array Fields

25.1.3.5.17 TCustomSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params (see page 1562) property is updated automatically to reflect any parameters that appear in the query specified by the CommandText (see page 1554) property (or the SQL property for TSQLQuery (see page 1654)) at runtime. That is, whenever the CommandText (see page 1554) property changes, the Params (see page 1562) property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params (see page 1562) property in response to changes in CommandText (see page 1554), set ParamCheck to false.

See Also

Params (see page 1562)

CommandText (see page 1554)

SQL (see page 1663)

25.1.3.5.18 TCustomSQLDataSet.ParamCount Property

Indicates the number of parameters listed by the Params (see page 1562) property.

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

ParamCount is the number of parameters listed in the Params (see page 1562) property. Applications can't use this protected property. It is used internally when preparing SQL statements for execution.

See Also

Params (see page 1562)

Prepared (see page 1563)

25.1.3.5.19 TCustomSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText (see page 1554) property (or the SQL property in TSQLQuery (see page 1654)), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText (see page 1554) property (or the StoredProcName property in TSQLStoredProc (see page 1671)), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource (see page 1557) property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1561)

SQL (see page 1663)

StoredProcName (see page 1680)

CommandText (see page 1554)

TParam

Value

DataType

DataSource (see page 1557)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

25.1.3.5.20 TCustomSQLDataSet.Prepared Property

Specifies whether the command is prepared before execution.

Pascal

```
property Prepared: Boolean;
```

C++

```
__property Boolean Prepared;
```

Description

SQL datasets are automatically prepared when you call Open or ExecSQL (see page 1529) (or ExecProc). When a SQL dataset is prepared, dbExpress and the server allocate resources for the SQL statement and its parameters. If CommandType (see page 1555) is ctTable, the dataset generates its SELECT query when it is prepared. Any parameters that are not bound by the server are folded into a query when it is prepared.

If you execute the query or stored procedure more than once, you can improve performance by setting Prepared to true before opening the dataset. Setting Prepared to true prepares the dataset at once, rather than waiting until the dataset is opened.

Conversely, set Prepared to false if you want to ensure that the dataset is re-prepared before it executes (for example, if you change a parameter value or the SortFieldNames (see page 1566) property).

See Also

Params (see page 1562)

ExecSQL (see page 1636)

Open

Executing the Command

Fetching the Data

25.1.3.5.21 TCustomSQLDataSet.ProcParams Property

ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.

Pascal

```
property ProcParams: TList;
```

C++

```
__property TList ProcParams;
```

Description

When the SQL dataset represents a stored procedure, it fetches information about stored procedure parameters from the server as a list of parameter descriptors. It stores this list as the value of ProcParams.

Applications never use the protected ProcParams directly. Instead, the information from the list is used to initialize the Params (see page 1562) property.

See Also

Params (see page 1562)

GetProcedureParams (see page 1597)

25.1.3.5.22 TCustomSQLDataSet.RecordCount Property

Indicates the total number of records associated with the dataset.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

Read (see page 3634) RecordCount to determine the total number of records in the dataset. An application can read RecordCount even if the dataset is not active.

Reading RecordCount will generate an exception if the dataset can't determine the number of records. Do not read RecordCount if

The dataset represents stored procedure.

The dataset represents a query that contains parameters.

The dataset represents a multi-table join.

See Also

RecNo

Eof

25.1.3.5.23 TCustomSQLDataSet.RowsAffected Property

Indicates the number of rows operated on by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

When the SQL dataset represents a query that does not return a result set, you can't use the RecordCount (see page 1564) property to ascertain the number of records affected by the query. In such cases, read RowsAffected to determine how many rows were updated or deleted by the query when it executed. If no rows were updated or deleted, RowsAffected is zero. RowsAffected is –1 if the SQL statement could not be executed due to an error condition. (This latter situation typically follows the throwing of an exception.)

See Also

ExecSQL (see page 1636)

RecordCount (see page 1564)

ExecProc (see page 1675)

25.1.3.5.24 TCustomSQLDataSet.SchemaInfo Property

Identifies the metadata with which to populate the dataset.

Pascal

```
property SchemaInfo: TSQLSchemaInfo;
```

C++

```
__property TSQLSchemaInfo SchemaInfo;
```

Description

SQL datasets can be used to represent metadata from the database server rather than the results of a query or stored procedure. SchemaInfo controls whether the dataset represents metadata, and if so, what kind.

When SchemaInfo specifies that the dataset represents schema information, the dataset is populated with that information when it is opened. Each record in the resulting dataset represents a single entity (a single table, stored procedure, field, parameter, or index).

Each type of schema information results in a different set of columns (fields), so it is not advisable to use persistent fields when accessing metadata through an SQL dataset.

Applications do not work directly with the protected SchemaInfo property. Instead, they specify the desired schema information by calling the SetSchemaInfo (see page 1550) method.

See Also

SetSchemaInfo (see page 1550)

Open

25.1.3.5.25 TCustomSQLDataSet.SchemaName Property

Specifies the name of the database schema.

Pascal

```
property SchemaName: string;
```

C++

```
__property UnicodeString SchemaName;
```

Description

This name will be available in the appropriate drop down list box after you have connected to a given data source.

25.1.3.5.26 TCustomSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType (see page 1555) is ctTable.

Pascal

```
property SortFieldNames: UnicodeString;
```

C++

```
__property UnicodeString SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType (see page 1555) is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 1555)

Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

25.1.3.5.27 TCustomSQLDataSet.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection (see page 1580) components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection (see page 1580) object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText (see page 1554) property or populate the dataset with data by setting Active (see page 1552) to true. Similarly, SQLConnection must be set before the dataset can initialize the Params (see page 1562) property when you set CommandText (see page 1554).

See Also

Active

CommandText (see page 1554)

Params (see page 1562)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

25.1.3.5.28 TCustomSQLDataSet.TransactionLevel Property

Indicates the transaction to which this dataset belongs.

Pascal

```
property TransactionLevel: SmallInt;
```

C++

```
__property SmallInt TransactionLevel;
```

Description

Use TransactionLevel to indicate the active transaction to use when opening this dataset or using it to execute commands. TransactionLevel is the same as the transaction identifier that is supplied to the connection's StartTransaction method. If TransactionLevel is unspecified (0), the dataset fetches its records or executes commands in the context of the most recently started transaction.

Note: Most applications do not need to use TransactionLevel because most database servers require that SQL commands belong to the most recently started transaction. However, the InterBase server allows multiple overlapping transactions to be active simultaneously.

See Also

InTransaction (see page 1613)

Commit (see page 1588)

Rollback (see page 1603)

Params (see page 1621)

TransactionsSupported (see page 1624)

StartTransaction (see page 1605)

Managing Transactions

25.1.3.6 TCustomSQLDataSet Events

The events of the TCustomSQLDataSet class are listed here.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.

	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

25.1.3.6.1 TCustomSQLDataSet.AfterClose Event

Occurs after a data source connection is closed.

Pascal

```
property AfterClose;
```

C++

```
__property AfterClose;
```

25.1.3.6.2 TCustomSQLDataSet.AfterOpen Event

Occurs after the connection to a data source is opened.

Pascal

```
property AfterOpen;
```

C++

```
__property AfterOpen;
```

25.1.3.6.3 TCustomSQLDataSet.AfterRefresh Event

Occurs after a DataSet has been changed and is refreshed.

Pascal

```
property AfterRefresh;
```

C++

```
__property AfterRefresh;
```

25.1.3.6.4 TCustomSQLDataSet.AfterScroll Event

Occurs after a result set is scrolled.

Pascal

```
property AfterScroll;
```

C++

```
__property AfterScroll;
```

25.1.3.6.5 TCustomSQLDataSet.BeforeClose Event

Occurs just before a data connection is closed.

Pascal

```
property BeforeClose;
```

C++

```
__property BeforeClose;
```

25.1.3.6.6 TCustomSQLDataSet.BeforeOpen Event

Occurs just before a data connection is opened, but after an open operation has been initiated.

Pascal

```
property BeforeOpen;
```

C++

```
__property BeforeOpen;
```

25.1.3.6.7 TCustomSQLDataSet.BeforeRefresh Event

Occurs before a DataSet refresh occurs.

Pascal

```
property BeforeRefresh;
```

C++

```
__property BeforeRefresh;
```

25.1.3.6.8 TCustomSQLDataSet.BeforeScroll Event

Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.

Pascal

```
property BeforeScroll;
```

C++

```
__property BeforeScroll;
```

25.1.3.6.9 TCustomSQLDataSet.OnCalcFields Event

Occurs when calculated or derived fields are updated.

Pascal

```
property OnCalcFields;
```

C++

```
__property OnCalcFields;
```

25.1.3.6.10 TCustomSQLDataSet.ParseDeleteSql Event

Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.

Pascal

```
property ParseDeleteSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseDeleteSql;
```

25.1.3.6.11 TCustomSQLDataSet.ParseInsertSql Event

Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.

Pascal

```
property ParseInsertSql: TParseInsertSqlEvent;
```

C++

```
__property TParseInsertSqlEvent ParseInsertSql;
```

25.1.3.6.12 TCustomSQLDataSet.ParseSelectSql Event

Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.

Pascal

```
property ParseSelectSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseSelectSql;
```

25.1.3.6.13 TCustomSQLDataSet.ParseUpdateSql Event

Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Pascal

```
property ParseUpdateSql: TParseSqlEvent;
```

C++

```
__property TParseSqlEvent ParseUpdateSql;
```

25.1.4 TFLDDesc Class

Class Hierarchy

Pascal

```
TFLDDesc = class;
```

C++

```
class TFLDDesc;
```

File

SqlExpr

Description

This is class SqlExpr.TFLDDesc.

25.1.4.1 TFLDDesc Members

The following tables list the members exposed by TFLDDesc.

Public Fields

	Name	Description
	FCalcField (see page 1572)	This is FCalcField, a member of class TFLDDesc.
	FFldNum (see page 1572)	This is FFldNum, a member of class TFLDDesc.
	FFLDRights (see page 1572)	This is FFLDRights, a member of class TFLDDesc.
	FFldType (see page 1572)	This is FFldType, a member of class TFLDDesc.
	FFLDVchk (see page 1573)	This is FFLDVchk, a member of class TFLDDesc.
	FLen (see page 1573)	This is FLen, a member of class TFLDDesc.
	FName (see page 1573)	This is FName, a member of class TFLDDesc.
	FNullOffset (see page 1573)	This is FNullOffset, a member of class TFLDDesc.
	FOffset (see page 1573)	This is FOffset, a member of class TFLDDesc.
	FSubType (see page 1574)	This is FSubType, a member of class TFLDDesc.
	FUnits1 (see page 1574)	This is FUnits1, a member of class TFLDDesc.
	FUnits2 (see page 1574)	This is FUnits2, a member of class TFLDDesc.

Public Properties

	Name	Description
	bCalcField (see page 1575)	This is bCalcField, a member of class TFLDDesc.
	efldrRights (see page 1575)	This is efldrRights, a member of class TFLDDesc.
	efldvVchk (see page 1575)	This is efldvVchk, a member of class TFLDDesc.
	iFldNum (see page 1575)	This is iFldNum, a member of class TFLDDesc.
	iFldType (see page 1576)	This is iFldType, a member of class TFLDDesc.
	iLen (see page 1576)	This is iLen, a member of class TFLDDesc.
	iNullOffset (see page 1576)	This is iNullOffset, a member of class TFLDDesc.
	iOffset (see page 1576)	This is iOffset, a member of class TFLDDesc.
	iSubType (see page 1576)	This is iSubType, a member of class TFLDDesc.
	iUnits1 (see page 1577)	This is iUnits1, a member of class TFLDDesc.
	iUnits2 (see page 1577)	This is iUnits2, a member of class TFLDDesc.
	szName (see page 1577)	This is szName, a member of class TFLDDesc.

25.1.4.2 TFLDDesc Fields

The fields of the TFLDDesc class are listed here.

Public Fields

	Name	Description
	FCalcField (see page 1572)	This is FCalcField, a member of class TFLDDesc.
	FFldNum (see page 1572)	This is FFldNum, a member of class TFLDDesc.
	FFLDRights (see page 1572)	This is FFLDRights, a member of class TFLDDesc.
	FFldType (see page 1572)	This is FFldType, a member of class TFLDDesc.
	FFLDVchk (see page 1573)	This is FFLDVchk, a member of class TFLDDesc.
	FLen (see page 1573)	This is FLen, a member of class TFLDDesc.
	FName (see page 1573)	This is FName, a member of class TFLDDesc.
	FNullOffset (see page 1573)	This is FNullOffset, a member of class TFLDDesc.

	FOffset (see page 1573)	This is FOffset, a member of class TFLDDesc.
	FSubType (see page 1574)	This is FSubType, a member of class TFLDDesc.
	FUnits1 (see page 1574)	This is FUnits1, a member of class TFLDDesc.
	FUnits2 (see page 1574)	This is FUnits2, a member of class TFLDDesc.

25.1.4.2.1 TFLDDesc.FCalcField Field

Pascal

```
FCalcField: WordBool;
```

C++

```
WordBool FCalcField;
```

Description

This is FCalcField, a member of class TFLDDesc.

25.1.4.2.2 TFLDDesc.FFldNum Field

Pascal

```
FFldNum: Word;
```

C++

```
Word FFldNum;
```

Description

This is FFldNum, a member of class TFLDDesc.

25.1.4.2.3 TFLDDesc.FFLDRights Field

Pascal

```
FFLDRights: FLDRights;
```

C++

```
FLDRights FFLDRights;
```

Description

This is FFLDRights, a member of class TFLDDesc.

25.1.4.2.4 TFLDDesc.FFldType Field

Pascal

```
FFldType: Word;
```

C++

```
Word FFldType;
```

Description

This is FFldType, a member of class TFLDDesc.

25.1.4.2.5 TFLDDesc.FFLDVchk Field

Pascal

```
FFLDVchk: FLDVchk;
```

C++

```
FLDVchk FFLDVchk;
```

Description

This is FFLDVchk, a member of class TFLDDesc.

25.1.4.2.6 TFLDDesc.FLen Field

Pascal

```
FLen: LongWord;
```

C++

```
LongWord FLen;
```

Description

This is FLen, a member of class TFLDDesc.

25.1.4.2.7 TFLDDesc.FName Field

Pascal

```
FName: string;
```

C++

```
UnicodeString FName;
```

Description

This is FName, a member of class TFLDDesc.

25.1.4.2.8 TFLDDesc.FNullOffset Field

Pascal

```
FNullOffset: Word;
```

C++

```
Word FNullOffset;
```

Description

This is FNullOffset, a member of class TFLDDesc.

25.1.4.2.9 TFLDDesc.FOffset Field

Pascal

```
FOffset: Word;
```

C++

```
Word FOffset;
```

Description

This is FOffset, a member of class TFLDDesc.

25.1.4.2.10 TFLDDesc.FSubType Field

Pascal

```
FSubType: Word;
```

C++

```
Word FSubType;
```

Description

This is FSubType, a member of class TFLDDesc.

25.1.4.2.11 TFLDDesc.FUnits1 Field

Pascal

```
FUnits1: SmallInt;
```

C++

```
SmallInt FUnits1;
```

Description

This is FUnits1, a member of class TFLDDesc.

25.1.4.2.12 TFLDDesc.FUnits2 Field

Pascal

```
FUnits2: SmallInt;
```

C++

```
SmallInt FUnits2;
```

Description

This is FUnits2, a member of class TFLDDesc.

25.1.4.3 TFLDDesc Properties

The properties of the TFLDDesc class are listed here.

Public Properties

	Name	Description
	bCalcField (see page 1575)	This is bCalcField, a member of class TFLDDesc.
	efldrRights (see page 1575)	This is efldrRights, a member of class TFLDDesc.
	efldvVchk (see page 1575)	This is efldvVchk, a member of class TFLDDesc.
	iFldNum (see page 1575)	This is iFldNum, a member of class TFLDDesc.
	iFldType (see page 1576)	This is iFldType, a member of class TFLDDesc.
	iLen (see page 1576)	This is iLen, a member of class TFLDDesc.
	iNullOffset (see page 1576)	This is iNullOffset, a member of class TFLDDesc.
	iOffset (see page 1576)	This is iOffset, a member of class TFLDDesc.

	iSubType (see page 1576)	This is iSubType, a member of class TFLDDesc.
	iUnits1 (see page 1577)	This is iUnits1, a member of class TFLDDesc.
	iUnits2 (see page 1577)	This is iUnits2, a member of class TFLDDesc.
	szName (see page 1577)	This is szName, a member of class TFLDDesc.

25.1.4.3.1 TFLDDesc.bCalcField Property

Pascal

```
property bCalcField: WordBool;
```

C++

```
__property WordBool bCalcField;
```

Description

This is bCalcField, a member of class TFLDDesc.

25.1.4.3.2 TFLDDesc.efldrRights Property

Pascal

```
property efldrRights: FLDRights;
```

C++

```
__property FLDRights efldrRights;
```

Description

This is efldrRights, a member of class TFLDDesc.

25.1.4.3.3 TFLDDesc.efldvVchk Property

Pascal

```
property efldvVchk: FLDVchk;
```

C++

```
__property FLDVchk efldvVchk;
```

Description

This is efldvVchk, a member of class TFLDDesc.

25.1.4.3.4 TFLDDesc.iFldNum Property

Pascal

```
property iFldNum: Word;
```

C++

```
__property Word iFldNum;
```

Description

This is iFldNum, a member of class TFLDDesc.

25.1.4.3.5 TFLDDesc.iFldType Property

Pascal

```
property iFldType: Word;
```

C++

```
__property Word iFldType;
```

Description

This is iFldType, a member of class TFLDDesc.

25.1.4.3.6 TFLDDesc.iLen Property

Pascal

```
property iLen: LongWord;
```

C++

```
__property LongWord iLen;
```

Description

This is iLen, a member of class TFLDDesc.

25.1.4.3.7 TFLDDesc.iNullOffset Property

Pascal

```
property iNullOffset: Word;
```

C++

```
__property Word iNullOffset;
```

Description

This is iNullOffset, a member of class TFLDDesc.

25.1.4.3.8 TFLDDesc.iOffset Property

Pascal

```
property iOffset: Word;
```

C++

```
__property Word iOffset;
```

Description

This is iOffset, a member of class TFLDDesc.

25.1.4.3.9 TFLDDesc.iSubType Property

Pascal

```
property iSubType: Word;
```

C++

```
__property Word iSubType;
```

Description

This is iSubType, a member of class TFLDDesc.

25.1.4.3.10 TFLDDesc.iUnits1 Property

Pascal

```
property iUnits1: SmallInt;
```

C++

```
__property SmallInt iUnits1;
```

Description

This is iUnits1, a member of class TFLDDesc.

25.1.4.3.11 TFLDDesc.iUnits2 Property

Pascal

```
property iUnits2: SmallInt;
```

C++

```
__property SmallInt iUnits2;
```

Description

This is iUnits2, a member of class TFLDDesc.

25.1.4.3.12 TFLDDesc.szName Property

Pascal

```
property szName: string;
```

C++

```
__property UnicodeString szName;
```

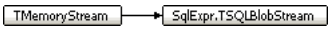
Description

This is szName, a member of class TFLDDesc.

25.1.5 TSQLBlobStream Class

TSQLBlobStream lets applications read from or write to BLOB fields in a client dataset.

Class Hierarchy



Pascal

```
TSQLBlobStream = class(TMemoryStream);
```

C++

```
class TSQLBlobStream : public TMemoryStream;
```

File

SqlExpr

Description

Use TSQLBlobStream to read the value of a Binary Large Object (BLOB) field in a dbExpress dataset (TSQLLDataSet, TSQLTable (see page 1680), TSQLQuery (see page 1654), or TSQLStoredProc (see page 1671)). BLOB fields are TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField. BLOB fields in client datasets use client BLOB streams to implement many of their data access properties and methods.

TSQLBlobStream allows objects that have no specialized knowledge of how data is stored in a BLOB field to read such data by employing the uniform stream interface.

To use an SQL BLOB stream, create an instance of TSQLBlobStream, use the methods of the stream to read the data, and then free the BLOB stream. If you use the same instance of TSQLBlobStream to access data from more than one record, you must call the ReadBlobData (see page 1580) method to update the stream when the current record changes.

Warning: Although TSQLBlobStream inherits several write methods from TMemoryStream, it can't be used to write data back to the dataset because dbExpress datasets are read-only. Writing to TSQLBlobStream simply alters the in-memory buffer that contains a BLOB field's data.

See Also

TBlobField

TGraphicField

TMemoField

TCustomSQLDataSet (see page 1520)

TSQLDataSet (see page 1631)

TSQLTable (see page 1680)

TSQLQuery (see page 1654)

TSQLStoredProc (see page 1671)

25.1.5.1 TSQLBlobStream Members

The following tables list the members exposed by TSQLBlobStream.

Public Methods

	Name	Description
	Create (see page 1578)	Creates and initializes an instance of TSQLBlobStream (see page 1577).
	Destroy (see page 1579)	Frees an instance of TSQLBlobStream (see page 1577).
	ReadBlobData (see page 1580)	Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

25.1.5.2 TSQLBlobStream.Create Constructor

Creates and initializes an instance of TSQLBlobStream (see page 1577).

Pascal

```
constructor Create(Field: TBlobField; Mode: TBlobStreamMode = bmRead);
```

C++

```
__fastcall TSQLBlobStream(TBlobField Field, TBlobStreamMode Mode = bmRead);
```

Description

Call Create to obtain an instance of TSQLBlobStream (see page 1577) for reading from a specific TBlobField object in a TCustomSQLDataSet (see page 1520) descendant.

Field specifies the BLOB field from which to read data.

The Mode parameter is provided for compatibility with other BLOB stream constructors. However, in TSQLBlobStream (see page 1577), the Mode parameter is ignored because dbExpress datasets are read-only. SQL Blob streams can only be used to read BLOB data.

Create links the TSQLBlobStream (see page 1577) to the field object specified by the Field parameter. It then calls ReadBlobData (see page 1580) to read the information from the BLOB field into a memory buffer, where it is available for reading.

Warning: If you write to the SQLBlobStream using inherited methods, you are overwriting the in-memory buffer that is filled by the constructor. This will cause subsequent read operations to provide inaccurate information, because the BLOB stream no longer accurately reflects the contents of the BLOB field.

Instead of calling the TSQLBlobStream (see page 1577) constructor, you can also use the dataset's CreateBlobStream method.

See Also

CreateBlobStream (see page 1528)

ReadBlobData (see page 1580)

25.1.5.3 TSQLBlobStream.Destroy Destructor

Frees an instance of TSQLBlobStream (see page 1577).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLBlobStream();
```

Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TBlobStream object is not nil and only then calls Destroy.

Destroy the TSQLBlobStream (see page 1577) object by calling Free when it is no longer needed for reading from the BLOB field. BLOB streams should be destroyed when the dataset changes the current record.

See Also

System::TObject::Free (see page 3372)

OnDataChange

25.1.5.4 TSQLBlobStream Methods

The methods of the TSQLBlobStream class are listed here.

Public Methods

	Name	Description
	ReadBlobData (see page 1580)	Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

25.1.5.4.1 TSQLBlobStream.ReadBlobData Method

Fills the BLOB stream's memory buffer with the contents of the associated BLOB field.

Pascal

```
procedure ReadBlobData;
```

C++

```
__fastcall ReadBlobData();
```

Description

Call ReadBlobData to refresh the contents of the BLOB stream when the current record in the dataset changes. ReadBlobData reads the contents of a BLOB field into the SQL BLOB stream's in-memory buffer and sets the Size property to the number of bytes of BLOB data. It then sets Position to 0 so that the stream is ready for you to read the BLOB data.

Note: It is not necessary to call ReadBlobData when you first create a SQL BLOB stream, because the constructor already performs this task. You need only call ReadBlobData when the current record changes, so that the stream reflects the BLOB data for the new record.

See Also

Size

Position

Create (see page 1578)

25.1.6 TSQLConnection Class

TSQLConnection encapsulates a dbExpress connection to a database server.

Class Hierarchy

Pascal

```
TSQLConnection = class(TCustomConnection);
```

C++

```
class TSQLConnection : public TCustomConnection;
```

File

SqlExpr

Description

Use TSQLConnection to represent a dbExpress database connection. The connection provided by a single TSQLConnection component can be shared by multiple SQL dataset components through their TSQLConnection properties.

TSQLConnection interacts with a dbExpress driver and two files. The first, dbxdrivers.ini on Windows or dbxdrivers on Linux, lists the installed driver types (Interbase, Oracle, and so on) and, for each driver, lists the libraries (DLLs or shared objects) it requires

as well as the default settings for all connection parameters. The second file, dbxconnections.ini on Windows or dbxconnections on Linux, lists named sets of connection configurations. Each configuration represents a set of settings for TSQLConnection and describes a particular database connection.

You can use TSQLConnection to choose one of the named configurations in `dbxconnections.ini` and use it to connect to a database or to define new configurations. Use the ConnectionName (see page 1610) property to select an already created connection. To add and manage connections, use the Data Explorer (**View** ▸ **Data Explorer**) included in the RAD Studio IDE.

In addition to describing and establishing database connections, TSQLConnection surfaces properties, methods, and events for controlling server login, managing transactions, and fetching metadata from the server.

See Also

TCustomSQLDataSet (see page 1520)

TSQLDataSet (see page 1631)

TSQLTable (see page 1680)

TSQLQuery (see page 1654)

TSQLStoredProc (see page 1671)

Connecting to Databases: Overview

Connecting Directly to a Database Server

Accessing Schema Information

Connecting to the Database Server

Setting Up TSQLConnection

25.1.6.1 TSQLConnection Members

The following tables list the members exposed by TSQLConnection.

Protected Methods

	Name	Description
	CheckConnection (see page 1586)	This is CheckConnection, a member of class TSQLConnection.
V	CheckDisconnect (see page 1587)	This is CheckDisconnect, a member of class TSQLConnection.
V	ConnectionOptions (see page 1589)	This is ConnectionOptions, a member of class TSQLConnection.
V	DoConnect (see page 1589)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 1589)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 1591)	Returns the value of the Connected property.
	GetDataSet (see page 1591)	Returns a dataset, given its index in the DataSets property array.
V	Loaded (see page 1600)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 1601)	Forwards notification messages to all owned components.
	OpenSchema (see page 1601)	This is the overview for the OpenSchema method overload.
V	RegisterClient (see page 1602)	Registers an object that uses the connection component.
	SQLError (see page 1605)	This is SQLError, a member of class TSQLConnection.
V	UnRegisterClient (see page 1606)	Removes a registered client so that it is no longer associated with the connection component.

Protected Properties

	Name	Description
R	ConnectionRegistryFile (see page 1610)	This is ConnectionRegistryFile, a member of class TSQLConnection.
R	QuoteChar (see page 1623)	This is QuoteChar, a member of class TSQLConnection.

	SQLDllHandle (see page 1623)	This is SQLDllHandle, a member of class TSQLConnection.

Public Events

	Name	Description
R	TraceCallbackEvent (see page 1628)	Provides access to the callback function that executes for every SQL command passed to or from the server.

Public Methods

	Name	Description
	AddConnectNotification (see page 1585)	This is AddConnectNotification, a member of class TSQLConnection.
	BeginTransaction (see page 1585)	This is the overview for the BeginTransaction method overload.
	CloneConnection (see page 1587)	Returns a copy of the SQL connection object.
	CloseDataSets (see page 1587)	Closes all datasets associated with the SQL connection object component without disconnecting from the database.
	Commit (see page 1588)	Commits an open transaction.
	CommitFreeAndNil (see page 1588)	Commits an open transaction.
V	Create (see page 1583)	Creates and initializes an instance TSQLConnection (see page 1580).
V	Destroy (see page 1584)	Destroys the SQL connection object.
	Execute (see page 1590)	Executes an SQL command on the server.
	ExecuteDirect (see page 1590)	Executes an SQL command that does not include parameters.
	GetCommandTypes (see page 1591)	This is GetCommandTypes, a member of class TSQLConnection.
	GetDefaultSchemaName (see page 1592)	Returns the default schema name of the current object.
	GetFieldNames (see page 1592)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 1593)	This is the overview for the GetIndexNames method overload.
	GetLoginUsername (see page 1594)	Returns the username that is logged in on the current connection.
	GetPackageNames (see page 1594)	This is the overview for the GetPackageNames method overload.
	GetProcedureNames (see page 1595)	This is the overview for the GetProcedureNames method overload.
	GetProcedureParams (see page 1596)	This is the overview for the GetProcedureParams method overload.
	GetSchemaNames (see page 1598)	This is the overview for the GetSchemaNames method overload.
	GetServerMethodNames (see page 1598)	This is GetServerMethodNames, a member of class TSQLConnection.
	GetTableNames (see page 1598)	This is the overview for the GetTableNames method overload.
	HasTransaction (see page 1599)	This is HasTransaction, a member of class TSQLConnection.
	LoadParamsFromIniFile (see page 1600)	Sets the DriverName (see page 1612) and Params (see page 1621) properties to the values associated with the ConnectionName (see page 1610) property.
	RemoveConnectNotification (see page 1602)	This is RemoveConnectNotification, a member of class TSQLConnection.
	Rollback (see page 1603)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	RollbackFreeAndNil (see page 1603)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	RollbackIncompleteFreeAndNil (see page 1604)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	SetTraceEvent (see page 1605)	Associates the SQL connection with a callback function that executes for every SQL command passed to or from the server.
	StartTransaction (see page 1605)	Initiates a new transaction in the associated database.

Public Properties

	Name	Description
R	ActiveStatements (see page 1607)	Indicates the number of statements currently active with the database server.
	AutoClone (see page 1608)	Specifies whether the SQL connection automatically clones database connections when they are needed.
R	ConnectionData (see page 1609)	This is ConnectionData, a member of class TSQLConnection.
	ConnectionState (see page 1611)	Indicates the current state of the TSQLConnection (see page 1580) object.
R	DataSets (see page 1611)	Lists all active datasets for the connection component.
R	DBXConnection (see page 1611)	Provides an interface to the dbExpress object that represents a connection.
R	InTransaction (see page 1613)	Indicates whether a transaction is in progress.
	LocaleCode (see page 1615)	Identifies the locale that is used for sorting items from datasets that use this connection.

R	MaxStmtsPerConn (see page 1620)	Indicates the server limit on the number of statements supported by a single database connection.
R	MetaData (see page 1620)	Provides an interface to the dbExpress object that supplies server metadata.
R	MultipleTransactionsSupported (see page 1621)	Indicates whether the database server supports multiple transactions.
	ParamsLoaded (see page 1622)	Indicates when parameter values have been loaded from dbxconnections.ini.
	SQLHourGlass (see page 1623)	Specifies whether the cursor should change to crSQLWait during lengthy operations.
R	TransactionsSupported (see page 1624)	Indicates whether the database server supports transactions.

Published Events

	Name	Description
	AfterConnect (see page 1626)	Occurs after a connection is established.
	AfterDisconnect (see page 1626)	Occurs after the connection closes.
	BeforeConnect (see page 1626)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1627)	Occurs immediately before the connection closes.
	OnLogin (see page 1627)	Occurs after the BeforeConnect (see page 1626) event and before a connection is established.

Published Properties

	Name	Description
	Connected (see page 1609)	Specifies whether the connection is active.
	ConnectionName (see page 1610)	Names the connection configuration.
	DriverName (see page 1612)	Specifies the database driver associated with the SQL connection.
	GetDriverFunc (see page 1613)	Specifies the function that returns the top-level dbExpress driver.
	KeepConnection (see page 1614)	Specifies whether the connection stays active when no datasets are open.
	LibraryName (see page 1614)	Specifies the dbExpress driver that TSQLConnection (see page 1580) uses to establish a connection.
	LoadParamsOnConnect (see page 1615)	Specifies whether the TSQLConnection (see page 1580) component loads the configuration named by ConnectionName (see page 1610) immediately before connecting to the server.
	LoginPrompt (see page 1616)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 1621)	Lists the connection parameters.
	TableScope (see page 1623)	Indicates what types of tables are returned when fetching schema information about tables.
	VendorLib (see page 1625)	Specifies the client software library (DLL or shared object) supplied by the database vendor.

25.1.6.2 TSQLConnection.Create Constructor

Creates and initializes an instance TSQLConnection (see page 1580).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLConnection(TComponent * AOwner);
```

Description

Use Create to instantiate TSQLConnection (see page 1580) at runtime. SQL connections added to forms or data modules at design time are created automatically.

The AOwner parameter is the component, typically a form or data module, that is responsible for freeing the SQL connection.

See Also

Destroy (see page 1584)

Owner

25.1.6.3 TSQLConnection.Destroy Destructor

Destroys the SQL connection object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLConnection();
```

Description

To free a TSQLConnection (see page 1580) instance, do not call Destroy. Instead, call Free. Free verifies that the ADO connection is not nil, and only then calls Destroy.

There is no need to explicitly free instances of TSQLConnection (see page 1580) that were created with an Owner.

See Also

Owner

Create (see page 1583)

delete

System::TObject::Free (see page 3372)

25.1.6.4 TSQLConnection Methods

The methods of the TSQLConnection class are listed here.

Protected Methods

	Name	Description
	CheckConnection (see page 1586)	This is CheckConnection, a member of class TSQLConnection.
V	CheckDisconnect (see page 1587)	This is CheckDisconnect, a member of class TSQLConnection.
V	ConnectionOptions (see page 1589)	This is ConnectionOptions, a member of class TSQLConnection.
V	DoConnect (see page 1589)	Provides the interface for a method that opens a connection.
V	DoDisconnect (see page 1589)	Provides the interface for a method that terminates the connection.
V	GetConnected (see page 1591)	Returns the value of the Connected property.
	GetDataSet (see page 1591)	Returns a dataset, given its index in the DataSets property array.
V	Loaded (see page 1600)	Initializes the connection component after all objects in the form or data module have been loaded.
V	Notification (see page 1601)	Forwards notification messages to all owned components.
	OpenSchema (see page 1601)	This is the overview for the OpenSchema method overload.
V	RegisterClient (see page 1602)	Registers an object that uses the connection component.
	SQLError (see page 1605)	This is SQLError, a member of class TSQLConnection.
V	UnRegisterClient (see page 1606)	Removes a registered client so that it is no longer associated with the connection component.

Public Methods

	Name	Description
	AddConnectNotification (see page 1585)	This is AddConnectNotification, a member of class TSQLConnection.
	BeginTransaction (see page 1585)	This is the overview for the BeginTransaction method overload.
	CloneConnection (see page 1587)	Returns a copy of the SQL connection object.

	CloseDataSets (see page 1587)	Closes all datasets associated with the SQL connection object component without disconnecting from the database.
	Commit (see page 1588)	Commits an open transaction.
	CommitFreeAndNil (see page 1588)	Commits an open transaction.
	Execute (see page 1590)	Executes an SQL command on the server.
	ExecuteDirect (see page 1590)	Executes an SQL command that does not include parameters.
	GetCommandTypes (see page 1591)	This is GetCommandTypes, a member of class TSQLConnection.
	GetDefaultSchemaName (see page 1592)	Returns the default schema name of the current object.
	GetFieldNames (see page 1592)	This is the overview for the GetFieldNames method overload.
	GetIndexNames (see page 1593)	This is the overview for the GetIndexNames method overload.
	GetLoginUsername (see page 1594)	Returns the username that is logged in on the current connection.
	GetPackageNames (see page 1594)	This is the overview for the GetPackageNames method overload.
	GetProcedureNames (see page 1595)	This is the overview for the GetProcedureNames method overload.
	GetProcedureParams (see page 1596)	This is the overview for the GetProcedureParams method overload.
	GetSchemaNames (see page 1598)	This is the overview for the GetSchemaNames method overload.
	GetServerMethodNames (see page 1598)	This is GetServerMethodNames, a member of class TSQLConnection.
	GetTableNames (see page 1598)	This is the overview for the GetTableNames method overload.
	HasTransaction (see page 1599)	This is HasTransaction, a member of class TSQLConnection.
	LoadParamsFromIniFile (see page 1600)	Sets the DriverName (see page 1612) and Params (see page 1621) properties to the values associated with the ConnectionName (see page 1610) property.
	RemoveConnectNotification (see page 1602)	This is RemoveConnectNotification, a member of class TSQLConnection.
	Rollback (see page 1603)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	RollbackFreeAndNil (see page 1603)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	RollbackIncompleteFreeAndNil (see page 1604)	Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.
	SetTraceEvent (see page 1605)	Associates the SQL connection with a callback function that executes for every SQL command passed to or from the server.
	StartTransaction (see page 1605)	Initiates a new transaction in the associated database.

25.1.6.4.1 TSQLConnection.AddConnectNotification Method

Pascal

```
procedure AddConnectNotification(Listener: TObject; Event: TConnectChangeEvent);
```

C++

```
__fastcall AddConnectNotification(TObject * Listener, TConnectChangeEvent Event);
```

Description

This is AddConnectNotification, a member of class TSQLConnection.

25.1.6.4.2 BeginTransaction Method

This is the overview for the BeginTransaction method overload.

Overload List

	Name	Description
	TSQLConnection.BeginTransaction () (see page 1586)	Initiates a new transaction in the associated database.
	TSQLConnection.BeginTransaction (TDBXIsolation) (see page 1586)	Initiates a new transaction in the associated database.

25.1.6.4.2.1 TSQLConnection.BeginTransaction Method ()

Initiates a new transaction in the associated database.

Pascal

```
function BeginTransaction: TDBXTransaction; overload;
function BeginTransaction(Isolation: TDBXIsolation): TDBXTransaction; overload;
```

C++

```
__fastcall TDBXTransaction BeginTransaction();
__fastcall TDBXTransaction BeginTransaction(TDBXIsolation Isolation);
```

Description

Call BeginTransaction to start a new transaction against a database server that supports transactions. (MySQL servers do not support transactions.) You can check the TransactionsSupported (see page 1624) property to determine whether the server supports transactions. If the server does not support transactions, BeginTransaction does nothing.

The method returns a TDBXTransactioninstance.

If the transaction is successfully started, BeginTransaction sets the InTransaction (see page 1613) property to true.

Updates, insertions, and deletions that take place after a call to BeginTransaction are enlisted in that transaction until another overlapping transaction is started or until the application calls CommitFreeAndNil (see page 1588) to save the changes or RollbackFreeAndNil (see page 1603) to cancel them.

Tip: If the server does not support multiple transactions, check the status of the InTransaction (see page 1613) property to be sure that another transaction is not already in progress.

Tip: When working with overlapped transactions, you can indicate the transaction to which a dataset belongs by setting its TransactionLevel property to the transaction identifier you assign in this method.

See Also

InTransaction (see page 1613)

CommitFreeAndNil (see page 1588)

RollbackFreeAndNil (see page 1603)

Params (see page 1621)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

25.1.6.4.3 TSQLConnection.CheckConnection Method

Pascal

```
procedure CheckConnection(eFlag: eConnectFlag);
```

C++

```
__fastcall CheckConnection(eConnectFlag eFlag);
```

Description

This is CheckConnection, a member of class TSQLConnection.

25.1.6.4.4 TSQLConnection.CheckDisconnect Method

Pascal

```
procedure CheckDisconnect; virtual;
```

C++

```
virtual __fastcall CheckDisconnect();
```

Description

This is CheckDisconnect, a member of class TSQLConnection.

25.1.6.4.5 TSQLConnection.CloneConnection Method

Returns a copy of the SQL connection object.

Pascal

```
function CloneConnection: TSQLConnection;
```

C++

```
__fastcall TSQLConnection CloneConnection();
```

Description

Call CloneConnection to create a duplicate of this TSQLConnection (see page 1580) object. CloneConnection instantiates a new TSQLConnection (see page 1580) object and assigns its properties to match those of this TSQLConnection (see page 1580) object.

The returned TSQLConnection (see page 1580) is owned by this TSQLConnection (see page 1580). That is, this SQL connection component frees the clone from its own destructor.

See Also

AutoClone (see page 1608)

25.1.6.4.6 TSQLConnection.CloseDataSets Method

Closes all datasets associated with the SQL connection object component without disconnecting from the database.

Pascal

```
procedure CloseDataSets;
```

C++

```
__fastcall CloseDataSets();
```

Description

Call CloseDataSets to close all active datasets without disconnecting from the database server. Ordinarily, when an application calls Close (see page 3538), all datasets are closed, and the connection to the database server is dropped. Calling CloseDataSets instead of Close (see page 3538) ensures that an application can close all active datasets without having to reconnect to the database server at a later time.

See Also

Close

Close

DataSets (see page 1611)

KeepConnection (see page 1614)

25.1.6.4.7 TSQLConnection.Commit Method

Commits an open transaction.

Pascal

```
procedure Commit(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall Commit(TTransactionDesc TransDesc);
```

Description

Call Commit to save any changes made during the transaction described by TransDesc and to end that transaction.

TransDesc is the transaction descriptor supplied to the StartTransaction (see page 1605) method when starting the transaction.

If the specified transaction is the only current transaction, Commit sets the InTransaction (see page 1613) property to false after committing the transaction.

Note: This method has been deprecated.

See Also

InTransaction (see page 1613)

StartTransaction (see page 1605)

Rollback (see page 1603)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

25.1.6.4.8 TSQLConnection.CommitFreeAndNil Method

Commits an open transaction.

Pascal

```
procedure CommitFreeAndNil(var Transaction: TDBXTransaction);
```

C++

```
__fastcall CommitFreeAndNil(TDBXTransaction Transaction);
```

Description

Call CommitFreeAndNil to save any changes made during the transaction described by **Transaction**and to end that transaction. **Transaction** is the TDBXTransaction instance returned by the BeginTransaction (see page 1586) method when starting the transaction.

If the specified transaction is the only current transaction, CommitFreeAndNil sets the InTransaction (see page 1613) property to `false` after committing the transaction.

Any active transactions that were begun after this transaction are effectively committed and freed as well.

The var **Transaction** instance is set to `nil`.

Note: This method raises an exception if Transaction

is not an active transaction.

See Also

InTransaction (see page 1613)

BeginTransaction (see page 1586)

RollbackFreeAndNil (see page 1603)

RollbackIncompleteFreeAndNil (see page 1604)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

25.1.6.4.9 TSQLConnection.ConnectionOptions Method

Pascal

```
procedure ConnectionOptions; virtual;
```

C++

```
virtual __fastcall ConnectionOptions();
```

Description

This is ConnectionOptions, a member of class TSQLConnection.

25.1.6.4.10 TSQLConnection.DoConnect Method

Provides the interface for a method that opens a connection.

Pascal

```
procedure DoConnect; override;
```

C++

```
virtual __fastcall DoConnect();
```

Description

The Connected property uses DoConnect to establish a connection. Descendant classes override the DoConnect method to establish their connection as appropriate. As implemented in TCustomConnection, DoConnect does nothing.

See Also

Connected

DoDisconnect

25.1.6.4.11 TSQLConnection.DoDisconnect Method

Provides the interface for a method that terminates the connection.

Pascal

```
procedure DoDisconnect; override;
```

C++

```
virtual __fastcall DoDisconnect();
```

Description

The Connected property uses DoDisconnect to close a connection. Descendant classes override the DoDisconnect method to

drop a connection. As implemented in TCustomConnection, DoDisconnect does nothing.

See Also

Connected

DoConnect

25.1.6.4.12 TSQLConnection.Execute Method

Executes an SQL command on the server.

Pascal

```
function Execute(const SQL: UnicodeString; Params: TParams; ResultSet: TPSResult = nil):
Integer;
```

C++

```
__fastcall int Execute(const UnicodeString SQL, TParams Params, TPSResult ResultSet = nil);
```

Description

Call Execute to execute a single command on the server without the overhead of using an SQL dataset. Execute is especially useful for statements that do not return a result set such as data definition language (DDL) statements.

SQL is the command to execute.

Params (see page 1621) is a TParams object that contains any parameters used by the SQL statement. Parameter binding is by index only (not by name), so the order of parameters is important and the order of the TParam objects in Params (see page 1621) corresponds to the order of the parameters in the SQL statement. Use properties and methods of TParams to create a TParams object, add TParam objects for each parameter, and assign parameter properties such as datatype and value. If the SQL statement does not include any parameters, pass a nil (Delphi) or NULL (C++) value for Params (see page 1621).

ResultSet is a pointer to a variable of type TCustomSQLDataSet (see page 1520)*. If the SQL statement returns a cursor, Execute creates a new TCustomSQLDataSet (see page 1520) instance and populates it with the result set. The new TCustomSQLDataSet (see page 1520) reference is returned as the value to which ResultSet points. If the SQL statement does not return a result set, pass a nil (Delphi) or NULL (C++) value for ResultSet. TPSResult is is set to Pointer for native code and TObject (see page 3354) for managed code.

Execute returns the number of records affected by executing the SQL statement.

Tip: If the SQL statement does not include any parameters, you can use the ExecuteDirect (see page 1590) method instead.

See Also

ExecSQL (see page 1636)

ExecuteDirect (see page 1590)

25.1.6.4.13 TSQLConnection.ExecuteDirect Method

Executes an SQL command that does not include parameters.

Pascal

```
function ExecuteDirect(const SQL: UnicodeString): Integer;
```

C++

```
__fastcall int ExecuteDirect(const UnicodeString SQL);
```

Description

Call ExecuteDirect to execute a single parameterless command on the server without the overhead of using an SQL dataset or

preparing the statement before it executes. If the statement ordinarily returns a cursor, ExecuteDirect executes the statement, but does not return a cursor.

SQL is the command to execute.

ExecuteDirect returns the number of rows that were affected by the execution of the SQL query.

Note: To execute an SQL statement that includes parameters or that returns a cursor, use the Execute (see page 1590) method instead.

See Also

ExecSQL (see page 1636)

Execute (see page 1590)

25.1.6.4.14 TSQLConnection.GetCommandTypes Method

Pascal

```
procedure GetCommandTypes(List: TWideStrings);
```

C++

```
__fastcall GetCommandTypes(TWideStrings List);
```

Description

This is GetCommandTypes, a member of class TSQLConnection.

25.1.6.4.15 TSQLConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

25.1.6.4.16 TSQLConnection.GetDataSet Method

Returns a dataset, given its index in the DataSets property array.

Pascal

```
function GetDataSet(Index: Integer): TCustomSQLDataSet; reintroduce;
```

C++

```
__fastcall TCustomSQLDataSet GetDataSet(int Index);
```

Description

GetDataSet is the protected read implementation of the DataSets property.

See Also

DataSets

25.1.6.4.17 TSQLConnection.GetDefaultSchemaName Method

Returns the default schema name of the current object.

Pascal

```
function GetDefaultSchemaName: UnicodeString;
```

C++

```
__fastcall UnicodeString GetDefaultSchemaName();
```

25.1.6.4.18 GetFieldNames Method

This is the overview for the GetFieldNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetFieldNames (string, string, TStrings) (see page 1592)	Populates a string list with the names of fields in a table.
	TSQLConnection.GetFieldNames (string, TStrings) (see page 1592)	Populates a string list with the names of fields in a table.
	TSQLConnection.GetFieldNames (WideString, TWideStrings) (see page 1592)	Populates a string list with the names of fields in a table.
	TSQLConnection.GetFieldNames (WideString, WideString, TWideStrings) (see page 1592)	Populates a string list with the names of fields in a table.

25.1.6.4.18.1 TSQLConnection.GetFieldNames Method (WideString, TWideStrings)

Populates a string list with the names of fields in a table.

Pascal

```
procedure GetFieldNames(const TableName: string; List: TStrings); overload; deprecated;
procedure GetFieldNames(const TableName: string; SchemaName: string; List: TStrings);
overload; deprecated;
procedure GetFieldNames(const TableName: WideString; SchemaName: WideString; List:
TWideStrings); overload;
procedure GetFieldNames(const TableName: WideString; List: TWideStrings); overload;
```

C++

```
__fastcall GetFieldNames(const UnicodeString TableName, TStrings List);
__fastcall GetFieldNames(const UnicodeString TableName, UnicodeString SchemaName, TStrings
List);
__fastcall GetFieldNames(const BSTR TableName, BSTR SchemaName, TWideStrings List);
__fastcall GetFieldNames(const BSTR TableName, TWideStrings List);
```

Description

Call GetFieldNames to retrieve a list of fields in a table.

TableName names the table whose field names you want added to the list.

List is a TStrings descendant that receives the field names. Any existing strings are deleted from the list before GetFieldNames adds the names of all the fields in TableName.

For example, the following line fills a list box with the names of all fields in the Employee table:

```
SQLConnection1.GetFieldNames('Employee', ListBox1.Items);
SQLConnection1->GetFieldNames("Employee", ListBox1->Items);
```

See Also

GetProcedureNames (see page 1595)

GetPackageNames (see page 1594)

GetTableNames (see page 1599)

GetIndexNames (see page 1593)

GetProcedureParams (see page 1597)

SetSchemaInfo (see page 1550)

25.1.6.4.19 GetIndexNames Method

This is the overview for the GetIndexNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetIndexNames (string, string, TStrings) (see page 1593)	Populates a string list with the names of indexes on a table.
	TSQLConnection.GetIndexNames (string, TStrings) (see page 1593)	Populates a string list with the names of indexes on a table.
	TSQLConnection.GetIndexNames (UnicodeString, TWideStrings) (see page 1593)	Populates a string list with the names of indexes on a table.
	TSQLConnection.GetIndexNames (UnicodeString, UnicodeString, TWideStrings) (see page 1593)	Populates a string list with the names of indexes on a table.

25.1.6.4.19.1 TSQLConnection.GetIndexNames Method (UnicodeString, TWideStrings)

Populates a string list with the names of indexes on a table.

Pascal

```
procedure GetIndexNames(const TableName: string; List: TStrings); overload; deprecated;
procedure GetIndexNames(const TableName: string; const SchemaName: string; List: TStrings);
overload; deprecated;
procedure GetIndexNames(const TableName: UnicodeString; List: TWideStrings); overload;
procedure GetIndexNames(const TableName: UnicodeString; const SchemaName: UnicodeString; List:
TWideStrings); overload;
```

C++

```
__fastcall GetIndexNames(const UnicodeString TableName, TStrings List);
__fastcall GetIndexNames(const UnicodeString TableName, const UnicodeString SchemaName,
TStrings List);
__fastcall GetIndexNames(const UnicodeString TableName, TWideStrings List);
__fastcall GetIndexNames(const UnicodeString TableName, const UnicodeString SchemaName,
TWideStrings List);
```

Description

Call GetIndexNames to retrieve a list of the indexes defined for a table.

List is a TStrings descendant that receives the index names. Any existing strings are deleted from the list before GetIndexNames adds the names of all the indexes on TableName.

TableName names the table whose indexes you want added to the list.

For example, the following line fills a list box with the names of all indexes on the Employee table:

```
SQLConnection1.GetIndexNames('Employee', ListBox1.Items);
SQLConnection1->GetIndexNames("Employee", ListBox1->Items);
```

See Also

GetProcedureNames (see page 1595)

GetTableNames (see page 1599)

GetPackageNames (see page 1594)

GetFieldNames (see page 1592)

GetProcedureParams (see page 1597)

SetSchemaInfo (see page 1550)

Obtaining Information About Indexes

25.1.6.4.20 TSQLConnection.GetLoginUsername Method

Returns the username that is logged in on the current connection.

Pascal

```
function GetLoginUsername: UnicodeString;
```

C++

```
__fastcall UnicodeString GetLoginUsername();
```

25.1.6.4.21 GetPackageNames Method

This is the overview for the GetPackageNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetPackageNames (TStrings) (see page 1594)	Populates a string list with the names of all packages defined on the server.
	TSQLConnection.GetPackageNames (TWideStrings) (see page 1594)	Populates a string list with the names of all packages defined on the server.

25.1.6.4.21.1 TSQLConnection.GetPackageNames Method (TStrings)

Populates a string list with the names of all packages defined on the server.

Pascal

```
procedure GetPackageNames(List: TStrings); overload; deprecated;
procedure GetPackageNames(List: TWideStrings); overload;
```

C++

```
__fastcall GetPackageNames(TStrings List);
__fastcall GetPackageNames(TWideStrings List);
```

Description

Call GetPackageNames to retrieve a list of packages defined on the associated database. This method is only relevant to Oracle servers.

List is a TStrings descendant that receives the package names. Any existing strings are deleted from the list before GetPackageNames adds the names of all packages that the database defines.

The following example fills a list box with the names of all packages defined by the database:

```
SQLConnection1.GetPackageNames(ListBox1.Items);
SQLConnection1->GetPackageNames(ListBox1->Items);
```

See Also

GetIndexNames (see page 1593)

GetProcedureNames (see page 1595)

GetTableNames (see page 1599)

GetFieldNames (see page 1592)

GetProcedureParams (see page 1597)

SetSchemaInfo (see page 1550)

25.1.6.4.22 GetProcedureNames Method

This is the overview for the GetProcedureNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetProcedureNames (string, string, TStrings) (see page 1595)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (string, TStrings) (see page 1595)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (TStrings) (see page 1595)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (TWideStrings) (see page 1595)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (UnicodeString, TWideStrings) (see page 1595)	Populates a string list with the names of stored procedures in the database.
	TSQLConnection.GetProcedureNames (UnicodeString, UnicodeString, TWideStrings) (see page 1595)	Populates a string list with the names of stored procedures in the database.

25.1.6.4.22.1 TSQLConnection.GetProcedureNames Method (TStrings)

Populates a string list with the names of stored procedures in the database.

Pascal

```
procedure GetProcedureNames(List: TStrings); overload; deprecated;
procedure GetProcedureNames(const PackageName: string; List: TStrings); overload; deprecated;
procedure GetProcedureNames(const PackageName: string; const SchemaName: string; List:
TStrings); overload; deprecated;
procedure GetProcedureNames(List: TWideStrings); overload;
procedure GetProcedureNames(const PackageName: UnicodeString; List: TWideStrings); overload;
procedure GetProcedureNames(const PackageName: UnicodeString; const SchemaName: UnicodeString;
List: TWideStrings); overload;
```

C++

```
__fastcall GetProcedureNames(TStrings List);
__fastcall GetProcedureNames(const UnicodeString PackageName, TStrings List);
__fastcall GetProcedureNames(const UnicodeString PackageName, const UnicodeString SchemaName,
TStrings List);
__fastcall GetProcedureNames(TWideStrings List);
__fastcall GetProcedureNames(const UnicodeString PackageName, TWideStrings List);
__fastcall GetProcedureNames(const UnicodeString PackageName, const UnicodeString SchemaName,
TWideStrings List);
```

Description

Call GetProcedureNames to retrieve a list of stored procedures in the associated database.

List is a TStrings descendant that receives the stored procedure names. Any existing strings are deleted from the list before GetProcedureNames adds the names of all stored procedures in the database.

PackageName is the name of an Oracle package that contains the stored procedures. This syntax should only be used with Oracle servers when you only want the stored procedures from a specific package.

The following example fills a list box with the names of all stored procedures in the database:

```
SQLConnection1.GetProcedureNames(ListBox1.Items);
SQLConnection1->GetProcedureNames(ListBox1->Items);
```

See Also

GetIndexNames (see page 1593)

GetTableNames (see page 1599)

GetPackageNames (see page 1594)

GetFieldNames (see page 1592)

GetProcedureParams (see page 1597)

SetSchemaInfo (see page 1550)

25.1.6.4.23 GetProcedureParams Method

This is the overview for the GetProcedureParams method overload.

Overload List

	Name	Description
	TSQLConnection.GetProcedureParams (UnicodeString, TList) (see page 1597)	Populates a list with parameter descriptors for the parameters in a stored procedure.
	TSQLConnection.GetProcedureParams (UnicodeString, UnicodeString, TList) (see page 1597)	Populates a list with parameter descriptors for the parameters in a stored procedure.

	TSQLConnection.GetProcedureParams (UnicodeString, UnicodeString, UnicodeString, TList) (see page 1597)	Populates a list with parameter descriptors for the parameters in a stored procedure.

25.1.6.4.23.1 TSQLConnection.GetProcedureParams Method (UnicodeString, TList)

Populates a list with parameter descriptors for the parameters in a stored procedure.

Pascal

```
procedure GetProcedureParams(ProcedureName: UnicodeString; List: TList); overload;
procedure GetProcedureParams(ProcedureName: UnicodeString; PackageName: UnicodeString; List:
TList); overload;
procedure GetProcedureParams(ProcedureName: UnicodeString; PackageName: UnicodeString;
SchemaName: UnicodeString; List: TList); overload;
```

C++

```
__fastcall GetProcedureParams(UnicodeString ProcedureName, TList List);
__fastcall GetProcedureParams(UnicodeString ProcedureName, UnicodeString PackageName, TList
List);
__fastcall GetProcedureParams(UnicodeString ProcedureName, UnicodeString PackageName,
UnicodeString SchemaName, TList List);
```

Description

Call GetProcedureParams to retrieve information about the parameters of a specified stored procedure.

ProcedureName is the name of the stored procedure for whose parameters you want descriptions.

PackageName is the name of the Oracle package to which the stored procedure belongs. This parameter should only be used if TSQLConnection (see page 1580) connects to an Oracle server.

List is a TList object that receives the parameter descriptions. It should be an empty list on entry. For each parameter, GetProcedureParams adds a reference to an SPParamDesc (see page 1512) class (Delphi) or structure (C++). Each SPParamDesc (see page 1512) entity describes a parameter of a specified stored procedure, including its name, index, parameter type, field type, and so on. You are responsible for freeing the individual SPPParamDesc entities that GetProcedureParams dynamically allocates. (You can use FreeProcParams (see page 1696) to free all the entries and the list object as well).

Note: You can convert the list of SPParamDesc (see page 1512) entities to the more familiar TParams object by calling the global LoadParamListItems (see page 1698) procedure.

See Also

GetIndexNames (see page 1593)

GetTableNames (see page 1599)

GetPackageNames (see page 1594)

GetFieldNames (see page 1592)

GetProcedureNames (see page 1595)

SetSchemaInfo (see page 1550)

LoadParamListItems (see page 1698)

FreeProcParams (see page 1696)

25.1.6.4.24 GetSchemaNames Method

This is the overview for the GetSchemaNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetSchemaNames (TStrings) (see page 1598)	Returns the schema names for all objects in the given data source.
	TSQLConnection.GetSchemaNames (TWideStrings) (see page 1598)	Returns the schema names for all objects in the given data source.

25.1.6.4.24.1 TSQLConnection.GetSchemaNames Method (TStrings)

Returns the schema names for all objects in the given data source.

Pascal

```
procedure GetSchemaNames(List: TStrings); overload;
procedure GetSchemaNames(List: TWideStrings); overload;
```

C++

```
__fastcall GetSchemaNames(TStrings List);
__fastcall GetSchemaNames(TWideStrings List);
```

25.1.6.4.25 TSQLConnection.GetServerMethodNames Method

Pascal

```
procedure GetServerMethodNames(List: TWideStrings);
```

C++

```
__fastcall GetServerMethodNames(TWideStrings List);
```

Description

This is GetServerMethodNames, a member of class TSQLConnection.

25.1.6.4.26 GetTableNames Method

This is the overview for the GetTableNames method overload.

Overload List

	Name	Description
	TSQLConnection.GetTableNames (TStrings, Boolean) (see page 1599)	Populates a string list with the names of tables in the database.
	TSQLConnection.GetTableNames (TStrings, WideString, Boolean) (see page 1599)	Populates a string list with the names of tables in the database.
	TSQLConnection.GetTableNames (TWideStrings, Boolean) (see page 1599)	Populates a string list with the names of tables in the database.
	TSQLConnection.GetTableNames (TWideStrings, WideString, Boolean) (see page 1599)	Populates a string list with the names of tables in the database.

25.1.6.4.26.1 TSQLConnection.GetTableNames Method (TStrings, Boolean)

Populates a string list with the names of tables in the database.

Pascal

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False); overload;
procedure GetTableNames(List: TStrings; SchemaName: WideString; SystemTables: Boolean =
False); overload;
procedure GetTableNames(List: TWideStrings; SchemaName: WideString; SystemTables: Boolean =
False); overload;
procedure GetTableNames(List: TWideStrings; SystemTables: Boolean = False); overload;
```

C++

```
__fastcall GetTableNames(TStrings List, Boolean SystemTables = False);
__fastcall GetTableNames(TStrings List, BSTR SchemaName, Boolean SystemTables = False);
__fastcall GetTableNames(TWideStrings List, BSTR SchemaName, Boolean SystemTables = False);
__fastcall GetTableNames(TWideStrings List, Boolean SystemTables = False);
```

Description

Call GetTableNames to retrieve a list of tables in the associated database.

List is a TStrings descendant that receives the table names. Any existing strings are deleted from the list before GetTableNames adds the names of all tables in the database.

SystemTables specifies whether the list of table names should include only the database's system tables. If SystemTables is true, only the system tables are added to List. If SystemTables is false, the list is filled with any tables that match the criteria specified by the TableScope (see page 1623) property.

For example, the following line fills a list box with the names of all data tables in the database:

```
SQLConnection1.GetTableNames(ListBox1.Items, False);
SQLConnection1->GetTableNames(ListBox1->Items, false);
```

See Also

GetIndexNames (see page 1593)

GetProcedureParams (see page 1597)

GetFieldNames (see page 1592)

GetProcedureNames (see page 1595)

GetPackageNames (see page 1594)

SetSchemaInfo (see page 1550)

TableScope (see page 1623)

25.1.6.4.27 TSQLConnection.HasTransaction Method

Pascal

```
function HasTransaction(Transaction: TDBXTransaction): Boolean;
```

C++

```
__fastcall Boolean HasTransaction(TDBXTransaction Transaction);
```

Description

This is HasTransaction, a member of class TSQLConnection.

25.1.6.4.28 TSQLConnection.Loaded Method

Initializes the connection component after all objects in the form or data module have been loaded.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. It is called automatically after the form or data module that contains the connection component has loaded all its owned objects. TCustomConnection overrides the inherited method to establish a connection if the Connected property is true.

See Also

StreamedConnected

Loaded

25.1.6.4.29 TSQLConnection.LoadParamsFromIniFile Method

Sets the DriverName (see page 1612) and Params (see page 1621) properties to the values associated with the ConnectionName (see page 1610) property.

Pascal

```
procedure LoadParamsFromIniFile(FFileName: UnicodeString = '');
```

C++

```
__fastcall LoadParamsFromIniFile(UnicodeString FFileName = '');
```

Description

Call LoadParamsFromIniFile to ensure that the DriverName (see page 1612) and Params (see page 1621) properties accurately reflect the connection configuration for the ConnectionName (see page 1610) property that is stored in the file specified by the AFileName parameter. If the AFileName parameter is omitted, LoadParamsFromIniFile uses the dbxconnections.ini file. Otherwise, itis a fully-qualified path name to a file that has the same format as the dbxconnections.ini file. Typically, this file is created by making a copy of the dbxconnections.ini file under a different name.

In most cases, it is unnecessary to load connection configurations at runtime because when you set ConnectionName (see page 1610) at design time, the DriverName (see page 1612) and Params (see page 1621) properties are automatically set as well. However, LoadParamsFromIniFile lets you use a customized set of connection configurations at runtime that differ from those stored in the dbxconnections.ini file, or to temporarily override the DriverName (see page 1612) or connection parameter values at design time and then reestablish the defaults at runtime.

Note: Do not set the LoadParamsOnConnect (see page 1615) property to true if your application calls LoadParamsFromIniFile and specifies an alternate connections file. When LoadParamsOnConnect (see page 1615) is true, it loads the connection configuration from the dbxconnections.ini

file immediately before opening a connection, overriding any values loaded by this method.

See Also

ConnectionName (see page 1610)

LoadParamsOnConnect (see page 1615)

DriverName (see page 1612)

Params (see page 1621)

Setting Up TSQLConnection

Configuring TSQL Connection

25.1.6.4.30 TSQLConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

25.1.6.4.31 OpenSchema Method

This is the overview for the OpenSchema method overload.

Overload List

	Name	Description
	TSQLConnection.OpenSchema (TSchemaType, UnicodeString, TWideStrings) (see page 1601)	This is OpenSchema, a member of class TSQLConnection.
	TSQLConnection.OpenSchema (TSchemaType, UnicodeString, UnicodeString, TWideStrings) (see page 1602)	This is OpenSchema, a member of class TSQLConnection.

25.1.6.4.31.1 TSQLConnection.OpenSchema Method (TSchemaType, UnicodeString, TWideStrings)

Pascal

```
procedure OpenSchema(eKind: TSchemaType; sInfo: UnicodeString; List: TWideStrings); overload;
```

C++

```
__fastcall OpenSchema(TSchemaType eKind, UnicodeString sInfo, TWideStrings List);
```

Description

This is OpenSchema, a member of class TSQLConnection.

25.1.6.4.31.2 TSQLConnection.OpenSchema Method (TSchemaType, UnicodeString, UnicodeString, TWideStrings)

Pascal

```
procedure OpenSchema(eKind: TSchemaType; sInfo: UnicodeString; SSchemaName: UnicodeString;
List: TWideStrings); overload;
```

C++

```
__fastcall OpenSchema(TSchemaType eKind, UnicodeString sInfo, UnicodeString SSchemaName,
TWideStrings List);
```

Description

This is OpenSchema, a member of class TSQLConnection.

25.1.6.4.32 TSQLConnection.RegisterClient Method

Registers an object that uses the connection component.

Pascal

```
procedure RegisterClient(Client: TObject; Event: TConnectChangeEvent = nil); override;
```

C++

```
virtual __fastcall RegisterClient(TObject * Client, TConnectChangeEvent Event = nil);
```

Description

RegisterClient is used internally to associate objects with the connection component. Registered components can supply an event handler that responds when the connection is created or terminated.

Client is the object to register. If Client is a dataset, it is added to the DataSets property.

Event is an event handler that is called after the connection is established or before it is terminated.

Datasets call this method when they start using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are activated. For client datasets, this is when the RemoteServer property is assigned.

See Also

DataSets

UnRegisterClient

25.1.6.4.33 TSQLConnection.RemoveConnectNotification Method

Pascal

```
procedure RemoveConnectNotification(Listener: TObject);
```

C++

```
__fastcall RemoveConnectNotification(TObject * Listener);
```

Description

This is RemoveConnectNotification, a member of class TSQLConnection.

25.1.6.4.34 TSQLConnection.Rollback Method

Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.

Pascal

```
procedure Rollback(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall Rollback(TTransactionDesc TransDesc);
```

Description

Call Rollback to cancel any changes made during the transaction described by TransDesc and to end it.

TransDesc is the transaction descriptor supplied to the StartTransaction (see page 1605) method when starting the transaction.

If the specified transaction is the only current transaction, Rollback sets the InTransaction (see page 1613) property to false after canceling the transaction.

Note: This method has been deprecated.

See Also

InTransaction (see page 1613)

StartTransaction (see page 1605)

Commit (see page 1588)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

25.1.6.4.35 TSQLConnection.RollbackFreeAndNil Method

Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.

Pascal

```
procedure RollbackFreeAndNil(var Transaction: TDBXTransaction);
```

C++

```
__fastcall RollbackFreeAndNil(TDBXTransaction Transaction);
```

Description

Call RollbackFreeAndNil to cancel any changes made during the transaction described by Transaction and to end it. **Transaction** is the TDBXTransaction returned by the BeginTransaction (see page 1586) method when starting the transaction.

If the specified transaction is the only current transaction, RollbackFreeAndNil sets the InTransaction (see page 1613) property to false after canceling the transaction.

Any active transactions that were begun after this transaction are effectively rolled back and freed as well.

The var **Transaction** instance is set to `nil`.

Note: This method raises an exception if Transaction

is not an active transaction.

See Also

TDBXTransaction

InTransaction (see page 1613)

BeginTransaction (see page 1586)

CommitFreeAndNil (see page 1588)

RollbackIncompleteFreeAndNil (see page 1604)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

25.1.6.4.36 TSQLConnection.RollbackIncompleteFreeAndNil Method

Cancels all updates, insertions, and deletions for a specified transaction and ends the transaction.

Pascal

```
procedure RollbackIncompleteFreeAndNil(var Transaction: TDBXTransaction);
```

C++

```
__fastcall RollbackIncompleteFreeAndNil(TDBXTransaction Transaction);
```

Description

Call RollbackIncompleteFreeAndNil to cancel any changes made during the transaction described by Transaction and to end it. **Transaction** is the TDBXTransaction returned by the BeginTransaction (see page 1586) method when starting the transaction.

If the specified transaction is the only current transaction, RollbackFreeAndNil (see page 1603) sets the InTransaction (see page 1613) property to false after canceling the transaction.

Any active transactions that were begun after this transaction are effectively rolled back and freed as well.

The var **Transaction** instance is set to `nil`.

Unlike the RollbackFreeAndNil (see page 1603) procedure, this procedure does not throw an exception if **Transaction** is invalid or set to `nil`. This method is ideally used in a `finally` (or `except`) block. The `try` block uses either CommitFreeAndNil (see page 1588) or RollbackFreeAndNil (see page 1603), and the **Transaction** has already been completed, freed, and set to `nil`. If an exception is not raised in the `try` block, a call to RollbackIncompleteFreeAndNil in the `finally` block does nothing, because the **Transaction** instance is no longer valid. If an exception is raised, a call to RollbackIncompleteFreeAndNil in the `finally` block rolls back, frees, and sets the **Transaction** instance to `nil`.

See Also

TDBXTransaction

InTransaction (see page 1613)

BeginTransaction (see page 1586)

CommitFreeAndNil (see page 1588)

RollbackFreeAndNil (see page 1603)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

25.1.6.4.37 TSQLConnection.SetTraceEvent Method

Associates the SQL connection with a callback function that executes for every SQL command passed to or from the server.

Pascal

```
procedure SetTraceEvent(Event: TDBXTraceEvent);
```

C++

```
__fastcall SetTraceEvent(TDBXTraceEvent Event);
```

Description

Call SetTraceEvent to assign a callback that responds every time an SQL command passes between the associated dbExpress driver and the database server. The callback function becomes the value of the TraceCallbackEvent (see page 1628) property.

Event is the TDBXTraceEvent event that is called for every message that passes between the dbExpress driver and the SQL server.

Tip: Read (see page 3634) the TraceCallbackEvent (see page 1628) property to determine whether a callback is already assigned before calling the SetTraceEvent method. That way, you can avoid interfering with an existing callback when assigning a new one.

See Also

TSQLMonitor (see page 1643)

TraceCallbackEvent (see page 1628)

Debugging dbExpress Applications

Debugging dbExpress Applications using TSQLMonitor

25.1.6.4.38 TSQLConnection.SQLError Method

Pascal

```
procedure SQLError(Error: TDBXError);
```

C++

```
__fastcall SQLError(TDBXError Error);
```

Description

This is SQLError, a member of class TSQLConnection.

25.1.6.4.39 TSQLConnection.StartTransaction Method

Initiates a new transaction in the associated database.

Pascal

```
procedure StartTransaction(TransDesc: TTransactionDesc); deprecated;
```

C++

```
__fastcall StartTransaction(TTransactionDesc TransDesc);
```

Description

Call StartTransaction to start a new transaction against a database server that supports transactions. (MySQL servers do not support transactions.) You can check the TransactionsSupported (see page 1624) property to determine whether the server supports transactions. If the server does not support transactions, StartTransaction does nothing.

TransDesc describes the transaction you are starting. It lets you specify

A user-defined transaction identifier. If the server supports nested or overlapped transactions, you can initiate multiple transactions by calling StartTransaction and providing each transaction with its own unique identifier.

The transaction isolation level. This indicates the degree to which this transaction can "see" changes made by other transactions.

If the transaction is successfully started, StartTransaction sets the InTransaction (see page 1613) property to true.

Updates, insertions, and deletions that take place after a call to StartTransaction are enlisted in that transaction until another overlapping transaction is started or until the application calls Commit (see page 1588) to save the changes or Rollback (see page 1603) to cancel them.

Tip: If the server does not support multiple transactions, check the status of the InTransaction (see page 1613) property to be sure that another transaction is not already in progress.

Tip: When working with overlapped transactions, you can indicate the transaction to which a dataset belongs by setting its TransactionLevel property to the transaction identifier you assign in this method.

Note: This method has been deprecated.

See Also

InTransaction (see page 1613)

Commit (see page 1588)

Rollback (see page 1603)

Params (see page 1621)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

25.1.6.4.40 TSQLConnection.UnRegisterClient Method

Removes a registered client so that it is no longer associated with the connection component.

Pascal

```
procedure UnRegisterClient(Client: TObject); override;
```

C++

```
virtual __fastcall UnRegisterClient(TObject * Client);
```

Description

UnRegisterClient is used internally to remove objects from the internal list of objects that are informed when a connection is created or terminated.

Client is the object that is removed from the list. It should be an object that was previously registered by a call to RegisterClient. If Client is a dataset, it is removed from the DataSets property list.

Datasets call this method when they stop using the connection component. For BDE-enabled datasets, ADO datasets, and dbExpress datasets, this is when they are deactivated. For client datasets, this is when the RemoteServer property is changed.

See Also

DataSets

RegisterClient

25.1.6.5 TSQLConnection Properties

The properties of the TSQLConnection class are listed here.

Protected Properties

	Name	Description
R	ConnectionRegistryFile (see page 1610)	This is ConnectionRegistryFile, a member of class TSQLConnection.
R	QuoteChar (see page 1623)	This is QuoteChar, a member of class TSQLConnection.
	SQLDllHandle (see page 1623)	This is SQLDllHandle, a member of class TSQLConnection.

Public Properties

	Name	Description
R	ActiveStatements (see page 1607)	Indicates the number of statements currently active with the database server.
	AutoClone (see page 1608)	Specifies whether the SQL connection automatically clones database connections when they are needed.
R	ConnectionData (see page 1609)	This is ConnectionData, a member of class TSQLConnection.
	ConnectionState (see page 1611)	Indicates the current state of the TSQLConnection (see page 1580) object.
R	DataSets (see page 1611)	Lists all active datasets for the connection component.
R	DBXConnection (see page 1611)	Provides an interface to the dbExpress object that represents a connection.
R	InTransaction (see page 1613)	Indicates whether a transaction is in progress.
	LocaleCode (see page 1615)	Identifies the locale that is used for sorting items from datasets that use this connection.
R	MaxStmtsPerConn (see page 1620)	Indicates the server limit on the number of statements supported by a single database connection.
R	MetaData (see page 1620)	Provides an interface to the dbExpress object that supplies server metadata.
R	MultipleTransactionsSupported (see page 1621)	Indicates whether the database server supports multiple transactions.
	ParamsLoaded (see page 1622)	Indicates when parameter values have been loaded from dbxconnections.ini.
	SQLHourGlass (see page 1623)	Specifies whether the cursor should change to crSQLWait during lengthy operations.
R	TransactionsSupported (see page 1624)	Indicates whether the database server supports transactions.

Published Properties

	Name	Description
	Connected (see page 1609)	Specifies whether the connection is active.
	ConnectionName (see page 1610)	Names the connection configuration.
	DriverName (see page 1612)	Specifies the database driver associated with the SQL connection.
	GetDriverFunc (see page 1613)	Specifies the function that returns the top-level dbExpress driver.
	KeepConnection (see page 1614)	Specifies whether the connection stays active when no datasets are open.
	LibraryName (see page 1614)	Specifies the dbExpress driver that TSQLConnection (see page 1580) uses to establish a connection.
	LoadParamsOnConnect (see page 1615)	Specifies whether the TSQLConnection (see page 1580) component loads the configuration named by ConnectionName (see page 1610) immediately before connecting to the server.
	LoginPrompt (see page 1616)	Specifies whether a login dialog appears immediately before opening a new connection.
	Params (see page 1621)	Lists the connection parameters.
	TableScope (see page 1623)	Indicates what types of tables are returned when fetching schema information about tables.
	VendorLib (see page 1625)	Specifies the client software library (DLL or shared object) supplied by the database vendor.

25.1.6.5.1 TSQLConnection.ActiveStatements Property

Indicates the number of statements currently active with the database server.

Pascal

```
property ActiveStatements: LongWord;
```

C++

```
__property LongWord ActiveStatements;
```

Description

Read (see page 3634) ActiveStatements to determine how many statements are currently executing using the database connection. While a dataset is prepared, it creates an active statement that lasts until the dataset is unprepared. When you set a dataset's Prepared property to true, the statement remains active until you change the Prepared property to false. When the Prepared property is false, the dataset implicitly prepares its statement when you open the dataset, and the statement remains active until you close the dataset. In addition, executing an SQL statement (for example, using the Execute (see page 1590) method) creates an active statement temporarily, until the statement finishes executing.

Use ActiveStatements to check whether an application has reached the server-imposed limit on the number of active statements allowed to a connection (the MaxStmtsPerConn (see page 1620) property). This is especially important if AutoClone (see page 1608) is false, because if you exceed MaxStmtsPerConn (see page 1620) when AutoClone (see page 1608) is false, the SQL connection raises an exception. To avoid exceeding the server-imposed limit, do not open any datasets or execute SQL statements when ActiveStatements is the same as MaxStmtsPerConn (see page 1620) (unless MaxStmtsPerConn (see page 1620) is 0).

See Also

DataSets (see page 1611)

Execute (see page 1590)

MaxStmtsPerConn (see page 1620)

Prepared (see page 1563)

AutoClone (see page 1608)

25.1.6.5.2 TSQLConnection.AutoClone Property

Specifies whether the SQL connection automatically clones database connections when they are needed.

Pascal

```
property AutoClone: Boolean;
```

C++

```
__property Boolean AutoClone;
```

Description

AutoClone controls whether the TSQLConnection (see page 1580) component automatically creates duplicate database connections when the number of statements allowed by the database server is exceeded. By default, AutoClone is true, meaning that when the ActiveStatements (see page 1607) property is the same as MaxStmtsPerConn (see page 1620), trying to execute another statement (open another dataset or call the Execute (see page 1590) method) causes TSQLConnection (see page 1580) to open a secondary (cloned) connection. The result behaves just like a connection to a server that does not limit the number of statements, except that it consumes more resources (for the additional, cloned connections).

Change AutoClone to false to prevent TSQLConnection (see page 1580) from cloning the database connection when the application attempts to exceed the maximum number of statements. When AutoClone is false, attempts to exceed MaxStmtsPerConn (see page 1620) result in an exception.

Tip: When AutoClone is false, you can check the ActiveStatements (see page 1607) property to ensure that you do not

exceed the server-imposed limit on the number of statements. If you are about to exceed that limit, you can explicitly call CloneConnection (see page 1587) to clone the connection yourself.

See Also

MaxStmtsPerConn (see page 1620)

ActiveStatements (see page 1607)

CloneConnection (see page 1587)

Connecting to the Database Server

25.1.6.5.3 TSQLConnection.Connected Property

Specifies whether the connection is active.

Pascal

```
property Connected;
```

C++

```
__property Connected;
```

Description

Set Connected to trueto establish a connection to the database server. When a dataset that uses the TSQLConnection (see page 1580) object is activated, it automatically sets Connected to true. Set Connected to false to close a connection.

Setting Connected to true generates a BeforeConnect (see page 1626) event, establishes the connection, and generates an AfterConnect (see page 1626) event. In addition, when setting Connected to true, TSQLConnection (see page 1580) may display a login dialog, depending on the value of LoginPrompt (see page 1616).

Setting Connected to false generates a BeforeDisconnect (see page 1627) event, drops the connection, and generates an AfterDisconnect (see page 1626) event.

See Also

Open

Close

ConnectionState (see page 1611)

BeforeConnect

AfterConnect

BeforeDisconnect

AfterDisconnect

LoginPrompt

25.1.6.5.4 TSQLConnection.ConnectionData Property

Pascal

```
property ConnectionData: TConnectionData;
```

C++

```
__property TConnectionData ConnectionData;
```

Description

This is ConnectionData, a member of class TSQLConnection.

25.1.6.5.5 TSQLConnection.ConnectionName Property

Names the connection configuration.

Pascal

```
property ConnectionName: string;
```

C++

```
__property UnicodeString ConnectionName;
```

Description

Set ConnectionName to use a named connection configuration. Setting ConnectionName automatically sets the DriverName (see page 1612) and Params (see page 1621) properties to reflect the driver and connection parameters stored under that name in the dbxconnections.ini file.

Tip: You can define new connection configurations using the Connection Editor. To launch the Connection Editor, double-click on the TSQLConnection (see page 1580) component.

If you are using a different dbxconnections.ini file at design time than the one used at runtime, you can force the configuration to reload at runtime by setting the LoadParamsOnConnect (see page 1615) property or calling LoadParamsFromIniFile (see page 1600). In most cases, however, this is unnecessary, because the DriverName (see page 1612) and Params (see page 1621) property settings that are loaded when you set ConnectionName are saved in the form file. In fact, unless you set LoadParamsOnConnect (see page 1615) or call LoadParamsFromIniFile (see page 1600), you do not need to deploy the dbxconnections.ini file with your application even if you are using named connections.

See Also

LoadParamsOnConnect (see page 1615)

LoadParamsFromIniFile (see page 1600)

DriverName (see page 1612)

Params (see page 1621)

Setting Up a Simple Dataset

Setting Up TSQLConnection

Configuring TSQL Connection

25.1.6.5.6 TSQLConnection.ConnectionRegistryFile Property

Pascal

```
property ConnectionRegistryFile: string;
```

C++

```
__property UnicodeString ConnectionRegistryFile;
```

Description

This is ConnectionRegistryFile, a member of class TSQLConnection.

25.1.6.5.7 TSQLConnection.ConnectionState Property

Indicates the current state of the TSQLConnection (see page 1580) object.

Pascal

```
property ConnectionState: TConnectionState;
```

C++

```
__property TConnectionState ConnectionState;
```

Description

Read (see page 3634) ConnectionState at runtime to determine the activity being performed, if any, by the connection at a specific point in time. Use ConnectionState to coordinate the activities of the connection component, so that you do not start one activity while it is currently engaged in another.

See Also

Open

Connected (see page 1609)

Close

Execute (see page 1590)

25.1.6.5.8 TSQLConnection.DataSets Property

Lists all active datasets for the connection component.

Pascal

```
property DataSets [Index: Integer]: TCustomSQLDataSet;
```

C++

```
__property TCustomSQLDataSet DataSets[int Index];
```

Description

Use DataSets to access active datasets associated with a connection component. An active dataset is one that is currently open.

The Index parameter ranges between 0 and DataSetCount.

The value of DataSets changes as the datasets that use this TSQLConnection (see page 1580) component open and close.

Note: If a TSQLClientDataSet uses the connection component to connect to a database server, DataSets lists the internal dataset that the client dataset creates to access data, not the client dataset itself.

See Also

DataSetCount

Active

MaxStmtsPerConn (see page 1620)

25.1.6.5.9 TSQLConnection.DBXConnection Property

Provides an interface to the dbExpress object that represents a connection.

Pascal

```
property DBXConnection: TDBXConnection;
```

C++

```
__property TDBXConnection DBXConnection;
```

Description

DBXConnection is an interface to the underlying dbExpress object for a database connection. Typically, you would not need to interact directly with this dbExpress object because all of its methods are surfaced, directly or indirectly, in the implementation of TSQLConnection (see page 1580).

See Also

Debugging dbExpress Applications using TSQLMonitor

Using TSQLQuery

Using TSQLStoredProc

Using TSQLTable

25.1.6.5.10 TSQLConnection.DriverName Property

Specifies the database driver associated with the SQL connection.

Pascal

```
property DriverName: string;
```

C++

```
__property UnicodeString DriverName;
```

Description

If you are not using named connections, set DriverName to the name of an installed dbExpress driver such as INTERBASE, MYSQL, INFORMIX, ORACLE, or DB2. Installed drivers are listed in the dbxdrivers.ini file, and their names can be retrieved using the GetDriverNames (see page 1697) function.

If you are using named connections, the DriverName property is automatically set when you set the ConnectionName (see page 1610) property.

When you set DriverName, TSQLConnection (see page 1580) automatically sets the LibraryName (see page 1614) and VendorLib (see page 1625) properties to the names of the libraries (DLLs or shared objects) that implement the specified driver.

See Also

LibraryName (see page 1614)

VendorLib (see page 1625)

ConnectionName (see page 1610)

GetDriverNames (see page 1697)

Setting Up TSQLConnection

Configuring TSQL Connection

Connecting to Databases with TDatabase

25.1.6.5.11 TSQLConnection.GetDriverFunc Property

Specifies the function that returns the top-level dbExpress driver.

Pascal

```
property GetDriverFunc: string;
```

C++

```
__property UnicodeString GetDriverFunc;
```

Description

GetDriverFunc is the name of a function in the dbExpress driver specified by the LibraryName (see page 1614) property. The specified function returns the top-level dbExpress interface. Typically, this property is set to the function name that is stored in the dbxdrivers.ini file.

Note: The GetDriverFunc, LibraryName (see page 1614), and VenderLib properties all reflect the values in the dbxdrivers.ini file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

LibraryName (see page 1614)

VendorLib (see page 1625)

25.1.6.5.12 TSQLConnection.InTransaction Property

Indicates whether a transaction is in progress.

Pascal

```
property InTransaction: Boolean;
```

C++

```
__property Boolean InTransaction;
```

Description

Read (see page 3634) InTransaction at runtime to ascertain whether a transaction is currently active for the SQL connection component. If the connection component is currently in a transaction, all communication with the server that passes through the connection is enlisted in that transaction.

InTransaction is true if there is an active transaction. If no transaction is active, InTransaction is false.

A transaction begins when the application calls BeginTransaction (see page 1586) and lasts until the application calls CommitFreeAndNil (see page 1588), RollbackFreeAndNil (see page 1603), or RollbackIncompleteFreeAndNil (see page 1604). If there are multiple transactions, InTransaction remains true until all transactions are either committed or rolled back.

See Also

BeginTransaction (see page 1586)

CommitFreeAndNil (see page 1588)

RollbackFreeAndNil (see page 1603)

RollbackIncompleteFreeAndNil (see page 1604)

TransactionsSupported (see page 1624)

TransactionLevel (see page 1567)

MultipleTransactionsSupported (see page 1621)

25.1.6.5.13 TSQLConnection.KeepConnection Property

Specifies whether the connection stays active when no datasets are open.

Pascal

```
property KeepConnection: Boolean;
```

C++

```
__property Boolean KeepConnection;
```

Description

Use KeepConnection to specify whether TSQLConnection (see page 1580) maintains an open database connection even if no associated dataset components are currently active. When KeepConnection is true (the default) the connection is maintained. For connections to remote database servers, or in applications that frequently open and close datasets, set KeepConnection to true to reduce network traffic, speed up applications, and avoid logging in to the server each time the connection is reestablished.

When KeepConnection is false a connection is dropped when there are no open datasets. Dropping a connection releases system resources allocated to the connection, but if a dataset is later opened that uses the database, the connection must be reestablished and initialized.

See Also

DataSets (see page 1611)

Connected (see page 1609)

25.1.6.5.14 TSQLConnection.LibraryName Property

Specifies the dbExpress driver that TSQLConnection (see page 1580) uses to establish a connection.

Pascal

```
property LibraryName: string;
```

C++

```
__property UnicodeString LibraryName;
```

Description

LibraryName is the dbExpress library associated with the driver specified by DriverName (see page 1612). Most applications do not need to use LibraryName directly, because this property is set automatically when the DriverName (see page 1612) property is set.

Note: The association between a dbExpress library and a driver name is stored in the dbxdrivers.ini

file when you install the dbExpress driver. The file named by this property is the only dbExpress file that must be deployed with your application.

Note: The LibraryName, GetDriverFunc (see page 1613), and VenderLib properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

DriverName (see page 1612)

VendorLib (see page 1625)

GetDriverFunc (see page 1613)

Setting Up TSQLConnection

Configuring TSQL Connection

25.1.6.5.15 TSQLConnection.LoadParamsOnConnect Property

Specifies whether the TSQLConnection (see page 1580) component loads the configuration named by ConnectionName (see page 1610) immediately before connecting to the server.

Pascal

```
property LoadParamsOnConnect: Boolean;
```

C++

```
__property Boolean LoadParamsOnConnect;
```

Description

Set LoadParamsOnConnect to true if you want the TSQLConnection (see page 1580) object to load the connection configuration (DriverName (see page 1612) and Params (see page 1621) properties) associated with ConnectionName (see page 1610) at runtime. Typically, this is unnecessary because when you set ConnectionName (see page 1610) at design time, the DriverName (see page 1612) and Params (see page 1621) properties are automatically set as well. However, if you use a different dbxconnections.ini file at runtime, or if you temporarily override the DriverName (see page 1612) or any connection parameter values at design time, setting LoadParamsOnConnect to true ensures that at runtime the connection is established using the configuration associated with ConnectionName (see page 1610) in the dbxconnections.ini file.

Note: If you set LoadParamsOnConnect to true, you must be sure to deploy the dbxconnections.ini

file with your application. When you rely on the design-time associations, dbxconnections.ini is not needed at runtime.

Warning: If you set LoadParamsOnConnect to true, you must specify a valid connection name as the value for ConnectionName (see page 1610).

See Also

ConnectionName (see page 1610)

LoadParamsFromIniFile (see page 1600)

DriverName (see page 1612)

Params (see page 1621)

Setting Up TSQLConnection

Configuring TSQL Connection

25.1.6.5.16 TSQLConnection.LocaleCode Property

Identifies the locale that is used for sorting items from datasets that use this connection.

Pascal

```
property LocaleCode: TLocaleCode;
```

C++

```
__property TLocaleCode LocaleCode;
```

Description

LocaleCode represents the locale that client datasets use to sort strings when they are provided in fields from a dataset associated with this connection. By default, this value is 0, which signals that strings should be sorted without regard to a locale. Changing LocaleCode can affect performance, because sorting is slower when it must take locale into account. Therefore, it is recommended that you only change LocaleCode if the data is known to contain extended characters that will affect the sort order.

Note: Locale codes are treated as connection parameters: they are stored in the dbxconnections.ini

file along with the other parameters, and appear in the Params (see page 1621) property list. Changes to this property change the entry in the Params (see page 1621) property list.

See Also

DataSets (see page 1611)

SQLConnection (see page 1566)

LoadParamsFromIniFile (see page 1600)

Params (see page 1621)

LoadParamsOnConnect (see page 1615)

TClientDataSet

25.1.6.5.17 TSQLConnection.LoginPrompt Property

Specifies whether a login dialog appears immediately before opening a new connection.

Pascal

```
property LoginPrompt;
```

C++

```
__property LoginPrompt;
```

Description

Set LoginPrompt to true to provide login support when establishing a connection. LoginPrompt controls two things: the occurrence of the OnLogin event, and the appearance of a default login dialog that prompts users for a name and password when you include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). When the OnLogin event occurs and when the default login dialog appears depend on the type of connection component:

For TDatabase, TIBDatabase, and TSQLConnection, the dialog appears after the BeforeConnect event and before the AfterConnect event, unless you supply an OnLogin event handler. If there is an OnLogin event handler, that event occurs in place of the login dialog, and there is no need to include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi). If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For TADOConnection components, the dialog appears after the OnWillConnect event and before the BeforeConnect event. If there is an OnLogin event handler, that event occurs after the login dialog. If you do not include DBLogDlg.hpp in your unit (C++) or add DBLogDlg to your uses clause (Delphi), the OnLogin event occurs but there is no default login dialog. If correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler, the connection fails. The OnLogin event does not fire unless LoginPrompt is set to true.

For DataSnap SOAP connection components, the dialog appears after the OnGetUsername event and before the BeforeConnect, AfterConnect, and OnLogin events. If the user cancels from the login dialog, no attempt is made to open a connection.

When LoginPrompt is false, the application must supply user name and password values programmatically:

For TDatabase, the user name and password can be supplied as USER_NAME and PASSWORD parameters in the Params property.

For TADOConnection, the user name and password can be supplied as the ConnectionString property.

For TSQLConnection, the user name, password, and database can be supplied as UserName, Password, and Database parameters in the Params property, or provided as connection parameters associated with the connection name.

For DataSnap SOAP connection components, there is no built-in use for the user name and password supplied by the login dialog. (The UserName and Password properties of TWebConnection are unrelated).

Warning: Storing hard-coded user name and password entries as property values or in code for an OnLogin event handler can compromise server security.

C++ Examples:

```
//
//This example demostrates the use of ADO for database conectivity.
//Example assumes that a TDBGrid is placed on the form.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    /* Login details */
    String UserName = "db_user_name";
    String PassWord = "db_pass_word";
    String Server = "my.db.server";

    /* Connection String */
    String ConnString =
        "Provider=SQLOLEDB.1;Persist Security Info=False;";
    ConnString +=
        "User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;";
    ConnString +=
        "Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;";
    ConnString +=
        "Tag with column collation when possible=False";

    /* SQL Query */
    String SQLStr = "SELECT * FROM customer WHERE customer_id = :AnId;";

    /* All ADO variables */
    TADOConnection* ADOConn;
    TADOQuery* ADOQuery;
    TDataSource* DataSrc;
    TParameter* Param;

    /* Create an ADO connection */
    ADOConn = new TADOConnection(this);

    /* Setup the connection string */
    ADOConn->ConnectionString = Format(ConnString,
        ARRAYOFCONST((UserName, PassWord, Server)));

    /* Disable login prompt */
    ADOConn->LoginPrompt = False;

    try
    {
        ADOConn->Connected = true;
    }
    catch (EADOError *e)
    {
```

```
        MessageDlg("Error while connecting", mtError,
                   TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the query */
    ADOQuery = new TADOQuery(this);
    ADOQuery->Connection = ADOConn;
    ADOQuery->SQL->Add(SQLStr);

    /* Update the parameter that was parsed from the SQL query: AnId */
    Param = ADOQuery->Parameters->ParamByName("AnId");
    Param->DataType = ftInteger;
    Param->Value = 1;

    /* Set the query to Prepared - will improve performance */
    ADOQuery->Prepared = true;

    try
    {
        ADOQuery->Active = true;
    }
    catch (EADOError *e)
    {
        MessageDlg("Error while connecting", mtError,
                   TMsgDlgButtons() << mbOK, 0);
        return;
    }

    /* Create the data source */
    DataSrc = new TDataSource(this);
    DataSrc->DataSet = ADOQuery;
    DataSrc->Enabled = true;

    /* Finally initilalize the grid */
    DBGrid1->DataSource = DataSrc;
}
```

Delphi Examples:

```
{
This example demostrates the use of ADO for database conectivity.
Example assumes that a TDBGrid is placed on the form.
}
procedure TForm2.FormCreate(Sender: TObject);
const
  { Connection string }
  ConnString =
  'Provider=SQLOLEDB.1;Persist Security Info=False;' +
  'User ID=%s;Password=%s;Data Source=%s;Use Procedure for Prepare=1;' +
  'Auto Translate=True;Packet Size=4096;Use Encryption for Data=False;'+
  'Tag with column collation when possible=False';

  { SQL Query }
  SQLStr = 'SELECT * FROM customer WHERE customer_id = :AnId;';

  { User access }
  UserName = 'db_user_name';
  PassWord = 'db_pass_word';
  Server = 'my.db.server';

var
  ADOConn  : TADOConnection;
  ADOQuery : TADOQuery;
  DataSrc  : TDataSource;
  Param    : TParameter;
```

```
begin
  { Create an ADO connection }
  ADOConn := TADOConnection.Create(Self);
  { Setup the provider engine }

  { Setup the connection string }
  ADOConn.ConnectionString := Format(ConnString,
    [UserName, PassWord, Server]);

  { Disable login prompt }
  ADOConn.LoginPrompt := False;

  try
    ADOConn.Connected := True;
  except
    on e: EADOError do
    begin
      MessageDlg('Error while connecting', mtError,
                 [mbOK], 0);

      Exit;
    end;
  end;

  { Create the query }
  ADOQuery := TADOQuery.Create(Self);
  ADOQuery.Connection := ADOConn;
  ADOQuery.SQL.Add(SQLStr);

  { Update the parameter that was parsed from the SQL query: AnId }
  Param := ADOQuery.Parameters.ParamByName('AnId');
  Param.DataType := ftInteger;
  Param.Value := 1;

  { Set the query to Prepared - will improve performance }
  ADOQuery.Prepared := true;

  try
    ADOQuery.Active := True;
  except
    on e: EADOError do
    begin
      MessageDlg('Error while doing query', mtError,
                 [mbOK], 0);

      Exit;
    end;
  end;

  { Create the data source }
  DataSrc := TDataSource.Create(Self);
  DataSrc.DataSet := ADOQuery;
  DataSrc.Enabled := true;

  { Finally initilalize the grid }
  DBGrid1.DataSource := DataSrc;
end;
```

See Also

OnLogin

BeforeConnect

AfterConnect

OnGetUsername

OnLogin (see page 1627)

OnLogin

Controlling Server Login

25.1.6.5.18 TSQLConnection.MaxStmtsPerConn Property

Indicates the server limit on the number of statements supported by a single database connection.

Pascal

```
property MaxStmtsPerConn: LongWord;
```

C++

```
__property LongWord MaxStmtsPerConn;
```

Description

Read (see page 3634) MaxStmtsPerConn when a connection is active to determine whether the database server imposes a limit on the number of statements that can be executed using the current connection. When MaxStmtsPerConn is 0, the server imposes no restrictions: a single database connection can be used to fetch multiple datasets and execute statements while there are active datasets. When MaxStmtsPerConn is greater than 0, the database server limits the number of statements that can execute. The server allows only MaxStmtsPerConn datasets to be opened using a single connection, and when that number of datasets are open, no more statements can be executed (for example, using the Execute (see page 1590) method).

By default, when MaxStmtsPerConn is greater than 0, TSQLConnection (see page 1580) clones the database connection when this server-imposed limit is reached. For example, if MaxStmtsPerConn is 1, then trying to open a second dataset using the same TSQLConnection (see page 1580) component causes TSQLConnection (see page 1580) to establish a second database connection for the second dataset. If AutoClone (see page 1608) is false, however, TSQLConnection (see page 1580) does not clone connections, but rather generates an exception when the application tries to exceed MaxStmtsPerConn.

Note: The value of MaxStmtsPerConn is only valid when the Connected (see page 1609) property is true.

See Also

DataSets (see page 1611)

Execute (see page 1590)

Connected (see page 1609)

AutoClone (see page 1608)

ActiveStatements (see page 1607)

Connecting to the Database Server

25.1.6.5.19 TSQLConnection.MetaData Property

Provides an interface to the dbExpress object that supplies server metadata.

Pascal

```
property MetaData: TDBXDatabaseMetaData;
```

C++

```
__property TDBXDatabaseMetaData MetaData;
```

Description

Typically, applications do not use the MetaData property to work directly with the dbExpress MetaData interface. Instead, they

use one of methods that assembles schema information into a list (GetFieldNames (see page 1592), GetIndexNames (see page 1593), GetProcedureNames (see page 1595), GetProcedureParams (see page 1597), or GetTableNames (see page 1599)). When more detailed metadata is required, you can also use the SQL dataset's SetSchemaInfo method to populate an SQL dataset with metadata.

See Also

GetFieldNames (see page 1592)

GetIndexNames (see page 1593)

GetProcedureNames (see page 1595)

GetProcedureParams (see page 1597)

GetTableNames (see page 1599)

SetSchemaInfo (see page 1550)

25.1.6.5.20 TSQLConnection.MultipleTransactionsSupported Property

Indicates whether the database server supports multiple transactions.

Pascal

```
property MultipleTransactionsSupported: LongBool;
```

C++

```
__property LongBool MultipleTransactionsSupported;
```

Description

Read (see page 3634) MultipleTransactionsSupported at runtime to ascertain whether the database server supports multiple transactions (either nested or overlapping).

Note: To determine whether the server supports transactions at all, use the TransactionsSupported (see page 1624) property. If a server supports only a single transaction at a time, TransactionsSupported (see page 1624) is true, and MultipleTransactionsSupported is false.

See Also

CommitFreeAndNil (see page 1588)

BeginTransaction (see page 1586)

RollbackFreeAndNil (see page 1603)

InTransaction (see page 1613)

TransactionsSupported (see page 1624)

25.1.6.5.21 TSQLConnection.Params Property

Lists the connection parameters.

Pascal

```
property Params: TStrings;
```

C++

```
__property TStrings Params;
```

Description

Params is a TStrings object listing the connection parameters. Each entry in the string list has the form

```
Name=Value
```

where Name is the name of the parameter and Value is its value. If you are using named connection configurations (the ConnectionName (see page 1610) property), the values for each parameter are loaded from the dbxconnections.ini file when you set ConnectName.

At design time, you can set parameter values using the string list editor by double clicking on the Params property in the Object Inspector. Any new values you set override the values loaded when you set ConnectionName (see page 1610) unless you load the connection configuration at runtime using LoadParamsFromIniFile (see page 1600) or LoadParamsOnConnect (see page 1615).

The particular parameters that appear depend on the database server. The Params property is initialized to include a string for each parameter when you set the DriverName (see page 1612) property. One particular parameter, Database, is required for all servers. Its value depends on the server you are using. For example, with InterBase, Database is the name of the .gdb file, while with MySQL, it is the database name that was assigned by the CREATE DATABASE command.

Other typical parameters include HostName (the machine name or IP address of where the server is located), and TransIsolation (the degree to which transactions are isolated from changes made by other users of the database).

See Also

ConnectionName (see page 1610)

LoadParamsFromIniFile (see page 1600)

LoadParamsOnConnect (see page 1615)

DriverName (see page 1612)

LocaleCode (see page 1615)

Setting Up TSQLConnection

25.1.6.5.22 TSQLConnection.ParamsLoaded Property

Indicates when parameter values have been loaded from dbxconnections.ini.

Pascal

```
property ParamsLoaded: Boolean;
```

C++

```
__property Boolean ParamsLoaded;
```

Description

Do not use the ParamsLoaded property. It is used internally by the Connections Editor to keep track of when parameter values have been loaded from the dbxconnections.ini file.

See Also

Params (see page 1621)

LoadParamsFromIniFile (see page 1600)

LoadParamsOnConnect (see page 1615)

25.1.6.5.23 TSQLConnection.QuoteChar Property

Pascal

```
property QuoteChar: UnicodeString;
```

C++

```
__property UnicodeString QuoteChar;
```

Description

This is QuoteChar, a member of class TSQLConnection.

25.1.6.5.24 TSQLConnection.SQLDllHandle Property

Pascal

```
property SQLDllHandle: THandle;
```

C++

```
__property THandle SQLDllHandle;
```

Description

This is SQLDllHandle, a member of class TSQLConnection.

25.1.6.5.25 TSQLConnection.SQLHourGlass Property

Specifies whether the cursor should change to crSQLWait during lengthy operations.

Pascal

```
property SQLHourGlass: Boolean;
```

C++

```
__property Boolean SQLHourGlass;
```

Description

Set SQLHourGlass to true to specify that the cursor should indicate when the connection is performing a potentially lengthy operation. Such operations include

Establishing a connection to the database server.

Fetching metadata from the database server.

Executing a statement using the Execute (see page 1590) method.

When SQLHourGlass is false, the cursor is not changed when the SQL connection performs such operations.

See Also

Execute (see page 1590)

Open

TCursorType

25.1.6.5.26 TSQLConnection.TableScope Property

Indicates what types of tables are returned when fetching schema information about tables.

Pascal

```
property TableScope: TTableScopes;
```

C++

```
__property TTableScopes TableScope;
```

Description

Set TableScope to determine the type of tables listed by the GetTableNames (see page 1599) method or that populate an SQL dataset after calling its SetSchemaInfo method with a parameter of stTables. TableScope lets you request information about a combination of table types. By default, TableScope requests only ordinary tables and views (tsTable and tsView).

Note: TableScope only affects GetTableNames (see page 1599) when the SystemTables parameter is false and only affects SetSchemaInfo when the SchemaType parameter is stTables. If an application calls GetTablesNames with SystemTables set to true, or calls SetSchemaInfo with SchemaType set to stSysTables, system tables, and only system tables are returned, regardless of the value of TableScope.

See Also

GetTableNames (see page 1599)

SetSchemaInfo (see page 1550)

Fetching Metadata into a Unidirectional Dataset

25.1.6.5.27 TSQLConnection.TransactionsSupported Property

Indicates whether the database server supports transactions.

Pascal

```
property TransactionsSupported: LongBool;
```

C++

```
__property LongBool TransactionsSupported;
```

Description

Read (see page 3634) TransactionsSupported at runtime to ascertain whether the database server supports transactions. If TransactionsSupported is false, then the BeginTransaction (see page 1586), CommitFreeAndNil (see page 1588), RollbackFreeAndNil (see page 1603), and RollbackIncompleteFreeAndNil (see page 1604) methods do nothing. If TransactionsSupported is true, you can use the SQL connection component to manage transactions.

Note: Some servers support multiple transactions (either nested or overlapping). When TransactionsSupported is true, you can test whether the server supports multiple transactions by reading the MultipleTransactionsSupported (see page 1621) property.

See Also

BeginTransaction (see page 1586)

CommitFreeAndNil (see page 1588)

RollbackFreeAndNil (see page 1603)

RollbackIncompleteFreeAndNil (see page 1604)

InTransaction (see page 1613)

MultipleTransactionsSupported (see page 1621)

25.1.6.5.28 TSQLConnection.VendorLib Property

Specifies the client software library (DLL or shared object) supplied by the database vendor.

Pascal

```
property VendorLib: string;
```

C++

```
__property UnicodeString VendorLib;
```

Description

VendorLib is the library supplied by the database vendor to support client-side use of the database. Most applications do not need to use VendorLib directly, because this property is set automatically when the DriverName (see page 1612) property is set.

Note: The association between a vendor library and a driver name is stored in the dbxdrivers.ini

file when you install the associated dbExpress driver. The file named by this property must be running on the same system as your application when it is deployed. Typically, it is installed when the database is installed rather than with your application.

Note: The VendorLib, LibraryName (see page 1614), and GetDriverFunc (see page 1613) properties all reflect the values in the dbxdrivers.ini

file. Because these property values are published (stored with the connection component), it is possible to deploy an application without deploying the dbxdrivers.ini file.

See Also

DriverName (see page 1612)

GetDriverFunc (see page 1613)

Setting Up TSQLConnection

Configuring TSQL Connection

25.1.6.6 TSQLConnection Events

The events of the TSQLConnection class are listed here.

Public Events

	Name	Description
	TraceCallbackEvent (see page 1628)	Provides access to the callback function that executes for every SQL command passed to or from the server.

Published Events

	Name	Description
	AfterConnect (see page 1626)	Occurs after a connection is established.
	AfterDisconnect (see page 1626)	Occurs after the connection closes.
	BeforeConnect (see page 1626)	Occurs immediately before establishing a connection.
	BeforeDisconnect (see page 1627)	Occurs immediately before the connection closes.
	OnLogin (see page 1627)	Occurs after the BeforeConnect (see page 1626) event and before a connection is established.

25.1.6.6.1 TSQLConnection.AfterConnect Event

Occurs after a connection is established.

Pascal

```
property AfterConnect;
```

C++

```
__property AfterConnect;
```

Description

Write an AfterConnect event handler to take application-specific actions immediately after the connection component opens a connection to the remote source of database information.

See Also

BeforeConnect

Connected

Connecting to a Database Server

25.1.6.6.2 TSQLConnection.AfterDisconnect Event

Occurs after the connection closes.

Pascal

```
property AfterDisconnect;
```

C++

```
__property AfterDisconnect;
```

Description

Write an AfterDisconnect event handler to take application-specific actions after the connection component drops a connection.

See Also

BeforeDisconnect

Connected

Disconnecting from a Database Server

25.1.6.6.3 TSQLConnection.BeforeConnect Event

Occurs immediately before establishing a connection.

Pascal

```
property BeforeConnect;
```

C++

```
__property BeforeConnect;
```

Description

Write a BeforeConnect event handler to take application-specific actions before the connection component opens a connection to the remote source of database information.

See Also

AfterConnect

Connected

Connecting to a Database Server

25.1.6.6.4 TSQLConnection.BeforeDisconnect Event

Occurs immediately before the connection closes.

Pascal

```
property BeforeDisconnect;
```

C++

```
__property BeforeDisconnect;
```

Description

Write a BeforeDisconnect event handler to take application-specific actions before dropping a connection.

Delphi Examples:

```
{
This BeforeDisconnect event handler makes sure that the
client dataset has applied all its pending updates before
terminating the connection to the server.
}
procedure TForm1.RemoteServer1BeforeDisconnect(Sender: TObject);
begin
  if (ClientDataSet1.ChangeCount > 0) then
    ClientDataSet1.ApplyUpdates(-1);
end;
```

See Also

AfterDisconnect

Connected

Disconnecting from a Database Server

25.1.6.6.5 TSQLConnection.OnLogin Event

Occurs after the BeforeConnect (see page 1626) event and before a connection is established.

Pascal

```
property OnLogin: TSQLConnectionLoginEvent;
```

C++

```
__property TSQLConnectionLoginEvent OnLogin;
```

Description

Use the OnLogin event to assign values to the User_Name, Password, and Database parameters immediately before TSQLConnection (see page 1580) attempts to connect to the database server. OnLogin only occurs if the LoginPrompt (see page 1616) property is true. If LoginPrompt (see page 1616) is true but there is no OnLogin event handler, a default login dialog appears in which the user can enter a user name and password. The connection fails if correct values for the user name and password are not supplied in the dialog or by the OnLogin event handler.

See Also

Params (see page 1621)

LoginPrompt

BeforeConnect

25.1.6.6.6 TSQLConnection.TraceCallbackEvent Event

Provides access to the callback function that executes for every SQL command passed to or from the server.

Pascal

```
property TraceCallbackEvent: TDBXTraceEvent;
```

C++

```
__property TDBXTraceEvent TraceCallbackEvent;
```

Description

Read (see page 3634) TraceCallbackEvent to determine whether the SQL connection is monitoring communication between the application and the SQL server to which it connects. For example, if the application uses a TSQLMonitor (see page 1643) component, that component assigns a value to TraceCallbackEvent, which it uses to monitor SQL commands. Changing the value of the callback would then interfere with the operation of the TSQLMonitor (see page 1643) component.

TraceCallbackEvent is a read-only property. To assign a callback for monitoring SQL commands, use the SetTraceEvent (see page 1605) method.

See Also

SetTraceEvent (see page 1605)

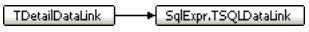
Procedural types

Creating events

25.1.7 TSQLDataLink Class

TSQLDataLink allows a unidirectional dataset act as the detail in a master/detail relationship.

Class Hierarchy



Pascal

```
TSQLDataLink = class(TDetailDataLink);
```

C++

```
class TSQLDataLink : public TDetailDataLink;
```

File

SqlExpr

Description

SQL datasets use TSQLDataLink to coordinate with another (master) dataset when they represent the detail records for that other dataset. TSQLDataLink passes on data events to the data source that owns it. It is also used to implement the DataSource

or MasterSource property of the dataset that owns it.

Do not confuse the dataset that is the value of the TSQLDataLink object's DataSet property with the dataset that uses TSQLDataLink to link to a master dataset. The DataSet property is the master dataset, while the dataset that owns the TSQLDataLink object is the detail set.

See Also

TSQLDataSet (see page 1631)

TSQLQuery (see page 1654)

TSQLTable (see page 1680)

TMasterDataLink

TCustomSQLDataSet (see page 1520)

25.1.7.1 TSQLDataLink Members

The following tables list the members exposed by TSQLDataLink.

Protected Methods

	Name	Description
	ActiveChanged (see page 1630)	Refreshes the parameter values of the detail dataset and re-executes it.
	CheckBrowseMode (see page 1630)	Informs the dataset that owns the TSQLDataLink (see page 1628) object when the current record in the source dataset is about to change.
	GetDetailDataSet (see page 1631)	Returns the value of the DetailDataSet property.
	RecordChanged (see page 1631)	Refreshes the parameter values of the detail dataset and re-executes it.

Public Methods

	Name	Description
	Create (see page 1629)	Creates an instance of TSQLDataLink (see page 1628).

25.1.7.2 TSQLDataLink.Create Constructor

Creates an instance of TSQLDataLink (see page 1628).

Pascal

```
constructor Create(ADataSet: TCustomSQLDataSet);
```

C++

```
__fastcall TSQLDataLink(TCustomSQLDataSet ADataSet);
```

Description

Do not create instances of TSQLDataLink (see page 1628). It is instantiated by the SQL dataset that uses it to link to a master dataset. The ADataSet parameter is the dataset that creates and owns this SQL datalink. It becomes the value of the DetailDataSet property.

See Also

DetailDataSet

25.1.7.3 TSQLDataLink Methods

The methods of the TSQLDataLink class are listed here.

Protected Methods

	Name	Description
	ActiveChanged (see page 1630)	Refreshes the parameter values of the detail dataset and re-executes it.
	CheckBrowseMode (see page 1630)	Informs the dataset that owns the TSQLDataLink (see page 1628) object when the current record in the source dataset is about to change.
	GetDetailDataSet (see page 1631)	Returns the value of the DetailDataSet property.
	RecordChanged (see page 1631)	Refreshes the parameter values of the detail dataset and re-executes it.

25.1.7.3.1 TSQLDataLink.ActiveChanged Method

Refreshes the parameter values of the detail dataset and re-executes it.

Pascal

```
procedure ActiveChanged; override;
```

C++

```
virtual __fastcall ActiveChanged();
```

Description

SQL datasets establish master/detail relationships by supplying parameter values from the master dataset. ActiveChanged updates these parameter values on the detail dataset to reflect the current values of the master dataset and then refreshes the dataset so that it is populated with the new detail set.

See Also

Active

Params (see page 1562)

DetailDataSet

25.1.7.3.2 TSQLDataLink.CheckBrowseMode Method

Informs the dataset that owns the TSQLDataLink (see page 1628) object when the current record in the source dataset is about to change.

Pascal

```
procedure CheckBrowseMode; override;
```

C++

```
virtual __fastcall CheckBrowseMode();
```

Description

Applications cannot call the protected CheckBrowseMode procedure. CheckBrowseMode is called immediately before the current record in the source dataset changes. It calls the CheckBrowseMode method of the dataset that owns the TSQLDataLink (see page 1628) object.

See Also

CheckBrowseMode

25.1.7.3.3 TSQLDataLink.GetDetailDataSet Method

Returns the value of the DetailDataSet property.

Pascal

```
function GetDetailDataSet: TDataSet; override;
```

C++

```
virtual __fastcall TDataSet GetDetailDataSet();
```

Description

GetDetailDataSet is the protected read implementation of the DetailDataSet property.

See Also

DetailDataSet

25.1.7.3.4 TSQLDataLink.RecordChanged Method

Refreshes the parameter values of the detail dataset and re-executes it.

Pascal

```
procedure RecordChanged(Field: TField); override;
```

C++

```
virtual __fastcall RecordChanged(TField Field);
```

Description

Applications cannot call the protected RecordChanged method. It is called automatically when the contents of the current record change. The Field parameter indicates which field in the current record has changed. If Field is nil (Delphi) or NULL (C++), the entire record has changed.

SQL datasets establish master/detail relationships by supplying parameter values from the master dataset. RecordChanged updates these parameter values on the detail dataset to reflect the current values of the master dataset and then refreshes the dataset so that it is populated with the new detail set.

See Also

Params (see page 1562)

DetailDataSet

25.1.8 TSQLDataSet Class

TSQLDataSet represents data retrieved using dbExpress.

Class Hierarchy

Pascal

```
TSQLDataSet = class(TCustomSQLDataSet);
```

C++

```
class TSQLDataSet : public TCustomSQLDataSet;
```

File

SqlExpr

Description

TSQLDataSet is a general-purpose unidirectional dataset for accessing database information using dbExpress. You can add a TSQLDataSet component to a form or data module at design time, or create one dynamically at runtime.

Use TSQLDataSet to

Represent the records in a database table, the result set of a SELECT query, or the result set returned by a stored procedure.

Execute a query or stored procedure that does not return a result set.

Represent metadata that describes what is available on the database server (tables, stored procedures, fields in a table, and so on).

TSQLDataSet is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL dataset by explicitly creating an SQL UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLDataSet component, it must be connected to the database server. Therefore, the first step to take when working with TSQLDataSet is to set the SQLConnection (see page 1642) property.

Once the dataset is connected to a server, you can use the Command Text editor to set the CommandText (see page 1637) property if you are using the dataset to execute a query. To execute a stored procedure or represent a single database table, set the CommandType (see page 1638) property and then choose the table or stored procedure name from a drop-down list on the CommandText (see page 1637) property. If you want to access metadata with the SQL dataset, you must specify the metadata you want at runtime by calling the SetSchemaInfo (see page 1550) method.

DefaultRowsetSize (see page 1713) is 20 (DefaultRowsetSize (see page 1713) = 20). To use a different RowSetSize for existing Oracle database connections, manually add the RowSetSize property (for example, RowsetSize = 200) to dbxconnections.ini (Windows) or dbxconnections.conf (Linux). To have the RowSetSize property included in new connections, manually add the RowSetSize property to dbxdrivers.ini (Windows) or dbxdrivers.conf (Linux). RowsetSize can also be modified in code (for example, SQLConnection1.Params.Values['RowsetSize'] := '200'). Currently, only Oracle databases support this feature.

See Also

TSQLQuery (see page 1654)

TSQLStoredProc (see page 1671)

TSQLTable (see page 1680)

TSQLConnection (see page 1580)

TClientDataSet

TDataSetProvider

Types of Datasets

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Types of Unidirectional Datasets

25.1.8.1 TSQLDataSet Members

The following tables list the members exposed by TSQLDataSet.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1539)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.

	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.
	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.
	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	RowsAffected (see page 1564)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.

Public Methods

	Name	Description
	Create (see page 1635)	Creates an instance of a TSQLDataSet (see page 1631) component.
	Create (see page 1524)	This is the overview for the Create constructor overload.
	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	Destroy (see page 1525)	Destroys the TCustomSQLDataSet (see page 1520) instance.
	ExecSQL (see page 1636)	Executes a query or stored procedure that does not return a set of records.
	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
	PrepareStatement (see page 1540)	Performs a prepare on the SQL statement.

	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Published Properties

	Name	Description
	Active (see page 1552)	Specifies whether or not a dataset is open.
	CommandText (see page 1637)	Specifies the command the dataset executes.
	CommandType (see page 1638)	Command type of CommandText (see page 1637) property.
	DataSource (see page 1639)	Links the SQL dataset to another (master) dataset.
	DbxCommandType (see page 1639)	Command type in String format.
	GetMetadata (see page 1558)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1640)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 1641)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1641)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1565)	Specifies the name of the database schema.
	SortFieldNames (see page 1642)	Indicates the sort order when CommandType (see page 1638) is ctTable.
	SQLConnection (see page 1642)	Specifies the SQL connection component that connects the dataset to a database server.

25.1.8.2 TSQLDataSet.Create Constructor

Creates an instance of a TSQLDataSet (see page 1631) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLDataSet(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL dataset component at runtime. SQL datasets placed in forms or data modules at design time are created automatically.

AOwner is the component, typically the form or data module, that is responsible for freeing the dataset. It becomes the value of the Owner property.

See Also

Owner

25.1.8.3 TSQLDataSet Methods

The methods of the TSQLDataSet class are listed here.

Public Methods

	Name	Description
	ExecSQL (see page 1636)	Executes a query or stored procedure that does not return a set of records.

25.1.8.3.1 TSQLDataSet.ExecSQL Method

Executes a query or stored procedure that does not return a set of records.

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

Call ExecSQL to execute an SQL command that does not return a set of records. This command is a query other than a SELECT query, such as an INSERT, UPDATE, DELETE, or CREATE TABLE query.

ExecDirect indicates that the query does not need to be prepared before it is executed. This parameter can be set to true if the query does not include any parameters. When ExecDirect is false, the query will be prepared before it is executed. Even when ExecDirect is false, you can avoid preparing the query every time ExecSQL is called by setting the Prepared (see page 1563) property to true before calling ExecSQL the first time.

ExecSQL returns the number of rows affected by the executed command. This becomes the value of the RowsAffected (see page 1564) property.

Note: Do not use ExecSQL for commands that return data. These include any use of the ctTable command type, SELECT queries, and stored procedures that return a cursor. When the command returns data, use the Open method or set the Active (see page 1552) property to true.

See Also

CommandText (see page 1554)

RowsAffected (see page 1564)

Open

Active

Prepared (see page 1563)

ExecuteDirect (see page 1590)

Executing the Command

25.1.8.4 TSQLDataSet Properties

The properties of the TSQLDataSet class are listed here.

Published Properties

	Name	Description
	CommandText (see page 1637)	Specifies the command the dataset executes.
	CommandType (see page 1638)	Command type of CommandText (see page 1637) property.
	DataSource (see page 1639)	Links the SQL dataset to another (master) dataset.
	DbxCommandType (see page 1639)	Command type in String format.
	MaxBlobSize (see page 1640)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 1641)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1641)	Represents the parameters for a query or stored procedure.
	SortFieldNames (see page 1642)	Indicates the sort order when CommandType (see page 1638) is ctTable.
	SQLConnection (see page 1642)	Specifies the SQL connection component that connects the dataset to a database server.

25.1.8.4.1 TSQLDataSet.CommandText Property

Specifies the command the dataset executes.

Pascal

```
property CommandText: UnicodeString;
```

C++

```
__property UnicodeString CommandText;
```

Description

The value of CommandText depends on the CommandType (see page 1638) property:

When CommandType (see page 1638) is ctQuery, CommandText is an SQL statement that the dataset executes. If the statement is a SELECT statement, the SQL dataset executes the statement when it is opened. If the statement does not return a result set, the SQL dataset executes the statement when the ExecSQL (see page 1636) method is called.

When CommandType (see page 1638) is ctStoredProc, CommandText is the name of a stored procedure. The SQL dataset executes the stored procedure when it is opened or when the ExecSQL (see page 1636) method is called, depending on whether the stored procedure returns a set of records.

When CommandType (see page 1638) is ctTable, CommandText is the name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table, sorting by the fields listed in the SortFieldNames (see page 1642) property. The generated statement is executed when the dataset is opened.

If CommandText is an SQL statement that includes parameters or the name of a stored procedure that has parameters, use the Params (see page 1641) property to supply parameter values.

This command is passed on to the TDBXCommand.Text property.

See Also

TDBXCommand

CommandType

Params (see page 1641)

ExecSQL (see page 1636)

Open

SortFieldNames (see page 1642)

Fetching Metadata into a Unidirectional Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Accessing Schema Information

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

25.1.8.4.2 TSQLDataSet.CommandType Property

Command type of CommandText (see page 1637) property.

Pascal

```
property CommandType: TSQLCommandType;
```

C++

```
__property TSQLCommandType CommandType;
```

Description

CommandType indicates the command type of the CommandText (see page 1637) property.

The following table lists the possible values:

CommandType	Corresponding CommandText (see page 1637)
ctQuery	An SQL statement that the dataset executes.
ctStoredProc	The name of a stored procedure.
ctTable	The name of a table on the database server. The SQL dataset automatically generates a SELECT statement to fetch all the records on all the fields in this table.

Note: Changing CommandType automatically changes the DbxCommandType (see page 1639) property and vice versa. The additional settings supported by DbxCommandType (see page 1639) are converted to ctQuery for CommandType.

See Also

CommandText (see page 1637)

DbxCommandType (see page 1639)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

25.1.8.4.3 TSQLDataSet.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (see page 1637) (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params (see page 1641) property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1637)

SQL (see page 1663)

Params (see page 1641)

MasterSource (see page 1692)

25.1.8.4.4 TSQLDataSet.DbxCommandType Property

Command type in String format.

Pascal

```
property DbxCommandType: UnicodeString;
```

C++

```
__property UnicodeString DbxCommandType;
```

Description

DbxCommandType indicates the type of command that is contained in the CommandText (see page 1637) property. This can be set to one of the constant strings in the TDBXCommandTypes class. However, DbxCommandType is a `String`, unlike the CommandType (see page 1638) property. This makes it easier to define new command types, making this approach more

extensible. DbxCommandType is a superset of CommandType (see page 1638).

Note: Changing DbxCommandType automatically changes the CommandType (see page 1638) property and vice versa. The additional settings supported by DbxCommandType are converted to ctQuery for CommandType (see page 1638).

See Also

TDBXCommandTypes

CommandText (see page 1554)

CommandType (see page 1638)

Setting Up a Simple Dataset

Representing the Records in a Table

Representing the Results of a Query

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Specifying What Data to Display

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSimpleDataSet

25.1.8.4.5 TSQLDataSet.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1530)

Params (see page 1641)

SQLConnection (see page 1642)

25.1.8.4.6 TSQLDataSet.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params (see page 1641) property is updated automatically to reflect any parameters that appear in the query specified by the CommandText (see page 1637) property (or the SQL property for TSQLQuery (see page 1654)). That is, whenever the CommandText (see page 1637) property changes, the Params (see page 1641) property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params (see page 1641) property in response to changes in CommandText (see page 1637), set ParamCheck to false.

See Also

Params (see page 1641)

CommandText (see page 1637)

SQL (see page 1663)

25.1.8.4.7 TSQLDataSet.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText (see page 1637) property (or the SQL property in TSQLQuery (see page 1654)), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText (see page 1637) property (or the StoredProcName property in TSQLStoredProc (see page 1671)), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource (see page 1639) property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1641)

SQL (see page 1663)

StoredProcName (see page 1680)

CommandText (see page 1637)

TParam

Value

DataType

DataSource (see page 1557)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

25.1.8.4.8 TSQLDataSet.SortFieldNames Property

Indicates the sort order when CommandType (see page 1638) is ctTable.

Pascal

```
property SortFieldNames: UnicodeString;
```

C++

```
__property UnicodeString SortFieldNames;
```

Description

Use SortFieldNames to specify the fields that should be used to order the records in the dataset. SortFieldNames is only used when CommandType (see page 1638) is ctTable.

SortFieldNames is a semicolon-delimited list of field names. These names are used to generate the ORDER BY clause of the SQL statement the dataset creates to fetch data from a server table. Records are sorted by the first named field, and within that sort order, by the second named field, and so on.

See Also

CommandType (see page 1638)

Representing the Records in a Table

Specifying the Data to Display using TSQLDataSet

25.1.8.4.9 TSQLDataSet.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection (see page 1580) components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection (see page 1580) object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText (see page 1637) property or populate the dataset with data by setting Active (see page 1552) to true. Similarly, SQLConnection must be set before the dataset can initialize the Params (see page 1641) property when you set CommandText (see page 1637).

See Also

Active

CommandText (see page 1637)

Params (see page 1641)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

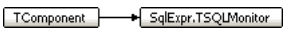
Connecting Directly to a Database Server

Connecting to the Database Server

25.1.9 TSQLMonitor Class

TSQLMonitor intercepts messages that pass between an SQL connection component and a database server and saves them in a string list.

Class Hierarchy



Pascal

```
TSQLMonitor = class(TComponent);
```

C++

```
class TSQLMonitor : public TComponent;
```

File

SqlExpr

Description

Use TSQLMonitor when debugging the communication between your application and a database server. Each TSQLMonitor instance logs the SQL commands from a particular SQL connection component, adding them to a string list. This allows you to see not only the commands you explicitly add to a SQL dataset or SQL connection component, but also any commands that are

generated on your behalf, such as those created by a dataset provider. Use the TraceList (see page 1652) property to access the list of SQL commands that has already been logged. The TraceList (see page 1652) property is automatically updated when the connection component passes a message to the database server.

Note: TSQLMonitor uses the TraceCallbackEvent property of the SQL connection component whose messages it monitors. If you assign your own callback to that property, the SQL monitor instance will cease to function.

See Also

TSQLConnection (see page 1580)

TStringList

TDataSetProvider

Debugging dbExpress Applications

25.1.9.1 TSQLMonitor Members

The following tables list the members exposed by TSQLMonitor.

Protected Methods

	Name	Description
	InvokeCallBack (see page 1646)	This is InvokeCallBack, a member of class TSQLMonitor.
	SetActive (see page 1647)	This is SetActive, a member of class TSQLMonitor.
	SetFileName (see page 1647)	This is SetFileName, a member of class TSQLMonitor.
	SetSQLConnection (see page 1647)	This is SetSQLConnection, a member of class TSQLMonitor.
	SetStreamedActive (see page 1648)	This is SetStreamedActive, a member of class TSQLMonitor.
	SetTraceList (see page 1648)	This is SetTraceList, a member of class TSQLMonitor.
	SwitchConnection (see page 1648)	This is SwitchConnection, a member of class TSQLMonitor.
V	Trace (see page 1648)	This is Trace, a member of class TSQLMonitor.
	UpdateTraceCallBack (see page 1648)	This is UpdateTraceCallBack, a member of class TSQLMonitor.

Public Methods

	Name	Description
V	Create (see page 1645)	Creates and initializes new TSQLMonitor (see page 1643) object.
V	Destroy (see page 1645)	Frees the memory associated with the TSQLMonitor (see page 1643) object.
	LoadFromFile (see page 1646)	Sets the contents of TraceList (see page 1652) to the strings contained in a file.
	SaveToFile (see page 1646)	Saves the contents of TraceList (see page 1652) to a file.

Public Properties

	Name	Description
	MaxTraceCount (see page 1651)	Indicates the maximum number of messages that can be logged.
R	TraceCount (see page 1651)	Indicates the number of messages currently logged.

Published Events

	Name	Description
	OnLogTrace (see page 1653)	Occurs immediately after a new message is logged to the list.
	OnTrace (see page 1653)	Occurs when a message is detected but before it is logged to the list.

Published Properties

	Name	Description
	Active (see page 1649)	Starts or stops the monitor from recording SQL commands passed to the database server.
	AutoSave (see page 1650)	Specifies whether the monitored messages are automatically saved to a file.

	FileName (see page 1650)	Specifies the default file to which monitored messages are saved.
	SQLConnection (see page 1651)	Specifies the connection components whose messages are to be monitored.
	TraceList (see page 1652)	Lists the messages that have passed between SQLConnection (see page 1651) and the database server.

25.1.9.2 TSQLMonitor.Create Constructor

Creates and initializes new TSQLMonitor (see page 1643) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLMonitor(TComponent * AOwner);
```

Description

Call Create to create a TSQLMonitor (see page 1643) component at runtime. SQL monitors added to forms or data modules at design time are created automatically.

The Owner parameter specifies the component (usually the form or data module) that is responsible for freeing the SQL monitor.

See Also

Owner

25.1.9.3 TSQLMonitor.Destroy Destructor

Frees the memory associated with the TSQLMonitor (see page 1643) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLMonitor();
```

Description

Do not call Destroy directly. Instead, call Free, which verifies that the SQL monitor instance is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

25.1.9.4 TSQLMonitor Methods

The methods of the TSQLMonitor class are listed here.

Protected Methods

	Name	Description
	InvokeCallBack (see page 1646)	This is InvokeCallBack, a member of class TSQLMonitor.
	SetActive (see page 1647)	This is SetActive, a member of class TSQLMonitor.
	SetFileName (see page 1647)	This is SetFileName, a member of class TSQLMonitor.
	SetSQLConnection (see page 1647)	This is SetSQLConnection, a member of class TSQLMonitor.
	SetStreamedActive (see page 1648)	This is SetStreamedActive, a member of class TSQLMonitor.
	SetTraceList (see page 1648)	This is SetTraceList, a member of class TSQLMonitor.

	SwitchConnection (see page 1648)	This is SwitchConnection, a member of class TSQLMonitor.
	Trace (see page 1648)	This is Trace, a member of class TSQLMonitor.
	UpdateTraceCallBack (see page 1648)	This is UpdateTraceCallBack, a member of class TSQLMonitor.

Public Methods

	Name	Description
	LoadFromFile (see page 1646)	Sets the contents of TraceList (see page 1652) to the strings contained in a file.
	SaveToFile (see page 1646)	Saves the contents of TraceList (see page 1652) to a file.

25.1.9.4.1 TSQLMonitor.InvokeCallBack Method

Pascal

```
function InvokeCallBack(TraceInfo: TDBXTraceInfo): CBRType;
```

C++

```
__fastcall CBRType InvokeCallBack(TDBXTraceInfo TraceInfo);
```

Description

This is InvokeCallBack, a member of class TSQLMonitor.

25.1.9.4.2 TSQLMonitor.LoadFromFile Method

Sets the contents of TraceList (see page 1652) to the strings contained in a file.

Pascal

```
procedure LoadFromFile(AFileName: string);
```

C++

```
__fastcall LoadFromFile(UnicodeString AFileName);
```

Description

Call LoadFromFile to initialize the message log with the contents of a file. This method can be used to start the log with standard header information you have saved in a file, or to pre-load the list with a log you saved previously using the SaveToFile (see page 1646) method.

AFileName is the file from which to load the contents of the message log. If AFileName is an empty string, LoadFromFile uses the value of the FileName (see page 1650) property.

Note: Any information currently in TraceList (see page 1652) is lost when you call LoadFromFile.

See Also

TraceList (see page 1652)

SaveToFile (see page 1646)

FileName (see page 1650)

25.1.9.4.3 TSQLMonitor.SaveToFile Method

Saves the contents of TraceList (see page 1652) to a file.

Pascal

```
procedure SaveToFile(AFileName: string);
```

C++

```
__fastcall SaveToFile(UnicodeString AFileName);
```

Description

Call SaveToFile to save all of the currently logged messages to a file.

AFileName specifies the file to which the strings in TraceList (see page 1652) are saved. If AFileName is an empty string, SaveToFile uses the value of the FileName (see page 1650) property.

Note: If the AutoSave (see page 1650) property is true, SaveToFile is called automatically whenever a message is added to TraceList (see page 1652).

See Also

TraceList (see page 1652)

LoadFromFile (see page 1646)

FileName (see page 1650)

AutoSave (see page 1650)

25.1.9.4.4 TSQLMonitor.SetActive Method

Pascal

```
procedure SetActive(Value: Boolean);
```

C++

```
__fastcall SetActive(Boolean Value);
```

Description

This is SetActive, a member of class TSQLMonitor.

25.1.9.4.5 TSQLMonitor.SetFileName Method

Pascal

```
procedure SetFileName(const Value: String);
```

C++

```
__fastcall SetFileName(const UnicodeString Value);
```

Description

This is SetFileName, a member of class TSQLMonitor.

25.1.9.4.6 TSQLMonitor.SetSQLConnection Method

Pascal

```
procedure SetSQLConnection(Value: TSQLConnection);
```

C++

```
__fastcall SetSQLConnection(TSQLConnection Value);
```

Description

This is SetSQLConnection, a member of class TSQLMonitor.

25.1.9.4.7 TSQLMonitor.SetStreamedActive Method

Pascal

```
procedure SetStreamedActive;
```

C++

```
__fastcall SetStreamedActive();
```

Description

This is SetStreamedActive, a member of class TSQLMonitor.

25.1.9.4.8 TSQLMonitor.SetTraceList Method

Pascal

```
procedure SetTraceList(Value: TWideStrings);
```

C++

```
__fastcall SetTraceList(TWideStrings Value);
```

Description

This is SetTraceList, a member of class TSQLMonitor.

25.1.9.4.9 TSQLMonitor.SwitchConnection Method

Pascal

```
procedure SwitchConnection(const Value: TSQLConnection);
```

C++

```
__fastcall SwitchConnection(const TSQLConnection Value);
```

Description

This is SwitchConnection, a member of class TSQLMonitor.

25.1.9.4.10 TSQLMonitor.Trace Method

Pascal

```
procedure Trace(TraceInfo: TDBXTraceInfo; LogTrace: Boolean); virtual;
```

C++

```
virtual __fastcall Trace(TDBXTraceInfo TraceInfo, Boolean LogTrace);
```

Description

This is Trace, a member of class TSQLMonitor.

25.1.9.4.11 TSQLMonitor.UpdateTraceCallBack Method

Pascal

```
procedure UpdateTraceCallBack;
```

C++

```
__fastcall UpdateTraceCallBack();
```

Description

This is UpdateTraceCallBack, a member of class TSQLMonitor.

25.1.9.5 TSQLMonitor Properties

The properties of the TSQLMonitor class are listed here.

Public Properties

	Name	Description
	MaxTraceCount (see page 1651)	Indicates the maximum number of messages that can be logged.
R	TraceCount (see page 1651)	Indicates the number of messages currently logged.

Published Properties

	Name	Description
	Active (see page 1649)	Starts or stops the monitor from recording SQL commands passed to the database server.
	AutoSave (see page 1650)	Specifies whether the monitored messages are automatically saved to a file.
	FileName (see page 1650)	Specifies the default file to which monitored messages are saved.
	SQLConnection (see page 1651)	Specifies the connection components whose messages are to be monitored.
	TraceList (see page 1652)	Lists the messages that have passed between SQLConnection (see page 1651) and the database server.

25.1.9.5.1 TSQLMonitor.Active Property

Starts or stops the monitor from recording SQL commands passed to the database server.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Set Active to true to start monitoring the communication between SQLConnection (see page 1651) and its database server. Set Active to false to stop the monitoring.

When you set Active to true, TSQLMonitor (see page 1643) sets the TraceCallbackEvent property of the associated SQL connection component. This causes all commands sent to the database server that match the criteria established by `TraceFlags` to be added to the monitor's string list.

When you set Active to false, TSQLMonitor (see page 1643) clears the TraceCallbackEvent property of the associated SQL connection component. No subsequent commands are added to TraceList (see page 1652), but the list is not cleared. That is, any commands currently in the list remain there until you explicitly clear the list.

See Also

TraceCallbackEvent (see page 1628)

SQLConnection (see page 1651)

TraceList (see page 1652)

Debugging dbExpress Applications

Fetching the Data using TSQLDataSet

Using TSimpleDataSet

Using TSQLQuery

Using TSQLStoredProc

25.1.9.5.2 TSQLMonitor.AutoSave Property

Specifies whether the monitored messages are automatically saved to a file.

Pascal

```
property AutoSave: Boolean;
```

C++

```
__property Boolean AutoSave;
```

Description

Set AutoSave to true to automatically save all messages that match the criteria established by `TraceFlags` to the file specified by the FileName (see page 1650) property. The FileName (see page 1650) property must be set to a valid file name.

When AutoSave is false, messages that match the criteria specified by `TraceFlags` are automatically added to TraceList (see page 1652), but are not saved to a file.

See Also

FileName (see page 1650)

TraceList (see page 1652)

SaveToFile (see page 1646)

Debugging dbExpress Applications

25.1.9.5.3 TSQLMonitor.FileName Property

Specifies the default file to which monitored messages are saved.

Pascal

```
property FileName: string;
```

C++

```
__property UnicodeString FileName;
```

Description

Set FileName to the name of a file to which the messages listed in TraceList (see page 1652) are saved. FileName can be a fully-qualified path name.

Once FileName is specified, setting AutoSave (see page 1650) to true causes all monitored messages to be automatically saved to the specified file. Even if AutoSave (see page 1650) is false, FileName specifies the default file name used by the LoadFromFile (see page 1646) and SaveToFile (see page 1646) methods when they are called with an empty string for the AFileName parameter.

See Also

AutoSave (see page 1650)

SaveToFile (see page 1646)

TraceList (see page 1652)

Debugging dbExpress Applications

25.1.9.5.4 TSQLMonitor.MaxTraceCount Property

Indicates the maximum number of messages that can be logged.

Pascal

```
property MaxTraceCount: Integer;
```

C++

```
__property int MaxTraceCount;
```

Description

Set MaxTraceCount to limit the number of messages that can be added to the TraceList (see page 1652) property. When MaxTraceCount is 0 or greater, the SQL monitor logs every message until the limit specified by MaxTraceCount. Once that limit has been reached, the SQL monitor discards messages until the number of messages in TraceList (see page 1652) has been reduced or the value of MaxTraceCount is changed.

If MaxTraceCount is –1, there is no limit on the number of messages that can be added to TraceList (see page 1652).

If MaxTraceCount is 0, the SQL monitor does not log any messages.

Note: Setting MaxTraceCount affects whether messages can be added to TraceList (see page 1652), but does not affect messages that have already been logged. Thus, if you set MaxTraceCount to a value less than the number of strings currently in TraceList (see page 1652), the number of logged messages will exceed MaxTraceCount.

See Also

TraceList (see page 1652)

TraceCount (see page 1651)

25.1.9.5.5 TSQLMonitor.SQLConnection Property

Specifies the connection components whose messages are to be monitored.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Set SQLConnection to identify the TSQLConnection (see page 1580) component whose communication with a database server you want to monitor.

Note: Changing the value of SQLConnection automatically sets the Active (see page 1649) property to false.

See Also

Active (see page 1649)

Debugging dbExpress Applications

25.1.9.5.6 TSQLMonitor.TraceCount Property

Indicates the number of messages currently logged.

Pascal

```
property TraceCount: Integer;
```

C++

```
__property int TraceCount;
```

Description

TraceCount indicates the number of messages listed by the TraceList (see page 1652) property.

See Also

TraceList (see page 1652)

MaxTraceCount (see page 1651)

25.1.9.5.7 TSQLMonitor.TraceList Property

Lists the messages that have passed between SQLConnection (see page 1651) and the database server.

Pascal

```
property TraceList: TWideStrings;
```

C++

```
__property TWideStrings TraceList;
```

Description

Use the TraceList property to access the list of SQL commands that has already been logged. While the SQL monitor is active, the TraceList property is automatically updated when the connection component passes a message to the database server.

Only messages that meet the criteria specified by the `TraceFlags` property are added to TraceList.

TraceList is a TStrings descendant. As such, you can use its methods to

Clear the list.

Access the individual commands in the list.

Add your own comments to the list. For example, you may want to indicate when your application initiates certain actions so that you can see clearly what SQL messages correspond to what actions.

See Also

SQLConnection (see page 1651)

Active (see page 1649)

SaveToFile (see page 1646)

LoadFromFile (see page 1646)

TraceCount (see page 1651)

Debugging dbExpress Applications

25.1.9.6 TSQLMonitor Events

The events of the TSQLMonitor class are listed here.

Published Events

	Name	Description
	OnLogTrace (see page 1653)	Occurs immediately after a new message is logged to the list.
	OnTrace (see page 1653)	Occurs when a message is detected but before it is logged to the list.

25.1.9.6.1 TSQLMonitor.OnLogTrace Event

Occurs immediately after a new message is logged to the list.

Pascal

```
property OnLogTrace: TTraceLogEvent;
```

C++

```
__property TTraceLogEvent OnLogTrace;
```

Description

Write (see page 3707) an OnLogTrace event handler to respond immediately after a new string is added to the TraceList (see page 1652) property. For example, you can use this event to periodically save the contents of the list to a file and clear the list so that it does not get prohibitively long. OnLogTrace occurs after the message is added to TraceList (see page 1652) but before the list is saved to a file when AutoSave (see page 1650) is true.

Note: OnLogTrace occurs only after logging messages that meet the criteria specified by TraceFlags.

See Also

TraceList (see page 1652)

OnTrace (see page 1653)

TraceFlags

AutoSave (see page 1650)

Debugging dbExpress Applications

25.1.9.6.2 TSQLMonitor.OnTrace Event

Occurs when a message is detected but before it is logged to the list.

Pascal

```
property OnTrace: TTraceEvent;
```

C++

```
__property TTraceEvent OnTrace;
```

Description

Write (see page 3707) an OnTrace event handler to screen or edit messages before they are added to TraceList (see page 1652). OnTrace occurs for every message that passes between the dbExpress driver and the database server, not just those that meet the criteria specified by `TraceFlags`.

See Also

TraceList (see page 1652)

OnLogTrace (see page 1653)

25.1.10 TSQLQuery Class

TSQLQuery represents a query that is executed using dbExpress.

Class Hierarchy

Pascal

```
TSQLQuery = class(TCustomSQLDataSet);
```

C++

```
class TSQLQuery : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLQuery to execute an SQL (see page 1663) command on a database server that is accessed via TSQLConnection (see page 1580). TSQLQuery can represent the results of a SELECT statement or perform actions on the database server using statements such as INSERT, DELETE, UPDATE, ALTER TABLE, and so on. You can add a TSQLQuery component to a form or data module at design time, or create one dynamically at runtime.

TSQLQuery is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL (see page 1663) query by explicitly creating an SQL (see page 1663) UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLQuery component, it must be connected to the database server. Therefore, the first step to take when working with TSQLQuery is to set the SQLConnection (see page 1664) property. Once the query is connected to a server, use the SQL (see page 1663) property to specify the command your query executes.

See Also

TSQLDataSet (see page 1631)

TSQLStoredProc (see page 1671)

TSQLTable (see page 1680)

TSQLConnection (see page 1580)

TClientDataSet

TDataSetProvider

Types of Datasets

Specifying the Command to Execute

Types of Unidirectional Datasets

25.1.10.1 TSQLQuery Members

The following tables list the members exposed by TSQLQuery.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1539)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.

	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1555)	Indicates the meaning of the CommandText (see page 1554) property.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.
	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DbxCommandType (see page 1557)	This is DbxCommandType, a member of class TCustomSQLDataSet.
	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.
	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1555) is ctTable.

Public Methods

	Name	Description
	Create (see page 1657)	Creates an instance of a TSQLQuery (see page 1654) component.
	Create (see page 1524)	This is the overview for the Create constructor overload.
	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	Destroy (see page 1658)	Destroys the TSQLQuery (see page 1654) instance.
	ExecSQL (see page 1658)	Executes a query that does not return a set of records.
	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
	PrepareStatement (see page 1659)	Prepares the query for execution.
	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1554)	Specifies the command the dataset executes.

	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	RowsAffected (see page 1663)	Indicates the number of rows operated on by the latest query execution.
R	Text (see page 1665)	Indicates the SQL (see page 1663) statement as a single string.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Published Properties

	Name	Description
	Active (see page 1552)	Specifies whether or not a dataset is open.
	DataSource (see page 1660)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 1660)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1661)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	ParamCheck (see page 1661)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1662)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1565)	Specifies the name of the database schema.
	SQL (see page 1663)	Specifies the SQL statement to execute on the database server.
	SQLConnection (see page 1664)	Specifies the SQL connection component that connects the dataset to a database server.

25.1.10.2 TSQLQuery.Create Constructor

Creates an instance of a TSQLQuery (see page 1654) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLQuery(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL (see page 1663) query component at runtime. SQL (see page 1663) queries placed in forms or data modules at design time are created automatically.

See Also

Create

25.1.10.3 TSQLQuery.Destroy Destructor

Destroys the TSQLQuery (see page 1654) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLQuery();
```

Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner

Create (see page 1657)

System::TObject::Free (see page 3372)

25.1.10.4 TSQLQuery Methods

The methods of the TSQLQuery class are listed here.

Public Methods

	Name	Description
	ExecSQL (see page 1658)	Executes a query that does not return a set of records.
	PrepareStatement (see page 1659)	Prepares the query for execution.

25.1.10.4.1 TSQLQuery.ExecSQL Method

Executes a query that does not return a set of records.

Pascal

```
function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

C++

```
virtual __fastcall int ExecSQL(Boolean ExecDirect = False);
```

Description

Call ExecSQL to execute an SQL (see page 1663) command that does not return a set of records. This command is a query other than a SELECT query, such as an INSERT, UPDATE, DELETE, or CREATE TABLE query.

ExecDirect indicates that the query does not need to be prepared before it is executed. This parameter can be set to true if the query does not include any parameters. When ExecDirect is false, the query will be prepared before it is executed. Even when ExecDirect is false, you can avoid preparing the query every time ExecSQL is called by setting the Prepared (see page 1563) property to true before calling ExecSQL the first time.

ExecSQL returns the number of rows affected by the executed command. This becomes the value of the RowsAffected (see page 1663) property.

Note: Do not use ExecSQL for commands such as SELECT queries that return data. When the command returns data, use the Open method or set the Active (see page 1552) property to true.

See Also

SQL (see page 1663)

RowsAffected (see page 1564)

Open

Active

Prepared (see page 1563)

ExecuteDirect (see page 1590)

25.1.10.4.2 TSQLQuery.PrepareStatement Method

Prepares the query for execution.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement performs most of the necessary tasks for preparing a query to be executed on the server. It ensures the dbExpress and the database server have allocated any resources they need for executing the query and binding its parameters. Any named parameters are temporarily converted to unnamed parameters because dbExpress does not support named parameters.

Do not call PrepareStatement when preparing a query for execution. Instead, set the Prepared (see page 1563) property to true. When you use the Prepared (see page 1563) property instead, the query keeps track of the fact that it has already been prepared and need not repeat the process every time it executes the query. In addition, when you set the Prepared (see page 1563) property, the query initializes internal properties that keep track of the number of rows affected by the query.

See Also

Prepared (see page 1563)

25.1.10.5 TSQLQuery Properties

The properties of the TSQLQuery class are listed here.

Public Properties

	Name	Description
	RowsAffected (see page 1663)	Indicates the number of rows operated on by the latest query execution.

R	Text (see page 1665)	Indicates the SQL (see page 1663) statement as a single string.

Published Properties

	Name	Description
	DataSource (see page 1660)	Links the SQL dataset to another (master) dataset.
	GetMetadata (see page 1660)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1661)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	ParamCheck (see page 1661)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1662)	Represents the parameters for a query or stored procedure.
	SQL (see page 1663)	Specifies the SQL statement to execute on the database server.
	SQLConnection (see page 1664)	Specifies the SQL connection component that connects the dataset to a database server.

25.1.10.5.1 TSQLQuery.DataSource Property

Links the SQL dataset to another (master) dataset.

Pascal

```
property DataSource: TDataSource;
```

C++

```
__property TDataSource DataSource;
```

Description

DataSource links this SQL dataset to another (master) dataset that supplies values for this dataset's parameters.

When the SQL dataset's CommandText (or SQL) property specifies a parameterized query but the application does not provide values for the corresponding parameters in the Params property, the SQL dataset fetches parameter values from the dataset identified with the specified data source. When the current record in the dataset associated with DataSource changes, the SQL dataset refreshes its data to reflect the new parameter values.

See Also

CommandText (see page 1554)

SQL (see page 1663)

Params (see page 1562)

MasterSource (see page 1692)

25.1.10.5.2 TSQLQuery.GetMetadata Property

Specifies whether the SQL dataset fetches metadata information along with the data.

Pascal

```
property GetMetadata: Boolean;
```

C++

```
__property Boolean GetMetadata;
```

Description

Use GetMetadata to turn on or off the fetching of metadata on a database object. Setting GetMetadata to False will improve performance because no additional SQL will execute in order to get index information. However, setting GetMetadata to False might slow any attempts to update data using a client dataset that is linked via a dataset provider because the resolver might not

generate optimized SQL to perform the update.

GetMetadata must be True to perform any operations that require an index. These operations include setting up master/detail relationships, updating data from a client dataset when the provider's UpdateMode is not upWhereAll, and delayed fetching of BLOB data.

Set GetMetadata to False if you are fetching a dataset for read-only purposes.

See Also

TClientDataSet

TDataSetProvider

Fetching the Data using TSQLDataSet

25.1.10.5.3 TSQLQuery.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1530)

Params (see page 1621)

SQLConnection (see page 1566)

25.1.10.5.4 TSQLQuery.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 1562)

CommandText (see page 1554)

SQL (see page 1663)

25.1.10.5.5 TSQLQuery.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1561)

SQL (see page 1663)

StoredProcName (see page 1680)

CommandText (see page 1554)

TParam

Value

DataType

DataSource (see page 1557)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

25.1.10.5.6 TSQLQuery.RowsAffected Property

Indicates the number of rows operated on by the latest query execution.

Pascal

```
property RowsAffected: Integer;
```

C++

```
__property int RowsAffected;
```

Description

When the SQL dataset represents a query that does not return a result set, you can't use the RecordCount property to ascertain the number of records affected by the query. In such cases, read RowsAffected to determine how many rows were updated or deleted by the query when it executed. If no rows were updated or deleted, RowsAffected is zero. RowsAffected is –1 if the SQL statement could not be executed due to an error condition. (This latter situation typically follows the throwing of an exception.)

See Also

ExecSQL (see page 1636)

RecordCount (see page 1564)

ExecProc (see page 1675)

25.1.10.5.7 TSQLQuery.SQL Property

Specifies the SQL statement to execute on the database server.

Pascal

```
property SQL: TWideStrings;
```

C++

```
__property TWideStrings SQL;
```

Description

Use the properties and methods of SQL to specify the query that should be executed on the database server. If SQL is a SELECT statement, the query executes the statement when you call Open or set the Active (see page 1552) property to true. If the statement does not return a result set, call the ExecSQL (see page 1658) method to execute the statement.

When you set the SQL property, TSQLQuery (see page 1654) checks the statement for parameters and initializes the Params (see page 1662) property to reflect any parameters in the statement. You can subsequently use the Params (see page 1662) property to adjust the datatype of any parameters and supply values for the parameters.

At design-time, edit the SQL statement using the property editor that appears when you click the ellipsis button for the property in the Object Inspector. At runtime, use properties and methods of TStrings to clear the current contents, add new contents, or to modify existing contents:

SQLQuery1.SQL.Clear;

```
SQLQuery1.SQL.Add('SELECT ' + Edit1.Text + ' FROM ' + Edit2.Text);
if Length(Edit3.Text) <> 0 then
```

SQLQuery1.SQL.Add('ORDER BY ' + Edit3.Text (see page 1665))

```
SQLQuery1->SQL->Clear();
SQLQuery1->SQL->Add("SELECT " + Edit1->Text + " FROM " + Edit2.Text);
if (!Edit3->Text->IsEmpty())
```

SQLQuery1->SQL->Add("ORDER BY " + Edit3.Text (see page 1665));

You can also use the methods of SQL to load a query from a file.

Note: The SQL you supply for this property must be valid for the database server to which the TSQLQuery (see page 1654) object is connected.

See Also

Add

Active

Open

ExecSQL (see page 1658)

Params (see page 1562)

Text (see page 1665)

Representing the Results of a Query

Specifying the Command to Execute

25.1.10.5.8 TSQLQuery.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active

CommandText (see page 1554)

Params (see page 1562)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

25.1.10.5.9 TSQLQuery.Text Property

Indicates the SQL (see page 1663) statement as a single string.

Pascal

```
property Text: string;
```

C++

```
__property UnicodeString Text;
```

Description

Use Text as a convenient way to read the entire SQL (see page 1663) query as a single string, rather than using the SQL (see page 1663) property, which may divide the query into substrings. Text is the same as the Text property of the TStrings object that implements the SQL (see page 1663) property.

See Also

SQL (see page 1663)

Text

25.1.11 TSqlServerMethod Class

TDataSet used to call DataSnap server methods.

Class Hierarchy

Pascal

```
TSqlServerMethod = class(TCustomSQLDataSet);
```

C++

```
class TSqlServerMethod : public TCustomSQLDataSet;
```

File

SqlExpr

Description

TSqlServerMethod is a TDataSet used to call DataSnap server methods and retrieve results from them.

See Also

TCustomSQLDataSet (see page 1520)

TDataSet

25.1.11.1 TSqlServerMethod Members

The following tables list the members exposed by TSqlServerMethod.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1529)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1539)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.

	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.
	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1555)	Indicates the meaning of the CommandText (see page 1554) property.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.
	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1557)	Links the SQL dataset to another (master) dataset.
	DbxCommandType (see page 1557)	This is DbxCommandType, a member of class TCustomSQLDataSet.
	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	MaxBlobSize (see page 1559)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 1561)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	RowsAffected (see page 1564)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1555) is ctTable.

Public Methods

	Name	Description
	Create (see page 1668)	Create new instance of TSqlServerMethod (see page 1665).
	Create (see page 1524)	This is the overview for the Create constructor overload.
	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	Destroy (see page 1525)	Destroys the TCustomSQLDataSet (see page 1520) instance.
	ExecuteMethod (see page 1669)	Executes remote server method.
	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.

	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
	PrepareStatement (see page 1540)	Performs a prepare on the SQL statement.
	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1554)	Specifies the command the dataset executes.
	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Published Properties

	Name	Description
	Active (see page 1552)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1558)	Specifies whether the SQL dataset fetches metadata information along with the data.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	Params (see page 1669)	TParams for method to be executed.
	SchemaName (see page 1565)	Specifies the name of the database schema.
	ServerMethodName (see page 1670)	Sets method name of remote DataSnap server method.
	SQLConnection (see page 1670)	TSQLConnection (see page 1580) used.

25.1.11.2 TSqlServerMethod.Create Constructor

Create new instance of TSqlServerMethod (see page 1665).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSqlServerMethod(TComponent * AOwner);
```

Description

This method creates a new instance of TSqlServerMethod (see page 1665).

25.1.11.3 TSqlServerMethod Methods

The methods of the TSqlServerMethod class are listed here.

Public Methods

	Name	Description
	ExecuteMethod (see page 1669)	Executes remote server method.

25.1.11.3.1 TSqlServerMethod.ExecuteMethod Method

Executes remote server method.

Pascal

```
procedure ExecuteMethod;
```

C++

```
__fastcall ExecuteMethod();
```

Description

ExecuteMethod executes a remote server method. It does not return a result set. Instead, it uses the property Params (see page 1669) for input and output parameters of the server method.

See Also

Params (see page 1669)

25.1.11.4 TSqlServerMethod Properties

The properties of the TSqlServerMethod class are listed here.

Published Properties

	Name	Description
	Params (see page 1669)	TParams for method to be executed.
	ServerMethodName (see page 1670)	Sets method name of remote DataSnap server method.
	SQLConnection (see page 1670)	TSQLConnection (see page 1580) used.

25.1.11.4.1 TSqlServerMethod.Params Property

TParams for method to be executed.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params contains TParams for a server method to be executed by ExecuteMethod (see page 1669). ExecuteMethod (see page 1669) uses Params for input and output parameters of the server method.

See Also

TParams

ExecuteMethod (see page 1669)

25.1.11.4.2 TSqlServerMethod.ServerMethodName Property

Sets method name of remote DataSnap server method.

Pascal

```
property ServerMethodName: UnicodeString;
```

C++

```
__property UnicodeString ServerMethodName;
```

Description

ServerMethodName sets the method name of the remote DataSnap server method to be executed by ExecuteMethod (see page 1669).

The method name may also be specified in CommandText (see page 1554). In this case, it is of the form *partialClassName.methodName*, where *partialClassName* is the name of the server methods class without the T prefix and *methodName* is the server method name.

See Also

CommandText (see page 1554)

Params (see page 1669)

ExecuteMethod (see page 1669)

25.1.11.4.3 TSqlServerMethod.SQLConnection Property

TSQLConnection (see page 1580) used.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

SQLConnection contains the TSQLConnection (see page 1580).

25.1.12 TSQLStoredProc Class

TSQLStoredProc represents a stored procedure that is executed using dbExpress.

Class Hierarchy

Pascal

```
TSQLStoredProc = class(TCustomSQLDataSet);
```

C++

```
class TSQLStoredProc : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLStoredProc to execute a stored procedure on a database server that is accessed via TSQLConnection (see page 1580). TSQLStoredProc can represent the result set if the stored procedure returns a cursor. You can add a TSQLStoredProc component to a form or data module at design time, or create one dynamically at runtime.

TSQLStoredProc is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL stored procedure by explicitly creating an SQL UPDATE command or by connecting the dataset to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLStoredProc component, it must be connected to the database server. Therefore, the first step to take when working with TSQLStoredProc is to set the SQLConnection (see page 1679) property. Once the dataset is connected to a server, use the StoredProcName (see page 1680) property to specify the stored procedure to execute.

See Also

TSQLDataSet (see page 1631)

TSQLQuery (see page 1654)

TSQLTable (see page 1680)

TSQLConnection (see page 1580)

TClientDataSet

TDataSetProvider

Types of Datasets

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Types of Unidirectional Datasets

25.1.12.1 TSQLStoredProc Members

The following tables list the members exposed by TSQLStoredProc.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
V	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
V	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
V	ExecSQL (see page 1529)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
V	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.
V	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.
	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1539)	This is OpenCursor, a member of class TCustomSQLDataSet.
V	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.

	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.
	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1555)	Indicates the meaning of the CommandText (see page 1554) property.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.
	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1557)	Links the SQL dataset to another (master) dataset.
	DbxCommandType (see page 1557)	This is DbxCommandType, a member of class TCustomSQLDataSet.
	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.
	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
	RowsAffected (see page 1564)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1555) is ctTable.

Public Methods

	Name	Description
	Create (see page 1674)	Creates an instance of a TSQLStoredProc (see page 1671) component.
	Create (see page 1524)	This is the overview for the Create constructor overload.
	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	Destroy (see page 1525)	Destroys the TCustomSQLDataSet (see page 1520) instance.
	ExecProc (see page 1675)	Executes the stored procedure when it does not return a cursor.
	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.
	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	NextRecordSet (see page 1676)	Provides access to a secondary recordset.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
	PrepareStatement (see page 1676)	Prepares the stored procedure for execution.
	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1554)	Specifies the command the dataset executes.
	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Published Properties

	Name	Description
	Active (see page 1552)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1558)	Specifies whether the SQL dataset fetches metadata information along with the data.
	MaxBlobSize (see page 1677)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	PackageName (see page 1677)	Specifies the name of the Oracle package to which the stored procedure belongs.
	ParamCheck (see page 1678)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1678)	Represents the parameters for a query or stored procedure.
	SchemaName (see page 1565)	Specifies the name of the database schema.
	SQLConnection (see page 1679)	Specifies the SQL connection component that connects the dataset to a database server.
	StoredProcName (see page 1680)	Specifies the SQL statement to execute on the database server.

25.1.12.2 TSQLStoredProc.Create Constructor

Creates an instance of a TSQLStoredProc (see page 1671) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLStoredProc(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL stored procedure component at runtime. SQL stored procedures placed in forms or data modules at design time are created automatically.

AOwner is the component, typically a form or data module, that is responsible for freeing the TSQLStoredProc (see page 1671) instance. It becomes the value of the Owner property.

See Also

Owner

Create

25.1.12.3 TSQLStoredProc Methods

The methods of the TSQLStoredProc class are listed here.

Public Methods

	Name	Description
	ExecProc (see page 1675)	Executes the stored procedure when it does not return a cursor.
	NextRecordSet (see page 1676)	Provides access to a secondary recordset.
	PrepareStatement (see page 1676)	Prepares the stored procedure for execution.

25.1.12.3.1 TSQLStoredProc.ExecProc Method

Executes the stored procedure when it does not return a cursor.

Pascal

```
function ExecProc: Integer; virtual;
```

C++

```
virtual __fastcall int ExecProc();
```

Description

Call ExecProc to execute the stored procedure specified by StoredProcName (see page 1680) if it does not return a cursor. ExecProc returns the number of rows affected by the stored procedure. This becomes the value of the RowsAffected (see page 1564) property.

To speed performance, an application should ordinarily prepare the stored procedure by setting the Prepared (see page 1563) property to true before calling ExecProc the first time.

Note: Do not use ExecProc for stored procedures that return a cursor. When the stored procedure returns data, use the Open method or set the Active (see page 1552) property to true.

See Also

StoredProcName (see page 1680)

RowsAffected (see page 1564)

Prepared (see page 1563)

Executing the Command

25.1.12.3.2 TSQLStoredProc.NextRecordSet Method

Provides access to a secondary recordset.

Pascal

```
function NextRecordSet: TCustomSQLDataSet;
```

C++

```
__fastcall TCustomSQLDataSet NextRecordSet();
```

Description

Call NextRecordSet to obtain secondary datasets when accessing the results of a stored procedure that returns multiple sets of records. Opening the dataset fetches the first set of records. NextRecordSet provides access to the subsequent sets.

NextRecordSet returns a newly created TCustomSQLDataSet (see page 1520) component that provides access to the next set of records. The first time you call NextRecordSet, it returns a dataset for the second set of records. The next call returns a third dataset, and so on, until there are no more sets of records. When there are no additional datasets, NextRecordSet returns nil (Delphi) or NULL (C++).

See Also

Fetching the Data

Fetching the Data using TSQLDataSet

25.1.12.3.3 TSQLStoredProc.PrepareStatement Method

Prepares the stored procedure for execution.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement performs most of the necessary tasks for preparing a stored procedure to be executed on the server. It ensures that dbExpress and the database server have allocated any resources they need for executing the stored procedure and binding its parameters.

Do not call PrepareStatement when preparing a stored procedure for execution. Instead, set the Prepared (see page 1563) property to true. When you use the Prepared (see page 1563) property instead, the stored procedure component keeps track of the fact that it has already been prepared and need not repeat the process every time it executes. In addition, when you set the Prepared (see page 1563) property, the component initializes internal properties that keep track of the number of rows affected by the stored procedure.

See Also

Prepared (see page 1563)

25.1.12.4 TSQLStoredProc Properties

The properties of the TSQLStoredProc class are listed here.

Published Properties

	Name	Description
	MaxBlobSize (see page 1677)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	PackageName (see page 1677)	Specifies the name of the Oracle package to which the stored procedure belongs.
	ParamCheck (see page 1678)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
	Params (see page 1678)	Represents the parameters for a query or stored procedure.
	SQLConnection (see page 1679)	Specifies the SQL connection component that connects the dataset to a database server.
	StoredProcName (see page 1680)	Specifies the SQL statement to execute on the database server.

25.1.12.4.1 TSQLStoredProc.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1530)

Params (see page 1621)

SQLConnection (see page 1566)

25.1.12.4.2 TSQLStoredProc.PackageName Property

Specifies the name of the Oracle package to which the stored procedure belongs.

Pascal

```
property PackageName: UnicodeString;
```

C++

```
__property UnicodeString PackageName;
```

Description

Set PackageName to specify the name of the Oracle package that contains the stored procedure which the StoredProcName (see page 1680) property specifies. This property is only used when the stored procedure is defined on an Oracle server.

If the stored procedure does not belong to a package, the value of PackageName is an empty string.

See Also

StoredProcName (see page 1680)

GetPackageNames (see page 1594)

25.1.12.4.3 TSQLStoredProc.ParamCheck Property

Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.

Pascal

```
property ParamCheck: Boolean;
```

C++

```
__property Boolean ParamCheck;
```

Description

By default, the Params property is updated automatically to reflect any parameters that appear in the query specified by the CommandText property (or the SQL property for TSQLQuery) at runtime. That is, whenever the CommandText property changes, the Params property is regenerated to reflect the new set of parameters.

There are times when you want to override this default behavior. For example, when specifying a data definition language (DDL) statement that defines a stored procedure, the resulting SQL command may include parameters. These parameters, however, should not be used as parameters of the SQL dataset, because they are not parameters of the DDL command, but rather, they are parameters of the stored procedure that the command creates.

In order to override the default generation of the Params property in response to changes in CommandText, set ParamCheck to false.

See Also

Params (see page 1562)

CommandText (see page 1554)

SQL (see page 1663)

25.1.12.4.4 TSQLStoredProc.Params Property

Represents the parameters for a query or stored procedure.

Pascal

```
property Params: TParams;
```

C++

```
__property TParams Params;
```

Description

Params is a collection of TParam objects that represent the parameters of a query or stored procedure specified by the SQL dataset. When you specify a query by setting the CommandText property (or the SQL property in TSQLQuery), the SQL dataset automatically parse the query and fills Params with a TParam object for every parameter in the query. Similarly, when you specify a stored procedure by setting the CommandText property (or the StoredProcName property in TSQLStoredProc), the SQL dataset automatically asks the server for a list of stored procedure parameters and fills Params with a TParam object for every parameter of the stored procedure.

At design-time, use the Params property editor to access the individual TParam objects. For queries, use this property editor to check the DataType property of each parameter and to assign values to the parameters. For stored procedures, you may need

to do more, changing the type or number of parameters if the server does not supply complete information about stored procedure parameters.

You can also use Params to assign parameter properties in code at runtime.

Note: If you do not assign values to input parameters before opening the dataset, it fetches values from the dataset identified by the DataSource property. However, this only works if the individual TParam objects have the same names as fields in the master dataset.

See Also

ParamCheck (see page 1561)

SQL (see page 1663)

StoredProcName (see page 1680)

CommandText (see page 1554)

TParam

Value

DataType

DataSource (see page 1557)

Configuring TSQL Connection

Connecting to Databases with TDatabase

Using TQuery

Using TSimpleDataSet

25.1.12.4.5 TSQLStoredProc.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active

CommandText (see page 1554)

Params (see page 1562)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

25.1.12.4.6 TSQLStoredProc.StoredProcName Property

Specifies the SQL statement to execute on the database server.

Pascal

```
property StoredProcName: UnicodeString;
```

C++

```
__property UnicodeString StoredProcName;
```

Description

Set StoredProcName to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, an exception is raised.

When you set the StoredProcName property, TSQLStoredProc (see page 1671) queries the database server to ascertain the parameters of the specified stored procedure. It uses this information to initialize the Params (see page 1678) property. You can subsequently use the Params (see page 1678) property to adjust the properties of any parameters and supply values for the parameters. Some servers do not supply information about stored procedure parameters. In those cases, you must explicitly add the parameters.

See Also

Params (see page 1562)

PackageName (see page 1677)

Representing the Results of a Stored Procedure

Specifying the Command to Execute

Executing the Commands using TSQLDataSet

Specifying the Data to Display using TSQLDataSet

Using TSQLStoredProc

25.1.13 TSQLTable Class

TSQLTable represents a database table that is accessed using dbExpress.

Class Hierarchy

Pascal

```
TSQLTable = class(TCustomSQLDataSet);
```

C++

```
class TSQLTable : public TCustomSQLDataSet;
```

File

SqlExpr

Description

Use TSQLTable to represent a table on a database server that is accessed via TSQLConnection (see page 1580). TSQLTable generates a query to fetch all of the rows and columns in a table you specify. You can add a TSQLTable component to a form or data module at design time, or create one dynamically at runtime.

TSQLTable is a unidirectional dataset. Unlike other datasets, unidirectional datasets do not buffer multiple records in memory. Because of this, you can only navigate using the First and Next methods. There is no built-in editing support: you can only edit the data in an SQL table by explicitly creating an SQL UPDATE command or by connecting the table to a client dataset using a provider. Features that require buffering multiple records, such as filters or lookup fields, are not available.

Before you can use a TSQLTable component, it must be connected to the database server. Therefore, the first step to take when working with TSQLTable is to set the SQLConnection (see page 1693) property. Once the dataset is connected to a server, use the TableName (see page 1693) property to specify the table this component represents.

See Also

TSQLDataSet (see page 1631)

TSQLQuery (see page 1654)

TSQLStoredProc (see page 1671)

TSQLConnection (see page 1580)

TClientDataSet

TDataSetProvider

Types of Datasets

Types of Unidirectional Datasets

25.1.13.1 TSQLTable Members

The following tables list the members exposed by TSQLTable.

Protected Methods

	Name	Description
	AddFieldDesc (see page 1527)	This is AddFieldDesc, a member of class TCustomSQLDataSet.
	AddIndexDefs (see page 1527)	This is AddIndexDefs, a member of class TCustomSQLDataSet.
	CheckPrepareError (see page 1527)	This is CheckPrepareError, a member of class TCustomSQLDataSet.
	ClearIndexDefs (see page 1528)	This is ClearIndexDefs, a member of class TCustomSQLDataSet.
	CloseCursor (see page 1528)	This is CloseCursor, a member of class TCustomSQLDataSet.
	CloseStatement (see page 1528)	This is CloseStatement, a member of class TCustomSQLDataSet.
	DefineProperties (see page 1529)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	ExecSQL (see page 1529)	This is ExecSQL, a member of class TCustomSQLDataSet.
	ExecuteStatement (see page 1529)	This is ExecuteStatement, a member of class TCustomSQLDataSet.
	FreeBuffers (see page 1530)	This is FreeBuffers, a member of class TCustomSQLDataSet.
	FreeCommand (see page 1530)	This is FreeCommand, a member of class TCustomSQLDataSet.
	FreeReader (see page 1530)	This is FreeReader, a member of class TCustomSQLDataSet.
	GetCanModify (see page 1531)	This is GetCanModify, a member of class TCustomSQLDataSet.
	GetDataSource (see page 1531)	This is GetDataSource, a member of class TCustomSQLDataSet.
	GetObjectTypeNames (see page 1533)	This is GetObjectTypeNames, a member of class TCustomSQLDataSet.

	GetOutputParams (see page 1534)	This is GetOutputParams, a member of class TCustomSQLDataSet.
V	GetRecord (see page 1534)	This is GetRecord, a member of class TCustomSQLDataSet.
V	GetRecordCount (see page 1534)	This is GetRecordCount, a member of class TCustomSQLDataSet.
	GetSortFieldNames (see page 1535)	This is GetSortFieldNames, a member of class TCustomSQLDataSet.
V	InitRecord (see page 1535)	This is InitRecord, a member of class TCustomSQLDataSet.
V	InternalClose (see page 1535)	This is InternalClose, a member of class TCustomSQLDataSet.
	InternalFreeCommand (see page 1535)	This is InternalFreeCommand, a member of class TCustomSQLDataSet.
V	InternalHandleException (see page 1535)	This is InternalHandleException, a member of class TCustomSQLDataSet.
V	InternalInitFieldDefs (see page 1536)	This is InternalInitFieldDefs, a member of class TCustomSQLDataSet.
V	InternalOpen (see page 1536)	This is InternalOpen, a member of class TCustomSQLDataSet.
V	InternalRefresh (see page 1536)	This is InternalRefresh, a member of class TCustomSQLDataSet.
V	IsCursorOpen (see page 1536)	This is IsCursorOpen, a member of class TCustomSQLDataSet.
V	Loaded (see page 1537)	Initializes the component after the form file has been read into memory.
	LocateRecord (see page 1538)	This is LocateRecord, a member of class TCustomSQLDataSet.
V	OpenCursor (see page 1686)	This is OpenCursor, a member of class TSQLTable.
V	OpenSchema (see page 1539)	This is OpenSchema, a member of class TCustomSQLDataSet.
	PropertyChanged (see page 1540)	This is PropertyChanged, a member of class TCustomSQLDataSet.
V	PSEndTransaction (see page 1540)	Ends a transaction that was started by a call to PSStartTransaction.
V	PSExecute (see page 1541)	Executes the SQL command associated with the dataset.
V	PSExecuteStatement (see page 1541)	This is PSExecuteStatement, a member of class TCustomSQLDataSet.
V	PSGetAttributes (see page 1541)	Returns information that is stored in data packets as Name/Value pairs.
V	PSGetCommandText (see page 1542)	Gets the SQL command to be executed by PSExecute.
V	PSGetCommandType (see page 1542)	Gets the command type to be executed by PSExecute.
V	PSGetDefaultOrder (see page 1542)	Returns the definition of an index that imposes the default order on the data included in data packets.
V	PSGetIndexDefs (see page 1543)	Returns the definitions of all specified indexes defined for the dataset.
V	PSGetKeyFieldsW (see page 1543)	This is PSGetKeyFieldsW, a member of class TCustomSQLDataSet.
V	PSGetParams (see page 1544)	Returns the current parameter values of the dataset.
V	PSGetQuoteCharW (see page 1544)	This is PSGetQuoteCharW, a member of class TCustomSQLDataSet.
V	PSGetTableNameW (see page 1544)	This is PSGetTableNameW, a member of class TCustomSQLDataSet.
V	PSGetUpdateException (see page 1544)	Generates an EUpdateError object based on another exception object.
V	PSInTransaction (see page 1545)	Indicates whether actions against the dataset occur in the context of a transaction.
V	PSIsSQLBased (see page 1545)	Indicates whether the dataset includes native SQL support.
V	PSIsSQLSupported (see page 1546)	Indicates whether the dataset can execute SQL statements.
V	PSReset (see page 1546)	Resets the dataset so that it is positioned on the first record.
V	PSSetCommandText (see page 1546)	Assigns a specified SQL command to be executed by PSExecute.
V	PSSetParams (see page 1547)	Assigns the specified parameter values to the dataset.
V	PSStartTransaction (see page 1547)	Starts a transaction that encapsulates all activity until a call to PSEndTransaction.
V	PSUpdateRecord (see page 1548)	Applies a single update to the underlying database table or tables.
V	SetBufListSize (see page 1548)	This is SetBufListSize, a member of class TCustomSQLDataSet.
V	SetCommandText (see page 1549)	This is SetCommandText, a member of class TCustomSQLDataSet.
	SetFCommandText (see page 1549)	This is SetFCommandText, a member of class TCustomSQLDataSet.
V	SetFieldData (see page 1549)	This is SetFieldData, a member of class TCustomSQLDataSet.
	SetIndexField (see page 1687)	This is SetIndexField, a member of class TSQLTable.
	SetMaxBlobSize (see page 1549)	This is SetMaxBlobSize, a member of class TCustomSQLDataSet.
	SetParamsFromCursor (see page 1549)	This is SetParamsFromCursor, a member of class TCustomSQLDataSet.
	SetSortFieldNames (see page 1551)	This is SetSortFieldNames, a member of class TCustomSQLDataSet.
V	UpdateIndexDefs (see page 1551)	This is UpdateIndexDefs, a member of class TCustomSQLDataSet.

Protected Properties

	Name	Description
	BlobBuffer (see page 1554)	Reserves a buffer for storing BLOB values.
	CommandType (see page 1555)	Indicates the meaning of the CommandText (see page 1554) property.
	CurrentBlobSize (see page 1556)	Indicates the size of the most recently read BLOB field.

R	DataLink (see page 1556)	Identifies the datalink that manages communication between the SQL dataset and another, master dataset.
	DataSource (see page 1557)	Links the SQL dataset to another (master) dataset.
	DbxCommandType (see page 1557)	This is DbxCommandType, a member of class TCustomSQLDataSet.
R	InternalConnection (see page 1559)	Indicates the component that connects the dataset to its database server, taking into account whether the connection was cloned.
R	MasterLink (see page 1692)	This is MasterLink, a member of class TSQLTable.
	NativeCommand (see page 1560)	Represents the SQL command that is sent to the server.
	ParamCheck (see page 1561)	Specifies whether the parameter list for an SQL dataset is regenerated when the SQL command changes.
R	ParamCount (see page 1562)	Indicates the number of parameters listed by the Params (see page 1562) property.
	Params (see page 1562)	Represents the parameters for a query or stored procedure.
	ProcParams (see page 1564)	ProcParams lists the parameter descriptors obtained from the server for any stored procedure parameters.
R	RowsAffected (see page 1564)	Indicates the number of rows operated on by the latest query execution.
	SchemaInfo (see page 1565)	Identifies the metadata with which to populate the dataset.
	SortFieldNames (see page 1566)	Indicates the sort order when CommandType (see page 1555) is ctTable.

Public Methods

	Name	Description
V	Create (see page 1684)	Creates an instance of a TSQLTable (see page 1680) component.
V	Create (see page 1524)	This is the overview for the Create constructor overload.
V	CreateBlobStream (see page 1528)	Creates a blob stream for a Binary large object (BLOB) field in the dataset.
	DeleteRecords (see page 1685)	Empties the underlying database table.
V	Destroy (see page 1685)	Destroys the TSQLTable (see page 1680) instance.
V	GetBlobFieldData (see page 1530)	Retrieves the current value of a BLOB field into a buffer.
	GetCommandNames (see page 1531)	This is GetCommandNames, a member of class TCustomSQLDataSet.
V	GetDetailLinkFields (see page 1531)	Lists the field components that link this dataset as a detail of a master dataset.
V	GetFieldData (see page 1532)	This is the overview for the GetFieldData method overload.
	GetIndexNames (see page 1686)	Retrieves a list of available indexes defined for the table.
	GetKeyFieldNames (see page 1533)	This is the overview for the GetKeyFieldNames method overload.
V	GetQuoteChar (see page 1534)	Returns the character or characters to use in generated SQL statements to enclose quoted strings.
V	IsSequenced (see page 1536)	Indicates whether the dataset can use record numbers to indicate the order of records.
V	Locate (see page 1538)	Implements a virtual method for searching a dataset for a specified record and making it the active record.
V	Lookup (see page 1538)	Implements a virtual method to retrieve field values from a record that matches specified search values.
	ParamByName (see page 1539)	Locates a parameter in the Params (see page 1562) property collection.
V	PrepareStatement (see page 1686)	Generates the query that TSQLTable (see page 1680) uses to fetch data from the database server.
	SetSchemaInfo (see page 1550)	Indicates whether the dataset represents metadata from the server, and if so, what kind.

Public Properties

	Name	Description
	CommandText (see page 1554)	Specifies the command the dataset executes.
	DesignerData (see page 1557)	Stores custom data.
	IndexDefs (see page 1558)	Contains the definitions of all indexes defined on the server for the dataset.
R	IndexFieldCount (see page 1689)	Indicates the number of fields that make up the current index.
	IndexFields (see page 1690)	Specifies the fields associated with the current index.
	Prepared (see page 1563)	Specifies whether the command is prepared before execution.
R	RecordCount (see page 1564)	Indicates the total number of records associated with the dataset.
	TransactionLevel (see page 1567)	Indicates the transaction to which this dataset belongs.

Published Events

	Name	Description
	AfterClose (see page 1568)	Occurs after a data source connection is closed.
	AfterOpen (see page 1568)	Occurs after the connection to a data source is opened.
	AfterRefresh (see page 1568)	Occurs after a DataSet has been changed and is refreshed.
	AfterScroll (see page 1568)	Occurs after a result set is scrolled.
	BeforeClose (see page 1569)	Occurs just before a data connection is closed.
	BeforeOpen (see page 1569)	Occurs just before a data connection is opened, but after an open operation has been initiated.
	BeforeRefresh (see page 1569)	Occurs before a DataSet refresh occurs.
	BeforeScroll (see page 1569)	Occurs when a DataSet scroll operation has been initiated but before the scrolling actually takes place.
	OnCalcFields (see page 1569)	Occurs when calculated or derived fields are updated.
	ParseDeleteSql (see page 1569)	Occurs when an application attempts to parse the DELETE statement stored in the CommandText (see page 1554) property.
	ParseInsertSql (see page 1570)	Occurs when an application attempts to parse the INSERT statement stored in the CommandText (see page 1554) property.
	ParseSelectSql (see page 1570)	Occurs when an application attempts to parse the SELECT statement specified in the CommandText (see page 1554) property.
	ParseUpdateSql (see page 1570)	Occurs when an application attempts to parse a SQL UPDATE statement stored in the CommandText (see page 1554) property.

Published Properties

	Name	Description
	Active (see page 1687)	Specifies whether or not a dataset is open.
	GetMetadata (see page 1558)	Specifies whether the SQL dataset fetches metadata information along with the data.
	IndexFieldNames (see page 1689)	Lists the fields to use for sorting records and for linking in master/detail relationships.
	IndexName (see page 1690)	Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.
	MasterFields (see page 1691)	Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.
	MasterSource (see page 1692)	Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.
	MaxBlobSize (see page 1692)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	NumericMapping (see page 1560)	Maps fldBCD to either ftFMTBCD or ftBCD field types.
	ObjectView (see page 1560)	Specifies whether fields are stored hierarchically or flattened out in the Fields property.
	SchemaName (see page 1565)	Specifies the name of the database schema.
	SQLConnection (see page 1693)	Specifies the SQL connection component that connects the dataset to a database server.
	TableName (see page 1693)	Specifies the table on the database server represented by this component.

25.1.13.2 TSQLTable.Create Constructor

Creates an instance of a TSQLTable (see page 1680) component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSQLTable(TComponent * AOwner);
```

Description

Call Create to instantiate an SQL table at runtime. SQL tables placed in forms or data modules at design time are created automatically.

See Also

Create

25.1.13.3 TSQLTable.Destroy Destructor

Destroys the TSQLTable (see page 1680) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSQLTable();
```

Description

Usually destruction of datasets is handled automatically by Delphi. If an application creates its own instances of a dataset, however, and does not assign an Owner that is responsible for freeing the dataset, then the application should call Free, which verifies that the dataset reference is not nil before calling Destroy.

See Also

Owner

Create (see page 1684)

System::TObject::Free (see page 3372)

25.1.13.4 TSQLTable Methods

The methods of the TSQLTable class are listed here.

Protected Methods

	Name	Description
	OpenCursor (see page 1686)	This is OpenCursor, a member of class TSQLTable.
	SetIndexField (see page 1687)	This is SetIndexField, a member of class TSQLTable.

Public Methods

	Name	Description
	DeleteRecords (see page 1685)	Empties the underlying database table.
	GetIndexNames (see page 1686)	Retrieves a list of available indexes defined for the table.
	PrepareStatement (see page 1686)	Generates the query that TSQLTable (see page 1680) uses to fetch data from the database server.

25.1.13.4.1 TSQLTable.DeleteRecords Method

Empties the underlying database table.

Pascal

```
procedure DeleteRecords;
```

C++

```
__fastcall DeleteRecords();
```

Description

Call DeleteRecords to delete all the records in the underlying database table.

See Also

RecordCount

25.1.13.4.2 TSQLTable.GetIndexNames Method

Retrieves a list of available indexes defined for the table.

Pascal

```
procedure GetIndexNames(List: TWideStrings);
```

C++

```
__fastcall GetIndexNames(TWideStrings List);
```

Description

Call GetIndexNames to retrieve a list of all indexes on the server defined for the table. List is a string list object, created and maintained by the application, into which to retrieve the index names.

See Also

IndexName (see page 1690)

IndexDefs (see page 1558)

Obtaining Information About Indexes

25.1.13.4.3 TSQLTable.OpenCursor Method

Pascal

```
procedure OpenCursor(InfoQuery: Boolean); override;
```

C++

```
virtual __fastcall OpenCursor(Boolean InfoQuery);
```

Description

This is OpenCursor, a member of class TSQLTable.

25.1.13.4.4 TSQLTable.PrepareStatement Method

Generates the query that TSQLTable (see page 1680) uses to fetch data from the database server.

Pascal

```
procedure PrepareStatement; override;
```

C++

```
virtual __fastcall PrepareStatement();
```

Description

PrepareStatement generates a SELECT query to fetch the records from the table specified by TableName (see page 1693), ordering by the fields in IndexFieldNames (see page 1689) or the index in IndexName (see page 1690). If you have specified a master/detail relationship, PrepareStatement sets that up as well.

Do not call PrepareStatement in your applications. Instead, set the Prepared (see page 1563) property to true. When you use

the Prepared (see page 1563) property instead, the SQL table keeps track of the fact that it has already been prepared and need not repeat the process every time it executes.

See Also

Prepared (see page 1563)

TableName (see page 1693)

IndexName (see page 1690)

IndexFieldNames (see page 1689)

25.1.13.4.5 TSQLTable.SetIndexField Method

Pascal

```
procedure SetIndexField(Index: Integer; Value: TField);
```

C++

```
__fastcall SetIndexField(int Index, TField Value);
```

Description

This is SetIndexField, a member of class TSQLTable.

25.1.13.5 TSQLTable Properties

The properties of the TSQLTable class are listed here.

Protected Properties

	Name	Description
	MasterLink (see page 1692)	This is MasterLink, a member of class TSQLTable.

Public Properties

	Name	Description
	IndexFieldCount (see page 1689)	Indicates the number of fields that make up the current index.
	IndexFields (see page 1690)	Specifies the fields associated with the current index.

Published Properties

	Name	Description
	Active (see page 1687)	Specifies whether or not a dataset is open.
	IndexFieldNames (see page 1689)	Lists the fields to use for sorting records and for linking in master/detail relationships.
	IndexName (see page 1690)	Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.
	MasterFields (see page 1691)	Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.
	MasterSource (see page 1692)	Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.
	MaxBlobSize (see page 1692)	Indicates the maximum number of bytes fetched for any BLOB field in the dataset.
	SQLConnection (see page 1693)	Specifies the SQL connection component that connects the dataset to a database server.
	TableName (see page 1693)	Specifies the table on the database server represented by this component.

25.1.13.5.1 TSQLTable.Active Property

Specifies whether or not a dataset is open.

Pascal

```
property Active;
```

C++

```
__property Active;
```

Description

Use Active to determine or set whether a dataset is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true:

1. Generates a BeforeOpen event.
2. Sets the dataset state to dsBrowse.
3. Establishes a way to fetch data (typically by opening a cursor).
4. Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1. Triggers a BeforeClose event.
2. Sets the State property to dsInactive.
3. Closes the cursor.
4. Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

Delphi Examples:

```
{
This example requires a database with the following name or
in the form of a *.DB file in the following path.  The table
should be closed when changing the database.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Customers.Active := False;
  try
  { First try to use an alias }
    Customers.DatabaseName := 'DBDEMOS';
    Customers.Active := True;
  except
    on EDatabaseError do
    begin
      { If that fails, try to use the drive and directory }
      Customers.TableName := 'Flights';
      // This path depends on the Demos directory of your RAD Studio installation.
      Customers.DatabaseName :=
        'c:\Documents and Settings\All Users\Documents\RAD
Studio\6.0\Demos\IntraWeb\Win32\FlightInformation\Data';
      Customers.Active := True;
    end;
```

```
  end;
end;
```

See Also

DataSource

State

AfterOpen

BeforeOpen

Close

Open

CanModify

BeforeClose

AfterClose

Using Datasets with Decision Support Components

Making the Table a Detail of Another Dataset

Opening and Closing Datasets

Fetching the Data

25.1.13.5.2 TSQLTable.IndexFieldCount Property

Indicates the number of fields that make up the current index.

Pascal

```
property IndexFieldCount: Integer;
```

C++

```
__property int IndexFieldCount;
```

Description

Check IndexFieldCount to determine how many fields are in the current index. If the IndexName (see page 1690) property is not set, IndexFieldCount is 0.

See Also

IndexFieldNames (see page 1689)

IndexDefs (see page 1558)

IndexName (see page 1690)

25.1.13.5.3 TSQLTable.IndexFieldNames Property

Lists the fields to use for sorting records and for linking in master/detail relationships.

Pascal

```
property IndexFieldNames: UnicodeString;
```

C++

```
__property UnicodeString IndexFieldNames;
```

Description

IndexFieldNames is a semicolon-delimited set of field names, where each name specifies a field in the table.

Use IndexFieldNames to identify the fields in the SQL table that are used to sort records in the table. IndexFieldNames are used to generate the ORDER BY clause of the SQL statement automatically generated to fetch records for the dataset.

IndexFieldNames is also used to link this table as the detail in a master/detail relationship. Each of these fields must be matched to a corresponding field from the MasterFields (see page 1691) property.

Instead of explicitly setting the sorting and linking fields using IndexFieldNames, you can indicate them using an existing index. To specify the sorting and linking fields implicitly through an index, use the IndexName (see page 1690) property instead.

Note: The IndexName (see page 1690) and IndexFieldNames properties are mutually exclusive. Setting one automatically clears the other.

See Also

IndexName (see page 1690)

MasterFields (see page 1691)

MasterSource (see page 1692)

Creating an Index with IndexFieldNames

Representing the Records in a Table

25.1.13.5.4 TSQLTable.IndexFields Property

Specifies the fields associated with the current index.

Pascal

```
property IndexFields [Index: Integer]: TField;
```

C++

```
__property TField IndexFields[int Index];
```

Description

IndexFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

Note: Do not set IndexFields directly. Instead use the IndexFieldNames (see page 1689) property to specify the sort order for records in the table.

See Also

IndexDefs (see page 1558)

IndexFieldNames (see page 1689)

IndexName (see page 1690)

25.1.13.5.5 TSQLTable.IndexName Property

Identifies an index to use for sorting records and for linking the table as the detail of a master/detail relationship.

Pascal

```
property IndexName: UnicodeString;
```

C++

```
__property UnicodeString IndexName;
```

Description

Use IndexName to specify an existing index that is defined for the underlying database table.

IndexName is used to sort records in the table by appearing in the ORDER BY clause of the SQL statement automatically generated to fetch records for the dataset.

IndexName is also used to link this table as the detail in a master/detail relationship. Each of the fields in the index must be matched to a corresponding field from the MasterFields (see page 1691) property.

Note: The IndexName and IndexFieldNames (see page 1689) properties are mutually exclusive. Setting one automatically clears the other.

See Also

IndexFieldNames (see page 1689)

MasterFields (see page 1691)

MasterSource (see page 1692)

GetIndexNames (see page 1686)

Representing the Records in a Table

25.1.13.5.6 TSQLTable.MasterFields Property

Identifies the fields from the master dataset to use for linking this table as the detail of a master/detail relationship.

Pascal

```
property MasterFields: UnicodeString;
```

C++

```
__property UnicodeString MasterFields;
```

Description

Use MasterFields to identify the fields in another (master) dataset that can be used as link fields in a master/detail relationship. MasterFields is a semicolon-delimited set of field names, where each name identifies a field in the master dataset. The master dataset is the dataset associated with the datasource specified by MasterSource (see page 1692). Each of the fields in MasterFields must be matched to a corresponding field from the IndexFieldNames (see page 1689) property.

At design-time, double-click on the MasterFields property in the Object Inspector to use the Field Link designer to help you define the master/detail relationship. You can only use the Field Link designer if you have already specified the IndexName (see page 1690) or IndexFieldNames (see page 1689) property and the MasterSource (see page 1692) property.

See Also

IndexFieldNames (see page 1689)

IndexName (see page 1690)

MasterSource (see page 1692)

Making the Table a Detail of Another Dataset

Setting Up Master/detail Linked Cursors

25.1.13.5.7 TSQLTable.MasterLink Property

Pascal

```
property MasterLink: TMasterDataLink;
```

C++

```
__property TMasterDataLink MasterLink;
```

Description

This is MasterLink, a member of class TSQLTable.

25.1.13.5.8 TSQLTable.MasterSource Property

Identifies the master dataset to use when linking this table as the detail of a master/detail relationship.

Pascal

```
property MasterSource: TDataSource;
```

C++

```
__property TDataSource MasterSource;
```

Description

Use MasterSource to indicate the dataset that serves as a master table when establishing a master/detail relationship. Specify a TDataSource component whose DataSet property identifies the desired master dataset.

A master/detail relationship is established when you set the MasterSource property to identify a master dataset, the MasterFields (see page 1691) property to identify the fields in that master dataset whose values must be matched by linking fields in this (detail) SQL table, and the IndexName (see page 1690) or IndexFieldNames (see page 1689) property to identify the linking fields in this SQL table.

MasterSource is the same as the DataSource (see page 1557) property, except that it is published.

See Also

IndexFieldNames (see page 1689)

IndexName (see page 1690)

MasterFields (see page 1691)

DataSource (see page 1557)

DataSet

Making the Table a Detail of Another Dataset

Setting Up Master/detail Linked Cursors

25.1.13.5.9 TSQLTable.MaxBlobSize Property

Indicates the maximum number of bytes fetched for any BLOB field in the dataset.

Pascal

```
property MaxBlobSize: Integer;
```

C++

```
__property int MaxBlobSize;
```

Description

MaxBlobSize limits the amount of data that can be fetched from Binary Large Object (BLOB) fields. BLOB fields can hold an indefinite amount of data. However, SQL datasets read that data into a memory buffer. By limiting the amount of data that can be fetched from a BLOB field, you limit the size of the buffer.

If MaxBlobSize is –1, the dataset imposes no maximum BLOB size. If MaxBlobSize is 0, the dataset uses the parameters of the associated SQL connection, if any, to determine the maximum BLOB size.

Note: MaxBlobSize should agree with any parameters on the associated SQL connection component that limit the size of BLOB data.

See Also

GetBlobFieldData (see page 1530)

Params (see page 1621)

SQLConnection (see page 1566)

25.1.13.5.10 TSQLTable.SQLConnection Property

Specifies the SQL connection component that connects the dataset to a database server.

Pascal

```
property SQLConnection: TSQLConnection;
```

C++

```
__property TSQLConnection SQLConnection;
```

Description

Use SQLConnection to specify an SQL connection object that connects the dataset to a database server.

At design-time, select from the list of available TSQLConnection components in the Object Inspector. At runtime, set SQLConnection to reference an existing TSQLConnection object.

SQLConnection should be the first property you set on the dataset. Only after the SQLConnection property is set can you use the Command Text editor to specify a query for the CommandText property or populate the dataset with data by setting Active to true. Similarly, SQLConnection must be set before the dataset can initialize the Params property when you set CommandText.

See Also

Active

CommandText (see page 1554)

Params (see page 1562)

Using Query-type Datasets

Using Stored Procedure-type Datasets

Using Table Type Datasets

Connecting Directly to a Database Server

Connecting to the Database Server

25.1.13.5.11 TSQLTable.TableName Property

Specifies the table on the database server represented by this component.

Pascal

```
property TableName: UnicodeString;
```

C++

```
__property UnicodeString TableName;
```

Description

Set TableName to the name of a table on the database server. TSQLTable (see page 1680) automatically generates an SQL query to fetch all of the records and all of the fields in that table.

Note: In the automatically generated SQL query, the ORDER BY clause is determined by the IndexName (see page 1690) or IndexFieldNames (see page 1689) property.

See Also

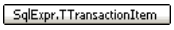
IndexFieldNames (see page 1689)

IndexName (see page 1690)

Representing the Records in a Table

25.1.14 TTransactionItem Class

Class Hierarchy



Pascal

```
TTransactionItem = class;
```

C++

```
class TTransactionItem;
```

File

SqlExpr

Description

This is class SqlExpr.TTransactionItem.

25.1.14.1 TTransactionItem Members

The following tables list the members exposed by TTransactionItem.

Public Fields

	Name	Description
	FNext (see page 1695)	This is FNext, a member of class TTransactionItem.
	FTransaction (see page 1695)	This is FTransaction, a member of class TTransactionItem.
	FTransactionDesc (see page 1695)	This is FTransactionDesc, a member of class TTransactionItem.

25.1.14.2 TTransactionItem Fields

The fields of the TTransactionItem class are listed here.

Public Fields

	Name	Description
	FNext (see page 1695)	This is FNext, a member of class TTransactionItem.
	FTransaction (see page 1695)	This is FTransaction, a member of class TTransactionItem.
	FTransactionDesc (see page 1695)	This is FTransactionDesc, a member of class TTransactionItem.

25.1.14.2.1 TTransactionItem.FNext Field

Pascal

```
FNext: TTransactionItem;
```

C++

```
TTransactionItem FNext;
```

Description

This is FNext, a member of class TTransactionItem.

25.1.14.2.2 TTransactionItem.FTransaction Field

Pascal

```
FTransaction: TDBXTransaction;
```

C++

```
TDBXTransaction FTransaction;
```

Description

This is FTransaction, a member of class TTransactionItem.

25.1.14.2.3 TTransactionItem.FTransactionDesc Field

Pascal

```
FTransactionDesc: TTransactionDesc;
```

C++

```
TTransactionDesc FTransactionDesc;
```

Description

This is FTransactionDesc, a member of class TTransactionItem.

25.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FreeProcParams (see page 1696)	Frees a list of procedure parameters that is allocated by GetProcedureParams.
	GetConnectionNames (see page 1696)	This is function SqlExpr.GetConnectionNames.
	GetConnectionRegistryFile (see page 1697)	This is function SqlExpr.GetConnectionRegistryFile.
	GetDelegateDriverNames (see page 1697)	This is function SqlExpr.GetDelegateDriverNames.
	GetDriverNames (see page 1697)	Populates a string list with the names of all available dbExpress drivers.
	GetDriverRegistryFile (see page 1698)	This is function SqlExpr.GetDriverRegistryFile.
	LoadParamListItems (see page 1698)	Fills a TParams object with information from a list of parameter descriptors.

25.2.1 SqlExpr.FreeProcParams Function

Frees a list of procedure parameters that is allocated by GetProcedureParams.

Pascal

```
procedure FreeProcParams(var ProcParams: TList);
```

C++

```
FreeProcParams(TList ProcParams);
```

File

SqlExpr

Description

Use FreeProcParams to free a list of stored procedure parameters after using them. FreeProcParams takes a single parameter, ProcParams, which is a list of parameter information that was returned from a call to the GetProcedureParams method of TSQLConnection (see page 1580). It frees all of the parameter structures in the list, and then frees the list itself, changing the parameter that was passed in to nil (Delphi) or NULL (C++).

See Also

TSQLConnection (see page 1580)

LoadParamListItems (see page 1698)

25.2.2 SqlExpr.GetConnectionNames Function

Pascal

```
procedure GetConnectionNames(List: TStrings; Driver: string = ''; DesignMode: Boolean = True);
```

C++

```
GetConnectionNames(TStrings List, UnicodeString Driver = '', Boolean DesignMode = True);
```

File

SqlExpr

Description

This is function SqlExpr.GetConnectionNames.

25.2.3 SqlExpr.GetConnectionRegistryFile Function

Pascal

```
function GetConnectionRegistryFile(DesignMode: Boolean = False): string;
```

C++

```
UnicodeString GetConnectionRegistryFile(Boolean DesignMode = False);
```

File

SqlExpr

Description

This is function SqlExpr.GetConnectionRegistryFile.

25.2.4 SqlExpr.GetDelegateDriverNames Function

Pascal

```
procedure GetDelegateDriverNames(List: TStrings; DesignMode: Boolean = True);
```

C++

```
GetDelegateDriverNames(TStrings List, Boolean DesignMode = True);
```

File

SqlExpr

Description

This is function SqlExpr.GetDelegateDriverNames.

25.2.5 SqlExpr.GetDriverNames Function

Populates a string list with the names of all available dbExpress drivers.

Pascal

```
procedure GetDriverNames(List: TStrings; DesignMode: Boolean = True);
```

C++

```
GetDriverNames(TStrings List, Boolean DesignMode = True);
```

File

SqlExpr

Description

GetDriverNames fills the string list specified by List with the names of all installed dbExpress drivers. GetDriverNames fetches the names of Installed drivers from the drivers.ini file.

List is a string list object, created and maintained by the application, into which to store the driver names.

DesignMode indicates whether the names are fetched at design time (true) or at runtime (false).

See Also

TSQLConnection (see page 1580)

25.2.6 SqlExpr.GetDriverRegistryFile Function

Pascal

```
function GetDriverRegistryFile(DesignMode: Boolean = False): string;
```

C++

```
UnicodeString GetDriverRegistryFile(Boolean DesignMode = False);
```

File

SqlExpr

Description

This is function SqlExpr.GetDriverRegistryFile.

25.2.7 SqlExpr.LoadParamListItems Function

Fills a TParams object with information from a list of parameter descriptors.

Pascal

```
procedure LoadParamListItems(Params: TParams; ProcParams: TList);
```

C++

```
LoadParamListItems(TParams Params, TList ProcParams);
```

File

SqlExpr

Description

Call LoadParamListItems to convert the list of parameter descriptors that is returned from the GetProcedureParams method of TSQLConnection (see page 1580) to a collection of TParam objects.

Params is an empty TParams collection that the procedure fills with information from the list of parameter descriptors.

ProcParams is the list of parameter descriptors that GetProcedureParams returns.

See Also

FreeProcParams (see page 1696)

TSQLConnection (see page 1580)

25.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	EConnectFlag (see page 1699)	This is record SqlExpr.EConnectFlag.
	TConnectionState (see page 1700)	TConnectionState describes the state of a TSQLConnection (see page 1580) component.
	TConnectionUserType (see page 1700)	This is record SqlExpr.TConnectionUserType.
	TSchemaType (see page 1701)	TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.
	TSQLExceptionType (see page 1702)	TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.
	TTableScope (see page 1703)	TTableScopes (see page 1710) and TTableScopess indicate the types of tables for which to fetch metadata.
	TTransIsolationLevel (see page 1704)	This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Records

	Name	Description
	TSQLSchemaInfo (see page 1703)	This is record SqlExpr.TSQLSchemaInfo.
	TTransactionDesc (see page 1704)	This is record SqlExpr.TTransactionDesc.

25.3.1 SqlExpr.EConnectFlag Enumeration

Pascal

```
EConnectFlag = (
  eConnect,
  eReconnect,
  eDisconnect
);
```

C++

```
enum EConnectFlag {
  eConnect,
  eReconnect,
  eDisconnect
};
```

File

SqlExpr

Description

This is record SqlExpr.EConnectFlag.

25.3.2 SqlExpr.TConnectionState Enumeration

TConnectionState describes the state of a TSQLConnection (see page 1580) component.

Pascal

```
TConnectionState = (
  csStateClosed,
  csStateOpen,
  csStateConnecting,
  csStateExecuting,
  csStateFetching,
  csStateDisconnecting
);
```

C++

```
enum TConnectionState {
  csStateClosed,
  csStateOpen,
  csStateConnecting,
  csStateExecuting,
  csStateFetching,
  csStateDisconnecting
};
```

File

SqlExpr

Description

TConnectionState can have any of the following values:

Value	Meaning
csStateClosed	The connection is closed. (The Connected property is false)
csStateOpen	The connection is open. (The Connected property is true)
csStateConnecting	The component is establishing a connection but has not completed the process.
csStateExecuting	The component has passed an SQL command to the server and it is currently executing.
csStateFetching	The component is fetching database information from the server.
csStateDisconnecting	The component is terminating a connection but has not completed the process.

25.3.3 SqlExpr.TConnectionUserType Enumeration

Pascal

```
TConnectionUserType = (
  eUserMonitor,
  eUserDataSet
);
```

C++

```
enum TConnectionUserType {
  eUserMonitor,
```

```
  eUserDataSet
};
```

File

SqlExpr

Description

This is record SqlExpr.TConnectionUserType.

25.3.4 SqlExpr.TSchemaType Enumeration

TSchemaType and TSchemaInfo describe schema information (metadata) to access using dbXpress.

Pascal

```
TSchemaType = (
  stNoSchema,
  stTables,
  stSysTables,
  stProcedures,
  stColumns,
  stProcedureParams,
  stIndexes,
  stPackages,
  stUserNames
);
```

C++

```
enum TSchemaType {
  stNoSchema,
  stTables,
  stSysTables,
  stProcedures,
  stColumns,
  stProcedureParams,
  stIndexes,
  stPackages,
  stUserNames
};
```

File

SqlExpr

Description

Use TSchemaInfo to describe the metadata you want to fetch from a database server when you are using an SQL dataset. TSchemaInfo has three members: FType, ObjectName, and Pattern, which are described below.

Use TSchemaInfo to describe the metadata you want to fetch from a database server when you are using an SQL dataset. TSchemaInfo has four members: FType, ObjectName, Pattern, and PackageName, which are described below.

FType is of type TSchemaType. It identifies the type of information you want. It can have any of the following values:

Value	Description
stNoSchema	No schema information. When FType is stNoSchema, the SQL dataset is populated with the results of its query or stored procedure rather than metadata from the server.
stTables	Information about all the data tables on the database server that match the criteria specified by the SQL connection's TableScope property.

stSysTables	Information about all of the system tables on the database server. Not all servers use system tables to store metadata. Requesting a list of system tables from a server that does not use them results in an empty dataset.
stProcedures	Information about all of the stored procedures on the database server.
stColumns	Information about all of the columns (fields) in a specified table.
stProcedureParams	Information about all of the parameters of a specified stored procedure.
stIndexes	Information about all of the indexes defined for a specified table.
stPackages	Information about all of the packages defined on the database server (Oracle only).

ObjectName is the name of the table or stored procedure about whose columns, indexes, or parameters you want information. When FType is stColumns or stIndexes, ObjectName is the name of a table. When FType is stProcedureParams, ObjectName is the name of a stored procedure. When FType is stNone, stTables, stSysTables, or stProcedures, ObjectName is ignored.

Pattern is an SQL pattern mask that limits the items returned. The name of each item (table, stored procedure, field, parameter, or index) must match this pattern mask or it does not appear as a record when the SQL dataset is opened. Pattern uses the wildcards '%' (to match a string of arbitrary characters of any length) and '_' (to match a single arbitrary character). To use a literal percent or underscore in a pattern, the character is doubled (%% or __). If you do not want to use a pattern, set Pattern to nil (Delphi) or NULL (C++).

PackageName is the name of an Oracle package that contains the stored procedure about which you are requesting information. This member is used when FType is stProcedures or stProcedureParams. PackageName is ignored if the server is not an Oracle database.

25.3.5 SqlExpr.TSQLExceptionType Enumeration

TSQLExceptionType indicates the type of operation that leads to an ESQLException exception.

Pascal

```
TSQLExceptionType = (
  exceptConnection,
  exceptCommand,
  exceptCursor,
  exceptMetaData,
  exceptUseLast
);
```

C++

```
enum TSQLExceptionType {
  exceptConnection,
  exceptCommand,
  exceptCursor,
  exceptMetaData,
  exceptUseLast
};
```

File

SqlExpr

Description

ESQLException objects use TSQLExceptionType to identify the dbExpress object from which an error code originated. By identifying the underlying dbExpress object, ESQLException identifies that object's interface, which it then uses to obtain an error message.

TSQLExceptionType includes the following values:

Value	Description
exceptConnection	Error (see page 3560) arose when establishing a connection to the database server, obtaining an interface for sending a command, fetching server metadata, or controlling a transaction.
exceptCommand	Error (see page 3560) occurred while preparing or executing an SQL command.
exceptCursor	Error (see page 3560) occurred while fetching data from a dbExpress cursor.
exceptMetaData	Error (see page 3560) occurred while trying to fetch metadata (schema information).
exceptUseLast	The error is cached by the TSQLConnection (see page 1580) component.

25.3.6 SqlExpr.TSQLSchemaInfo Record

Pascal

```
TSQLSchemaInfo = record
  FType: TSchemaType;
  ObjectName: UnicodeString;
  Pattern: UnicodeString;
  PackageName: UnicodeString;
end;
```

C++

```
struct TSQLSchemaInfo {
  TSchemaType FType;
  UnicodeString ObjectName;
  UnicodeString Pattern;
  UnicodeString PackageName;
};
```

File

SqlExpr

Description

This is record SqlExpr.TSQLSchemaInfo.

25.3.7 SqlExpr.TTableScope Enumeration

TTableScopes (see page 1710) and TTableScopess indicate the types of tables for which to fetch metadata.

Pascal

```
TTableScope = (
  tsSynonym,
  tsSysTable,
  tsTable,
  tsView
);
```

C++

```
enum TTableScope {
  tsSynonym,
  tsSysTable,
```

```
  tsTable,
  tsView
};
```

File

SqlExpr

Description

TTableScopess is a set that indicates the types of tables for which to fetch metadata. It is a set of TTableScopes (see page 1710) values. The following table describes the possible values:

Value	Meaning
tsSynonym	synonyms
tsSysTable	system tables
tsTable	ordinary tables
tsView	views

25.3.8 SqlExpr.TTransactionDesc Record

Pascal

```
TTransactionDesc = packed record
  TransactionID: LongWord;
  GlobalID: LongWord;
  IsolationLevel: TTransIsolationLevel;
  CustomIsolation: LongWord;
end;
```

C++

```
struct TTransactionDesc {
  LongWord TransactionID;
  LongWord GlobalID;
  TTransIsolationLevel IsolationLevel;
  LongWord CustomIsolation;
};
```

File

SqlExpr

Description

This is record SqlExpr.TTransactionDesc.

25.3.9 SqlExpr.TTransIsolationLevel Enumeration

This enumerated type (in C++) or type (in Delphi) identifies degree to which transactions are isolated from each other.

Pascal

```
TTransIsolationLevel = (
  xilREADCOMMITTED,
  xilREPEATABLEREAD,
  xilDIRTYREAD,
```

```
    xilCUSTOM
  );
```

C++

```
  enum TTransIsolationLevel {
    xilREADCOMMITTED,
    xilREPEATABLEREAD,
    xilDIRTYREAD,
    xilCUSTOM
  };
```

File

SqlExpr

Description

The following table describes the possible values.

Element	Description
xilDIRTYREAD	Current transaction sees all changes made by other transactions, even if they have not yet been committed. This option is not available for the Oracle driver.
xilREADCOMMITTED	Current transaction sees only changes that have been committed, but can receive an inconsistent view of the data if additional changes are committed before the transaction ends.
xilREPEATABLEREAD	Current transaction is guaranteed a consistent view of the data, which includes only changes committed by other transactions at the start of the transaction.
xilCUSTOM	Current transaction uses a database-specific isolation level. When the eConnTxnIsoLevel option is xilCUSTOM, the transaction descriptor gives the actual isolation level. This is not currently supported by any drivers, but allows future drivers the ability to expand the number of isolation levels.

25.4 Types

The following table lists types in this documentation.

Types

Name	Description
SQLResult (see page 1706)	SQLResult is the return type for dbExpress function calls.
TFieldDescList (see page 1707)	This is type SqlExpr.TFieldDescList.
TFieldList (see page 1707)	This is type SqlExpr.TFieldList.
TGetDriverFunc (see page 1708)	This is type SqlExpr.TGetDriverFunc.
TLocale (see page 1708)	This is type SqlExpr.TLocale.
TLocaleCode (see page 1708)	TLocaleCode identifies a locale, which imposes a sort order on data.
TParseInsertSqlEvent (see page 1709)	This is type SqlExpr.TParseInsertSqlEvent.
TParseSqlEvent (see page 1709)	This is type SqlExpr.TParseSqlEvent.
TSQLConnectionLoginEvent (see page 1709)	This is type SqlExpr.TSQLConnectionLoginEvent.
TTableScopes (see page 1710)	This is type SqlExpr.TTableScopes.
TTraceEvent (see page 1710)	This is type SqlExpr.TTraceEvent.
TTraceLogEvent (see page 1710)	This is type SqlExpr.TTraceLogEvent.

25.4.1 SqlExpr.SQLResult Type

SQLResult is the return type for dbExpress function calls.

Pascal

```
SQLResult = TDBXErrorCode;
```

C++

```
TDBXErrorCode SQLResult;
```

File

SqlExpr

Description

SQLResult indicates the success or failure of a dbExpress function. The following table lists the error codes defined for dbExpress.

SQLResult Value	Meaning
SQL_SUCCESS	The function call was successful.
DBXERR_NONE	Same as SQL_SUCCESS.
DBXERR_NOMEMORY	Insufficient memory.
DBXERR_INVALIDFLDTYPE	Invalid field type.
DBXERR_INVALIDHNDL	Invalid handle.
DBXERR_INVALIDTIME	Invalid time value.
DBXERR_NOTSUPPORTED	Feature not supported.
DBXERR_INVALIDXLATION	Invalid translation.

DBXERR_INVALIDPARAM	Invalid parameter.
DBXERR_OUTOFRANGE	Value out of range.
DBXERR_SQLPARAMNOTSET	Parameter not set.
DBXERR_EOF	Unexpected end-of-file.
DBXERR_INVALIDUSRPASS	Invalid user password.
DBXERR_INVALIDPRECISION	Invalid precision on a BCD value.
DBXERR_INVALIDLEN	Invalid length.
DBXERR_INVALIDTXNISOLEVEL	Invalid transaction isolation level.
DBXERR_INVALIDTXNID	Invalid transaction id.
DBXERR_DUPLICATETXNID	Duplicate transaction id.
DBXERR_DRIVERRESTRICTED	Driver restricted.
DBXERR_LOCALTRANSACTIVE	An attempt to start multiple local transactions (only 1 allowed).

25.4.2 SqlExpr.TFieldDescList Type

Pascal

```
TFieldDescList = array of TFLDDesc;
```

C++

```
array of TFLDDesc TFieldDescList;
```

File

SqlExpr

Description

This is type SqlExpr.TFieldDescList.

25.4.3 SqlExpr.TFieldList Type

Pascal

```
TFieldList = TList;
```

C++

```
TList TFieldList;
```

File

SqlExpr

Description

This is type SqlExpr.TFieldList.

25.4.4 SqlExpr.TGetDriverFunc Type

Pascal

```
TGetDriverFunc = function (SVendorLib, SResourceFile: PChar; out Obj): TDBXErrorCode;
```

C++

```
(SVendorLib, SResourceFile: PChar; out Obj): TDBXErrorCode ( TGetDriverFunc)();
```

File

SqlExpr

Description

This is type SqlExpr.TGetDriverFunc.

25.4.5 SqlExpr.TLocale Type

Pascal

```
TLocale = Pointer;
```

C++

```
void * TLocale;
```

File

SqlExpr

Description

This is type SqlExpr.TLocale.

25.4.6 SqlExpr.TLocaleCode Type

TLocaleCode identifies a locale, which imposes a sort order on data.

Pascal

```
TLocaleCode = Integer;
```

C++

```
int TLocaleCode;
```

File

SqlExpr

Description

TLocaleCode is an integer that corresponds to a locale. A locale identifies the language of a string, and character set. The TLocaleCode type corresponds to the Windows LCID, but omits an explicit sort order.

25.4.7 SqlExpr.TParseInsertSqlEvent Type

Pascal

```
TParseInsertSqlEvent = procedure (var FieldNames: TWideStrings; SQL: UnicodeString; var
BindAllFields: Boolean; var TableName: UnicodeString) of object;
```

C++

```
(var FieldNames: TWideStrings; SQL: UnicodeString; var BindAllFields: Boolean; var TableName:
UnicodeString) ( TParseInsertSqlEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TParseInsertSqlEvent.

25.4.8 SqlExpr.TParseSqlEvent Type

Pascal

```
TParseSqlEvent = procedure (var FieldNames: TWideStrings; SQL: UnicodeString; var TableName:
UnicodeString) of object;
```

C++

```
(var FieldNames: TWideStrings; SQL: UnicodeString; var TableName: UnicodeString) (
TParseSqlEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TParseSqlEvent.

25.4.9 SqlExpr.TSQLConnectionLoginEvent Type

Pascal

```
TSQLConnectionLoginEvent = procedure (Database: TSQLConnection; LoginParams: TWideStrings) of
object;
```

C++

```
(Database: TSQLConnection; LoginParams: TWideStrings) ( TSQLConnectionLoginEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TSQLConnectionLoginEvent.

25.4.10 SqlExpr.TTableScopes Type

Pascal

```
TTableScopes = set of TTableScope;
```

C++

```
TTableScope TTableScopes;
```

File

SqlExpr

Description

This is type SqlExpr.TTableScopes.

25.4.11 SqlExpr.TTraceEvent Type

Pascal

```
TTraceEvent = procedure (Sender: TObject; TraceInfo: TDBXTraceInfo; var LogTrace: Boolean) of
object;
```

C++

```
(Sender: TObject; TraceInfo: TDBXTraceInfo; var LogTrace: Boolean) ( TTraceEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceEvent.

25.4.12 SqlExpr.TTraceLogEvent Type

Pascal

```
TTraceLogEvent = procedure (Sender: TObject; TraceInfo: TDBXTraceInfo) of object;
```

C++

```
(Sender: TObject; TraceInfo: TDBXTraceInfo) ( TTraceLogEvent)();
```

File

SqlExpr

Description

This is type SqlExpr.TTraceLogEvent.

25.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
BlobTypeMap (see page 1711)	This is constant SqlExpr.BlobTypeMap.
DataTypeMap (see page 1712)	This is constant SqlExpr.DataTypeMap.
DefaultCursor (see page 1712)	This is constant SqlExpr.DefaultCursor.
DefaultMaxBlobSize (see page 1713)	This is constant SqlExpr.DefaultMaxBlobSize.
DefaultRowsetSize (see page 1713)	DefaultRowsetSize represents the default row buffering size.
eSQLAutoIncr (see page 1713)	This is constant SqlExpr.eSQLAutoIncr.
eSQLDefault (see page 1714)	This is constant SqlExpr.eSQLDefault.
eSQLFunction (see page 1714)	This is constant SqlExpr.eSQLFunction.
eSQLLocal (see page 1714)	This is constant SqlExpr.eSQLLocal.
eSQLNonUnique (see page 1715)	This is constant SqlExpr.eSQLNonUnique.
eSQLPackage (see page 1715)	This is constant SqlExpr.eSQLPackage.
eSQLPrimaryKey (see page 1715)	This is constant SqlExpr.eSQLPrimaryKey.
eSQLProcedure (see page 1716)	This is constant SqlExpr.eSQLProcedure.
eSQLRowId (see page 1716)	This is constant SqlExpr.eSQLRowId.
eSQLRowVersion (see page 1716)	This is constant SqlExpr.eSQLRowVersion.
eSQLSynonym (see page 1717)	This is constant SqlExpr.eSQLSynonym.
eSQLSysProcedure (see page 1717)	This is constant SqlExpr.eSQLSysProcedure.
eSQLSystemTable (see page 1717)	This is constant SqlExpr.eSQLSystemTable.
eSQLTable (see page 1718)	This is constant SqlExpr.eSQLTable.
eSQLTempTable (see page 1718)	This is constant SqlExpr.eSQLTempTable.
eSQLUnique (see page 1718)	This is constant SqlExpr.eSQLUnique.
eSQLView (see page 1719)	This is constant SqlExpr.eSQLView.
FldSubTypeMap (see page 1719)	This is constant SqlExpr.FldSubTypeMap.
FldTypeMap (see page 1719)	This is constant SqlExpr.FldTypeMap.
HourGlassCursor (see page 1720)	This is constant SqlExpr.HourGlassCursor.
SAnd (see page 1721)	This is constant SqlExpr.SAnd.
SOrderBy (see page 1721)	This is constant SqlExpr.SOrderBy.
SParam (see page 1721)	This is constant SqlExpr.SParam.
SSelect (see page 1722)	This is constant SqlExpr.SSelect.
SSelectSpaces (see page 1722)	This is constant SqlExpr.SSelectSpaces.
SSelectStar (see page 1722)	This is constant SqlExpr.SSelectStar.
SSelectStarFrom (see page 1723)	This is constant SqlExpr.SSelectStarFrom.
SUB_TYPE_MEMO (see page 1723)	This is constant SqlExpr.SUB_TYPE_MEMO.
SWhere (see page 1723)	This is constant SqlExpr.SWhere.
TErrorMessageSize (see page 1724)	This is constant SqlExpr.TErrorMessageSize.

25.5.1 SqlExpr.BlobTypeMap Constant

Pascal

```
BlobTypeMap: array[SUB_TYPE_MEMO..TDBXDataTypes.BFileSubType] of TFieldType = ( ftMemo,
ftBlob, ftFmtMemo, ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob,
ftWideMemo, ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

C++

```
array[SUB_TYPE_MEMO..TDBXDataTypes.BFileSubType] of TFieldType BlobTypeMap = ( ftMemo, ftBlob,
ftFmtMemo, ftParadoxOle, ftGraphic, ftDBaseOle, ftTypedBinary, ftBlob, ftBlob, ftBlob,
ftWideMemo, ftOraClob, ftOraBlob, ftBlob, ftBlob);
```

File

SqlExpr

Description

This is constant SqlExpr.BlobTypeMap.

25.5.2 SqlExpr.DataTypeMap Constant

Pascal

```
DataTypeMap: array[0..TDBXDataTypes.MaxBaseTypes - 1] of TFieldType = ( ftUnknown, ftString,
ftDate, ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftFMTBCD, ftBytes, ftTime,
ftDateTime, ftWord, ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftCursor, ftLargeInt,
ftLargeInt, ftADT, ftArray, ftReference, ftDataSet, ftTimeStamp, ftBCD, ftWideString);
```

C++

```
array[0..TDBXDataTypes.MaxBaseTypes - 1] of TFieldType DataTypeMap = ( ftUnknown, ftString,
ftDate, ftBlob, ftBoolean, ftSmallint, ftInteger, ftFloat, ftFMTBCD, ftBytes, ftTime,
ftDateTime, ftWord, ftInteger, ftUnknown, ftVarBytes, ftUnknown, ftCursor, ftLargeInt,
ftLargeInt, ftADT, ftArray, ftReference, ftDataSet, ftTimeStamp, ftBCD, ftWideString);
```

File

SqlExpr

Description

This is constant SqlExpr.DataTypeMap.

25.5.3 SqlExpr.DefaultCursor Constant

Pascal

```
DefaultCursor = 0;
```

C++

```
DefaultCursor = 0;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultCursor.

25.5.4 SqlExpr.DefaultMaxBlobSize Constant

Pascal

```
DefaultMaxBlobSize = -1;
```

C++

```
DefaultMaxBlobSize = -1;
```

File

SqlExpr

Description

This is constant SqlExpr.DefaultMaxBlobSize.

25.5.5 SqlExpr.DefaultRowsetSize Constant

DefaultRowsetSize represents the default row buffering size.

Pascal

```
DefaultRowsetSize = DBXDefaultRowSetSize;
```

C++

```
DefaultRowsetSize = DBXDefaultRowSetSize;
```

File

SqlExpr

Description

The value of the DefaultRowsetSize constant is 20. To use a different RowSetSize for existing Oracle database connections, manually add the RowSetSize property (for example, RowSetSize = 200) to dbxconnections.ini (Windows) or dbxconnections.conf (Linux).

See Also

DBXDefaultRowSetSize

RowSetSize

25.5.6 SqlExpr.eSQLAutoIncr Constant

Pascal

```
eSQLAutoIncr = $0004;
```

C++

```
eSQLAutoIncr = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLAutoIncr.

25.5.7 SqlExpr.eSQLDefault Constant

Pascal

```
eSQLDefault = $0008;
```

C++

```
eSQLDefault = 0x0008;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLDefault.

25.5.8 SqlExpr.eSQLFunction Constant

Pascal

```
eSQLFunction = $0002;
```

C++

```
eSQLFunction = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLFunction.

25.5.9 SqlExpr.eSQLLocal Constant

Pascal

```
eSQLLocal = $0020;
```

C++

```
eSQLLocal = 0x0020;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLLocal.

25.5.10 SqlExpr.eSQLNonUnique Constant

Pascal

```
eSQLNonUnique = $0001;
```

C++

```
eSQLNonUnique = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLNonUnique.

25.5.11 SqlExpr.eSQLPackage Constant

Pascal

```
eSQLPackage = $0004;
```

C++

```
eSQLPackage = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLPackage.

25.5.12 SqlExpr.eSQLPrimaryKey Constant

Pascal

```
eSQLPrimaryKey = $0004;
```

C++

```
eSQLPrimaryKey = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLPrimaryKey.

25.5.13 SqlExpr.eSQLProcedure Constant

Pascal

```
eSQLProcedure = $0001;
```

C++

```
eSQLProcedure = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLProcedure.

25.5.14 SqlExpr.eSQLRowId Constant

Pascal

```
eSQLRowId = $0001;
```

C++

```
eSQLRowId = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLRowId.

25.5.15 SqlExpr.eSQLRowVersion Constant

Pascal

```
eSQLRowVersion = $0002;
```

C++

```
eSQLRowVersion = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLRowVersion.

25.5.16 SqlExpr.eSQLSynonym Constant

Pascal

```
eSQLSynonym = $0008;
```

C++

```
eSQLSynonym = 0x0008;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLSynonym.

25.5.17 SqlExpr.eSQLSysProcedure Constant

Pascal

```
eSQLSysProcedure = $0008;
```

C++

```
eSQLSysProcedure = 0x0008;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLSysProcedure.

25.5.18 SqlExpr.eSQLSystemTable Constant

Pascal

```
eSQLSystemTable = $0004;
```

C++

```
eSQLSystemTable = 0x0004;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLSystemTable.

25.5.19 SqlExpr.eSQLTable Constant

Pascal

```
eSQLTable = $0001;
```

C++

```
eSQLTable = 0x0001;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLTable.

25.5.20 SqlExpr.eSQLTempTable Constant

Pascal

```
eSQLTempTable = $0010;
```

C++

```
eSQLTempTable = 0x0010;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLTempTable.

25.5.21 SqlExpr.eSQLUnique Constant

Pascal

```
eSQLUnique = $0002;
```

C++

```
eSQLUnique = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLUnique.

25.5.22 SqlExpr.eSQLView Constant

Pascal

```
eSQLView = $0002;
```

C++

```
eSQLView = 0x0002;
```

File

SqlExpr

Description

This is constant SqlExpr.eSQLView.

25.5.23 SqlExpr.FldSubTypeMap Constant

Pascal

```
FldSubTypeMap: array[TFieldType] of Word = ( 0, 0, 0, 0, 0, 0, 0, TDBXDataTypes.MoneySubType,
0, 0, 0, 0, 0, 0, TDBXDataTypes.AutoIncSubType, TDBXDataTypes.BinarySubType,
TDBXDataTypes.MemoSubType, 0, 0, 0, 0, 0, 0, TDBXDataTypes.FixedSubType, 0, 0, 0, 0, 0, 0,
TDBXDataTypes.HBinarySubType, TDBXDataTypes.HMemoSubType, 0, 0, 0, 0, 0, 0,
TDBXDataTypes.FixedSubType, TDBXDataTypes.WideMemoSubType,
TDBXDataTypes.OracleTimeStampSubType, TDBXDataTypes.OracleIntervalSubType, 0, 0, 0, 0, 0, 0,
0);
```

C++

```
array[TFieldType] of Word FldSubTypeMap = ( 0, 0, 0, 0, 0, 0, 0, TDBXDataTypes.MoneySubType,
0, 0, 0, 0, 0, 0, TDBXDataTypes.AutoIncSubType, TDBXDataTypes.BinarySubType,
TDBXDataTypes.MemoSubType, 0, 0, 0, 0, 0, 0, TDBXDataTypes.FixedSubType, 0, 0, 0, 0, 0, 0,
TDBXDataTypes.HBinarySubType, TDBXDataTypes.HMemoSubType, 0, 0, 0, 0, 0, 0,
TDBXDataTypes.FixedSubType, TDBXDataTypes.WideMemoSubType,
TDBXDataTypes.OracleTimeStampSubType, TDBXDataTypes.OracleIntervalSubType, 0, 0, 0, 0, 0, 0,
0);
```

File

SqlExpr

Description

This is constant SqlExpr.FldSubTypeMap.

25.5.24 SqlExpr.FldTypeMap Constant

Pascal

```
FldTypeMap: TFieldMap = ( TDBXDataTypes.UnknownType, TDBXDataTypes.AnsiStringType,
TDBXDataTypes.Int16Type, TDBXDataTypes.Int32Type, TDBXDataTypes.UInt16Type,
TDBXDataTypes.BooleanType, TDBXDataTypes.DoubleType, TDBXDataTypes.DoubleType,
TDBXDataTypes.CurrencyType, TDBXDataTypes.DateType, TDBXDataTypes.TimeType,
TDBXDataTypes.TimeStampType, TDBXDataTypes.BytesType, TDBXDataTypes.VarBytesType,
```

```
  TDBXDataTypes.Int32Type, TDBXDataTypes.BlobType, TDBXDataTypes.BlobType,
  TDBXDataTypes.BlobType, TDBXDataTypes.BlobType, TDBXDataTypes.BlobType,
  TDBXDataTypes.BlobType, TDBXDataTypes.BlobType, TDBXDataTypes.CursorType,
  TDBXDataTypes.AnsiStringType, TDBXDataTypes.WideStringType, TDBXDataTypes.Int64Type,
  TDBXDataTypes.AdtType, TDBXDataTypes.ArrayType, TDBXDataTypes.RefType,
  TDBXDataTypes.TableType, TDBXDataTypes.BlobType, TDBXDataTypes.BlobType,
  TDBXDataTypes.VariantType, TDBXDataTypes.UnknownType, TDBXDataTypes.UnknownType,
  TDBXDataTypes.AnsiStringType, TDBXDataTypes.TimeStampType, TDBXDataTypes.BcdType,
  TDBXDataTypes.WideStringType, TDBXDataTypes.BlobType, TDBXDataTypes.TimeStampType,
  TDBXDataTypes.AnsiStringType, TDBXDataTypes.UnknownType, TDBXDataTypes.UnknownType,
  TDBXDataTypes.UnknownType, TDBXDataTypes.UnknownType, TDBXDataTypes.DBXConnectionType,
  TDBXDataTypes.TableType, TDBXDataTypes.BinaryBlobType);
```

C++

```
  TFieldMap FldTypeMap = ( TDBXDataTypes.UnknownType, TDBXDataTypes.AnsiStringType,
  TDBXDataTypes.Int16Type, TDBXDataTypes.Int32Type, TDBXDataTypes.UInt16Type,
  TDBXDataTypes.BooleanType, TDBXDataTypes.DoubleType, TDBXDataTypes.DoubleType,
  TDBXDataTypes.CurrencyType, TDBXDataTypes.DateType, TDBXDataTypes.TimeType,
  TDBXDataTypes.TimeStampType, TDBXDataTypes.BytesType, TDBXDataTypes.VarBytesType,
  TDBXDataTypes.Int32Type, TDBXDataTypes.BlobType, TDBXDataTypes.BlobType,
  TDBXDataTypes.BlobType, TDBXDataTypes.BlobType, TDBXDataTypes.BlobType,
  TDBXDataTypes.BlobType, TDBXDataTypes.BlobType, TDBXDataTypes.CursorType,
  TDBXDataTypes.AnsiStringType, TDBXDataTypes.WideStringType, TDBXDataTypes.Int64Type,
  TDBXDataTypes.AdtType, TDBXDataTypes.ArrayType, TDBXDataTypes.RefType,
  TDBXDataTypes.TableType, TDBXDataTypes.BlobType, TDBXDataTypes.BlobType,
  TDBXDataTypes.VariantType, TDBXDataTypes.UnknownType, TDBXDataTypes.UnknownType,
  TDBXDataTypes.AnsiStringType, TDBXDataTypes.TimeStampType, TDBXDataTypes.BcdType,
  TDBXDataTypes.WideStringType, TDBXDataTypes.BlobType, TDBXDataTypes.TimeStampType,
  TDBXDataTypes.AnsiStringType, TDBXDataTypes.UnknownType, TDBXDataTypes.UnknownType,
  TDBXDataTypes.UnknownType, TDBXDataTypes.UnknownType, TDBXDataTypes.DBXConnectionType,
  TDBXDataTypes.TableType, TDBXDataTypes.BinaryBlobType);
```

File

SqlExpr

Description

This is constant SqlExpr.FldTypeMap.

25.5.25 SqlExpr.HourGlassCursor Constant

Pascal

```
  HourGlassCursor = -11;
```

C++

```
  HourGlassCursor = -11;
```

File

SqlExpr

Description

This is constant SqlExpr.HourGlassCursor.

25.5.26 SqlExpr.SAnd Constant

Pascal

```
SAnd = ' and ';
```

C++

```
SAnd = ' and ';
```

File

SqlExpr

Description

This is constant SqlExpr.SAnd.

25.5.27 SqlExpr.SOrderBy Constant

Pascal

```
SOrderBy = ' order by ';
```

C++

```
SOrderBy = ' order by ';
```

File

SqlExpr

Description

This is constant SqlExpr.SOrderBy.

25.5.28 SqlExpr.SParam Constant

Pascal

```
SParam = '?';
```

C++

```
SParam = '?';
```

File

SqlExpr

Description

This is constant SqlExpr.SParam.

25.5.29 SqlExpr.SSelect Constant

Pascal

```
SSelect = 'select';
```

C++

```
SSelect = 'select';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelect.

25.5.30 SqlExpr.SSelectSpaces Constant

Pascal

```
SSelectSpaces = ' select ';
```

C++

```
SSelectSpaces = ' select ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectSpaces.

25.5.31 SqlExpr.SSelectStar Constant

Pascal

```
SSelectStar = ' select * ';
```

C++

```
SSelectStar = ' select * ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectStar.

25.5.32 SqlExpr.SSelectStarFrom Constant

Pascal

```
SSelectStarFrom = ' select * from ';
```

C++

```
SSelectStarFrom = ' select * from ';
```

File

SqlExpr

Description

This is constant SqlExpr.SSelectStarFrom.

25.5.33 SqlExpr.SUB_TYPE_MEMO Constant

Pascal

```
SUB_TYPE_MEMO = TDBXDataTypes.MemoSubType;
```

C++

```
SUB_TYPE_MEMO = TDBXDataTypes.MemoSubType;
```

File

SqlExpr

Description

This is constant SqlExpr.SUB_TYPE_MEMO.

25.5.34 SqlExpr.SWhere Constant

Pascal

```
SWhere = ' where ';
```

C++

```
SWhere = ' where ';
```

File

SqlExpr

Description

This is constant SqlExpr.SWhere.

25.5.35 SqlExpr.TErrorMessageSize Constant

Pascal

```
TErrorMessageSize = 2048;
```

C++

```
TErrorMessageSize = 2048;
```

File

SqlExpr

Description

This is constant SqlExpr.TErrorMessageSize.

26 SqlTimSt Namespace

This is namespace SqlTimSt.

Topics

Name	Description
Classes (see page 1727)	The following table lists classes in this documentation.
Functions (see page 1729)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 1737)	The following table lists structs, records, enums in this documentation.
Types (see page 1739)	The following table lists types in this documentation.
Constants (see page 1740)	The following table lists constants in this documentation.

Classes

	Name	Description
	TTimeZone (see page 1727)	This is class SqlTimSt.TTimeZone.

Constants

Name	Description
NullSQLTimeStamp (see page 1740)	Represents a NULL TSQLTimeStamp (see page 1737) value.

Functions

	Name	Description
	CheckSqlTimeStamp (see page 1729)	Checks whether a TSQLTimeStamp (see page 1737) value represents a valid date and time.
	DateTimeToSQLTimeStamp (see page 1730)	Converts a TDateTime (see page 3323) value to a TSQLTimeStamp (see page 1737) value.
	LocalToUTC (see page 1730)	This is function SqlTimSt.LocalToUTC.
	SQLDayOfWeek (see page 1731)	Indicates the day of the week when a specified TSQLTimeStamp (see page 1737) value occurs.
	SQLTimeStampToDateTime (see page 1731)	Converts a TSQLTimeStamp (see page 1737) value to a TDateTime (see page 3323) value.
	SQLTimeStampToStr (see page 1732)	Converts a TSQLTimeStamp (see page 1737) value to a string.
	StrToSQLTimeStamp (see page 1732)	Converts a string to a TSQLTimeStamp (see page 1737) value.
	TryStrToSQLTimeStamp (see page 1733)	Converts a string to a TSQLTimeStamp (see page 1737) value.
	UTCToLocal (see page 1733)	This is function SqlTimSt.UTCToLocal.
	VarIsSQLTimeStamp (see page 1734)	Indicates whether a specified Variant (see page 3421) represents a TSQLTimeStamp (see page 1737) value.

	VarSQLTimeStamp (see page 1734)	Returns the TVarType (see page 3801) constant that identifies the internal type of a Variant (see page 3421) as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarToSQLTimeStamp (see page 1736)	Converts a Variant (see page 3421) that represents a TSQLTimeStamp (see page 1737) value to the corresponding TSQLTimeStamp (see page 1737).

Structs, Records, Enums

	Name	Description
	TSQLTimeStamp (see page 1737)	TSQLTimeStamp represents time and date values with a high degree of accuracy.

Types

Name	Description
PSQLTimeStamp (see page 1739)	This is type SqlTimSt.PSQLTimeStamp.

26.1 Classes

The following table lists classes in this documentation.

Records

	Name	Description
	TTimeZone (see page 1727)	This is class SqlTimSt.TTimeZone.

26.1.1 TTimeZone Record

Pascal

```
TTimeZone = record
  FUsesDayligtSavings: Boolean;
  FInfo: TTimeZoneInformation;
end;
```

C++

```
struct TTimeZone {
  Boolean FUsesDayligtSavings;
  TTimeZoneInformation FInfo;
};
```

File

SqlTimSt

Description

This is class SqlTimSt.TTimeZone.

26.1.1.1 TTimeZone Members

The following tables list the members exposed by TTimeZone.

Public Fields

	Name	Description
	FInfo (see page 1728)	This is FInfo, a member of class TTimeZone.
	FUsesDayligtSavings (see page 1728)	This is FUsesDayligtSavings, a member of class TTimeZone.

Public Methods

	Name	Description
	GetTimeZone (see page 1728)	This is GetTimeZone, a member of class TTimeZone.

26.1.1.2 TTimeZone Fields

The fields of the TTimeZone class are listed here.

Public Fields

	Name	Description
	FInfo (see page 1728)	This is FInfo, a member of class TTimeZone.
	FUsesDayligtSavings (see page 1728)	This is FUsesDayligtSavings, a member of class TTimeZone.

26.1.1.2.1 TTimeZone.FInfo Field

Pascal

```
FInfo: TTimeZoneInformation;
```

C++

```
TTimeZoneInformation FInfo;
```

Description

This is FInfo, a member of class TTimeZone.

26.1.1.2.2 TTimeZone.FUsesDayligtSavings Field

Pascal

```
FUsesDayligtSavings: Boolean;
```

C++

```
Boolean FUsesDayligtSavings;
```

Description

This is FUsesDayligtSavings, a member of class TTimeZone.

26.1.1.3 TTimeZone Methods

The methods of the TTimeZone class are listed here.

Public Methods

	Name	Description
	GetTimeZone (see page 1728)	This is GetTimeZone, a member of class TTimeZone.

26.1.1.3.1 TTimeZone.GetTimeZone Method

Pascal

```
class function GetTimeZone: TTimeZone; static;
```

C++

```
static __fastcall TTimeZone GetTimeZone();
```

Description

This is GetTimeZone, a member of class TTimeZone.

26.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckSqlTimeStamp (see page 1729)	Checks whether a TSQLTimeStamp (see page 1737) value represents a valid date and time.
	DateTimeToSQLTimeStamp (see page 1730)	Converts a TDateTime (see page 3323) value to a TSQLTimeStamp (see page 1737) value.
	LocalToUTC (see page 1730)	This is function SqlTimSt.LocalToUTC.
	SQLDayOfWeek (see page 1731)	Indicates the day of the week when a specified TSQLTimeStamp (see page 1737) value occurs.
	SQLTimeStampToDateTime (see page 1731)	Converts a TSQLTimeStamp (see page 1737) value to a TDateTime (see page 3323) value.
	SQLTimeStampToStr (see page 1732)	Converts a TSQLTimeStamp (see page 1737) value to a string.
	StrToSQLTimeStamp (see page 1732)	Converts a string to a TSQLTimeStamp (see page 1737) value.
	TryStrToSQLTimeStamp (see page 1733)	Converts a string to a TSQLTimeStamp (see page 1737) value.
	UTCToLocal (see page 1733)	This is function SqlTimSt.UTCToLocal.
	VarIsSQLTimeStamp (see page 1734)	Indicates whether a specified Variant (see page 3421) represents a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStamp (see page 1734)	Returns the TVarType (see page 3801) constant that identifies the internal type of a Variant (see page 3421) as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarSQLTimeStampCreate (see page 1735)	Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.
	VarToSQLTimeStamp (see page 1736)	Converts a Variant (see page 3421) that represents a TSQLTimeStamp (see page 1737) value to the corresponding TSQLTimeStamp (see page 1737).

26.2.1 SqlTimSt.CheckSqlTimeStamp Function

Checks whether a TSQLTimeStamp (see page 1737) value represents a valid date and time.

Pascal

```
procedure CheckSqlTimeStamp(const ASQLTimeStamp: TSQLTimeStamp);
```

C++

```
CheckSqlTimeStamp(const TSQLTimeStamp ASQLTimeStamp);
```

File

SqlTimSt

Description

CheckSqlTimeStamp checks that the specified TSQLTimeStamp (see page 1737) value represents a valid date and time. If

the fields of the ASQLTimeStamp parameter are out of the valid range, CheckSqlTimeStamp raises an EConversionError exception.

See Also

EConvertError (see page 3919)

26.2.2 SqlTimSt.DateTimeToSQLTimeStamp Function

Converts a TDateTime (see page 3323) value to a TSQLTimeStamp (see page 1737) value.

Pascal

```
function DateTimeToSQLTimeStamp(const DateTime: TDateTime): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp DateTimeToSQLTimeStamp(const TDateTime DateTime);
```

File

SqlTimSt

Description

Use DateTimeToSQLTimeStamp to convert the TDateTime (see page 3323) value specified by DateTime to a TSQLTimeStamp (see page 1737) value.

See Also

SQLTimeStampToDateTime (see page 1731)

SQLTimeStampToStr (see page 1732)

StrToSQLTimeStamp (see page 1732)

TSQLTimeStampField

26.2.3 SqlTimSt.LocalToUTC Function

Pascal

```
function LocalToUTC(var TZInfo: TTimeZone; var Value: TSQLTimeStamp): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp LocalToUTC(TTimeZone TZInfo, TSQLTimeStamp Value);
```

File

SqlTimSt

Description

This is function SqlTimSt.LocalToUTC.

26.2.4 SqlTimSt.SQLDayOfWeek Function

Indicates the day of the week when a specified TSQLTimeStamp (see page 1737) value occurs.

Pascal

```
function SQLDayOfWeek(const DateTime: TSQLTimeStamp): integer;
```

C++

```
int SQLDayOfWeek(const TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

SQLDayOfWeek returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

See Also

DayOfWeek (see page 4094)

26.2.5 SqlTimSt.SQLTimeStampToDateTime Function

Converts a TSQLTimeStamp (see page 1737) value to a TDateTime (see page 3323) value.

Pascal

```
function SQLTimeStampToDateTime(const DateTime: TSQLTimeStamp): TDateTime;
```

C++

```
TDateTime SQLTimeStampToDateTime(const TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

Use SQLTimeStampToDateTime to convert the TSQLTimeStamp (see page 1737) value specified by DateTime to a TDateTime (see page 3323) value.

See Also

DateTimeToSQLTimeStamp (see page 1730)

SQLTimeStampToStr (see page 1732)

StrToSQLTimeStamp (see page 1732)

TSQLTimeStampField

26.2.6 SqlTimSt.SQLTimeStampToStr Function

Converts a TSQLTimeStamp (see page 1737) value to a string.

Pascal

```
function SQLTimeStampToStr(const Format: string; DateTime: TSQLTimeStamp): string;
```

C++

```
UnicodeString SQLTimeStampToStr(const UnicodeString Format, TSQLTimeStamp DateTime);
```

File

SqlTimSt

Description

Use SQLTimeStampToStr to convert a TSQLTimeStamp (see page 1737) value to a string an AnsiString.

DateTimeToStr converts the TSQLTimeStamp (see page 1737) value given by DateTime using the format given by Format and returns the resulting string.

If the string specified by the Format parameter is empty, the TSQLTimeStamp (see page 1737) value is formatted using the ShortDateFormat global variable followed by the LongTimeFormat global variable. The time is not displayed if all time fields in the DateTime parameter are 0.

To create a format string for the Format parameter, use the standard date-time format specifiers:

See Also

DateTimeToSQLTimeStamp (see page 1730)

SQLTimeStampToDateTime (see page 1731)

StrToSQLTimeStamp (see page 1732)

TSQLTimeStampField

TryStrToSQLTimeStamp (see page 1733)

26.2.7 SqlTimSt.StrToSQLTimeStamp Function

Converts a string to a TSQLTimeStamp (see page 1737) value.

Pascal

```
function StrToSQLTimeStamp(const S: string): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp StrToSQLTimeStamp(const UnicodeString S);
```

File

SqlTimSt

Description

Use StrToSQLTimeStamp to convert the string S to a TSQLTimeStamp (see page 1737) value. If S does not represent a valid date-and-time value, StrToSQLTimeStamp raises an EConvertError exception.

Tip: To convert strings to TSQLTimeStamp (see page 1737) values without raising an exception when the string is not a valid date-and-time, use TryStrToSQLTimeStamp (see page 1733) instead.

See Also

DateTimeToSQLTimeStamp (see page 1730)

SQLTimeStampToStr (see page 1732)

SQLTimeStampToDateTime (see page 1731)

TSQLTimeStampField

EConvertError (see page 3919)

TryStrToSQLTimeStamp (see page 1733)

26.2.8 SqlTimSt.TryStrToSQLTimeStamp Function

Converts a string to a TSQLTimeStamp (see page 1737) value.

Pascal

```
function TryStrToSQLTimeStamp(const S: string; var TimeStamp: TSQLTimeStamp): Boolean;
```

C++

```
Boolean TryStrToSQLTimeStamp(const UnicodeString S, TSQLTimeStamp TimeStamp);
```

File

SqlTimSt

Description

Use TryStrToSQLTimeStamp to convert the string S to a TSQLTimeStamp (see page 1737) value.

If S can be converted to a TSQLTimeStamp (see page 1737) value, the result is returned as the TimeStamp parameter and TryStrToSQLTimeStamp returns true.

If S can't be converted to a TSQLTimeStamp (see page 1737) value, TimeStamp is set to NullSQLTimeStamp (see page 1740) and TryStrToSQLTimeStamp returns false.

See Also

DateTimeToSQLTimeStamp (see page 1730)

SQLTimeStampToStr (see page 1732)

SQLTimeStampToDateTime (see page 1731)

NullSQLTimeStamp (see page 1740)

StrToSQLTimeStamp (see page 1732)

26.2.9 SqlTimSt.UTCToLocal Function

Pascal

```
function UTCToLocal(var TZInfo: TTimeZone; var Value: TSQLTimeStamp): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp UTCToLocal(TTimeZone TZInfo, TSQLTimeStamp Value);
```

File

SqlTimSt

Description

This is function SqlTimSt.UTCToLocal.

26.2.10 SqlTimSt.VarIsSQLTimeStamp Function

Indicates whether a specified Variant (see page 3421) represents a TSQLTimeStamp (see page 1737) value.

Pascal

```
function VarIsSQLTimeStamp(const aValue: Variant): Boolean; overload;
```

C++

```
Boolean VarIsSQLTimeStamp(const Variant aValue);
```

File

SqlTimSt

Description

VarIsSQLTimeStamp returns true if the Variant (see page 3421) aValue stores its value internally as a TSQLTimeStamp (see page 1737) value. VarIsSQLTimeStamp returns false otherwise.

See Also

VarToSQLTimeStamp (see page 1736)

TSQLTimeStamp (see page 1737)

26.2.11 SqlTimSt.VarSQLTimeStamp Function

Returns the TVarType (see page 3801) constant that identifies the internal type of a Variant (see page 3421) as a TSQLTimeStamp (see page 1737) value.

Pascal

```
function VarSQLTimeStamp: TVarType;
```

C++

```
TVarType VarSQLTimeStamp();
```

File

SqlTimSt

Description

VarSQLTimeStamp returns the TVarType (see page 3801) constant that identifies a TSQLTimeStamp (see page 1737)-valued Variant (see page 3421). This is the value that the VarType (see page 5090) function returns when the Variant (see page 3421) stores its value as a TSQLTimeStamp (see page 1737) value.

The Variant (see page 3421) type codes for custom Variants (see page 4973) are dynamically allocated when the application loads the unit that implements them. VarSQLTimeStamp allows the application to identify the Variant (see page 3421) type code that is currently used for custom Variants (see page 4973) that represent an SQL time stamp. This value, however, may not match the type code used by another application that uses a different set of custom Variant (see page 3421) types.

See Also

VarIsSQLTimeStamp (see page 1734)

VarType (see page 5090)

TSQLTimeStamp (see page 1737)

VarToSQLTimeStamp (see page 1736)

26.2.12 SqlTimSt.VarSQLTimeStampCreate Function

Creates a new Variant (see page 3421) that stores its value internally as a TSQLTimeStamp (see page 1737) value.

Pascal

```
procedure VarSQLTimeStampCreate(var aDest: Variant; const ASQLTimeStamp: TSQLTimeStamp);
overload;
function VarSQLTimeStampCreate: Variant; overload;
function VarSQLTimeStampCreate(const AValue: string): Variant; overload;
function VarSQLTimeStampCreate(const AValue: TDateTime): Variant; overload;
function VarSQLTimeStampCreate(const ASQLTimeStamp: TSQLTimeStamp): Variant; overload;
```

C++

```
VarSQLTimeStampCreate(Variant aDest, const TSQLTimeStamp ASQLTimeStamp);
Variant VarSQLTimeStampCreate();
Variant VarSQLTimeStampCreate(const UnicodeString AValue);
Variant VarSQLTimeStampCreate(const TDateTime AValue);
Variant VarSQLTimeStampCreate(const TSQLTimeStamp ASQLTimeStamp);
```

File

SqlTimSt

Description

Call VarSQLTimeStampCreate to obtain a Variant (see page 3421) that stores its value as a TSQLTimeStamp (see page 1737) value.

Using the first syntax, the Variant (see page 3421) returned as aDest has the same value as ASQLTimeStamp.

Using the second syntax (passing no arguments), the returned Variant (see page 3421) has a value of NullSQLTimeStamp (see page 1740).

Using the third syntax, the returned Variant (see page 3421) has the same value as ASQLTimeStamp.

Using the fourth syntax, the returned Variant (see page 3421) represents the same date and time as the TDateTime (see page 3323) value AValue.

See Also

VarIsSQLTimeStamp (see page 1734)

VarToSQLTimeStamp (see page 1736)

NullSQLTimeStamp (see page 1740)

26.2.13 SqlTimSt.VarToSQLTimeStamp Function

Converts a Variant (see page 3421) that represents a TSQLTimeStamp (see page 1737) value to the corresponding TSQLTimeStamp (see page 1737).

Pascal

```
function VarToSQLTimeStamp(const aValue: Variant): TSQLTimeStamp;
```

C++

```
TSQLTimeStamp VarToSQLTimeStamp(const Variant aValue);
```

File

SqlTimSt

Description

Use VarToSQLTimeStamp to obtain a TSQLTimeStamp (see page 1737) from a variant whose value represents a date-and-time value. If the Variant (see page 3421) is a Null (see page 5056) Variant (see page 3421), VarToSQLTimeStamp returns NullSQLTimeStamp (see page 1740).

See Also

VarIsSQLTimeStamp (see page 1734)

TSQLTimeStamp (see page 1737)

NullSQLTimeStamp (see page 1740)

26.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TSQLTimeStamp (see page 1737)	TSQLTimeStamp represents time and date values with a high degree of accuracy.

26.3.1 SqlTimSt.TSQLTimeStamp Record

TSQLTimeStamp represents time and date values with a high degree of accuracy.

Pascal

```
TSQLTimeStamp = packed record
  Year: SmallInt;
  Month: Word;
  Day: Word;
  Hour: Word;
  Minute: Word;
  Second: Word;
  Fractions: LongWord;
end;
```

C++

```
struct TSQLTimeStamp {
  SmallInt Year;
  Word Month;
  Word Day;
  Word Hour;
  Word Minute;
  Word Second;
  LongWord Fractions;
};
```

File

SqlTimSt

Description

The dbExpress database drivers use TSQLTimeStamp when working with date and time information. Year specifies a year from 1 to 9999. Month specifies a month from 1 to 12. Day specifies a day of the month, from 1 through 28, 29, 30, or 31, depending on the Month value. Hour specifies the hour from 0 to 23. Minute and Second can range from 0 to 59. Fractions specifies milliseconds from 0 to 999.

You can create a Variant (see page 3421) that represents a TSQLTimeStamp value using the VarSQLTimeStampCreate (see page 1735) function. In fact, the easiest way to manipulate TSQLTimeStamp values is to create such a Variant (see page 3421) and use the built-in operators provided by the Variant (see page 3421) type.

See Also

TDateTime (see page 3787)

TTimeStamp (see page 4334)

TSQLTimeStampField

SQLTimeStampToDateTime (see page 1731)

SQLTimeStampToStr (see page 1732)

DateTimeToSQLTimeStamp (see page 1730)

SQLDayOfWeek (see page 1731)

26.4 Types

The following table lists types in this documentation.

Types

Name	Description
PSQLTimeStamp (see page 1739)	This is type SqlTimSt.PSQLTimeStamp.

26.4.1 SqlTimSt.PSQLTimeStamp Type

Pascal

```
PSQLTimeStamp = ^TSQLTimeStamp;
```

C++

```
^TSQLTimeStamp PSQLTimeStamp;
```

File

SqlTimSt

Description

This is type SqlTimSt.PSQLTimeStamp.

26.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NullSQLTimeStamp (see page 1740)	Represents a NULL TSQLTimeStamp (see page 1737) value.

26.5.1 SqlTimSt.NullSQLTimeStamp Constant

Represents a NULL TSQLTimeStamp (see page 1737) value.

Pascal

```
NullSQLTimeStamp: TSQLTimeStamp = (Year: 0; Month: 0; Day: 0; Hour: 0; Minute: 0; Second: 0;
Fractions: 0);
```

C++

```
TSQLTimeStamp NullSQLTimeStamp = (Year: 0; Month: 0; Day: 0; Hour: 0; Minute: 0; Second: 0;
Fractions: 0);
```

File

SqlTimSt

Description

Use NullSQLTimeStamp to indicate an unknown or missing TSQLTimeStamp (see page 1737) value.

See Also

Null (see page 5056)

27 StdActnMenus Namespace

This is namespace StdActnMenus.

Topics

Name	Description
Classes (see page 1742)	The following table lists classes in this documentation.
Functions (see page 1760)	The following table lists functions in this documentation.

Classes

	Name	Description
	TStandardAddRemoveItem (see page 1742)	TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.
	TStandardButtonControl (see page 1744)	This is class StdActnMenus.TStandardButtonControl.
	TStandardCustomizePopup (see page 1745)	This is class StdActnMenus.TStandardCustomizePopup.
	TStandardDropDownButton (see page 1747)	This is class StdActnMenus.TStandardDropDownButton.
	TStandardMenuButton (see page 1748)	TStandardMenuButton is the standard ActionBand menu button class.
	TStandardMenuExpandBtn (see page 1750)	This is class StdActnMenus.TStandardMenuExpandBtn.
	TStandardMenuItem (see page 1751)	TStandardMenuItem is the default class for menu items in an action band.
	TStandardMenuPopup (see page 1756)	TStandardMenuPopup is the default menu popup style.
	TStandardToolScrollBtn (see page 1758)	This is class StdActnMenus.TStandardToolScrollBtn.

Functions

	Name	Description
	RegisterStandardMenus (see page 1760)	This is function StdActnMenus.RegisterStandardMenus.

27.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TStandardAddRemoveItem (see page 1742)	TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.
	TStandardButtonControl (see page 1744)	This is class StdActnMenus.TStandardButtonControl.
	TStandardCustomizePopup (see page 1745)	This is class StdActnMenus.TStandardCustomizePopup.
	TStandardDropDownButton (see page 1747)	This is class StdActnMenus.TStandardDropDownButton.
	TStandardMenuButton (see page 1748)	TStandardMenuButton is the standard ActionBand menu button class.
	TStandardMenuExpandBtn (see page 1750)	This is class StdActnMenus.TStandardMenuExpandBtn.
	TStandardMenuItem (see page 1751)	TStandardMenuItem is the default class for menu items in an action band.
	TStandardMenuPopup (see page 1756)	TStandardMenuPopup is the default menu popup style.
	TStandardToolScrollBtn (see page 1758)	This is class StdActnMenus.TStandardToolScrollBtn.

27.1.1 TStandardAddRemoveItem Class

TStandardAddRemoveItem is the ActionBand menu class for customizing action bands.

Class Hierarchy

Pascal

```
TStandardAddRemoveItem = class(TCustomAddRemoveItem);
```

C++

```
class TStandardAddRemoveItem : public TCustomAddRemoveItem;
```

File

StdActnMenus

Description

TStandardAddRemoveItem is the standard ActionBand menu for customizing action bands at runtime.

Do not create instances of TStandardAddRemoveItem as this will be done for you automatically if the class is the default customization menu. If you would like to replace the standard customization menu, use the class TCustomAddRemoveItem as a parent class for your replacement class. To make the change for your application, set the value of MenuAddRemoveItemClass (a global variable) to your own subclass of TCustomAddRemoveItem instead of TStandardAddRemoveItem.

See Also

TCustomAddRemoveItem

27.1.1.1 TStandardAddRemoveItem Members

The following tables list the members exposed by TStandardAddRemoveItem.

Protected Methods

	Name	Description
	DrawBackground (see page 1743)	Draws a background image on the customization menu.
	DrawGlyph (see page 1743)	Draws a symbol on the control.

27.1.1.2 TStandardAddRemoveItem Methods

The methods of the TStandardAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 1743)	Draws a background image on the customization menu.
	DrawGlyph (see page 1743)	Draws a symbol on the control.

27.1.1.2.1 TStandardAddRemoveItem.DrawBackground Method

Draws a background image on the customization menu.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

This method is largely the same as the inherited method, with some modifications to account for the method of selection. See the inherited method for more details.

See Also

DrawBackground

DrawGlyph (see page 1743)

DrawText

27.1.1.2.2 TStandardAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

DrawGlyph performs the same function as the method it inherits from, and it does call the inherited method. The override sets many of the draw parameters based on the present state of the menu and how it has been selected.

See Also

DrawGlyph

DrawBackground (see page 1743)

DrawText

27.1.2 TStandardButtonControl Class

Class Hierarchy

Pascal

```
TStandardButtonControl = class(TCustomButtonControl);
```

C++

```
class TStandardButtonControl : public TCustomButtonControl;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardButtonControl.

27.1.2.1 TStandardButtonControl Members

The following tables list the members exposed by TStandardButtonControl.

Protected Methods

	Name	Description
	DrawFrame (see page 1744)	This is DrawFrame, a member of class TStandardButtonControl.
	DrawGlyph (see page 1745)	Draws a symbol on the control.
	DrawText (see page 1745)	This is DrawText, a member of class TStandardButtonControl.

27.1.2.2 TStandardButtonControl Methods

The methods of the TStandardButtonControl class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 1744)	This is DrawFrame, a member of class TStandardButtonControl.
	DrawGlyph (see page 1745)	Draws a symbol on the control.
	DrawText (see page 1745)	This is DrawText, a member of class TStandardButtonControl.

27.1.2.2.1 TStandardButtonControl.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardButtonControl.

27.1.2.2.2 TStandardButtonControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

27.1.2.2.3 TStandardButtonControl.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TStandardButtonControl.

27.1.3 TStandardCustomizePopup Class

Class Hierarchy

Pascal

```
TStandardCustomizePopup = class(TCustomizeActionToolBar);
```

C++

```
class TStandardCustomizePopup : public TCustomizeActionToolBar;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardCustomizePopup.

27.1.3.1 TStandardCustomizePopup Members

The following tables list the members exposed by TStandardCustomizePopup.

Protected Methods

	Name	Description
	CanAutoSize (see page 1746)	Indicates how the control can be resized.
	NCPaint (see page 1746)	This is NCPaint, a member of class TStandardCustomizePopup.
	PositionPopup (see page 1747)	This is PositionPopup, a member of class TStandardCustomizePopup.

27.1.3.2 TStandardCustomizePopup Methods

The methods of the TStandardCustomizePopup class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 1746)	Indicates how the control can be resized.
	NCPaint (see page 1746)	This is NCPaint, a member of class TStandardCustomizePopup.
	PositionPopup (see page 1747)	This is PositionPopup, a member of class TStandardCustomizePopup.

27.1.3.2.1 TStandardCustomizePopup.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints

ConstrainedResize

AdjustSize

27.1.3.2.2 TStandardCustomizePopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TStandardCustomizePopup.

27.1.3.2.3 TStandardCustomizePopup.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TStandardCustomizePopup.

27.1.4 TStandardDropDownButton Class

Class Hierarchy

Pascal

```
TStandardDropDownButton = class(TCustomDropDownButton);
```

C++

```
class TStandardDropDownButton : public TCustomDropDownButton;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardDropDownButton.

27.1.4.1 TStandardDropDownButton Members

The following tables list the members exposed by TStandardDropDownButton.

Protected Methods

	Name	Description
	DrawFrame (see page 1748)	This is DrawFrame, a member of class TStandardDropDownButton.
	GetPopupClass (see page 1748)	This is GetPopupClass, a member of class TStandardDropDownButton.

27.1.4.2 TStandardDropDownButton Methods

The methods of the TStandardDropDownButton class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 1748)	This is DrawFrame, a member of class TStandardDropDownButton.
	GetPopupClass (see page 1748)	This is GetPopupClass, a member of class TStandardDropDownButton.

27.1.4.2.1 TStandardDropDownButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardDropDownButton.

27.1.4.2.2 TStandardDropDownButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; override;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TStandardDropDownButton.

27.1.5 TStandardMenuButton Class

TStandardMenuButton is the standard ActionBand menu button class.

Class Hierarchy

Pascal

```
TStandardMenuButton = class(TCustomMenuButton);
```

C++

```
class TStandardMenuButton : public TCustomMenuButton;
```

File

StdActnMenus

Description

TStandardMenuButton is the standard menu button class. If you would like to replace the standard menu button behaviors, use the class TCustomMenuButton as a parent class for your replacement class. To make the change for your application, set the value of MenuButtonControlClass (a global variable) to your own subclass of TCustomMenuButton instead of TStandardMenuButton.

See Also

TCustomMenuButton

27.1.5.1 TStandardMenuButton Members

The following tables list the members exposed by TStandardMenuButton.

Protected Methods

	Name	Description
	DrawBackground (see page 1749)	Draws a background image on the control.
	DrawDesignFocus (see page 1749)	This is DrawDesignFocus, a member of class TStandardMenuButton.
	DrawFrame (see page 1750)	This is DrawFrame, a member of class TStandardMenuButton.
	DrawText (see page 1750)	This is DrawText, a member of class TStandardMenuButton.

27.1.5.2 TStandardMenuButton Methods

The methods of the TStandardMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 1749)	Draws a background image on the control.
	DrawDesignFocus (see page 1749)	This is DrawDesignFocus, a member of class TStandardMenuButton.
	DrawFrame (see page 1750)	This is DrawFrame, a member of class TStandardMenuButton.
	DrawText (see page 1750)	This is DrawText, a member of class TStandardMenuButton.

27.1.5.2.1 TStandardMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

27.1.5.2.2 TStandardMenuButton.DrawDesignFocus Method

Pascal

```
procedure DrawDesignFocus(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawDesignFocus(TRect PaintRect);
```

Description

This is DrawDesignFocus, a member of class TStandardMenuButton.

27.1.5.2.3 TStandardMenuButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardMenuButton.

27.1.5.2.4 TStandardMenuButton.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TStandardMenuButton.

27.1.6 TStandardMenuExpandBtn Class

Class Hierarchy

Pascal

```
TStandardMenuExpandBtn = class(TCustomMenuExpandBtn);
```

C++

```
class TStandardMenuExpandBtn : public TCustomMenuExpandBtn;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardMenuExpandBtn.

27.1.6.1 TStandardMenuExpandBtn Members

The following tables list the members exposed by TStandardMenuExpandBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 1751)	Draws a background image on the control.
	DrawFrame (see page 1751)	This is DrawFrame, a member of class TStandardMenuExpandBtn.

27.1.6.2 TStandardMenuExpandBtn Methods

The methods of the TStandardMenuExpandBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 1751)	Draws a background image on the control.
	DrawFrame (see page 1751)	This is DrawFrame, a member of class TStandardMenuExpandBtn.

27.1.6.2.1 TStandardMenuExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

27.1.6.2.2 TStandardMenuExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

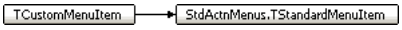
Description

This is DrawFrame, a member of class TStandardMenuExpandBtn.

27.1.7 TStandardMenuItem Class

TStandardMenuItem is the default class for menu items in an action band.

Class Hierarchy



Pascal

```
TStandardMenuItem = class(TCustomMenuItem);
```

C++

```
class TStandardMenuItem : public TCustomMenuItem;
```

File

StdActnMenus

Description

TStandardMenuItem is the class that action bands use to generate child controls that are menu items. Do not create instances of TStandardMenuItem. The action manager automatically handles the creating of TStandardMenuItem instances to represent the actions it contains.

If you want to replace the class used for menu items, derive your own replacement class from TCustomMenuItem. To substitute your replacement class, set the value of the global MenuItemControlClass variable to your TCustomMenuItem descendant.

See Also

TStandardMenuButton (see page 1748)

TStandardMenuPopup (see page 1756)

MenuItemControlClass

27.1.7.1 TStandardMenuItem Members

The following tables list the members exposed by TStandardMenuItem.

Protected Methods

	Name	Description
	CMTextchanged (see page 1753)	Responds when this menu item's caption changes.
V	DrawBackground (see page 1753)	Draws the background of the menu item.
V	DrawDesignFocus (see page 1753)	This is DrawDesignFocus, a member of class TStandardMenuItem.
V	DrawEdge (see page 1754)	This is DrawEdge, a member of class TStandardMenuItem.
V	DrawGlyph (see page 1754)	Draws a symbol on the menu item.
	DrawGlyphFrame (see page 1754)	This is DrawGlyphFrame, a member of class TStandardMenuItem.
V	DrawLargeGlyph (see page 1755)	This is DrawLargeGlyph, a member of class TStandardMenuItem.
V	DrawSeparator (see page 1755)	Draws a separator.
V	DrawText (see page 1755)	Draws the menu item's text.

27.1.7.2 TStandardMenuItem Methods

The methods of the TStandardMenuItem class are listed here.

Protected Methods

	Name	Description
	CMTextchanged (see page 1753)	Responds when this menu item's caption changes.
V	DrawBackground (see page 1753)	Draws the background of the menu item.
V	DrawDesignFocus (see page 1753)	This is DrawDesignFocus, a member of class TStandardMenuItem.
V	DrawEdge (see page 1754)	This is DrawEdge, a member of class TStandardMenuItem.
V	DrawGlyph (see page 1754)	Draws a symbol on the menu item.
	DrawGlyphFrame (see page 1754)	This is DrawGlyphFrame, a member of class TStandardMenuItem.
V	DrawLargeGlyph (see page 1755)	This is DrawLargeGlyph, a member of class TStandardMenuItem.

	DrawSeparator (see page 1755)	Draws a separator.
	DrawText (see page 1755)	Draws the menu item's text.

27.1.7.2.1 TStandardMenuItem.CMTextchanged Method

Responds when this menu item's caption changes.

Pascal

```
procedure CMTextchanged(var Message: TMessage); message CM_TEXTCHANGED;
```

C++

```
__fastcall CMTextchanged(TMessage Message);
```

Description

This method is called automatically in response to Windows messages containing a notification that the text of this menu item has changed.

Message represents the Windows message that notifies this menu item of a change to its text.

CMTextchanged recalculates the string it displays when the menu item is not underlining its accelerator key.

See Also

PersistentHotKeys

27.1.7.2.2 TStandardMenuItem.DrawBackground Method

Draws the background of the menu item.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

This method is called automatically when the menu item needs to render its background (everything except the text and glyph).

PaintRect indicates the dimensions of the menu item on its canvas.

DrawBackground paints the background of the menu item so that its color reflects whether the item currently has focus, adjusting PaintRect to accommodate any glyph or checkmark that appears on the menu item.

See Also

DrawBackground

DrawGlyph (see page 1754)

DrawText (see page 1755)

27.1.7.2.3 TStandardMenuItem.DrawDesignFocus Method

Pascal

```
procedure DrawDesignFocus(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawDesignFocus(TRect PaintRect);
```

Description

This is DrawDesignFocus, a member of class TStandardMenuItem.

27.1.7.2.4 TStandardMenuItem.DrawEdge Method

Pascal

```
procedure DrawEdge(Edges: TMenuEdges); override;
```

C++

```
virtual __fastcall DrawEdge(TMenuEdges Edges);
```

Description

This is DrawEdge, a member of class TStandardMenuItem.

27.1.7.2.5 TStandardMenuItem.DrawGlyph Method

Draws a symbol on the menu item.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

DrawGlyph is called automatically when the menu item needs to render its glyph. This occurs if the menu item is checked or if the associated action client item specifies a glyph for this menu item using its ImageIndex property.

Location is the position on this menu item's canvas for the upper left corner of the glyph.

See Also

DrawGlyph

DrawBackground (see page 1753)

DrawText (see page 1755)

ImageIndex

27.1.7.2.6 TStandardMenuItem.DrawGlyphFrame Method

Pascal

```
procedure DrawGlyphFrame(const Location: TPoint);
```

C++

```
__fastcall DrawGlyphFrame(const TPoint Location);
```

Description

This is DrawGlyphFrame, a member of class TStandardMenuItem.

27.1.7.2.7 TStandardMenuItem.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TStandardMenuItem.

27.1.7.2.8 TStandardMenuItem.DrawSeparator Method

Draws a separator.

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

DrawSeparator is called automatically when the menu item represents a separator bar in the menu and it needs to paint itself. This method paints the separator bar.

Offset indicates the offset of the separator, in pixels, from the top of the menu item. DrawSeparator ignores this value and always uses a fixed value for menu items.

27.1.7.2.9 TStandardMenuItem.DrawText Method

Draws the menu item's text.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

DrawText is called automatically when the menu item needs to render its text.

ARect indicates the rectangle on the menu item's canvas where the text label should appear.

Flags supplies the value to use for the TextFlags property of the canvas.

Text is the text of the menu item.

DrawText checks whether the menu item should display underlines on accelerator keys, and draws the text accordingly, using a font color that reflects whether the menu item is selected and whether it is enabled.

See Also

DrawText

DrawBackground (see page 1753)

DrawGlyph (see page 1754)

PersistentHotKeys

TextFlags

27.1.8 TStandardMenuPopup Class

TStandardMenuPopup is the default menu popup style.

Class Hierarchy

Pascal

```
TStandardMenuPopup = class(TCustomActionPopupMenu);
```

C++

```
class TStandardMenuPopup : public TCustomActionPopupMenu;
```

File

StdActnMenus

Description

TStandardMenuPopup is the default menu popup style. If you would like to replace it with a style of your own, subclass TCustomActionPopupMenu to provide your own behaviors. To make the change for your application, set the value of MenuPopupClass (a global variable) to your own subclass of TCustomActionPopupMenu instead of TStandardMenuPopup.

See Also

TActionMainMenuBar

TStandardMenuButton (see page 1748)

TStandardMenuItem (see page 1751)

27.1.8.1 TStandardMenuPopup Members

The following tables list the members exposed by TStandardMenuPopup.

Protected Methods

	Name	Description
	CanAutoSize (see page 1757)	Indicates how the control can be resized.
	GetExpandBtnClass (see page 1757)	This is GetExpandBtnClass, a member of class TStandardMenuPopup.
	NCPaint (see page 1757)	This is NCPaint, a member of class TStandardMenuPopup.
	PositionPopup (see page 1758)	This is PositionPopup, a member of class TStandardMenuPopup.

27.1.8.2 TStandardMenuPopup Methods

The methods of the TStandardMenuPopup class are listed here.

Protected Methods

	Name	Description
	CanAutoSize (see page 1757)	Indicates how the control can be resized.

	GetExpandBtnClass (see page 1757)	This is GetExpandBtnClass, a member of class TStandardMenuPopup.
	NCPaint (see page 1757)	This is NCPaint, a member of class TStandardMenuPopup.
	PositionPopup (see page 1758)	This is PositionPopup, a member of class TStandardMenuPopup.

27.1.8.2.1 TStandardMenuPopup.CanAutoSize Method

Indicates how the control can be resized.

Pascal

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanAutoSize(int NewWidth, int NewHeight);
```

Description

CanAutoSize is called automatically when the AutoSize property is true and an attempt is made to resize the control. It allows controls to implement the AutoSize property before the resize sequence that begins with the OnCanResize and OnConstrainedResize events and ends with the OnResize event.

When CanAutoSize is called, the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control. CanAutoSize adjusts these values so that the control's size adjusts to its contents. If the adjustment results in values of NewWidth and NewHeight that differ from the control's current width and height, CanAutoSize returns true, indicating that the resize sequence should proceed. If, after adjustment, NewWidth and NewHeight equal Width and Height, CanAutoSize returns false, and the resize sequence is aborted.

As implemented in TWinControl, CanAutoSize returns true, adjusting NewWidth and NewHeight so that all the control's children fit.

See Also

Constraints

ConstrainedResize

AdjustSize

27.1.8.2.2 TStandardMenuPopup.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TStandardMenuPopup.

27.1.8.2.3 TStandardMenuPopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TStandardMenuPopup.

27.1.8.2.4 TStandardMenuPopup.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TStandardMenuPopup.

27.1.9 TStandardToolScrollBtn Class

Class Hierarchy

Pascal

```
TStandardToolScrollBtn = class(TCustomToolScrollBtn);
```

C++

```
class TStandardToolScrollBtn : public TCustomToolScrollBtn;
```

File

StdActnMenus

Description

This is class StdActnMenus.TStandardToolScrollBtn.

27.1.9.1 TStandardToolScrollBtn Members

The following tables list the members exposed by TStandardToolScrollBtn.

Protected Methods

	Name	Description
	DrawFrame (see page 1759)	This is DrawFrame, a member of class TStandardToolScrollBtn.

27.1.9.2 TStandardToolScrollBtn Methods

The methods of the TStandardToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawFrame (see page 1759)	This is DrawFrame, a member of class TStandardToolScrollBtn.

27.1.9.2.1 TStandardToolScrollBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TStandardToolScrollBtn.

27.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	RegisterStandardMenus (see page 1760)	This is function StdActnMenus.RegisterStandardMenus.

27.2.1 StdActnMenus.RegisterStandardMenus Function

Pascal

```
procedure RegisterStandardMenus; deprecated;
```

C++

```
RegisterStandardMenus();
```

File

StdActnMenus

Description

This is function StdActnMenus.RegisterStandardMenus.

28 StdActns Namespace

This is namespace StdActns.

Topics

Name	Description
Classes (see page 1763)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 1947)	The following table lists structs, records, enums in this documentation.
Types (see page 1949)	The following table lists types in this documentation.

Classes

	Name	Description
	TBrowseForFolder (see page 1764)	This is class StdActns.TBrowseForFolder.
	TColorSelect (see page 1774)	TColorSelect is the standard action for displaying a color selection dialog.
	TCommonDialogAction (see page 1784)	TCommonDialogAction is the base class for action objects that display a dialog.
	TEditAction (see page 1791)	TEditAction is the base class for action objects that perform standard edits on an edit control.
	TEditCopy (see page 1795)	TEditCopy copies the selected text in a target edit control to the Windows clipboard.
	TEditCut (see page 1797)	TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.
	TEditDelete (see page 1799)	TEditDelete deletes the selected text in a target edit or memo control.
	TEditPaste (see page 1801)	TEditPaste pastes text from the clipboard into the active edit or memo control.
	TEditSelectAll (see page 1803)	TEditSelectAll selects all text in an active text control.
	TEditUndo (see page 1805)	TEditUndo backs out the last change in an edit control.
	TFileAction (see page 1807)	TFileAction is the base class for action objects that display a file selection dialog.
	TFileExit (see page 1810)	TFileExit is the standard action for shutting down an application.
	TFileOpen (see page 1817)	TFileOpen is the standard action for displaying a file open dialog.
	TFileOpenWith (see page 1829)	TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.
	TFilePageSetup (see page 1836)	TFilePageSetup is the standard action for displaying a page setup dialog.
	TFilePrintSetup (see page 1845)	TFilePrintSetup is the standard action for displaying a printer setup dialog.
	TFileSaveAs (see page 1855)	TFileSaveAs is the standard action for displaying a file save as dialog.
	TFontEdit (see page 1864)	TFontEdit is the standard action for displaying a font selection dialog.
	THelpAction (see page 1874)	THelpAction is the base class for action objects that interact with the help system.
	THelpContents (see page 1876)	THelpContents brings up the contents for the default help system.
	THelpContextAction (see page 1877)	THelpContextAction brings up the help topic for the active control.
	THelpOnHelp (see page 1879)	THelpOnHelp brings up a help topic on how to use Help.

	THelpTopicSearch (see page 1881)	THelpTopicSearch brings up the help index.
	THintAction (see page 1882)	THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.
	TPrintDlg (see page 1884)	TPrintDlg is the standard action for displaying a print dialog.
	TSearchAction (see page 1894)	TSearchAction is the base class for action objects that display a search or search and replace dialog.
	TSearchFind (see page 1899)	TSearchFind is the standard action for displaying a find dialog.
	TSearchFindFirst (see page 1909)	TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.
	TSearchFindNext (see page 1912)	TSearchFindNext is the standard action for locating the next instance of a string in an edit control.
	TSearchReplace (see page 1921)	TSearchReplace is the standard action for displaying a find-and-replace dialog.
	TWindowAction (see page 1932)	TWindowAction is the base class for action objects that work with MDI forms.
	TWindowArrange (see page 1936)	TWindowArrange arranges the icons of minimized MDI child forms.
	TWindowCascade (see page 1937)	TWindowCascade cascades the MDI child forms.
	TWindowClose (see page 1939)	TWindowClose closes the active MDI child form.
	TWindowMinimizeAll (see page 1941)	TWindowMinimizeAll minimizes all MDI child forms.
	TWindowTileHorizontal (see page 1943)	TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.
	TWindowTileVertical (see page 1944)	TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

Structs, Records, Enums

	Name	Description
	TBrowseForFolderOption (see page 1947)	TBrowseForFolderOption contains the options for creating a dialog box used when browsing for folders.

Types

Name	Description
TBrowseForFolderCallbackEvent (see page 1949)	This is type StdActns.TBrowseForFolderCallbackEvent.
TBrowseOptions (see page 1949)	TBrowseOptions is a set type of options used in customizing the browse dialog box to create.
TCommonDialogClass (see page 1950)	This is type StdActns.TCommonDialogClass.

28.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TBrowseForFolder (see page 1764)	This is class StdActns.TBrowseForFolder.
	TColorSelect (see page 1774)	TColorSelect is the standard action for displaying a color selection dialog.
	TCommonDialogAction (see page 1784)	TCommonDialogAction is the base class for action objects that display a dialog.
	TEditAction (see page 1791)	TEditAction is the base class for action objects that perform standard edits on an edit control.
	TEditCopy (see page 1795)	TEditCopy copies the selected text in a target edit control to the Windows clipboard.
	TEditCut (see page 1797)	TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.
	TEditDelete (see page 1799)	TEditDelete deletes the selected text in a target edit or memo control.
	TEditPaste (see page 1801)	TEditPaste pastes text from the clipboard into the active edit or memo control.
	TEditSelectAll (see page 1803)	TEditSelectAll selects all text in an active text control.
	TEditUndo (see page 1805)	TEditUndo backs out the last change in an edit control.
	TFileAction (see page 1807)	TFileAction is the base class for action objects that display a file selection dialog.
	TFileExit (see page 1810)	TFileExit is the standard action for shutting down an application.
	TFileOpen (see page 1817)	TFileOpen is the standard action for displaying a file open dialog.
	TFileOpenWith (see page 1829)	TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.
	TFilePageSetup (see page 1836)	TFilePageSetup is the standard action for displaying a page setup dialog.
	TFilePrintSetup (see page 1845)	TFilePrintSetup is the standard action for displaying a printer setup dialog.
	TFileSaveAs (see page 1855)	TFileSaveAs is the standard action for displaying a file save as dialog.
	TFontEdit (see page 1864)	TFontEdit is the standard action for displaying a font selection dialog.
	THelpAction (see page 1874)	THelpAction is the base class for action objects that interact with the help system.
	THelpContents (see page 1876)	THelpContents brings up the contents for the default help system.
	THelpContextAction (see page 1877)	THelpContextAction brings up the help topic for the active control.
	THelpOnHelp (see page 1879)	THelpOnHelp brings up a help topic on how to use Help.
	THelpTopicSearch (see page 1881)	THelpTopicSearch brings up the help index.
	THintAction (see page 1882)	THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.
	TPrintDlg (see page 1884)	TPrintDlg is the standard action for displaying a print dialog.
	TSearchAction (see page 1894)	TSearchAction is the base class for action objects that display a search or search and replace dialog.
	TSearchFind (see page 1899)	TSearchFind is the standard action for displaying a find dialog.
	TSearchFindFirst (see page 1909)	TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.
	TSearchFindNext (see page 1912)	TSearchFindNext is the standard action for locating the next instance of a string in an edit control.
	TSearchReplace (see page 1921)	TSearchReplace is the standard action for displaying a find-and-replace dialog.
	TWindowAction (see page 1932)	TWindowAction is the base class for action objects that work with MDI forms.
	TWindowArrange (see page 1936)	TWindowArrange arranges the icons of minimized MDI child forms.
	TWindowCascade (see page 1937)	TWindowCascade cascades the MDI child forms.
	TWindowClose (see page 1939)	TWindowClose closes the active MDI child form.
	TWindowMinimizeAll (see page 1941)	TWindowMinimizeAll minimizes all MDI child forms.
	TWindowTileHorizontal (see page 1943)	TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.
	TWindowTileVertical (see page 1944)	TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

28.1.1 TBrowseForFolder Class

Class Hierarchy

Pascal

```
TBrowseForFolder = class(TCommonDialogAction);
```

C++

```
class TBrowseForFolder : public TCommonDialogAction;
```

File

StdActns

Description

This is class StdActns.TBrowseForFolder.

28.1.1.1 TBrowseForFolder Members

The following tables list the members exposed by TBrowseForFolder.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCallback (see page 1765)	This is DoCallback, a member of class TBrowseForFolder.
V	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	DoValidate (see page 1766)	This is DoValidate, a member of class TBrowseForFolder.
V	GetDialogClass (see page 1766)	This is GetDialogClass, a member of class TBrowseForFolder.
V	Notification (see page 1789)	Responds when components are created or destroyed.
V	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1785)	Creates an instance of TCommonDialogAction.
V	Destroy (see page 1786)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1766)	Executes the associated dialog and generates an OnAccept or OnCancel event.
V	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.
	Folder (see page 1768)	This is Folder, a member of class TBrowseForFolder.

Published Events

	Name	Description
	BeforeExecute (see page 1772)	Occurs immediately before the dialog appears.

	OnAccept (see page 1772)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCallback (see page 1773)	This is OnCallback, a member of class TBrowseForFolder.
	OnCancel (see page 1773)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1774)	Occurs when the mouse pauses over a client control or menu item.
	OnValidate (see page 1774)	This is OnValidate, a member of class TBrowseForFolder.

Published Properties

	Name	Description
	BrowseOptions (see page 1767)	This is BrowseOptions, a member of class TBrowseForFolder.
	Caption (see page 1767)	Represents the caption of client controls and menu items.
	DialogCaption (see page 1768)	This is DialogCaption, a member of class TBrowseForFolder.
	Enabled (see page 1768)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1768)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1769)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1769)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1770)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1770)	Indicates the ImageIndex property for client controls and menu items.
	RootDir (see page 1770)	This is RootDir, a member of class TBrowseForFolder.
	SecondaryShortCuts (see page 1771)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1771)	Specifies the ShortCut property for client menu items.
	Visible (see page 1771)	Specifies the Visible property for client controls and menu items.

28.1.1.2 TBrowseForFolder Methods

The methods of the TBrowseForFolder class are listed here.

Protected Methods

	Name	Description
	DoCallback (see page 1765)	This is DoCallback, a member of class TBrowseForFolder.
	DoValidate (see page 1766)	This is DoValidate, a member of class TBrowseForFolder.
V	GetDialogClass (see page 1766)	This is GetDialogClass, a member of class TBrowseForFolder.

Public Methods

	Name	Description
V	ExecuteTarget (see page 1766)	Executes the associated dialog and generates an OnAccept or OnCancel event.

28.1.1.2.1 TBrowseForFolder.DoCallback Method

Pascal

```
function DoCallback(Wnd: HWND; uMsg: UINT; lParam: LPARAM; lpData: LPARAM): Integer;
```

C++

```
__fastcall int DoCallback(HWND Wnd, UINT uMsg, LPARAM lParam, LPARAM lpData);
```

Description

This is DoCallback, a member of class TBrowseForFolder.

28.1.1.2.2 TBrowseForFolder.DoValidate Method

Pascal

```
function DoValidate(Wnd: HWND; uMsg: UINT; lParam: LPARAM; lpData: LPARAM): Integer;
```

C++

```
__fastcall int DoValidate(HWND Wnd, UINT uMsg, LPARAM lParam, LPARAM lpData);
```

Description

This is DoValidate, a member of class TBrowseForFolder.

28.1.1.2.3 TBrowseForFolder.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept or OnCancel event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept event or an OnCancel event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Execute

BeforeExecute (see page 1790)

28.1.1.2.4 TBrowseForFolder.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TBrowseForFolder.

28.1.1.3 TBrowseForFolder Properties

The properties of the TBrowseForFolder class are listed here.

Public Properties

	Name	Description
	Folder (see page 1768)	This is Folder, a member of class TBrowseForFolder.

Published Properties

	Name	Description
	BrowseOptions (see page 1767)	This is BrowseOptions, a member of class TBrowseForFolder.
	Caption (see page 1767)	Represents the caption of client controls and menu items.
	DialogCaption (see page 1768)	This is DialogCaption, a member of class TBrowseForFolder.
	Enabled (see page 1768)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1768)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1769)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1769)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1770)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1770)	Indicates the ImageIndex property for client controls and menu items.
	RootDir (see page 1770)	This is RootDir, a member of class TBrowseForFolder.
	SecondaryShortCuts (see page 1771)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1771)	Specifies the ShortCut property for client menu items.
	Visible (see page 1771)	Specifies the Visible property for client controls and menu items.

28.1.1.3.1 TBrowseForFolder.BrowseOptions Property

Pascal

```
property BrowseOptions: TBrowseOptions;
```

C++

```
__property TBrowseOptions BrowseOptions;
```

Description

This is BrowseOptions, a member of class TBrowseForFolder.

28.1.1.3.2 TBrowseForFolder.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.1.3.3 TBrowseForFolder.DialogCaption Property

Pascal

```
property DialogCaption: string;
```

C++

```
__property UnicodeString DialogCaption;
```

Description

This is DialogCaption, a member of class TBrowseForFolder.

28.1.1.3.4 TBrowseForFolder.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.1.3.5 TBrowseForFolder.Folder Property

Pascal

```
property Folder: string;
```

C++

```
__property UnicodeString Folder;
```

Description

This is Folder, a member of class TBrowseForFolder.

28.1.1.3.6 TBrowseForFolder.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.1.3.7 TBrowseForFolder.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.1.3.8 TBrowseForFolder.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.1.3.9 TBrowseForFolder.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.1.3.10 TBrowseForFolder.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.1.3.11 TBrowseForFolder.RootDir Property

Pascal

```
property RootDir: string;
```

C++

```
__property UnicodeString RootDir;
```

Description

This is RootDir, a member of class TBrowseForFolder.

28.1.1.3.12 TBrowseForFolder.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.1.3.13 TBrowseForFolder.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.1.3.14 TBrowseForFolder.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.1.4 TBrowseForFolder Events

The events of the TBrowseForFolder class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1772)	Occurs immediately before the dialog appears.
	OnAccept (see page 1772)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCallback (see page 1773)	This is OnCallback, a member of class TBrowseForFolder.
	OnCancel (see page 1773)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1774)	Occurs when the mouse pauses over a client control or menu item.
	OnValidate (see page 1774)	This is OnValidate, a member of class TBrowseForFolder.

28.1.1.4.1 TBrowseForFolder.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.1.4.2 TBrowseForFolder.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.1.4.3 TBrowseForFolder.OnCallback Event

Pascal

```
property OnCallback: TBrowseForFolderCallbackEvent;
```

C++

```
__property TBrowseForFolderCallbackEvent OnCallback;
```

Description

This is OnCallback, a member of class TBrowseForFolder.

28.1.1.4.4 TBrowseForFolder.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.1.4.5 TBrowseForFolder.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.1.4.6 TBrowseForFolder.OnValidate Event

Pascal

```
property OnValidate: TBrowseForFolderCallbackEvent;
```

C++

```
__property TBrowseForFolderCallbackEvent OnValidate;
```

Description

This is OnValidate, a member of class TBrowseForFolder.

28.1.2 TColorSelect Class

TColorSelect is the standard action for displaying a color selection dialog.

Class Hierarchy

Pascal

```
TColorSelect = class(TCommonDialogAction);
```

C++

```
class TColorSelect : public TCommonDialogAction;
```

File

StdActns

Description

Add TColorSelect to an action list to add a modal color selection dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the color selection dialog when invoked. Use the OnAccept (see page 1781) event to respond after the user selects a color using the dialog or the OnCancel (see page 1782) event to respond if the user cancels from the dialog. In the event handler for these events, you can read the color from the dialog specified by the Dialog (see page 1777) property.

See Also

TColorDialog

28.1.2.1 TColorSelect Members

The following tables list the members exposed by TColorSelect.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1776)	This is GetDialogClass, a member of class TColorSelect.
V	Notification (see page 1789)	Responds when components are created or destroyed.
V	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1785)	Creates an instance of TCommonDialogAction.
V	Destroy (see page 1786)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
V	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1781)	Occurs immediately before the dialog appears.
	OnAccept (see page 1781)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1782)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1782)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1782)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1776)	Represents the caption of client controls and menu items.
R	Dialog (see page 1777)	Provides access the color selection dialog that the action executes.
	Enabled (see page 1777)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1777)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1778)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1778)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1779)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1779)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1780)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1780)	Specifies the ShortCut property for client menu items.

	Visible (see page 1780)	Specifies the Visible property for client controls and menu items.

28.1.2.2 TColorSelect Methods

The methods of the TColorSelect class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 1776)	This is GetDialogClass, a member of class TColorSelect.

28.1.2.2.1 TColorSelect.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TColorSelect.

28.1.2.3 TColorSelect Properties

The properties of the TColorSelect class are listed here.

Published Properties

	Name	Description
	Caption (see page 1776)	Represents the caption of client controls and menu items.
R	Dialog (see page 1777)	Provides access the color selection dialog that the action executes.
	Enabled (see page 1777)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1777)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1778)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1778)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1779)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1779)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1780)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1780)	Specifies the ShortCut property for client menu items.
	Visible (see page 1780)	Specifies the Visible property for client controls and menu items.

28.1.2.3.1 TColorSelect.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and

menu items.

See Also

Caption

Caption

28.1.2.3.2 TColorSelect.Dialog Property

Provides access the color selection dialog that the action executes.

Pascal

```
property Dialog: TColorDialog;
```

C++

```
__property TColorDialog Dialog;
```

Description

Use Dialog to configure the color selection dialog before the action fires or to read the selected color after the action executes the dialog. For example, you can set the initially selected color in the dialog at design time using the Object Inspector or read the dialog's Color property in an OnAccept (see page 1781) event handler to make use of the selected color.

When using a color select action, you do not place a TColorDialog component on the form or data module at design time. Instead, you add the TColorSelect (see page 1774) to your action list and then use the Dialog property. The color selection dialog component is created by the TColorSelect (see page 1774) constructor.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Color

28.1.2.3.3 TColorSelect.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.2.3.4 TColorSelect.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.2.3.5 TColorSelect.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.2.3.6 TColorSelect.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.2.3.7 TColorSelect.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.2.3.8 TColorSelect.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.2.3.9 TColorSelect.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.2.3.10 TColorSelect.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.2.3.11 TColorSelect.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.2.4 TColorSelect Events

The events of the TColorSelect class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1781)	Occurs immediately before the dialog appears.
	OnAccept (see page 1781)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1782)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1782)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1782)	Occurs when the application is idle or when the action list updates.

28.1.2.4.1 TColorSelect.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.2.4.2 TColorSelect.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the

user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.2.4.3 TColorSelect.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.2.4.4 TColorSelect.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.2.4.5 TColorSelect.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
```

```
end;
```

See Also

UpdateAction

28.1.3 TCommonDialogAction Class

TCommonDialogAction is the base class for action objects that display a dialog.

Class Hierarchy

Pascal

```
TCommonDialogAction = class(TCustomAction);
```

C++

```
class TCommonDialogAction : public TCustomAction;
```

File

StdActns

Description

TCommonDialogAction implements the common behavior for actions that display a dialog when executed. Each descendant class represents a specific dialog. Do not use TCommonDialogAction directly in an application. Instead, use the specific TCommonDialogAction descendant that represents the type of dialog you want to execute. The various TCommonDialogAction descendants are listed in the See also list.

TCommonDialogAction objects do not require a specific type of control to be active: they handle any target. When executed, the dialog action displays its dialog, and then generates an OnAccept (see page 1790) or OnCancel (see page 1791) event, which you can use to process the results from the dialog.

When using TCommonDialogAction desendants, there is no need to place the dialog component in your form or data module. The action object creates its own dialog component.

See Also

TColorSelect (see page 1774)

TFileOpen (see page 1817)

TFilePrintSetup (see page 1845)

TFileSaveAs (see page 1855)

TSearchFind (see page 1899)

TSearchReplace (see page 1921)

TFontEdit (see page 1864)

TOpenPicture

TSavePicture

28.1.3.1 TCommonDialogAction Members

The following tables list the members exposed by TCommonDialogAction.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1788)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 1789)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	BeforeExecute (see page 1790)	Occurs immediately before the dialog appears.
	OnAccept (see page 1790)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1791)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Public Methods

	Name	Description
	Create (see page 1785)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1786)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

28.1.3.2 TCommonDialogAction.Create Constructor

Creates an instance of TCommonDialogAction.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCommonDialogAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TCommonDialogAction constructor because actions are added at design time to an action list or action manager.

The TCommonDialogAction constructor creates an instance of its associated dialog, which descendant classes make available through the Dialog property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TCommonDialogAction

instance.

See Also

Owner

Actions

28.1.3.3 TCommonDialogAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCommonDialogAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create

Actions

28.1.3.4 TCommonDialogAction Fields

The fields of the TCommonDialogAction class are listed here.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

28.1.3.4.1 TCommonDialogAction.FDialog Field

Pascal

```
FDialog: TCommonDialog;
```

C++

```
TCommonDialog FDialog;
```

Description

This is FDialog, a member of class TCommonDialogAction.

28.1.3.5 TCommonDialogAction Methods

The methods of the TCommonDialogAction class are listed here.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1788)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 1789)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

28.1.3.5.1 TCommonDialogAction.DoAccept Method

Pascal

```
procedure DoAccept; virtual;
```

C++

```
virtual __fastcall DoAccept();
```

Description

This is DoAccept, a member of class TCommonDialogAction.

28.1.3.5.2 TCommonDialogAction.DoCancel Method

Pascal

```
procedure DoCancel; virtual;
```

C++

```
virtual __fastcall DoCancel();
```

Description

This is DoCancel, a member of class TCommonDialogAction.

28.1.3.5.3 TCommonDialogAction.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute (see page 1790) event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept (see page 1790) event or an OnCancel (see page 1791) event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Execute

BeforeExecute (see page 1790)

28.1.3.5.4 TCommonDialogAction.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; virtual;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TCommonDialogAction.

28.1.3.5.5 TCommonDialogAction.HandlesTarget Method

Indicates whether the action can execute its dialog.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true, because the execution of the dialog does not depend on any target object.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Execute

28.1.3.5.6 TCommonDialogAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

28.1.3.5.7 TCommonDialogAction.SetupDialog Method

Pascal

```
procedure SetupDialog; virtual;
```

C++

```
virtual __fastcall SetupDialog();
```

Description

This is SetupDialog, a member of class TCommonDialogAction.

28.1.3.6 TCommonDialogAction Properties

The properties of the TCommonDialogAction class are listed here.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

28.1.3.6.1 TCommonDialogAction.ExecuteResult Property

Indicates the result of executing the associated dialog.

Pascal

```
property ExecuteResult: Boolean;
```

C++

```
__property Boolean ExecuteResult;
```

Description

When the dialog action fires, it executes the dialog specified by the CommonDialog property and sets ExecuteResult the value returned by that dialog's Execute method. Thus, ExecuteResult is false if the dialog has not executed or the user canceled when

the dialog executed, and true if the user exited the dialog by clicking OK.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Execute

28.1.3.7 TCommonDialogAction Events

The events of the TCommonDialogAction class are listed here.

Public Events

	Name	Description
	BeforeExecute (see page 1790)	Occurs immediately before the dialog appears.
	OnAccept (see page 1790)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1791)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

28.1.3.7.1 TCommonDialogAction.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write (see page 3707) a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.3.7.2 TCommonDialogAction.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write (see page 3707) an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is

modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction (see page 1784) descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.3.7.3 TCommonDialogAction.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write (see page 3707) an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.4 TEditAction Class

TEditAction is the base class for action objects that perform standard edits on an edit control.

Class Hierarchy

Pascal

```
TEditAction = class(TAction);
```

C++

```
class TEditAction : public TAction;
```

File

StdActns

Description

TEditAction is the common base class for actions that work with edit controls. Each descendant class performs a specific type of edit on the target edit control. Do not use TEditAction directly in an application. Instead, use a TEditAction descendant that performs a specific function (for example, Cut, Copy (see page 3547), Paste, Delete (see page 3552), Select All, or Undo). The various TEditAction descendants are listed in the See also list.

See Also

TEditCut (see page 1797)

TEditDelete (see page 1799)

TEditPaste (see page 1801)

TEditSelectAll (see page 1803)

TEditUndo (see page 1805)

28.1.4.1 TEditAction Members

The following tables list the members exposed by TEditAction.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.
	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Destroy (see page 1792)	Frees an instance of TCustomAction.
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1794)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.4.2 TEditAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEditAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create

Actions

28.1.4.3 TEditAction Methods

The methods of the TEditAction class are listed here.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.
	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1794)	Enables or disables the action, depending on the currently active control in the form.

28.1.4.3.1 TEditAction.GetControl Method

Returns the target of the edit action.

Pascal

```
function GetControl(Target: TObject): TCustomEdit; virtual;
```

C++

```
virtual __fastcall TCustomEdit GetControl(TObject * Target);
```

Description

TEditAction (see page 1791) uses the protected GetControl method internally to obtain the target of the action. `Target` is the currently active object or the value of the `Control (see page 1795)` property. GetControl returns the target object as an edit control.

See Also

Notification (see page 1794)

28.1.4.3.2 TEditAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the `Target` parameter as a “target”. `Target` is the currently active object. HandlesTarget checks whether the `Target` is a TCustomEdit (see page 2345) or any subclass of

TCustomEdit (see page 2345). If the `Target` is nil (Delphi) or NULL (C++), HandlesTarget checks whether the Control (see page 1795) property of this action is a TCustomEdit (see page 2345) or any subclass of TCustomEdit (see page 2345). HandlesTarget returns True if there is an edit control to act as target, or False otherwise.

See Also

UpdateTarget (see page 1794)

28.1.4.3.3 TEditAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

28.1.4.3.4 TEditAction.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly. In TEditAction (see page 1791), UpdateTarget checks whether the current target object (specified by the `Target` parameter) is an edit control and that some of its text is selected (if the action function requires it). If so, UpdateTarget sets Enabled to True. If not, UpdateTarget sets Enabled to False.

See Also

HandlesTarget (see page 1793)

28.1.4.4 TEditAction Properties

The properties of the TEditAction class are listed here.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.4.4.1 TEditAction.Control Property

Provides access to the target of the action.

Pascal

```
property Control: TCustomEdit;
```

C++

```
__property TCustomEdit Control;
```

Description

Control is the edit control on which the action performs its function. Set the value of Control to limit the action so that it only operates on a specific edit control. If Control is unassigned, the action dynamically locates the target edit control when it executes.

See Also

GetControl (see page 1793)

28.1.5 TEditCopy Class

TEditCopy copies the selected text in a target edit control to the Windows clipboard.

Class Hierarchy

Pascal

```
TEditCopy = class(TEditAction);
```

C++

```
class TEditCopy : public TEditAction;
```

File

StdActns

Description

Add TEditCopy to an action list to allow users to copy selected text to the clipboard using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to copy the currently selected text in an active text control to the clipboard.

TEditCopy is designed to work with an edit control (TCustomEdit (see page 2345) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if that edit control does not have any text selected.

See Also

TEditCut (see page 1797)

TEditDelete (see page 1799)

TEditPaste (see page 1801)

TEditSelectAll (see page 1803)

TEditUndo (see page 1805)

TCustomEdit (see page 2345)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1921)

28.1.5.1 TEditCopy Members

The following tables list the members exposed by TEditCopy.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.
	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Destroy (see page 1792)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1796)	Copies selected text in the target control to the Windows clipboard.
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1794)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.5.2 TEditCopy Methods

The methods of the TEditCopy class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1796)	Copies selected text in the target control to the Windows clipboard.

28.1.5.2.1 TEditCopy.ExecuteTarget Method

Copies selected text in the target control to the Windows clipboard.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for

example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is an edit control (TCustomEdit (see page 2345) descendant), and if so, calls its CopyToClipboard method, copying the selected text to the clipboard.

See Also

TCustomEdit (see page 2345)

CopyToClipboard (see page 2349)

28.1.6 TEditCut Class

TEditCut cuts the selected text in a target edit or memo control to the Windows clipboard.

Class Hierarchy

Pascal

```
TEditCut = class(TEditAction);
```

C++

```
class TEditCut : public TEditAction;
```

File

StdActns

Description

Add TEditCut to an action list to allow users to cut selected text to the clipboard using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to cut the currently selected text in an active text control to the clipboard.

TEditCut is designed to work with an edit control (TCustomEdit (see page 2345) or TCustomMemo (see page 2443) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if that edit control does not have a selection to cut.

See Also

TEditCopy (see page 1795)

TEditDelete (see page 1799)

TEditPaste (see page 1801)

TEditSelectAll (see page 1803)

TEditUndo (see page 1805)

TCustomEdit (see page 2345)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1921)

28.1.6.1 TEditCut Members

The following tables list the members exposed by TEditCut.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.
	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Destroy (see page 1792)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1798)	Cuts selected text in the target control to the Windows clipboard.
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1794)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.6.2 TEditCut Methods

The methods of the TEditCut class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1798)	Cuts selected text in the target control to the Windows clipboard.

28.1.6.2.1 TEditCut.ExecuteTarget Method

Cuts selected text in the target control to the Windows clipboard.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is an edit control (TCustomEditControl descendant), and if so, calls its CutToClipboard method, cutting the selected text to the clipboard.

See Also

TCustomEdit (see page 2345)

CutToClipboard (see page 2352)

28.1.7 TEditDelete Class

TEditDelete deletes the selected text in a target edit or memo control.

Class Hierarchy

Pascal

```
TEditDelete = class(TEditAction);
```

C++

```
class TEditDelete : public TEditAction;
```

File

StdActns

Description

Add TEditDelete to an action list to let users delete selected text using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to delete the currently selected text in an active text control.

TEditDelete is designed to work with an edit control (TCustomEdit (see page 2345) or TCustomMemo (see page 2443) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if the active edit control does not have any selected text.

See Also

TEditCopy (see page 1795)

TEditCut (see page 1797)

TEditPaste (see page 1801)

TEditSelectAll (see page 1803)

TEditUndo (see page 1805)

TCustomEdit (see page 2345)

TCustomMemo (see page 2443)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1921)

28.1.7.1 TEditDelete Members

The following tables list the members exposed by TEditDelete.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.

	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Destroy (see page 1792)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1800)	Deletes selected text in the target control.
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1800)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.7.2 TEditDelete Methods

The methods of the TEditDelete class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1800)	Deletes selected text in the target control.
	UpdateTarget (see page 1800)	Enables or disables the action, depending on the currently active control in the form.

28.1.7.2.1 TEditDelete.ExecuteTarget Method

Deletes selected text in the target control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2345) descendant, and if so, calls its ClearSelection method, deleting the selected text.

See Also

TCustomEdit (see page 2345)

ClearSelection (see page 2349)

28.1.7.2.2 TEditDelete.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TEditDelete (see page 1799), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control that has some of its text selected. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

28.1.8 TEditPaste Class

TEditPaste pastes text from the clipboard into the active edit or memo control.

Class Hierarchy

Pascal

```
TEditPaste = class(TEditAction);
```

C++

```
class TEditPaste : public TEditAction;
```

File

StdActns

Description

Add TEditPaste to an action list to enable users to paste text from the clipboard using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to paste text from the clipboard into an active text control.

TEditPaste is designed to work with an edit control (TCustomEdit (see page 2345) or TCustomMemo (see page 2443) descendant) as a target. It automatically disables itself if the clipboard does not contain text.

See Also

TEditCopy (see page 1795)

TEditCut (see page 1797)

TEditDelete (see page 1799)

TEditSelectAll (see page 1803)

TEditUndo (see page 1805)

TCustomEdit (see page 2345)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1921)

28.1.8.1 TEditPaste Members

The following tables list the members exposed by TEditPaste.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.
	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Destroy (see page 1792)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1802)	Pastes text from the clipboard into the target edit control.
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1803)	Enables or disables the action, depending on the contents of the clipboard.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.8.2 TEditPaste Methods

The methods of the TEditPaste class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1802)	Pastes text from the clipboard into the target edit control.
	UpdateTarget (see page 1803)	Enables or disables the action, depending on the contents of the clipboard.

28.1.8.2.1 TEditPaste.ExecuteTarget Method

Pastes text from the clipboard into the target edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2345) descendant, and if so, calls its PasteFromClipboard method, pasting in any text on the clipboard.

See Also

TCustomEdit (see page 2345)

PasteFromClipboard (see page 2358)

28.1.8.2.2 TEditPaste.UpdateTarget Method

Enables or disables the action, depending on the contents of the clipboard.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditPaste (see page 1801), UpdateTarget checks whether the clipboard contains text. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

28.1.9 TEditSelectAll Class

TEditSelectAll selects all text in an active text control.

Class Hierarchy

Pascal

```
TEditSelectAll = class(TEditAction);
```

C++

```
class TEditSelectAll : public TEditAction;
```

File

StdActns

Description

Add TEditSelectAll to an action list to allow users to select all the text in a text control using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to select all text in an active text control.

TEditSelectAll is designed to work with an edit control (TCustomEdit (see page 2345) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control or if the edit control does not have any text.

See Also

TEditCopy (see page 1795)

TEditCut (see page 1797)

TEditDelete (see page 1799)

TEditPaste (see page 1801)

TEditUndo (see page 1805)

TCustomEdit (see page 2345)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1921)

28.1.9.1 TEditSelectAll Members

The following tables list the members exposed by TEditSelectAll.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.
	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Destroy (see page 1792)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1804)	Selects all text in the target edit control.
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1805)	Enables or disables the action, depending on the contents of the target edit control.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.9.2 TEditSelectAll Methods

The methods of the TEditSelectAll class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1804)	Selects all text in the target edit control.
	UpdateTarget (see page 1805)	Enables or disables the action, depending on the contents of the target edit control.

28.1.9.2.1 TEditSelectAll.ExecuteTarget Method

Selects all text in the target edit control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2345) descendant, and if so, calls its SelectAll method, selecting all of its text.

See Also

TCustomEdit (see page 2345)

SelectAll (see page 2359)

28.1.9.2.2 TEditSelectAll.UpdateTarget Method

Enables or disables the action, depending on the contents of the target edit control.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditSelectAll (see page 1803), UpdateTarget checks whether Target is a text control that contains text. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

28.1.10 TEditUndo Class

TEditUndo backs out the last change in an edit control.

Class Hierarchy

Pascal

```
TEditUndo = class(TEditAction);
```

C++

```
class TEditUndo : public TEditAction;
```

File

StdActns

Description

Add TEditUndo to an action list to allow users to back out edits using menu items or toolbar buttons. Controls such as menu

items and tool buttons linked to this action cause the application to undo the last edit in an active text control.

TEditUndo is designed to work with an edit control (TCustomEdit (see page 2345) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control that can back out the last edit.

See Also

TEditCopy (see page 1795)

TEditCut (see page 1797)

TEditPaste (see page 1801)

TEditSelectAll (see page 1803)

TEditDelete (see page 1799)

TCustomEdit (see page 2345)

TRichEditBold

TRichEditItalic

TRichEditUnderline

TSearchReplace (see page 1921)

28.1.10.1 TEditUndo Members

The following tables list the members exposed by TEditUndo.

Protected Methods

	Name	Description
	GetControl (see page 1793)	Returns the target of the edit action.
	Notification (see page 1794)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Destroy (see page 1792)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1807)	Backs out the last edit from the target control.
	HandlesTarget (see page 1793)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1807)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	Control (see page 1795)	Provides access to the target of the action.

28.1.10.2 TEditUndo Methods

The methods of the TEditUndo class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1807)	Backs out the last edit from the target control.
	UpdateTarget (see page 1807)	Enables or disables the action, depending on the currently active control in the form.

28.1.10.2.1 TEditUndo.ExecuteTarget Method

Backs out the last edit from the target control.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget checks that Target is a TCustomEdit (see page 2345) descendant, and if so, calls its Undo method, backing out the last change.

See Also

TCustomEdit (see page 2345)

Undo (see page 2364)

28.1.10.2.2 TEditUndo.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TEditUndo (see page 1805), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control that is in a state where it can back out changes. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

CanUndo (see page 2367)

28.1.11 TFileAction Class

TFileAction is the base class for action objects that display a file selection dialog.

Class Hierarchy

Pascal

```
TFileAction = class(TCommonDialogAction);
```

C++

```
class TFileAction : public TCommonDialogAction;
```

File

StdActns

Description

TFileAction implements the FileName (see page 1809) property that is common to action objects that display a file selection object.

See Also

TFileOpen (see page 1817)

TFileSaveAs (see page 1855)

TOpenDialog

28.1.11.1 TFileAction Members

The following tables list the members exposed by TFileAction.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1809)	This is GetDialog, a member of class TFileAction.
	GetDialogClass (see page 1788)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 1789)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Protected Properties

	Name	Description
	FileName (see page 1809)	Specifies the file name for the file selection dialog.

Public Events

	Name	Description
	BeforeExecute (see page 1790)	Occurs immediately before the dialog appears.
	OnAccept (see page 1790)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1791)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Public Methods

	Name	Description
	Create (see page 1785)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1786)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.

	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

28.1.11.2 TFileAction Methods

The methods of the TFileAction class are listed here.

Protected Methods

	Name	Description
	GetDialog (see page 1809)	This is GetDialog, a member of class TFileAction.

28.1.11.2.1 TFileAction.GetDialog Method

Pascal

```
function GetDialog: TOpenDialog;
```

C++

```
__fastcall TOpenDialog GetDialog();
```

Description

This is GetDialog, a member of class TFileAction.

28.1.11.3 TFileAction Properties

The properties of the TFileAction class are listed here.

Protected Properties

	Name	Description
	FileName (see page 1809)	Specifies the file name for the file selection dialog.

28.1.11.3.1 TFileAction.FileName Property

Specifies the file name for the file selection dialog.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Applications can't use the protected FileName property. To access the name of the file selected by the user, use the Dialog property of a TFileAction (see page 1807) descendant.

FileName is the same as the FileName property of the associated dialog.

See Also

FileName

OnAccept (see page 1790)

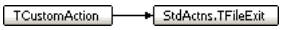
Dialog (see page 1821)

Dialog (see page 1858)

28.1.12 TFileExit Class

TFileExit is the standard action for shutting down an application.

Class Hierarchy



Pascal

```
TFileExit = class(TCustomAction);
```

C++

```
class TFileExit : public TCustomAction;
```

File

StdActns

Description

Add TFileExit to an action list to let users shut down the application using a menu or tool bar. Controls linked to this action cause the application to shut down by closing the main form.

See Also

TCustomForm

TApplication

28.1.12.1 TFileExit Members

The following tables list the members exposed by TFileExit.

Public Methods

	Name	Description
	ExecuteTarget (see page 1811)	Shuts down the main form of the application.
	HandlesTarget (see page 1811)	Indicates whether the action can execute.

Published Events

	Name	Description
	OnHint (see page 1816)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1816)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1812)	Represents the caption of client controls and menu items.
	Enabled (see page 1812)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1813)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1813)	Indicates the help keyword for client controls and menu items.

	HelpType (see page 1814)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1814)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1814)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1815)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1815)	Specifies the ShortCut property for client menu items.
	Visible (see page 1815)	Specifies the Visible property for client controls and menu items.

28.1.12.2 TFileExit Methods

The methods of the TFileExit class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1811)	Shuts down the main form of the application.
	HandlesTarget (see page 1811)	Indicates whether the action can execute.

28.1.12.2.1 TFileExit.ExecuteTarget Method

Shuts down the main form of the application.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and calls the Close (see page 3538) method of the application's main form. This causes the application to shut down.

See Also

MainForm

Close

28.1.12.2.2 TFileExit.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to

indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

Target is the currently active object.

HandlesTarget always returns true, because shutting down the application does not depend on any target object.

28.1.12.3 TFileExit Properties

The properties of the TFileExit class are listed here.

Published Properties

	Name	Description
	Caption (see page 1812)	Represents the caption of client controls and menu items.
	Enabled (see page 1812)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1813)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1813)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1814)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1814)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1814)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1815)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1815)	Specifies the ShortCut property for client menu items.
	Visible (see page 1815)	Specifies the Visible property for client controls and menu items.

28.1.12.3.1 TFileExit.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.12.3.2 TFileExit.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.12.3.3 TFileExit.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.12.3.4 TFileExit.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.12.3.5 TFileExit.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.12.3.6 TFileExit.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.12.3.7 TFileExit.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an

image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.12.3.8 TFileExit.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.12.3.9 TFileExit.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.12.3.10 TFileExit.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.12.4 TFileExit Events

The events of the TFileExit class are listed here.

Published Events

	Name	Description
	OnHint (see page 1816)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1816)	Occurs when the application is idle or when the action list updates.

28.1.12.4.1 TFileExit.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.12.4.2 TFileExit.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
```

```
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

28.1.13 TFileOpen Class

TFileOpen is the standard action for displaying a file open dialog.

Class Hierarchy

Pascal

```
TFileOpen = class(TFileAction);
```

C++

```
class TFileOpen : public TFileAction;
```

File

StdActns

Description

Add TFileOpen to an action list to add a common file open dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display a file open dialog when invoked. Use the OnAccept (see page 1826) event to respond after the user selects a file name in the file open dialog or the OnCancel (see page 1827) event to respond when the user cancels from the dialog. In the event handler for these events, you can read the file name from the dialog specified by the Dialog (see page 1821) property.

See Also

TFileOpenWith (see page 1829)

TFileSaveAs (see page 1855)

TOpenDialog

TOpenPicture

28.1.13.1 TFileOpen Members

The following tables list the members exposed by TFileOpen.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1809)	This is GetDialog, a member of class TFileAction.
	GetDialogClass (see page 1820)	This is GetDialogClass, a member of class TFileOpen.
	Notification (see page 1789)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Protected Properties

	Name	Description
	FileName (see page 1809)	Specifies the file name for the file selection dialog.

Public Methods

	Name	Description
	Create (see page 1819)	Creates an instance of TFileOpen (see page 1817).
	Destroy (see page 1786)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1820)	Executes the associated dialog and generates an OnAccept or OnCancel event.

V	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1826)	Occurs immediately before the dialog appears.
	OnAccept (see page 1826)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1827)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1827)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1828)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1821)	Represents the caption of client controls and menu items.
R	Dialog (see page 1821)	Provides access the open dialog that the action executes.
	Enabled (see page 1822)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1822)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1823)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1823)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1824)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1824)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1824)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1825)	Specifies the ShortCut property for client menu items.
	UseDefaultApp (see page 1825)	Use the default application to open the file.
	Visible (see page 1825)	Specifies the Visible property for client controls and menu items.

28.1.13.2 TFileOpen.Create Constructor

Creates an instance of TFileOpen (see page 1817).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFileOpen(TComponent * AOwner);
```

Description

Most applications do not need to call the TFileOpen (see page 1817) constructor because actions are added at design time to an action manager or action list.

The TFileOpen (see page 1817) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 1821) property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TFileOpen (see page 1817) instance.

See Also

Dialog (see page 1821)

Owner

28.1.13.3 TFileOpen Methods

The methods of the TFileOpen class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1820)	This is GetDialogClass, a member of class TFileOpen.

Public Methods

	Name	Description
	ExecuteTarget (see page 1820)	Executes the associated dialog and generates an OnAccept or OnCancel event.

28.1.13.3.1 TFileOpen.ExecuteTarget Method

Executes the associated dialog and generates an OnAccept or OnCancel event.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and

1. Generates a BeforeExecute event, so that the application can initialize the associated dialog according to current conditions.
2. Calls the Execute method of the dialog.
3. Generates an OnAccept event or an OnCancel event, depending on the value returned by the Execute method. You can write code to make use of the results from executing the dialog in these event handlers.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Execute

BeforeExecute (see page 1790)

28.1.13.3.2 TFileOpen.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFileOpen.

28.1.13.4 TFileOpen Properties

The properties of the TFileOpen class are listed here.

Published Properties

	Name	Description
	Caption (see page 1821)	Represents the caption of client controls and menu items.
R	Dialog (see page 1821)	Provides access the open dialog that the action executes.
	Enabled (see page 1822)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1822)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1823)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1823)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1824)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1824)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1824)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1825)	Specifies the ShortCut property for client menu items.
	UseDefaultApp (see page 1825)	Use the default application to open the file.
	Visible (see page 1825)	Specifies the Visible property for client controls and menu items.

28.1.13.4.1 TFileOpen.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.13.4.2 TFileOpen.Dialog Property

Provides access the open dialog that the action executes.

Pascal

```
property Dialog: TOpenDialog;
```

C++

```
__property TOpenDialog Dialog;
```

Description

Use Dialog to configure the open dialog before the action fires. For example, you can set the open dialog's default extension or initially selected file at design time using the Object Inspector. From an OnAccept (see page 1826) event handler, use Dialog to read the selected file name.

When using a file open action, you do not place a TOpenDialog component on the form or data module at design time. Instead, you add the TFileOpen (see page 1817) action to your action list and then use the Dialog property. The open dialog component is created by the TFileOpen (see page 1817) constructor.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

FileName

28.1.13.4.3 TFileOpen.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.13.4.4 TFileOpen.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.13.4.5 TFileOpen.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.13.4.6 TFileOpen.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.13.4.7 TFileOpen.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.13.4.8 TFileOpen.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.13.4.9 TFileOpen.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.13.4.10 TFileOpen.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.13.4.11 TFileOpen.UseDefaultApp Property

Use the default application to open the file.

Pascal

```
property UseDefaultApp: Boolean;
```

C++

```
__property Boolean UseDefaultApp;
```

Description

When set to True, UseDefaultApp automatically uses the default application as specified in the registry to open the file. Otherwise, TFileOpen (see page 1817) displays the FileOpen (see page 4143) dialog. The default is False.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

FileName

28.1.13.4.12 TFileOpen.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.13.5 TFileOpen Events

The events of the TFileOpen class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1826)	Occurs immediately before the dialog appears.
	OnAccept (see page 1826)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1827)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1827)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1828)	Occurs when the application is idle or when the action list updates.

28.1.13.5.1 TFileOpen.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.13.5.2 TFileOpen.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.13.5.3 TFileOpen.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.13.5.4 TFileOpen.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.13.5.5 TFileOpen.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
```

```
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

28.1.14 TFileOpenWith Class

TFileOpenWith is the standard action for displaying the dialog that lets users choose what application to user for opening a file.

Class Hierarchy

Pascal

```
TFileOpenWith = class(TFileOpen);
```

C++

```
class TFileOpenWith : public TFileOpen;
```

File

StdActns

Description

Add TFileOpenWith to an action list to add the Open With dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the Open With dialog and then use the chosen application to open the file specified by the FileName (see page 1832) property. If the FileName (see page 1832) property was not previously set, TFileOpenWith first displays an Open dialog in which the user can select a file name before displaying the Open With dialog for the user to specify what application to use when opening that file.

Note: TFileOpenWith always displays the Open With dialog, even if the specified file already has an extension with a registered association. To let users open a file by letting Windows choose the application based on the extension (or bring up the Open With dialog automatically when there is no association for the extension), use the TFileRun action instead.

You do not need to do anything to launch the selected application. TFileOpenWith automatically opens the file using the application that the user selects.

Note: The Dialog (see page 1821) property of TFileOpenWith represents the Open dialog in which a user selects the file to open, not the Open With dialog in which they choose the application to user for opening that file. Similarly, the OnAccept (see page 1826) and OnCancel (see page 1827) events do not occur after the action displays the Open With dialog – rather, they occur after the Open dialog that appears when the FileName (see page 1832) property is not preset.

See Also

TFileSaveAs (see page 1855)

TFileOpen (see page 1817)

TOpenPicture

TFileRun

28.1.14.1 TFileOpenWith Members

The following tables list the members exposed by TFileOpenWith.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1809)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 1788)	This is GetDialogClass, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1820)	This is GetDialogClass, a member of class TFileOpen.
V	Notification (see page 1789)	Responds when components are created or destroyed.
V	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	OnAccept (see page 1790)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1791)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Public Methods

	Name	Description
V	Create (see page 1785)	Creates an instance of TCommonDialogAction.
V	Create (see page 1819)	Creates an instance of TFileOpen (see page 1817).
V	Destroy (see page 1786)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1831)	Displays the Open With dialog and then opens a file with the application the user selects.
V	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	AfterOpen (see page 1834)	Occurs after an application completes opening an Open With dialog.
	BeforeExecute (see page 1834)	Occurs immediately before the dialog appears.
	OnAccept (see page 1826)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1827)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1827)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1835)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1821)	Represents the caption of client controls and menu items.
R	Dialog (see page 1821)	Provides access the open dialog that the action executes.

	Enabled (see page 1822)	Indicates whether client controls and menu items are enabled.
	FileName (see page 1832)	Specifies the file to be opened when the action executes.
	HelpContext (see page 1832)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1833)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1833)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1824)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1824)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1824)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1825)	Specifies the ShortCut property for client menu items.
	UseDefaultApp (see page 1825)	Use the default application to open the file.
	Visible (see page 1825)	Specifies the Visible property for client controls and menu items.

28.1.14.2 TFileOpenWith Methods

The methods of the TFileOpenWith class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1831)	Displays the Open With dialog and then opens a file with the application the user selects.

28.1.14.2.1 TFileOpenWith.ExecuteTarget Method

Displays the Open With dialog and then opens a file with the application the user selects.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget ignores the Target parameter and does the following:

1. If the FileName (see page 1832) property is not set, it displays the Open dialog (specified by the Dialog (see page 1821) property) to let the user select a file to open.
2. If it displays the Open dialog, it generates an OnAccept (see page 1826) or OnCancel (see page 1827) event, depending on whether the user exited the Open dialog by clicking OK or Cancel.
3. It displays the Open With dialog for the user to choose an application with which to view the file specified by FileName (see page 1832) or selected in the Open dialog.
4. It launches the selected application, displaying the specified File.

See Also

FileName (see page 1832)

Dialog (see page 1821)

OnAccept (see page 1790)

OnCancel (see page 1791)

28.1.14.3 TFileOpenWith Properties

The properties of the TFileOpenWith class are listed here.

Published Properties

	Name	Description
	FileName (see page 1832)	Specifies the file to be opened when the action executes.
	HelpContext (see page 1832)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1833)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1833)	Indicates the mechanism for client controls and menu items to use when invoking help.

28.1.14.3.1 TFileOpenWith.FileName Property

Specifies the file to be opened when the action executes.

Pascal

```
property FileName: TFileName;
```

C++

```
__property TFileName FileName;
```

Description

Set FileName to specify the file to open with the application that the user selects in the Open With dialog. If the FileName property is not set, then when the action executes, it displays an Open dialog (specified by the Dialog (see page 1821) property) in which the user can select a file before displaying the Open With dialog.

Note: When the action displays the Open dialog, it uses the selected file to set FileName. Thus, if you want to let the user select a file the next time the action executes, you must clear the FileName property in an OnAccept (see page 1826) event handler.

See Also

Dialog (see page 1821)

ExecuteTarget (see page 1831)

OnAccept (see page 1790)

28.1.14.3.2 TFileOpenWith.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated

action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.14.3.3 TFileOpenWith.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.14.3.4 TFileOpenWith.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.14.4 TFileOpenWith Events

The events of the TFileOpenWith class are listed here.

Published Events

	Name	Description
	AfterOpen (see page 1834)	Occurs after an application completes opening an Open With dialog.
	BeforeExecute (see page 1834)	Occurs immediately before the dialog appears.
	OnUpdate (see page 1835)	Occurs when the application is idle or when the action list updates.

28.1.14.4.1 TFileOpenWith.AfterOpen Event

Occurs after an application completes opening an Open With dialog.

Pascal

```
property AfterOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterOpen;
```

Description

Write (see page 3707) an AfterOpen event handler to take specific action immediately after an application opens the Open With dialog, but before the user responds to the dialog.

28.1.14.4.2 TFileOpenWith.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.14.4.3 TFileOpenWith.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
```

```
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

28.1.15 TFilePageSetup Class

TFilePageSetup is the standard action for displaying a page setup dialog.

Class Hierarchy

Pascal

```
TFilePageSetup = class(TCommonDialogAction);
```

C++

```
class TFilePageSetup : public TCommonDialogAction;
```

File

StdActns

Description

Add TFilePageSetup to an action list to add a common page setup dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the page setup dialog when invoked. Because the page setup dialog automatically configures the pages as specified when the user clicks OK, there is usually no need to write an OnAccept (see page 1843) or OnCancel (see page 1843) event handler to continue processing after the dialog shuts down.

See Also

TPrinterSetupDialog

TPrintDlg (see page 1884)

28.1.15.1 TFilePageSetup Members

The following tables list the members exposed by TFilePageSetup.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.

	GetDialogClass (see page 1837)	This is GetDialogClass, a member of class TFilePageSetup.
	Notification (see page 1789)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1785)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1786)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1842)	Occurs immediately before the dialog appears.
	OnAccept (see page 1843)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1843)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1844)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1844)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1838)	Represents the caption of client controls and menu items.
	Dialog (see page 1838)	Provides access the page setup dialog that the action executes.
	Enabled (see page 1839)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1839)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1840)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1840)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1840)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1841)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1841)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1842)	Specifies the ShortCut property for client menu items.
	Visible (see page 1842)	Specifies the Visible property for client controls and menu items.

28.1.15.2 TFilePageSetup Methods

The methods of the TFilePageSetup class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1837)	This is GetDialogClass, a member of class TFilePageSetup.

28.1.15.2.1 TFilePageSetup.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFilePageSetup.

28.1.15.3 TFilePageSetup Properties

The properties of the TFilePageSetup class are listed here.

Published Properties

	Name	Description
	Caption (see page 1838)	Represents the caption of client controls and menu items.
R	Dialog (see page 1838)	Provides access the page setup dialog that the action executes.
	Enabled (see page 1839)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1839)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1840)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1840)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1840)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1841)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1841)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1842)	Specifies the ShortCut property for client menu items.
	Visible (see page 1842)	Specifies the Visible property for client controls and menu items.

28.1.15.3.1 TFilePageSetup.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.15.3.2 TFilePageSetup.Dialog Property

Provides access the page setup dialog that the action executes.

Pascal

```
property Dialog: TPageSetupDialog;
```

C++

```
__property TPageSetupDialog Dialog;
```

Description

When using a file page setup action, you do not place a TPageSetupDialog component on the form or data module at design time. Instead, you add the TFilePageSetup (see page 1836) action to your action list. If you want to configure the dialog, for example to assign a help context, you can access the dialog component using the Dialog property. The page setup dialog component is created by the TFilePageSetup (see page 1836) constructor.

See Also

Create (see page 1785)

28.1.15.3.3 TFilePageSetup.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.15.3.4 TFilePageSetup.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.15.3.5 TFilePageSetup.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.15.3.6 TFilePageSetup.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.15.3.7 TFilePageSetup.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.15.3.8 TFilePageSetup.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.15.3.9 TFilePageSetup.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.15.3.10 TFilePageSetup.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.15.3.11 TFilePageSetup.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.15.4 TFilePageSetup Events

The events of the TFilePageSetup class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1842)	Occurs immediately before the dialog appears.
	OnAccept (see page 1843)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1843)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1844)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1844)	Occurs when the application is idle or when the action list updates.

28.1.15.4.1 TFilePageSetup.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.15.4.2 TFilePageSetup.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.15.4.3 TFilePageSetup.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.15.4.4 TFilePageSetup.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.15.4.5 TFilePageSetup.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
```

```
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

28.1.16 TFilePrintSetup Class

TFilePrintSetup is the standard action for displaying a printer setup dialog.

Class Hierarchy

Pascal

```
TFilePrintSetup = class(TCommonDialogAction);
```

C++

```
class TFilePrintSetup : public TCommonDialogAction;
```

File

StdActns

Description

Add TFilePrintSetup to an action list to add a common printer setup dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the printer setup dialog when invoked. Because the printer setup dialog automatically configures the printers as specified when the user clicks OK, there is usually no need to write an OnAccept (see page 1852) or OnCancel (see page 1853) event handler to continue processing after the dialog shuts down.

See Also

TPrinterSetupDialog

TPrintDlg (see page 1884)

28.1.16.1 TFilePrintSetup Members

The following tables list the members exposed by TFilePrintSetup.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1847)	This is GetDialogClass, a member of class TFilePrintSetup.
	Notification (see page 1789)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1785)	Creates an instance of TCommonDialogAction.
	Destroy (see page 1786)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1852)	Occurs immediately before the dialog appears.
	OnAccept (see page 1852)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1853)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1853)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1854)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1848)	Represents the caption of client controls and menu items.
R	Dialog (see page 1848)	Provides access the printer setup dialog that the action executes.
	Enabled (see page 1848)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1849)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1849)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1850)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1850)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1850)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1851)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1851)	Specifies the ShortCut property for client menu items.
	Visible (see page 1851)	Specifies the Visible property for client controls and menu items.

28.1.16.2 TFilePrintSetup Methods

The methods of the TFilePrintSetup class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 1847)	This is GetDialogClass, a member of class TFilePrintSetup.

28.1.16.2.1 TFilePrintSetup.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFilePrintSetup.

28.1.16.3 TFilePrintSetup Properties

The properties of the TFilePrintSetup class are listed here.

Published Properties

	Name	Description
	Caption (see page 1848)	Represents the caption of client controls and menu items.
R	Dialog (see page 1848)	Provides access the printer setup dialog that the action executes.
	Enabled (see page 1848)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1849)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1849)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1850)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1850)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1850)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1851)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1851)	Specifies the ShortCut property for client menu items.

	Visible (see page 1851)	Specifies the Visible property for client controls and menu items.

28.1.16.3.1 TFilePrintSetup.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.16.3.2 TFilePrintSetup.Dialog Property

Provides access the printer setup dialog that the action executes.

Pascal

```
property Dialog: TPrinterSetupDialog;
```

C++

```
__property TPrinterSetupDialog Dialog;
```

Description

When using a file print setup action, you do not place a TPrinterSetupDialog component on the form or data module at design time. Instead, you add the TFilePrintSetup (see page 1845) action to your action list. If you want to configure the dialog, for example to assign a help context, you can access the dialog component using the Dialog property. The printer setup dialog component is created by the TFilePrintSetup (see page 1845) constructor.

See Also

Create (see page 1785)

28.1.16.3.3 TFilePrintSetup.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.16.3.4 TFilePrintSetup.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.16.3.5 TFilePrintSetup.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.16.3.6 TFilePrintSetup.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.16.3.7 TFilePrintSetup.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.16.3.8 TFilePrintSetup.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an

image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.16.3.9 TFilePrintSetup.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.16.3.10 TFilePrintSetup.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.16.3.11 TFilePrintSetup.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.16.4 TFilePrintSetup Events

The events of the TFilePrintSetup class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1852)	Occurs immediately before the dialog appears.
	OnAccept (see page 1852)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1853)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1853)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1854)	Occurs when the application is idle or when the action list updates.

28.1.16.4.1 TFilePrintSetup.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.16.4.2 TFilePrintSetup.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.16.4.3 TFilePrintSetup.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.16.4.4 TFilePrintSetup.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The

OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.16.4.5 TFilePrintSetup.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
```

```
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

28.1.17 TFileSaveAs Class

TFileSaveAs is the standard action for displaying a file save as dialog.

Class Hierarchy

Pascal

```
TFileSaveAs = class(TFileAction);
```

C++

```
class TFileSaveAs : public TFileAction;
```

File

StdActns

Description

Add TFileSaveAs to an action list to add a common file save dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display a file save dialog when invoked. Use the OnAccept (see page 1862) event to respond after the user selects a file name in the file save dialog or the OnCancel (see page 1862) event to respond when the user cancels from the dialog. In the event handler for these events, you can read the file name from the dialog specified by the Dialog (see page 1858) property.

See Also

TFileOpen (see page 1817)

TSaveDialog

TSavePicture

28.1.17.1 TFileSaveAs Members

The following tables list the members exposed by TFileSaveAs.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialog (see page 1809)	This is GetDialog, a member of class TFileAction.
V	GetDialogClass (see page 1857)	This is GetDialogClass, a member of class TFileSaveAs.
V	Notification (see page 1789)	Responds when components are created or destroyed.
V	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Protected Properties

	Name	Description
	FileName (see page 1809)	Specifies the file name for the file selection dialog.

Public Methods

	Name	Description
V	Create (see page 1856)	Creates an instance of TFileSaveAs (see page 1855).
V	Destroy (see page 1786)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
V	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1861)	Occurs immediately before the dialog appears.
	OnAccept (see page 1862)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1862)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1863)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1863)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1858)	Represents the caption of client controls and menu items.
R	Dialog (see page 1858)	Provides access the save dialog that the action executes.
	Enabled (see page 1858)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1859)	Indicates the help context ID for client controls and menu items.
	Hint (see page 1859)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1860)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1860)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1860)	Specifies the ShortCut property for client menu items.
	Visible (see page 1861)	Specifies the Visible property for client controls and menu items.

28.1.17.2 TFileSaveAs.Create Constructor

Creates an instance of TFileSaveAs (see page 1855).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TFileSaveAs(TComponent * AOwner);
```

Description

Most applications do not need to call the TFileSaveAs (see page 1855) constructor because actions are added at design time to an action manager or action list.

The TFileSaveAs (see page 1855) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 1858) property.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TFileSaveAs (see page 1855) instance.

See Also

Dialog (see page 1858)

Owner

28.1.17.3 TFileSaveAs Methods

The methods of the TFileSaveAs class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1857)	This is GetDialogClass, a member of class TFileSaveAs.

28.1.17.3.1 TFileSaveAs.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFileSaveAs.

28.1.17.4 TFileSaveAs Properties

The properties of the TFileSaveAs class are listed here.

Published Properties

	Name	Description
	Caption (see page 1858)	Represents the caption of client controls and menu items.
R	Dialog (see page 1858)	Provides access the save dialog that the action executes.
	Enabled (see page 1858)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1859)	Indicates the help context ID for client controls and menu items.
	Hint (see page 1859)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1860)	Indicates the ImageIndex property for client controls and menu items.

	SecondaryShortCuts (see page 1860)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1860)	Specifies the ShortCut property for client menu items.
	Visible (see page 1861)	Specifies the Visible property for client controls and menu items.

28.1.17.4.1 TFileSaveAs.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.17.4.2 TFileSaveAs.Dialog Property

Provides access the save dialog that the action executes.

Pascal

```
property Dialog: TSaveDialog;
```

C++

```
__property TSaveDialog Dialog;
```

Description

Use Dialog to configure the save as dialog before the action fires. For example, you can set the save as dialog's default extension at design time using the Object Inspector. Dialog is also useful for reading the selected file name from an OnAccept (see page 1862) event handler.

When using a file save as action, you do not place a TSaveDialog component on the form or data module at design time. Instead, you add the TFileSaveAs (see page 1855) to your action list and then use the Dialog property. The save dialog component is created by the TFileSaveAs (see page 1855) constructor.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

28.1.17.4.3 TFileSaveAs.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.17.4.4 TFileSaveAs.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.17.4.5 TFileSaveAs.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.17.4.6 TFileSaveAs.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.17.4.7 TFileSaveAs.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.17.4.8 TFileSaveAs.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.17.4.9 TFileSaveAs.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.17.5 TFileSaveAs Events

The events of the TFileSaveAs class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1861)	Occurs immediately before the dialog appears.
	OnAccept (see page 1862)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1862)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1863)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1863)	Occurs when the application is idle or when the action list updates.

28.1.17.5.1 TFileSaveAs.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.17.5.2 TFileSaveAs.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.17.5.3 TFileSaveAs.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.17.5.4 TFileSaveAs.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.17.5.5 TFileSaveAs.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
```

```
    }
//    pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

28.1.18 TFontEdit Class

TFontEdit is the standard action for displaying a font selection dialog.

Class Hierarchy

Pascal

```
TFontEdit = class(TCommonDialogAction);
```

C++

```
class TFontEdit : public TCommonDialogAction;
```

File

StdActns

Description

Add TFontEdit to an action list to add a modal font selection dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display a font dialog when invoked. Use the OnAccept (see page 1871) event to respond after the user selects a font in the dialog or the OnCancel (see page 1872) event to respond when the user cancels from the dialog. In the event handler, you can read the selected font from the dialog specified by the Dialog (see page 1867) property.

See Also

TFontDialog

28.1.18.1 TFontEdit Members

The following tables list the members exposed by TFontEdit.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1866)	This is GetDialogClass, a member of class TFontEdit.
V	Notification (see page 1789)	Responds when components are created or destroyed.
V	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1785)	Creates an instance of TCommonDialogAction.
V	Destroy (see page 1786)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
V	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1871)	Occurs immediately before the dialog appears.
	OnAccept (see page 1871)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1872)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1872)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1872)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1866)	Represents the caption of client controls and menu items.
R	Dialog (see page 1867)	Provides access the font selection dialog that the action executes.
	Enabled (see page 1867)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1867)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1868)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1868)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1869)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1869)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1870)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1870)	Specifies the ShortCut property for client menu items.

	Visible (see page 1870)	Specifies the Visible property for client controls and menu items.

28.1.18.2 TFontEdit Methods

The methods of the TFontEdit class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 1866)	This is GetDialogClass, a member of class TFontEdit.

28.1.18.2.1 TFontEdit.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TFontEdit.

28.1.18.3 TFontEdit Properties

The properties of the TFontEdit class are listed here.

Published Properties

	Name	Description
	Caption (see page 1866)	Represents the caption of client controls and menu items.
R	Dialog (see page 1867)	Provides access the font selection dialog that the action executes.
	Enabled (see page 1867)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1867)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1868)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1868)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1869)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1869)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1870)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1870)	Specifies the ShortCut property for client menu items.
	Visible (see page 1870)	Specifies the Visible property for client controls and menu items.

28.1.18.3.1 TFontEdit.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and

menu items.

See Also

Caption

Caption

28.1.18.3.2 TFontEdit.Dialog Property

Provides access the font selection dialog that the action executes.

Pascal

```
property Dialog: TFontDialog;
```

C++

```
__property TFontDialog Dialog;
```

Description

Use Dialog to configure the font selection dialog before the action fires or to read the selected font after the action executes the dialog. For example, you can set the font dialog's target device or limit the available font sizes at design time using the Object Inspector or read the Font property in an OnAccept (see page 1871) event handler to make use of the font that the user selects.

When using a font edit action, you do not place a TFontDialog component on the form or data module at design time. Instead, you add the TFontEdit (see page 1864) to your action list and then use the Dialog property. The font selection dialog component is created by the TFontEdit (see page 1864) constructor.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Font

28.1.18.3.3 TFontEdit.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.18.3.4 TFontEdit.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.18.3.5 TFontEdit.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.18.3.6 TFontEdit.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.18.3.7 TFontEdit.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.18.3.8 TFontEdit.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.18.3.9 TFontEdit.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.18.3.10 TFontEdit.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.18.3.11 TFontEdit.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.18.4 TFontEdit Events

The events of the TFontEdit class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1871)	Occurs immediately before the dialog appears.
	OnAccept (see page 1871)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1872)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1872)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1872)	Occurs when the application is idle or when the action list updates.

28.1.18.4.1 TFontEdit.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.18.4.2 TFontEdit.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the

user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.18.4.3 TFontEdit.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.18.4.4 TFontEdit.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.18.4.5 TFontEdit.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
```

```
end;
```

See Also

UpdateAction

28.1.19 THelpAction Class

THelpAction is the base class for action objects that interact with the help system.

Class Hierarchy

Pascal

```
THelpAction = class(TAction);
```

C++

```
class THelpAction : public TAction;
```

File

StdActns

Description

THelpAction implements the common behavior for actions that work the help system. Each descendant class performs a specific help function. Do not use THelpAction directly in an application. Instead, use a THelpAction descendant such as THelpContents (see page 1876), THelpContextAction (see page 1877), THelpOnHelp (see page 1879), or THelpTopicSearch (see page 1881).

THelpAction ensures that the action is enabled as long as there is an application object to pass commands on to a registered help system. THelpAction descendants override the ExecuteTarget method to execute their specific help command.

See Also

THelpContents (see page 1876)

THelpContextAction (see page 1877)

THelpOnHelp (see page 1879)

THelpTopicSearch (see page 1881)

TApplication

28.1.19.1 THelpAction Members

The following tables list the members exposed by THelpAction.

Public Methods

	Name	Description
	Create (see page 1875)	Creates and initializes a TAction object.
	HandlesTarget (see page 1875)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1875)	Enables or disables the action, depending on the availability of the Application object.

28.1.19.2 THelpAction.Create Constructor

Creates and initializes a TAction object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THelpAction(TComponent * AOwner);
```

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time. When instantiating TAction at runtime, assign a TActionList or TActionManager component to the new action's ActionList property after calling Create.

See Also

ActionList

28.1.19.3 THelpAction Methods

The methods of the THelpAction class are listed here.

Public Methods

	Name	Description
	HandlesTarget (see page 1875)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1875)	Enables or disables the action, depending on the availability of the Application object.

28.1.19.3.1 THelpAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

As implemented in THelpAction (see page 1874), HandlesTarget ignores the Target parameter and always returns true, because the help actions do not rely on a specific target object.

28.1.19.3.2 THelpAction.UpdateTarget Method

Enables or disables the action, depending on the availability of the Application object.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In THelpAction (see page 1874), UpdateTarget checks whether the global Application variable is assigned, because this object is required to pass help commands on to WinHelp HyperHelp. If Application is assigned, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

Application

28.1.20 THelpContents Class

THelpContents brings up the contents for the default help system.

Class Hierarchy

Pascal

```
THelpContents = class(THelpAction);
```

C++

```
class THelpContents : public THelpAction;
```

File

StdActns

Description

Add THelpContents to an action list to allow users to bring up the help contents by using menu items or toolbar buttons. Controls such as menu items and tool buttons linked to this action cause the application to send the default help system a command to show the help contents.

On WinHelp, THelpContents displays the Help Topics dialog on the tab (Contents, Index or Find) that was last used.

See Also

THelpContextAction (see page 1877)

THelpOnHelp (see page 1879)

THelpTopicSearch (see page 1881)

28.1.20.1 THelpContents Members

The following tables list the members exposed by THelpContents.

Public Methods

	Name	Description
	Create (see page 1875)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1877)	Displays a dialog that lists the help topics.
	HandlesTarget (see page 1875)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1875)	Enables or disables the action, depending on the availability of the Application object.

28.1.20.2 THelpContents Methods

The methods of the THelpContents class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1877)	Displays a dialog that lists the help topics.

28.1.20.2.1 THelpContents.ExecuteTarget Method

Displays a dialog that lists the help topics.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to list the help topics.

28.1.21 THelpContextAction Class

THelpContextAction brings up the help topic for the active control.

Class Hierarchy

Pascal

```
THelpContextAction = class(THelpAction);
```

C++

```
class THelpContextAction : public THelpAction;
```

File

StdActns

Description

Add THelpContextAction to an action list to allow users to bring up the help for an active control. Controls such as menu items and tool buttons linked to this action cause the application to send the help engine a command to display the help topic specified by the active control's HelpContext property.

See Also

THelpContents (see page 1876)

THelpTopicSearch (see page 1881)

THelpOnHelp (see page 1879)

TControl

TScreen

28.1.21.1 THelpContextAction Members

The following tables list the members exposed by THelpContextAction.

Public Methods

	Name	Description
	Create (see page 1875)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1878)	Displays the help topic associated with the active control's HelpContext property.
	HandlesTarget (see page 1875)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1879)	Enables or disables the action, depending on whether there is an active object.

28.1.21.2 THelpContextAction Methods

The methods of the THelpContextAction class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1878)	Displays the help topic associated with the active control's HelpContext property.
	UpdateTarget (see page 1879)	Enables or disables the action, depending on whether there is an active object.

28.1.21.2.1 THelpContextAction.ExecuteTarget Method

Displays the help topic associated with the active control's HelpContext property.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to display the topic specified by the active control's

HelpContext property. ExecuteTarget gets the active control from the global Screen object rather than using the Target parameter.

See Also

HelpCommand

ActiveControl

HelpContext

28.1.21.2.2 THelpContextAction.UpdateTarget Method

Enables or disables the action, depending on whether there is an active object.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In THelpContextAction (see page 1877), UpdateTarget checks whether the global Screen object's ActiveConrol property is assigned. If there is an active control, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

See Also

Enabled

ActiveControl

28.1.22 THelpOnHelp Class

THelpOnHelp brings up a help topic on how to use Help.

Class Hierarchy

Pascal

```
THelpOnHelp = class(THelpAction);
```

C++

```
class THelpOnHelp : public THelpAction;
```

File

StdActns

Description

Add THelpOnHelp to an action list to allow users to bring up the help system that describes how to use help. Controls such as menu items and tool buttons linked to this action cause the application to send the default help system a command to display its help.

Note that on recent versions of Windows with a default help system of WinHelp, this brings up an HTML help file that does not describe the WinHelp system.

See Also

THelpContents (see page 1876)

THelpContextAction (see page 1877)

THelpTopicSearch (see page 1881)

28.1.22.1 THelpOnHelp Members

The following tables list the members exposed by THelpOnHelp.

Public Methods

	Name	Description
	Create (see page 1875)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1880)	Brings up the help system's help file.
	HandlesTarget (see page 1875)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1875)	Enables or disables the action, depending on the availability of the Application object.

28.1.22.2 THelpOnHelp Methods

The methods of the THelpOnHelp class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1880)	Brings up the help system's help file.

28.1.22.2.1 THelpOnHelp.ExecuteTarget Method

Brings up the help system's help file.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to display the Help system's own help file.

See Also

HelpCommand

28.1.23 THelpTopicSearch Class

THelpTopicSearch brings up the help index.

Class Hierarchy

Pascal

```
THelpTopicSearch = class(THelpAction);
```

C++

```
class THelpTopicSearch : public THelpAction;
```

File

StdActns

Description

Add THelpTopicSearch to an action list to allow users to bring up the help index. Controls such as menu items and tool buttons linked to this action cause the application to send the default help system a command to display the help index.

See Also

THelpContextAction (see page 1877)

THelpOnHelp (see page 1879)

28.1.23.1 THelpTopicSearch Members

The following tables list the members exposed by THelpTopicSearch.

Public Methods

	Name	Description
	Create (see page 1875)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1881)	Displays the Help Topics dialog with the index tab selected.
	HandlesTarget (see page 1875)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1875)	Enables or disables the action, depending on the availability of the Application object.

28.1.23.2 THelpTopicSearch Methods

The methods of the THelpTopicSearch class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1881)	Displays the Help Topics dialog with the index tab selected.

28.1.23.2.1 THelpTopicSearch.ExecuteTarget Method

Displays the Help Topics dialog with the index tab selected.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

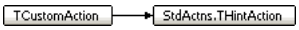
Target is the currently active object when the action fires.

ExecuteTarget calls the application's HelpCommand method, telling it to display the Help index.

28.1.24 THintAction Class

THintAction is the action that executes when the user pauses the mouse over a control or menu item that has a hint.

Class Hierarchy



Pascal

```
THintAction = class(TCustomAction);
```

C++

```
class THintAction : public TCustomAction;
```

File

StdActns

Description

Do not use THintAction in an action manager or action list the way you would use other predefined actions. THintAction is not intended to work with a client control. Rather, it is used by the application object.

When the user pauses the mouse over a control or menu item that defines its Hint (see page 1884) property, the application responds in one of two ways:

If the application has an OnHint event handler, it generates an OnHint event.

If there is no OnHint event handler, the application executes a THintAction object.

When THintAction executes, it calls the ExecuteAction method of potential target controls until a control is found to respond to the action. Some controls, such as TStatusBar, override the ExecuteAction method to respond to THintAction.

Component writers can override the ExecuteAction method of a new component to respond to THintAction by displaying hint messages. Such an ExecuteAction method might look like the following:

```
function TMyControl.ExecuteAction(Action: TBasicAction): Boolean;
var
  HintText: string;
begin
if Action is THintAction then
begin
    HintText := THintAction(Action).Hint;
    { do something to display the hint text }
    Result := True;
```

```
end
else Result := inherited ExecuteAction(Action);
end;
bool __fastcall TMyControl::ExecuteAction(TBasicAction *Action)
{
  THintAction *pHint = dynamic_cast<THintAction *>(Action);
if (pHint)
  {
    // do something to display pHint->Hint
returntrue;
  }
return TComponent::ExecuteAction(Action);
}
```

See Also

TControl

TMenuItem

TComponent

TApplication

28.1.24.1 THintAction Members

The following tables list the members exposed by THintAction.

Public Methods

	Name	Description
	Create (see page 1883)	Creates an instance of THintAction (see page 1882).

Published Properties

	Name	Description
	Hint (see page 1884)	Indicates the Help hint for client controls and menu items.

28.1.24.2 THintAction.Create Constructor

Creates an instance of THintAction (see page 1882).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THintAction(TComponent * AOwner);
```

Description

Typically, you do not need to create THintAction (see page 1882) instances. THintAction (see page 1882) is created by the application object when it detects the need to display a hint.

You can, however, create a THintAction (see page 1882) object, set its Hint (see page 1884) property, and then call its Execute method to cause other controls in the application (such as status bars) to display a message for some application-generated condition (such as a status message).

AOwner is the component, typically the application object, that is responsible for freeing the THintAction (see page 1882) object.

See Also

Owner

Hint

Execute

28.1.24.3 THintAction Properties

The properties of the THintAction class are listed here.

Published Properties

	Name	Description
	Hint (see page 1884)	Indicates the Help hint for client controls and menu items.

28.1.24.3.1 THintAction.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.25 TPrintDlg Class

TPrintDlg is the standard action for displaying a print dialog.

Class Hierarchy

Pascal

```
TPrintDlg = class(TCommonDialogAction);
```

C++

```
class TPrintDlg : public TCommonDialogAction;
```

File

StdActns

Description

Add TPrintDlg to an action list to add a print dialog to your application. Controls such as menu items and tool buttons linked to

this action cause the application to display the print dialog (TPrintDialog) when invoked. Write (see page 3707) an OnAccept (see page 1891) event handler to perform the actual printing when the user clicks OK. You can read details about the user's selections in the dialog from the Dialog (see page 1887) property.

See Also

TPrintDialog

TFilePrintSetup (see page 1845)

28.1.25.1 TPrintDlg Members

The following tables list the members exposed by TPrintDlg.

Protected Fields

	Name	Description
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.

Protected Methods

	Name	Description
V	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
V	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
V	GetDialogClass (see page 1886)	This is GetDialogClass, a member of class TPrintDlg.
V	Notification (see page 1789)	Responds when components are created or destroyed.
V	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
V	Create (see page 1785)	Creates an instance of TCommonDialogAction.
V	Destroy (see page 1786)	Frees an instance of TCustomAction.
V	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
V	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.

Public Properties

	Name	Description
R	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1891)	Occurs immediately before the dialog appears.
	OnAccept (see page 1891)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1892)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1892)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1893)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1886)	Represents the caption of client controls and menu items.
R	Dialog (see page 1887)	Provides access the print dialog that the action executes.
	Enabled (see page 1887)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1888)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1888)	Indicates the help keyword for client controls and menu items.

	HelpType (see page 1889)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1889)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1889)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1890)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1890)	Specifies the ShortCut property for client menu items.
	Visible (see page 1890)	Specifies the Visible property for client controls and menu items.

28.1.25.2 TPrintDlg Methods

The methods of the TPrintDlg class are listed here.

Protected Methods

	Name	Description
V	GetDialogClass (see page 1886)	This is GetDialogClass, a member of class TPrintDlg.

28.1.25.2.1 TPrintDlg.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TPrintDlg.

28.1.25.3 TPrintDlg Properties

The properties of the TPrintDlg class are listed here.

Published Properties

	Name	Description
	Caption (see page 1886)	Represents the caption of client controls and menu items.
R	Dialog (see page 1887)	Provides access the print dialog that the action executes.
	Enabled (see page 1887)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1888)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1888)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1889)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1889)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1889)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1890)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1890)	Specifies the ShortCut property for client menu items.
	Visible (see page 1890)	Specifies the Visible property for client controls and menu items.

28.1.25.3.1 TPrintDlg.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.25.3.2 TPrintDlg.Dialog Property

Provides access the print dialog that the action executes.

Pascal

```
property Dialog: TPrintDialog;
```

C++

```
__property TPrintDialog Dialog;
```

Description

When using a print dialog action, you do not place a TPrintDialog component on the form or data module at design time. Instead, you add the TPrintDlg (see page 1884) action to your action list. If you want to configure the dialog, for example to set the Options that are available to users, you can access the dialog component using the Dialog property. From an OnAccept (see page 1891) event handler, use Dialog to read the choices that the user made in the print dialog and then perform the printing based on those choices.

The print dialog component is created by the TPrintDlg (see page 1884) constructor.

See Also

Create (see page 1785)

Options

OnAccept (see page 1790)

28.1.25.3.3 TPrintDlg.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.25.3.4 TPrintDlg.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.25.3.5 TPrintDlg.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.25.3.6 TPrintDlg.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.25.3.7 TPrintDlg.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.25.3.8 TPrintDlg.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.25.3.9 TPrintDlg.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.25.3.10 TPrintDlg.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.25.3.11 TPrintDlg.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.25.4 TPrintDlg Events

The events of the TPrintDlg class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1891)	Occurs immediately before the dialog appears.
	OnAccept (see page 1891)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1892)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1892)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1893)	Occurs when the application is idle or when the action list updates.

28.1.25.4.1 TPrintDlg.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.25.4.2 TPrintDlg.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.25.4.3 TPrintDlg.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.25.4.4 TPrintDlg.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.25.4.5 TPrintDlg.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
```

```
  Then set an object's (such as a button or menu item) Action
  to Action1.
  }
  procedure TMainForm.Action1Update(Sender: TObject);
  begin
    if ((Sender as TAction).Checked <> ToolBar1.Visible) then
      lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
      { Indicate whether ToolBar1 is currently visible }
      (Sender as TAction).Checked := ToolBar1.Visible;
  end;
```

See Also

UpdateAction

28.1.26 TSearchAction Class

TSearchAction is the base class for action objects that display a search or search and replace dialog.

Class Hierarchy

Pascal

```
TSearchAction = class(TCommonDialogAction);
```

C++

```
class TSearchAction : public TCommonDialogAction;
```

File

StdActns

Description

TSearchAction implements the common behavior for actions that display a modeless dialog where the user can enter a search string for searching an edit control. Do not use TSearchAction directly in an application. Instead, use one of the TSearchAction descendants: TSearchFind (see page 1899) or TSearchFindFirst (see page 1909) for a simple locate dialog, TSearchReplace (see page 1921) for a locate-and-replace dialog.

TSearchAction descendants are designed to work with an edit control (TCustomEdit (see page 2345) descendant) as a target. They automatically disable themselves if the active control in a form is not an edit control.

Unlike other dialog actions, search actions represent a modeless dialog. As a result, the OnAccept (see page 1790) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for or replace a string, the search action automatically performs that search, selecting or replacing text in the target edit control.

See Also

TSearchFind (see page 1899)

TSearchReplace (see page 1921)

TSearchFindNext (see page 1912)

TCustomEdit (see page 2345)

TSearchFindFirst (see page 1909)

28.1.26.1 TSearchAction Members

The following tables list the members exposed by TSearchAction.

Protected Fields

	Name	Description
	FControl (see page 1896)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1897)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1788)	This is GetDialogClass, a member of class TCommonDialogAction.
	Notification (see page 1898)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	BeforeExecute (see page 1790)	Occurs immediately before the dialog appears.
	OnAccept (see page 1790)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1791)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Public Methods

	Name	Description
	Create (see page 1895)	Creates and initializes an instance of TSearchAction (see page 1894).
	Destroy (see page 1896)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1897)	Launches the associated dialog.
	HandlesTarget (see page 1898)	Indicates whether the action can execute its dialog.
	Search (see page 1898)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1899)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

28.1.26.2 TSearchAction.Create Constructor

Creates and initializes an instance of TSearchAction (see page 1894).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSearchAction(TComponent * AOwner);
```

Description

Most applications do not need to call the TSearchAction (see page 1894) constructor because actions are added at design time to action lists or action managers.

The TSearchAction (see page 1894) constructor creates an instance of its associated dialog, which descendant classes surface as the Dialog property. It then assigns the Search (see page 1898) method as an event handler for the dialog's OnFind event, so that the action responds automatically when user's direct the dialog to search for a specified text string.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchAction (see page 1894) instance.

See Also

Dialog (see page 1902)

Dialog (see page 1925)

Owner

Search (see page 1898)

OnFind

28.1.26.3 TSearchAction.Destroy Destructor

Frees an instance of TCustomAction.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSearchAction();
```

Description

Do not call the destructor directly in an application. Instead, call Free. Free ensures that the action is not nil before calling the destructor.

If the action is contained in an action list or action manager, Destroy removes the action from that list.

See Also

Create

Actions

28.1.26.4 TSearchAction Fields

The fields of the TSearchAction class are listed here.

Protected Fields

	Name	Description
	FControl (see page 1896)	This is FControl, a member of class TSearchAction.
	FFindFirst (see page 1897)	This is FFindFirst, a member of class TSearchAction.

28.1.26.4.1 TSearchAction.FControl Field

Pascal

```
FControl: TCustomEdit;
```

C++

```
TCustomEdit FControl;
```

Description

This is FControl, a member of class TSearchAction.

28.1.26.4.2 TSearchAction.FFindFirst Field

Pascal

```
FFindFirst: Boolean;
```

C++

```
Boolean FFindFirst;
```

Description

This is FFindFirst, a member of class TSearchAction.

28.1.26.5 TSearchAction Methods

The methods of the TSearchAction class are listed here.

Protected Methods

	Name	Description
	Notification (see page 1898)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	ExecuteTarget (see page 1897)	Launches the associated dialog.
	HandlesTarget (see page 1898)	Indicates whether the action can execute its dialog.
	Search (see page 1898)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1899)	Enables or disables the action, depending on the currently active control in the form.

28.1.26.5.1 TSearchAction.ExecuteTarget Method

Launches the associated dialog.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget first launches its search or search and replace dialog. This is a modeless dialog, so the result of the dialog's Execute method simply indicates whether the dialog successfully appeared or was brought into focus. This result is set as the value of the ExecuteResult (see page 1789) property, and an OnAccept (see page 1790) or OnCancel (see page 1791) event occurs, depending on whether the dialog was brought into focus.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Execute

ExecuteResult (see page 1789)

28.1.26.5.2 TSearchAction.HandlesTarget Method

Indicates whether the action can execute its dialog.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method lets the action object indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

HandlesTarget returns true if the target is an edit control that has keyboard focus.

28.1.26.5.3 TSearchAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

28.1.26.5.4 TSearchAction.Search Method

Selects the next occurrence of the search string in the target edit control.

Pascal

```
procedure Search(Sender: TObject); virtual;
```

C++

```
virtual __fastcall Search(TObject * Sender);
```

Description

Most applications have no need to call the Search method: the TSearchAction (see page 1894) constructor automatically assigns Search as the OnFind event handler of its associated dialog. The dialog's OnFind event occurs when the user directs the dialog to find a specified string.

Search searches the target edit control for the string that is the FindText property of the search dialog, using the options specified by the dialog's Options property. If it finds the specified string, it selects it in the edit control. Otherwise, it displays a message to the user indicating that the string was not found.

Note: If the search action is an instance of TSearchFind (see page 1899), Search looks for the first occurrence of FindText after the current selection in the edit control. If the search action is an instance of TSearchFindFirst (see page 1909), Search looks for the first occurrence of FindText, regardless of the current selection.

See Also

OnFind

Options

FindText

28.1.26.5.5 TSearchAction.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the application is idle so that actions can update themselves according to current conditions.

In TSearchAction (see page 1894), UpdateTarget checks whether the current target object (specified by the Target parameter) is an edit control. If so, UpdateTarget sets Enabled to true. If not, UpdateTarget sets Enabled to false.

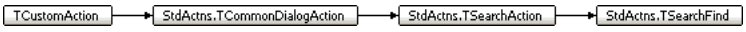
See Also

Enabled

28.1.27 TSearchFind Class

TSearchFind is the standard action for displaying a find dialog.

Class Hierarchy



Pascal

```
TSearchFind = class(TSearchAction);
```

C++

```
class TSearchFind : public TSearchAction;
```

File

StdActns

Description

Add TSearchFind to an action list to add a modeless find dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the find dialog or bring it to the fore.

TSearchFind is designed to work with an edit control (TCustomEdit (see page 2345) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 1898) actions represent a modeless dialog. As a result, the OnAccept (see page 1907) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for a string, TSearchFind automatically performs that search, selecting the specified text in the target edit control.

See Also

TFindDialog

TSearchReplace (see page 1921)

TSearchFindNext (see page 1912)

TCustomEdit (see page 2345)

TSearchFindFirst (see page 1909)

28.1.27.1 TSearchFind Members

The following tables list the members exposed by TSearchFind.

Protected Fields

	Name	Description
	FControl (see page 1896)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1897)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1901)	This is GetDialogClass, a member of class TSearchFind.
	Notification (see page 1789)	Responds when components are created or destroyed.
	Notification (see page 1898)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1785)	Creates an instance of TCommonDialogAction.
	Create (see page 1895)	Creates and initializes an instance of TSearchAction (see page 1894).
	Destroy (see page 1786)	Frees an instance of TCustomAction.
	Destroy (see page 1896)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
	ExecuteTarget (see page 1897)	Launches the associated dialog.
	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.
	HandlesTarget (see page 1898)	Indicates whether the action can execute its dialog.

	Search (see page 1898)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1899)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1906)	Occurs immediately before the dialog appears.
	OnAccept (see page 1907)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1907)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1908)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1908)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1902)	Represents the caption of client controls and menu items.
	Dialog (see page 1902)	Provides access the find dialog that the action executes.
	Enabled (see page 1903)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1903)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1903)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1904)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1904)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1905)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1905)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1905)	Specifies the ShortCut property for client menu items.
	Visible (see page 1906)	Specifies the Visible property for client controls and menu items.

28.1.27.2 TSearchFind Methods

The methods of the TSearchFind class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1901)	This is GetDialogClass, a member of class TSearchFind.

28.1.27.2.1 TSearchFind.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSearchFind.

28.1.27.3 TSearchFind Properties

The properties of the TSearchFind class are listed here.

Published Properties

	Name	Description
	Caption (see page 1902)	Represents the caption of client controls and menu items.
R	Dialog (see page 1902)	Provides access the find dialog that the action executes.
	Enabled (see page 1903)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1903)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1903)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1904)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1904)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1905)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1905)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1905)	Specifies the ShortCut property for client menu items.
	Visible (see page 1906)	Specifies the Visible property for client controls and menu items.

28.1.27.3.1 TSearchFind.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.27.3.2 TSearchFind.Dialog Property

Provides access the find dialog that the action executes.

Pascal

```
property Dialog: TFindDialog;
```

C++

```
__property TFindDialog Dialog;
```

Description

Use Dialog to configure the find dialog before the action fires or to close the modeless dialog programmatically. For example, you can set the find dialog's options at design time using the Object Inspector or call the dialog's CloseDialog method to close it.

When using a search-find action, you do not place a TFindDialog component on the form or data module at design time. Instead, you add the TSearchFind (see page 1899) action to your action list and then use the Dialog property. The find dialog

component is created by the TSearchFind (see page 1899) constructor.

See Also

ExecuteTarget (see page 1897)

CloseDialog

28.1.27.3.3 TSearchFind.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.27.3.4 TSearchFind.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.27.3.5 TSearchFind.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.27.3.6 TSearchFind.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.27.3.7 TSearchFind.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.27.3.8 TSearchFind.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.27.3.9 TSearchFind.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.27.3.10 TSearchFind.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.27.3.11 TSearchFind.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.27.4 TSearchFind Events

The events of the TSearchFind class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1906)	Occurs immediately before the dialog appears.
	OnAccept (see page 1907)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1907)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1908)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1908)	Occurs when the application is idle or when the action list updates.

28.1.27.4.1 TSearchFind.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.27.4.2 TSearchFind.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.27.4.3 TSearchFind.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel

occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.27.4.4 TSearchFind.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.27.4.5 TSearchFind.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
```

```
    // Indicate whether ToolBar1 is currently visible.
    TAction* pa = dynamic_cast<TAction*>(Sender);
    if (pa)
    {
      if (checked != ToolBar1->Visible)
      {
        lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
      }
      checked = ToolBar1->Visible;
    }
//    pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

28.1.28 TSearchFindFirst Class

TSearchFindFirst is the standard action for displaying a find dialog that looks for the first occurrence of a string.

Class Hierarchy

Pascal

```
TSearchFindFirst = class(TSearchFind);
```

C++

```
class TSearchFindFirst : public TSearchFind;
```

File

StdActns

Description

Add TSearchFindFirst to an action list to add a modeless find dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the find dialog or bring it to the fore.

TSearchFindFirst is designed to work with an edit control (TCustomEdit (see page 2345) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 1898) actions represent a modeless dialog. As a result, the OnAccept (see page 1907) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to search for a string, TSearchFindFirst automatically performs that search, selecting the specified text in the target edit control. Unlike the TSearchFind (see page 1899) action, which locates the first occurrence of the search string following the cursor position in the edit control, TSearchFindFirst locates the first occurrence of the search string, regardless of the cursor position.

See Also

TFindDialog

TSearchReplace (see page 1921)

TSearchFindNext (see page 1912)

TCustomEdit (see page 2345)

TSearchFind (see page 1899)

28.1.28.1 TSearchFindFirst Members

The following tables list the members exposed by TSearchFindFirst.

Protected Fields

	Name	Description
	FControl (see page 1896)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1897)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1788)	This is GetDialogClass, a member of class TCommonDialogAction.
	GetDialogClass (see page 1901)	This is GetDialogClass, a member of class TSearchFind.
	Notification (see page 1789)	Responds when components are created or destroyed.
	Notification (see page 1898)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Events

	Name	Description
	BeforeExecute (see page 1790)	Occurs immediately before the dialog appears.
	OnAccept (see page 1790)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1791)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Public Methods

	Name	Description
	Create (see page 1911)	Creates and initializes an instance of TSearchFindFirst (see page 1909).

	Destroy (see page 1786)	Frees an instance of TCustomAction.
	Destroy (see page 1896)	Frees an instance of TCustomAction.
	ExecuteTarget (see page 1787)	Executes the associated dialog and generates an OnAccept (see page 1790) or OnCancel (see page 1791) event.
	ExecuteTarget (see page 1897)	Launches the associated dialog.
	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.
	HandlesTarget (see page 1898)	Indicates whether the action can execute its dialog.
	Search (see page 1898)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1899)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1906)	Occurs immediately before the dialog appears.
	OnAccept (see page 1907)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1907)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1908)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1908)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1902)	Represents the caption of client controls and menu items.
	Dialog (see page 1902)	Provides access the find dialog that the action executes.
	Enabled (see page 1903)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1903)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1903)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1904)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1904)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1905)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1905)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1905)	Specifies the ShortCut property for client menu items.
	Visible (see page 1906)	Specifies the Visible property for client controls and menu items.

28.1.28.2 TSearchFindFirst.Create Constructor

Creates and initializes an instance of TSearchFindFirst (see page 1909).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSearchFindFirst(TComponent * AOwner);
```

Description

Most applications do not need to call the TSearchFindFirst (see page 1909) constructor because actions are added at design time to an action manager or action list.

The TSearchFindFirst (see page 1909) constructor creates an instance of its associated dialog and assigns that component as the value of the Dialog (see page 1902) property. It then assigns the Search (see page 1898) method as an event handler

for the dialog's OnFind event, so that the action responds automatically when user's direct the dialog to search for a specified text string.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchFindFirst (see page 1909) instance.

See Also

Dialog (see page 1902)

Owner

Search (see page 1898)

OnFind

28.1.29 TSearchFindNext Class

TSearchFindNext is the standard action for locating the next instance of a string in an edit control.

Class Hierarchy

Pascal

```
TSearchFindNext = class(TCustomAction);
```

C++

```
class TSearchFindNext : public TCustomAction;
```

File

StdActns

Description

Add TSearchFindNext to an action list to add the ability to repeat a search initiated by a TSearchFind (see page 1899) or TSearchFindFirst (see page 1909) action. Controls such as menu items and tool buttons linked to this action cause the application to repeat the search initiated by the TSearchFind (see page 1899) action's dialog.

TSearchFindNext objects must be associated with a TSearchFind (see page 1899) action by setting the SearchFind (see page 1918) property.

See Also

TSearchFind (see page 1899)

TSearchFindFirst (see page 1909)

28.1.29.1 TSearchFindNext Members

The following tables list the members exposed by TSearchFindNext.

Public Methods

	Name	Description
	Create (see page 1913)	Creates an instance of TSearchFindNext (see page 1912).
	ExecuteTarget (see page 1914)	Searches the target edit control for the next instance of the search string.
	HandlesTarget (see page 1914)	Indicates whether the action can execute.

	UpdateTarget (see page 1915)	Enables or disables the action, depending on the currently active control in the form.

Published Events

	Name	Description
	OnHint (see page 1920)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1920)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1915)	Represents the caption of client controls and menu items.
	Enabled (see page 1916)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1916)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1917)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1917)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1917)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1918)	Indicates the ImageIndex property for client controls and menu items.
	SearchFind (see page 1918)	Associates the TSearchFindNext (see page 1912) action with a TSearchFind (see page 1899) action that initiates the search.
	SecondaryShortCuts (see page 1919)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1919)	Specifies the ShortCut property for client menu items.
	Visible (see page 1919)	Specifies the Visible property for client controls and menu items.

28.1.29.2 TSearchFindNext.Create Constructor

Creates an instance of TSearchFindNext (see page 1912).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TSearchFindNext(TComponent * AOwner);
```

Description

Most applications do not need to call the TSearchFindNext (see page 1912) constructor because actions are added at design time to an action manager or action list.

AOwner specifies the Owner for the action. This is the component that is responsible for freeing the TSearchFindNext (see page 1912) instance.

See Also

Owner

Actions

28.1.29.3 TSearchFindNext Methods

The methods of the TSearchFindNext class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1914)	Searches the target edit control for the next instance of the search string.
	HandlesTarget (see page 1914)	Indicates whether the action can execute.

	UpdateTarget (see page 1915)	Enables or disables the action, depending on the currently active control in the form.

28.1.29.3.1 TSearchFindNext.ExecuteTarget Method

Searches the target edit control for the next instance of the search string.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget calls the Search method of the TSearchFind (see page 1899) action specified by the SearchFind (see page 1918) property. This method ignores the Target parameter, and instead searches the edit control that was the target when SearchFind (see page 1918) first launched the search dialog. If another instance of the search dialog's current target string appears in the edit control, the next instance of the string is selected. Otherwise, a message dialog appears to inform the user that the string could not be found.

See Also

SearchFind (see page 1918)

Search (see page 1898)

28.1.29.3.2 TSearchFindNext.HandlesTarget Method

Indicates whether the action can execute.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a “target”.

Target is the currently active object.

HandlesTarget always ignores the Target parameter, and checks whether the associated TSearchFind (see page 1899) action is enabled, and if so, whether its associated dialog has a search string. If the action specified by SearchFind (see page 1918) is enabled, and the FindText property of its associated dialog has been set, then HandlesTarget returns true, because it can search for the FindText string in the TSearchFind (see page 1899) action's edit control.

See Also

SearchFind (see page 1918)

FindText

28.1.29.3.3 TSearchFindNext.UpdateTarget Method

Enables or disables the action, depending on the currently active control in the form.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically when the potential target changes so that actions can update themselves accordingly.

In TSearchFindNext (see page 1912), UpdateTarget ignores the current target object (specified by the Target parameter). Instead, it checks the associated TSearchFind (see page 1899) action (the SearchFind (see page 1918) property), and sets Enabled (see page 1916) to true if the search dialog has already specified a search string. If SearchFind (see page 1918) is not set, or if there is no search string in its associated dialog, UpdateTarget sets Enabled (see page 1916) to false.

See Also

Enabled

SearchFind (see page 1918)

FindText

28.1.29.4 TSearchFindNext Properties

The properties of the TSearchFindNext class are listed here.

Published Properties

	Name	Description
	Caption (see page 1915)	Represents the caption of client controls and menu items.
	Enabled (see page 1916)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1916)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1917)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1917)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1917)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1918)	Indicates the ImageIndex property for client controls and menu items.
	SearchFind (see page 1918)	Associates the TSearchFindNext (see page 1912) action with a TSearchFind (see page 1899) action that initiates the search.
	SecondaryShortCuts (see page 1919)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1919)	Specifies the ShortCut property for client menu items.
	Visible (see page 1919)	Specifies the Visible property for client controls and menu items.

28.1.29.4.1 TSearchFindNext.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.29.4.2 TSearchFindNext.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.29.4.3 TSearchFindNext.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.29.4.4 TSearchFindNext.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.29.4.5 TSearchFindNext.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.29.4.6 TSearchFindNext.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.29.4.7 TSearchFindNext.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.29.4.8 TSearchFindNext.SearchFind Property

Associates the TSearchFindNext (see page 1912) action with a TSearchFind (see page 1899) action that initiates the search.

Pascal

```
property SearchFind: TSearchFind;
```

C++

```
__property TSearchFind SearchFind;
```

Description

SearchFind is the TSearchFind (see page 1899) action that displays the modeless search dialog whose current search should be repeated. If SearchFind is not set, then the TSearchFindNext (see page 1912) action remains disabled.

TSearchFindNext (see page 1912) uses the SearchFind property to obtain a search string from its associated dialog and to search its associated control.

See Also

ExecuteTarget (see page 1914)

Enabled

Search (see page 1898)

28.1.29.4.9 TSearchFindNext.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.29.4.10 TSearchFindNext.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.29.4.11 TSearchFindNext.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.29.5 TSearchFindNext Events

The events of the TSearchFindNext class are listed here.

Published Events

	Name	Description
	OnHint (see page 1920)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1920)	Occurs when the application is idle or when the action list updates.

28.1.29.5.1 TSearchFindNext.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.29.5.2 TSearchFindNext.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
```

```
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

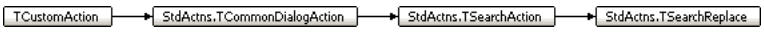
See Also

UpdateAction

28.1.30 TSearchReplace Class

TSearchReplace is the standard action for displaying a find-and-replace dialog.

Class Hierarchy



Pascal

```
TSearchReplace = class(TSearchAction);
```

C++

```
class TSearchReplace : public TSearchAction;
```

File

StdActns

Description

Add TSearchReplace to an action list to add a modeless find-and-replace dialog to your application. Controls such as menu items and tool buttons linked to this action cause the application to display the find-and-replace dialog or bring it to the fore.

TSearchReplace is designed to work with an edit control (TCustomEdit (see page 2345) descendant) as a target. It automatically disables itself if the active control in a form is not an edit control.

Search (see page 1898)/replace actions represent a modeless dialog. As a result, the OnAccept (see page 1930) event occurs not when the user exits the dialog, but when the dialog first appears or comes into focus. When the user directs the dialog to replace one or all instances of a string, TSearchReplace automatically performs the search and replaces the specified text in the target edit control.

See Also

TReplaceDialog

TSearchFind (see page 1899)

TCustomEdit (see page 2345)

TSearchFindFirst (see page 1909)

28.1.30.1 TSearchReplace Members

The following tables list the members exposed by TSearchReplace.

Protected Fields

	Name	Description
	FControl (see page 1896)	This is FControl, a member of class TSearchAction.
	FDialog (see page 1786)	This is FDialog, a member of class TCommonDialogAction.
	FFindFirst (see page 1897)	This is FFindFirst, a member of class TSearchAction.

Protected Methods

	Name	Description
	DoAccept (see page 1787)	This is DoAccept, a member of class TCommonDialogAction.
	DoCancel (see page 1787)	This is DoCancel, a member of class TCommonDialogAction.
	GetDialogClass (see page 1924)	This is GetDialogClass, a member of class TSearchReplace.
	Notification (see page 1789)	Responds when components are created or destroyed.
	Notification (see page 1898)	Responds when components are created or destroyed.
	SetupDialog (see page 1789)	This is SetupDialog, a member of class TCommonDialogAction.

Public Methods

	Name	Description
	Create (see page 1785)	Creates an instance of TCommonDialogAction.
	Create (see page 1895)	Creates and initializes an instance of TSearchAction (see page 1894).
	Destroy (see page 1786)	Frees an instance of TCustomAction.
	Destroy (see page 1896)	Frees an instance of TCustomAction.

	ExecuteTarget (see page 1923)	Launches the associated dialog and assigns an event handler to replace found text.
	HandlesTarget (see page 1788)	Indicates whether the action can execute its dialog.
	HandlesTarget (see page 1898)	Indicates whether the action can execute its dialog.
	Search (see page 1898)	Selects the next occurrence of the search string in the target edit control.
	UpdateTarget (see page 1899)	Enables or disables the action, depending on the currently active control in the form.

Public Properties

	Name	Description
	ExecuteResult (see page 1789)	Indicates the result of executing the associated dialog.

Published Events

	Name	Description
	BeforeExecute (see page 1929)	Occurs immediately before the dialog appears.
	OnAccept (see page 1930)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1930)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1930)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1931)	Occurs when the application is idle or when the action list updates.

Published Properties

	Name	Description
	Caption (see page 1925)	Represents the caption of client controls and menu items.
	Dialog (see page 1925)	Provides access the find-and-replace dialog that the action executes.
	Enabled (see page 1925)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1926)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1926)	Indicates the help keyword for client controls and menu items.
	HelpType (see page 1927)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1927)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1927)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1928)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1928)	Specifies the ShortCut property for client menu items.
	Visible (see page 1929)	Specifies the Visible property for client controls and menu items.

28.1.30.2 TSearchReplace Methods

The methods of the TSearchReplace class are listed here.

Protected Methods

	Name	Description
	GetDialogClass (see page 1924)	This is GetDialogClass, a member of class TSearchReplace.

Public Methods

	Name	Description
	ExecuteTarget (see page 1923)	Launches the associated dialog and assigns an event handler to replace found text.

28.1.30.2.1 TSearchReplace.ExecuteTarget Method

Launches the associated dialog and assigns an event handler to replace found text.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget first launches the search-and-replace dialog specified by Dialog (see page 1925). This is a modeless dialog, so the result of the dialog's Execute method simply indicates whether the dialog successfully appeared or was brought into focus. This result is set as the value of the ExecuteResult (see page 1789) property, and an OnAccept (see page 1930) event occurs if the dialog was brought into focus, or an OnCancel (see page 1930) event occurs if the dialog failed to appear.

Finally, ExecuteTarget sets the OnReplace event handler of the search-and-replace dialog to a private method that performs the replacement specified by the dialog.

See Also

OnAccept (see page 1790)

OnCancel (see page 1791)

Dialog (see page 1925)

Execute

ExecuteResult (see page 1789)

OnReplace

28.1.30.2.2 TSearchReplace.GetDialogClass Method

Pascal

```
function GetDialogClass: TCommonDialogClass; override;
```

C++

```
virtual __fastcall TCommonDialogClass GetDialogClass();
```

Description

This is GetDialogClass, a member of class TSearchReplace.

28.1.30.3 TSearchReplace Properties

The properties of the TSearchReplace class are listed here.

Published Properties

	Name	Description
	Caption (see page 1925)	Represents the caption of client controls and menu items.
R	Dialog (see page 1925)	Provides access the find-and-replace dialog that the action executes.
	Enabled (see page 1925)	Indicates whether client controls and menu items are enabled.
	HelpContext (see page 1926)	Indicates the help context ID for client controls and menu items.
	HelpKeyword (see page 1926)	Indicates the help keyword for client controls and menu items.

	HelpType (see page 1927)	Indicates the mechanism for client controls and menu items to use when invoking help.
	Hint (see page 1927)	Indicates the Help hint for client controls and menu items.
	ImageIndex (see page 1927)	Indicates the ImageIndex property for client controls and menu items.
	SecondaryShortCuts (see page 1928)	Specifies the short cuts (in addition to ShortCut) for triggering clients.
	ShortCut (see page 1928)	Specifies the ShortCut property for client menu items.
	Visible (see page 1929)	Specifies the Visible property for client controls and menu items.

28.1.30.3.1 TSearchReplace.Caption Property

Represents the caption of client controls and menu items.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Caption holds the string that is the caption for the action when it is set. The value of Caption is propagated to client controls and menu items.

See Also

Caption

Caption

28.1.30.3.2 TSearchReplace.Dialog Property

Provides access the find-and-replace dialog that the action executes.

Pascal

```
property Dialog: TReplaceDialog;
```

C++

```
__property TReplaceDialog Dialog;
```

Description

Use Dialog to configure the find-and-replace dialog before the action fires or to close the modeless dialog programmatically. For example, you can set the dialog's options at design time using the Object Inspector or call the dialog's CloseDialog method to shut down the modeless find-and-replace dialog.

When using a search/replace action, you do not place a TReplaceDialog component on the form or data module at design time. Instead, you add the TSearchReplace (see page 1921) action to your action list and then use the Dialog property. The TReplaceDialog component is created by the TSearchReplace (see page 1921) constructor.

See Also

ExecuteTarget (see page 1897)

CloseDialog

28.1.30.3.3 TSearchReplace.Enabled Property

Indicates whether client controls and menu items are enabled.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Enabled specifies the enabled state for the action. The value of Enabled is propagated to client controls and menu items.

See Also

Enabled

Enabled

28.1.30.3.4 TSearchReplace.HelpContext Property

Indicates the help context ID for client controls and menu items.

Pascal

```
property HelpContext;
```

C++

```
__property HelpContext;
```

Description

HelpContext specifies the help context for the action when it is set. This value is propagated to client controls and menu items.

When a client control has focus and the user presses the F1 key, the control uses the value of HelpContext for this associated action to invoke the online help targeting a specific topic. The target topic is uniquely identified by a nonzero HelpContext value. Leave the HelpContext property of the client control at its default value of zero. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpContext is only used when the HelpType property of this action or the client control is htInteger.

See Also

HelpType

HelpKeyword

28.1.30.3.5 TSearchReplace.HelpKeyword Property

Indicates the help keyword for client controls and menu items.

Pascal

```
property HelpKeyword;
```

C++

```
__property HelpKeyword;
```

Description

HelpKeyword, when set, specifies the keyword that identifies a help topic for this action. This value is propagated to the HelpKeyword property of client controls and menu items. The interpretation of HelpKeyword depends on which help system is used to display the topic. For example, when using HyperHelp (on Linux) or WinHelp (on Windows), HelpKeyword refers to an A link.

When a client control has focus and the user presses the F1 key (or the key specified by the application's HelpKey property), the control uses the value of HelpKeyword for this associated action object to invoke the online help targeting a specific topic. Leave the HelpKeyword property of the client control unassigned. This allows multiple controls (and multiple types of controls) to be associated with the same help topic through the one action object.

Note: HelpKeyword is only used when the HelpType property of this action or the client control is htKeyword.

See Also

HelpKeyword

HelpContext

HelpType

28.1.30.3.6 TSearchReplace.HelpType Property

Indicates the mechanism for client controls and menu items to use when invoking help.

Pascal

```
property HelpType;
```

C++

```
__property HelpType;
```

Description

HelpType, when set, indicates whether client controls and menu items should use the HelpContext or the HelpKeyword property for invoking the help system. This value is propagated to the HelpType property of client controls and menu items.

See Also

HelpType

HelpContext

HelpKeyword

28.1.30.3.7 TSearchReplace.Hint Property

Indicates the Help hint for client controls and menu items.

Pascal

```
property Hint;
```

C++

```
__property Hint;
```

Description

Hint holds the string that is the hint for that action when it is set. This value is propagated to client controls and menu items.

See Also

Hint

Hint

28.1.30.3.8 TSearchReplace.ImageIndex Property

Indicates the ImageIndex property for client controls and menu items.

Pascal

```
property ImageIndex;
```

C++

```
__property ImageIndex;
```

Description

ImageIndex is an index into the Images property of the action list or action manager that contains the action. It identifies an image that is associated with controls and menu items that use this action.

See Also

ImageIndex

ImageIndex

Images

28.1.30.3.9 TSearchReplace.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut) for triggering clients.

Pascal

```
property SecondaryShortCuts;
```

C++

```
__property SecondaryShortCuts;
```

Description

Actions can execute in response to multiple short cut key combinations. SecondaryShortCuts lists all the short cut key combinations (other than the one specified by the ShortCut property) that can trigger the action. This lets you specify shortcuts for clients that do not have a ShortCut property or provide additional, secondary shortcuts.

When the user types one of the key combinations listed in SecondaryShortCuts, the action's Execute method is called.

See Also

ShortCut

Execute

28.1.30.3.10 TSearchReplace.ShortCut Property

Specifies the ShortCut property for client menu items.

Pascal

```
property ShortCut;
```

C++

```
__property ShortCut;
```

Description

ShortCut holds the value that is the shortcut for the action when it is set. This value is propagated to client menu items.

See Also

ShortCut

SecondaryShortCuts

28.1.30.3.11 TSearchReplace.Visible Property

Specifies the Visible property for client controls and menu items.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Visible specifies the visible state for the action. This value is propagated to client controls and menu items.

See Also

Visible

Visible

28.1.30.4 TSearchReplace Events

The events of the TSearchReplace class are listed here.

Published Events

	Name	Description
	BeforeExecute (see page 1929)	Occurs immediately before the dialog appears.
	OnAccept (see page 1930)	Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.
	OnCancel (see page 1930)	Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.
	OnHint (see page 1930)	Occurs when the mouse pauses over a client control or menu item.
	OnUpdate (see page 1931)	Occurs when the application is idle or when the action list updates.

28.1.30.4.1 TSearchReplace.BeforeExecute Event

Occurs immediately before the dialog appears.

Pascal

```
property BeforeExecute: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeExecute;
```

Description

Write a BeforeExecute event handler to initialize the dialog immediately before it executes. When the action fires, it generates a BeforeExecute event, and then executes its associated dialog.

See Also

ExecuteTarget (see page 1787)

Execute

Handle

28.1.30.4.2 TSearchReplace.OnAccept Event

Occurs after the user exits a modal dialog by selecting “OK” or when a modeless dialog is successfully displayed.

Pascal

```
property OnAccept: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnAccept;
```

Description

Write an OnAccept event handler to make use the dialog results after it executes. When the action fires, it executes its associated dialog. OnAccept occurs when the dialog component's Execute method returns true. Thus, if the dialog is modal, OnAccept occurs when the user closes the dialog by clicking the OK button. If the dialog is modeless, OnAccept occurs when the dialog first appears.

For modal dialogs, use the OnAccept event to read the user's selections from the dialog and act on them. TCommonDialogAction descendants introduce a Dialog property that the event handler can use to obtain information about the user's selections.

See Also

ExecuteResult (see page 1789)

OnCancel (see page 1791)

Execute

28.1.30.4.3 TSearchReplace.OnCancel Event

Occurs after the user exits a modal dialog by selecting “Cancel” or when a modeless dialog fails to come up.

Pascal

```
property OnCancel: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCancel;
```

Description

Write an OnCancel event handler to respond when the user cancels from the associated dialog. When the action fires, it executes its associated dialog. OnCancel occurs when the dialog component's Execute method returns false. Thus, if the dialog is modal, OnCancel occurs when the user closes the dialog by clicking the Cancel button. If the dialog is modeless, OnCancel occurs if the dialog fails to appear.

See Also

ExecuteResult (see page 1789)

OnAccept (see page 1790)

Execute

28.1.30.4.4 TSearchReplace.OnHint Event

Occurs when the mouse pauses over a client control or menu item.

Pascal

```
property OnHint;
```

C++

```
__property OnHint;
```

Description

Use OnHint to override the default hint that appears when the user pauses the mouse over a client control or menu item. The OnHint event handler can change the hint that appears (by changing the HintStr parameter), or display the hint in some application-specific way.

28.1.30.4.5 TSearchReplace.OnUpdate Event

Occurs when the application is idle or when the action list updates.

Pascal

```
property OnUpdate;
```

C++

```
__property OnUpdate;
```

Description

Write an OnUpdate event handler to execute centralized code while an application is idle. For example, actions may want to update enabling and disabling, or checking and unchecking of client targets.

C++ Examples:

```
/*
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
*/
Boolean checked;

void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible.
   TAction* pa = dynamic_cast<TAction*>(Sender);
   if (pa)
   {
     if (checked != ToolBar1->Visible)
     {
       lbOther->Items->Add("Event Action1Update: toolbar visibility changed");
     }
     checked = ToolBar1->Visible;
   }
//   pa->Checked = True;  // set this statically and don't change it!
}
```

Delphi Examples:

```
{
When the application is idle, the OnUpdate event occurs for
every action that is linked to a visible control or menu
item that is showing. This provides an opportunity for
applications to execute centralized code for enabling and
disabling, checking and unchecking, and so on. For example,
the following code illustrates the OnUpdate event handler
for an action that is "checked" when the toolbar is visible.
This example requires a TActionList (with one TAction) and a
TToolBar on a form. This example requires a TActionList
(with one TAction) and a TToolBar on a form.  Double click
on the TActionList and create an TAction named Action1.  The
TAction has an OnUpdate event handler called Action1Update.
Then set an object's (such as a button or menu item) Action
to Action1.
}
procedure TMainForm.Action1Update(Sender: TObject);
begin
  if ((Sender as TAction).Checked <> ToolBar1.Visible) then
    lbOther.Items.Add('Event Action1Update: toolbar visibility changed');
    { Indicate whether ToolBar1 is currently visible }
    (Sender as TAction).Checked := ToolBar1.Visible;
end;
```

See Also

UpdateAction

28.1.31 TWindowAction Class

TWindowAction is the base class for action objects that work with MDI forms.

Class Hierarchy

Pascal

```
TWindowAction = class(TAction);
```

C++

```
class TWindowAction : public TAction;
```

File

StdActns

Description

TWindowAction implements the common behavior for actions that work on MDI forms. Each descendant class performs a specific task, such as arranging, cascading, closing, tiling, or minimizing MDI child forms. Do not use TWindowAction directly in an application. Instead, use a TWindowAction descendant such as TWindowClose (see page 1939), TWindowCascade (see page 1937), TWindowMinimizeAll (see page 1941), TWindowArrange (see page 1936), TWindowTileHorizontal (see page 1943), or TWindowTileVertical (see page 1944).

TWindowAction ensures that the action is enabled as long as the target is an MDI parent form. TWindowAction descendants override the ExecuteTarget method to execute their specific function on the targets child forms.

See Also

TWindowArrange (see page 1936)

TWindowCascade (see page 1937)

TWindowClose (see page 1939)

TWindowMinimizeAll (see page 1941)

TWindowTileHorizontal (see page 1943)

TWindowTileVertical (see page 1944)

28.1.31.1 TWindowAction Members

The following tables list the members exposed by TWindowAction.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1933)	Creates and initializes a TAction object.
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1935)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.31.2 TWindowAction.Create Constructor

Creates and initializes a TAction object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWindowAction(TComponent * AOwner);
```

Description

Applications do not usually create action objects explicitly. Actions are created automatically when you add them to an action manager or action list component at design time. When instantiating TAction at runtime, assign a TActionList or TActionManager component to the new action's ActionList property after calling Create.

See Also

ActionList

28.1.31.3 TWindowAction Methods

The methods of the TWindowAction class are listed here.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.

	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1935)	Enables or disables the action, depending on whether the target form has MDI children.

28.1.31.3.1 TWindowAction.GetForm Method

Pascal

```
function GetForm(Target: TObject): TForm; virtual;
```

C++

```
virtual __fastcall TForm * GetForm(TObject * Target);
```

Description

This is GetForm, a member of class TWindowAction.

28.1.31.3.2 TWindowAction.HandlesTarget Method

Indicates whether the action can execute its function.

Pascal

```
function HandlesTarget(Target: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean HandlesTarget(TObject * Target);
```

Description

Do not call the HandlesTarget method. This method is called automatically when the user invokes an object (such as a tool button or menu item) that is linked to this action. The HandlesTarget method provides the action object with an opportunity to indicate whether it is appropriate to execute at this time with the object specified by the Target parameter as a "target".

As implemented in TWindowAction (see page 1932), HandlesTarget makes sure that the target is an MDI parent form. If the target is appropriate, HandlesTarget sets the Form (see page 1935) property.

See Also

Form (see page 1935)

FormStyle

28.1.31.3.3 TWindowAction.Notification Method

Responds when components are created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

TBasicAction overrides the Notification method to check whether the component that is the value of the ActionComponent property is about to be freed. If so, it sets the ActionComponent property to nil (Delphi) or NULL (C++).

See Also

ActionComponent

28.1.31.3.4 TWindowAction.UpdateTarget Method

Enables or disables the action, depending on whether the target form has MDI children.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TWindowAction (see page 1932), UpdateTarget checks that the target object is an MDI parent form with an MDIChildCount property greater than 0. If so, UpdateTarget sets the Enabled property to true. Otherwise, it sets Enabled to false.

See Also

Enabled

MDIChildCount

28.1.31.4 TWindowAction Properties

The properties of the TWindowAction class are listed here.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.31.4.1 TWindowAction.Form Property

Provides access to the target of the action.

Pascal

```
property Form: TCustomForm;
```

C++

```
__property TCustomForm Form;
```

Description

Form is the MDI parent form on which the action performs its function.

28.1.32 TWindowArrange Class

TWindowArrange arranges the icons of minimized MDI child forms.

Class Hierarchy

Pascal

```
TWindowArrange = class(TWindowAction);
```

C++

```
class TWindowArrange : public TWindowAction;
```

File

StdActns

Description

Add TWindowArrange to an action list to allow users to arrange the icons that represent minimized MDI child forms. Controls such as menu items and tool buttons linked to this action cause the application to call the ArrangeIcons method of the MDI parent form.

TWindowArrange is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1939)

TWindowCascade (see page 1937)

TWindowMinimizeAll (see page 1941)

TWindowTileHorizontal (see page 1943)

TWindowTileVertical (see page 1944)

TForm

28.1.32.1 TWindowArrange Members

The following tables list the members exposed by TWindowArrange.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1933)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1937)	Arranges the icons of minimized MDI child forms
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1935)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.32.2 TWindowArrange Methods

The methods of the TWindowArrange class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1937)	Arranges the icons of minimized MDI child forms

28.1.32.2.1 TWindowArrange.ExecuteTarget Method

Arranges the icons of minimized MDI child forms

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1935) property to Target, which must be an MDI parent form. It then calls that form's ArrangeIcons method to arrange the icons of minimized MDI children.

See Also

Form (see page 1935)

ArrangeIcons

28.1.33 TWindowCascade Class

TWindowCascade cascades the MDI child forms.

Class Hierarchy

Pascal

```
TWindowCascade = class(TWindowAction);
```

C++

```
class TWindowCascade : public TWindowAction;
```

File

StdActns

Description

Add TWindowCascade to an action list to allow users to arrange the MDI child forms in an overlapping cascade. Controls such as menu items and tool buttons linked to this action cause the application to call the Cascade method of the MDI parent form.

TWindowCascade is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1939)

TWindowArrange (see page 1936)

TWindowMinimizeAll (see page 1941)

TWindowTileHorizontal (see page 1943)

TWindowTileVertical (see page 1944)

TForm

28.1.33.1 TWindowCascade Members

The following tables list the members exposed by TWindowCascade.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1933)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1938)	Arranges the MDI child forms in an overlapping cascade.
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1935)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.33.2 TWindowCascade Methods

The methods of the TWindowCascade class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1938)	Arranges the MDI child forms in an overlapping cascade.

28.1.33.2.1 TWindowCascade.ExecuteTarget Method

Arranges the MDI child forms in an overlapping cascade.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1935) property to Target, which must be an MDI parent form. It then calls that form's Cascade method to arrange MDI child windows.

See Also

Form (see page 1935)

Cascade

28.1.34 TWindowClose Class

TWindowClose closes the active MDI child form.

Class Hierarchy

Pascal

```
TWindowClose = class(TWindowAction);
```

C++

```
class TWindowClose : public TWindowAction;
```

File

StdActns

Description

Add TWindowClose to an action list to allow users to close the active MDI child form. Controls such as menu items and tool buttons linked to this action cause the application to call the Close (see page 3538) method of the active MDI child.

TWindowClose is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no active MDI child.

See Also

TWindowArrange (see page 1936)

TWindowCascade (see page 1937)

TWindowMinimizeAll (see page 1941)

TWindowTileHorizontal (see page 1943)

TWindowTileVertical (see page 1944)

TCustomForm

28.1.34.1 TWindowClose Members

The following tables list the members exposed by TWindowClose.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1933)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1940)	Closes the active MDI child.
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1941)	Enables or disables the action, depending on whether the target form an active child.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.34.2 TWindowClose Methods

The methods of the TWindowClose class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1940)	Closes the active MDI child.
	UpdateTarget (see page 1941)	Enables or disables the action, depending on whether the target form an active child.

28.1.34.2.1 TWindowClose.ExecuteTarget Method

Closes the active MDI child.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active object when the action fires.

ExecuteTarget sets the Form (see page 1935) property to Target, which must be an MDI parent form. It then obtains a reference to the active MDI child by reading the form's ActiveMDIChild property. It calls this child form's Close (see page 3538) method to close the active child form.

See Also

Form (see page 1935)

ActiveMDIChild

Close

28.1.34.2.2 TWindowClose.UpdateTarget Method

Enables or disables the action, depending on whether the target form an active child.

Pascal

```
procedure UpdateTarget(Target: TObject); override;
```

C++

```
virtual __fastcall UpdateTarget(TObject * Target);
```

Description

Do not call the UpdateTarget method. This method is called automatically so that actions can update themselves according to current conditions.

In TWindowClose (see page 1939), UpdateTarget checks that the target object is an MDI parent form with a non-nil (Delphi) or NULL (C++) ActiveMDIChild property. If so, UpdateTarget sets the Enabled property to true. Otherwise, it sets Enabled to false.

See Also

Enabled

ActiveMDIChild

28.1.35 TWindowMinimizeAll Class

TWindowMinimizeAll minimizes all MDI child forms.

Class Hierarchy

Pascal

```
TWindowMinimizeAll = class(TWindowAction);
```

C++

```
class TWindowMinimizeAll : public TWindowAction;
```

File

StdActns

Description

Add TWindowMinimizeAll to an action list to allow users to minimize the MDI child forms in an MDI application. Controls such as menu items and tool buttons linked to this action cause the application to iterate through the MDI child forms, setting their WindowState to wsMinimized.

TWindowMinimizeAll is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1939)

TWindowArrange (see page 1936)

TWindowCascade (see page 1937)

TWindowTileHorizontal (see page 1943)

TWindowTileVertical (see page 1944)

TCustomForm

28.1.35.1 TWindowMinimizeAll Members

The following tables list the members exposed by TWindowMinimizeAll.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1933)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1942)	Minimizes all MDI child forms of the target MDI parent.
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1935)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.35.2 TWindowMinimizeAll Methods

The methods of the TWindowMinimizeAll class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1942)	Minimizes all MDI child forms of the target MDI parent.

28.1.35.2.1 TWindowMinimizeAll.ExecuteTarget Method

Minimizes all MDI child forms of the target MDI parent.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1935) property to Target, which must be an MDI parent form. It then iterates through that form's MDI children, setting each child form's WindowState to wsMinimized.

See Also

Form (see page 1935)

MDIChildren

WindowState

28.1.36 TWindowTileHorizontal Class

TWindowTileHorizontal arranges MDI child forms so that they are all the same size, tiled horizontally.

Class Hierarchy

Pascal

```
TWindowTileHorizontal = class(TWindowAction);
```

C++

```
class TWindowTileHorizontal : public TWindowAction;
```

File

StdActns

Description

Add TWindowTileHorizontal to an action list to allow users to tile the MDI child forms horizontally. Controls such as menu items and tool buttons linked to this action cause the MDI parent form to tile its children horizontally.

TWindowTileHorizontal is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1939)

TWindowArrange (see page 1936)

TWindowCascade (see page 1937)

TWindowMinimizeAll (see page 1941)

TWindowTileVertical (see page 1944)

TCustomForm

28.1.36.1 TWindowTileHorizontal Members

The following tables list the members exposed by TWindowTileHorizontal.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1933)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1944)	Tiles the child forms of the target MDI parent horizontally.
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1935)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.36.2 TWindowTileHorizontal Methods

The methods of the TWindowTileHorizontal class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1944)	Tiles the child forms of the target MDI parent horizontally.

28.1.36.2.1 TWindowTileHorizontal.ExecuteTarget Method

Tiles the child forms of the target MDI parent horizontally.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1935) property to Target, which must be an MDI parent form. It then instructs the form to tile its child forms horizontally.

See Also

Form (see page 1935)

MDIChildren

28.1.37 TWindowTileVertical Class

TWindowTileVertical arranges MDI child forms so that they are all the same size, tiled vertically.

Class Hierarchy

Pascal

```
TWindowTileVertical = class(TWindowAction);
```

C++

```
class TWindowTileVertical : public TWindowAction;
```

File

StdActns

Description

Add TWindowTileVertical to an action list to allow users to tile the MDI child forms vertically. Controls such as menu items and tool buttons linked to this action cause the MDI parent form to tile its children vertically.

TWindowTileVertical is designed to work with an MDI parent form as a target. It automatically disables itself if the active form is not an MDI parent or if it has no MDI children.

See Also

TWindowClose (see page 1939)

TWindowArrange (see page 1936)

TWindowCascade (see page 1937)

TWindowMinimizeAll (see page 1941)

TCustomForm

28.1.37.1 TWindowTileVertical Members

The following tables list the members exposed by TWindowTileVertical.

Protected Methods

	Name	Description
	GetForm (see page 1934)	This is GetForm, a member of class TWindowAction.
	Notification (see page 1934)	Responds when components are created or destroyed.

Public Methods

	Name	Description
	Create (see page 1933)	Creates and initializes a TAction object.
	ExecuteTarget (see page 1946)	Tiles the child forms of the target MDI parent vertically.
	HandlesTarget (see page 1934)	Indicates whether the action can execute its function.
	UpdateTarget (see page 1935)	Enables or disables the action, depending on whether the target form has MDI children.

Public Properties

	Name	Description
	Form (see page 1935)	Provides access to the target of the action.

28.1.37.2 TWindowTileVertical Methods

The methods of the TWindowTileVertical class are listed here.

Public Methods

	Name	Description
	ExecuteTarget (see page 1946)	Tiles the child forms of the target MDI parent vertically.

28.1.37.2.1 TWindowTileVertical.ExecuteTarget Method

Tiles the child forms of the target MDI parent vertically.

Pascal

```
procedure ExecuteTarget(Target: TObject); override;
```

C++

```
virtual __fastcall ExecuteTarget(TObject * Target);
```

Description

Do not call the ExecuteTarget method. This method is called automatically when the action fires in response to a user action (for example, when the user selects a menu item or presses a tool button that is linked to this action).

Target is the currently active form when the action fires.

ExecuteTarget sets the Form (see page 1935) property to Target, which must be an MDI parent form. It then instructs the form to tile its child forms vertically.

See Also

Form (see page 1935)

MDIChildren

28.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBrowseForFolderOption (see page 1947)	TBrowseForFolderOption contains the options for creating a dialog box used when browsing for folders.

28.2.1 StdActns.TBrowseForFolderOption Enumeration

TBrowseForFolderOption contains the options for creating a dialog box used when browsing for folders.

Pascal

```
TBrowseForFolderOption = (
  bifBrowseForComputer,
  bifBrowseForPrinter,
  bifBrowseIncludeFiles,
  bifBrowseIncludeURLS,
  bifDontGoBelowDomain,
  bifEditBox,
  bifNewDialogStyle,
  bifNoNewFolderButton,
  bifNoTranslateTargets,
  bifReturnFSAncestors,
  bifReturnOnlyFSDirs,
  bifShareable,
  bifStatusText,
  bifUAHint,
  bifUseNewUI,
  bifValidate
);
```

C++

```
enum TBrowseForFolderOption {
  bifBrowseForComputer,
  bifBrowseForPrinter,
  bifBrowseIncludeFiles,
  bifBrowseIncludeURLS,
  bifDontGoBelowDomain,
  bifEditBox,
  bifNewDialogStyle,
  bifNoNewFolderButton,
  bifNoTranslateTargets,
  bifReturnFSAncestors,
  bifReturnOnlyFSDirs,
  bifShareable,
  bifStatusText,
  bifUAHint,
  bifUseNewUI,
  bifValidate
};
```

File

StdActns

Description

TBrowseForFolderOption is an enumeration type containing the options that specify the type of dialog box used when browsing for folders.

The meaning of these options is given in the following table.

Option	Meaning
`bifBrowseForComputer`	Only allow computers to be returned. If anything else is selected, the **OK** button is disabled.
`bifBrowseForPrinter`	Only allow printers to be selected. If anything else is selected, the **OK** button is disabled.
`bifBrowseIncludeFiles`	The browse dialog box also displays files.
`bifBrowseIncludeURLS`	The browse dialog box also displays URL adresses. This automatically enables the options `bifBrowseIncludeFiles` and `bifUseNewUI`.
`bifDontGoBelowDomain`	Do not display network folders below the current domain level in the current tree view.
`bifEditBox`	Display an edit box containing the currently selected folder/file. This also allows typing the name of the folder/file to select.
`bifNewDialogStyle`	Use the new type of dialog, that is sizeable and includes a **Make a New (see page 3619) Folder** button. The dialog also allows dragging and dropping and using the context menu on either folders or files.
`bifNoNewFolderButton`	Hides the **Make New (see page 3619) Folder** button, when using the `bifNewDialogStyle` option.
`bifNoTranslateTargets`	If the selected folder/file is a shortuct, return the PIDL of the shortcut instead of its target.
`bifReturnFSAncestors`	Only allow file system ancestors to be returned. Ancestors are subfolders that live beneath the root of the namespace hierarchy. If the selected ancestor is not part of the file system, the **OK** button is disabled.
`bifReturnOnlyFSDirs`	Only allow file system directories to be returned. If anything else is selected, the **OK** button is disabled.
`bifShareable`	Shareable resources on remote systems are also displayed inside the browse dialog box. This automatically enables the option `bifNewDialogStyle`.
`bifStatusText`	Display a status message in the browse dialog box. The message is returned by the callback function. This option only works if the `bifNewDialogStyle` is disabled.
`bifUAHint`	Displays an usage hint instead of the edit box, when using the `bifNewDialogStyle` option
`bifUseNewUI`	Use the new user interface. This displays an edit box at the bottom of the dialog window, containing the name of the currently selected folder/file. The name of the folder/file to select can also be typed into this edit box.
`bifValidate`	Validates the name of the folder/file inserted into the edit box, when using the `bifEditBox` option. The validation triggers an event of type TBrowseForFolderCallbackEvent (see page 1949).

See Also

TBrowseForFolder (see page 1764)

28.3 Types

The following table lists types in this documentation.

Types

Name	Description
TBrowseForFolderCallbackEvent (see page 1949)	This is type StdActns.TBrowseForFolderCallbackEvent.
TBrowseOptions (see page 1949)	TBrowseOptions is a set type of options used in customizing the browse dialog box to create.
TCommonDialogClass (see page 1950)	This is type StdActns.TCommonDialogClass.

28.3.1 StdActns.TBrowseForFolderCallbackEvent Type

Pascal

```
TBrowseForFolderCallbackEvent = procedure (Sender: TBrowseForFolder; Wnd: HWND; uMsg: UINT;
lParam, lpData: LPARAM; var Result: Integer) of object;
```

C++

```
(Sender: TBrowseForFolder; Wnd: HWND; uMsg: UINT; lParam, lpData: LPARAM; var Result: Integer)
( TBrowseForFolderCallbackEvent)();
```

File

StdActns

Description

This is type StdActns.TBrowseForFolderCallbackEvent.

28.3.2 StdActns.TBrowseOptions Type

TBrowseOptions is a set type of options used in customizing the browse dialog box to create.

Pascal

```
TBrowseOptions = set of TBrowseForFolderOption;
```

C++

```
TBrowseForFolderOption TBrowseOptions;
```

File

StdActns

Description

TBrowseOptions is a set type, with elements of type TBrowseForFolderOption (see page 1947).

Each element included in a variable of type TBrowseOptions enables the corresponding option, when creating the browse dialog box.

The possible options are given in the following table.

Option	Meaning
bifBrowseForComputer	Only allow computers to be returned. If anything else is selected, the **OK** button is disabled.
bifBrowseForPrinter	Only allow printers to be selected. If anything else is selected, the **OK** button is disabled.
bifBrowseIncludeFiles	The browse dialog box also displays files.
bifBrowseIncludeURLS	The browse dialog box also displays URL adresses. This automatically enables the options bifBrowseIncludeFiles and bifUseNewUI.
bifDontGoBelowDomain	Do not display network folders below the current domain level in the current tree view.
bifEditBox	Display an edit box containing the currently selected folder/file. This also allows typing the name of the folder/file to select.
bifNewDialogStyle	Use the new type of dialog, that is sizeable and includes a **Make a New (see page 3619) Folder** button. The dialog also allows dragging and dropping and using the context menu on either folders or files.
bifNoNewFolderButton	Hides the **Make New (see page 3619) Folder** button, when using the bifNewDialogStyle option.
bifNoTranslateTargets	If the selected folder/file is a shortcut, return the PIDL of the shortcut instead of its target.
bifReturnFSAncestors	Only allow file system ancestors to be returned. Ancestors are subfolders that live beneath the root of the namespace hierarchy. If the selected ancestor is not part of the file system, the **OK** button is disabled.
bifReturnOnlyFSDirs	Only allow file system directories to be returned. If anything else is selected, the **OK** button is disabled.
bifShareable	Shareable resources on remote systems are also displayed inside the browse dialog box. This automatically enables the option bifNewDialogStyle.
bifStatusText	Display a status message in the browse dialog box. The message is returned by the callback function. This option only works if the bifNewDialogStyle is disabled.
bifUAHint	Displays an usage hint instead of the edit box, when using the bifNewDialogStyle option
bifUseNewUI	Use the new user interface. This displays an edit box at the bottom of the dialog window, containing the name of the currently selected folder/file. The name of the folder/file to select can also be typed into this edit box.
bifValidate	Validates the name of the folder/file inserted into the edit box, when using the bifEditBox option. The validation triggers an event of type TBrowseForFolderCallbackEvent (see page 1949).

See Also

TBrowseForFolder (see page 1764)

28.3.3 StdActns.TCommonDialogClass Type

Pascal

```
TCommonDialogClass = class of TCommonDialog;
```

C++

```
TCommonDialog TCommonDialogClass;
```

File

StdActns

Description

This is type StdActns.TCommonDialogClass.

29 StdConvs Namespace

This is namespace StdConvs.

Topics

Name	Description
Functions (see page 1958)	The following table lists functions in this documentation.
Variables (see page 1960)	The following table lists variables in this documentation.
Constants (see page 2010)	The following table lists constants in this documentation.

Constants

Name	Description
CubicMetersPerAcreFoot (see page 2011)	This is constant StdConvs.CubicMetersPerAcreFoot.
CubicMetersPerAcreInch (see page 2011)	This is constant StdConvs.CubicMetersPerAcreInch.
CubicMetersPerCord (see page 2012)	This is constant StdConvs.CubicMetersPerCord.
CubicMetersPerCordFoot (see page 2012)	This is constant StdConvs.CubicMetersPerCordFoot.
CubicMetersPerCubicFoot (see page 2012)	This is constant StdConvs.CubicMetersPerCubicFoot.
CubicMetersPerCubicInch (see page 2013)	This is constant StdConvs.CubicMetersPerCubicInch.
CubicMetersPerCubicMile (see page 2013)	This is constant StdConvs.CubicMetersPerCubicMile.
CubicMetersPerCubicYard (see page 2013)	This is constant StdConvs.CubicMetersPerCubicYard.
CubicMetersPerUKBucket (see page 2014)	This is constant StdConvs.CubicMetersPerUKBucket.
CubicMetersPerUKBushel (see page 2014)	This is constant StdConvs.CubicMetersPerUKBushel.
CubicMetersPerUKGallon (see page 2014)	This is constant StdConvs.CubicMetersPerUKGallon.
CubicMetersPerUKGill (see page 2015)	This is constant StdConvs.CubicMetersPerUKGill.
CubicMetersPerUKOunce (see page 2015)	This is constant StdConvs.CubicMetersPerUKOunce.
CubicMetersPerUKPeck (see page 2015)	This is constant StdConvs.CubicMetersPerUKPeck.
CubicMetersPerUKPint (see page 2016)	This is constant StdConvs.CubicMetersPerUKPint.
CubicMetersPerUKPottle (see page 2016)	This is constant StdConvs.CubicMetersPerUKPottle.
CubicMetersPerUKQuart (see page 2016)	This is constant StdConvs.CubicMetersPerUKQuart.
CubicMetersPerUSDryBucket (see page 2017)	This is constant StdConvs.CubicMetersPerUSDryBucket.
CubicMetersPerUSDryBushel (see page 2017)	This is constant StdConvs.CubicMetersPerUSDryBushel.
CubicMetersPerUSDryGallon (see page 2017)	This is constant StdConvs.CubicMetersPerUSDryGallon.
CubicMetersPerUSDryPeck (see page 2018)	This is constant StdConvs.CubicMetersPerUSDryPeck.
CubicMetersPerUSDryPint (see page 2018)	This is constant StdConvs.CubicMetersPerUSDryPint.
CubicMetersPerUSDryQuart (see page 2018)	This is constant StdConvs.CubicMetersPerUSDryQuart.
CubicMetersPerUSFluidCup (see page 2019)	This is constant StdConvs.CubicMetersPerUSFluidCup.
CubicMetersPerUSFluidGallon (see page 2019)	This is constant StdConvs.CubicMetersPerUSFluidGallon.

CubicMetersPerUSFluidGill (see page 2019)	This is constant StdConvs.CubicMetersPerUSFluidGill.
CubicMetersPerUSFluidOunce (see page 2020)	This is constant StdConvs.CubicMetersPerUSFluidOunce.
CubicMetersPerUSFluidPint (see page 2020)	This is constant StdConvs.CubicMetersPerUSFluidPint.
CubicMetersPerUSFluidQuart (see page 2020)	This is constant StdConvs.CubicMetersPerUSFluidQuart.
CubicMetersPerUSFluidTablespoon (see page 2021)	This is constant StdConvs.CubicMetersPerUSFluidTablespoon.
CubicMetersPerUSFluidTeaspoon (see page 2021)	This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.
GramsPerDrams (see page 2021)	This is constant StdConvs.GramsPerDrams.
GramsPerGrains (see page 2022)	This is constant StdConvs.GramsPerGrains.
GramsPerLongTons (see page 2022)	This is constant StdConvs.GramsPerLongTons.
GramsPerOunces (see page 2022)	This is constant StdConvs.GramsPerOunces.
GramsPerPound (see page 2023)	This is constant StdConvs.GramsPerPound.
GramsPerStones (see page 2023)	This is constant StdConvs.GramsPerStones.
GramsPerTons (see page 2023)	This is constant StdConvs.GramsPerTons.
MetersPerAstronomicalUnit (see page 2024)	This is constant StdConvs.MetersPerAstronomicalUnit.
MetersPerChain (see page 2024)	This is constant StdConvs.MetersPerChain.
MetersPerCubit (see page 2024)	This is constant StdConvs.MetersPerCubit.
MetersPerFathom (see page 2025)	This is constant StdConvs.MetersPerFathom.
MetersPerFoot (see page 2025)	This is constant StdConvs.MetersPerFoot.
MetersPerFurlong (see page 2025)	This is constant StdConvs.MetersPerFurlong.
MetersPerHand (see page 2026)	This is constant StdConvs.MetersPerHand.
MetersPerInch (see page 2026)	This is constant StdConvs.MetersPerInch.
MetersPerLightSecond (see page 2026)	This is constant StdConvs.MetersPerLightSecond.
MetersPerLightYear (see page 2027)	This is constant StdConvs.MetersPerLightYear.
MetersPerLink (see page 2027)	This is constant StdConvs.MetersPerLink.
MetersPerMile (see page 2027)	This is constant StdConvs.MetersPerMile.
MetersPerNauticalMiles (see page 2028)	This is constant StdConvs.MetersPerNauticalMiles.
MetersPerPace (see page 2028)	This is constant StdConvs.MetersPerPace.
MetersPerParsec (see page 2028)	This is constant StdConvs.MetersPerParsec.
MetersPerPica (see page 2029)	This is constant StdConvs.MetersPerPica.
MetersPerPoint (see page 2029)	This is constant StdConvs.MetersPerPoint.
MetersPerRod (see page 2029)	This is constant StdConvs.MetersPerRod.
MetersPerYard (see page 2030)	This is constant StdConvs.MetersPerYard.
SquareMetersPerAcre (see page 2030)	This is constant StdConvs.SquareMetersPerAcre.
SquareMetersPerSquareFoot (see page 2030)	This is constant StdConvs.SquareMetersPerSquareFoot.
SquareMetersPerSquareInch (see page 2031)	This is constant StdConvs.SquareMetersPerSquareInch.
SquareMetersPerSquareMile (see page 2031)	This is constant StdConvs.SquareMetersPerSquareMile.
SquareMetersPerSquareRod (see page 2031)	This is constant StdConvs.SquareMetersPerSquareRod.
SquareMetersPerSquareYard (see page 2032)	This is constant StdConvs.SquareMetersPerSquareYard.

Functions

	Name	Description
	CelsiusToFahrenheit (see page 1958)	Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.
	FahrenheitToCelsius (see page 1958)	Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.
	InitStdConvs (see page 1959)	Initialize (see page 3599) conversion variables for C++ program.

Variables

Name	Description
auAcres (see page 1962)	This is variable StdConvs.auAcres.
auAres (see page 1963)	This is variable StdConvs.auAres.
auCentares (see page 1963)	This is variable StdConvs.auCentares.
auHectares (see page 1963)	This is variable StdConvs.auHectares.
auSquareCentimeters (see page 1964)	This is variable StdConvs.auSquareCentimeters.
auSquareDecameters (see page 1964)	This is variable StdConvs.auSquareDecameters.

auSquareDecimeters (see page 1964)	This is variable StdConvs.auSquareDecimeters.
auSquareFeet (see page 1965)	This is variable StdConvs.auSquareFeet.
auSquareHectometers (see page 1965)	This is variable StdConvs.auSquareHectometers.
auSquareInches (see page 1965)	This is variable StdConvs.auSquareInches.
auSquareKilometers (see page 1966)	This is variable StdConvs.auSquareKilometers.
auSquareMeters (see page 1966)	This is variable StdConvs.auSquareMeters.
auSquareMiles (see page 1966)	This is variable StdConvs.auSquareMiles.
auSquareMillimeters (see page 1967)	This is variable StdConvs.auSquareMillimeters.
auSquareRods (see page 1967)	This is variable StdConvs.auSquareRods.
auSquareYards (see page 1967)	This is variable StdConvs.auSquareYards.
cbArea (see page 1968)	Represent units that measure cbArea.
cbDistance (see page 1968)	Represent units that measure cbDistance.
cbMass (see page 1969)	This is variable StdConvs.cbMass.
cbTemperature (see page 1969)	This is variable StdConvs.cbTemperature.
cbTime (see page 1969)	Represent units that measure cbTime.
cbVolume (see page 1970)	Represent units that measure volume.
duAngstroms (see page 1971)	This is variable StdConvs.duAngstroms.
duAstronomicalUnits (see page 1971)	This is variable StdConvs.duAstronomicalUnits.
duCentimeters (see page 1972)	This is variable StdConvs.duCentimeters.
duChains (see page 1972)	This is variable StdConvs.duChains.
duCubits (see page 1972)	This is variable StdConvs.duCubits.
duDecameters (see page 1973)	This is variable StdConvs.duDecameters.
duDecimeters (see page 1973)	This is variable StdConvs.duDecimeters.
duFathoms (see page 1973)	This is variable StdConvs.duFathoms.
duFeet (see page 1974)	This is variable StdConvs.duFeet.
duFurlongs (see page 1974)	This is variable StdConvs.duFurlongs.
duGigameters (see page 1974)	This is variable StdConvs.duGigameters.
duHands (see page 1975)	This is variable StdConvs.duHands.
duHectometers (see page 1975)	This is variable StdConvs.duHectometers.
duInches (see page 1975)	This is variable StdConvs.duInches.
duKilometers (see page 1976)	This is variable StdConvs.duKilometers.
duLightYears (see page 1976)	This is variable StdConvs.duLightYears.
duLinks (see page 1976)	This is variable StdConvs.duLinks.
duMegameters (see page 1977)	This is variable StdConvs.duMegameters.
duMeters (see page 1977)	This is variable StdConvs.duMeters.
duMicromicrons (see page 1977)	This is variable StdConvs.duMicromicrons.
duMicrons (see page 1978)	This is variable StdConvs.duMicrons.
duMiles (see page 1978)	This is variable StdConvs.duMiles.
duMillimeters (see page 1978)	This is variable StdConvs.duMillimeters.
duMillimicrons (see page 1979)	This is variable StdConvs.duMillimicrons.
duNauticalMiles (see page 1979)	This is variable StdConvs.duNauticalMiles.
duPaces (see page 1979)	This is variable StdConvs.duPaces.
duParsecs (see page 1980)	This is variable StdConvs.duParsecs.
duPicas (see page 1980)	This is variable StdConvs.duPicas.
duPoints (see page 1980)	This is variable StdConvs.duPoints.
duRods (see page 1981)	This is variable StdConvs.duRods.
duYards (see page 1981)	This is variable StdConvs.duYards.
muCentigrams (see page 1981)	This is variable StdConvs.muCentigrams.
muDecagrams (see page 1982)	This is variable StdConvs.muDecagrams.
muDecigrams (see page 1982)	This is variable StdConvs.muDecigrams.
muDrams (see page 1982)	This is variable StdConvs.muDrams.
muGrains (see page 1983)	This is variable StdConvs.muGrains.
muGrams (see page 1983)	This is variable StdConvs.muGrams.
muHectograms (see page 1983)	This is variable StdConvs.muHectograms.

muKilograms (see page 1984)	This is variable StdConvs.muKilograms.
muLongTons (see page 1984)	This is variable StdConvs.muLongTons.
muMetricTons (see page 1984)	This is variable StdConvs.muMetricTons.
muMicrograms (see page 1985)	This is variable StdConvs.muMicrograms.
muMilligrams (see page 1985)	This is variable StdConvs.muMilligrams.
muNanograms (see page 1985)	This is variable StdConvs.muNanograms.
muOunces (see page 1986)	This is variable StdConvs.muOunces.
muPounds (see page 1986)	This is variable StdConvs.muPounds.
muStones (see page 1986)	This is variable StdConvs.muStones.
muTons (see page 1987)	This is variable StdConvs.muTons.
tuCelsius (see page 1987)	This is variable StdConvs.tuCelsius.
tuCenturies (see page 1987)	This is variable StdConvs.tuCenturies.
tuDateTime (see page 1988)	This is variable StdConvs.tuDateTime.
tuDays (see page 1988)	This is variable StdConvs.tuDays.
tuDecades (see page 1988)	This is variable StdConvs.tuDecades.
tuFahrenheit (see page 1989)	This is variable StdConvs.tuFahrenheit.
tuFortnights (see page 1989)	This is variable StdConvs.tuFortnights.
tuHours (see page 1989)	This is variable StdConvs.tuHours.
tuJulianDate (see page 1990)	This is variable StdConvs.tuJulianDate.
tuKelvin (see page 1990)	This is variable StdConvs.tuKelvin.
tuMillennia (see page 1990)	This is variable StdConvs.tuMillennia.
tuMilliSeconds (see page 1991)	This is variable StdConvs.tuMilliSeconds.
tuMinutes (see page 1991)	This is variable StdConvs.tuMinutes.
tuModifiedJulianDate (see page 1991)	This is variable StdConvs.tuModifiedJulianDate.
tuMonths (see page 1992)	This is variable StdConvs.tuMonths.
tuRankine (see page 1992)	This is variable StdConvs.tuRankine.
tuReaumur (see page 1992)	This is variable StdConvs.tuReaumur.
tuSeconds (see page 1993)	This is variable StdConvs.tuSeconds.
tuWeeks (see page 1993)	This is variable StdConvs.tuWeeks.
tuYears (see page 1993)	This is variable StdConvs.tuYears.
vuAcreFeet (see page 1994)	This is variable StdConvs.vuAcreFeet.
vuAcreInches (see page 1994)	This is variable StdConvs.vuAcreInches.
vuCentiLiters (see page 1994)	This is variable StdConvs.vuCentiLiters.
vuCordFeet (see page 1995)	This is variable StdConvs.vuCordFeet.
vuCords (see page 1995)	This is variable StdConvs.vuCords.
vuCubicCentimeters (see page 1995)	This is variable StdConvs.vuCubicCentimeters.
vuCubicDecameters (see page 1996)	This is variable StdConvs.vuCubicDecameters.
vuCubicDecimeters (see page 1996)	This is variable StdConvs.vuCubicDecimeters.
vuCubicFeet (see page 1996)	This is variable StdConvs.vuCubicFeet.
vuCubicHectometers (see page 1997)	This is variable StdConvs.vuCubicHectometers.
vuCubicInches (see page 1997)	This is variable StdConvs.vuCubicInches.
vuCubicKilometers (see page 1997)	This is variable StdConvs.vuCubicKilometers.
vuCubicMeters (see page 1998)	This is variable StdConvs.vuCubicMeters.
vuCubicMiles (see page 1998)	This is variable StdConvs.vuCubicMiles.
vuCubicMillimeters (see page 1998)	This is variable StdConvs.vuCubicMillimeters.
vuCubicYards (see page 1999)	This is variable StdConvs.vuCubicYards.
vuDecaLiters (see page 1999)	This is variable StdConvs.vuDecaLiters.
vuDecasteres (see page 1999)	This is variable StdConvs.vuDecasteres.
vuDeciLiters (see page 2000)	This is variable StdConvs.vuDeciLiters.
vuDecisteres (see page 2000)	This is variable StdConvs.vuDecisteres.
vuDryBuckets (see page 2000)	This is variable StdConvs.vuDryBuckets.
vuDryBushels (see page 2001)	This is variable StdConvs.vuDryBushels.
vuDryGallons (see page 2001)	This is variable StdConvs.vuDryGallons.
vuDryPecks (see page 2001)	This is variable StdConvs.vuDryPecks.

vuDryPints (see page 2002)	This is variable StdConvs.vuDryPints.
vuDryQuarts (see page 2002)	This is variable StdConvs.vuDryQuarts.
vuFluidCups (see page 2002)	This is variable StdConvs.vuFluidCups.
vuFluidGallons (see page 2003)	This is variable StdConvs.vuFluidGallons.
vuFluidGills (see page 2003)	This is variable StdConvs.vuFluidGills.
vuFluidOunces (see page 2003)	This is variable StdConvs.vuFluidOunces.
vuFluidPints (see page 2004)	This is variable StdConvs.vuFluidPints.
vuFluidQuarts (see page 2004)	This is variable StdConvs.vuFluidQuarts.
vuFluidTablespoons (see page 2004)	This is variable StdConvs.vuFluidTablespoons.
vuFluidTeaspoons (see page 2005)	This is variable StdConvs.vuFluidTeaspoons.
vuHectoLiters (see page 2005)	This is variable StdConvs.vuHectoLiters.
vuKiloLiters (see page 2005)	This is variable StdConvs.vuKiloLiters.
vuLiters (see page 2006)	This is variable StdConvs.vuLiters.
vuMilliLiters (see page 2006)	This is variable StdConvs.vuMilliLiters.
vuSteres (see page 2006)	This is variable StdConvs.vuSteres.
vuUKBuckets (see page 2007)	This is variable StdConvs.vuUKBuckets.
vuUKBushels (see page 2007)	This is variable StdConvs.vuUKBushels.
vuUKGallons (see page 2007)	This is variable StdConvs.vuUKGallons.
vuUKGills (see page 2008)	This is variable StdConvs.vuUKGills.
vuUKOunces (see page 2008)	This is variable StdConvs.vuUKOunces.
vuUKPecks (see page 2008)	This is variable StdConvs.vuUKPecks.
vuUKPints (see page 2009)	This is variable StdConvs.vuUKPints.
vuUKPottles (see page 2009)	This is variable StdConvs.vuUKPottles.
vuUKQuarts (see page 2009)	This is variable StdConvs.vuUKQuarts.

29.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CelsiusToFahrenheit (see page 1958)	Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.
	FahrenheitToCelsius (see page 1958)	Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.
	InitStdConvs (see page 1959)	Initialize (see page 3599) conversion variables for C++ program.

29.1.1 StdConvs.CelsiusToFahrenheit Function

Converts a temperature expressed in degrees Celsius to the corresponding temperature in degrees Fahrenheit.

Pascal

```
function CelsiusToFahrenheit(const AValue: Double): Double;
```

C++

```
Double CelsiusToFahrenheit(const Double AValue);
```

File

StdConvs

Description

Call CelsiusToFahrenheit to convert the temperature specified by AValue, which is in degrees Celsius, to the corresponding temperature in degrees Fahrenheit.

See Also

FahrenheitToCelsius (see page 1958)

Convert

29.1.2 StdConvs.FahrenheitToCelsius Function

Converts a temperature expressed in degrees Fahrenheit to the corresponding temperature in degrees Celsius.

Pascal

```
function FahrenheitToCelsius(const AValue: Double): Double;
```

C++

```
Double FahrenheitToCelsius(const Double AValue);
```

File

StdConvs

Description

Call FahrenheitToCelsius to convert the temperature specified by AValue, which is in degrees Fahrenheit, to the corresponding temperature in degrees Celsius.

See Also

CelsiusToFahrenheit (see page 1958)

Convert

29.1.3 StdConvs.InitStdConvs Function

Initialize (see page 3599) conversion variables for C++ program.

Pascal

```
procedure InitStdConvs;
```

C++

```
InitStdConvs();
```

File

StdConvs

Description

Insert (see page 3601) InitStdConvs into a C++ program to ensure that StdConvs (see page 1953) global variables are properly initialized. InitStdConvs does nothing and does not need to be called. A reference to InitStdConvs in a C++ program guarantees that the linker will provide a call to the unit's initialization code, even if no other StdConvs (see page 1953) routines are present. This step is not necessary in Delphi code, which always calls the initialization code for referenced units.

See Also

FahrenheitToCelsius (see page 1958)

Convert

29.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
auAcres (see page 1962)	This is variable StdConvs.auAcres.
auAres (see page 1963)	This is variable StdConvs.auAres.
auCentares (see page 1963)	This is variable StdConvs.auCentares.
auHectares (see page 1963)	This is variable StdConvs.auHectares.
auSquareCentimeters (see page 1964)	This is variable StdConvs.auSquareCentimeters.
auSquareDecameters (see page 1964)	This is variable StdConvs.auSquareDecameters.
auSquareDecimeters (see page 1964)	This is variable StdConvs.auSquareDecimeters.
auSquareFeet (see page 1965)	This is variable StdConvs.auSquareFeet.
auSquareHectometers (see page 1965)	This is variable StdConvs.auSquareHectometers.
auSquareInches (see page 1965)	This is variable StdConvs.auSquareInches.
auSquareKilometers (see page 1966)	This is variable StdConvs.auSquareKilometers.
auSquareMeters (see page 1966)	This is variable StdConvs.auSquareMeters.
auSquareMiles (see page 1966)	This is variable StdConvs.auSquareMiles.
auSquareMillimeters (see page 1967)	This is variable StdConvs.auSquareMillimeters.
auSquareRods (see page 1967)	This is variable StdConvs.auSquareRods.
auSquareYards (see page 1967)	This is variable StdConvs.auSquareYards.
cbArea (see page 1968)	Represent units that measure cbArea.
cbDistance (see page 1968)	Represent units that measure cbDistance.
cbMass (see page 1969)	This is variable StdConvs.cbMass.
cbTemperature (see page 1969)	This is variable StdConvs.cbTemperature.
cbTime (see page 1969)	Represent units that measure cbTime.
cbVolume (see page 1970)	Represent units that measure volume.
duAngstroms (see page 1971)	This is variable StdConvs.duAngstroms.
duAstronomicalUnits (see page 1971)	This is variable StdConvs.duAstronomicalUnits.
duCentimeters (see page 1972)	This is variable StdConvs.duCentimeters.
duChains (see page 1972)	This is variable StdConvs.duChains.
duCubits (see page 1972)	This is variable StdConvs.duCubits.
duDecameters (see page 1973)	This is variable StdConvs.duDecameters.
duDecimeters (see page 1973)	This is variable StdConvs.duDecimeters.
duFathoms (see page 1973)	This is variable StdConvs.duFathoms.
duFeet (see page 1974)	This is variable StdConvs.duFeet.
duFurlongs (see page 1974)	This is variable StdConvs.duFurlongs.
duGigameters (see page 1974)	This is variable StdConvs.duGigameters.
duHands (see page 1975)	This is variable StdConvs.duHands.
duHectometers (see page 1975)	This is variable StdConvs.duHectometers.
duInches (see page 1975)	This is variable StdConvs.duInches.
duKilometers (see page 1976)	This is variable StdConvs.duKilometers.
duLightYears (see page 1976)	This is variable StdConvs.duLightYears.
duLinks (see page 1976)	This is variable StdConvs.duLinks.
duMegameters (see page 1977)	This is variable StdConvs.duMegameters.
duMeters (see page 1977)	This is variable StdConvs.duMeters.
duMicromicrons (see page 1977)	This is variable StdConvs.duMicromicrons.
duMicrons (see page 1978)	This is variable StdConvs.duMicrons.
duMiles (see page 1978)	This is variable StdConvs.duMiles.

duMillimeters (see page 1978)	This is variable StdConvs.duMillimeters.
duMillimicrons (see page 1979)	This is variable StdConvs.duMillimicrons.
duNauticalMiles (see page 1979)	This is variable StdConvs.duNauticalMiles.
duPaces (see page 1979)	This is variable StdConvs.duPaces.
duParsecs (see page 1980)	This is variable StdConvs.duParsecs.
duPicas (see page 1980)	This is variable StdConvs.duPicas.
duPoints (see page 1980)	This is variable StdConvs.duPoints.
duRods (see page 1981)	This is variable StdConvs.duRods.
duYards (see page 1981)	This is variable StdConvs.duYards.
muCentigrams (see page 1981)	This is variable StdConvs.muCentigrams.
muDecagrams (see page 1982)	This is variable StdConvs.muDecagrams.
muDecigrams (see page 1982)	This is variable StdConvs.muDecigrams.
muDrams (see page 1982)	This is variable StdConvs.muDrams.
muGrains (see page 1983)	This is variable StdConvs.muGrains.
muGrams (see page 1983)	This is variable StdConvs.muGrams.
muHectograms (see page 1983)	This is variable StdConvs.muHectograms.
muKilograms (see page 1984)	This is variable StdConvs.muKilograms.
muLongTons (see page 1984)	This is variable StdConvs.muLongTons.
muMetricTons (see page 1984)	This is variable StdConvs.muMetricTons.
muMicrograms (see page 1985)	This is variable StdConvs.muMicrograms.
muMilligrams (see page 1985)	This is variable StdConvs.muMilligrams.
muNanograms (see page 1985)	This is variable StdConvs.muNanograms.
muOunces (see page 1986)	This is variable StdConvs.muOunces.
muPounds (see page 1986)	This is variable StdConvs.muPounds.
muStones (see page 1986)	This is variable StdConvs.muStones.
muTons (see page 1987)	This is variable StdConvs.muTons.
tuCelsius (see page 1987)	This is variable StdConvs.tuCelsius.
tuCenturies (see page 1987)	This is variable StdConvs.tuCenturies.
tuDateTime (see page 1988)	This is variable StdConvs.tuDateTime.
tuDays (see page 1988)	This is variable StdConvs.tuDays.
tuDecades (see page 1988)	This is variable StdConvs.tuDecades.
tuFahrenheit (see page 1989)	This is variable StdConvs.tuFahrenheit.
tuFortnights (see page 1989)	This is variable StdConvs.tuFortnights.
tuHours (see page 1989)	This is variable StdConvs.tuHours.
tuJulianDate (see page 1990)	This is variable StdConvs.tuJulianDate.
tuKelvin (see page 1990)	This is variable StdConvs.tuKelvin.
tuMillennia (see page 1990)	This is variable StdConvs.tuMillennia.
tuMilliSeconds (see page 1991)	This is variable StdConvs.tuMilliSeconds.
tuMinutes (see page 1991)	This is variable StdConvs.tuMinutes.
tuModifiedJulianDate (see page 1991)	This is variable StdConvs.tuModifiedJulianDate.
tuMonths (see page 1992)	This is variable StdConvs.tuMonths.
tuRankine (see page 1992)	This is variable StdConvs.tuRankine.
tuReaumur (see page 1992)	This is variable StdConvs.tuReaumur.
tuSeconds (see page 1993)	This is variable StdConvs.tuSeconds.
tuWeeks (see page 1993)	This is variable StdConvs.tuWeeks.
tuYears (see page 1993)	This is variable StdConvs.tuYears.
vuAcreFeet (see page 1994)	This is variable StdConvs.vuAcreFeet.
vuAcreInches (see page 1994)	This is variable StdConvs.vuAcreInches.
vuCentiLiters (see page 1994)	This is variable StdConvs.vuCentiLiters.
vuCordFeet (see page 1995)	This is variable StdConvs.vuCordFeet.
vuCords (see page 1995)	This is variable StdConvs.vuCords.
vuCubicCentimeters (see page 1995)	This is variable StdConvs.vuCubicCentimeters.
vuCubicDecameters (see page 1996)	This is variable StdConvs.vuCubicDecameters.
vuCubicDecimeters (see page 1996)	This is variable StdConvs.vuCubicDecimeters.

vuCubicFeet (see page 1996)	This is variable StdConvs.vuCubicFeet.
vuCubicHectometers (see page 1997)	This is variable StdConvs.vuCubicHectometers.
vuCubicInches (see page 1997)	This is variable StdConvs.vuCubicInches.
vuCubicKilometers (see page 1997)	This is variable StdConvs.vuCubicKilometers.
vuCubicMeters (see page 1998)	This is variable StdConvs.vuCubicMeters.
vuCubicMiles (see page 1998)	This is variable StdConvs.vuCubicMiles.
vuCubicMillimeters (see page 1998)	This is variable StdConvs.vuCubicMillimeters.
vuCubicYards (see page 1999)	This is variable StdConvs.vuCubicYards.
vuDecaLiters (see page 1999)	This is variable StdConvs.vuDecaLiters.
vuDecasteres (see page 1999)	This is variable StdConvs.vuDecasteres.
vuDeciLiters (see page 2000)	This is variable StdConvs.vuDeciLiters.
vuDecisteres (see page 2000)	This is variable StdConvs.vuDecisteres.
vuDryBuckets (see page 2000)	This is variable StdConvs.vuDryBuckets.
vuDryBushels (see page 2001)	This is variable StdConvs.vuDryBushels.
vuDryGallons (see page 2001)	This is variable StdConvs.vuDryGallons.
vuDryPecks (see page 2001)	This is variable StdConvs.vuDryPecks.
vuDryPints (see page 2002)	This is variable StdConvs.vuDryPints.
vuDryQuarts (see page 2002)	This is variable StdConvs.vuDryQuarts.
vuFluidCups (see page 2002)	This is variable StdConvs.vuFluidCups.
vuFluidGallons (see page 2003)	This is variable StdConvs.vuFluidGallons.
vuFluidGills (see page 2003)	This is variable StdConvs.vuFluidGills.
vuFluidOunces (see page 2003)	This is variable StdConvs.vuFluidOunces.
vuFluidPints (see page 2004)	This is variable StdConvs.vuFluidPints.
vuFluidQuarts (see page 2004)	This is variable StdConvs.vuFluidQuarts.
vuFluidTablespoons (see page 2004)	This is variable StdConvs.vuFluidTablespoons.
vuFluidTeaspoons (see page 2005)	This is variable StdConvs.vuFluidTeaspoons.
vuHectoLiters (see page 2005)	This is variable StdConvs.vuHectoLiters.
vuKiloLiters (see page 2005)	This is variable StdConvs.vuKiloLiters.
vuLiters (see page 2006)	This is variable StdConvs.vuLiters.
vuMilliLiters (see page 2006)	This is variable StdConvs.vuMilliLiters.
vuSteres (see page 2006)	This is variable StdConvs.vuSteres.
vuUKBuckets (see page 2007)	This is variable StdConvs.vuUKBuckets.
vuUKBushels (see page 2007)	This is variable StdConvs.vuUKBushels.
vuUKGallons (see page 2007)	This is variable StdConvs.vuUKGallons.
vuUKGills (see page 2008)	This is variable StdConvs.vuUKGills.
vuUKOunces (see page 2008)	This is variable StdConvs.vuUKOunces.
vuUKPecks (see page 2008)	This is variable StdConvs.vuUKPecks.
vuUKPints (see page 2009)	This is variable StdConvs.vuUKPints.
vuUKPottles (see page 2009)	This is variable StdConvs.vuUKPottles.
vuUKQuarts (see page 2009)	This is variable StdConvs.vuUKQuarts.

29.2.1 StdConvs.auAcres Variable

Pascal

```
auAcres: TConvType;
```

C++

```
TConvType auAcres;
```

File

StdConvs

Description

This is variable StdConvs.auAcres.

29.2.2 StdConvs.auAres Variable

Pascal

```
auAres: TConvType;
```

C++

```
TConvType auAres;
```

File

StdConvs

Description

This is variable StdConvs.auAres.

29.2.3 StdConvs.auCentares Variable

Pascal

```
auCentares: TConvType;
```

C++

```
TConvType auCentares;
```

File

StdConvs

Description

This is variable StdConvs.auCentares.

29.2.4 StdConvs.auHectares Variable

Pascal

```
auHectares: TConvType;
```

C++

```
TConvType auHectares;
```

File

StdConvs

Description

This is variable StdConvs.auHectares.

29.2.5 StdConvs.auSquareCentimeters Variable

Pascal

```
auSquareCentimeters: TConvType;
```

C++

```
TConvType auSquareCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareCentimeters.

29.2.6 StdConvs.auSquareDecameters Variable

Pascal

```
auSquareDecameters: TConvType;
```

C++

```
TConvType auSquareDecameters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareDecameters.

29.2.7 StdConvs.auSquareDecimeters Variable

Pascal

```
auSquareDecimeters: TConvType;
```

C++

```
TConvType auSquareDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareDecimeters.

29.2.8 StdConvs.auSquareFeet Variable

Pascal

```
auSquareFeet: TConvType;
```

C++

```
TConvType auSquareFeet;
```

File

StdConvs

Description

This is variable StdConvs.auSquareFeet.

29.2.9 StdConvs.auSquareHectometers Variable

Pascal

```
auSquareHectometers: TConvType;
```

C++

```
TConvType auSquareHectometers;
```

File

StdConvs

Description

This is variable StdConvs.auSquareHectometers.

29.2.10 StdConvs.auSquareInches Variable

Pascal

```
auSquareInches: TConvType;
```

C++

```
TConvType auSquareInches;
```

File

StdConvs

Description

This is variable StdConvs.auSquareInches.

29.2.11 StdConvs.auSquareKilometers Variable

Pascal

```
auSquareKilometers: TConvType;
```

C++

```
TConvType auSquareKilometers;
```

File

StdConvs

Description

This is variable StdConvs.auSquareKilometers.

29.2.12 StdConvs.auSquareMeters Variable

Pascal

```
auSquareMeters: TConvType;
```

C++

```
TConvType auSquareMeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMeters.

29.2.13 StdConvs.auSquareMiles Variable

Pascal

```
auSquareMiles: TConvType;
```

C++

```
TConvType auSquareMiles;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMiles.

29.2.14 StdConvs.auSquareMillimeters Variable

Pascal

```
auSquareMillimeters: TConvType;
```

C++

```
TConvType auSquareMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.auSquareMillimeters.

29.2.15 StdConvs.auSquareRods Variable

Pascal

```
auSquareRods: TConvType;
```

C++

```
TConvType auSquareRods;
```

File

StdConvs

Description

This is variable StdConvs.auSquareRods.

29.2.16 StdConvs.auSquareYards Variable

Pascal

```
auSquareYards: TConvType;
```

C++

```
TConvType auSquareYards;
```

File

StdConvs

Description

This is variable StdConvs.auSquareYards.

29.2.17 StdConvs.cbArea Variable

Represent units that measure cbArea.

Pascal

```
cbArea: TConvFamily;
```

C++

```
TConvFamily cbArea;
```

File

StdConvs

Description

Use the cbArea unit variables as parameters to the Convert function to convert an cbArea measurement to a different set of units. The StdConvs (see page 1953) unit registers these cbArea measurement units in the cbArea family and assigns the resulting TConvType values to these global variables.

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1959) procedure.

See Also

Convert

RegisterConversionType

InitStdConvs (see page 1959)

TConvType

29.2.18 StdConvs.cbDistance Variable

Represent units that measure cbDistance.

Pascal

```
cbDistance: TConvFamily;
```

C++

```
TConvFamily cbDistance;
```

File

StdConvs

Description

Use the cbDistance unit variables as parameters to the Convert function to convert a cbDistance measurement to a different set of units. The StdConvs (see page 1953) unit registers these cbDistance measurement units in the cbDistance family and assigns the resulting TConvType values to these global variables.

Note: The cbDistance units vary greatly in scale. If you are converting between cbDistances that are all at one extreme (for example between parsecs and light years or between angstroms and microns), you may want to register your own conversion family (for example astronomical distances or microscopic distances) so that the conversions can be made with greater precision.

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1959) procedure.

See Also

Convert

RegisterConversionType

InitStdConvs (see page 1959)

TConvType

29.2.19 StdConvs.cbMass Variable

Pascal

```
cbMass: TConvFamily;
```

C++

```
TConvFamily cbMass;
```

File

StdConvs

Description

This is variable StdConvs.cbMass.

29.2.20 StdConvs.cbTemperature Variable

Pascal

```
cbTemperature: TConvFamily;
```

C++

```
TConvFamily cbTemperature;
```

File

StdConvs

Description

This is variable StdConvs.cbTemperature.

29.2.21 StdConvs.cbTime Variable

Represent units that measure cbTime.

Pascal

```
cbTime: TConvFamily;
```

C++

```
TConvFamily cbTime;
```

File

StdConvs

Description

Use the cbTime unit variables as parameters to the Convert function to convert a cbTime measurement to a different set of units. The StdConvs (see page 1953) unit registers these cbTime measurement units in the cbTime family and assigns the resulting TConvType values to these global variables.

Note: Conversions involving tuMonths (see page 1992), tuYears (see page 1993), tuDecades (see page 1988), tuCenturies (see page 1987), and tuMillennia (see page 1990) are approximations, due to the fact that not all months nor all years are the same length. If you are only using units that are multiples of years, you may want to register your own conversion family that uses years as a base unit so that conversions are more accurate.

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1959) procedure.

See Also

Convert

RegisterConversionType

FahrenheitToCelsius (see page 1958)

CelsiusToFahrenheit (see page 1958)

InitStdConvs (see page 1959)

TConvType

29.2.22 StdConvs.cbVolume Variable

Represent units that measure volume.

Pascal

```
cbVolume: TConvFamily;
```

C++

```
TConvFamily cbVolume;
```

File

StdConvs

Description

Use the cbVolume unit variables as parameters to the Convert function to convert a cbVolume measurement to a different set of units. The StdConvs (see page 1953) unit registers these cbVolume measurement units in the cbVolume family and assigns the resulting TConvType values to these global variables.

The interpretation of cbVolume measurements (ounces, gills, pints, quarts, gallons, pecks, bushels, and buckets) varies, depending on whether you are using American or British Imperial measurements. The vuFluidOunces (see page 2003), vuFluidGills (see page 2003), vuFluidPints (see page 2004), vuFluidQuarts (see page 2004), vuFluidGallons (see page 2003), vuDryPecks (see page 2001), vuDryBushels (see page 2001), and vuDryBuckets (see page 2000) variables refer

to the American units, while the vuUKOunces (see page 2008), vuUKGills (see page 2008), vuUKPints (see page 2009), vuUKQuarts (see page 2009), vuUKGallons (see page 2007), vuUKPecks (see page 2008), vuUKBushels (see page 2007), and vuUKBuckets (see page 2007) variables refer to the British Imperial units. Similarly, certain American dry measurements have the same names as American fluid cbVolume measurements (dry pints, dry quarts, and dry gallons), and are distinguished by using the variables vuDryPints (see page 2002), vuDryQuarts (see page 2002), and vuDryGallons (see page 2001).

Note: To initialize these variables in C++ programs, use the InitStdConvs (see page 1959) procedure.

See Also

Convert

RegisterConversionType

InitStdConvs (see page 1959)

TConvType

29.2.23 StdConvs.duAngstroms Variable

Pascal

```
duAngstroms: TConvType;
```

C++

```
TConvType duAngstroms;
```

File

StdConvs

Description

This is variable StdConvs.duAngstroms.

29.2.24 StdConvs.duAstronomicalUnits Variable

Pascal

```
duAstronomicalUnits: TConvType;
```

C++

```
TConvType duAstronomicalUnits;
```

File

StdConvs

Description

This is variable StdConvs.duAstronomicalUnits.

29.2.25 StdConvs.duCentimeters Variable

Pascal

```
duCentimeters: TConvType;
```

C++

```
TConvType duCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.duCentimeters.

29.2.26 StdConvs.duChains Variable

Pascal

```
duChains: TConvType;
```

C++

```
TConvType duChains;
```

File

StdConvs

Description

This is variable StdConvs.duChains.

29.2.27 StdConvs.duCubits Variable

Pascal

```
duCubits: TConvType;
```

C++

```
TConvType duCubits;
```

File

StdConvs

Description

This is variable StdConvs.duCubits.

29.2.28 StdConvs.duDecameters Variable

Pascal

```
duDecameters: TConvType;
```

C++

```
TConvType duDecameters;
```

File

StdConvs

Description

This is variable StdConvs.duDecameters.

29.2.29 StdConvs.duDecimeters Variable

Pascal

```
duDecimeters: TConvType;
```

C++

```
TConvType duDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.duDecimeters.

29.2.30 StdConvs.duFathoms Variable

Pascal

```
duFathoms: TConvType;
```

C++

```
TConvType duFathoms;
```

File

StdConvs

Description

This is variable StdConvs.duFathoms.

29.2.31 StdConvs.duFeet Variable

Pascal

```
duFeet: TConvType;
```

C++

```
TConvType duFeet;
```

File

StdConvs

Description

This is variable StdConvs.duFeet.

29.2.32 StdConvs.duFurlongs Variable

Pascal

```
duFurlongs: TConvType;
```

C++

```
TConvType duFurlongs;
```

File

StdConvs

Description

This is variable StdConvs.duFurlongs.

29.2.33 StdConvs.duGigameters Variable

Pascal

```
duGigameters: TConvType;
```

C++

```
TConvType duGigameters;
```

File

StdConvs

Description

This is variable StdConvs.duGigameters.

29.2.34 StdConvs.duHands Variable

Pascal

```
duHands: TConvType;
```

C++

```
TConvType duHands;
```

File

StdConvs

Description

This is variable StdConvs.duHands.

29.2.35 StdConvs.duHectometers Variable

Pascal

```
duHectometers: TConvType;
```

C++

```
TConvType duHectometers;
```

File

StdConvs

Description

This is variable StdConvs.duHectometers.

29.2.36 StdConvs.duInches Variable

Pascal

```
duInches: TConvType;
```

C++

```
TConvType duInches;
```

File

StdConvs

Description

This is variable StdConvs.duInches.

29.2.37 StdConvs.duKilometers Variable

Pascal

```
duKilometers: TConvType;
```

C++

```
TConvType duKilometers;
```

File

StdConvs

Description

This is variable StdConvs.duKilometers.

29.2.38 StdConvs.duLightYears Variable

Pascal

```
duLightYears: TConvType;
```

C++

```
TConvType duLightYears;
```

File

StdConvs

Description

This is variable StdConvs.duLightYears.

29.2.39 StdConvs.duLinks Variable

Pascal

```
duLinks: TConvType;
```

C++

```
TConvType duLinks;
```

File

StdConvs

Description

This is variable StdConvs.duLinks.

29.2.40 StdConvs.duMegameters Variable

Pascal

```
duMegameters: TConvType;
```

C++

```
TConvType duMegameters;
```

File

StdConvs

Description

This is variable StdConvs.duMegameters.

29.2.41 StdConvs.duMeters Variable

Pascal

```
duMeters: TConvType;
```

C++

```
TConvType duMeters;
```

File

StdConvs

Description

This is variable StdConvs.duMeters.

29.2.42 StdConvs.duMicromicrons Variable

Pascal

```
duMicromicrons: TConvType;
```

C++

```
TConvType duMicromicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMicromicrons.

29.2.43 StdConvs.duMicrons Variable

Pascal

```
duMicrons: TConvType;
```

C++

```
TConvType duMicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMicrons.

29.2.44 StdConvs.duMiles Variable

Pascal

```
duMiles: TConvType;
```

C++

```
TConvType duMiles;
```

File

StdConvs

Description

This is variable StdConvs.duMiles.

29.2.45 StdConvs.duMillimeters Variable

Pascal

```
duMillimeters: TConvType;
```

C++

```
TConvType duMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.duMillimeters.

29.2.46 StdConvs.duMillimicrons Variable

Pascal

```
duMillimicrons: TConvType;
```

C++

```
TConvType duMillimicrons;
```

File

StdConvs

Description

This is variable StdConvs.duMillimicrons.

29.2.47 StdConvs.duNauticalMiles Variable

Pascal

```
duNauticalMiles: TConvType;
```

C++

```
TConvType duNauticalMiles;
```

File

StdConvs

Description

This is variable StdConvs.duNauticalMiles.

29.2.48 StdConvs.duPaces Variable

Pascal

```
duPaces: TConvType;
```

C++

```
TConvType duPaces;
```

File

StdConvs

Description

This is variable StdConvs.duPaces.

29.2.49 StdConvs.duParsecs Variable

Pascal

```
duParsecs: TConvType;
```

C++

```
TConvType duParsecs;
```

File

StdConvs

Description

This is variable StdConvs.duParsecs.

29.2.50 StdConvs.duPicas Variable

Pascal

```
duPicas: TConvType;
```

C++

```
TConvType duPicas;
```

File

StdConvs

Description

This is variable StdConvs.duPicas.

29.2.51 StdConvs.duPoints Variable

Pascal

```
duPoints: TConvType;
```

C++

```
TConvType duPoints;
```

File

StdConvs

Description

This is variable StdConvs.duPoints.

29.2.52 StdConvs.duRods Variable

Pascal

```
duRods: TConvType;
```

C++

```
TConvType duRods;
```

File

StdConvs

Description

This is variable StdConvs.duRods.

29.2.53 StdConvs.duYards Variable

Pascal

```
duYards: TConvType;
```

C++

```
TConvType duYards;
```

File

StdConvs

Description

This is variable StdConvs.duYards.

29.2.54 StdConvs.muCentigrams Variable

Pascal

```
muCentigrams: TConvType;
```

C++

```
TConvType muCentigrams;
```

File

StdConvs

Description

This is variable StdConvs.muCentigrams.

29.2.55 StdConvs.muDecagrams Variable

Pascal

```
muDecagrams: TConvType;
```

C++

```
TConvType muDecagrams;
```

File

StdConvs

Description

This is variable StdConvs.muDecagrams.

29.2.56 StdConvs.muDecigrams Variable

Pascal

```
muDecigrams: TConvType;
```

C++

```
TConvType muDecigrams;
```

File

StdConvs

Description

This is variable StdConvs.muDecigrams.

29.2.57 StdConvs.muDrams Variable

Pascal

```
muDrams: TConvType;
```

C++

```
TConvType muDrams;
```

File

StdConvs

Description

This is variable StdConvs.muDrams.

29.2.58 StdConvs.muGrains Variable

Pascal

```
muGrains: TConvType;
```

C++

```
TConvType muGrains;
```

File

StdConvs

Description

This is variable StdConvs.muGrains.

29.2.59 StdConvs.muGrams Variable

Pascal

```
muGrams: TConvType;
```

C++

```
TConvType muGrams;
```

File

StdConvs

Description

This is variable StdConvs.muGrams.

29.2.60 StdConvs.muHectograms Variable

Pascal

```
muHectograms: TConvType;
```

C++

```
TConvType muHectograms;
```

File

StdConvs

Description

This is variable StdConvs.muHectograms.

29.2.61 StdConvs.muKilograms Variable

Pascal

```
muKilograms: TConvType;
```

C++

```
TConvType muKilograms;
```

File

StdConvs

Description

This is variable StdConvs.muKilograms.

29.2.62 StdConvs.muLongTons Variable

Pascal

```
muLongTons: TConvType;
```

C++

```
TConvType muLongTons;
```

File

StdConvs

Description

This is variable StdConvs.muLongTons.

29.2.63 StdConvs.muMetricTons Variable

Pascal

```
muMetricTons: TConvType;
```

C++

```
TConvType muMetricTons;
```

File

StdConvs

Description

This is variable StdConvs.muMetricTons.

29.2.64 StdConvs.muMicrograms Variable

Pascal

```
muMicrograms: TConvType;
```

C++

```
TConvType muMicrograms;
```

File

StdConvs

Description

This is variable StdConvs.muMicrograms.

29.2.65 StdConvs.muMilligrams Variable

Pascal

```
muMilligrams: TConvType;
```

C++

```
TConvType muMilligrams;
```

File

StdConvs

Description

This is variable StdConvs.muMilligrams.

29.2.66 StdConvs.muNanograms Variable

Pascal

```
muNanograms: TConvType;
```

C++

```
TConvType muNanograms;
```

File

StdConvs

Description

This is variable StdConvs.muNanograms.

29.2.67 StdConvs.muOunces Variable

Pascal

```
muOunces: TConvType;
```

C++

```
TConvType muOunces;
```

File

StdConvs

Description

This is variable StdConvs.muOunces.

29.2.68 StdConvs.muPounds Variable

Pascal

```
muPounds: TConvType;
```

C++

```
TConvType muPounds;
```

File

StdConvs

Description

This is variable StdConvs.muPounds.

29.2.69 StdConvs.muStones Variable

Pascal

```
muStones: TConvType;
```

C++

```
TConvType muStones;
```

File

StdConvs

Description

This is variable StdConvs.muStones.

29.2.70 StdConvs.muTons Variable

Pascal

```
muTons: TConvType;
```

C++

```
TConvType muTons;
```

File

StdConvs

Description

This is variable StdConvs.muTons.

29.2.71 StdConvs.tuCelsius Variable

Pascal

```
tuCelsius: TConvType;
```

C++

```
TConvType tuCelsius;
```

File

StdConvs

Description

This is variable StdConvs.tuCelsius.

29.2.72 StdConvs.tuCenturies Variable

Pascal

```
tuCenturies: TConvType;
```

C++

```
TConvType tuCenturies;
```

File

StdConvs

Description

This is variable StdConvs.tuCenturies.

29.2.73 StdConvs.tuDateTime Variable

Pascal

```
tuDateTime: TConvType;
```

C++

```
TConvType tuDateTime;
```

File

StdConvs

Description

This is variable StdConvs.tuDateTime.

29.2.74 StdConvs.tuDays Variable

Pascal

```
tuDays: TConvType;
```

C++

```
TConvType tuDays;
```

File

StdConvs

Description

This is variable StdConvs.tuDays.

29.2.75 StdConvs.tuDecades Variable

Pascal

```
tuDecades: TConvType;
```

C++

```
TConvType tuDecades;
```

File

StdConvs

Description

This is variable StdConvs.tuDecades.

29.2.76 StdConvs.tuFahrenheit Variable

Pascal

```
tuFahrenheit: TConvType;
```

C++

```
TConvType tuFahrenheit;
```

File

StdConvs

Description

This is variable StdConvs.tuFahrenheit.

29.2.77 StdConvs.tuFortnights Variable

Pascal

```
tuFortnights: TConvType;
```

C++

```
TConvType tuFortnights;
```

File

StdConvs

Description

This is variable StdConvs.tuFortnights.

29.2.78 StdConvs.tuHours Variable

Pascal

```
tuHours: TConvType;
```

C++

```
TConvType tuHours;
```

File

StdConvs

Description

This is variable StdConvs.tuHours.

29.2.79 StdConvs.tuJulianDate Variable

Pascal

```
tuJulianDate: TConvType;
```

C++

```
TConvType tuJulianDate;
```

File

StdConvs

Description

This is variable StdConvs.tuJulianDate.

29.2.80 StdConvs.tuKelvin Variable

Pascal

```
tuKelvin: TConvType;
```

C++

```
TConvType tuKelvin;
```

File

StdConvs

Description

This is variable StdConvs.tuKelvin.

29.2.81 StdConvs.tuMillennia Variable

Pascal

```
tuMillennia: TConvType;
```

C++

```
TConvType tuMillennia;
```

File

StdConvs

Description

This is variable StdConvs.tuMillennia.

29.2.82 StdConvs.tuMilliSeconds Variable

Pascal

```
tuMilliSeconds: TConvType;
```

C++

```
TConvType tuMilliSeconds;
```

File

StdConvs

Description

This is variable StdConvs.tuMilliSeconds.

29.2.83 StdConvs.tuMinutes Variable

Pascal

```
tuMinutes: TConvType;
```

C++

```
TConvType tuMinutes;
```

File

StdConvs

Description

This is variable StdConvs.tuMinutes.

29.2.84 StdConvs.tuModifiedJulianDate Variable

Pascal

```
tuModifiedJulianDate: TConvType;
```

C++

```
TConvType tuModifiedJulianDate;
```

File

StdConvs

Description

This is variable StdConvs.tuModifiedJulianDate.

29.2.85 StdConvs.tuMonths Variable

Pascal

```
tuMonths: TConvType;
```

C++

```
TConvType tuMonths;
```

File

StdConvs

Description

This is variable StdConvs.tuMonths.

29.2.86 StdConvs.tuRankine Variable

Pascal

```
tuRankine: TConvType;
```

C++

```
TConvType tuRankine;
```

File

StdConvs

Description

This is variable StdConvs.tuRankine.

29.2.87 StdConvs.tuReaumur Variable

Pascal

```
tuReaumur: TConvType;
```

C++

```
TConvType tuReaumur;
```

File

StdConvs

Description

This is variable StdConvs.tuReaumur.

29.2.88 StdConvs.tuSeconds Variable

Pascal

```
tuSeconds: TConvType;
```

C++

```
TConvType tuSeconds;
```

File

StdConvs

Description

This is variable StdConvs.tuSeconds.

29.2.89 StdConvs.tuWeeks Variable

Pascal

```
tuWeeks: TConvType;
```

C++

```
TConvType tuWeeks;
```

File

StdConvs

Description

This is variable StdConvs.tuWeeks.

29.2.90 StdConvs.tuYears Variable

Pascal

```
tuYears: TConvType;
```

C++

```
TConvType tuYears;
```

File

StdConvs

Description

This is variable StdConvs.tuYears.

29.2.91 StdConvs.vuAcreFeet Variable

Pascal

```
vuAcreFeet: TConvType;
```

C++

```
TConvType vuAcreFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuAcreFeet.

29.2.92 StdConvs.vuAcreInches Variable

Pascal

```
vuAcreInches: TConvType;
```

C++

```
TConvType vuAcreInches;
```

File

StdConvs

Description

This is variable StdConvs.vuAcreInches.

29.2.93 StdConvs.vuCentiLiters Variable

Pascal

```
vuCentiLiters: TConvType;
```

C++

```
TConvType vuCentiLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuCentiLiters.

29.2.94 StdConvs.vuCordFeet Variable

Pascal

```
vuCordFeet: TConvType;
```

C++

```
TConvType vuCordFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuCordFeet.

29.2.95 StdConvs.vuCords Variable

Pascal

```
vuCords: TConvType;
```

C++

```
TConvType vuCords;
```

File

StdConvs

Description

This is variable StdConvs.vuCords.

29.2.96 StdConvs.vuCubicCentimeters Variable

Pascal

```
vuCubicCentimeters: TConvType;
```

C++

```
TConvType vuCubicCentimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicCentimeters.

29.2.97 StdConvs.vuCubicDecameters Variable

Pascal

```
vuCubicDecameters: TConvType;
```

C++

```
TConvType vuCubicDecameters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicDecameters.

29.2.98 StdConvs.vuCubicDecimeters Variable

Pascal

```
vuCubicDecimeters: TConvType;
```

C++

```
TConvType vuCubicDecimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicDecimeters.

29.2.99 StdConvs.vuCubicFeet Variable

Pascal

```
vuCubicFeet: TConvType;
```

C++

```
TConvType vuCubicFeet;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicFeet.

29.2.100 StdConvs.vuCubicHectometers Variable

Pascal

```
vuCubicHectometers: TConvType;
```

C++

```
TConvType vuCubicHectometers;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicHectometers.

29.2.101 StdConvs.vuCubicInches Variable

Pascal

```
vuCubicInches: TConvType;
```

C++

```
TConvType vuCubicInches;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicInches.

29.2.102 StdConvs.vuCubicKilometers Variable

Pascal

```
vuCubicKilometers: TConvType;
```

C++

```
TConvType vuCubicKilometers;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicKilometers.

29.2.103 StdConvs.vuCubicMeters Variable

Pascal

```
vuCubicMeters: TConvType;
```

C++

```
TConvType vuCubicMeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMeters.

29.2.104 StdConvs.vuCubicMiles Variable

Pascal

```
vuCubicMiles: TConvType;
```

C++

```
TConvType vuCubicMiles;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMiles.

29.2.105 StdConvs.vuCubicMillimeters Variable

Pascal

```
vuCubicMillimeters: TConvType;
```

C++

```
TConvType vuCubicMillimeters;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicMillimeters.

29.2.106 StdConvs.vuCubicYards Variable

Pascal

```
vuCubicYards: TConvType;
```

C++

```
TConvType vuCubicYards;
```

File

StdConvs

Description

This is variable StdConvs.vuCubicYards.

29.2.107 StdConvs.vuDecaLiters Variable

Pascal

```
vuDecaLiters: TConvType;
```

C++

```
TConvType vuDecaLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuDecaLiters.

29.2.108 StdConvs.vuDecasteres Variable

Pascal

```
vuDecasteres: TConvType;
```

C++

```
TConvType vuDecasteres;
```

File

StdConvs

Description

This is variable StdConvs.vuDecasteres.

29.2.109 StdConvs.vuDeciLiters Variable

Pascal

```
vuDeciLiters: TConvType;
```

C++

```
TConvType vuDeciLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuDeciLiters.

29.2.110 StdConvs.vuDecisteres Variable

Pascal

```
vuDecisteres: TConvType;
```

C++

```
TConvType vuDecisteres;
```

File

StdConvs

Description

This is variable StdConvs.vuDecisteres.

29.2.111 StdConvs.vuDryBuckets Variable

Pascal

```
vuDryBuckets: TConvType;
```

C++

```
TConvType vuDryBuckets;
```

File

StdConvs

Description

This is variable StdConvs.vuDryBuckets.

29.2.112 StdConvs.vuDryBushels Variable

Pascal

```
vuDryBushels: TConvType;
```

C++

```
TConvType vuDryBushels;
```

File

StdConvs

Description

This is variable StdConvs.vuDryBushels.

29.2.113 StdConvs.vuDryGallons Variable

Pascal

```
vuDryGallons: TConvType;
```

C++

```
TConvType vuDryGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuDryGallons.

29.2.114 StdConvs.vuDryPecks Variable

Pascal

```
vuDryPecks: TConvType;
```

C++

```
TConvType vuDryPecks;
```

File

StdConvs

Description

This is variable StdConvs.vuDryPecks.

29.2.115 StdConvs.vuDryPints Variable

Pascal

```
vuDryPints: TConvType;
```

C++

```
TConvType vuDryPints;
```

File

StdConvs

Description

This is variable StdConvs.vuDryPints.

29.2.116 StdConvs.vuDryQuarts Variable

Pascal

```
vuDryQuarts: TConvType;
```

C++

```
TConvType vuDryQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuDryQuarts.

29.2.117 StdConvs.vuFluidCups Variable

Pascal

```
vuFluidCups: TConvType;
```

C++

```
TConvType vuFluidCups;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidCups.

29.2.118 StdConvs.vuFluidGallons Variable

Pascal

```
vuFluidGallons: TConvType;
```

C++

```
TConvType vuFluidGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidGallons.

29.2.119 StdConvs.vuFluidGills Variable

Pascal

```
vuFluidGills: TConvType;
```

C++

```
TConvType vuFluidGills;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidGills.

29.2.120 StdConvs.vuFluidOunces Variable

Pascal

```
vuFluidOunces: TConvType;
```

C++

```
TConvType vuFluidOunces;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidOunces.

29.2.121 StdConvs.vuFluidPints Variable

Pascal

```
vuFluidPints: TConvType;
```

C++

```
TConvType vuFluidPints;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidPints.

29.2.122 StdConvs.vuFluidQuarts Variable

Pascal

```
vuFluidQuarts: TConvType;
```

C++

```
TConvType vuFluidQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidQuarts.

29.2.123 StdConvs.vuFluidTablespoons Variable

Pascal

```
vuFluidTablespoons: TConvType;
```

C++

```
TConvType vuFluidTablespoons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidTablespoons.

29.2.124 StdConvs.vuFluidTeaspoons Variable

Pascal

```
vuFluidTeaspoons: TConvType;
```

C++

```
TConvType vuFluidTeaspoons;
```

File

StdConvs

Description

This is variable StdConvs.vuFluidTeaspoons.

29.2.125 StdConvs.vuHectoLiters Variable

Pascal

```
vuHectoLiters: TConvType;
```

C++

```
TConvType vuHectoLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuHectoLiters.

29.2.126 StdConvs.vuKiloLiters Variable

Pascal

```
vuKiloLiters: TConvType;
```

C++

```
TConvType vuKiloLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuKiloLiters.

29.2.127 StdConvs.vuLiters Variable

Pascal

```
vuLiters: TConvType;
```

C++

```
TConvType vuLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuLiters.

29.2.128 StdConvs.vuMilliLiters Variable

Pascal

```
vuMilliLiters: TConvType;
```

C++

```
TConvType vuMilliLiters;
```

File

StdConvs

Description

This is variable StdConvs.vuMilliLiters.

29.2.129 StdConvs.vuSteres Variable

Pascal

```
vuSteres: TConvType;
```

C++

```
TConvType vuSteres;
```

File

StdConvs

Description

This is variable StdConvs.vuSteres.

29.2.130 StdConvs.vuUKBuckets Variable

Pascal

```
vuUKBuckets: TConvType;
```

C++

```
TConvType vuUKBuckets;
```

File

StdConvs

Description

This is variable StdConvs.vuUKBuckets.

29.2.131 StdConvs.vuUKBushels Variable

Pascal

```
vuUKBushels: TConvType;
```

C++

```
TConvType vuUKBushels;
```

File

StdConvs

Description

This is variable StdConvs.vuUKBushels.

29.2.132 StdConvs.vuUKGallons Variable

Pascal

```
vuUKGallons: TConvType;
```

C++

```
TConvType vuUKGallons;
```

File

StdConvs

Description

This is variable StdConvs.vuUKGallons.

29.2.133 StdConvs.vuUKGills Variable

Pascal

```
vuUKGills: TConvType;
```

C++

```
TConvType vuUKGills;
```

File

StdConvs

Description

This is variable StdConvs.vuUKGills.

29.2.134 StdConvs.vuUKOunces Variable

Pascal

```
vuUKOunces: TConvType;
```

C++

```
TConvType vuUKOunces;
```

File

StdConvs

Description

This is variable StdConvs.vuUKOunces.

29.2.135 StdConvs.vuUKPecks Variable

Pascal

```
vuUKPecks: TConvType;
```

C++

```
TConvType vuUKPecks;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPecks.

29.2.136 StdConvs.vuUKPints Variable

Pascal

```
vuUKPints: TConvType;
```

C++

```
TConvType vuUKPints;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPints.

29.2.137 StdConvs.vuUKPottles Variable

Pascal

```
vuUKPottles: TConvType;
```

C++

```
TConvType vuUKPottles;
```

File

StdConvs

Description

This is variable StdConvs.vuUKPottles.

29.2.138 StdConvs.vuUKQuarts Variable

Pascal

```
vuUKQuarts: TConvType;
```

C++

```
TConvType vuUKQuarts;
```

File

StdConvs

Description

This is variable StdConvs.vuUKQuarts.

29.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CubicMetersPerAcreFoot (see page 2011)	This is constant StdConvs.CubicMetersPerAcreFoot.
CubicMetersPerAcreInch (see page 2011)	This is constant StdConvs.CubicMetersPerAcreInch.
CubicMetersPerCord (see page 2012)	This is constant StdConvs.CubicMetersPerCord.
CubicMetersPerCordFoot (see page 2012)	This is constant StdConvs.CubicMetersPerCordFoot.
CubicMetersPerCubicFoot (see page 2012)	This is constant StdConvs.CubicMetersPerCubicFoot.
CubicMetersPerCubicInch (see page 2013)	This is constant StdConvs.CubicMetersPerCubicInch.
CubicMetersPerCubicMile (see page 2013)	This is constant StdConvs.CubicMetersPerCubicMile.
CubicMetersPerCubicYard (see page 2013)	This is constant StdConvs.CubicMetersPerCubicYard.
CubicMetersPerUKBucket (see page 2014)	This is constant StdConvs.CubicMetersPerUKBucket.
CubicMetersPerUKBushel (see page 2014)	This is constant StdConvs.CubicMetersPerUKBushel.
CubicMetersPerUKGallon (see page 2014)	This is constant StdConvs.CubicMetersPerUKGallon.
CubicMetersPerUKGill (see page 2015)	This is constant StdConvs.CubicMetersPerUKGill.
CubicMetersPerUKOunce (see page 2015)	This is constant StdConvs.CubicMetersPerUKOunce.
CubicMetersPerUKPeck (see page 2015)	This is constant StdConvs.CubicMetersPerUKPeck.
CubicMetersPerUKPint (see page 2016)	This is constant StdConvs.CubicMetersPerUKPint.
CubicMetersPerUKPottle (see page 2016)	This is constant StdConvs.CubicMetersPerUKPottle.
CubicMetersPerUKQuart (see page 2016)	This is constant StdConvs.CubicMetersPerUKQuart.
CubicMetersPerUSDryBucket (see page 2017)	This is constant StdConvs.CubicMetersPerUSDryBucket.
CubicMetersPerUSDryBushel (see page 2017)	This is constant StdConvs.CubicMetersPerUSDryBushel.
CubicMetersPerUSDryGallon (see page 2017)	This is constant StdConvs.CubicMetersPerUSDryGallon.
CubicMetersPerUSDryPeck (see page 2018)	This is constant StdConvs.CubicMetersPerUSDryPeck.
CubicMetersPerUSDryPint (see page 2018)	This is constant StdConvs.CubicMetersPerUSDryPint.
CubicMetersPerUSDryQuart (see page 2018)	This is constant StdConvs.CubicMetersPerUSDryQuart.
CubicMetersPerUSFluidCup (see page 2019)	This is constant StdConvs.CubicMetersPerUSFluidCup.
CubicMetersPerUSFluidGallon (see page 2019)	This is constant StdConvs.CubicMetersPerUSFluidGallon.
CubicMetersPerUSFluidGill (see page 2019)	This is constant StdConvs.CubicMetersPerUSFluidGill.
CubicMetersPerUSFluidOunce (see page 2020)	This is constant StdConvs.CubicMetersPerUSFluidOunce.
CubicMetersPerUSFluidPint (see page 2020)	This is constant StdConvs.CubicMetersPerUSFluidPint.
CubicMetersPerUSFluidQuart (see page 2020)	This is constant StdConvs.CubicMetersPerUSFluidQuart.
CubicMetersPerUSFluidTablespoon (see page 2021)	This is constant StdConvs.CubicMetersPerUSFluidTablespoon.
CubicMetersPerUSFluidTeaspoon (see page 2021)	This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.
GramsPerDrams (see page 2021)	This is constant StdConvs.GramsPerDrams.
GramsPerGrains (see page 2022)	This is constant StdConvs.GramsPerGrains.
GramsPerLongTons (see page 2022)	This is constant StdConvs.GramsPerLongTons.
GramsPerOunces (see page 2022)	This is constant StdConvs.GramsPerOunces.
GramsPerPound (see page 2023)	This is constant StdConvs.GramsPerPound.
GramsPerStones (see page 2023)	This is constant StdConvs.GramsPerStones.
GramsPerTons (see page 2023)	This is constant StdConvs.GramsPerTons.
MetersPerAstronomicalUnit (see page 2024)	This is constant StdConvs.MetersPerAstronomicalUnit.
MetersPerChain (see page 2024)	This is constant StdConvs.MetersPerChain.
MetersPerCubit (see page 2024)	This is constant StdConvs.MetersPerCubit.
MetersPerFathom (see page 2025)	This is constant StdConvs.MetersPerFathom.
MetersPerFoot (see page 2025)	This is constant StdConvs.MetersPerFoot.
MetersPerFurlong (see page 2025)	This is constant StdConvs.MetersPerFurlong.

MetersPerHand (see page 2026)	This is constant StdConvs.MetersPerHand.
MetersPerInch (see page 2026)	This is constant StdConvs.MetersPerInch.
MetersPerLightSecond (see page 2026)	This is constant StdConvs.MetersPerLightSecond.
MetersPerLightYear (see page 2027)	This is constant StdConvs.MetersPerLightYear.
MetersPerLink (see page 2027)	This is constant StdConvs.MetersPerLink.
MetersPerMile (see page 2027)	This is constant StdConvs.MetersPerMile.
MetersPerNauticalMiles (see page 2028)	This is constant StdConvs.MetersPerNauticalMiles.
MetersPerPace (see page 2028)	This is constant StdConvs.MetersPerPace.
MetersPerParsec (see page 2028)	This is constant StdConvs.MetersPerParsec.
MetersPerPica (see page 2029)	This is constant StdConvs.MetersPerPica.
MetersPerPoint (see page 2029)	This is constant StdConvs.MetersPerPoint.
MetersPerRod (see page 2029)	This is constant StdConvs.MetersPerRod.
MetersPerYard (see page 2030)	This is constant StdConvs.MetersPerYard.
SquareMetersPerAcre (see page 2030)	This is constant StdConvs.SquareMetersPerAcre.
SquareMetersPerSquareFoot (see page 2030)	This is constant StdConvs.SquareMetersPerSquareFoot.
SquareMetersPerSquareInch (see page 2031)	This is constant StdConvs.SquareMetersPerSquareInch.
SquareMetersPerSquareMile (see page 2031)	This is constant StdConvs.SquareMetersPerSquareMile.
SquareMetersPerSquareRod (see page 2031)	This is constant StdConvs.SquareMetersPerSquareRod.
SquareMetersPerSquareYard (see page 2032)	This is constant StdConvs.SquareMetersPerSquareYard.

29.3.1 StdConvs.CubicMetersPerAcreFoot Constant

Pascal

```
CubicMetersPerAcreFoot = SquareMetersPerAcre * MetersPerFoot;
```

C++

```
CubicMetersPerAcreFoot = SquareMetersPerAcre * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerAcreFoot.

29.3.2 StdConvs.CubicMetersPerAcreInch Constant

Pascal

```
CubicMetersPerAcreInch = SquareMetersPerAcre * MetersPerInch;
```

C++

```
CubicMetersPerAcreInch = SquareMetersPerAcre * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerAcreInch.

29.3.3 StdConvs.CubicMetersPerCord Constant

Pascal

```
CubicMetersPerCord = CubicMetersPerCubicFoot * 128;
```

C++

```
CubicMetersPerCord = CubicMetersPerCubicFoot * 128;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCord.

29.3.4 StdConvs.CubicMetersPerCordFoot Constant

Pascal

```
CubicMetersPerCordFoot = CubicMetersPerCubicFoot * 16;
```

C++

```
CubicMetersPerCordFoot = CubicMetersPerCubicFoot * 16;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCordFoot.

29.3.5 StdConvs.CubicMetersPerCubicFoot Constant

Pascal

```
CubicMetersPerCubicFoot = MetersPerFoot * MetersPerFoot * MetersPerFoot;
```

C++

```
CubicMetersPerCubicFoot = MetersPerFoot * MetersPerFoot * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicFoot.

29.3.6 StdConvs.CubicMetersPerCubicInch Constant

Pascal

```
CubicMetersPerCubicInch = MetersPerInch * MetersPerInch * MetersPerInch;
```

C++

```
CubicMetersPerCubicInch = MetersPerInch * MetersPerInch * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicInch.

29.3.7 StdConvs.CubicMetersPerCubicMile Constant

Pascal

```
CubicMetersPerCubicMile = MetersPerMile * MetersPerMile * MetersPerMile;
```

C++

```
CubicMetersPerCubicMile = MetersPerMile * MetersPerMile * MetersPerMile;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicMile.

29.3.8 StdConvs.CubicMetersPerCubicYard Constant

Pascal

```
CubicMetersPerCubicYard = MetersPerYard * MetersPerYard * MetersPerYard;
```

C++

```
CubicMetersPerCubicYard = MetersPerYard * MetersPerYard * MetersPerYard;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerCubicYard.

29.3.9 StdConvs.CubicMetersPerUKBucket Constant

Pascal

```
CubicMetersPerUKBucket = CubicMetersPerUKPeck * 2;
```

C++

```
CubicMetersPerUKBucket = CubicMetersPerUKPeck * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKBucket.

29.3.10 StdConvs.CubicMetersPerUKBushel Constant

Pascal

```
CubicMetersPerUKBushel = CubicMetersPerUKBucket * 2;
```

C++

```
CubicMetersPerUKBushel = CubicMetersPerUKBucket * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKBushel.

29.3.11 StdConvs.CubicMetersPerUKGallon Constant

Pascal

```
CubicMetersPerUKGallon = 0.00454609;
```

C++

```
CubicMetersPerUKGallon = 0.00454609;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKGallon.

29.3.12 StdConvs.CubicMetersPerUKGill Constant

Pascal

```
CubicMetersPerUKGill = CubicMetersPerUKPint / 4;
```

C++

```
CubicMetersPerUKGill = CubicMetersPerUKPint / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKGill.

29.3.13 StdConvs.CubicMetersPerUKOunce Constant

Pascal

```
CubicMetersPerUKOunce = CubicMetersPerUKPint / 20;
```

C++

```
CubicMetersPerUKOunce = CubicMetersPerUKPint / 20;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKOunce.

29.3.14 StdConvs.CubicMetersPerUKPeck Constant

Pascal

```
CubicMetersPerUKPeck = CubicMetersPerUKGallon * 2;
```

C++

```
CubicMetersPerUKPeck = CubicMetersPerUKGallon * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPeck.

29.3.15 StdConvs.CubicMetersPerUKPint Constant

Pascal

```
CubicMetersPerUKPint = CubicMetersPerUKQuart / 2;
```

C++

```
CubicMetersPerUKPint = CubicMetersPerUKQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPint.

29.3.16 StdConvs.CubicMetersPerUKPottle Constant

Pascal

```
CubicMetersPerUKPottle = CubicMetersPerUKGallon / 2;
```

C++

```
CubicMetersPerUKPottle = CubicMetersPerUKGallon / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKPottle.

29.3.17 StdConvs.CubicMetersPerUKQuart Constant

Pascal

```
CubicMetersPerUKQuart = CubicMetersPerUKPottle / 2;
```

C++

```
CubicMetersPerUKQuart = CubicMetersPerUKPottle / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUKQuart.

29.3.18 StdConvs.CubicMetersPerUSDryBucket Constant

Pascal

```
CubicMetersPerUSDryBucket = CubicMetersPerUSDryPeck * 2;
```

C++

```
CubicMetersPerUSDryBucket = CubicMetersPerUSDryPeck * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryBucket.

29.3.19 StdConvs.CubicMetersPerUSDryBushel Constant

Pascal

```
CubicMetersPerUSDryBushel = CubicMetersPerUSDryBucket * 2;
```

C++

```
CubicMetersPerUSDryBushel = CubicMetersPerUSDryBucket * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryBushel.

29.3.20 StdConvs.CubicMetersPerUSDryGallon Constant

Pascal

```
CubicMetersPerUSDryGallon = CubicMetersPerCubicInch * 268.8025;
```

C++

```
CubicMetersPerUSDryGallon = CubicMetersPerCubicInch * 268.8025;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryGallon.

29.3.21 StdConvs.CubicMetersPerUSDryPeck Constant

Pascal

```
CubicMetersPerUSDryPeck = CubicMetersPerUSDryGallon * 2;
```

C++

```
CubicMetersPerUSDryPeck = CubicMetersPerUSDryGallon * 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryPeck.

29.3.22 StdConvs.CubicMetersPerUSDryPint Constant

Pascal

```
CubicMetersPerUSDryPint = CubicMetersPerUSDryQuart / 2;
```

C++

```
CubicMetersPerUSDryPint = CubicMetersPerUSDryQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryPint.

29.3.23 StdConvs.CubicMetersPerUSDryQuart Constant

Pascal

```
CubicMetersPerUSDryQuart = CubicMetersPerUSDryGallon / 4;
```

C++

```
CubicMetersPerUSDryQuart = CubicMetersPerUSDryGallon / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSDryQuart.

29.3.24 StdConvs.CubicMetersPerUSFluidCup Constant

Pascal

```
CubicMetersPerUSFluidCup = CubicMetersPerUSFluidPint / 2;
```

C++

```
CubicMetersPerUSFluidCup = CubicMetersPerUSFluidPint / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidCup.

29.3.25 StdConvs.CubicMetersPerUSFluidGallon Constant

Pascal

```
CubicMetersPerUSFluidGallon = CubicMetersPerCubicInch * 231;
```

C++

```
CubicMetersPerUSFluidGallon = CubicMetersPerCubicInch * 231;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidGallon.

29.3.26 StdConvs.CubicMetersPerUSFluidGill Constant

Pascal

```
CubicMetersPerUSFluidGill = CubicMetersPerUSFluidCup / 2;
```

C++

```
CubicMetersPerUSFluidGill = CubicMetersPerUSFluidCup / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidGill.

29.3.27 StdConvs.CubicMetersPerUSFluidOunce Constant

Pascal

```
CubicMetersPerUSFluidOunce = CubicMetersPerUSFluidCup / 8;
```

C++

```
CubicMetersPerUSFluidOunce = CubicMetersPerUSFluidCup / 8;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidOunce.

29.3.28 StdConvs.CubicMetersPerUSFluidPint Constant

Pascal

```
CubicMetersPerUSFluidPint = CubicMetersPerUSFluidQuart / 2;
```

C++

```
CubicMetersPerUSFluidPint = CubicMetersPerUSFluidQuart / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidPint.

29.3.29 StdConvs.CubicMetersPerUSFluidQuart Constant

Pascal

```
CubicMetersPerUSFluidQuart = CubicMetersPerUSFluidGallon / 4;
```

C++

```
CubicMetersPerUSFluidQuart = CubicMetersPerUSFluidGallon / 4;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidQuart.

29.3.30 StdConvs.CubicMetersPerUSFluidTablespoon Constant

Pascal

```
CubicMetersPerUSFluidTablespoon = CubicMetersPerUSFluidOunce / 2;
```

C++

```
CubicMetersPerUSFluidTablespoon = CubicMetersPerUSFluidOunce / 2;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidTablespoon.

29.3.31 StdConvs.CubicMetersPerUSFluidTeaspoon Constant

Pascal

```
CubicMetersPerUSFluidTeaspoon = CubicMetersPerUSFluidOunce / 6;
```

C++

```
CubicMetersPerUSFluidTeaspoon = CubicMetersPerUSFluidOunce / 6;
```

File

StdConvs

Description

This is constant StdConvs.CubicMetersPerUSFluidTeaspoon.

29.3.32 StdConvs.GramsPerDrams Constant

Pascal

```
GramsPerDrams = GramsPerPound / 256;
```

C++

```
GramsPerDrams = GramsPerPound / 256;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerDrams.

29.3.33 StdConvs.GramsPerGrains Constant

Pascal

```
GramsPerGrains = GramsPerPound / 7000;
```

C++

```
GramsPerGrains = GramsPerPound / 7000;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerGrains.

29.3.34 StdConvs.GramsPerLongTons Constant

Pascal

```
GramsPerLongTons = GramsPerPound * 2240;
```

C++

```
GramsPerLongTons = GramsPerPound * 2240;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerLongTons.

29.3.35 StdConvs.GramsPerOunces Constant

Pascal

```
GramsPerOunces = GramsPerPound / 16;
```

C++

```
GramsPerOunces = GramsPerPound / 16;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerOunces.

29.3.36 StdConvs.GramsPerPound Constant

Pascal

```
GramsPerPound = 453.59237;
```

C++

```
GramsPerPound = 453.59237;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerPound.

29.3.37 StdConvs.GramsPerStones Constant

Pascal

```
GramsPerStones = GramsPerPound * 14;
```

C++

```
GramsPerStones = GramsPerPound * 14;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerStones.

29.3.38 StdConvs.GramsPerTons Constant

Pascal

```
GramsPerTons = GramsPerPound * 2000;
```

C++

```
GramsPerTons = GramsPerPound * 2000;
```

File

StdConvs

Description

This is constant StdConvs.GramsPerTons.

29.3.39 StdConvs.MetersPerAstronomicalUnit Constant

Pascal

```
MetersPerAstronomicalUnit = 1.49598E11;
```

C++

```
MetersPerAstronomicalUnit = 1.49598E11;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerAstronomicalUnit.

29.3.40 StdConvs.MetersPerChain Constant

Pascal

```
MetersPerChain = MetersPerRod * 4;
```

C++

```
MetersPerChain = MetersPerRod * 4;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerChain.

29.3.41 StdConvs.MetersPerCubit Constant

Pascal

```
MetersPerCubit = 0.4572;
```

C++

```
MetersPerCubit = 0.4572;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerCubit.

29.3.42 StdConvs.MetersPerFathom Constant

Pascal

```
MetersPerFathom = MetersPerFoot * 6;
```

C++

```
MetersPerFathom = MetersPerFoot * 6;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFathom.

29.3.43 StdConvs.MetersPerFoot Constant

Pascal

```
MetersPerFoot = MetersPerInch * 12;
```

C++

```
MetersPerFoot = MetersPerInch * 12;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFoot.

29.3.44 StdConvs.MetersPerFurlong Constant

Pascal

```
MetersPerFurlong = MetersPerYard * 220;
```

C++

```
MetersPerFurlong = MetersPerYard * 220;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerFurlong.

29.3.45 StdConvs.MetersPerHand Constant

Pascal

```
MetersPerHand = MetersPerInch * 4;
```

C++

```
MetersPerHand = MetersPerInch * 4;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerHand.

29.3.46 StdConvs.MetersPerInch Constant

Pascal

```
MetersPerInch = 0.0254;
```

C++

```
MetersPerInch = 0.0254;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerInch.

29.3.47 StdConvs.MetersPerLightSecond Constant

Pascal

```
MetersPerLightSecond = 2.99792458E8;
```

C++

```
MetersPerLightSecond = 2.99792458E8;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLightSecond.

29.3.48 StdConvs.MetersPerLightYear Constant

Pascal

```
MetersPerLightYear = MetersPerLightSecond * 31556925.9747;
```

C++

```
MetersPerLightYear = MetersPerLightSecond * 31556925.9747;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLightYear.

29.3.49 StdConvs.MetersPerLink Constant

Pascal

```
MetersPerLink = MetersPerChain / 100;
```

C++

```
MetersPerLink = MetersPerChain / 100;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerLink.

29.3.50 StdConvs.MetersPerMile Constant

Pascal

```
MetersPerMile = MetersPerFoot * 5280;
```

C++

```
MetersPerMile = MetersPerFoot * 5280;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerMile.

29.3.51 StdConvs.MetersPerNauticalMiles Constant

Pascal

```
MetersPerNauticalMiles = 1852;
```

C++

```
MetersPerNauticalMiles = 1852;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerNauticalMiles.

29.3.52 StdConvs.MetersPerPace Constant

Pascal

```
MetersPerPace = MetersPerInch * 30;
```

C++

```
MetersPerPace = MetersPerInch * 30;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPace.

29.3.53 StdConvs.MetersPerParsec Constant

Pascal

```
MetersPerParsec = MetersPerAstronomicalUnit * 206264.806247096;
```

C++

```
MetersPerParsec = MetersPerAstronomicalUnit * 206264.806247096;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerParsec.

29.3.54 StdConvs.MetersPerPica Constant

Pascal

```
MetersPerPica = MetersPerPoint * 12;
```

C++

```
MetersPerPica = MetersPerPoint * 12;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPica.

29.3.55 StdConvs.MetersPerPoint Constant

Pascal

```
MetersPerPoint = MetersPerInch * 0.013837;
```

C++

```
MetersPerPoint = MetersPerInch * 0.013837;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerPoint.

29.3.56 StdConvs.MetersPerRod Constant

Pascal

```
MetersPerRod = MetersPerFoot * 16.5;
```

C++

```
MetersPerRod = MetersPerFoot * 16.5;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerRod.

29.3.57 StdConvs.MetersPerYard Constant

Pascal

```
MetersPerYard = MetersPerFoot * 3;
```

C++

```
MetersPerYard = MetersPerFoot * 3;
```

File

StdConvs

Description

This is constant StdConvs.MetersPerYard.

29.3.58 StdConvs.SquareMetersPerAcre Constant

Pascal

```
SquareMetersPerAcre = SquareMetersPerSquareYard * 4840;
```

C++

```
SquareMetersPerAcre = SquareMetersPerSquareYard * 4840;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerAcre.

29.3.59 StdConvs.SquareMetersPerSquareFoot Constant

Pascal

```
SquareMetersPerSquareFoot = MetersPerFoot * MetersPerFoot;
```

C++

```
SquareMetersPerSquareFoot = MetersPerFoot * MetersPerFoot;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareFoot.

29.3.60 StdConvs.SquareMetersPerSquareInch Constant

Pascal

```
SquareMetersPerSquareInch = MetersPerInch * MetersPerInch;
```

C++

```
SquareMetersPerSquareInch = MetersPerInch * MetersPerInch;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareInch.

29.3.61 StdConvs.SquareMetersPerSquareMile Constant

Pascal

```
SquareMetersPerSquareMile = MetersPerMile * MetersPerMile;
```

C++

```
SquareMetersPerSquareMile = MetersPerMile * MetersPerMile;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareMile.

29.3.62 StdConvs.SquareMetersPerSquareRod Constant

Pascal

```
SquareMetersPerSquareRod = MetersPerRod * MetersPerRod;
```

C++

```
SquareMetersPerSquareRod = MetersPerRod * MetersPerRod;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareRod.

29.3.63 StdConvs.SquareMetersPerSquareYard Constant

Pascal

```
SquareMetersPerSquareYard = MetersPerYard * MetersPerYard;
```

C++

```
SquareMetersPerSquareYard = MetersPerYard * MetersPerYard;
```

File

StdConvs

Description

This is constant StdConvs.SquareMetersPerSquareYard.

30 StdCtrls Namespace

This is namespace StdCtrls.

Topics

Name	Description
Classes (see page 2035)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 2999)	The following table lists structs, records, enums in this documentation.
Types (see page 3008)	The following table lists types in this documentation.

Classes

	Name	Description
	TButton (see page 2035)	TButton is a push button control.
	TButtonActionLink (see page 2106)	TButtonActionLink is an action link designed specifically for button control clients.
	TButtonControl (see page 2109)	TButtonControl is the base class for objects that represent Windows button controls.
	TCheckBox (see page 2116)	TCheckBox represents a check box that can be on (checked) or off (unchecked)
	TComboBox (see page 2184)	TComboBox combines an edit box with a scrollable list.
	TCustomButton (see page 2259)	Base class for a push button control.
	TCustomCheckBox (see page 2276)	TCustomCheckBox is the ancestor of all check-box components.
	TCustomCombo (see page 2283)	TCustomCombo is the base class for all combo boxes.
	TCustomComboBox (see page 2322)	TCustomComboBox is the base class for combobox components.
	TCustomComboBoxStrings (see page 2339)	TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.
	TCustomEdit (see page 2345)	TCustomEdit is the base class from which all edit-box and memo controls are derived.
	TCustomGroupBox (see page 2381)	TCustomGroupBox is the base class for all group-box components.
	TCustomLabel (see page 2384)	TCustomLabel is the base class for non-windowed controls that display text on a form.
	TCustomListBox (see page 2398)	TCustomListBox is the base class for listbox objects that are wrappers for list box controls.
	TCustomMemo (see page 2443)	TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 2737) and TDBMemo.
	TCustomStaticText (see page 2457)	TCustomStaticText is the base class for TStaticText (see page 2945).
	TEdit (see page 2463)	TEdit is a wrapper for a Windows single-line edit control.
	TGroupBox (see page 2535)	TGroupBox represents a Windows group box.
	TImageMargins (see page 2602)	Margins of image on button.
	TLabel (see page 2607)	TLabel is a nonwindowed control that displays text on a form.
	TListBox (see page 2657)	TListBox displays a collection of items in a scrollable list.

	TMemo (see page 2737)	TMemo is a wrapper for a Windows multiline edit control.
	TPushButtonActionLink (see page 2811)	TPushButtonActionLink is an action link designed specifically for button control clients.
	TRadioButton (see page 2813)	TRadioButton is a wrapper for a system radio button.
	TScrollBar (see page 2881)	TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.
	TStaticText (see page 2945)	TStaticText is a windowed control that displays text on a form.

Structs, Records, Enums

	Name	Description
	TCheckBoxState (see page 2999)	TCheckBoxState indicates the state of a check box control.
	TComboBoxStyle (see page 3000)	TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls unit.
	TEditCharCase (see page 3000)	TEditCharCase indicates how a control forces the case of characters, if at all.
	TEllipsisPosition (see page 3001)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	TImageAlignment (see page 3002)	Alignment of image on TButton (see page 2035).
	TListBoxStyle (see page 3003)	TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.
	TScrollCode (see page 3004)	TScrollCode indicates what action the user performed on a scroll bar.
	TScrollStyle (see page 3005)	Identifies a set of scrollbars by their orientation.
	TSelection (see page 3006)	TSelection definese the start and position of a selection.
	TStaticBorderStyle (see page 3006)	TStaticBorderStyle determines the appearance of the border on a static text control.
	TTextLayout (see page 3007)	TTextLayout indicates where text appears within the client area of a control.

Types

Name	Description
TButtonActionLinkClass (see page 3008)	TButtonActionLinkClass defines the metaclass for TButtonActionLink (see page 2106).
TCustomComboBoxStringsClass (see page 3008)	TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 2339).
TDrawItemEvent (see page 3009)	TDrawItemEvent is the type of an OnDrawItem event handler.
TFNDrawText (see page 3010)	TFNDrawText represents the procedure type for the events that specify the details of the Text property of a control.
TLBFindDataEvent (see page 3010)	TLBFindDataEvent is the type of the list box OnDataFind event handler.
TLBGetDataEvent (see page 3011)	TLBGetDataEvent is the type of the list box OnData event handler.
TLBGetDataObjectEvent (see page 3011)	TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.
TMeasureItemEvent (see page 3012)	TMeasureItemEvent is the type for the OnMeasureItem event handler.
TOwnerDrawState (see page 3012)	Indicates state information that can influence how an item is drawn.
TPushButtonActionLinkClass (see page 3013)	TPushButtonActionLinkClass defines the metaclass for TPushButtonActionLink (see page 2811).
TScrollEvent (see page 3013)	TScrollEvent is the type of an OnScroll event handler.

30.1 Classes

The following table lists classes in this documentation.

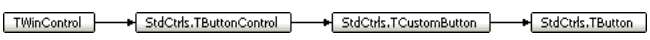
Classes

	Name	Description
	TButton (see page 2035)	TButton is a push button control.
	TButtonActionLink (see page 2106)	TButtonActionLink is an action link designed specifically for button control clients.
	TButtonControl (see page 2109)	TButtonControl is the base class for objects that represent Windows button controls.
	TCheckBox (see page 2116)	TCheckBox represents a check box that can be on (checked) or off (unchecked)
	TComboBox (see page 2184)	TComboBox combines an edit box with a scrollable list.
	TCustomButton (see page 2259)	Base class for a push button control.
	TCustomCheckBox (see page 2276)	TCustomCheckBox is the ancestor of all check-box components.
	TCustomCombo (see page 2283)	TCustomCombo is the base class for all combo boxes.
	TCustomComboBox (see page 2322)	TCustomComboBox is the base class for combobox components.
	TCustomComboBoxStrings (see page 2339)	TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.
	TCustomEdit (see page 2345)	TCustomEdit is the base class from which all edit-box and memo controls are derived.
	TCustomGroupBox (see page 2381)	TCustomGroupBox is the base class for all group-box components.
	TCustomLabel (see page 2384)	TCustomLabel is the base class for non-windowed controls that display text on a form.
	TCustomListBox (see page 2398)	TCustomListBox is the base class for listbox objects that are wrappers for list box controls.
	TCustomMemo (see page 2443)	TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 2737) and TDBMemo.
	TCustomStaticText (see page 2457)	TCustomStaticText is the base class for TStaticText (see page 2945).
	TEdit (see page 2463)	TEdit is a wrapper for a Windows single-line edit control.
	TGroupBox (see page 2535)	TGroupBox represents a Windows group box.
	TImageMargins (see page 2602)	Margins of image on button.
	TLabel (see page 2607)	TLabel is a nonwindowed control that displays text on a form.
	TListBox (see page 2657)	TListBox displays a collection of items in a scrollable list.
	TMemo (see page 2737)	TMemo is a wrapper for a Windows multiline edit control.
	TPushButtonActionLink (see page 2811)	TPushButtonActionLink is an action link designed specifically for button control clients.
	TRadioButton (see page 2813)	TRadioButton is a wrapper for a system radio button.
	TScrollBar (see page 2881)	TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.
	TStaticText (see page 2945)	TStaticText is a windowed control that displays text on a form.

30.1.1 TButton Class

TButton is a push button control.

Class Hierarchy



Pascal

```
TButton = class(TCustomButton);
```

C++

```
class TButton : public TCustomButton;
```

File

StdCtrls

Description

Use TButton to put a standard push button on a form. TButton introduces several properties to control its behavior in a dialog box setting. Users choose button controls to initiate actions.

To use a button that displays a bitmap instead of a label, use TBitBtn. To use a button that can remain in a depressed position, use TSpeedButton.

Note: Since the TButton caption is always centered, changing the BiDi alignment has no effect.

See Also

TBitBtn

TCustomButton (see page 2259)

TSpeedButton

Button Controls

Buttons and Similar Controls

30.1.1.1 TButton Members

The following tables list the members exposed by TButton.

Enumerations

	Name	Description
	TButtonStyle (see page 2261)	Style (see page 2275) of TButton (see page 2035).

Protected Methods

	Name	Description
	ActionChange (see page 2110)	Updates the button control to reflect changes in its associated action.
	ActionChange (see page 2262)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2111)	Initializes a window-creation parameter data structure.
	CreateParams (see page 2263)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2264)	Creates the underlying window.
	GetActionLinkClass (see page 2111)	Returns the appropriate action link class that is associated with button controls.
	GetActionLinkClass (see page 2264)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2112)	Returns whether the button control is checked.
	Loaded (see page 2264)	Initializes the control after it is loaded from a stream.
	Notification (see page 2265)	Responds to notifications that components are being created or destroyed.
	SetButtonStyle (see page 2265)	Sets the Style (see page 2275) property for a button.
	SetChecked (see page 2112)	Sets the Checked (see page 2113) property.
	UpdateImageList (see page 2266)	Rebuilds the image list associated with the button.
	UpdateImages (see page 2266)	Updates the image list associated with the button.
	WndProc (see page 2112)	Handles Windows messages.

Protected Properties

	Name	Description
	Checked (see page 2113)	Specifies whether the button control is checked.

	ClicksDisabled (see page 2115)	Specifies whether the button control accepts mouse clicks.

Public Methods

	Name	Description
	Click (see page 2263)	Simulates a mouse click, as if the user had clicked the button.
	Create (see page 2110)	Creates a button control.
	Create (see page 2260)	Creates and initializes a button instance.
	Destroy (see page 2261)	Destroys an instance of TWinControl.
	UseRightToLeftAlignment (see page 2266)	Specifies whether the control's alignment is in a right-to-left mode.

Published Events

	Name	Description
	OnClick (see page 2073)	Occurs when the user clicks the control.
	OnContextPopup (see page 2075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2076)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2079)	Occurs when the user drags an object over a control.
	OnDropDownClick (see page 2082)	Down arrow part of Vista style split button is clicked.
	OnEndDock (see page 2082)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2083)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2084)	Occurs when a control receives the input focus.
	OnExit (see page 2086)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2090)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2094)	Occurs when a key is pressed.
	OnKeyUp (see page 2095)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2096)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2097)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2100)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2100)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2100)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2103)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2105)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2105)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 2039)	Designates the action associated with the control.
	Align (see page 2040)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2044)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2044)	Specifies the bi-directional mode for the control.
	Cancel (see page 2045)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	Caption (see page 2045)	Specifies a text string that identifies the control to the user.
	CommandLinkHint (see page 2048)	Text displayed as hint below button caption for Command Link.
	Constraints (see page 2048)	Specifies the size constraints for the control.
	Default (see page 2049)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DisabledImageIndex (see page 2050)	Image index for disabled button state.
	DoubleBuffered (see page 2050)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

	DragCursor (see page 2052)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2052)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2053)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownMenu (see page 2054)	Split button drop down menu.
	ElevationRequired (see page 2055)	Put shield icon on button indicating elevated access rights required.
	Enabled (see page 2055)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2056)	Controls the attributes of text written on or in the control.
	HotImageIndex (see page 2057)	Image index for hot button state.
	ImageAlignment (see page 2057)	Alignment of image on button.
	ImageIndex (see page 2058)	Image index for normal button state.
	ImageMargins (see page 2058)	Margins of image on button.
	Images (see page 2059)	List of images for a button.
	ModalResult (see page 2059)	Determines whether and how the button closes its (modal) parent form.
	ParentBiDiMode (see page 2060)	Specifies whether the control uses its parent's BiDiMode.
	ParentDoubleBuffered (see page 2060)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2061)	Determines where a control looks for its font information.
	ParentShowHint (see page 2061)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2062)	Identifies the pop-up menu associated with the control.
	PressedImageIndex (see page 2067)	Image index for pressed button state.
	SelectedImageIndex (see page 2067)	Image index for selected button state.
	ShowHint (see page 2068)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 2069)	Style of a button.
	TabOrder (see page 2069)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2070)	Determines if the user can tab to a control.
	Visible (see page 2070)	Determines whether the component appears onscreen.
	WordWrap (see page 2072)	Specifies whether the button text wraps to fit the width of the control.

30.1.1.2 TButton Properties

The properties of the TButton class are listed here.

Published Properties

	Name	Description
	Action (see page 2039)	Designates the action associated with the control.
	Align (see page 2040)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2044)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2044)	Specifies the bi-directional mode for the control.
	Cancel (see page 2045)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	Caption (see page 2045)	Specifies a text string that identifies the control to the user.
	CommandLinkHint (see page 2048)	Text displayed as hint below button caption for Command Link.
	Constraints (see page 2048)	Specifies the size constraints for the control.
	Default (see page 2049)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DisabledImageIndex (see page 2050)	Image index for disabled button state.
	DoubleBuffered (see page 2050)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2052)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2052)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2053)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownMenu (see page 2054)	Split button drop down menu.

	ElevationRequired (see page 2055)	Put shield icon on button indicating elevated access rights required.
	Enabled (see page 2055)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2056)	Controls the attributes of text written on or in the control.
	HotImageIndex (see page 2057)	Image index for hot button state.
	ImageAlignment (see page 2057)	Alignment of image on button.
	ImageIndex (see page 2058)	Image index for normal button state.
	ImageMargins (see page 2058)	Margins of image on button.
	Images (see page 2059)	List of images for a button.
	ModalResult (see page 2059)	Determines whether and how the button closes its (modal) parent form.
	ParentBiDiMode (see page 2060)	Specifies whether the control uses its parent's BiDiMode.
	ParentDoubleBuffered (see page 2060)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2061)	Determines where a control looks for its font information.
	ParentShowHint (see page 2061)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2062)	Identifies the pop-up menu associated with the control.
	PressedImageIndex (see page 2067)	Image index for pressed button state.
	SelectedImageIndex (see page 2067)	Image index for selected button state.
	ShowHint (see page 2068)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Style (see page 2069)	Style of a button.
	TabOrder (see page 2069)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2070)	Determines if the user can tab to a control.
	Visible (see page 2070)	Determines whether the component appears onscreen.
	WordWrap (see page 2072)	Specifies whether the button text wraps to fit the width of the control.

30.1.1.2.1 TButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

C++ Examples:

```
/*
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
*/
void __fastcall TForm1::ChangeColor(TObject *Sender)
{
  // randomly change the color of the form
  Color = Random(0xFFFF);
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // allocate memory for the Action property
  Action = new TBasicAction(this);

  // set the OnExecute event handler
  Action->OnExecute = ChangeColor;

  // initialize the random number generator
  Randomize();
}
```

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
  Action.OnExecute := ChangeColor;

  { initialize the random number generator }
  Randomize;
end;
```

30.1.1.2.2 TButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the

edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
```

```
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
```

```
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

30.1.1.2.3 TButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.1.2.4 TButton.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.1.2.5 TButton.Cancel Property

Determines whether the button's OnClick event handler executes when the Escape key is pressed.

Pascal

```
property Cancel: Boolean;
```

C++

```
__property Boolean Cancel;
```

Description

If Cancel is true, the button's OnClick event handler executes when the user presses Esc. Although an application can have more than one Cancel button, the form calls the OnClick event handler only for the first visible button in the tab order.

See Also

Default (see page 2268)

OnClick

30.1.1.2.6 TButton.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
```

```
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

30.1.1.2.7 TButton.CommandLinkHint Property

Text displayed as hint below button caption for Command Link.

Pascal

```
property CommandLinkHint: string;
```

C++

```
__property UnicodeString CommandLinkHint;
```

Description

CommandLinkHint is the text displayed as a hint below the button caption when the Style property is **bsCommandLink** to specify a Command Link button.

This is available on Windows Vista only.

See Also

Style (see page 2275)

TButtonStyle (see page 2261)

30.1.1.2.8 TButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.1.2.9 TButton.Default Property

Determines whether the button's OnClick event handler executes when the Enter key is pressed.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

If Default is true, the button's OnClick event handler executes when the user presses Enter.

Although an application can have more than one Default button, the form calls the OnClick event handler only for the first visible button in the tab order. Moreover, any button that has focus becomes the Default button temporarily; hence, if the user selects another button before pressing Enter, the selected button's OnClick event handler executes instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
```

```
end;
```

See Also

Cancel (see page 2267)

OnClick

30.1.1.2.10 TButton.DisabledImageIndex Property

Image index for disabled button state.

Pascal

```
property DisabledImageIndex: TImageIndex;
```

C++

```
__property TImageIndex DisabledImageIndex;
```

Description

DisabledImageIndex is an image index into the Images property for the disabled button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

HotImageIndex (see page 2270)

ImageIndex (see page 2271)

Images (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

30.1.1.2.11 TButton.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
```

```
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.1.2.12 TButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.1.2.13 TButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.1.2.14 TButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
```

```
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.1.2.15 TButton.DropDownMenu Property

Split button drop down menu.

Pascal

```
property DropDownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropDownMenu;
```

Description

When the Style property is **bsSplitButton**, the button is a Vista style split button with a drop down menu. DropDownMenu specifies the drop down menu for the down arrow button part. The OnDropDownClick event also occurs when the down arrow button part is clicked.

This is available on Windows Vista only.

See Also

TPopupMenu

Style (see page 2275)

OnDropDownClick (see page 2276)

TButtonStyle (see page 2261)

30.1.1.2.16 TButton.ElevationRequired Property

Put shield icon on button indicating elevated access rights required.

Pascal

```
property ElevationRequired: Boolean;
```

C++

```
__property Boolean ElevationRequired;
```

Description

When ElevationRequired is true, the button has a shield icon, indicating that elevated access rights are required. Vista UI guidelines specify that when there is a button that performs an operation requiring elevated access rights, it should have this shield icon.

This icon overrides any of the properties specifying a button image for various button states, as in ImageIndex.

This is available on Windows Vista only.

See Also

ImageIndex (see page 2271)

Images (see page 2272)

30.1.1.2.17 TButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.1.2.18 TButton.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.1.2.19 TButton.HotImageIndex Property

Image index for hot button state.

Pascal

```
property HotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex HotImageIndex;
```

Description

HotImageIndex is an image index into the Images property for the hot button state, i.e., the cursor hovering over the button. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

ImageIndex (see page 2271)

Images (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

30.1.1.2.20 TButton.ImageAlignment Property

Alignment of image on button.

Pascal

```
property ImageAlignment: TImageAlignment;
```

C++

```
__property TImageAlignment ImageAlignment;
```

Description

TImageAlignment indicates the alignment of an image on a button. It is a TImageAlignment type.

See Also

ImageMargins (see page 2272)

Images (see page 2272)

TImageAlignment (see page 3002)

30.1.1.2.21 TButton.ImageIndex Property

Image index for normal button state.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an image index into the Images property for the normal button state. If the index value is -1, no image is used.

In Vista, a standard green arrow is used when the index is -1 and the Style property is **bsCommandLink** for a Command Link button.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

Images (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

Style (see page 2275)

TButtonStyle (see page 2261)

30.1.1.2.22 TButton.ImageMargins Property

Margins of image on button.

Pascal

```
property ImageMargins: TImageMargins;
```

C++

```
__property TImageMargins ImageMargins;
```

Description

A button image may be positioned using margins. ImageMargins holds aTImageMargins object containing the margins of an image on a button. The TImageMargins.OnChange event is triggered when a margin changes.

See Also

ImageAlignment (see page 2271)

TImageMargins (see page 2602)

Images (see page 2272)

OnChange (see page 2607)

30.1.1.2.23 TButton.Images Property

List of images for a button.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a TCustomImageList object listing the various images that can be placed on a button.

The button's image depends on the button state. These images are determined by the value of the image state properties that index ImageIndex: DisabledImageIndex, HotImageIndex, ImageIndex, PressedImageIndex, and SelectedImageIndex. When the ImageIndex index is -1, it generally means that no image is used for the normal state. For the other state image indexes, -1 means to use the same image as the normal state.

The ImageMargins and ImageAlignment properties orient the image inside the button. The image is not sized to fit the button, and the button is not stretched.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

ImageAlignment (see page 2271)

ImageIndex (see page 2271)

ImageMargins (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

30.1.1.2.24 TButton.ModalResult Property

Determines whether and how the button closes its (modal) parent form.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Setting the button component's ModalResult property is an easy way to make clicking the button close a modal form. When a button is clicked, the ModalResult property of its parent form is set to the same value as the button's ModalResult property.

For example, if a dialog box has OK and Cancel buttons, their ModalResult properties could be set at design time to mrOk and mrCancel, respectively. At runtime, clicking the OK button then changes the dialog's ModalResult property to mrOk, and clicking the Cancel button changes the dialog's ModalResult property to mrCancel. Unless further processing is required, no OnClick event handlers are required for the buttons.

The following constants are of TModalResult type and defined for the ModalResult property:

Constant	Value	Meaning
mrNone	0	None. Used as a default value before the user exits.
mrOk	idOK	The user exited with OK button.
mrCancel	idCancel	The user exited with the CANCEL button.
mrAbort	idAbort	The user exited with the ABORT button.
mrRetry	idRetry	The user exited with the RETRY button.
mrIgnore	idIgnore	The user exited with the IGNORE button.
mrYes	idYes	The user exited with the YES button.
mrNo	idNo	The user exited with the NO button.
mrAll	mrNo + 1	The user exited with the ALL button.
mrNoToAll	mrAll + 1	The user exited with the NO TO ALL button.
mrYesToAll	mrNoToAll + 1	The user exited with the YES TO ALL button.
mrClose	mrYesToAll + 1	The user exited with the CLOSE button.

See Also

TModalResult

30.1.1.2.25 TButton.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.1.2.26 TButton.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.1.2.27 TButton.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.1.2.28 TButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.1.2.29 TButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
```

```
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
```

```
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
```

```
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.1.2.30 TButton.PressedImageIndex Property

Image index for pressed button state.

Pascal

```
property PressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex PressedImageIndex;
```

Description

PressedImageIndex is an image index into the Images property for the pressed button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

ImageIndex (see page 2271)

Images (see page 2272)

SelectedImageIndex (see page 2274)

30.1.1.2.31 TButton.SelectedImageIndex Property

Image index for selected button state.

Pascal

```
property SelectedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex SelectedImageIndex;
```

Description

SelectedImageIndex is an image index into the Images property for the selected button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

ImageIndex (see page 2271)

Images (see page 2272)

PressedImageIndex (see page 2274)

30.1.1.2.32 TButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
```

```
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.1.2.33 TButton.Style Property

Style of a button.

Pascal

```
property Style: TButtonStyle;
```

C++

```
__property TButtonStyle Style;
```

Description

Style indicates a button's style, which is a TButtonStyle type.

See Also

TButtonStyle (see page 2261)

30.1.1.2.34 TButton.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.1.2.35 TButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.1.2.36 TButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.1.2.37 TButton.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

30.1.1.3 TButton Events

The events of the TButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 2073)	Occurs when the user clicks the control.
	OnContextPopup (see page 2075)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2076)	Occurs when the user drops an object being dragged.

	OnDragOver (see page 2079)	Occurs when the user drags an object over a control.
	OnDropDownClick (see page 2082)	Down arrow part of Vista style split button is clicked.
	OnEndDock (see page 2082)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2083)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2084)	Occurs when a control receives the input focus.
	OnExit (see page 2086)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2090)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2094)	Occurs when a key is pressed.
	OnKeyUp (see page 2095)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2096)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2097)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2100)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2100)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2100)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2103)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2105)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2105)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.1.3.1 TButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
```

```
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.1.3.2 TButton.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.1.3.3 TButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.1.3.4 TButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.1.3.5 TButton.OnDropDownClick Event

Down arrow part of Vista style split button is clicked.

Pascal

```
property OnDropDownClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDownClick;
```

Description

OnDropDownClick occurs when the down arrow part of a Vista style split button is clicked. This only applies when the button's Style property is **bsSplitButton**.

This is available on Windows Vista only.

See Also

DropDownMenu (see page 2269)

Style (see page 2275)

TButtonStyle (see page 2261)

30.1.1.3.6 TButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.1.3.7 TButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.1.3.8 TButton.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.1.3.9 TButton.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
```

```
        hostmask1 < 0 || hostmask1 > 255 ||
        hostmask2 < 0 || hostmask2 > 255)
      throw(Exception("Not a valid subnet mask."));

    // compute the subnet address using bit-wise AND
    int sub_net1 = net1 & netmask1;
    int sub_net2 = net2 & netmask2;
    int sub_host1 = host1 & hostmask1;
    int sub_host2 = host2 & hostmask2;

    // display the subnet address
    MaskEdit3->Text =
      String(sub_net1) + "." + String(sub_net2) + "." +
      String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';
```

```
  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
```

```
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

30.1.1.3.10 TButton.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
```

```
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
```

```
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.1.3.11 TButton.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
```

```
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.1.3.12 TButton.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
```

```
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

30.1.1.3.13 TButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.1.3.14 TButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
```

```
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
```

```
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.1.3.15 TButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.1.3.16 TButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.1.3.17 TButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
```

```
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.1.3.18 TButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.1.3.19 TButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.1.3.20 TButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

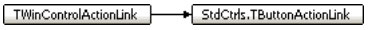
OnDragOver

OnDragDrop

30.1.2 TButtonActionLink Class

TButtonActionLink is an action link designed specifically for button control clients.

Class Hierarchy



Pascal

```
TButtonActionLink = class(TWinControlActionLink);
```

C++

```
class TButtonActionLink : public TWinControlActionLink;
```

File

StdCtrls

Description

TButtonActionLink is designed to specifically link actions to button control clients. TButtonActionLink overrides methods to determine whether the action and client are linked, and to ensure that action events are appropriate for properties of button control clients.

TButtonActionLink inherits, from its ancestors, support for linking actions with the properties of components, generic controls, and windowed controls. TButtonActionLink adds support for linking the checked property to the appropriate property of a button

control.

30.1.2.1 TButtonActionLink Members

The following tables list the members exposed by TButtonActionLink.

Protected Fields

	Name	Description
	FClient (see page 2107)	Specifies a client control.

Protected Methods

	Name	Description
	AssignClient (see page 2108)	Assigns a client button that is linked to the associated action.
	IsCheckedLinked (see page 2108)	Indicates whether the checked state for the action and the client are linked.
	SetChecked (see page 2108)	Sets the checked state of a button control if the client is linked to the action for this property.

30.1.2.2 TButtonActionLink Fields

The fields of the TButtonActionLink class are listed here.

Protected Fields

	Name	Description
	FClient (see page 2107)	Specifies a client control.

30.1.2.2.1 TButtonActionLink.FClient Field

Specifies a client control.

Pascal

```
FClient: TButtonControl;
```

C++

```
TButtonControl FClient;
```

Description

FClient is a protected field that specifies a client control.

See Also

AssignClient

30.1.2.3 TButtonActionLink Methods

The methods of the TButtonActionLink class are listed here.

Protected Methods

	Name	Description
	AssignClient (see page 2108)	Assigns a client button that is linked to the associated action.
	IsCheckedLinked (see page 2108)	Indicates whether the checked state for the action and the client are linked.
	SetChecked (see page 2108)	Sets the checked state of a button control if the client is linked to the action for this property.

30.1.2.3.1 TButtonActionLink.AssignClient Method

Assigns a client button that is linked to the associated action.

Pascal

```
procedure AssignClient(AClient: TObject); override;
```

C++

```
virtual __fastcall AssignClient(TObject * AClient);
```

Description

TButtonActionLink (see page 2106) overrides AssignClient to ensure that the client for the action is a button control.

See Also

Action

30.1.2.3.2 TButtonActionLink.IsCheckedLinked Method

Indicates whether the checked state for the action and the client are linked.

Pascal

```
function IsCheckedLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsCheckedLinked();
```

Description

IsCheckedLinked indicates whether the Checked property is linked to the appropriate client property. IsCheckedLinked returns true if the client requests that the action control this property, false otherwise.

See Also

Checked

Checked (see page 2113)

SetChecked (see page 2108)

30.1.2.3.3 TButtonActionLink.SetChecked Method

Sets the checked state of a button control if the client is linked to the action for this property.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked sets the checked state of a button control, and is called only if the client is linked for this property. Value specifies whether the checked state should be set.

See Also

Checked

Checked (see page 2113)

IsCheckedLinked (see page 2108)

30.1.3 TButtonControl Class

TButtonControl is the base class for objects that represent Windows button controls.

Class Hierarchy

Pascal

```
TButtonControl = class(TWinControl);
```

C++

```
class TButtonControl : public TWinControl;
```

File

StdCtrls

Description

TButtonControl encapsulates behavior common to button controls, check boxes, and radio buttons. Use TButtonControl as a base class when defining a button that is a wrapper for a Windows button. Do not create instances of TButtonControl directly in an application. Instead, use a descendant of TButtonControl to put a push button, check box, or radio button on a form.

Note: Some buttons, such as TSpeedButton, are not wrappers for Windows buttons. These controls do not descend from TButtonControl.

See Also

TButton (see page 2035)

TCustomCheckBox (see page 2276)

TRadioButton (see page 2813)

TSpeedButton

30.1.3.1 TButtonControl Members

The following tables list the members exposed by TButtonControl.

Protected Methods

	Name	Description
	ActionChange (see page 2110)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2111)	Initializes a window-creation parameter data structure.
	GetActionLinkClass (see page 2111)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2112)	Returns whether the button control is checked.
	SetChecked (see page 2112)	Sets the Checked (see page 2113) property.
	WndProc (see page 2112)	Handles Windows messages.

Protected Properties

	Name	Description
	Checked (see page 2113)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2115)	Specifies whether the button control accepts mouse clicks.

	WordWrap (see page 2115)	Specifies whether the button text wraps to fit the width of the control.

Public Methods

	Name	Description
	Create (see page 2110)	Creates a button control.

30.1.3.2 TButtonControl.Create Constructor

Creates a button control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TButtonControl(TComponent * AOwner);
```

Description

AOwner is the component that is responsible for freeing the button. It becomes the value of the Owner property.

30.1.3.3 TButtonControl Methods

The methods of the TButtonControl class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 2110)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2111)	Initializes a window-creation parameter data structure.
	GetActionLinkClass (see page 2111)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2112)	Returns whether the button control is checked.
	SetChecked (see page 2112)	Sets the Checked (see page 2113) property.
	WndProc (see page 2112)	Handles Windows messages.

30.1.3.3.1 TButtonControl.ActionChange Method

Updates the button control to reflect changes in its associated action.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the associated action changes. It updates the button control to reflect the action's current Checked (see page 2113) state. Then it calls the inherited method, which checks the associated properties for that client type.

The Sender parameter is the associated action. The CheckDefaults parameter specifies whether properties and event handlers should be changed if they differ from the default values. When CheckDefaults is true, properties and event handlers are only changed when the current values are the default values (that is, when the values have not been previously changed). When CheckDefaults is false, properties and event handlers are changed, regardless of their current values.

See Also

TCustomAction

30.1.3.3.2 TButtonControl.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

30.1.3.3.3 TButtonControl.GetActionLinkClass Method

Returns the appropriate action link class that is associated with button controls.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass is used to get the appropriate class type for the action link used with button controls. GetActionLinkClass is used internally to create action link objects for clients when a button control has an associated action. The action link is used to link the action to the button control client. Each Action link class type is designed to link specific actions to clients based upon the types of properties the client supports. For button controls, GetActionLinkClass returns the TButtonActionLink (see page 2106) class type.

See Also

TButtonActionLink (see page 2106)

30.1.3.3.4 TButtonControl.GetChecked Method

Returns whether the button control is checked.

Pascal

```
function GetChecked: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the read implementation of the Checked (see page 2113) property.

See Also

Checked (see page 2113)

30.1.3.3.5 TButtonControl.SetChecked Method

Sets the Checked (see page 2113) property.

Pascal

```
procedure SetChecked(Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked is the write implementer for the Checked (see page 2113) property.

See Also

Checked (see page 2113)

30.1.3.3.6 TButtonControl.WndProc Method

Handles Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc enables button objects to receive input focus when WM_LBUTTONDOWN and WM_LBUTTONDBLCLK messages occur.

See Also

WndProc

30.1.3.4 TButtonControl Properties

The properties of the TButtonControl class are listed here.

Protected Properties

	Name	Description
	Checked (see page 2113)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2115)	Specifies whether the button control accepts mouse clicks.
	WordWrap (see page 2115)	Specifies whether the button text wraps to fit the width of the control.

30.1.3.4.1 TButtonControl.Checked Property

Specifies whether the button control is checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to determine whether a button control is checked.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1); // PageControl1 is the parent, so it will
clean up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
```

```
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

WndProc

30.1.3.4.2 TButtonControl.ClicksDisabled Property

Specifies whether the button control accepts mouse clicks.

Pascal

```
property ClicksDisabled: Boolean;
```

C++

```
__property Boolean ClicksDisabled;
```

Description

Use ClicksDisabled to determine whether a button control accepts mouse clicks. If ClicksDisabled is true, the button control doesn't accept mouse clicks. Otherwise, the button control accepts mouse clicks.

See Also

WndProc

30.1.3.4.3 TButtonControl.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

30.1.4 TCheckBox Class

TCheckBox represents a check box that can be on (checked) or off (unchecked)

Class Hierarchy

Pascal

```
TCheckBox = class(TCustomCheckBox);
```

C++

```
class TCheckBox : public TCustomCheckBox;
```

File

StdCtrls

Description

A TCheckBox component presents an option for the user. The user can check the box to select the option, or uncheck it to deselect the option.

See Also

TCustomCheckBox (see page 2276)

Buttons and Similar Controls

Check Boxes

30.1.4.1 TCheckBox Members

The following tables list the members exposed by TCheckBox.

Protected Methods

	Name	Description
	ActionChange (see page 2110)	Updates the button control to reflect changes in its associated action.
	Click (see page 2278)	Click generates an OnClick event.
	CreateParams (see page 2111)	Initializes a window-creation parameter data structure.
	CreateParams (see page 2278)	Initializes a window-creation parameter record.
	CreateWnd (see page 2279)	Creates a Windows control corresponding to the check box.
	GetActionLinkClass (see page 2111)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2112)	Returns whether the button control is checked.
	GetChecked (see page 2279)	Returns the value of the Checked (see page 2113) property.
	SetChecked (see page 2112)	Sets the Checked (see page 2113) property.
	SetChecked (see page 2280)	Sets the value of the Checked (see page 2113) property.
	Toggle (see page 2280)	Changes the state of the check box.
	WndProc (see page 2112)	Handles Windows messages.

Protected Properties

	Name	Description
	ClicksDisabled (see page 2115)	Specifies whether the button control accepts mouse clicks.

Public Methods

	Name	Description
	Create (see page 2110)	Creates a button control.
	Create (see page 2277)	Creates and initializes a TCustomCheckBox (see page 2276) instance.
	GetControlsAlignment (see page 2279)	Returns the alignment of the check box caption.

Published Events

	Name	Description
	OnClick (see page 2151)	Occurs when the user clicks the control.
	OnContextPopup (see page 2153)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2154)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2157)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2160)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2160)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2161)	Occurs when a control receives the input focus.
	OnExit (see page 2164)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2168)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2171)	Occurs when a key is pressed.
	OnKeyUp (see page 2172)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2174)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2175)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2177)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2178)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2178)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2180)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2182)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2183)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 2119)	Designates the action associated with the control.
	Align (see page 2120)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2123)	Determines whether the check box label aligns to the left or to the right of the tick box.
	AllowGrayed (see page 2124)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 2124)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2125)	Specifies the bi-directional mode for the control.
	Caption (see page 2126)	Specifies a text string that identifies the control to the user.
	Checked (see page 2128)	Specifies whether the button control is checked.
	Color (see page 2130)	Specifies the background color of the control.
	Constraints (see page 2131)	Specifies the size constraints for the control.
	Ctl3D (see page 2132)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2132)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2134)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2134)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2135)	Determines how the control initiates drag-and-drop or drag-and-dock operations.

	Enabled (see page 2136)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2137)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2138)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2138)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2139)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2139)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2139)	Determines where a control looks for its font information.
	ParentShowHint (see page 2140)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2140)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2145)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	State (see page 2147)	Indicates whether the check box is selected, deselected, or grayed.
	TabOrder (see page 2148)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2148)	Determines if the user can tab to a control.
	Visible (see page 2149)	Determines whether the component appears onscreen.
	WordWrap (see page 2150)	Specifies whether the button text wraps to fit the width of the control.

30.1.4.2 TCheckBox Properties

The properties of the TCheckBox class are listed here.

Published Properties

	Name	Description
	Action (see page 2119)	Designates the action associated with the control.
	Align (see page 2120)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2123)	Determines whether the check box label aligns to the left or to the right of the tick box.
	AllowGrayed (see page 2124)	Determines whether check box can be in a "grayed" state.
	Anchors (see page 2124)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2125)	Specifies the bi-directional mode for the control.
	Caption (see page 2126)	Specifies a text string that identifies the control to the user.
	Checked (see page 2128)	Specifies whether the button control is checked.
	Color (see page 2130)	Specifies the background color of the control.
	Constraints (see page 2131)	Specifies the size constraints for the control.
	Ctl3D (see page 2132)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2132)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2134)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2134)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2135)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2136)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2137)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2138)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2138)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2139)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2139)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2139)	Determines where a control looks for its font information.
	ParentShowHint (see page 2140)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2140)	Identifies the pop-up menu associated with the control.

	ShowHint (see page 2145)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	State (see page 2147)	Indicates whether the check box is selected, deselected, or grayed.
	TabOrder (see page 2148)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2148)	Determines if the user can tab to a control.
	Visible (see page 2149)	Determines whether the component appears onscreen.
	WordWrap (see page 2150)	Specifies whether the button text wraps to fit the width of the control.

30.1.4.2.1 TCheckBox.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

C++ Examples:

```
/*
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
*/
void __fastcall TForm1::ChangeColor(TObject *Sender)
{
  // randomly change the color of the form
  Color = Random(0xFFFF);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // allocate memory for the Action property
  Action = new TBasicAction(this);

  // set the OnExecute event handler
  Action->OnExecute = ChangeColor;

  // initialize the random number generator
  Randomize();
}
```

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
```

```
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
  Action.OnExecute := ChangeColor;

  { initialize the random number generator }
  Randomize;
end;
```

30.1.4.2.2 TCheckBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
```

```
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
    }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
```

```
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

30.1.4.2.3 TCheckBox.Alignment Property

Determines whether the check box label aligns to the left or to the right of the tick box.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

The Alignment property determines the position of the label, relative to the tick box. The TLeftRight alignment type may have one of the following values.

Value	Meaning
taLeftJustify	The label is aligned to the left of the control and the tick box is aligned to the right of the control.
taRightJustify	This is the default value. The label is aligned to the right of the control and the tick box is aligned to the left of the control.

See Also

TLeftRight

30.1.4.2.4 TCheckBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to false, the check box has only two possible states: checked and unchecked.

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

Checked (see page 2113)

State (see page 2282)

30.1.4.2.5 TCheckBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.4.2.6 TCheckBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.4.2.7 TCheckBox.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>
```

```
void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
```

```
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

30.1.4.2.8 TCheckBox.Checked Property

Specifies whether the button control is checked.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Use Checked to determine whether a button control is checked.

C++ Examples:

```
/*
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
```

```
*/

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->MultiLine = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->Caption = "Stack Tabs?";
  for (int i = 0; i < 20; i++)
  {
    TTabSheet *pPage = new TTabSheet(PageControl1); // PageControl1 is the parent, so it will
clean up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
  }
}
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This example requires a PageControl with no pages created,
and a TCheckBox. During the form create, a series of tabs
will be created on the Page Control that exceed the Page
Control's width.  When the Check Box control is clicked, the
setting of the MultiLine property is toggled, alternately
stacking the tabs vertically, or along a single line
horizontally.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.MultiLine := CheckBox1.Checked;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
var
  i: Integer;
begin
  Checkbox1.Caption := 'Stack Tabs?';
  for i := 0 to 20 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'Tab #' + IntToStr(i);
    end;
end;
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

WndProc

30.1.4.2.9 TCheckBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.4.2.10 TCheckBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict,

the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.4.2.11 TCheckBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.4.2.12 TCheckBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to

false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.4.2.13 TCheckBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.4.2.14 TCheckBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.4.2.15 TCheckBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.4.2.16 TCheckBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled

controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.4.2.17 TCheckBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.4.2.18 TCheckBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.4.2.19 TCheckBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example,

if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.4.2.20 TCheckBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.4.2.21 TCheckBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.4.2.22 TCheckBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.4.2.23 TCheckBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.4.2.24 TCheckBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
```

```
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;
```

```
procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.4.2.25 TCheckBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
```

```
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.4.2.26 TCheckBox.State Property

Indicates whether the check box is selected, deselected, or grayed.

Pascal

```
property State: TCheckBoxState;
```

C++

```
__property TCheckBoxState State;
```

Description

Use State to determine whether the check box is checked (cbChecked), unchecked (cbUnchecked), or grayed (cbGrayed).

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

AllowGrayed (see page 2281)

Checked (see page 2113)

30.1.4.2.27 TCheckBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.4.2.28 TCheckBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.4.2.29 TCheckBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.4.2.30 TCheckBox.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

30.1.4.3 TCheckBox Events

The events of the TCheckBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 2151)	Occurs when the user clicks the control.
	OnContextPopup (see page 2153)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2154)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2157)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2160)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2160)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2161)	Occurs when a control receives the input focus.
	OnExit (see page 2164)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2168)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2171)	Occurs when a key is pressed.
	OnKeyUp (see page 2172)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2174)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2175)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2177)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2178)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2178)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2180)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2182)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2183)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.4.3.1 TCheckBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).

- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.4.3.2 TCheckBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple

times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.4.3.3 TCheckBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
```

```
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.4.3.4 TCheckBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
```

```
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.4.3.5 TCheckBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.4.3.6 TCheckBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
```

```
  control was successfully dropped into and accepted by a
  another control.
  */
  void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
       int Y)
  {
    if (Sender->ClassNameIs("TLabel"))
    {
      AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
      if (Target)
        S = S + "accepted!";
      else
        S = S + "rejected.";
      Status->Caption = S;
    }
  }
```

Delphi Examples:

```
  {
  This code displays a message in a label named Status. The
  message displayed depends on whether the dragged label
  control was successfully dropped into and accepted by a
  another control.
  }
  procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
  var
    S: string;
  begin
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.4.3.7 TCheckBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter

event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
```

```
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.4.3.8 TCheckBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
```

```
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;
```

```
  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;
```

```
procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

30.1.4.3.9 TCheckBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
```

```
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
```

```
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
```

```
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.4.3.10 TCheckBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.4.3.11 TCheckBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

30.1.4.3.12 TCheckBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no

control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.4.3.13 TCheckBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
```

```
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  */
  int StartX, StartY; // Declare at the top of the form's unit
  // Use this code as the OnMouseDown event handler of the form:
  void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
  int X, int Y)
  {
    StartX = X;
    StartY = Y;
  }

  // Use this code as the OnMouseUp event handler of the form:
  void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
  int X, int Y)
  {
    Form1->Canvas->Rectangle(StartX, StartY, X, Y);
    StatusBar1->Panels->Items[0]->Text = "";
    StatusBar1->Panels->Items[1]->Text = "";
    StatusBar1->Panels->Items[2]->Text = "";
    StatusBar1->Panels->Items[3]->Text = "";
  }

  // Use this code as the OnMouseMove event handler of the form:
  void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
  {
    if (Shift.Contains(ssLeft)) // make sure button is down
    {
      if (Y > StartY)
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
  {
  The following example requires a form with a four-paneled
  status bar.  (Set the Width of the status panels to 150
  before running this example).  When the user presses a mouse
  button, moves the mouse, and releases the mouse button, a
  rectangle is drawn on the form. When the mouse button is
  released, the rectangle appears on the form's canvas. Its
  top-left and bottom-right corners are defined by the
  location of the mouse pointer when the user pressed and
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
```

```
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.4.3.14 TCheckBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.4.3.15 TCheckBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.4.3.16 TCheckBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
```

```
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.4.3.17 TCheckBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.4.3.18 TCheckBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.4.3.19 TCheckBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

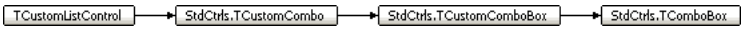
OnDragOver

OnDragDrop

30.1.5 TComboBox Class

TComboBox combines an edit box with a scrollable list.

Class Hierarchy



Pascal

```
TComboBox = class(TCustomComboBox);
```

C++

```
class TComboBox : public TCustomComboBox;
```

File

StdCtrls

Description

A TComboBox component is an edit box with a scrollable drop-down list attached to it. Users can select an item from the list or type directly into the edit box.

Note: The width of the button in a TComboBox is equal to the width that Windows uses for scrollbars. This width depends on the color scheme that the user has chosen (by right-clicking the desktop, selecting Properties, and choosing Appearance). On Windows XP, the user can specifically change the width of the scrollbar by clicking the Advanced button and selecting scrollbar as the Item. If you carefully size your combo box so all the text is visible, then a user running under a different color scheme may find that the text is obscured by the button. One solution is to set the scrollbar wide when developing the application. Most color schemes use a scrollbar that is at most 21 pixels wide.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
```

```
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

TCustomComboBox (see page 2322)

TEdit (see page 2463)

TListBox (see page 2657)

Combo Boxes

List Controls

30.1.5.1 TComboBox Members

The following tables list the members exposed by TComboBox.

Protected Fields

	Name	Description
	FDefEditProc (see page 2286)	Represents the address of the edit box procedure assigned by Windows.
	FDefListProc (see page 2287)	Represents the address of the list box procedure assigned by Windows.
	FDropHandle (see page 2287)	Represents the window handle for the entire combo box.
	FDroppingDown (see page 2287)	Specifies whether the list box is dropping down.
	FEditHandle (see page 2288)	Represents the window handle for the edit portion of the combo box.
	FEditInstance (see page 2288)	Represents the address of an internal routine associated with the edit control.
	FFocusChanged (see page 2288)	Specifies whether the combo box changed its focus state.
	FIsFocused (see page 2288)	Specifies whether the combo box is focused.
	FListHandle (see page 2289)	Represents the window handle for the list of items in the combo box.
	FListInstance (see page 2289)	Represents the address of an internal routine associated with the list box.
	FSaveIndex (see page 2289)	Represents the index of an item from the combo box.

Protected Methods

	Name	Description
V	AdjustDropDown (see page 2292)	Adjusts the size and position of the drop down list.
	Change (see page 2292)	Generates an OnChange (see page 2318) event.
	CloseUp (see page 2293)	Generates an OnCloseUp (see page 2320) event.
V	ComboWndProc (see page 2294)	Responds to Windows messages sent to the edit control or drop-down list.
V	CreateParams (see page 2326)	Initializes the window-creation parameter record.
V	CreateWnd (see page 2295)	Creates the Windows control that represents the combo box.
V	CreateWnd (see page 2327)	Creates the Windows control that represents the combo box.
V	DestroyWindowHandle (see page 2296)	Destroys all of the windows controls that implement the combo box.
V	DestroyWnd (see page 2327)	Destroys the Windows control that represents the combo box.
V	DoSetTextHint (see page 2328)	Sets the value of TextHint (see page 2337).
V	DrawItem (see page 2328)	Generates an OnDrawItem (see page 2338) event.
	DropDown (see page 2296)	Generates an OnDropDown (see page 2320) event.
V	EditWndProc (see page 2296)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2297)	Returns the value of the DroppedDown (see page 2306) property.
A	GetItemCount (see page 2298)	Returns the value of the ItemCount (see page 2307) property.
V	GetItemCount (see page 2328)	Returns the number of items in the Items (see page 2308) property list.
A	GetItemHt (see page 2298)	Returns the value of the ItemHeight (see page 2307) property.
V	GetItemHt (see page 2329)	Returns the value of the ItemHeight (see page 2307) property.
V	GetItemIndex (see page 2298)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 2299)	Indicates the class that implements the Items (see page 2308) property.
V	GetItemsClass (see page 2329)	Returns the class used to implement the Items (see page 2308) property.
	GetSelLength (see page 2299)	Returns the value of the SelLength (see page 2311) property.
	GetSelStart (see page 2299)	Returns the value of the SelStart (see page 2314) property.
	GetSelText (see page 2329)	Returns the value of the SelText (see page 2335) property.
V	KeyPress (see page 2329)	Responds to WM_CHAR messages sent to the combo box.
	ListWndProc (see page 2300)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2300)	Initializes the combo box after its form is loaded into memory.
V	MeasureItem (see page 2330)	Generates an OnMeasureItem (see page 2338) event.
	Select (see page 2301)	Generates an OnSelect (see page 2322) event.
	SelectItem (see page 2330)	Selects a specified item from the drop-down list.
V	SetDropDownCount (see page 2301)	Sets the value of the DropDownCount (see page 2306) property.
	SetDroppedDown (see page 2302)	Sets the value of the DroppedDown (see page 2306) property.
V	SetItemHeight (see page 2302)	Sets the value of the ItemHeight (see page 2307) property.
V	SetItemIndex (see page 2302)	Sets the value of the ItemIndex property.
V	SetItems (see page 2303)	Sets the value of the Items (see page 2308) property.
	SetMaxLength (see page 2303)	Sets the value of the MaxLength (see page 2310) property.

	SetSelLength (see page 2303)	Sets the value of the SelLength (see page 2311) property.
	SetSelStart (see page 2304)	Sets the value of the SelStart (see page 2314) property.
V	SetStyle (see page 2331)	Sets the Style (see page 2337) property.
V	WndProc (see page 2304)	Processes messages received by the combo box.
V	WndProc (see page 2331)	Processes messages received by the combo box.

Protected Properties

	Name	Description
R	EditHandle (see page 2307)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 2307)	Specifies the number of items in the drop-down list.
R	ListHandle (see page 2310)	Provides access to the window handle of the list of items in the combo box.

Public Methods

	Name	Description
V	AddItem (see page 2291)	Adds an item to the drop-down list of items.
V	Clear (see page 2293)	Deletes all text from the edit box and all items from the Items (see page 2308) list.
V	ClearSelection (see page 2293)	Deselects any selected items in the drop-down list.
V	CopySelection (see page 2294)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2285)	Creates and initializes a TCustomCombo (see page 2283) instance.
V	Create (see page 2325)	Creates and initializes a TCustomComboBox (see page 2322) instance.
V	DeleteSelected (see page 2295)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2286)	Destroys the combo box and frees its memory.
V	Destroy (see page 2326)	Destroys the combo box and frees its memory.
V	Focused (see page 2297)	Indicates whether the combo box has input focus.
V	GetCount (see page 2297)	Returns the value of the ItemCount (see page 2307) property.
V	SelectAll (see page 2301)	Selects the entire block of text in the edit box.

Public Properties

	Name	Description
R	Canvas (see page 2305)	Provides access to the drawing surface of the combo box.
	DroppedDown (see page 2306)	Indicates whether the drop-down list is currently displayed.
	SelLength (see page 2311)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2314)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 2335)	Represents the selected text in the edit region.

Published Events

	Name	Description
	OnChange (see page 2228)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 2230)	Occurs when the user clicks the control.
	OnCloseUp (see page 2232)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 2233)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2233)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2234)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2237)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2240)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 2240)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 2242)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2242)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2243)	Occurs when a control receives the input focus.
	OnExit (see page 2246)	Occurs when the input focus shifts away from one control to another.

	OnKeyDown (see page 2250)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2253)	Occurs when a key is pressed.
	OnKeyUp (see page 2254)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2256)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 2257)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2257)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 2257)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 2257)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2258)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2190)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2193)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 2194)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2194)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2195)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2195)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 2195)	Specifies which edges of the control are beveled.
	BevelInner (see page 2196)	Specifies the cut of the inner bevel.
	BevelKind (see page 2196)	Specifies the control's bevel style.
	BevelOuter (see page 2197)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2197)	Specifies the bi-directional mode for the control.
	CharCase (see page 2197)	Determines the case of the text in the combo box.
	Color (see page 2198)	Specifies the background color of the control.
	Constraints (see page 2199)	Specifies the size constraints for the control.
	Ctl3D (see page 2199)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2200)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2202)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2202)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2202)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 2204)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 2204)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2205)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2206)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2206)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 2207)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemIndex (see page 2208)	Specifies the index of the selected item.
	Items (see page 2210)	Provides access to the list of items (strings) in the list portion of the combo box.
	MaxLength (see page 2212)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentBiDiMode (see page 2212)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2213)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2213)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2214)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2214)	Determines where a control looks for its font information.
	ParentShowHint (see page 2214)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2215)	Identifies the pop-up menu associated with the control.

	ShowHint (see page 2220)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2221)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 2223)	Determines the display style of the combo box.
	TabOrder (see page 2223)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2224)	Determines if the user can tab to a control.
	Text (see page 2224)	Contains a text string associated with the control.
	TextHint (see page 2226)	Specifies the text that is displayed as a text watermark in the edit box of the combo box control.
	Visible (see page 2226)	Determines whether the component appears onscreen.

30.1.5.2 TComboBox Properties

The properties of the TComboBox class are listed here.

Published Properties

	Name	Description
	Align (see page 2190)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2193)	Specifies how the control is anchored to its parent.
	AutoCloseUp (see page 2194)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2194)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2195)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2195)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	BevelEdges (see page 2195)	Specifies which edges of the control are beveled.
	BevelInner (see page 2196)	Specifies the cut of the inner bevel.
	BevelKind (see page 2196)	Specifies the control's bevel style.
	BevelOuter (see page 2197)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2197)	Specifies the bi-directional mode for the control.
	CharCase (see page 2197)	Determines the case of the text in the combo box.
	Color (see page 2198)	Specifies the background color of the control.
	Constraints (see page 2199)	Specifies the size constraints for the control.
	Ctl3D (see page 2199)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2200)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2202)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2202)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2202)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownCount (see page 2204)	Specifies the maximum number of items displayed in the drop-down list.
	Enabled (see page 2204)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2205)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2206)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2206)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	ItemHeight (see page 2207)	Specifies the height, in pixels, of the items in the drop-down list.
	ItemIndex (see page 2208)	Specifies the index of the selected item.
	Items (see page 2210)	Provides access to the list of items (strings) in the list portion of the combo box.
	MaxLength (see page 2212)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentBiDiMode (see page 2212)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2213)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2213)	Determines where a component looks to determine if it should appear three dimensional.

	ParentDoubleBuffered (see page 2214)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2214)	Determines where a control looks for its font information.
	ParentShowHint (see page 2214)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2215)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2220)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2221)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 2223)	Determines the display style of the combo box.
	TabOrder (see page 2223)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2224)	Determines if the user can tab to a control.
	Text (see page 2224)	Contains a text string associated with the control.
	TextHint (see page 2226)	Specifies the text that is displayed as a text watermark in the edit box of the combo box control.
	Visible (see page 2226)	Determines whether the component appears onscreen.

30.1.5.2.1 TComboBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
```

```
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
    }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
```

```
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

30.1.5.2.2 TComboBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control

has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.5.2.3 TComboBox.AutoCloseUp Property

Specifies whether the drop-down closes up automatically when the user selects an item.

Pascal

```
property AutoCloseUp: Boolean;
```

C++

```
__property Boolean AutoCloseUp;
```

Description

When AutoCloseUp is true, the drop-down closes up automatically when the user selects an item.

See Also

AutoDropDown (see page 2334)

30.1.5.2.4 TComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, sometimes saving you keystrokes.

As you type, the ItemIndex (see page 2208) property is updated with the list entry that matches the characters you have entered. As you type, different list enrties may match, and ItemIndex (see page 2208) is changed accordingly.

There is a delay of AutoCompleteDelay (see page 2195) milliseconds between the characters typed and the autocomplete operation.

See Also

AutoCompleteDelay (see page 2195)

30.1.5.2.5 TComboBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2194) is true.

See Also

AutoComplete (see page 2194)

30.1.5.2.6 TComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

AutoComplete (see page 2332)

OnDropDown (see page 2320)

DroppedDown (see page 2306)

30.1.5.2.7 TComboBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

30.1.5.2.8 TComboBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

30.1.5.2.9 TComboBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

30.1.5.2.10 TComboBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

30.1.5.2.11 TComboBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.5.2.12 TComboBox.CharCase Property

Determines the case of the text in the combo box.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the combo box into upper or lower case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types into the combo box. Changing the CharCase property changes the actual text, not just its appearance; case information is lost and cannot be recaptured by changing CharCase back to ecNormal.

See Also

Text

30.1.5.2.13 TComboBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.5.2.14 TComboBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.5.2.15 TComboBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.5.2.16 TComboBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
```

```
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.5.2.17 TComboBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.5.2.18 TComboBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.5.2.19 TComboBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.5.2.20 TComboBox.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items property, the drop down list will display a scroll bar.

See Also

Items (see page 2308)

ItemCount (see page 2307)

30.1.5.2.21 TComboBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.5.2.22 TComboBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
```

```
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.5.2.23 TComboBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

30.1.5.2.24 TComboBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

30.1.5.2.25 TComboBox.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 2308)

OnMeasureItem (see page 2338)

Style (see page 2337)

30.1.5.2.26 TComboBox.ItemIndex Property

Specifies the index of the selected item.

Pascal

```
property ItemIndex;
```

C++

```
__property ItemIndex;
```

Description

Read ItemIndex to determine which item is selected. The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1. If the list control supports multiple selected items, ItemIndex is the index of the selected item that has focus.

Set ItemIndex programmatically to select an item by passing in the index value.

Note: To implement the ItemIndex property in a TCustomListControl descendant, override the protected GetItemIndex and SetItemIndex methods.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
           mtInformation, TMsgDlgButtons() << mbOK, 0);
}
/*
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
selects an item and clicks the button, the selected item in
the list box is moved to the top of the list box:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->MultiSelect = false;
```

```
  Button1->Caption = "Move to Top";
  for (int i = 1; i <= 10; i++)
    ListBox1->Items->Add("Item " + IntToStr(i));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ListBox1->ItemIndex >= 0 && ListBox1->ItemIndex <= 9)
    ListBox1->Items->Move(ListBox1->ItemIndex, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
             mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
{
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
clicks the button, the selected item in the list box is
moved to the top of the list box:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (ListBox1.ItemIndex in [0..9]) then
    ListBox1.Items.Move(ListBox1.ItemIndex, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  ListBox1.MultiSelect := False;
  Button1.Caption := 'Move to Top';
  for I := 1 to 10 do
    ListBox1.Items.Add('Item ' + IntToStr(I));
end;
```

See Also

GetItemIndex

SetItemIndex

30.1.5.2.27 TComboBox.Items Property

Provides access to the list of items (strings) in the list portion of the combo box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Read Items to access the list of items that appears in the combo box. Use the methods of Items to add, insert, delete and move items. Set the value of Items to copy the items from another string list.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", reinterpret_cast<TObject *>(vsIcon));
```

```
  ComboBox1->Items->AddObject("vsList", reinterpret_cast<TObject *>(vsList));
  ComboBox1->Items->AddObject("vsReport", reinterpret_cast<TObject *>(vsReport));
  ComboBox1->Items->AddObject("vsSmallIcon", reinterpret_cast<TObject *>(vsSmallIcon));
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 2';
    NewColumn.Width := 200;
    // Add View styles and constants to the Combo Box
    ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
    ComboBox1.Items.AddObject('vsList', TObject(vsList));
    ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
    ComboBox1.Items.AddObject(
      'vsSmallIcon', TObject(vsSmallIcon));
    // Display first item in the Combo Box
    ComboBox1.ItemIndex := 0;
```

```
  end;
end;

procedure TComboForm.ComboBox1Click(Sender: TObject);
begin
  with ComboBox1 do
    ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

end;
```

See Also

ItemIndex

TStrings

Assign

30.1.5.2.28 TComboBox.MaxLength Property

Specifies the maximum number of characters the user can type into the edit portion of the combo box.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that the user can enter into the edit portion of the combo box. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

See Also

Text

30.1.5.2.29 TComboBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.5.2.30 TComboBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.5.2.31 TComboBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.5.2.32 TComboBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.5.2.33 TComboBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.5.2.34 TComboBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.5.2.35 TComboBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
      TNotifyEvent();
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
```

```
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    PopupMenu1.AutoPopup := True;
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
```

```
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.5.2.36 TComboBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
```

```
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.5.2.37 TComboBox.Sorted Property

Determines whether the list portion of the combo box is alphabetized.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to sort the items in the Items list alphabetically. New items added to the list while Sorted is true are inserted in

the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

Items (see page 2308)

Using TDBListBox and TDBComboBox

30.1.5.2.38 TComboBox.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TComboBoxStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

When Style is csOwnerDrawFixed or csOwnerDrawVariable, the OnDrawItem event occurs when the list items need to be drawn. If Style is csOwnerDrawVariable, an OnMeasureItem event occurs as well.

See Also

Height

ItemHeight (see page 2307)

OnDrawItem (see page 2338)

OnMeasureItem (see page 2338)

30.1.5.2.39 TComboBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.5.2.40 TComboBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.5.2.41 TComboBox.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                       //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines (see page 2451)

30.1.5.2.42 TComboBox.TextHint Property

Specifies the text that is displayed as a text watermark in the edit box of the combo box control.

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

Use TextHint to get or set the watermark text that is shown in the edit box of the combo box control.

The disabled watermark is displayed as long as you do not modify the Text property of the combo box.

Note: This feature is only available in Vista.

See Also

TCustomComboBox (see page 2322)

30.1.5.2.43 TComboBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
```

```
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.5.3 TComboBox Events

The events of the TComboBox class are listed here.

Published Events

	Name	Description
	OnChange (see page 2228)	Occurs when the user changes the text displayed in the edit region.
	OnClick (see page 2230)	Occurs when the user clicks the control.
	OnCloseUp (see page 2232)	Occurs when the drop-down list closes up due to some user action.
	OnContextPopup (see page 2233)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2233)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2234)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2237)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2240)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 2240)	Occurs when the user opens the drop-down list.
	OnEndDock (see page 2242)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2242)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2243)	Occurs when a control receives the input focus.
	OnExit (see page 2246)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2250)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2253)	Occurs when a key is pressed.
	OnKeyUp (see page 2254)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2256)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnMouseEnter (see page 2257)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2257)	Occurs when the user moves the mouse outside of a control.
	OnSelect (see page 2257)	Occurs when the user selects a string in the drop-down list.
	OnStartDock (see page 2257)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2258)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.5.3.1 TComboBox.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
```

```
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = dynamic_cast<TTabSheet *>(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
  }
  ComboBox1->ItemIndex = 0;
  PageControl1->ActivePage = dynamic_cast<TTabSheet
*>(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
```

```
  TRichEdit to enter a string.  Select the string before
  justifying.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  const
    colorarray : Array[0..4] of TColor = (
      clYellow, clGreen, clBlue, clLime, clFuchsia);
  begin
    for i := 0 to PageControl1.PageCount - 1 do
      begin
      PageControl1.Pages[i].Brush.Color := colorarray[i];
      ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
        PageControl1.Pages[i]);
      end;
    ComboBox1.ItemIndex := 0;
    PageControl1.ActivePage := TTabSheet(ComboBox1.Items.Objects[ComboBox1.ItemIndex]);
  end;

  procedure TForm1.ComboBox1Change(Sender: TObject);
  begin
    if (Sender is TComboBox) then
      with (Sender as TComboBox) do
        PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
  end;
  {
  The following example assumes that a drive combo box, a file
  list box, and a directory list box are on a form. Add this
  code as the OnChange event handler for the drive combo box
  and the OnChange event handler for the directory list box.
  When the user changes the drive using the combo box, the
  directory list box and file list box will update to reflect
  the new drive and the current directory on that drive.  When
  the user double clicks on a directory in the file list box
  will update to reflect the new directory.
  }
  procedure TForm1.DirectoryListBox1Change(Sender: TObject);
  begin
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;

  procedure TForm1.DriveComboBox1Change(Sender: TObject);
  begin
    DirectoryListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;
```

See Also

Text

OnCloseUp (see page 2320)

30.1.5.3.2 TComboBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
  property OnClick;
```

C++

```
  __property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.5.3.3 TComboBox.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown (see page 2320)

DroppedDown (see page 2306)

30.1.5.3.4 TComboBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.5.3.5 TComboBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.5.3.6 TComboBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.5.3.7 TComboBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.5.3.8 TComboBox.OnDrawItem Event

Occurs when an item in an owner-draw combo box needs to be displayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Write an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

See Also

Canvas (see page 2305)

Items (see page 2308)

OnMeasureItem (see page 2338)

Style (see page 2337)

30.1.5.3.9 TComboBox.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

DroppedDown (see page 2306)

OnCloseUp (see page 2320)

30.1.5.3.10 TComboBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.5.3.11 TComboBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
```

```
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.5.3.12 TComboBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.5.3.13 TComboBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
```

```
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;
```

```
begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

30.1.5.3.14 TComboBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
```

```
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
```

```
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.5.3.15 TComboBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII

character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.5.3.16 TComboBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can

respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
```

```
  end;

  procedure TForm1.FormKeyDown(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := clAqua;
      if (ComboBox1.Text = 'KeyDown') then
        ComboBox1.Text := 'KeyDown2'
      else
        ComboBox1.Text := 'KeyDown';
      WriteLn(ComboBox1.Text);
    end;
  end;

  procedure TForm1.FormKeyUp(
    Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
```

See Also

OnKeyDown

OnKeyPress

30.1.5.3.17 TComboBox.OnMeasureItem Event

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

When Style is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem. Write an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

See Also

ItemHeight (see page 2307)

Items (see page 2308)

OnDrawItem (see page 2338)

Style (see page 2337)

30.1.5.3.18 TComboBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.5.3.19 TComboBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.5.3.20 TComboBox.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

AutoComplete (see page 2332)

ItemIndex

Text

30.1.5.3.21 TComboBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.5.3.22 TComboBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

30.1.6 TCustomButton Class

Base class for a push button control.

Class Hierarchy

Pascal

```
TCustomButton = class(TButtonControl);
```

C++

```
class TCustomButton : public TButtonControl;
```

File

StdCtrls

Description

Use TCustomButton as a base class for a standard push button on a form.

See Also

TButton (see page 2035)

30.1.6.1 TCustomButton Members

The following tables list the members exposed by TCustomButton.

Enumerations

	Name	Description
	TButtonStyle (see page 2261)	Style (see page 2275) of TButton (see page 2035).

Protected Methods

	Name	Description
	ActionChange (see page 2262)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2263)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2264)	Creates the underlying window.
	GetActionLinkClass (see page 2264)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2112)	Returns whether the button control is checked.
	Loaded (see page 2264)	Initializes the control after it is loaded from a stream.
	Notification (see page 2265)	Responds to notifications that components are being created or destroyed.
	SetButtonStyle (see page 2265)	Sets the Style (see page 2275) property for a button.

	SetChecked (see page 2112)	Sets the Checked (see page 2113) property.
	UpdateImageList (see page 2266)	Rebuilds the image list associated with the button.
	UpdateImages (see page 2266)	Updates the image list associated with the button.
	WndProc (see page 2112)	Handles Windows messages.

Protected Properties

	Name	Description
	Checked (see page 2113)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2115)	Specifies whether the button control accepts mouse clicks.
	WordWrap (see page 2115)	Specifies whether the button text wraps to fit the width of the control.

Public Events

	Name	Description
	OnDropDownClick (see page 2276)	Down arrow part of Vista style split button is clicked.

Public Methods

	Name	Description
	Click (see page 2263)	Simulates a mouse click, as if the user had clicked the button.
	Create (see page 2260)	Creates and initializes a button instance.
	Destroy (see page 2261)	Destroys an instance of TWinControl.
	UseRightToLeftAlignment (see page 2266)	Specifies whether the control's alignment is in a right-to-left mode.

Public Properties

	Name	Description
	Cancel (see page 2267)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	CommandLinkHint (see page 2267)	Text displayed as hint below button caption for Command Link.
	Default (see page 2268)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DisabledImageIndex (see page 2269)	Image index for disabled button state.
	DropDownMenu (see page 2269)	Split button drop down menu.
	ElevationRequired (see page 2270)	Put shield icon on button indicating elevated access rights required.
	HotImageIndex (see page 2270)	Image index for hot button state.
	ImageAlignment (see page 2271)	Alignment of image on button.
	ImageIndex (see page 2271)	Image index for normal button state.
	ImageMargins (see page 2272)	Margins of image on button.
	Images (see page 2272)	List of images for a button.
	ModalResult (see page 2273)	Determines whether and how the button closes its (modal) parent form.
	PressedImageIndex (see page 2274)	Image index for pressed button state.
	SelectedImageIndex (see page 2274)	Image index for selected button state.
	Style (see page 2275)	Style of a button.
	TabStop (see page 2275)	Determines if the user can tab to a control.

30.1.6.2 TCustomButton.Create Constructor

Creates and initializes a button instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomButton(TComponent * AOwner);
```

Description

Use Create to programmatically instantiate a button. Buttons added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the button instance. It becomes the value of the Owner property.

See Also

Owner

30.1.6.3 TCustomButton.Destroy Destructor

Destroys an instance of TWinControl.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomButton();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWinControl reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

Destroy

30.1.6.4 TCustomButton Enumerations

The enumerations of the TCustomButton class are listed here.

Enumerations

	Name	Description
	TButtonStyle (see page 2261)	Style (see page 2275) of TButton (see page 2035).

30.1.6.4.1 StdCtrls.TCustomButton.TButtonStyle Enumeration

Style (see page 2275) of TButton (see page 2035).

Pascal

```
TButtonStyle = (
  bsPushButton,
  bsCommandLink,
  bsSplitButton
);
```

C++

```
enum TButtonStyle {
  bsPushButton,
  bsCommandLink,
  bsSplitButton
};
```

File

StdCtrls

Description

TButtonStyle indicates a TButton (see page 2035)'s style, which is the value of its Style (see page 2275) property. Following are the possible values of TButtonStyle:

Value	Meaning
bsPushButton	The button is a standard push button.
bsCommandLink	Command Link style button. Display the text in CommandLinkHint (see page 2267) as a hint below the button caption. Windows Vista only.
bsSplitButton	Make the button a Vista style split button, which has a drop down menu. Windows Vista only.

See Also

TButton (see page 2035)

CommandLinkHint (see page 2267)

Style (see page 2275)

30.1.6.5 TCustomButton Methods

The methods of the TCustomButton class are listed here.

Protected Methods

	Name	Description
	ActionChange (see page 2262)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2263)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2264)	Creates the underlying window.
	GetActionLinkClass (see page 2264)	Returns the appropriate action link class that is associated with button controls.
	Loaded (see page 2264)	Initializes the control after it is loaded from a stream.
	Notification (see page 2265)	Responds to notifications that components are being created or destroyed.
	SetButtonStyle (see page 2265)	Sets the Style (see page 2275) property for a button.
	UpdateImageList (see page 2266)	Rebuilds the image list associated with the button.
	UpdateImages (see page 2266)	Updates the image list associated with the button.

Public Methods

	Name	Description
	Click (see page 2263)	Simulates a mouse click, as if the user had clicked the button.
	UseRightToLeftAlignment (see page 2266)	Specifies whether the control's alignment is in a right-to-left mode.

30.1.6.5.1 TCustomButton.ActionChange Method

Updates the button control to reflect changes in its associated action.

Pascal

```
procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;
```

C++

```
virtual __fastcall ActionChange(TObject * Sender, Boolean CheckDefaults);
```

Description

ActionChange is called automatically when the associated action changes. It updates the button control to reflect the action's current Checked state. Then it calls the inherited method, which checks the associated properties for that client type.

The Sender parameter is the associated action. The CheckDefaults parameter specifies whether properties and event handlers should be changed if they differ from the default values. When CheckDefaults is true, properties and event handlers are only changed when the current values are the default values (that is, when the values have not been previously changed). When CheckDefaults is false, properties and event handlers are changed, regardless of their current values.

See Also

TCustomAction

30.1.6.5.2 TCustomButton.Click Method

Simulates a mouse click, as if the user had clicked the button.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Calling Click assigns the value of the button's ModalResult (see page 2273) property to the ModalResult (see page 2273) property of its parent form, and then generates an OnClick event.

See Also

ModalResult (see page 2273)

OnClick

30.1.6.5.3 TCustomButton.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

30.1.6.5.4 TCustomButton.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

30.1.6.5.5 TCustomButton.GetActionLinkClass Method

Returns the appropriate action link class that is associated with button controls.

Pascal

```
function GetActionLinkClass: TControlActionLinkClass; override;
```

C++

```
virtual __fastcall TControlActionLinkClass GetActionLinkClass();
```

Description

GetActionLinkClass is used to get the appropriate class type for the action link used with button controls. GetActionLinkClass is used internally to create action link objects for clients when a button control has an associated action. The action link is used to link the action to the button control client. Each Action link class type is designed to link specific actions to clients based upon the types of properties the client supports. For button controls, GetActionLinkClass returns the TButtonActionLink class type.

See Also

TButtonActionLink (see page 2106)

30.1.6.5.6 TCustomButton.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

30.1.6.5.7 TCustomButton.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

30.1.6.5.8 TCustomButton.SetButtonStyle Method

Sets the Style (see page 2275) property for a button.

Pascal

```
procedure SetButtonStyle(ADefault: Boolean); virtual;
```

C++

```
virtual __fastcall SetButtonStyle(Boolean ADefault);
```

Description

The SetButtonStyle method is called to change the Style (see page 2275) property according to the button's state, whenever a button changes its focus state.

See Also

TButtonStyle (see page 2261)

Style (see page 2275)

30.1.6.5.9 TCustomButton.UpdateImageList Method

Rebuilds the image list associated with the button.

Pascal

```
procedure UpdateImageList; virtual;
```

C++

```
virtual __fastcall UpdateImageList();
```

Description

The UpdateImageList method is called to update the image list associated with the button control.

See Also

UpdateImages (see page 2266)

30.1.6.5.10 TCustomButton.UpdateImages Method

Updates the image list associated with the button.

Pascal

```
procedure UpdateImages; virtual;
```

C++

```
virtual __fastcall UpdateImages();
```

Description

The UpdateImages method is called to add images to the image list associated with the button control.

See Also

UpdateImageList (see page 2266)

AddImage

30.1.6.5.11 TCustomButton.UseRightToLeftAlignment Method

Specifies whether the control's alignment is in a right-to-left mode.

Pascal

```
function UseRightToLeftAlignment: Boolean; override;
```

C++

```
virtual __fastcall Boolean UseRightToLeftAlignment();
```

Description

For this class, UseRightToLeftAlignment always returns false. This is because the alignment of a button should never be reversed automatically when running in Middle Eastern locales.

See Also

ModalResult (see page 2273)

OnClick

30.1.6.6 TCustomButton Properties

The properties of the TCustomButton class are listed here.

Public Properties

	Name	Description
	Cancel (see page 2267)	Determines whether the button's OnClick event handler executes when the Escape key is pressed.
	CommandLinkHint (see page 2267)	Text displayed as hint below button caption for Command Link.
	Default (see page 2268)	Determines whether the button's OnClick event handler executes when the Enter key is pressed.
	DisabledImageIndex (see page 2269)	Image index for disabled button state.
	DropDownMenu (see page 2269)	Split button drop down menu.
	ElevationRequired (see page 2270)	Put shield icon on button indicating elevated access rights required.
	HotImageIndex (see page 2270)	Image index for hot button state.
	ImageAlignment (see page 2271)	Alignment of image on button.
	ImageIndex (see page 2271)	Image index for normal button state.
	ImageMargins (see page 2272)	Margins of image on button.
	Images (see page 2272)	List of images for a button.
	ModalResult (see page 2273)	Determines whether and how the button closes its (modal) parent form.
	PressedImageIndex (see page 2274)	Image index for pressed button state.
	SelectedImageIndex (see page 2274)	Image index for selected button state.
	Style (see page 2275)	Style of a button.
	TabStop (see page 2275)	Determines if the user can tab to a control.

30.1.6.6.1 TCustomButton.Cancel Property

Determines whether the button's OnClick event handler executes when the Escape key is pressed.

Pascal

```
property Cancel: Boolean;
```

C++

```
__property Boolean Cancel;
```

Description

If Cancel is true, the button's OnClick event handler executes when the user presses Esc. Although an application can have more than one Cancel button, the form calls the OnClick event handler only for the first visible button in the tab order.

See Also

Default (see page 2268)

OnClick

30.1.6.6.2 TCustomButton.CommandLinkHint Property

Text displayed as hint below button caption for Command Link.

Pascal

```
property CommandLinkHint: string;
```

C++

```
__property UnicodeString CommandLinkHint;
```

Description

CommandLinkHint is the text displayed as a hint below the button caption when the Style (see page 2275) property is **bsCommandLink** to specify a Command Link button.

This is available on Windows Vista only.

See Also

Style (see page 2275)

TButtonStyle (see page 2261)

30.1.6.6.3 TCustomButton.Default Property

Determines whether the button's OnClick event handler executes when the Enter key is pressed.

Pascal

```
property Default: Boolean;
```

C++

```
__property Boolean Default;
```

Description

If Default is true, the button's OnClick event handler executes when the user presses Enter.

Although an application can have more than one Default button, the form calls the OnClick event handler only for the first visible button in the tab order. Moreover, any button that has focus becomes the Default button temporarily; hence, if the user selects another button before pressing Enter, the selected button's OnClick event handler executes instead.

C++ Examples:

```
/*
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  FileListBox1->FileEdit = Edit1;
  FilterComboBox1->FileList = FileListBox1;
  DirectoryListBox1->FileList = FileListBox1;
  DirectoryListBox1->DirLabel = Label1;
  DriveComboBox1->DirList = DirectoryListBox1;
  Button1->Default = true;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FileListBox1->ApplyFilePath(Edit1->Text);
}
```

Delphi Examples:

```
{
This example uses a file list box, a directory list box, a
filter combo box, a drive combo box, a label, an edit box,
and a button on a form. When the user runs the application
and enters a path or filename in the edit box and presses
the button, all the controls update:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FileListBox1.ApplyFilePath(Edit1.Text);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FileListBox1.FileEdit := Edit1;
  FilterComboBox1.FileList := FileListBox1;
  DirectoryListBox1.FileList := FileListBox1;
  DirectoryListBox1.DirLabel := Label1;
  DriveComboBox1.DirList := DirectoryListBox1;
  Button1.Default := True;
end;
```

See Also

Cancel (see page 2267)

OnClick

30.1.6.6.4 TCustomButton.DisabledImageIndex Property

Image index for disabled button state.

Pascal

```
property DisabledImageIndex: TImageIndex;
```

C++

```
__property TImageIndex DisabledImageIndex;
```

Description

DisabledImageIndex is an image index into the Images (see page 2272) property for the disabled button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

HotImageIndex (see page 2270)

ImageIndex (see page 2271)

Images (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

30.1.6.6.5 TCustomButton.DropDownMenu Property

Split button drop down menu.

Pascal

```
property DropDownMenu: TPopupMenu;
```

C++

```
__property TPopupMenu DropDownMenu;
```

Description

When the Style (see page 2275) property is **bsSplitButton**, the button is a Vista style split button with a drop down menu. DropDownMenu specifies the drop down menu for the down arrow button part. The OnDropDownClick (see page 2276) event also occurs when the down arrow button part is clicked.

This is available on Windows Vista only.

See Also

TPopupMenu

Style (see page 2275)

OnDropDownClick (see page 2276)

TButtonStyle (see page 2261)

30.1.6.6.6 TCustomButton.ElevationRequired Property

Put shield icon on button indicating elevated access rights required.

Pascal

```
property ElevationRequired: Boolean;
```

C++

```
__property Boolean ElevationRequired;
```

Description

When ElevationRequired is true, the button has a shield icon, indicating that elevated access rights are required. Vista UI guidelines specify that when there is a button that performs an operation requiring elevated access rights, it should have this shield icon.

This icon overrides any of the properties specifying a button image for various button states, as in ImageIndex (see page 2271).

This is available on Windows Vista only.

See Also

ImageIndex (see page 2271)

Images (see page 2272)

30.1.6.6.7 TCustomButton.HotImageIndex Property

Image index for hot button state.

Pascal

```
property HotImageIndex: TImageIndex;
```

C++

```
__property TImageIndex HotImageIndex;
```

Description

HotImageIndex is an image index into the Images (see page 2272) property for the hot button state, i.e., the cursor hovering over the button. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

ImageIndex (see page 2271)

Images (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

30.1.6.6.8 TCustomButton.ImageAlignment Property

Alignment of image on button.

Pascal

```
property ImageAlignment: TImageAlignment;
```

C++

```
__property TImageAlignment ImageAlignment;
```

Description

TImageAlignment (see page 3002) indicates the alignment of an image on a button. It is a TImageAlignment (see page 3002) type.

See Also

ImageMargins (see page 2272)

Images (see page 2272)

TImageAlignment (see page 3002)

30.1.6.6.9 TCustomButton.ImageIndex Property

Image index for normal button state.

Pascal

```
property ImageIndex: TImageIndex;
```

C++

```
__property TImageIndex ImageIndex;
```

Description

ImageIndex is an image index into the Images (see page 2272) property for the normal button state. If the index value is -1, no image is used.

In Vista, a standard green arrow is used when the index is -1 and the Style (see page 2275) property is **bsCommandLink** for a Command Link button.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

Images (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

Style (see page 2275)

TButtonStyle (see page 2261)

30.1.6.6.10 TCustomButton.ImageMargins Property

Margins of image on button.

Pascal

```
property ImageMargins: TImageMargins;
```

C++

```
__property TImageMargins ImageMargins;
```

Description

A button image may be positioned using margins. ImageMargins holds aTImageMargins object containing the margins of an image on a button. The TImageMargins.OnChange event is triggered when a margin changes.

See Also

ImageAlignment (see page 2271)

TImageMargins (see page 2602)

Images (see page 2272)

OnChange (see page 2607)

30.1.6.6.11 TCustomButton.Images Property

List of images for a button.

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

Images is a TCustomImageList object listing the various images that can be placed on a button.

The button's image depends on the button state. These images are determined by the value of the image state properties that index ImageIndex (see page 2271): DisabledImageIndex (see page 2269), HotImageIndex (see page 2270), ImageIndex (see page 2271), PressedImageIndex (see page 2274), and SelectedImageIndex (see page 2274). When the ImageIndex (see page 2271) index is -1, it generally means that no image is used for the normal state. For the other state image indexes,

-1 means to use the same image as the normal state.

The ImageMargins (see page 2272) and ImageAlignment (see page 2271) properties orient the image inside the button. The image is not sized to fit the button, and the button is not stretched.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

ImageAlignment (see page 2271)

ImageIndex (see page 2271)

ImageMargins (see page 2272)

PressedImageIndex (see page 2274)

SelectedImageIndex (see page 2274)

30.1.6.6.12 TCustomButton.ModalResult Property

Determines whether and how the button closes its (modal) parent form.

Pascal

```
property ModalResult: TModalResult;
```

C++

```
__property TModalResult ModalResult;
```

Description

Setting the button component's ModalResult property is an easy way to make clicking the button close a modal form. When a button is clicked, the ModalResult property of its parent form is set to the same value as the button's ModalResult property.

For example, if a dialog box has OK and Cancel (see page 2267) buttons, their ModalResult properties could be set at design time to mrOk and mrCancel, respectively. At runtime, clicking the OK button then changes the dialog's ModalResult property to mrOk, and clicking the Cancel (see page 2267) button changes the dialog's ModalResult property to mrCancel. Unless further processing is required, no OnClick event handlers are required for the buttons.

The following constants are of TModalResult type and defined for the ModalResult property:

Constant	Value	Meaning
mrNone	0	None. Used as a default value before the user exits.
mrOk	idOK	The user exited with OK button.
mrCancel	idCancel	The user exited with the CANCEL button.
mrAbort	idAbort	The user exited with the ABORT button.
mrRetry	idRetry	The user exited with the RETRY button.
mrIgnore	idIgnore	The user exited with the IGNORE button.
mrYes	idYes	The user exited with the YES button.
mrNo	idNo	The user exited with the NO button.
mrAll	mrNo + 1	The user exited with the ALL button.

mrNoToAll	mrAll + 1	The user exited with the NO TO ALL button.
mrYesToAll	mrNoToAll + 1	The user exited with the YES TO ALL button.
mrClose	mrYesToAll + 1	The user exited with the CLOSE button.

See Also

TModalResult

30.1.6.6.13 TCustomButton.PressedImageIndex Property

Image index for pressed button state.

Pascal

```
property PressedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex PressedImageIndex;
```

Description

PressedImageIndex is an image index into the Images (see page 2272) property for the pressed button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

ImageIndex (see page 2271)

Images (see page 2272)

SelectedImageIndex (see page 2274)

30.1.6.6.14 TCustomButton.SelectedImageIndex Property

Image index for selected button state.

Pascal

```
property SelectedImageIndex: TImageIndex;
```

C++

```
__property TImageIndex SelectedImageIndex;
```

Description

SelectedImageIndex is an image index into the Images (see page 2272) property for the selected button state. If the index value is -1, the same image is used as in the normal button state.

This is available only on Windows XP and Vista.

See Also

TCustomImageList

DisabledImageIndex (see page 2269)

HotImageIndex (see page 2270)

ImageIndex (see page 2271)

Images (see page 2272)

PressedImageIndex (see page 2274)

30.1.6.6.15 TCustomButton.Style Property

Style of a button.

Pascal

```
property Style: TButtonStyle;
```

C++

```
__property TButtonStyle Style;
```

Description

Style indicates a button's style, which is a TButtonStyle (see page 2261) type.

See Also

TButtonStyle (see page 2261)

30.1.6.6.16 TCustomButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.6.7 TCustomButton Events

The events of the TCustomButton class are listed here.

Public Events

	Name	Description
	OnDropDownClick (see page 2276)	Down arrow part of Vista style split button is clicked.

30.1.6.7.1 TCustomButton.OnDropDownClick Event

Down arrow part of Vista style split button is clicked.

Pascal

```
property OnDropDownClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDownClick;
```

Description

OnDropDownClick occurs when the down arrow part of a Vista style split button is clicked. This only applies when the button's Style (see page 2275) property is **bsSplitButton**.

This is available on Windows Vista only.

See Also

DropDownMenu (see page 2269)

Style (see page 2275)

TButtonStyle (see page 2261)

30.1.7 TCustomCheckBox Class

TCustomCheckBox is the ancestor of all check-box components.

Class Hierarchy

Pascal

```
TCustomCheckBox = class(TButtonControl);
```

C++

```
class TCustomCheckBox : public TButtonControl;
```

File

StdCtrls

Description

TCustomCheckBox is an abstract class from which check-box components, including TCheckBox (see page 2116) and TDBCheckBox, descend. Check boxes present the user with options that can be selected (checked) or deselected (unchecked).

See Also

TCheckBox (see page 2116)

TDBCheckBox

30.1.7.1 TCustomCheckBox Members

The following tables list the members exposed by TCustomCheckBox.

Protected Methods

	Name	Description
	ActionChange (see page 2110)	Updates the button control to reflect changes in its associated action.
	Click (see page 2278)	Click generates an OnClick event.
	CreateParams (see page 2278)	Initializes a window-creation parameter record.
	CreateWnd (see page 2279)	Creates a Windows control corresponding to the check box.
	GetActionLinkClass (see page 2111)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2279)	Returns the value of the Checked (see page 2113) property.
	SetChecked (see page 2280)	Sets the value of the Checked (see page 2113) property.
	Toggle (see page 2280)	Changes the state of the check box.
	WndProc (see page 2112)	Handles Windows messages.

Protected Properties

	Name	Description
	Alignment (see page 2281)	Controls the position of the check box's caption.
	AllowGrayed (see page 2281)	Determines whether check box can be in a "grayed" state.
	Checked (see page 2113)	Specifies whether the button control is checked.
	ClicksDisabled (see page 2115)	Specifies whether the button control accepts mouse clicks.
	State (see page 2282)	Indicates whether the check box is selected, deselected, or grayed.
	WordWrap (see page 2115)	Specifies whether the button text wraps to fit the width of the control.

Public Methods

	Name	Description
	Create (see page 2277)	Creates and initializes a TCustomCheckBox (see page 2276) instance.
	GetControlsAlignment (see page 2279)	Returns the alignment of the check box caption.

Published Properties

	Name	Description
	TabStop (see page 2282)	Determines if the user can tab to a control.

30.1.7.2 TCustomCheckBox.Create Constructor

Creates and initializes a TCustomCheckBox (see page 2276) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomCheckBox(TComponent * AOwner);
```

Description

Call Create to generate a check box at runtime. Check boxes placed on forms at design time, Create is called automatically.

AOwner is a component, typically the form, that is responsible for freeing the check box instance.

See Also

Owner

30.1.7.3 TCustomCheckBox Methods

The methods of the TCustomCheckBox class are listed here.

Protected Methods

	Name	Description
	Click (see page 2278)	Click generates an OnClick event.
	CreateParams (see page 2278)	Initializes a window-creation parameter record.
	CreateWnd (see page 2279)	Creates a Windows control corresponding to the check box.
	GetChecked (see page 2279)	Returns the value of the Checked (see page 2113) property.
	SetChecked (see page 2280)	Sets the value of the Checked (see page 2113) property.
	Toggle (see page 2280)	Changes the state of the check box.

Public Methods

	Name	Description
	GetControlsAlignment (see page 2279)	Returns the alignment of the check box caption.

30.1.7.3.1 TCustomCheckBox.Click Method

Click generates an OnClick event.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

The protected Click method is called automatically when the check box is clicked with the mouse. In addition to generating an OnClick event, it sends a message to the parent control's window to indicate that the state of the check box has changed.

See Also

OnClick

Parent

30.1.7.3.2 TCustomCheckBox.CreateParams Method

Initializes a window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2279) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. CreateParams fills in the parameter record that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

In TCustomCheckBox (see page 2276), CreateParams calls the inherited method, then makes specific adjustments for check boxes.

See Also

CreateWnd

CreateWindowHandle

30.1.7.3.3 TCustomCheckBox.CreateWnd Method

Creates a Windows control corresponding to the check box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the check box is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. In TCustomCheckBox (see page 2276), this method calls the inherited method to generate the underlying windows control, then sends a BM_SETCHECK message to set the check state of the control.

See Also

CreateParams (see page 2278)

30.1.7.3.4 TCustomCheckBox.GetChecked Method

Returns the value of the Checked (see page 2113) property.

Pascal

```
function GetChecked: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the protected implementation of the Checked (see page 2113) property. Override this method and the SetChecked (see page 2280) method to change the Checked (see page 2113) property in descendant classes.

See Also

Checked (see page 2113)

SetChecked (see page 2280)

30.1.7.3.5 TCustomCheckBox.GetControlsAlignment Method

Returns the alignment of the check box caption.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

GetControlsAlignment indicates the alignment of the check box caption. Usually, the caption is right-justified, but if the UseRightToLeftAlignment method returns true and the check mark is on the left, then GetControlsAlignment indicates that the caption should be left-justified.

See Also

Caption

Alignment (see page 2281)

UseRightToLeftAlignment

30.1.7.3.6 TCustomCheckBox.SetChecked Method

Sets the value of the Checked (see page 2113) property.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked is the protected implementation of the Checked (see page 2113) property. Override this method and the GetChecked (see page 2279) method to change the Checked (see page 2113) property in descendant classes.

See Also

Checked (see page 2113)

GetChecked (see page 2279)

30.1.7.3.7 TCustomCheckBox.Toggle Method

Changes the state of the check box.

Pascal

```
procedure Toggle; virtual;
```

C++

```
virtual __fastcall Toggle();
```

Description

Use the Toggle method to select or deselect the check box programmatically. Toggle switches among the valid check-box states—from checked to unchecked to grayed, or from checked to unchecked.

See Also

AllowGrayed (see page 2281)

State (see page 2282)

30.1.7.4 TCustomCheckBox Properties

The properties of the TCustomCheckBox class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 2281)	Controls the position of the check box's caption.
	AllowGrayed (see page 2281)	Determines whether check box can be in a "grayed" state.
	State (see page 2282)	Indicates whether the check box is selected, deselected, or grayed.

Published Properties

	Name	Description
	TabStop (see page 2282)	Determines if the user can tab to a control.

30.1.7.4.1 TCustomCheckBox.Alignment Property

Controls the position of the check box's caption.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

If Alignment is set to taRightJustify, the caption appears to the right of the check box. If Alignment is set to taLeftJustify, the caption appears to the left of the check box.

30.1.7.4.2 TCustomCheckBox.AllowGrayed Property

Determines whether check box can be in a "grayed" state.

Pascal

```
property AllowGrayed: Boolean;
```

C++

```
__property Boolean AllowGrayed;
```

Description

If AllowGrayed is set to true, the check box has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to false, the check box has only two possible states: checked and unchecked.

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

Checked (see page 2113)

State (see page 2282)

30.1.7.4.3 TCustomCheckBox.State Property

Indicates whether the check box is selected, deselected, or grayed.

Pascal

```
property State: TCheckBoxState;
```

C++

```
__property TCheckBoxState State;
```

Description

Use State to determine whether the check box is checked (cbChecked), unchecked (cbUnchecked), or grayed (cbGrayed).

C++ Examples:

```
/*
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  CheckBox1->AllowGrayed = true;
  CheckBox1->State = cbChecked;
}
```

Delphi Examples:

```
{
This example uses a check box on a form. When the
application runs, the check box is initially checked. When
the user clicks it, the check box is unchecked. Clicking it
again grays the check box.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  CheckBox1.AllowGrayed := True;
  CheckBox1.State := cbChecked;
end;
```

See Also

AllowGrayed (see page 2281)

Checked (see page 2113)

30.1.7.4.4 TCustomCheckBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

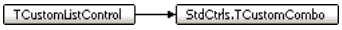
See Also

TabOrder

30.1.8 TCustomCombo Class

TCustomCombo is the base class for all combo boxes.

Class Hierarchy



Pascal

```
TCustomCombo = class(TCustomListControl);
```

C++

```
class TCustomCombo : public TCustomListControl;
```

File

StdCtrls

Description

Use TCustomCombo as a base class when defining a control that represents a specialized combo box. A combo box is an edit box with a scrollable list attached to it; users can select an item from the list or type directly into the edit box. Combo boxes can present the list as a drop-down list or as a list box that is always visible.

Typically, combo box classes do not descend directly from TCustomCombo. Rather, they descend from one of TCustomCombo's descendants, TCustomComboBox (see page 2322) or TCustomComboBoxEx. Use TCustomCombo as a base class for a simple combo box, or TCustomComboBoxEx as a base class for a combo box that supports the extended combo box features such as images on the list entries.

Do not create instances of TCustomCombo. To put a combo box in a form, use a descendant of TCustomCombo, such as TComboBox (see page 2184) or TComboBoxEx.

See Also

TComboBox (see page 2184)

TComboBoxEx

TDBComboBox

TColorBox

TDriveComboBox

TFilterComboBox

TCustomComboBox (see page 2322)

TCustomComboBoxEx

30.1.8.1 TCustomCombo Members

The following tables list the members exposed by TCustomCombo.

Protected Events

	Name	Description
	OnChange (see page 2318)	Occurs when the user changes the text displayed in the edit region.
	OnCloseUp (see page 2320)	Occurs when the drop-down list closes up due to some user action.
	OnDropDown (see page 2320)	Occurs when the user opens the drop-down list.
	OnSelect (see page 2322)	Occurs when the user selects a string in the drop-down list.

Protected Fields

	Name	Description
	FDefEditProc (see page 2286)	Represents the address of the edit box procedure assigned by Windows.
	FDefListProc (see page 2287)	Represents the address of the list box procedure assigned by Windows.
	FDropHandle (see page 2287)	Represents the window handle for the entire combo box.
	FDroppingDown (see page 2287)	Specifies whether the list box is dropping down.
	FEditHandle (see page 2288)	Represents the window handle for the edit portion of the combo box.
	FEditInstance (see page 2288)	Represents the address of an internal routine associated with the edit control.
	FFocusChanged (see page 2288)	Specifies whether the combo box changed its focus state.
	FIsFocused (see page 2288)	Specifies whether the combo box is focused.
	FListHandle (see page 2289)	Represents the window handle for the list of items in the combo box.
	FListInstance (see page 2289)	Represents the address of an internal routine associated with the list box.
	FSaveIndex (see page 2289)	Represents the index of an item from the combo box.

Protected Methods

	Name	Description
V	AdjustDropDown (see page 2292)	Adjusts the size and position of the drop down list.
	Change (see page 2292)	Generates an OnChange (see page 2318) event.
	CloseUp (see page 2293)	Generates an OnCloseUp (see page 2320) event.
V	ComboWndProc (see page 2294)	Responds to Windows messages sent to the edit control or drop-down list.
V	CreateWnd (see page 2295)	Creates the Windows control that represents the combo box.
V	DestroyWindowHandle (see page 2296)	Destroys all of the windows controls that implement the combo box.
	DropDown (see page 2296)	Generates an OnDropDown (see page 2320) event.
V	EditWndProc (see page 2296)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2297)	Returns the value of the DroppedDown (see page 2306) property.
A	GetItemCount (see page 2298)	Returns the value of the ItemCount (see page 2307) property.
A	GetItemHt (see page 2298)	Returns the value of the ItemHeight (see page 2307) property.
V	GetItemIndex (see page 2298)	Returns the value of the ItemIndex property.
A	GetItemsClass (see page 2299)	Indicates the class that implements the Items (see page 2308) property.
	GetSelLength (see page 2299)	Returns the value of the SelLength (see page 2311) property.
	GetSelStart (see page 2299)	Returns the value of the SelStart (see page 2314) property.
	ListWndProc (see page 2300)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2300)	Initializes the combo box after its form is loaded into memory.
	Select (see page 2301)	Generates an OnSelect (see page 2322) event.
V	SetDropDownCount (see page 2301)	Sets the value of the DropDownCount (see page 2306) property.
	SetDroppedDown (see page 2302)	Sets the value of the DroppedDown (see page 2306) property.
V	SetItemHeight (see page 2302)	Sets the value of the ItemHeight (see page 2307) property.
V	SetItemIndex (see page 2302)	Sets the value of the ItemIndex property.
V	SetItems (see page 2303)	Sets the value of the Items (see page 2308) property.

	SetMaxLength (see page 2303)	Sets the value of the MaxLength (see page 2310) property.
	SetSelLength (see page 2303)	Sets the value of the SelLength (see page 2311) property.
	SetSelStart (see page 2304)	Sets the value of the SelStart (see page 2314) property.
V	WndProc (see page 2304)	Processes messages received by the combo box.

Protected Properties

	Name	Description
	DropDownCount (see page 2306)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 2307)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 2307)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 2307)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 2310)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 2310)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentColor (see page 2311)	Determines where a control looks for its color information.

Public Methods

	Name	Description
V	AddItem (see page 2291)	Adds an item to the drop-down list of items.
V	Clear (see page 2293)	Deletes all text from the edit box and all items from the Items (see page 2308) list.
V	ClearSelection (see page 2293)	Deselects any selected items in the drop-down list.
V	CopySelection (see page 2294)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2285)	Creates and initializes a TCustomCombo (see page 2283) instance.
V	DeleteSelected (see page 2295)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2286)	Destroys the combo box and frees its memory.
V	Focused (see page 2297)	Indicates whether the combo box has input focus.
V	GetCount (see page 2297)	Returns the value of the ItemCount (see page 2307) property.
V	SelectAll (see page 2301)	Selects the entire block of text in the edit box.

Public Properties

	Name	Description
R	Canvas (see page 2305)	Provides access to the drawing surface of the combo box.
	DroppedDown (see page 2306)	Indicates whether the drop-down list is currently displayed.
	Items (see page 2308)	Provides access to the list of items (strings) in the list portion of the combo box.
	SelLength (see page 2311)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2314)	Specifies the position of the first selected character in the edit portion of the combo box.
	TabStop (see page 2317)	Determines if the user can tab to a control.

30.1.8.2 TCustomCombo.Create Constructor

Creates and initializes a TCustomCombo (see page 2283) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomCombo(TComponent * AOwner);
```

Description

Call Create to generate a combo box at runtime. Combo boxes placed on forms at design time are created automatically.

AOwner is a component (typically a form) that is responsible for freeing the combo box instance.

See Also

Owner

30.1.8.3 TCustomCombo.Destroy Destructor

Destroys the combo box and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomCombo();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the combo box is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

30.1.8.4 TCustomCombo Fields

The fields of the TCustomCombo class are listed here.

Protected Fields

	Name	Description
	FDefEditProc (see page 2286)	Represents the address of the edit box procedure assigned by Windows.
	FDefListProc (see page 2287)	Represents the address of the list box procedure assigned by Windows.
	FDropHandle (see page 2287)	Represents the window handle for the entire combo box.
	FDroppingDown (see page 2287)	Specifies whether the list box is dropping down.
	FEditHandle (see page 2288)	Represents the window handle for the edit portion of the combo box.
	FEditInstance (see page 2288)	Represents the address of an internal routine associated with the edit control.
	FFocusChanged (see page 2288)	Specifies whether the combo box changed its focus state.
	FIsFocused (see page 2288)	Specifies whether the combo box is focused.
	FListHandle (see page 2289)	Represents the window handle for the list of items in the combo box.
	FListInstance (see page 2289)	Represents the address of an internal routine associated with the list box.
	FSaveIndex (see page 2289)	Represents the index of an item from the combo box.

30.1.8.4.1 TCustomCombo.FDefEditProc Field

Represents the address of the edit box procedure assigned by Windows.

Pascal

```
FDefEditProc: TWindowProcPtr;
```

C++

```
TWindowProcPtr FDefEditProc;
```

Description

The FDefEditProc field is used to handle Windows messages sent to the edit box control inside the combo box, that cannot be handled directly by Delphi.

See Also

FDefListProc (see page 2287)

30.1.8.4.2 TCustomCombo.FDefListProc Field

Represents the address of the list box procedure assigned by Windows.

Pascal

```
FDefListProc: TWindowProcPtr;
```

C++

```
TWindowProcPtr FDefListProc;
```

Description

The FDefListProc field is used to handle Windows messages sent to the list box control inside the combo box, that cannot be handled directly by Delphi.

See Also

FDefEditProc (see page 2286)

30.1.8.4.3 TCustomCombo.FDropHandle Field

Represents the window handle for the entire combo box.

Pascal

```
FDropHandle: HWnd;
```

C++

```
HWnd FDropHandle;
```

Description

FDropHandle is the window handle for the combo box.

See Also

FListHandle (see page 2289)

FEditHandle (see page 2288)

30.1.8.4.4 TCustomCombo.FDroppingDown Field

Specifies whether the list box is dropping down.

Pascal

```
FDroppingDown: Boolean;
```

C++

```
Boolean FDroppingDown;
```

Description

FDroppingDown field is used when adjusting the drop-down list box size.

See Also

AdjustDropDown (see page 2292)

30.1.8.4.5 TCustomCombo.FEditHandle Field

Represents the window handle for the edit portion of the combo box.

Pascal

```
FEditHandle: HWnd;
```

C++

```
HWnd FEditHandle;
```

Description

FEditHandle is the window handle for the edit control portion of the combo box.

See Also

EditHandle (see page 2307)

30.1.8.4.6 TCustomCombo.FEditInstance Field

Represents the address of an internal routine associated with the edit control.

Pascal

```
FEditInstance: Pointer;
```

C++

```
void * FEditInstance;
```

Description

FEditInstance represents the address of a Windows message handling procedure associated with the edit control inside the combo box.

See Also

FListInstance (see page 2289)

30.1.8.4.7 TCustomCombo.FFocusChanged Field

Specifies whether the combo box changed its focus state.

Pascal

```
FFocusChanged: Boolean;
```

C++

```
Boolean FFocusChanged;
```

Description

FFocusChanged field specifies whether the control changed its focus state.

See Also

Focused (see page 2297)

30.1.8.4.8 TCustomCombo.FIsFocused Field

Specifies whether the combo box is focused.

Pascal

```
FIsFocused: Boolean;
```

C++

```
Boolean FIsFocused;
```

Description

FIsFocused field is set to True if the combo box is focused and to False otherwise.

See Also

Focused (see page 2297)

30.1.8.4.9 TCustomCombo.FListHandle Field

Represents the window handle for the list of items in the combo box.

Pascal

```
FListHandle: HWnd;
```

C++

```
HWnd FListHandle;
```

Description

FListHandle is the window handle for the drop-down list.

See Also

ListHandle (see page 2310)

30.1.8.4.10 TCustomCombo.FListInstance Field

Represents the address of an internal routine associated with the list box.

Pascal

```
FListInstance: Pointer;
```

C++

```
void * FListInstance;
```

Description

FListInstance represents the address of a Windows message handling procedure associated with the list box inside the combo box.

See Also

FEditInstance (see page 2288)

30.1.8.4.11 TCustomCombo.FSaveIndex Field

Represents the index of an item from the combo box.

Pascal

```
FSaveIndex: Integer;
```

C++

```
int FSaveIndex;
```

Description

FSaveIndex field is used to store the index of an item from the combo box.

See Also

ItemIndex

30.1.8.5 TCustomCombo Methods

The methods of the TCustomCombo class are listed here.

Protected Methods

	Name	Description
	AdjustDropDown (see page 2292)	Adjusts the size and position of the drop down list.
	Change (see page 2292)	Generates an OnChange (see page 2318) event.
	CloseUp (see page 2293)	Generates an OnCloseUp (see page 2320) event.
	ComboWndProc (see page 2294)	Responds to Windows messages sent to the edit control or drop-down list.
	CreateWnd (see page 2295)	Creates the Windows control that represents the combo box.
	DestroyWindowHandle (see page 2296)	Destroys all of the windows controls that implement the combo box.
	DropDown (see page 2296)	Generates an OnDropDown (see page 2320) event.
	EditWndProc (see page 2296)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2297)	Returns the value of the DroppedDown (see page 2306) property.
	GetItemCount (see page 2298)	Returns the value of the ItemCount (see page 2307) property.
	GetItemHt (see page 2298)	Returns the value of the ItemHeight (see page 2307) property.
	GetItemIndex (see page 2298)	Returns the value of the ItemIndex property.
	GetItemsClass (see page 2299)	Indicates the class that implements the Items (see page 2308) property.
	GetSelLength (see page 2299)	Returns the value of the SelLength (see page 2311) property.
	GetSelStart (see page 2299)	Returns the value of the SelStart (see page 2314) property.
	ListWndProc (see page 2300)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
	Loaded (see page 2300)	Initializes the combo box after its form is loaded into memory.
	Select (see page 2301)	Generates an OnSelect (see page 2322) event.
	SetDropDownCount (see page 2301)	Sets the value of the DropDownCount (see page 2306) property.
	SetDroppedDown (see page 2302)	Sets the value of the DroppedDown (see page 2306) property.
	SetItemHeight (see page 2302)	Sets the value of the ItemHeight (see page 2307) property.
	SetItemIndex (see page 2302)	Sets the value of the ItemIndex property.
	SetItems (see page 2303)	Sets the value of the Items (see page 2308) property.
	SetMaxLength (see page 2303)	Sets the value of the MaxLength (see page 2310) property.
	SetSelLength (see page 2303)	Sets the value of the SelLength (see page 2311) property.
	SetSelStart (see page 2304)	Sets the value of the SelStart (see page 2314) property.
	WndProc (see page 2304)	Processes messages received by the combo box.

Public Methods

	Name	Description
	AddItem (see page 2291)	Adds an item to the drop-down list of items.
	Clear (see page 2293)	Deletes all text from the edit box and all items from the Items (see page 2308) list.
	ClearSelection (see page 2293)	Deselects any selected items in the drop-down list.
	CopySelection (see page 2294)	Copies the selected item in the drop-down list to another list control.
	DeleteSelected (see page 2295)	Deletes the selected item from the drop-down list.

	Focused (see page 2297)	Indicates whether the combo box has input focus.
	GetCount (see page 2297)	Returns the value of the ItemCount (see page 2307) property.
	SelectAll (see page 2301)	Selects the entire block of text in the edit box.

30.1.8.5.1 TCustomCombo.AddItem Method

Adds an item to the drop-down list of items.

Pascal

```
procedure AddItem(Item: String; AObject: TObject); override;
```

C++

```
virtual __fastcall AddItem(UnicodeString Item, TObject * AObject);
```

Description

Call AddItem to add a string, with an associated object, to the drop-down list.

Item is the string to add to the drop-down list.

AObject is an object associated with that string. It can be accessed using the Objects property of the TStrings object that implements the Items (see page 2308) property.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
```

```
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

Items (see page 2308)

Objects

30.1.8.5.2 TCustomCombo.AdjustDropDown Method

Adjusts the size and position of the drop down list.

Pascal

```
procedure AdjustDropDown; virtual;
```

C++

```
virtual __fastcall AdjustDropDown();
```

Description

TCustomCombo (see page 2283) calls AdjustDropDown when it drops down the drop-down list. This method adjusts the size and position of the drop-down list to reflect the values of the ItemHeight (see page 2307) and DropDownCount (see page 2306) properties.

See Also

ItemHeight (see page 2307)

DroppedDown (see page 2306)

DropDownCount (see page 2306)

30.1.8.5.3 TCustomCombo.Change Method

Generates an OnChange (see page 2318) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Change is called automatically in response to notifications that the text has changed. It responds by generating an OnChange (see page 2318) event. Override this method to add additional processing when the text changes, or to block the OnChange (see page 2318) event.

See Also

OnChange (see page 2318)

Changed

30.1.8.5.4 TCustomCombo.Clear Method

Deletes all text from the edit box and all items from the Items (see page 2308) list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the combo box at runtime.

See Also

Items (see page 2308)

Text

30.1.8.5.5 TCustomCombo.ClearSelection Method

Deselects any selected items in the drop-down list.

Pascal

```
procedure ClearSelection; override;
```

C++

```
virtual __fastcall ClearSelection();
```

Description

ClearSelection checks whether an item in the drop-down list is currently selected. If so, it deselects the item (sets ItemIndex to –1), and changes the Text property to an empty string.

See Also

ItemIndex

Text

30.1.8.5.6 TCustomCombo.CloseUp Method

Generates an OnCloseUp (see page 2320) event.

Pascal

```
procedure CloseUp; dynamic;
```

C++

```
__fastcall CloseUp();
```

Description

CloseUp is called automatically when the DroppedDown (see page 2306) property changes to false. It generates an OnCloseUp (see page 2320) event.

Override CloseUp to implement additional class-specific responses when the drop-down list closes up or to suppress the event.

See Also

DroppedDown (see page 2306)

OnCloseUp (see page 2320)

30.1.8.5.7 TCustomCombo.ComboWndProc Method

Responds to Windows messages sent to the edit control or drop-down list.

Pascal

```
procedure ComboWndProc(var Message: TMessage; ComboWnd: HWnd; ComboProc: TWindowProcPtr);
virtual;
```

C++

```
virtual __fastcall ComboWndProc(TMessage Message, HWnd ComboWnd, TWindowProcPtr ComboProc);
```

Description

ComboWndProc handles the common processing of Windows messages sent to the edit region or list portion of the combo box. It is called by the EditWndProc (see page 2296) and ListWndProc (see page 2300) methods to take care of all processing that is not specific to the edit region or drop-down list.

The Message parameter describes the Windows message. The ComboWnd parameter is the windows handle for the edit region or list. The ComboProc parameter is a pointer to the default window procedure for the window that received the message. ComboWndProc provides default responses to focus events, mouse events, and key strokes.

Override ComboWndProc to change the way the combo box responds to Windows messages. When overriding ComboWndProc, be sure to call the parent class's method to process all unhandled messages.

Note: Do not confuse ComboWndProc with the WndProc (see page 2304) method. ComboWndProc responds to messages that are sent to the edit region or drop-down list. WndProc (see page 2304) responds to messages sent to the combo box itself.

See Also

WndProc (see page 2304)

EditWndProc (see page 2296)

ListWndProc (see page 2300)

30.1.8.5.8 TCustomCombo.CopySelection Method

Copies the selected item in the drop-down list to another list control.

Pascal

```
procedure CopySelection(Destination: TCustomListControl); override;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination);
```

Description

CopySelection checks whether an item is selected in the drop-down list, and if so, adds it to the control specified by Destination, along with its associated object.

See Also

Items (see page 2308)

ItemIndex

30.1.8.5.9 TCustomCombo.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo (see page 2283) overrides the inherited method to ensure that the newly-created window and the TCustomCombo (see page 2283) properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo (see page 2283) instance.

See Also

CreateHandle

30.1.8.5.10 TCustomCombo.DeleteSelected Method

Deletes the selected item from the drop-down list.

Pascal

```
procedure DeleteSelected; override;
```

C++

```
virtual __fastcall DeleteSelected();
```

Description

DeleteSelected deletes the item specified by the ItemIndex property from the drop-down list.

See Also

ItemIndex

Items (see page 2308)

30.1.8.5.11 TCustomCombo.DestroyWindowHandle Method

Destroys all of the windows controls that implement the combo box.

Pascal

```
procedure DestroyWindowHandle; override;
```

C++

```
virtual __fastcall DestroyWindowHandle();
```

Description

The DestroyWnd method calls DestroyWindowHandle to release the underlying Windows control. TCustomCombo (see page 2283) overrides the inherited method to reset the EditHandle (see page 2307) and ListHandle (see page 2310) properties once the underlying Windows controls are gone.

See Also

DestroyWnd

EditHandle (see page 2307)

ListHandle (see page 2310)

30.1.8.5.12 TCustomCombo.DropDown Method

Generates an OnDropDown (see page 2320) event.

Pascal

```
procedure DropDown; dynamic;
```

C++

```
__fastcall DropDown();
```

Description

DropDown is called automatically when the DroppedDown (see page 2306) property changes to true. It responds by generating an OnDropDown (see page 2320) event. To perform additional processing or block the OnDropDown (see page 2320) event, override DropDown.

See Also

OnDropDown (see page 2320)

DroppedDown (see page 2306)

30.1.8.5.13 TCustomCombo.EditWndProc Method

Responds to Windows messages that are sent to the edit control portion of the combo box.

Pascal

```
procedure EditWndProc(var Message: TMessage); virtual;
```

C++

```
virtual __fastcall EditWndProc(TMessage Message);
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. EditWndProc is the

window procedure for the edit control portion of the combo box.

See Also

EditHandle (see page 2307)

WndProc (see page 2304)

ListWndProc (see page 2300)

ComboWndProc (see page 2294)

30.1.8.5.14 TCustomCombo.Focused Method

Indicates whether the combo box has input focus.

Pascal

```
function Focused: Boolean; override;
```

C++

```
virtual __fastcall Boolean Focused();
```

Description

Call Focused to determine whether the combo box is currently the recipient of keyboard input. When Focused returns true, the combo box is the active control and keystroke messages are sent to it. When Focused is false, the combo box is not active (although it may be selected).

See Also

ActiveControl

30.1.8.5.15 TCustomCombo.GetCount Method

Returns the value of the ItemCount (see page 2307) property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount returns the number of items in the drop-down list.

See Also

ItemCount (see page 2307)

30.1.8.5.16 TCustomCombo.GetDroppedDown Method

Returns the value of the DroppedDown (see page 2306) property.

Pascal

```
function GetDroppedDown: Boolean;
```

C++

```
__fastcall Boolean GetDroppedDown();
```

Description

GetDroppedDown is the read implementation of the DroppedDown (see page 2306) property. It returns true if the drop-down list is showing.

See Also

DroppedDown (see page 2306)

30.1.8.5.17 TCustomCombo.GetItemCount Method

Returns the value of the ItemCount (see page 2307) property.

Pascal

```
function GetItemCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemCount() = 0;
```

Description

GetItemCount is the protected read access method for the ItemCount (see page 2307) property. In TCustomCombo (see page 2283), this method is abstract (pure virtual), meaning it has no implementation. Descendant classes override this method to return the number of items in the drop-down list.

See Also

ItemCount (see page 2307)

30.1.8.5.18 TCustomCombo.GetItemHt Method

Returns the value of the ItemHeight (see page 2307) property.

Pascal

```
function GetItemHt: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetItemHt() = 0;
```

Description

GetItemHt is the protected read access method for the ItemHeight (see page 2307) property. In TCustomCombo (see page 2283), this method is abstract (pure virtual), meaning it has no implementation. Descendant classes override this method to return the height of items in the drop-down list.

See Also

ItemHeight (see page 2307)

30.1.8.5.19 TCustomCombo.GetItemIndex Method

Returns the value of the ItemIndex property.

Pascal

```
function GetItemIndex: Integer; override;
```

C++

```
virtual __fastcall int GetItemIndex();
```

Description

GetItemIndex is the read implementation of the ItemIndex property. It returns the index of the selected item in the drop-down list. The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1.

See Also

ItemIndex

30.1.8.5.20 TCustomCombo.GetItemsClass Method

Indicates the class that implements the Items (see page 2308) property.

Pascal

```
function GetItemsClass: TCustomComboBoxStringsClass; virtual; abstract;
```

C++

```
virtual __fastcall TCustomComboBoxStringsClass GetItemsClass() = 0;
```

Description

TCustomCombo (see page 2283) uses GetItemsClass internally to determine what class to instantiate when it implements the Items (see page 2308) property.

In TCustomCombo (see page 2283), GetItemsClass is abstract pure virtual, meaning it has no implementation. TCustomCombo (see page 2283) descendants override this method to indicate a TCustomComboBoxStrings (see page 2339) descendant that implements the Items (see page 2308) property.

See Also

Items (see page 2308)

30.1.8.5.21 TCustomCombo.GetSelLength Method

Returns the value of the SelLength (see page 2311) property.

Pascal

```
function GetSelLength: Integer;
```

C++

```
__fastcall int GetSelLength();
```

Description

GetSelLength is the read implementation of the SelLength (see page 2311) property. It returns the number of selected characters.

See Also

SelLength (see page 2311)

30.1.8.5.22 TCustomCombo.GetSelStart Method

Returns the value of the SelStart (see page 2314) property.

Pascal

```
function GetSelStart: Integer;
```

C++

```
__fastcall int GetSelStart();
```

Description

GetSelStart is the read implementation of the SelStart (see page 2314) property. It returns the position of the first selected character, or the position of the cursor if no characters are selected.

See Also

SelStart (see page 2314)

30.1.8.5.23 TCustomCombo.ListWndProc Method

Responds to Windows messages that are sent to the drop-down list portion of the combo box.

Pascal

```
procedure ListWndProc(var Message: TMessage);
```

C++

```
__fastcall ListWndProc(TMessage Message);
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. ListWndProc is the window procedure for the drop-down list.

See Also

ListHandle (see page 2310)

WndProc (see page 2304)

EditWndProc (see page 2296)

ComboWndProc (see page 2294)

30.1.8.5.24 TCustomCombo.Loaded Method

Initializes the combo box after its form is loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The component streaming system calls this method after it loads the combo box's form from a stream.

TCustomCombo (see page 2283) overrides the inherited method to select the item specified by the ItemIndex property once the form is fully loaded.

See Also

Loaded

ItemIndex

30.1.8.5.25 TCustomCombo.Select Method

Generates an OnSelect (see page 2322) event.

Pascal

```
procedure Select; dynamic;
```

C++

```
__fastcall Select();
```

Description

Select is called automatically when the selected item in the drop-down list changes. It responds by generating an OnSelect (see page 2322) event. To perform additional processing or block the OnSelect (see page 2322) event, override Select.

See Also

OnSelect (see page 2322)

ItemIndex

30.1.8.5.26 TCustomCombo.SelectAll Method

Selects the entire block of text in the edit box.

Pascal

```
procedure SelectAll; override;
```

C++

```
virtual __fastcall SelectAll();
```

Description

Use SelectAll to select the entire block of text in the edit box. To select only part of the text, use the SelStart (see page 2314) and SelLength (see page 2311) properties.

See Also

SelStart (see page 2314)

SelLength (see page 2311)

30.1.8.5.27 TCustomCombo.SetDropDownCount Method

Sets the value of the DropDownCount (see page 2306) property.

Pascal

```
procedure SetDropDownCount(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetDropDownCount(const int Value);
```

Description

SetDropDownCount is the protected write implementation of the DropDownCount (see page 2306) property.

See Also

DropDownCount (see page 2306)

30.1.8.5.28 TCustomCombo.SetDroppedDown Method

Sets the value of the DroppedDown (see page 2306) property.

Pascal

```
procedure SetDroppedDown(Value: Boolean);
```

C++

```
__fastcall SetDroppedDown(Boolean Value);
```

Description

SetDroppedDown is the protected write implementation of the DroppedDown (see page 2306) property.

See Also

DroppedDown (see page 2306)

30.1.8.5.29 TCustomCombo.SetItemHeight Method

Sets the value of the ItemHeight (see page 2307) property.

Pascal

```
procedure SetItemHeight(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetItemHeight(int Value);
```

Description

SetItemHeight is the protected write implementation of the ItemHeight (see page 2307) property. Although SetItemHeight never raises an exception, in some cases it does not change the value of the ItemHeight (see page 2307) property. ItemHeight (see page 2307) can only be changed for some TCustomCombo (see page 2283) descendants, and then only under specific circumstances.

For example, TComboBoxEx never allows the ItemHeight (see page 2307) property to change.

See Also

ItemHeight (see page 2307)

30.1.8.5.30 TCustomCombo.SetItemIndex Method

Sets the value of the ItemIndex property.

Pascal

```
procedure SetItemIndex(const Value: Integer); override;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

SetItemIndex is the protected write implementation of the ItemIndex property. It selects an item in the drop-down list.

Value is the item to select, where 0 indicates the first item, 1 indicates the second item, and so on. A negative Value causes all items to be unselected.

See Also

ItemIndex

30.1.8.5.31 TCustomCombo.SetItems Method

Sets the value of the Items (see page 2308) property.

Pascal

```
procedure SetItems(const Value: TStrings); virtual;
```

C++

```
virtual __fastcall SetItems(const TStrings Value);
```

Description

SetItems is the protected write implementation for the Items (see page 2308) property. It copies the strings (and possibly their associated objects) from Value to the Items (see page 2308) property.

See Also

Items (see page 2308)

Assign

30.1.8.5.32 TCustomCombo.SetMaxLength Method

Sets the value of the MaxLength (see page 2310) property.

Pascal

```
procedure SetMaxLength(Value: Integer);
```

C++

```
__fastcall SetMaxLength(int Value);
```

Description

SetMaxLength is the protected write implementation of the MaxLength (see page 2310) property.

See Also

MaxLength (see page 2310)

30.1.8.5.33 TCustomCombo.SetSelLength Method

Sets the value of the SelLength (see page 2311) property.

Pascal

```
procedure SetSelLength(Value: Integer);
```

C++

```
__fastcall SetSelLength(int Value);
```

Description

SetSelLength is the protected write implementation of the SelLength (see page 2311) property.

See Also

SelLength (see page 2311)

30.1.8.5.34 TCustomCombo.SetSelStart Method

Sets the value of the SelStart (see page 2314) property.

Pascal

```
procedure SetSelStart(Value: Integer);
```

C++

```
__fastcall SetSelStart(int Value);
```

Description

SetSelStart is the protected write implementation of the SelStart (see page 2314) property.

See Also

SelStart (see page 2314)

30.1.8.5.35 TCustomCombo.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc (see page 2296) or ListWndProc (see page 2300) method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboWndProc (see page 2294)

EditWndProc (see page 2296)

ListWndProc (see page 2300)

30.1.8.6 TCustomCombo Properties

The properties of the TCustomCombo class are listed here.

Protected Properties

	Name	Description
	DropDownCount (see page 2306)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 2307)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 2307)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 2307)	Specifies the height, in pixels, of the items in the drop-down list.
R	ListHandle (see page 2310)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 2310)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.

	ParentColor (see page 2311)	Determines where a control looks for its color information.

Public Properties

	Name	Description
R	Canvas (see page 2305)	Provides access to the drawing surface of the combo box.
	DroppedDown (see page 2306)	Indicates whether the drop-down list is currently displayed.
	Items (see page 2308)	Provides access to the list of items (strings) in the list portion of the combo box.
	SelLength (see page 2311)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2314)	Specifies the position of the first selected character in the edit portion of the combo box.
	TabStop (see page 2317)	Determines if the user can tab to a control.

30.1.8.6.1 TCustomCombo.Canvas Property

Provides access to the drawing surface of the combo box.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Use Canvas as the drawing surface when customizing the way the combo box paints itself. For example, when using a TCustomCombo (see page 2283) descendant that allows applications to draw combo box items in an OnDrawItem even, the OnDrawItem event handler uses Canvas for painting the customized items.

C++ Examples:

```
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TComboBox *myComboBox;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TComboBox")
  {
    myComboBox = dynamic_cast<TComboBox *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myComboBox->Canvas->MoveTo(myComboBox->ClientRect.Left, myComboBox->ClientRect.Top);
    myComboBox->Canvas->LineTo(myComboBox->ClientRect.Right, myComboBox->ClientRect.Bottom);
    myComboBox = NULL;
  }
}
```

Delphi Examples:

```
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TComboBox) then
```

```
begin
  myComboBox:= (ListBox1.Items.Objects[Index] as TComboBox);
  with myComboBox.ClientRect do
  begin // make sure we are using the TControl's canvas and not the form's!
    myComboBox.Canvas.MoveTo(Left, Top);
    myComboBox.Canvas.LineTo(Right, Bottom);
  end;
myComboBox:= nil;
end;
```

See Also

OnDrawItem (see page 2338)

30.1.8.6.2 TCustomCombo.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

Pascal

```
property DropDownCount: Integer;
```

C++

```
__property int DropDownCount;
```

Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items (see page 2308) property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller then the number of item in the Items (see page 2308) property, the drop down list will display a scroll bar.

See Also

Items (see page 2308)

ItemCount (see page 2307)

30.1.8.6.3 TCustomCombo.DroppedDown Property

Indicates whether the drop-down list is currently displayed.

Pascal

```
property DroppedDown: Boolean;
```

C++

```
__property Boolean DroppedDown;
```

Description

Read (see page 3634) DroppedDown to determine whether the drop-down list is visible. Set DropDown (see page 2296) to drop the list down or close it up. DroppedDown is true when the drop-down list is open, and false when it is closed.

See Also

OnDropDown (see page 2320)

AutoDropDown (see page 2334)

30.1.8.6.4 TCustomCombo.EditHandle Property

Returns the window handle for the edit portion of the combo box.

Pascal

```
property EditHandle: HWnd;
```

C++

```
__property HWnd EditHandle;
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. EditHandle is the window handle of the edit control portion. (The Handle property and the ListHandle (see page 2310) property give the handles for the combo box itself and the drop-down list, respectively.)

Use EditHandle when making Windows API calls addressed to the edit portion of the combo box.

See Also

Handle

ListHandle (see page 2310)

30.1.8.6.5 TCustomCombo.ItemCount Property

Specifies the number of items in the drop-down list.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Read (see page 3634) ItemCount to get the number of items in the drop-down list. ItemCount is the number of all items, not just those that are visible at one time.

See Also

Items (see page 2308)

DropDownCount (see page 2306)

30.1.8.6.6 TCustomCombo.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Read (see page 3634) ItemHeight to determine the height of the items in the drop-down list.

The effect of setting ItemHeight varies, depending on the type of combo box and the style. For example, TComboBox (see

page 2184) lets you set the item height only when Style is set to csOwnerDrawFixed or csOwnerDrawVariable. If Style is csOwnerDrawVariable, ItemHeight is the default height for drawing list items, but this value can be overridden by an OnMeasureItem event handler.

TComboBoxEx does not let you change the value of ItemHeight.

Note: Combo boxes that do not support changes to ItemHeight do not raise an exception when you attempt to change ItemHeight; they simply ignore the attempt.

See Also

Items (see page 2308)

OnMeasureItem (see page 2338)

Style (see page 2337)

30.1.8.6.7 TCustomCombo.Items Property

Provides access to the list of items (strings) in the list portion of the combo box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Read (see page 3634) Items to access the list of items that appears in the combo box. Use the methods of Items to add, insert, delete and move items. Set the value of Items to copy the items from another string list.

C++ Examples:

```
/*
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {"C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {"Documentation for the C++ icon.", "Borland icon.", "Delphi
icon."};
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
```

```
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  // Add View styles and constants to the Combo Box
  ComboBox1->Items->AddObject("vsIcon", reinterpret_cast<TObject *>(vsIcon));
  ComboBox1->Items->AddObject("vsList", reinterpret_cast<TObject *>(vsList));
  ComboBox1->Items->AddObject("vsReport", reinterpret_cast<TObject *>(vsReport));
  ComboBox1->Items->AddObject("vsSmallIcon", reinterpret_cast<TObject *>(vsSmallIcon));
  // Display first item in the Combo Box
  ComboBox1->ItemIndex = 0;
}

void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
  ListView1->ViewStyle = (TViewStyle) ComboBox1->Items->Objects[ComboBox1->ItemIndex];
}
```

Delphi Examples:

```
{
This example requires a TListView, a TImageList and a
TComboBox. You will need to double click the image list and
add several images to the image list prior to running the
project. You can use *.bmp or *.ico files from the
\Images\Icons directory. During the form's OnCreate event
handler, items for the List View control are created for
each image in the Image List and the ImageIndex is assigned
the number of the image within the ImageList. Two columns are
created so that when ViewStyle is vsReport, you will have
columns to view.  Also within the form's OnCreate event
handler assign the ComboBox each of the four TViewStyle
constants to the Items' Objects property. You could also
simply code this within a series of OnClick event handlers
as, for instance, ListView1.->ViewStyle := vsIcon.
}

procedure TComboForm.FormCreate(Sender: TObject);
var
  I: Integer;
  ListItem: TListItem;
  NewColumn: TListColumn;
begin
  Col2Array[0]:= 'Documentation for Image0.';
  Col2Array[1]:= 'Documentation for Image1.';
  Col2Array[2]:= 'Documentation for Image2.';
  // Create a ListView item for each image in the ImageList
  with ListView1 do
  begin
    SmallImages := ImageList1;
    LargeImages := ImageList1;
    for I := 0 to ImageList1.Count - 1 do
    begin
      ListItem := Items.Add;
      Listitem.Caption := 'Image' + IntToStr(I);
      ListItem.ImageIndex := I;
      Listitem.SubItems.Add(Col2Array[I]);
    end;
    // Create two columns to show during viewing as vsReport
    NewColumn := Columns.Add;
    NewColumn.Caption := 'Column 1';
    NewColumn.Width := 200;
```

```
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      // Add View styles and constants to the Combo Box
      ComboBox1.Items.AddObject('vsIcon', TObject(vsIcon));
      ComboBox1.Items.AddObject('vsList', TObject(vsList));
      ComboBox1.Items.AddObject('vsReport', TObject(vsReport));
      ComboBox1.Items.AddObject(
        'vsSmallIcon', TObject(vsSmallIcon));
      // Display first item in the Combo Box
      ComboBox1.ItemIndex := 0;
    end;
  end;

  procedure TComboForm.ComboBox1Click(Sender: TObject);
  begin
    with ComboBox1 do
      ListView1.ViewStyle := TViewStyle(Items.Objects[ItemIndex]);

  end;
```

See Also

ItemIndex

TStrings

Assign

30.1.8.6.8 TCustomCombo.ListHandle Property

Provides access to the window handle of the list of items in the combo box.

Pascal

```
property ListHandle: HWnd;
```

C++

```
__property HWnd ListHandle;
```

Description

A combo box is composed of several windows: the combo box itself, the edit control, and the drop-down list. ListHandle is the window handle of the drop-down list. (The Handle property and the EditHandle (see page 2307) property give the handles for the combo box itself and the edit portion, respectively.)

Use ListHandle when making Windows API calls addressed to the list portion of the combo box.

Note: ListHandle is only available when the combo box has a "simple" style (an edit box with a fixed list below rather than a drop-down list).

See Also

Handle

EditHandle (see page 2307)

30.1.8.6.9 TCustomCombo.MaxLength Property

Specifies the maximum number of characters the user can type into the edit portion of the combo box.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that the user can enter into the edit portion of the combo box. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

See Also

Text

30.1.8.6.10 TCustomCombo.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.8.6.11 TCustomCombo.SelLength Property

Specifies the length, in characters, of the selected text in the edit portion of the combo box.

Pascal

```
property SelLength: Integer;
```

C++

```
__property int SelLength;
```

Description

The selected portion of the text in the edit region of the combo box is the string of SelLength characters that starts at the character specified by the SelStart (see page 2314) property. The selected text appears highlighted in the edit region of the combo box.

To select a section of text in the edit region, first set the SelStart (see page 2314) property to position the cursor, and then set SelLength to the number of characters that should be selected. The SelText property is the substring that is selected.

Note: The SelLength property is only valid when the combo box has focus. When the combo box does not have focus, reading SelLength always yields a value of 0, and setting it has no effect.

C++ Examples:

```
/*
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
```

```
      Found = true;
      break;
    }
  if (Found) // select the untyped end of the string
  {
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}

TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(String S)
{
  CharSetList->Add(S);
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<TStringList> _CharSetListCleaner(CharSetList = new TStringList);
  GetCharsetValues(AddCharacterSet);
  CharSetList->Sort();
  ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
```

```
      begin
        Text:= '';
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

SelStart (see page 2314)

SelText (see page 2335)

Text

30.1.8.6.12 TCustomCombo.SelStart Property

Specifies the position of the first selected character in the edit portion of the combo box.

Pascal

```
property SelStart: Integer;
```

C++

```
__property int SelStart;
```

Description

Read (see page 3634) SelStart to determine where the selection starts in the Text property. The first character is designated with 0, the second with 1, and so on. When SelLength (see page 2311) is 0, SelStart is the position of the cursor.

Set SelStart to position the cursor in the edit region of the combo box. Setting SelStart to 0 positions the cursor before the first

character, setting SelStart to 1 positions the cursor before the second character, and so on.

Note: Setting SelStart changes the SelLength (see page 2311) property to 0. When using SelStart and SelLength (see page 2311) to programmatically select a portion of the text in the edit region, set SelStart before setting SelLength (see page 2311).

Note: The SelStart property is only valid when the combo box has focus. When the combo box does not have focus, reading SelStart always yields a value of 0, and setting it has no effect.

C++ Examples:

```
/*
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
*/
void __fastcall TForm1::ComboBox1KeyPress(TObject *Sender, char &Key)
{
  // first, process the keystroke to obtain the current string
  // This code requires all items in list to be uppercase
  if (Key >= 'a' && Key <= 'z')
    Key -= 32; // Force Uppercase only!
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  AnsiString TmpStr;
  bool BackSpace = (Key == (char)VK_BACK);
  if (BackSpace && pCB->SelLength)
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  else if (BackSpace) // SelLength == 0
    TmpStr = pCB->Text.SubString(1,pCB->SelStart-1)+
             pCB->Text.SubString(pCB->SelStart+1,255);
  else //Key is a visible character
    TmpStr = pCB->Text.SubString(1,pCB->SelStart)+ Key +
             pCB->Text.SubString(pCB->SelLength+pCB->SelStart+1,255);
  if (TmpStr.IsEmpty())
    return;
  // set SelSt to the current insertion point
  int SelSt = pCB->SelStart;
  if (BackSpace && SelSt > 0)
    SelSt--;
  else if (!BackSpace)
    SelSt++;
  Key = 0; // indicate that key was handled
  if (SelSt == 0)
  {
    pCB->Text = "";
    return;
  }
  // Now that TmpStr is the currently typed string, see if we can locate a match
  bool Found = false;
  for (int i = 1; i < pCB->Items->Count; i++)
    if (TmpStr == pCB->Items->Strings[i-1].SubString(1,TmpStr.Length()))
    {
      pCB->Text = pCB->Items->Strings[i-1]; // update to the match that was found
      pCB->ItemIndex = i-1;
      Found = true;
      break;
```

```
    }
  if (Found) // select the untyped end of the string
  {
    pCB->SelStart = SelSt;
    pCB->SelLength = pCB->Text.Length() - SelSt;
  }
  else Beep();
}

TStringList *CharSetList;

void __fastcall TForm1::AddCharacterSet(String S)
{
  CharSetList->Add(S);
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<TStringList> _CharSetListCleaner(CharSetList = new TStringList);
  GetCharsetValues(AddCharacterSet);
  CharSetList->Sort();
  ComboBox1->Items = CharSetList;
}
```

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
```

```
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

SelLength (see page 2311)

SelText (see page 2335)

Text

30.1.8.6.13 TCustomCombo.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.8.7 TCustomCombo Events

The events of the TCustomCombo class are listed here.

Protected Events

	Name	Description
	OnChange (see page 2318)	Occurs when the user changes the text displayed in the edit region.
	OnCloseUp (see page 2320)	Occurs when the drop-down list closes up due to some user action.
	OnDropDown (see page 2320)	Occurs when the user opens the drop-down list.
	OnSelect (see page 2322)	Occurs when the user selects a string in the drop-down list.

30.1.8.7.1 TCustomCombo.OnChange Event

Occurs when the user changes the text displayed in the edit region.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3707) an OnChange event handler to take specific action immediately after the user edits the text in the edit region or selects an item from the list. The Text property gives the new value in the edit region.

Note: OnChange only occurs in response to user actions. Changing the Text property programmatically does not trigger an OnChange event.

C++ Examples:

```
/*
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
*/
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  TComboBox *pCB = dynamic_cast<TComboBox *>(Sender);
  if (pCB)
    PageControl1->ActivePage = dynamic_cast<TTabSheet *>(pCB->Items->Objects[pCB->ItemIndex]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  const TColor colorPalette[12] = {
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
    clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  for (int i = 0; i < PageControl1->PageCount; i++)
  {
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
    ComboBox1->Items->AddObject(PageControl1->Pages[i]->Name, PageControl1->Pages[i]);
  }
  ComboBox1->ItemIndex = 0;
  PageControl1->ActivePage = dynamic_cast<TTabSheet
*>(ComboBox1->Items->Objects[ComboBox1->ItemIndex]);
}
/*
The following example assumes that a drive combo box, a file
list box, and a directory list box are on a form. Add this
code as the OnChange event handler for the drive combo box
and the OnChange event handler for the directory list box.
When the user changes the drive using the combo box, the
directory list box and file list box will update to reflect
the new drive and the current directory on that drive.  When
the user double clicks on a directory in the file list box
will update to reflect the new directory.
*/
void __fastcall TForm1::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
}

void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
}
```

Delphi Examples:

```
{
This example requires a TPageControl already on the form.
Also, you must add pages to the TPageControl by right
clicking and selecting New Page.
The example code will allow the user to select the
ActivePage property through selection of a ComboBox item.
During the form create, the Combo Box control is loaded with
the names of each of the tabs, as well as the instance
pointers to the corresponding tab.  When the user selects
the Combo Box item, the associated TTabSheet object
contained in the Combo Box Objects array is used to set the
ActivePage property.  Select the Lines property of the
TRichEdit to enter a string.  Select the string before
justifying.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
const
  colorarray : Array[0..4] of TColor = (
    clYellow, clGreen, clBlue, clLime, clFuchsia);
begin
  for i := 0 to PageControl1.PageCount - 1 do
    begin
    PageControl1.Pages[i].Brush.Color := colorarray[i];
    ComboBox1.Items.AddObject(PageControl1.Pages[i].Name,
      PageControl1.Pages[i]);
    end;
  ComboBox1.ItemIndex := 0;
```

```
    PageControl1.ActivePage := TTabSheet(ComboBox1.Items.Objects[ComboBox1.ItemIndex]);
  end;

  procedure TForm1.ComboBox1Change(Sender: TObject);
  begin
    if (Sender is TComboBox) then
      with (Sender as TComboBox) do
        PageControl1.ActivePage := TTabSheet(Items.Objects[ItemIndex]);
  end;
  {
  The following example assumes that a drive combo box, a file
  list box, and a directory list box are on a form. Add this
  code as the OnChange event handler for the drive combo box
  and the OnChange event handler for the directory list box.
  When the user changes the drive using the combo box, the
  directory list box and file list box will update to reflect
  the new drive and the current directory on that drive.  When
  the user double clicks on a directory in the file list box
  will update to reflect the new directory.
  }
  procedure TForm1.DirectoryListBox1Change(Sender: TObject);
  begin
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;

  procedure TForm1.DriveComboBox1Change(Sender: TObject);
  begin
    DirectoryListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Drive := DriveComboBox1.Drive;
    FileListBox1.Directory := DirectoryListBox1.Directory;
  end;
```

See Also

Text

OnCloseUp (see page 2320)

30.1.8.7.2 TCustomCombo.OnCloseUp Event

Occurs when the drop-down list closes up due to some user action.

Pascal

```
property OnCloseUp: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnCloseUp;
```

Description

Write (see page 3707) an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

See Also

OnDropDown (see page 2320)

DroppedDown (see page 2306)

30.1.8.7.3 TCustomCombo.OnDropDown Event

Occurs when the user opens the drop-down list.

Pascal

```
property OnDropDown: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDropDown;
```

Description

Write (see page 3707) an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
```

```
      begin
        if (Screen.CustomForms[I].Visible) then
          Items.Add(Screen.CustomForms[I].Caption); { add form name }
      end;
      Items.EndUpdate; {reenable painting }
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ComboBox1.Sorted := True; { make sure the form names are sorted }
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Form2.Show;
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    Form3.Show;
  end;
```

See Also

DroppedDown (see page 2306)

OnCloseUp (see page 2320)

30.1.8.7.4 TCustomCombo.OnSelect Event

Occurs when the user selects a string in the drop-down list.

Pascal

```
property OnSelect: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnSelect;
```

Description

Write (see page 3707) an OnSelect event handler to respond when the user changes the selected item in the drop-down list. OnSelect occurs after the selection has changed and the Text property reflects the new item.

See Also

AutoComplete (see page 2332)

ItemIndex

Text

30.1.9 TCustomComboBox Class

TCustomComboBox is the base class for combobox components.

Class Hierarchy

Pascal

```
TCustomComboBox = class(TCustomCombo);
```

C++

```
class TCustomComboBox : public TCustomCombo;
```

File

StdCtrls

Description

Use TCustomComboBox as a base class when defining a control that represents a specialized Windows combo box. A combo box is an edit box with a scrollable list attached to it; users can select an item from the list or type directly into the edit box. Combo boxes can present the list as a drop-down list or as a list box that is always visible.

Note: To create a combo box class that uses the extended combo box features such as images associated with entries, do not use TCustomComboBox. Instead, use the TCustomComboBoxEx class instead.

Do not create instances of TCustomComboBox. To put a combo box in a form, use a descendant of TCustomComboBox, such as TComboBox (see page 2184).

See Also

TComboBox (see page 2184)

TDBComboBox

TDriveComboBox

TEdit (see page 2463)

TFilterComboBox

TListBox (see page 2657)

TColorBox

TCustomComboBoxEx

30.1.9.1 TCustomComboBox Members

The following tables list the members exposed by TCustomComboBox.

Protected Events

	Name	Description
	OnChange (see page 2318)	Occurs when the user changes the text displayed in the edit region.
	OnCloseUp (see page 2320)	Occurs when the drop-down list closes up due to some user action.
	OnDrawItem (see page 2338)	Occurs when an item in an owner-draw combo box needs to be displayed.
	OnDropDown (see page 2320)	Occurs when the user opens the drop-down list.
	OnMeasureItem (see page 2338)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
	OnSelect (see page 2322)	Occurs when the user selects a string in the drop-down list.

Protected Fields

	Name	Description
	FDefEditProc (see page 2286)	Represents the address of the edit box procedure assigned by Windows.
	FDefListProc (see page 2287)	Represents the address of the list box procedure assigned by Windows.
	FDropHandle (see page 2287)	Represents the window handle for the entire combo box.
	FDroppingDown (see page 2287)	Specifies whether the list box is dropping down.

	FEditHandle (see page 2288)	Represents the window handle for the edit portion of the combo box.
	FEditInstance (see page 2288)	Represents the address of an internal routine associated with the edit control.
	FFocusChanged (see page 2288)	Specifies whether the combo box changed its focus state.
	FIsFocused (see page 2288)	Specifies whether the combo box is focused.
	FListHandle (see page 2289)	Represents the window handle for the list of items in the combo box.
	FListInstance (see page 2289)	Represents the address of an internal routine associated with the list box.
	FSaveIndex (see page 2289)	Represents the index of an item from the combo box.

Protected Methods

	Name	Description
V	AdjustDropDown (see page 2292)	Adjusts the size and position of the drop down list.
	Change (see page 2292)	Generates an OnChange (see page 2318) event.
	CloseUp (see page 2293)	Generates an OnCloseUp (see page 2320) event.
V	ComboWndProc (see page 2294)	Responds to Windows messages sent to the edit control or drop-down list.
V	CreateParams (see page 2326)	Initializes the window-creation parameter record.
V	CreateWnd (see page 2327)	Creates the Windows control that represents the combo box.
V	DestroyWindowHandle (see page 2296)	Destroys all of the windows controls that implement the combo box.
V	DestroyWnd (see page 2327)	Destroys the Windows control that represents the combo box.
V	DoSetTextHint (see page 2328)	Sets the value of TextHint (see page 2337).
V	DrawItem (see page 2328)	Generates an OnDrawItem (see page 2338) event.
	DropDown (see page 2296)	Generates an OnDropDown (see page 2320) event.
V	EditWndProc (see page 2296)	Responds to Windows messages that are sent to the edit control portion of the combo box.
	GetDroppedDown (see page 2297)	Returns the value of the DroppedDown (see page 2306) property.
V	GetItemCount (see page 2328)	Returns the number of items in the Items (see page 2308) property list.
V	GetItemHt (see page 2329)	Returns the value of the ItemHeight (see page 2307) property.
V	GetItemIndex (see page 2298)	Returns the value of the ItemIndex property.
V	GetItemsClass (see page 2329)	Returns the class used to implement the Items (see page 2308) property.
	GetSelLength (see page 2299)	Returns the value of the SelLength (see page 2311) property.
	GetSelStart (see page 2299)	Returns the value of the SelStart (see page 2314) property.
	GetSelText (see page 2329)	Returns the value of the SelText (see page 2335) property.
V	KeyPress (see page 2329)	Responds to WM_CHAR messages sent to the combo box.
	ListWndProc (see page 2300)	Responds to Windows messages that are sent to the drop-down list portion of the combo box.
V	Loaded (see page 2300)	Initializes the combo box after its form is loaded into memory.
V	MeasureItem (see page 2330)	Generates an OnMeasureItem (see page 2338) event.
	Select (see page 2301)	Generates an OnSelect (see page 2322) event.
	SelectItem (see page 2330)	Selects a specified item from the drop-down list.
V	SetDropDownCount (see page 2301)	Sets the value of the DropDownCount (see page 2306) property.
	SetDroppedDown (see page 2302)	Sets the value of the DroppedDown (see page 2306) property.
V	SetItemHeight (see page 2302)	Sets the value of the ItemHeight (see page 2307) property.
V	SetItemIndex (see page 2302)	Sets the value of the ItemIndex property.
V	SetItems (see page 2303)	Sets the value of the Items (see page 2308) property.
	SetMaxLength (see page 2303)	Sets the value of the MaxLength (see page 2310) property.
	SetSelLength (see page 2303)	Sets the value of the SelLength (see page 2311) property.
	SetSelStart (see page 2304)	Sets the value of the SelStart (see page 2314) property.
V	SetStyle (see page 2331)	Sets the Style (see page 2337) property.
V	WndProc (see page 2331)	Processes messages received by the combo box.

Protected Properties

	Name	Description
	DropDownCount (see page 2306)	Specifies the maximum number of items displayed in the drop-down list.
R	EditHandle (see page 2307)	Returns the window handle for the edit portion of the combo box.
R	ItemCount (see page 2307)	Specifies the number of items in the drop-down list.
	ItemHeight (see page 2307)	Specifies the height, in pixels, of the items in the drop-down list.

R	ListHandle (see page 2310)	Provides access to the window handle of the list of items in the combo box.
	MaxLength (see page 2310)	Specifies the maximum number of characters the user can type into the edit portion of the combo box.
	ParentColor (see page 2311)	Determines where a control looks for its color information.
	Sorted (see page 2335)	Determines whether the list portion of the combo box is alphabetized.
	Style (see page 2337)	Determines the display style of the combo box.

Public Methods

	Name	Description
V	AddItem (see page 2291)	Adds an item to the drop-down list of items.
V	Clear (see page 2293)	Deletes all text from the edit box and all items from the Items (see page 2308) list.
V	ClearSelection (see page 2293)	Deselects any selected items in the drop-down list.
V	CopySelection (see page 2294)	Copies the selected item in the drop-down list to another list control.
V	Create (see page 2325)	Creates and initializes a TCustomComboBox (see page 2322) instance.
V	DeleteSelected (see page 2295)	Deletes the selected item from the drop-down list.
V	Destroy (see page 2326)	Destroys the combo box and frees its memory.
V	Focused (see page 2297)	Indicates whether the combo box has input focus.
V	GetCount (see page 2297)	Returns the value of the ItemCount (see page 2307) property.
V	SelectAll (see page 2301)	Selects the entire block of text in the edit box.

Public Properties

	Name	Description
	AutoCloseUp (see page 2332)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2332)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2333)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2334)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
R	Canvas (see page 2305)	Provides access to the drawing surface of the combo box.
	CharCase (see page 2334)	Determines the case of the text in the combo box.
	DroppedDown (see page 2306)	Indicates whether the drop-down list is currently displayed.
	Items (see page 2308)	Provides access to the list of items (strings) in the list portion of the combo box.
	SelLength (see page 2311)	Specifies the length, in characters, of the selected text in the edit portion of the combo box.
	SelStart (see page 2314)	Specifies the position of the first selected character in the edit portion of the combo box.
	SelText (see page 2335)	Represents the selected text in the edit region.
	TabStop (see page 2317)	Determines if the user can tab to a control.
	TextHint (see page 2337)	Specifies the text that is displayed as a text watermark in the edit box of the combo box control.

30.1.9.2 TCustomComboBox.Create Constructor

Creates and initializes a TCustomComboBox (see page 2322) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomComboBox(TComponent * AOwner);
```

Description

Call Create to generate a combo box at runtime. Combo boxes placed on forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the combo box instance.

See Also

Owner

30.1.9.3 TCustomComboBox.Destroy Destructor

Destroys the combo box and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomComboBox();
```

Description

Do not call Destroy directly in an application. Call Free instead. Free verifies that the combo box is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

30.1.9.4 TCustomComboBox Methods

The methods of the TCustomComboBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2326)	Initializes the window-creation parameter record.
	CreateWnd (see page 2327)	Creates the Windows control that represents the combo box.
	DestroyWnd (see page 2327)	Destroys the Windows control that represents the combo box.
	DoSetTextHint (see page 2328)	Sets the value of TextHint (see page 2337).
	DrawItem (see page 2328)	Generates an OnDrawItem (see page 2338) event.
	GetItemCount (see page 2328)	Returns the number of items in the Items (see page 2308) property list.
	GetItemHt (see page 2329)	Returns the value of the ItemHeight (see page 2307) property.
	GetItemsClass (see page 2329)	Returns the class used to implement the Items (see page 2308) property.
	GetSelText (see page 2329)	Returns the value of the SelText (see page 2335) property.
	KeyPress (see page 2329)	Responds to WM_CHAR messages sent to the combo box.
	MeasureItem (see page 2330)	Generates an OnMeasureItem (see page 2338) event.
	SelectItem (see page 2330)	Selects a specified item from the drop-down list.
	SetStyle (see page 2331)	Sets the Style (see page 2337) property.
	WndProc (see page 2331)	Processes messages received by the combo box.

30.1.9.4.1 TCustomComboBox.CreateParams Method

Initializes the window-creation parameter record.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2327) method calls CreateParams to initialize the parameters it uses to create the window for the combo box. CreateParams overrides the inherited method to implement the Style (see page 2337) property.

See Also

CreateWnd (see page 2327)

Style (see page 2337)

30.1.9.4.2 TCustomComboBox.CreateWnd Method

Creates the Windows control that represents the combo box.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

The CreateHandle method calls CreateWnd to handle the particulars of creating the combo box window. TCustomCombo overrides the inherited method to ensure that the newly-created window and the TCustomCombo properties correctly reflect each other.

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. Thus, CreateWnd may be called more than once during the lifetime of the TCustomCombo instance.

See Also

CreateHandle

30.1.9.4.3 TCustomComboBox.DestroyWnd Method

Destroys the Windows control that represents the combo box.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

The Windows control for the combo box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. TCustomComboBox (see page 2322) calls DestroyWnd to destroy this Windows control when necessary.

DestroyWnd saves the contents of Items (see page 2308) list before calling the inherited DestroyWnd method. The saved list is used if the window is recreated.

See Also

CreateWnd (see page 2327)

30.1.9.4.4 TCustomComboBox.DoSetTextHint Method

Sets the value of TextHint (see page 2337).

Pascal

```
procedure DoSetTextHint; virtual;
```

C++

```
virtual __fastcall DoSetTextHint();
```

Description

DoSetTextHint is a protected method used to set the value of the TextHint (see page 2337) property.

See Also

TextHint (see page 2380)

30.1.9.4.5 TCustomComboBox.DrawItem Method

Generates an OnDrawItem (see page 2338) event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); virtual;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

When Style (see page 2337) is csOwnerDrawFixed or csOwnerDrawVariable, DrawItem is called whenever an item in the list needs to be drawn. If an OnDrawItem (see page 2338) event handler is assigned, DrawItem generates an OnDrawItem (see page 2338) event, passing the Index, Rect, and State parameters to the event handler. Otherwise, OnDrawItem (see page 2338) fills the rectangle passed in Rect and draws any text associated with the indexed item, ignoring the State parameter.

Override DrawItem to provide class-specific drawing code for the items in the list.

See Also

OnDrawItem (see page 2338)

Style (see page 2337)

30.1.9.4.6 TCustomComboBox.GetItemCount Method

Returns the number of items in the Items (see page 2308) property list.

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

GetItemCount is used internally to quickly determine the number of items in the drop-down list of the combo box.

See Also

Items (see page 2308)

30.1.9.4.7 TCustomComboBox.GetItemHt Method

Returns the value of the ItemHeight (see page 2307) property.

Pascal

```
function GetItemHt: Integer; override;
```

C++

```
virtual __fastcall int GetItemHt();
```

Description

GetItemHt is the protected read implementation of the ItemHeight (see page 2307) property.

See Also

ItemHeight (see page 2307)

30.1.9.4.8 TCustomComboBox.GetItemsClass Method

Returns the class used to implement the Items (see page 2308) property.

Pascal

```
function GetItemsClass: TCustomComboBoxStringsClass; override;
```

C++

```
virtual __fastcall TCustomComboBoxStringsClass GetItemsClass();
```

Description

Combo box controls use GetItemsClass internally to determine what class to use when implementing the Items (see page 2308) property. This allows the TCustomCombo (see page 2283) base class to implement the Items (see page 2308) property for both TCustomComboBox (see page 2322) descendants and TCustomComboBoxEx descendants.

See Also

Items (see page 2308)

30.1.9.4.9 TCustomComboBox.GetSelText Method

Returns the value of the SelText (see page 2335) property.

Pascal

```
function GetSelText: string;
```

C++

```
__fastcall UnicodeString GetSelText();
```

Description

GetSelText is the protected read implementation of the SelText (see page 2335) property.

See Also

SelText (see page 2335)

30.1.9.4.10 TCustomComboBox.KeyPress Method

Responds to WM_CHAR messages sent to the combo box.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress processes standard keystroke messages for the combo box control. If the KeyPress method does not handle the keystroke, it calls the inherited method, which generates an OnKeyPress event.

Key is the character represented by the key that was pressed. If the KeyPress method changes Key to 0, no further processing occurs for the keystroke.

See Also

KeyDown

KeyUp

OnKeyPress

30.1.9.4.11 TCustomComboBox.MeasureItem Method

Generates an OnMeasureItem (see page 2338) event.

Pascal

```
procedure MeasureItem(Index: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(int Index, int Height);
```

Description

MeasureItem is the protected implementation method for the OnMeasureItem (see page 2338) event. Override MeasureItem to provide class-specific code to specify the size of list items when Style (see page 2337) is csOwnerDrawVariable.

See Also

OnMeasureItem (see page 2338)

Style (see page 2337)

30.1.9.4.12 TCustomComboBox.SelectItem Method

Selects a specified item from the drop-down list.

Pascal

```
function SelectItem(const AnItem: string): Boolean;
```

C++

```
__fastcall Boolean SelectItem(const UnicodeString AnItem);
```

Description

SelectItem selects the first item in the drop-down list that matches the string specified by AnItem. If the list does not contain a string that matches AnItem, SelectItem does nothing.

See Also

Items (see page 2308)

ItemIndex

30.1.9.4.13 TCustomComboBox.SetStyle Method

Sets the Style (see page 2337) property.

Pascal

```
procedure SetStyle(Value: TComboBoxStyle); virtual;
```

C++

```
virtual __fastcall SetStyle(TComboBoxStyle Value);
```

Description

SetStyle is the protected implementation method for the Style (see page 2337) property. Override SetStyle to change the implementation of the Style (see page 2337) property.

See Also

Style (see page 2337)

30.1.9.4.14 TCustomComboBox.WndProc Method

Processes messages received by the combo box.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc is the first method to received Windows messages that are sent to the combo box. Messages sent to the child windows of the combo box (for the edit region and the list portion) go to the EditWndProc (see page 2296) or ListWndProc (see page 2300) method instead. WndProc handles messages about changes to the system colors defined in the Windows control panel, and closes up the drop-down list when the Enter key or the Escape key is pressed. All other messages are passed to Dispatch for default processing.

See Also

ComboWndProc (see page 2294)

System::TObject::Dispatch (see page 3369)

WndProc

EditWndProc (see page 2296)

ListWndProc (see page 2300)

30.1.9.5 TCustomComboBox Properties

The properties of the TCustomComboBox class are listed here.

Protected Properties

	Name	Description
	Sorted (see page 2335)	Determines whether the list portion of the combo box is alphabetized.

	Style (see page 2337)	Determines the display style of the combo box.

Public Properties

	Name	Description
	AutoCloseUp (see page 2332)	Specifies whether the drop-down closes up automatically when the user selects an item.
	AutoComplete (see page 2332)	Positions to matching list items as you type.
	AutoCompleteDelay (see page 2333)	Specifies the delay between a key press and an attempt to autocomplete the field.
	AutoDropDown (see page 2334)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
	CharCase (see page 2334)	Determines the case of the text in the combo box.
	SelText (see page 2335)	Represents the selected text in the edit region.
	TextHint (see page 2337)	Specifies the text that is displayed as a text watermark in the edit box of the combo box control.

30.1.9.5.1 TCustomComboBox.AutoCloseUp Property

Specifies whether the drop-down closes up automatically when the user selects an item.

Pascal

```
property AutoCloseUp: Boolean;
```

C++

```
__property Boolean AutoCloseUp;
```

Description

When AutoCloseUp is true, the drop-down closes up automatically when the user selects an item.

See Also

AutoDropDown (see page 2334)

30.1.9.5.2 TCustomComboBox.AutoComplete Property

Positions to matching list items as you type.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

AutoComplete will position to matching list values as you type, saving you keystrokes.

As you type, the ItemIndex property is updated with the list entry that matches the characters you have entered. As you type, different list entries may match, and ItemIndex changes accordingly.

There is a delay of AutoCompleteDelay (see page 2333) milliseconds between the characters typed and the autocomplete operation.

C++ Examples:

```
/*
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
//comboBox initialize
comb = new TComboBox(this);
comb->Parent = this;

//visual options
comb->Align = TAlign::alLeft;
comb->DoubleBuffered = true;
comb->AutoComplete = true;

//adding items to the combo box
comb->AddItem("firstChoice",NULL);
comb->AddItem("secondChoice",NULL);
comb->AddItem("thirdChoice",NULL);

//setting the default value
comb->ItemIndex = 1;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
MessageDlg("Selected text: " + comb->Text,
            mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This is a very simple example showing how to create a comboBox, add to
it some choices, selecting a default value and getting the selected
value.
}
procedure TForm3.Button1Click(Sender: TObject);
begin
  MessageDlg('Selected text: ' + comb.Text, mtInformation,
            mbYesNo, 0);
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  //initialize the combo box
  comb := TComboBox.Create(Self);
  comb.Parent := Self;

  //visual options
  comb.Align := alLeft;
  comb.DoubleBuffered := true;
  comb.AutoComplete := true;

  //adding items to the combo box
  comb.AddItem('firstChoice', nil);
  comb.AddItem('secondChoice', nil);
  comb.AddItem('thirdChoice', nil);

  //setting the default value
  comb.ItemIndex := 1;
end;
```

See Also

AutoCompleteDelay (see page 2333)

30.1.9.5.3 TCustomComboBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2332) is true.

See Also

AutoComplete (see page 2332)

30.1.9.5.4 TCustomComboBox.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

Pascal

```
property AutoDropDown: Boolean;
```

C++

```
__property Boolean AutoDropDown;
```

Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

See Also

AutoComplete (see page 2332)

OnDropDown (see page 2320)

DroppedDown (see page 2306)

30.1.9.5.5 TCustomComboBox.CharCase Property

Determines the case of the text in the combo box.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the combo box into upper or lower case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types into the combo box. Changing the CharCase property changes the actual text, not just its appearance; case information is lost and cannot be recaptured by changing CharCase back to ecNormal.

See Also

Text

30.1.9.5.6 TCustomComboBox.SelText Property

Represents the selected text in the edit region.

Pascal

```
property SelText: string;
```

C++

```
__property UnicodeString SelText;
```

Description

Read (see page 3634) SelText to obtain the text that is selected in the edit region of the combo box. When the combo box is the active control, SelText appears highlighted. Set SelText to replace the current selection with a different substring. If no text is selected when a value is assigned to SelText, the SelText string is inserted in the text at the cursor.

Note: The SelText property is only valid when the combo box has focus. When the combo box does not have focus, reading SelText always yields an empty string and setting it inserts the text at the cursor rather than replacing the selected text.

See Also

SelLength (see page 2311)

SelStart (see page 2314)

30.1.9.5.7 TCustomComboBox.Sorted Property

Determines whether the list portion of the combo box is alphabetized.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to sort the items in the Items (see page 2308) list alphabetically. New (see page 3619) items added to the list while Sorted is true are inserted in the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

C++ Examples:

```
/*
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
*/
#include "Unit2.h"
#include "Unit3.h"

void __fastcall TForm1::ComboBox1DropDown(TObject *Sender)
{
  ComboBox1->Items->BeginUpdate(); // prevent repaints until done
```

```
  ComboBox1->Items->Clear(); // empty the list of any old values
  for (int i = 0; i < Screen->CustomFormCount; i++)
  {
    if ((reinterpret_cast<TForm1 *>(Screen->CustomForms[i]))->Visible)
      ComboBox1->Items->Add(Screen->CustomForms[i]->Caption);  // add form names
  }
  ComboBox1->Items->EndUpdate(); //reenable painting
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ComboBox1->Sorted = True; // make sure the form names are sorted
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Form2->Show();
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form3->Show();
}
```

Delphi Examples:

```
{
This example fills the list of a combo box with the set of
open forms when the user drops down the list. The list is
regenerated every time the list is dropped down, so if the
application opens or closes forms, the combo box stays current.
}
procedure TForm1.ComboBox1DropDown(Sender: TObject);
var
  I: Integer;
begin
  with ComboBox1 do
  begin
    Items.BeginUpdate; { prevent repaints until done }
    Items.Clear; { empty the list of any old values }
    for I := 0 to Screen.CustomFormCount - 1 do
    begin
      if (Screen.CustomForms[I].Visible) then
        Items.Add(Screen.CustomForms[I].Caption); { add form name }
    end;
    Items.EndUpdate; {reenable painting }
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Sorted := True; { make sure the form names are sorted }
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form3.Show;
end;
```

See Also

Items (see page 2308)

Using TDBListBox and TDBComboBox

30.1.9.5.8 TCustomComboBox.Style Property

Determines the display style of the combo box.

Pascal

```
property Style: TComboBoxStyle;
```

C++

```
__property TComboBoxStyle Style;
```

Description

Set Style to specify how the list and edit region of the combo box should appear.

When Style is csOwnerDrawFixed or csOwnerDrawVariable, the OnDrawItem (see page 2338) event occurs when the list items need to be drawn. If Style is csOwnerDrawVariable, an OnMeasureItem (see page 2338) event occurs as well.

See Also

Height

ItemHeight (see page 2307)

OnDrawItem (see page 2338)

OnMeasureItem (see page 2338)

30.1.9.5.9 TCustomComboBox.TextHint Property

Specifies the text that is displayed as a text watermark in the edit box of the combo box control.

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

Use TextHint to get or set the watermark text that is shown in the edit box of the combo box control.

The disabled watermark is displayed as long as you do not modify the Text property of the combo box.

Note: This feature is only available in Vista.

See Also

TCustomComboBox (see page 2322)

30.1.9.6 TCustomComboBox Events

The events of the TCustomComboBox class are listed here.

Protected Events

	Name	Description
	OnDrawItem (see page 2338)	Occurs when an item in an owner-draw combo box needs to be displayed.

	OnMeasureItem (see page 2338)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

30.1.9.6.1 TCustomComboBox.OnDrawItem Event

Occurs when an item in an owner-draw combo box needs to be displayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Write (see page 3707) an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas (see page 2305) using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style (see page 2337) is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

See Also

Canvas (see page 2305)

Items (see page 2308)

OnMeasureItem (see page 2338)

Style (see page 2337)

30.1.9.6.2 TCustomComboBox.OnMeasureItem Event

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

When Style (see page 2337) is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem (see page 2338). Write (see page 3707) an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

See Also

ItemHeight (see page 2307)

Items (see page 2308)

OnDrawItem (see page 2338)

Style (see page 2337)

30.1.10 TCustomComboBoxStrings Class

TCustomComboBoxStrings represents the strings in the drop-down list of a combo box.

Class Hierarchy

Pascal

```
TCustomComboBoxStrings = class(TStrings);
```

C++

```
class TCustomComboBoxStrings : public TStrings;
```

File

StdCtrls

Description

TCustomComboBoxStrings is the base class for objects that represent the items in the drop-down list of a combo box. Each type of combo box (TComboBox (see page 2184) or TComboBoxEx) uses its own TCustomComboBoxStrings descendant to manage the list of items.

Do not create instances of TCustomComboBoxStrings. This class is designed to work only with a combo box. TCustomCombo (see page 2283) creates the TCustomComboBoxStrings descendant it uses to implement its Items property.

See Also

TCustomCombo (see page 2283)

TComboBox (see page 2184)

TComboBoxEx

30.1.10.1 TCustomComboBoxStrings Members

The following tables list the members exposed by TCustomComboBoxStrings.

Protected Methods

	Name	Description
	Get (see page 2342)	Returns a string given its index.
	GetCount (see page 2342)	Returns the number of strings in the list
	GetObject (see page 2342)	Returns the object associated with the item at a specified index.
	PutObject (see page 2343)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 2344)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	ComboBox (see page 2344)	Identifies the combo box whose items this object represents.

Public Methods

	Name	Description
	Clear (see page 2341)	Deletes all the strings from the list.
	Create (see page 2340)	Constructs an object and initializes its data before the object is first used.

	Delete (see page 2341)	Removes the item specified by the Index parameter.
	Destroy (see page 2340)	Destroys the TPersistent instance and frees its memory.
	IndexOf (see page 2343)	Returns the position of a string in the list.

30.1.10.2 TCustomComboBoxStrings.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TCustomComboBoxStrings();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

30.1.10.3 TCustomComboBoxStrings.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomComboBoxStrings();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

System::TObject::Destroy (see page 3356)

30.1.10.4 TCustomComboBoxStrings Methods

The methods of the TCustomComboBoxStrings class are listed here.

Protected Methods

	Name	Description
	Get (see page 2342)	Returns a string given its index.
	GetCount (see page 2342)	Returns the number of strings in the list

	GetObject (see page 2342)	Returns the object associated with the item at a specified index.
	PutObject (see page 2343)	Changes the object associated with the string at a specified index.
	SetUpdateState (see page 2344)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Clear (see page 2341)	Deletes all the strings from the list.
	Delete (see page 2341)	Removes the item specified by the Index parameter.
	IndexOf (see page 2343)	Returns the position of a string in the list.

30.1.10.4.1 TCustomComboBoxStrings.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the drop-down list of the combo box. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 2341)

Objects

Strings

30.1.10.4.2 TCustomComboBoxStrings.Delete Method

Removes the item specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single item from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string in the drop-down list, where 0 is the first string, 1 is the second string, and so on.

See Also

Add

AddObject

Clear (see page 2341)

IndexOf (see page 2343)

InsertObject

Objects

Strings

30.1.10.4.3 TCustomComboBoxStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString Get(int Index);
```

Description

Get is the protected read implementation of the Strings property. It fetches the specified string from the combo box identified by the ComboBox (see page 2344) property.

Index identifies the string to return, where 0 is the index of the first string, 1 is the index of the second string, and so on.

See Also

Strings

IndexOf (see page 2343)

ComboBox (see page 2344)

30.1.10.4.4 TCustomComboBoxStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property. It fetches the number of items from the combo box identified by the ComboBox (see page 2344) property.

See Also

Count

ComboBox (see page 2344)

30.1.10.4.5 TCustomComboBoxStrings.GetObject Method

Returns the object associated with the item at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property. It fetches the associated object for the identified item from the combo box identified by the ComboBox (see page 2344) property.

Index is the index of the item with which the object is associated.

See Also

Objects

ComboBox (see page 2344)

30.1.10.4.6 TCustomComboBoxStrings.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: string): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const UnicodeString S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. The string must be an exact match (S can't be the prefix of a string in the combo box).

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

IndexOfName

IndexOfObject

Strings

30.1.10.4.7 TCustomComboBoxStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property. It associates the object specified by AObject with the item identified by Index.

Index identifies an item in the drop-down list of the combo box, where 0 is the index of the first item, 1 is the index of the second item, and so on.

AObject is the object to associated with the specified item.

See Also

Objects

30.1.10.4.8 TCustomComboBoxStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TCustomComboBoxStrings (see page 2339) object is not already in the middle of an update, SetUpdateState is called with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, SetUpdateState is called with Updating set to false.

SetUpdateState refreshes the combo box when Updating is false, so that the combo box reflects the new set of strings.

See Also

BeginUpdate

EndUpdate

30.1.10.5 TCustomComboBoxStrings Properties

The properties of the TCustomComboBoxStrings class are listed here.

Protected Properties

	Name	Description
	ComboBox (see page 2344)	Identifies the combo box whose items this object represents.

30.1.10.5.1 TCustomComboBoxStrings.ComboBox Property

Identifies the combo box whose items this object represents.

Pascal

```
property ComboBox: TCustomCombo;
```

C++

```
__property TCustomCombo ComboBox;
```

Description

TCustomComboBoxStrings (see page 2339) uses ComboBox internally so that the strings in the list and the items in the drop-down list of the combo box remain synchronized.

See Also

Items (see page 2308)

30.1.11 TCustomEdit Class

TCustomEdit is the base class from which all edit-box and memo controls are derived.

Class Hierarchy

Pascal

```
TCustomEdit = class(TWinControl);
```

C++

```
class TCustomEdit : public TWinControl;
```

File

StdCtrls

Description

TCustomEdit encapsulates the behavior common to all components for editing text by introducing methods and properties that provide:

Basic text editing functions such as selecting text, modifying selected text, and case conversions.

Ability to respond to changes in the contents of the text.

Access control of the text for making it read-only or introducing a password character to hide the actual value.

Do not create instances of TCustomEdit. Use TCustomEdit as a base class when declaring control objects that permit the user to enter or modify text. Properties and methods of TCustomEdit provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TCustomMaskEdit

TCustomMemo (see page 2443)

TEdit (see page 2463)

30.1.11.1 TCustomEdit Members

The following tables list the members exposed by TCustomEdit.

Protected Events

	Name	Description
	OnChange (see page 2381)	Occurs when the text for the edit control may have changed.

Protected Methods

	Name	Description
	Change (see page 2348)	Generates an OnChange (see page 2381) event.
V	CreateParams (see page 2351)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2351)	Creates a Windows control to represent the control.
V	CreateWnd (see page 2352)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2354)	Destroys the Windows control that corresponds to the edit control.

	DoSetMaxLength (see page 2354)	Sets the MaxLength (see page 2368) property.
	DoSetTextHint (see page 2355)	Sets the value of TextHint (see page 2380).
	GetSelLength (see page 2356)	Returns the SelLength (see page 2371) property.
	GetSelStart (see page 2356)	Returns the SelStart (see page 2374) property.
	GetSelText (see page 2356)	Returns the SelText (see page 2377) property.
	SetAlignment (see page 2361)	Sets the Alignment (see page 2365) property.
	SetAutoSize (see page 2361)	Sets the AutoSize (see page 2366) property.
	SetSelLength (see page 2361)	Sets the SelLength (see page 2371) property.
	SetSelStart (see page 2362)	Sets the SelStart (see page 2374) property.

Protected Properties

	Name	Description
	AutoSelect (see page 2365)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2366)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderStyle (see page 2367)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2367)	Determines the case of the text within the edit control.
	HideSelection (see page 2368)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	MaxLength (see page 2368)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2369)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2369)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentColor (see page 2370)	Determines where a control looks for its color information.
	PasswordChar (see page 2370)	Indicates the character, if any, to display in place of the actual characters typed in the control.

Public Methods

	Name	Description
	Clear (see page 2348)	Deletes all text from the edit control.
	ClearSelection (see page 2349)	Removes the selected text from the edit control.
	ClearUndo (see page 2349)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2349)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	Create (see page 2347)	Creates an instance of TCustomEdit (see page 2345).
	CutToClipboard (see page 2352)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 2354)	Provides message handling for all messages that the control does not fully process itself.
	GetControlsAlignment (see page 2355)	Indicates how text is aligned within the control.
	GetSelTextBuf (see page 2357)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2358)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2359)	Selects all text in the edit control.
	SetSelText (see page 2362)	Sets the SelText (see page 2377) property.
	SetSelTextBuf (see page 2363)	Replaces the selected text with a null-terminated string.
	Undo (see page 2364)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
	Alignment (see page 2365)	Determines how the text is aligned within the text edit control.
R	CanUndo (see page 2367)	Indicates whether the edit control contains changes that can be backed out.
	Modified (see page 2369)	Indicates whether the user edited the text of the edit control.
	ReadOnly (see page 2371)	Determines whether the user can change the text of the edit control.
	SelLength (see page 2371)	Specifies the number of characters (bytes) that are selected.

	SelStart (see page 2374)	Specifies the position of the first selected character in the text.
	SelText (see page 2377)	Specifies the selected portion of the edit control's text.
	Text (see page 2379)	Contains a text string associated with the control.
	TextHint (see page 2380)	A hint or message to be displayed when the Text (see page 2379) property is empty.

Published Properties

	Name	Description
	TabStop (see page 2379)	Determines if the user can tab to a control.

30.1.11.2 TCustomEdit.Create Constructor

Creates an instance of TCustomEdit (see page 2345).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomEdit(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TCustomEdit (see page 2345).

AOwner is another component, typically the form, that is responsible for freeing the edit control. It becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before performing any additional initialization.

See Also

Owner

30.1.11.3 TCustomEdit Methods

The methods of the TCustomEdit class are listed here.

Protected Methods

	Name	Description
	Change (see page 2348)	Generates an OnChange (see page 2381) event.
V	CreateParams (see page 2351)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2351)	Creates a Windows control to represent the control.
V	CreateWnd (see page 2352)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2354)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 2354)	Sets the MaxLength (see page 2368) property.
V	DoSetTextHint (see page 2355)	Sets the value of TextHint (see page 2380).
V	GetSelLength (see page 2356)	Returns the SelLength (see page 2371) property.
V	GetSelStart (see page 2356)	Returns the SelStart (see page 2374) property.
V	GetSelText (see page 2356)	Returns the SelText (see page 2377) property.
	SetAlignment (see page 2361)	Sets the Alignment (see page 2365) property.
V	SetAutoSize (see page 2361)	Sets the AutoSize (see page 2366) property.
V	SetSelLength (see page 2361)	Sets the SelLength (see page 2371) property.
V	SetSelStart (see page 2362)	Sets the SelStart (see page 2374) property.

Public Methods

	Name	Description
	Clear (see page 2348)	Deletes all text from the edit control.
	ClearSelection (see page 2349)	Removes the selected text from the edit control.
	ClearUndo (see page 2349)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2349)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	CutToClipboard (see page 2352)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 2354)	Provides message handling for all messages that the control does not fully process itself.
	GetControlsAlignment (see page 2355)	Indicates how text is aligned within the control.
	GetSelTextBuf (see page 2357)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2358)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2359)	Selects all text in the edit control.
	SetSelText (see page 2362)	Sets the SelText (see page 2377) property.
	SetSelTextBuf (see page 2363)	Replaces the selected text with a null-terminated string.
	Undo (see page 2364)	Backs out all changes in the undo buffer.

30.1.11.3.1 TCustomEdit.Change Method

Generates an OnChange (see page 2381) event.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Applications do not call the Change method directly. It is triggered automatically by Windows events. Change is exposed as a protected method so that derived classes can make additional responses when the text changes beyond invoking the OnChange (see page 2381) event handler. If Derived classes that override the Change method do not call the inherited method, the OnChange (see page 2381) event handler will not be called.

See Also

Handle

OnChange (see page 2381)

Text

30.1.11.3.2 TCustomEdit.Clear Method

Deletes all text from the edit control.

Pascal

```
procedure Clear; virtual;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to replace the Text (see page 2379) property with an empty string. Clear removes all text, not just the selected text.

To remove just the selected text, use ClearSelection (see page 2349).

See Also

ClearSelection (see page 2349)

Text

30.1.11.3.3 TCustomEdit.ClearSelection Method

Removes the selected text from the edit control.

Pascal

```
procedure ClearSelection;
```

C++

```
__fastcall ClearSelection();
```

Description

Use ClearSelection to delete the selected text from the edit control. If no text is selected, ClearSelection does nothing. If all of the text is selected, ClearSelection clears all text, like the Clear (see page 2348) method.

Note: Calling ClearSelection does the same thing as sending the edit control a WM_CLEAR message.

See Also

Clear (see page 2348)

CutToClipboard (see page 2352)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

30.1.11.3.4 TCustomEdit.ClearUndo Method

Clears the undo buffer so that no changes to the text can be backed out.

Pascal

```
procedure ClearUndo;
```

C++

```
__fastcall ClearUndo();
```

Description

Use ClearUndo to commit changes to the Text (see page 2379) property. After calling ClearUndo, the CanUndo (see page 2367) property is false and the Undo (see page 2364) method does nothing.

See Also

Undo (see page 2364)

CanUndo (see page 2367)

Text

30.1.11.3.5 TCustomEdit.CopyToClipboard Method

Copies the selected text in the edit control to the Clipboard in CF_TEXT format.

Pascal

```
procedure CopyToClipboard;
```

C++

```
__fastcall CopyToClipboard();
```

Description

Use CopyToClipboard to replace the contents of the Clipboard with the selected text. CopyToClipboard does not clear the Clipboard if no text is selected. If no text is selected, CopyToClipboard does nothing.

Note: Calling CopyToClipboard does the same thing as sending the edit control a WM_COPY message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->SelectAll();
  RichEdit1->CopyToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("..\\overview.rtf");
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;
  RichEdit1.CopyToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
```

```
    // you may need to change this path to suit your environment
    Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
```

See Also

CutToClipboard (see page 2352)

GetSelTextBuf (see page 2357)

PasteFromClipboard (see page 2358)

SelText (see page 2377)

Displaying and Editing Text in a Memo Control

30.1.11.3.6 TCustomEdit.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2352) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 2351). TCustomEdit (see page 2345) overrides the inherited method to specify a scrollable single-line edit control that reflects the current values of the BorderStyle (see page 2367), HideSelection (see page 2368), PasswordChar (see page 2370), and ReadOnly (see page 2371) properties.

See Also

BorderStyle (see page 2367)

CreateWindowHandle (see page 2351)

CreateWnd (see page 2352)

HideSelection (see page 2368)

PasswordChar (see page 2370)

ReadOnly (see page 2371)

30.1.11.3.7 TCustomEdit.CreateWindowHandle Method

Creates a Windows control to represent the control.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

CreateWindowHandle calls the inherited CreateWindowHandle method after working around a Win95 bug involving setting the read-only style on machines with Far East locales.

See Also

CreateParams (see page 2351)

CreateWnd (see page 2352)

DestroyWnd (see page 2354)

CreateWindowHandle

30.1.11.3.8 TCustomEdit.CreateWnd Method

Creates a Windows control corresponding to the edit control.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called when the edit control is created, and whenever a change to the edit control requires the window to be recreated. After calling the inherited method, CreateWnd sends the newly created window messages to enforce the MaxLength (see page 2368) and PasswordChar (see page 2370) properties.

See Also

CreateParams (see page 2351)

CreateWindowHandle

DestroyWnd (see page 2354)

MaxLength (see page 2368)

PasswordChar (see page 2370)

CreateWnd

30.1.11.3.9 TCustomEdit.CutToClipboard Method

Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.

Pascal

```
procedure CutToClipboard;
```

C++

```
__fastcall CutToClipboard();
```

Description

Use CutToClipboard to replace the contents of the Clipboard with the selected text while clearing the selected text. If no text is selected, CutToClipboard does nothing.

Note: Calling CopyToClipboard (see page 2349) does the same thing as sending the edit control a WM_CUT message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // ensure that all the text is cut, not just the current selection
  RichEdit1->SelectAll();
  RichEdit1->CutToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;  { ensure that all the text is cut, not just the current selection }
  RichEdit1.CutToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

ClearSelection (see page 2349)

CopyToClipboard (see page 2349)

GetSelTextBuf (see page 2357)

PasteFromClipboard (see page 2358)

SelText (see page 2377)

Displaying and Editing Text in a Memo Control

30.1.11.3.10 TCustomEdit.DefaultHandler Method

Provides message handling for all messages that the control does not fully process itself.

Pascal

```
procedure DefaultHandler(var Message); override;
```

C++

```
virtual __fastcall DefaultHandler( Message);
```

Description

Override DefaultHandler to change the default message handling for the edit control. The Message parameter can be cast to a TMessage type, to obtain the WParam, LParam, and Result of the message. If the Result of the message is non-zero, the message has already been handled. Set the Result field to a non-zero value to prevent further processing of the message by the inherited method.

Note: In Delphi, calling inherited in a message-handling method results in a call to the ancestor's DefaultHandler method if that ancestor does not specify a handler for the message being handled.

See Also

WindowProc

DefWndProc

TMessage

30.1.11.3.11 TCustomEdit.DestroyWnd Method

Destroys the Windows control that corresponds to the edit control.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

The Windows control for the edit box can be temporarily destroyed and recreated in order to set properties that require changing the window creation parameters. When this happens, DestroyWnd is called.

Before the window is destroyed, DestroyWnd saves a copy of any properties that are needed to recreate the window after it is destroyed.

See Also

CreateWnd (see page 2352)

30.1.11.3.12 TCustomEdit.DoSetMaxLength Method

Sets the MaxLength (see page 2368) property.

Pascal

```
procedure DoSetMaxLength(Value: Integer); virtual;
```

C++

```
virtual __fastcall DoSetMaxLength(int Value);
```

Description

DoSetMaxLength is the property access function for the MaxLength (see page 2368) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2345) can customize the implementation of the MaxLength (see page 2368) property.

DoSetMaxLength sends an EM_LIMITTEXT message to the window of the edit control.

See Also

MaxLength (see page 2368)

30.1.11.3.13 TCustomEdit.DoSetTextHint Method

Sets the value of TextHint (see page 2380).

Pascal

```
procedure DoSetTextHint(const Value: string); virtual;
```

C++

```
virtual __fastcall DoSetTextHint(const UnicodeString Value);
```

Description

DoSetTextHint is a protected method used to set the value of the TextHint (see page 2380) property.

See Also

TextHint (see page 2380)

30.1.11.3.14 TCustomEdit.GetControlsAlignment Method

Indicates how text is aligned within the control.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

Use GetControlsAlignment to determine the alignment of the control. For controls that do not have an Alignment property, GetControlsAlignment returns taLeftJustify.

Note: GetControlsAlignment returns the alignment of text before it is altered by the BiDiMode property. To check if the alignment should be reversed to reflect the BiDiMode property, call UseRightToLeftAlignment.

See Also

BiDiMode

Text

Caption

UseRightToLeftAlignment

30.1.11.3.15 TCustomEdit.GetSelLength Method

Returns the SelLength (see page 2371) property.

Pascal

```
function GetSelLength: Integer; virtual;
```

C++

```
virtual __fastcall int GetSelLength();
```

Description

GetSelLength is the property access function for the SelLength (see page 2371) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2345) can customize the implementation of the SelLength (see page 2371) property.

GetSelLength sends an EM_GETSEL message to the window of the edit control and returns the difference between the starting position and the ending position.

See Also

SelLength (see page 2371)

SetSelLength (see page 2361)

30.1.11.3.16 TCustomEdit.GetSelStart Method

Returns the SelStart (see page 2374) property.

Pascal

```
function GetSelStart: Integer; virtual;
```

C++

```
virtual __fastcall int GetSelStart();
```

Description

GetSelStart is the property access method for the SelStart (see page 2374) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2345) can customize the implementation of the SelStart (see page 2374) property.

GetSelStart sends an EM_GETSEL message to the window of the edit control and returns the starting position.

See Also

SelStart (see page 2374)

SetSelStart (see page 2362)

30.1.11.3.17 TCustomEdit.GetSelText Method

Returns the SelText (see page 2377) property.

Pascal

```
function GetSelText: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetSelText();
```

Description

GetSelText is the property access method for the SelText (see page 2377) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2345) can customize the implementation of the SelText (see page 2377) property.

See Also

SelText (see page 2377)

30.1.11.3.18 TCustomEdit.GetSelTextBuf Method

Copies the selected text into a buffer and returns the number of characters copied.

Pascal

```
function GetSelTextBuf(Buffer: PChar; BufSize: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetSelTextBuf(const char * Buffer, int BufSize);
```

Description

Use GetSelTextBuf to copy the selected text into a character buffer. If there is no selection, the buffer receives an empty string. If the selection contains more than (BufSize - 1) characters, only the first (BufSize - 1) characters are copied. GetSelTextBuf returns the number of characters that were actually copied into the buffer.

C++ Examples:

```
/*
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
*/
Char *Buffer;

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->SelLength; //Get length of selected text in Edit1
  Size++;                      //Add room for null character
  static std::auto_ptr<Char> _BufferCleaner(Buffer = new Char[Size]);
  Edit1->GetSelTextBuf(Buffer, Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
}
//---------------------------------------------------------------------------
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->SelText;
}
```

Delphi Examples:

```
{
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  Buffer: PChar;
  Size: Integer;
begin
  Size := Edit1.SelLength; {Get length of selected text in Edit1}
  Inc(Size); {Add room for null character}
  GetMem(Buffer, Size); {Creates Buffer dynamic variable}
  Edit1.GetSelTextBuf(Buffer,Size); {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);{Converts Buffer into Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.SelText;
end;
```

See Also

CopyToClipboard (see page 2349)

GetTextBuf

SelLength (see page 2371)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

30.1.11.3.19 TCustomEdit.PasteFromClipboard Method

Pastes the contents of the Clipboard into edit control, replacing the current selection.

Pascal

```
procedure PasteFromClipboard;
```

C++

```
__fastcall PasteFromClipboard();
```

Description

Use PasteFromClipboard to replace the selected text with the contents of the Clipboard, or, if no text is selected, to insert the contents of the Clipboard at the cursor. If the Clipboard is empty, or if it does not contain anything in CF_TEXT format, PasteFromClipboard does nothing.

Note: Calling PasteFromClipboard does the same thing as sending the edit control a WM_PASTE message.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  RichEdit1->SelectAll();
  RichEdit1->CopyToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("..\\overview.rtf");
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, the first
line of text is copied from the rich edit control and pasted
into the edit box.  Note that only the text is pasted. If
the rich edit control includes any formatting information
that is not pasted into the edit control.  If the
destination were a rich edit control, the formatting
information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;
  RichEdit1.CopyToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  // you may need to change this path to suit your environment
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

CutToClipboard (see page 2352)

SelStart (see page 2374)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

Displaying and Editing Text in a Memo Control

30.1.11.3.20 TCustomEdit.SelectAll Method

Selects all text in the edit control.

Pascal

```
procedure SelectAll;
```

C++

```
__fastcall SelectAll();
```

Description

Use SelectAll to select all text in the edit control. To select only part of the text, use the SelStart (see page 2374) and SelLength (see page 2371) properties.

C++ Examples:

```
/*
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // ensure that all the text is cut, not just the current selection
  RichEdit1->SelectAll();
  RichEdit1->CutToClipboard();
  Edit1->Clear();
  Edit1->PasteFromClipboard();
  RichEdit1->SetFocus();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  RichEdit1->ScrollBars = ssVertical;
}
```

Delphi Examples:

```
{
This example uses an edit box, a rich edit control, and a
button on a form. When the user clicks the button, text is
cut from the rich edit control and pasted into the edit box.
Note that only the text is pasted. If the rich edit control
includes any formatting information that is not pasted into
the edit control.  If the destination where a rich edit
control, the formatting information would be copied as well.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  RichEdit1.SelectAll;  { ensure that all the text is cut, not just the current selection }
  RichEdit1.CutToClipboard;
  Edit1.Clear;
  Edit1.PasteFromClipboard;
  RichEdit1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
const
  Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
```

See Also

SelLength (see page 2371)

SelStart (see page 2374)

SelText (see page 2377)

Text

30.1.11.3.21 TCustomEdit.SetAlignment Method

Sets the Alignment (see page 2365) property.

Pascal

```
procedure SetAlignment(Value: TAlignment);
```

C++

```
__fastcall SetAlignment(TAlignment Value);
```

Description

SetAlignment is exposed as a protected method so derived classes can customize their implementation of the Alignment (see page 2365) property.

See Also

Alignment (see page 2365)

30.1.11.3.22 TCustomEdit.SetAutoSize Method

Sets the AutoSize (see page 2366) property.

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

SetAutoSize is the protected write implementation of the AutoSize (see page 2366) property.

TCustomEdit (see page 2345) overrides the inherited method to update the height of the edit control to reflect the current font when the value of AutoSize (see page 2366) changes to true.

See Also

AutoSize (see page 2366)

30.1.11.3.23 TCustomEdit.SetSelLength Method

Sets the SelLength (see page 2371) property.

Pascal

```
procedure SetSelLength(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSelLength(int Value);
```

Description

SetSelLength is the property access method for the SelLength (see page 2371) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2345) can customize the implementation of the SelLength (see page 2371) property.

SetSelLength sends an EM_SETSEL message to the window of the edit control, specifying a selection that begins at SelStart (see page 2374) and ends at SelStart (see page 2374) + Value.

See Also

GetSelLength (see page 2356)

SelLength (see page 2371)

SelStart (see page 2374)

30.1.11.3.24 TCustomEdit.SetSelStart Method

Sets the SelStart (see page 2374) property.

Pascal

```
procedure SetSelStart(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetSelStart(int Value);
```

Description

SetSelStart is the property access method for the SelStart (see page 2374) property. It is exposed as a protected method so that descendants of TCustomEdit (see page 2345) can customize the implementation of the SelStart (see page 2374) property.

SetSelLength (see page 2361) sends an EM_SETSEL message to the window of the edit control to position the cursor on the position indicated by Value.

See Also

GetSelStart (see page 2356)

SelStart (see page 2374)

30.1.11.3.25 TCustomEdit.SetSelText Method

Sets the SelText (see page 2377) property.

Pascal

```
procedure SetSelText(const Value: string);
```

C++

```
__fastcall SetSelText(const UnicodeString Value);
```

Description

SetSelText is the property access method for the SelText (see page 2377) property.

See Also

GetSelTextBuf (see page 2357)

SelText (see page 2377)

30.1.11.3.26 TCustomEdit.SetSelTextBuf Method

Replaces the selected text with a null-terminated string.

Pascal

```
procedure SetSelTextBuf(Buffer: PChar);
```

C++

```
__fastcall SetSelTextBuf(const char * Buffer);
```

Description

Use SetSelTextBuf to replace the current selection by the contents of the null-terminated string pointed to by Buffer. If no text is selected, the contents of Buffer are inserted at the cursor. SetSelTextBuf does the same thing as setting the SelText (see page 2377) property, except that it takes a pointer to a char type rather than a string.

C++ Examples:

```
/*
This example inserts the contents of temp_string at the
current location of the cursor.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   const AnsiString Path = "../readme.txt";
   Memo1->Lines->LoadFromFile(Path);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  WideChar *temp_string = L"I am now here";
  Memo1->SetSelTextBuf(temp_string);
}
```

Delphi Examples:

```
{
This example inserts the contents of temp_string at the
current location of the cursor.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  temp_string: array[0..15] of Char;
begin
  temp_string := 'I am now here';
  Memo1.SetSelTextBuf(temp_string);
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'readme.txt';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
```

See Also

GetSelTextBuf (see page 2357)

PasteFromClipboard (see page 2358)

SelText (see page 2377)

SetTextBuf

30.1.11.3.27 TCustomEdit.Undo Method

Backs out all changes in the undo buffer.

Pascal

```
procedure Undo;
```

C++

```
__fastcall Undo();
```

Description

Call Undo to cancel all changes made to the Text (see page 2379) property since the last call to the ClearUndo (see page 2349) method. If ClearUndo (see page 2349) was never called, Undo rolls back all changes.

To determine whether there are any changes in the undo buffer, check the CanUndo (see page 2367) property.

See Also

ClearUndo (see page 2349)

CanUndo (see page 2367)

Text

30.1.11.4 TCustomEdit Properties

The properties of the TCustomEdit class are listed here.

Protected Properties

	Name	Description
	AutoSelect (see page 2365)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2366)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderStyle (see page 2367)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2367)	Determines the case of the text within the edit control.
	HideSelection (see page 2368)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	MaxLength (see page 2368)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2369)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2369)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentColor (see page 2370)	Determines where a control looks for its color information.
	PasswordChar (see page 2370)	Indicates the character, if any, to display in place of the actual characters typed in the control.

Public Properties

	Name	Description
	Alignment (see page 2365)	Determines how the text is aligned within the text edit control.
R	CanUndo (see page 2367)	Indicates whether the edit control contains changes that can be backed out.
	Modified (see page 2369)	Indicates whether the user edited the text of the edit control.
	ReadOnly (see page 2371)	Determines whether the user can change the text of the edit control.
	SelLength (see page 2371)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2374)	Specifies the position of the first selected character in the text.
	SelText (see page 2377)	Specifies the selected portion of the edit control's text.

	Text (see page 2379)	Contains a text string associated with the control.
	TextHint (see page 2380)	A hint or message to be displayed when the Text (see page 2379) property is empty.

Published Properties

	Name	Description
	TabStop (see page 2379)	Determines if the user can tab to a control.

30.1.11.4.1 TCustomEdit.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 2361)

Text

Displaying and Editing Fields in an Edit Box

30.1.11.4.2 TCustomEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 2359)

SelLength (see page 2371)

SelStart (see page 2374)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

30.1.11.4.3 TCustomEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 2367)

ClientHeight

ControlStyle

Height

30.1.11.4.4 TCustomEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

30.1.11.4.5 TCustomEdit.CanUndo Property

Indicates whether the edit control contains changes that can be backed out.

Pascal

```
property CanUndo: Boolean;
```

C++

```
__property Boolean CanUndo;
```

Description

Read (see page 3634) CanUndo to determine whether the user has made any changes to the text that can be undone by calling the Undo (see page 2364) method. This is useful when enabling or disabling menu items that correspond to these commands.

See Also

Undo (see page 2364)

ClearUndo (see page 2349)

Text

30.1.11.4.6 TCustomEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

30.1.11.4.7 TCustomEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 2371)

SelStart (see page 2374)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

30.1.11.4.8 TCustomEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

30.1.11.4.9 TCustomEdit.Modified Property

Indicates whether the user edited the text of the edit control.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Use Modified to determine whether the user changed the Text (see page 2379) property of the edit control. Modified is only reset to **False** when you assign a value to the Text (see page 2379) property. In particular, it is not reset when the control receives focus.

See Also

Change (see page 2348)

OnChange (see page 2381)

SetTextBuf

Text

30.1.11.4.10 TCustomEdit.NumbersOnly Property

Allows only numbers to be typed into the text edit.

Pascal

```
property NumbersOnly: Boolean;
```

C++

```
__property Boolean NumbersOnly;
```

Description

Use NumbersOnly to prohibit entry of non-numeric characters in the textfield. Note, however, that a user can paste non-numeric characters in the textfield even when this property is set.

30.1.11.4.11 TCustomEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

30.1.11.4.12 TCustomEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.11.4.13 TCustomEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text (see page 2379) property reflects the actual characters that are typed.

See Also

Text

30.1.11.4.14 TCustomEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 2349)

GetSelTextBuf (see page 2357)

30.1.11.4.15 TCustomEdit.SelLength Property

Specifies the number of characters (bytes) that are selected.

Pascal

```
property SelLength: Integer;
```

C++

```
__property int SelLength;
```

Description

Read (see page 3634) SelLength to determine the length, in bytes, of the selected text. This is the same as the number of characters, unless you are using a multi-byte character set. Set SelLength to change the selection to consist of the first SelLength bytes starting at SelStart (see page 2374).

Note: Setting SelLength to a value greater than the number of characters from SelStart (see page 2374) to the end of the text results in the selection of all characters from SelStart (see page 2374) to the end of the text. Reading SelLength immediately after setting it to a value greater than the number of available characters returns the number of characters actually selected, not the value that was just set.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection. Notice that the
"Match whole word only" and "Match case" work.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
```

```
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes = TSearchTypes();
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
*/
void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  for (int I = 0; I < Memo1->Lines->Count; I++)
  {
    int PosReturn = Memo1->Lines->Strings[I].Pos(FindDialog1->FindText);
    if (PosReturn) //found!
    {
      int Skipchars = 0;
      for (int J = 0; J < I; J++)
        Skipchars += Memo1->Lines->Strings[J].Length();
      Skipchars += I*2; // add CR/LF for all skipped lines
      Skipchars += PosReturn - 1;
      Memo1->SetFocus();
      Memo1->SelStart = Skipchars;
      Memo1->SelLength = FindDialog1->FindText.Length();
      break;
    }
```

```
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  mySearchTypes := [];
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
    ToEnd := Length(Text) - StartPos;
    FoundAt :=
      FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
    if FoundAt <> -1 then
    begin
      SetFocus;
      SelStart := FoundAt;
      SelLength := Length(FindDialog1.FindText);
    end
    else Beep;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'OverView.RTF';
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
```

```
  compare strings, and stores the number of characters to skip
  when determining the selection position in the SkipChars
  variable. Because there is no handling of case, whole word,
  or search direction in this algorithm, it is assumed that
  the Options property of FindDialog1 is set to
  [frHideMatchCase, frHideWholeWord, frHideUpDown].
  }
  procedure TForm1.FindDialog1Find(Sender: TObject);
  var
    I, J, PosReturn, SkipChars: Integer;
  begin
    for I := 0 to Memo1.Lines.Count do
    begin
      PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
      if PosReturn <> 0 then {found!}
      begin
        SkipChars := 0;
        for J := 0 to I - 1 do
          SkipChars := SkipChars + Length(Memo1.Lines[J]);
        SkipChars := SkipChars + (I*2);
        SkipChars := SkipChars + PosReturn - 1;
        Memo1.SetFocus;
        Memo1.SelStart := SkipChars;
        Memo1.SelLength := Length(FindDialog1.FindText);
        Break;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'country.list';
  begin
    Memo1.Lines.LoadFromFile(Path);
  end;
```

See Also

SelStart (see page 2374)

SelText (see page 2377)

30.1.11.4.16 TCustomEdit.SelStart Property

Specifies the position of the first selected character in the text.

Pascal

```
property SelStart: Integer;
```

C++

```
__property int SelStart;
```

Description

Read (see page 3634) SelStart to determine the position of the first selected character, where 0 indicates the first character. If there is no selected text, SelStart indicates the position of the cursor. Set SelStart to remove the current selection and position the cursor just before the indicated character.

To select a particular range of the text, first set SelStart to position the cursor, and then set SelLength (see page 2371) to extend the selection.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
```

```
  TFindDialog.  Clicking the button will display a Find
  Dialog to the right of the edit control.  Filling in the
  "Find what" text and pressing the Find Next button will
  select the first matching string in the Rich Edit control
  that follows the previous selection. Notice that the
  "Match whole word only" and "Match case" work.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
    FindDialog1->Execute();
  }

  void __fastcall TForm1::FindDialog1Find(TObject *Sender)
  {
    int FoundAt, StartPos, ToEnd;
    TSearchTypes mySearchTypes = TSearchTypes();
    // begin the search after the current selection
    // if there is one
    // otherwise, begin at the start of the text
    if (FindDialog1->Options.Contains(frMatchCase))
      mySearchTypes << stMatchCase;
    if (FindDialog1->Options.Contains(frWholeWord))
      mySearchTypes << stWholeWord;
    if (RichEdit1->SelLength)
      StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
    else
      StartPos = 0;
    // ToEnd is the length from StartPos
    // to the end of the text in the rich edit control
    ToEnd = RichEdit1->Text.Length() - StartPos;
    FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
    if (FoundAt != -1)
    {
      RichEdit1->SetFocus();
      RichEdit1->SelStart = FoundAt;
      RichEdit1->SelLength = FindDialog1->FindText.Length();
    }
    else Beep();
  }

  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    RichEdit1->PlainText = False;
    RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
    RichEdit1->ScrollBars = ssVertical;
  }
  /*
  The following OnFind event handler searches a memo component
  for the text specified in the FindText property of a find
  dialog component. If found, the first occurrence of the text
  in Memo1 is selected. The code uses the Pos function to
  compare strings, and stores the number of characters to skip
  when determining the selection position in the SkipChars
  variable. Because there is no handling of case, whole word,
  or search direction in this algorithm, it is assumed that
  the Options property of FindDialog1 is set to
  [frHideMatchCase, frHideWholeWord, frHideUpDown].
  */
  void __fastcall TForm1::FindDialog1Find(TObject *Sender)
  {
    for (int I = 0; I < Memo1->Lines->Count; I++)
    {
      int PosReturn = Memo1->Lines->Strings[I].Pos(FindDialog1->FindText);
      if (PosReturn) //found!
      {
```

```
      int Skipchars = 0;
      for (int J = 0; J < I; J++)
        Skipchars += Memo1->Lines->Strings[J].Length();
      Skipchars += I*2; // add CR/LF for all skipped lines
      Skipchars += PosReturn - 1;
      Memo1->SetFocus();
      Memo1->SelStart = Skipchars;
      Memo1->SelLength = FindDialog1->FindText.Length();
      break;
    }
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  mySearchTypes := [];
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { ToEnd is the length from StartPos to the end of the
      text in the rich edit control }
    ToEnd := Length(Text) - StartPos;
    FoundAt :=
      FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
    if FoundAt <> -1 then
    begin
      SetFocus;
      SelStart := FoundAt;
      SelLength := Length(FindDialog1.FindText);
    end
    else Beep;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'OverView.RTF';
```

```
begin
  RichEdit1.PlainText := False;
  RichEdit1.Lines.LoadFromFile(Path);
  RichEdit1.ScrollBars := ssVertical;
end;
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'country.list';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
```

See Also

SelLength (see page 2371)

SelText (see page 2377)

30.1.11.4.17 TCustomEdit.SelText Property

Specifies the selected portion of the edit control's text.

Pascal

```
property SelText: string;
```

C++

```
__property UnicodeString SelText;
```

Description

Read (see page 3634) SelText to determine the value of the selected text. Set SelText to replace the selected text with a new string. If there is no selection, but the edit control has focus, set SelText to insert a new string into the text at the cursor.

C++ Examples:

```
/*
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
*/
Char *Buffer;

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->SelLength; //Get length of selected text in Edit1
  Size++;                      //Add room for null character
  static std::auto_ptr<Char> _BufferCleaner(Buffer = new Char[Size]);
  Edit1->GetSelTextBuf(Buffer, Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer;
}
//---------------------------------------------------------------------------
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->SelText;
}
```

Delphi Examples:

```
{
This example copies selected text from one text edit to
another.  This example requires two text edits and two buttons.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Integer;
begin
  Size := Edit1.SelLength; {Get length of selected text in Edit1}
  Inc(Size); {Add room for null character}
  GetMem(Buffer, Size); {Creates Buffer dynamic variable}
  Edit1.GetSelTextBuf(Buffer,Size); {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);{Converts Buffer into Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.SelText;
end;
```

See Also

ClearSelection (see page 2349)

GetSelTextBuf (see page 2357)

SelLength (see page 2371)

SelStart (see page 2374)

SetSelTextBuf (see page 2363)

Text

30.1.11.4.18 TCustomEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.11.4.19 TCustomEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                       //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
```

```
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines (see page 2451)

30.1.11.4.20 TCustomEdit.TextHint Property

A hint or message to be displayed when the Text (see page 2379) property is empty.

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

30.1.11.5 TCustomEdit Events

The events of the TCustomEdit class are listed here.

Protected Events

	Name	Description
	OnChange (see page 2381)	Occurs when the text for the edit control may have changed.

30.1.11.5.1 TCustomEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3707) an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified (see page 2369) property to see if a change actually occurred. The Text (see page 2379) property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 2348)

Text

30.1.12 TCustomGroupBox Class

TCustomGroupBox is the base class for all group-box components.

Class Hierarchy

Pascal

```
TCustomGroupBox = class(TCustomControl);
```

C++

```
class TCustomGroupBox : public TCustomControl;
```

File

StdCtrls

Description

Group boxes are visual components that act as containers for sets of related controls, such as radio buttons. Group boxes may also contain other group boxes. TCustomGroupBox is the class from which all group-box components—including TGroupBox (see page 2535), TRadioGroup, and TDBRadioGroup—descend.

See Also

TDBRadioGroup

TRadioGroup

TGroupBox (see page 2535)

TCustomControl

TControl

30.1.12.1 TCustomGroupBox Members

The following tables list the members exposed by TCustomGroupBox.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2382)	Adjusts the ClientRect property for idiosyncrasies of the window.
	CreateParams (see page 2383)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 2383)	Renders the onscreen image of the group box.

Public Methods

	Name	Description
	Create (see page 2382)	Creates and initializes a TCustomGroupBox (see page 2381) instance.

30.1.12.2 TCustomGroupBox.Create Constructor

Creates and initializes a TCustomGroupBox (see page 2381) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomGroupBox(TComponent * AOwner);
```

Description

Call Create to instantiate a group box at runtime. Group boxes placed on forms at design time are created automatically.

AOwner is the component, typically a form, that is responsible for freeing the group box. It becomes the value of the Owner property.

See Also

Owner

30.1.12.3 TCustomGroupBox Methods

The methods of the TCustomGroupBox class are listed here.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2382)	Adjusts the ClientRect property for idiosyncrasies of the window.
	CreateParams (see page 2383)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 2383)	Renders the onscreen image of the group box.

30.1.12.3.1 TCustomGroupBox.AdjustClientRect Method

Adjusts the ClientRect property for idiosyncrasies of the window.

Pascal

```
procedure AdjustClientRect(var Rect: TRect); override;
```

C++

```
virtual __fastcall AdjustClientRect(TRect Rect);
```

Description

AdjustClientRect is called internally when the group box needs accurate information on where to place child controls within the client area. It indents the client rectangle slightly so that child controls are not placed on the frame (or caption) of the group box.

See Also

ClientRect

Caption

30.1.12.3.2 TCustomGroupBox.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a group box creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

Specify the parameter record as the value of the Params parameter. The TCreateParams type is a data structure holding information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TCustomGroupBox (see page 2381) adjusts the window style by turning off the CS_HREDRAW and CS_VREDRAW flags.

See Also

CreateWnd

CreateWindowHandle

30.1.12.3.3 TCustomGroupBox.Paint Method

Renders the onscreen image of the group box.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

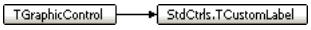
Description

Paint draws a border around the group box and fills in the caption.

30.1.13 TCustomLabel Class

TCustomLabel is the base class for non-windowed controls that display text on a form.

Class Hierarchy



Pascal

```
TCustomLabel = class(TGraphicControl);
```

C++

```
class TCustomLabel : public TGraphicControl;
```

File

StdCtrls

Description

Use TCustomLabel as a base class when defining objects that display text that users can't edit. TCustomLabel introduces several new properties and methods to control the appearance of the text.

TCustomLabel is not a descendant of TWinControl, so it does not have its own window and can't receive direct input from the keyboard. To define an object that responds to direct keyboard input in addition to displaying text, use TCustomStaticText (see page 2457) as a base class.

Do not create instances of TCustomLabel. To put a label on a form, use a TLabel (see page 2607) object.

See Also

TCustomStaticText (see page 2457)

TLabel (see page 2607)

30.1.13.1 TCustomLabel Members

The following tables list the members exposed by TCustomLabel.

Protected Methods

	Name	Description
	AdjustBounds (see page 2385)	Resizes the label when the text changes.
	DoDrawText (see page 2386)	Renders the text in the client area.
V	GetLabelText (see page 2386)	Returns the value of the Caption (see page 2392) property.
V	Loaded (see page 2387)	Adjusts the size of the label when it is first loaded. .
V	Notification (see page 2387)	Responds to notifications when objects are about to be created or destroyed.
V	Paint (see page 2387)	Draws the image of the label.
V	SetAutoSize (see page 2388)	Sets the AutoSize (see page 2389) property.

Protected Properties

	Name	Description
	Alignment (see page 2388)	Controls the horizontal placement of the text within the label.
	AutoSize (see page 2389)	Determines whether the size of the label automatically resizes to accommodate the text.
	EllipsisPosition (see page 2395)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.

	FocusControl (see page 2395)	Designates a windowed control associated with the label.
	Layout (see page 2396)	Specifies the vertical placement of the text within the label.
	ShowAccelChar (see page 2396)	Determines how an ampersand in the label text is displayed.
	Transparent (see page 2397)	Specifies whether controls that sit below the label on a form can be seen through the label.
	WordWrap (see page 2397)	Specifies whether the label text wraps when it is too long for the width of the label.

Public Methods

	Name	Description
	Create (see page 2385)	Creates and initializes an instance of TCustomLabel (see page 2384).

Public Properties

	Name	Description
	Canvas (see page 2390)	Provides the drawing surface used by the graphic control.
	Caption (see page 2392)	Specifies a text string that identifies the control to the user.
	GlowSize (see page 2395)	Specifies the radius of the glow around the label.

30.1.13.2 TCustomLabel.Create Constructor

Creates and initializes an instance of TCustomLabel (see page 2384).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomLabel(TComponent * AOwner);
```

Description

Call Create to create an instance of TCustomLabel (see page 2384) at runtime. Labels placed on forms at design time are created automatically.

AOwner is a component, typically the form, that is responsible for freeing the label.

See Also

Owner

30.1.13.3 TCustomLabel Methods

The methods of the TCustomLabel class are listed here.

Protected Methods

	Name	Description
	AdjustBounds (see page 2385)	Resizes the label when the text changes.
	DoDrawText (see page 2386)	Renders the text in the client area.
	GetLabelText (see page 2386)	Returns the value of the Caption (see page 2392) property.
	Loaded (see page 2387)	Adjusts the size of the label when it is first loaded. .
	Notification (see page 2387)	Responds to notifications when objects are about to be created or destroyed.
	Paint (see page 2387)	Draws the image of the label.
	SetAutoSize (see page 2388)	Sets the AutoSize (see page 2389) property.

30.1.13.3.1 TCustomLabel.AdjustBounds Method

Resizes the label when the text changes.

Pascal

```
procedure AdjustBounds; dynamic;
```

C++

```
__fastcall AdjustBounds();
```

Description

When the text in the label or the font changes, AdjustBounds is implicitly called. It redraws the text in the client area, and then resizes the label if the AutoSize (see page 2389) property is true.

When WordWrap (see page 2397) is true, the width of the label is fixed. If AutoSize (see page 2389) is also true, changes to the text cause the label to change in height. When AutoSize (see page 2389) is true and WordWrap (see page 2397) is false, the height of the label is determined by the font, and changes to the text cause the label to change in width.

Override AdjustBounds to make additional adjustments when the text changes.

See Also

DoDrawText (see page 2386)

AutoSize (see page 2389)

30.1.13.3.2 TCustomLabel.DoDrawText Method

Renders the text in the client area.

Pascal

```
procedure DoDrawText(var Rect: TRect; Flags: Longint); dynamic;
```

C++

```
__fastcall DoDrawText(TRect Rect, Longint Flags);
```

Description

When the label needs to draw itself, DoDrawText is implicitly called. DoDrawText uses the Windows API function, DrawText, to render the caption in the client area.

The first parameter, Rect, specifies the boundaries of the text within the client area. The second parameter, Flags, specifies how the text is aligned within those boundaries, how tab characters are handled, etc. See the Microsoft Windows API DrawText function for all the possible values of Flags.

Override DoDrawText to render text differently.

See Also

Caption

AdjustBounds (see page 2385)

30.1.13.3.3 TCustomLabel.GetLabelText Method

Returns the value of the Caption (see page 2392) property.

Pascal

```
function GetLabelText: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetLabelText();
```

Description

Call GetLabelText to obtain the string that appears as the text of the label.

See Also

Caption

30.1.13.3.4 TCustomLabel.Loaded Method

Adjusts the size of the label when it is first loaded. .

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The protected Loaded method is called immediately after a saved version of the form that contains the label is loaded into memory. TCustomLabel (see page 2384) uses this method to adjust the size of the label to the text it contains if the AutoSize (see page 2389) property is true.

See Also

AutoSize (see page 2389)

30.1.13.3.5 TCustomLabel.Notification Method

Responds to notifications when objects are about to be created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are created or destroyed. After calling the inherited method, Notification checks whether the windowed control specified by the FocusControl (see page 2395) property is about to be destroyed. If the windowed control is going away, Notification sets the FocusControl (see page 2395) property to nil (Delphi) or NULL (C++).

See Also

FocusControl (see page 2395)

30.1.13.3.6 TCustomLabel.Paint Method

Draws the image of the label.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically when the label receives a WM_PAINT message from its Parent. Paint writes the text returned by the GetLabelText (see page 2386) method.

See Also

GetLabelText (see page 2386)

30.1.13.3.7 TCustomLabel.SetAutoSize Method

Sets the AutoSize (see page 2389) property.

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

SetAutoSize is the protected access method for the AutoSize (see page 2389) property. It is exposed as a protected method so descendants can override it to change the implementation of the AutoSize (see page 2389) property.

See Also

AutoSize (see page 2389)

30.1.13.4 TCustomLabel Properties

The properties of the TCustomLabel class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 2388)	Controls the horizontal placement of the text within the label.
	AutoSize (see page 2389)	Determines whether the size of the label automatically resizes to accommodate the text.
	EllipsisPosition (see page 2395)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	FocusControl (see page 2395)	Designates a windowed control associated with the label.
	Layout (see page 2396)	Specifies the vertical placement of the text within the label.
	ShowAccelChar (see page 2396)	Determines how an ampersand in the label text is displayed.
	Transparent (see page 2397)	Specifies whether controls that sit below the label on a form can be seen through the label.
	WordWrap (see page 2397)	Specifies whether the label text wraps when it is too long for the width of the label.

Public Properties

	Name	Description
	Canvas (see page 2390)	Provides the drawing surface used by the graphic control.
	Caption (see page 2392)	Specifies a text string that identifies the control to the user.
	GlowSize (see page 2395)	Specifies the radius of the glow around the label.

30.1.13.4.1 TCustomLabel.Alignment Property

Controls the horizontal placement of the text within the label.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap (see page 2397) property is true and the label includes more than one line of text.

See Also

ClientRect

GetLabelText (see page 2386)

Layout (see page 2396)

WordWrap (see page 2397)

30.1.13.4.2 TCustomLabel.AutoSize Property

Determines whether the size of the label automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the label adjust its size automatically so the client area accommodates the height and width of the text. When AutoSize is false, the label is fixed in size. When AutoSize is true, the size of the label readjusts whenever the text changes. The size of the label is also readjusts when the Font property changes.

When WordWrap (see page 2397) is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap (see page 2397) is false, the font determines the height of the label, and changes to the text cause the label to change in width.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
```

```
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
```

See Also

BoundsRect

GetLabelText (see page 2386)

Font

WordWrap (see page 2397)

Layout (see page 2396)

30.1.13.4.3 TCustomLabel.Canvas Property

Provides the drawing surface used by the graphic control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control.

Since a TGraphicControl instance does not have its own Windows screen object, it obtains its device context from its parent control.

C++ Examples:

```
/*
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clRed;
   PaintBox1->Canvas->FillRect(PaintBox1->Canvas->ClipRect);
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}

/*
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
```

```
internal bitmap that stores this drawing information for
you.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
   PaintBox1->Canvas->Brush->Color = clBlue;
   PaintBox1->Canvas->Ellipse(0,0,100,100);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TLabel *myLabel;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TLabel")
  {
    myLabel = dynamic_cast<TLabel *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myLabel->Canvas->MoveTo(myLabel->ClientRect.Left, myLabel->ClientRect.Top);
    myLabel->Canvas->LineTo(myLabel->ClientRect.Right, myLabel->ClientRect.Bottom);
    myLabel = NULL;
  }
}
/*
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PaintBox1->Canvas->Draw(0,0, Image1->Picture->Graphic);
}
```

Delphi Examples:

```
{
When you do custom drawing, such as shown in the code for
the Button's OnClick event handler below, this drawing will
not be persistent and will be "erased" the first time the
PaintBox component needs to repaint.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clRed;
   PaintBox1.Canvas.FillRect(PaintBox1.Canvas.ClipRect);
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
In order for the drawing to be persistent, you need to also
include an OnPaint event handler that tells the PaintBox how
to redraw itself when it needs to. In the code below, an
ellipse will be drawn everytime the PaintBox renders itself,
but the PaintBox will only paint the client area of the
control red after the Button is clicked. This client area
will only stay red until it is invalidated, but the ellipse
will persist. This behavior is different from the TImage
component, because the TImage component maintains an
internal bitmap that stores this drawing information for
you.
}
```

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
   PaintBox1.Canvas.Brush.Color := clBlue;
   PaintBox1.Canvas.Ellipse(0,0,100,50);
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TLabel) then
  begin
    myLabel:= (ListBox1.Items.Objects[Index] as TLabel);
    with myLabel.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myLabel.Canvas.MoveTo(Left, Top);
      myLabel.Canvas.LineTo(Right, Bottom);
    end;
  myLabel:= nil;
  end;
{
The following code draws the graphic in Picture1 in the
top-left corner PaintBox1.  Place a TPaintBox in the form
and a TImage in the form that contains a picture.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
end;
```

30.1.13.4.4 TCustomLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
```

```
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
```

```
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

30.1.13.4.5 TCustomLabel.EllipsisPosition Property

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
property EllipsisPosition: TEllipsisPosition;
```

C++

```
__property TEllipsisPosition EllipsisPosition;
```

Description

Set EllipsisPosition to determine how an ellipsis is placed in text that does not fit in its allocated rectangle. This is a TEllipsisPosition (see page 3001) constant and its value corresponds to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

See Also

WordWrap (see page 2397)

30.1.13.4.6 TCustomLabel.FocusControl Property

Designates a windowed control associated with the label.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Set FocusControl to the windowed control that should receive focus when the user presses the accelerator key specified by the label. Specify an accelerator key by preceding a character in the label text with an ampersand (&) and setting the ShowAccelChar (see page 2396) property to true.

See Also

ShowAccelChar (see page 2396)

GetLabelText (see page 2386)

30.1.13.4.7 TCustomLabel.GlowSize Property

Specifies the radius of the glow around the label.

Pascal

```
property GlowSize: Integer;
```

C++

```
__property int GlowSize;
```

Description

Set the value of GlowSize to specify the radius of the glow around the label.

Note: This is available only on Windows Vista with the Aero theme enabled.

See Also

Hint

Transparent (see page 2397)

Alignment (see page 2388)

30.1.13.4.8 TCustomLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout: TTextLayout;
```

C++

```
__property TTextLayout Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment (see page 2388) property.

See Also

Alignment (see page 2388)

ClientRect

30.1.13.4.9 TCustomLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl (see page 2395) property is set, the windowed control specified by the FocusControl (see page 2395) property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl (see page 2395) property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
```

```
  label also includes an ampersand, but it does not appear as
  an accelerator character.
  */
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Label1->ShowAccelChar = true;
    Label1->Caption = "An &Underlined character appears here";
    Label2->ShowAccelChar = false;
    Label2->Caption = "An ampersand (&) appears here";
  }
```

Delphi Examples:

```
  {
  This example uses two labels on a form. The first label has
  a caption with an accelerator character in it. The second
  label also includes an ampersand, but it does not appear as
  an accelerator character.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Label1.ShowAccelChar := True;
    Label1.Caption := 'An &Underlined character appears here';
    Label2.ShowAccelChar := False;
    Label2.Caption := 'An ampersand (&) appears here';
  end;
```

See Also

FocusControl (see page 2395)

GetLabelText (see page 2386)

30.1.13.4.10 TCustomLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color

ParentColor

30.1.13.4.11 TCustomLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth

GetLabelText (see page 2386)

30.1.14 TCustomListBox Class

TCustomListBox is the base class for listbox objects that are wrappers for list box controls.

Class Hierarchy

Pascal

```
TCustomListBox = class(TCustomMultiSelectListControl);
```

C++

```
class TCustomListBox : public TCustomMultiSelectListControl;
```

File

StdCtrls

Description

TCustomListBox encapsulates behavior common to all list box controls. List box objects are used to display a scrollable list from which users can select one or more items. Use a list box control to limit the user to selecting from a list of items. Items (see page 2429) can be added at design time or at runtime.

As a wrapper for a list box TCustomListBox introduces methods for:

Displaying, sorting, selecting, manipulating, and querying items in the list.

Creating owner-draw style list boxes that can display graphical images such as icons and bitmaps.

Creating multiple-column and multiple-selection lists.

By default, TCustomListBox is a standard list box that displays items as text strings. These items are contained in a TStrings object, and the methods of the TStrings object are used to add, insert or delete a string. The following code demonstrates adding a new string to the list box:

MyListBox.Items.Add('New (see page 3619) item'); //Delphi

MyListBox->Items (see page 2429)->Add("New (see page 3619) item"); // C++

Descendants of TCustomListBox inherit the basic functionality of TCustomListBox. Some descendants also have specialized behavior and can be linked to file, directory, drive, or database information. Use these list boxes to easily access and list this type of information.

A combo box is a list box with an edit control and an optional drop down style for the list. Use a combo box to allow users to type their selection into an edit box or to save space on a form by having the list drop down.

For lists of items that require a matrix or expandable tree-like structure, use list views or tree views. These controls allow the grouping of items under column headings or in expandable and contractible rows.

See Also

TStrings

30.1.14.1 TCustomListBox Members

The following tables list the members exposed by TCustomListBox.

Protected Events

	Name	Description
	OnData (see page 2438)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2439)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2439)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDrawItem (see page 2439)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnMeasureItem (see page 2442)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Protected Fields

	Name	Description
	FMoving (see page 2401)	Specifies whether the strings of the list box are changing their position.

Protected Methods

	Name	Description
V	CreateParams (see page 2404)	Initializes the window-creation parameter record passed in the Params parameter.
V	CreateWnd (see page 2405)	Creates a Windows control corresponding to the list box component.
	DeleteString (see page 2406)	Deletes a string from the list box.
V	DestroyWnd (see page 2406)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoFindData (see page 2407)	Returns the index of a string in the list box.
	DoGetData (see page 2407)	Returns a string in the list box given its index.
	DoGetDataObject (see page 2408)	Returns the object associated with an item in a virtual list box.
V	DragCanceled (see page 2408)	Sends a mouse message canceling the drag-and-drop operation.
V	DrawItem (see page 2408)	Generates an OnDrawItem (see page 2439) event.
	GetItemData (see page 2409)	Returns a 32-bit value associated with the specified item in a list box.
V	GetItemIndex (see page 2410)	Returns the value of the ItemIndex property.
V	GetSelCount (see page 2410)	Returns the number of selected items.
	GetSelected (see page 2410)	Indicates whether a particular item is selected.
	InternalGetItemData (see page 2411)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 2411)	Associates an integer with a specified item in the list box.
V	KeyPress (see page 2415)	Responds when the user presses a key.
V	LoadRecreateItems (see page 2415)	Assigns new strings to the list.
V	MeasureItem (see page 2415)	Generates an OnMeasureItem (see page 2442) event.
	ResetContent (see page 2416)	Removes all of the items from a list box.
V	SaveRecreateItems (see page 2416)	Saves the items from a list box.
	SetItemData (see page 2417)	Sets the 32-bit value associated with the specified item in a list box.

	SetItemIndex (see page 2417)	Gives focus to the item with a specified index.
	SetMultiSelect (see page 2417)	Sets the MultiSelect property.
	WndProc (see page 2418)	Handles Windows messages.

Protected Properties

	Name	Description
	BorderStyle (see page 2423)	Determines whether the list box has a border.
	Columns (see page 2426)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	ExtendedSelect (see page 2427)	Determines if the user can select a sequential range of items in the list box.
	IntegralHeight (see page 2428)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2428)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 2431)	Determines where a control looks for its color information.
	Sorted (see page 2434)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2436)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 2437)	Specifies the size of the tabs in the list box.

Public Methods

	Name	Description
	AddItem (see page 2402)	Adds an item to the list box
	Clear (see page 2403)	Deletes all items from the list box.
	ClearSelection (see page 2403)	Deselects all selected items in the list box, if possible.
	CopySelection (see page 2404)	Copies the selected items to another list-type control
	Create (see page 2400)	Instantiates a list box control.
	DeleteSelected (see page 2406)	Deletes the selected items from the list box
	Destroy (see page 2401)	Destroys the instance of a list box control.
	GetCount (see page 2409)	Returns the value of the Count (see page 2427) property.
	ItemAtPos (see page 2411)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 2413)	Returns the rectangle that surrounds the item specified in the Item parameter.
	SelectAll (see page 2416)	Selects the entire block of text in the list box.

Public Properties

	Name	Description
	AutoComplete (see page 2423)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2423)	Specifies the delay between a key press and an attempt to autocomplete the field.
R	Canvas (see page 2424)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2439) event of an owner-draw list box.
	Count (see page 2427)	Indicates the number of items in the list box.
	Items (see page 2429)	Contains the strings that appear in the list box.
	ScrollWidth (see page 2432)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	Selected (see page 2432)	Indicates whether a particular item is selected.
	TopIndex (see page 2437)	Specifies the index number of the item that appears at the top of the list box.

Published Properties

	Name	Description
	TabStop (see page 2436)	Determines if the user can tab to a control.

30.1.14.2 TCustomListBox.Create Constructor

Instantiates a list box control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomListBox(TComponent * AOwner);
```

Description

Call Create to instantiate a list box control at runtime. For list boxes created at design time, Create is called automatically.AOwner is the component that is responsible for freeing the list box. It becomes the value of the Owner property. If AOwner is nil, the caller is responsible for explicitly freeing the control.

See Also

Owner

30.1.14.3 TCustomListBox.Destroy Destructor

Destroys the instance of a list box control.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomListBox();
```

Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free checks that the list box reference is not nil before it calls Destroy.

Destroy frees the memory associated with the objects for the Canvas (see page 2424) and Items (see page 2429) properties. Then it calls the inherited destructor to destroy the list box object.

See Also

System::TObject::Free (see page 3372)

30.1.14.4 TCustomListBox Fields

The fields of the TCustomListBox class are listed here.

Protected Fields

	Name	Description
	FMoving (see page 2401)	Specifies whether the strings of the list box are changing their position.

30.1.14.4.1 TCustomListBox.FMoving Field

Specifies whether the strings of the list box are changing their position.

Pascal

```
FMoving: Boolean;
```

C++

```
Boolean FMoving;
```

Description

FMoving is an internal field specifying whether the strings of the list box are moving inside the list box.

See Also

Move

30.1.14.5 TCustomListBox Methods

The methods of the TCustomListBox class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2404)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateWnd (see page 2405)	Creates a Windows control corresponding to the list box component.
	DeleteString (see page 2406)	Deletes a string from the list box.
	DestroyWnd (see page 2406)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoFindData (see page 2407)	Returns the index of a string in the list box.
	DoGetData (see page 2407)	Returns a string in the list box given its index.
	DoGetDataObject (see page 2408)	Returns the object associated with an item in a virtual list box.
	DragCanceled (see page 2408)	Sends a mouse message canceling the drag-and-drop operation.
	DrawItem (see page 2408)	Generates an OnDrawItem (see page 2439) event.
	GetItemData (see page 2409)	Returns a 32-bit value associated with the specified item in a list box.
	GetItemIndex (see page 2410)	Returns the value of the ItemIndex property.
	GetSelCount (see page 2410)	Returns the number of selected items.
	GetSelected (see page 2410)	Indicates whether a particular item is selected.
	InternalGetItemData (see page 2411)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 2411)	Associates an integer with a specified item in the list box.
	KeyPress (see page 2415)	Responds when the user presses a key.
	LoadRecreateItems (see page 2415)	Assigns new strings to the list.
	MeasureItem (see page 2415)	Generates an OnMeasureItem (see page 2442) event.
	ResetContent (see page 2416)	Removes all of the items from a list box.
	SaveRecreateItems (see page 2416)	Saves the items from a list box.
	SetItemData (see page 2417)	Sets the 32-bit value associated with the specified item in a list box.
	SetItemIndex (see page 2417)	Gives focus to the item with a specified index.
	SetMultiSelect (see page 2417)	Sets the MultiSelect property.
	WndProc (see page 2418)	Handles Windows messages.

Public Methods

	Name	Description
	AddItem (see page 2402)	Adds an item to the list box
	Clear (see page 2403)	Deletes all items from the list box.
	ClearSelection (see page 2403)	Deselects all selected items in the list box, if possible.
	CopySelection (see page 2404)	Copies the selected items to another list-type control
	DeleteSelected (see page 2406)	Deletes the selected items from the list box
	GetCount (see page 2409)	Returns the value of the Count (see page 2427) property.
	ItemAtPos (see page 2411)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 2413)	Returns the rectangle that surrounds the item specified in the Item parameter.
	SelectAll (see page 2416)	Selects the entire block of text in the list box.

30.1.14.5.1 TCustomListBox.AddItem Method

Adds an item to the list box

Pascal

```
procedure AddItem(Item: String; AObject: TObject); override;
```

C++

```
virtual __fastcall AddItem(UnicodeString Item, TObject * AObject);
```

Description

Call AddItem to add a new item to the end of the list.

Item is the text of the item to add.

AObject is the object associated with the new item.

See Also

AddObject

Items (see page 2429)

30.1.14.5.2 TCustomListBox.Clear Method

Deletes all items from the list box.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Use Clear to delete all of the items in the list box at once. Although the items in a standard list box are of type TStrings, use the Clear method of the list box rather than calling the Clear method of the Items (see page 2429) property. This allows descendants of TCustomListBox (see page 2398) to perform any other necessary clean-up in the Clear method, in addition to deleting the items from the Items (see page 2429) property. Thus, applications should use

```
ListBox1.Clear;
ListBox1->Clear();
```

rather than

```
ListBox1.Items.Clear;
ListBox1->Items->Clear();
```

See Also

Clear

Items (see page 2429)

30.1.14.5.3 TCustomListBox.ClearSelection Method

Deselects all selected items in the list box, if possible.

Pascal

```
procedure ClearSelection; override;
```

C++

```
virtual __fastcall ClearSelection();
```

Description

Use ClearSelection to deselect all the currently selected items in the list box. When the list box receives focus, after ClearSelection is called, the last item that was selected is the item that receives the current focus in the list box. Although this item receives focus, the item itself is not selected.

See Also

DeleteSelected (see page 2406)

ExtendedSelect (see page 2427)

MultiSelect

SelCount

Selected (see page 2432)

ItemIndex

30.1.14.5.4 TCustomListBox.CopySelection Method

Copies the selected items to another list-type control

Pascal

```
procedure CopySelection(Destination: TCustomListControl); override;
```

C++

```
virtual __fastcall CopySelection(TCustomListControl Destination);
```

Description

Call CopySelection to copy all the items selected in the list box to a combo box, list view, or another list box. If the destination control is not sorted, the selected items are added to the end of its list.

Destination is the list box, combo box, or list view to which the selected items are copied.

See Also

MoveSelection

ExtendedSelect (see page 2427)

MultiSelect

SelCount

Selected (see page 2432)

ItemIndex

30.1.14.5.5 TCustomListBox.CreateParams Method

Initializes the window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams is used to pass information, in its Params parameter, to CreateWindowHandle when the list box control is created.

TCustomListBox (see page 2398) overrides CreateParams to customize the representation of the window. It uses information from the list box Columns (see page 2426), ExtendedSelect (see page 2427), IntegralHeight (see page 2428), MultiSelect, Sorted (see page 2434) and Style (see page 2436) properties and sets WS_HSCROLL, WS_VSCROLL, LBS_HASSTRINGS and LBS_NOTIFY Windows styles for creation of the list box.

When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

See Also

Columns (see page 2426)

CreateWindowHandle

CreateWnd

ExtendedSelect (see page 2427)

IntegralHeight (see page 2428)

MultiSelect

Sorted (see page 2434)

Style (see page 2436)

30.1.14.5.6 TCustomListBox.CreateWnd Method

Creates a Windows control corresponding to the list box component.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the list box must be displayed. CreateWnd calls the inherited CreateWnd method, which calls

CreateParams (see page 2404) to initialize the window-creation parameters.

CreateWindowHandle to create the window handle for the control.

CreateWnd additionally sets up the list box position, tabs, items, and columns.

See Also

Columns (see page 2426)

CreateHandle

CreateParams (see page 2404)

CreateWindowHandle

CreateWnd

ItemIndex

Items (see page 2429)

TabWidth (see page 2437)

TopIndex (see page 2437)

30.1.14.5.7 TCustomListBox.DeleteSelected Method

Deletes the selected items from the list box

Pascal

```
procedure DeleteSelected; override;
```

C++

```
virtual __fastcall DeleteSelected();
```

Description

Call DeleteSelected to delete the selected items from the list box.

See Also

ClearSelection

MoveSelection

CopySelection (see page 2404)

ExtendedSelect (see page 2427)

MultiSelect

SelCount

Selected (see page 2432)

ItemIndex

30.1.14.5.8 TCustomListBox.DeleteString Method

Deletes a string from the list box.

Pascal

```
procedure DeleteString(Index: Integer); dynamic;
```

C++

```
__fastcall DeleteString(int Index);
```

Description

DeleteString deletes the string, specified by the Index parameter, from the list box control.

30.1.14.5.9 TCustomListBox.DestroyWnd Method

Saves the list box items in a string list before freeing the underlying Windows control.

Pascal

```
procedure DestroyWnd; override;
```

C++

```
virtual __fastcall DestroyWnd();
```

Description

DestroyWnd saves the list box text string items in the Items (see page 2429) property to internal storage in a TStringList object that it creates. It then calls the inherited DestroyWnd method to destroy the windowed control's window handle and free associated device contexts.

See Also

DestroyWnd

Items (see page 2429)

TStringList

30.1.14.5.10 TCustomListBox.DoFindData Method

Returns the index of a string in the list box.

Pascal

```
function DoFindData(const Data: String): Integer;
```

C++

```
__fastcall int DoFindData(const UnicodeString Data);
```

Description

This protected method is called automatically when the list box needs to indicate the index for an item.

If the list box is virtual (that is, if the value of the Style (see page 2436) property is lbVirtual or lbVirtualOwnerDraw), DoFindData generates an OnDataFind (see page 2439) event so that the index can be provided dynamically. If the list box is not virtual (any other Style (see page 2436)), DoFindData locates the specified string from the Items (see page 2429) property and returns its index.

Data is the text of the item to locate.

See Also

OnDataFind (see page 2439)

Style (see page 2436)

Items (see page 2429)

30.1.14.5.11 TCustomListBox.DoGetData Method

Returns a string in the list box given its index.

Pascal

```
function DoGetData(const Index: Integer): String;
```

C++

```
__fastcall UnicodeString DoGetData(const int Index);
```

Description

This protected method is called automatically when a virtual list box needs to supply the text of an item. DoGetData is never called unless the list box is virtual (that is, unless the value of the Style (see page 2436) property is lbVirtual or lbVirtualOwnerDraw).

DoGetData generates an OnData (see page 2438) event and returns the string that is supplied dynamically by the event handler.

See Also

OnData (see page 2438)

Style (see page 2436)

Items (see page 2429)

30.1.14.5.12 TCustomListBox.DoGetDataObject Method

Returns the object associated with an item in a virtual list box.

Pascal

```
function DoGetDataObject(const Index: Integer): TObject;
```

C++

```
__fastcall TObject * DoGetDataObject(const int Index);
```

Description

This protected method is called automatically when a virtual list box needs to supply the object associated with an item. DoGetDataObject is never called unless the list box is virtual (that is, unless the value of the Style (see page 2436) property is lbVirtual or lbVirtualOwnerDraw).

DoGetDataObject generates an OnDataObject (see page 2439) event and returns the object that is supplied dynamically by the event handler.

See Also

OnDataObject (see page 2439)

Style (see page 2436)

Items (see page 2429)

30.1.14.5.13 TCustomListBox.DragCanceled Method

Sends a mouse message canceling the drag-and-drop operation.

Pascal

```
procedure DragCanceled; override;
```

C++

```
virtual __fastcall DragCanceled();
```

Description

DragCanceled is called in response to the cancellation of a drag-and-drop operation that had started. Users can cancel the operation by dropping the item over a component that does not accept the item, or by pressing Esc while dragging.

The OnEndDrag event occurs after the call to DragCanceled.

See Also

OnEndDrag

30.1.14.5.14 TCustomListBox.DrawItem Method

Generates an OnDrawItem (see page 2439) event.

Pascal

```
procedure DrawItem(Index: Integer; Rect: TRect; State: TOwnerDrawState); virtual;
```

C++

```
virtual __fastcall DrawItem(int Index, TRect Rect, TOwnerDrawState State);
```

Description

DrawItem is called whenever a visual aspect of an owner-draw list box changes. The list box calls DrawItem for each visible item in its list, passing the index of the item in the Index parameter.

By default, the DrawItem method for a list box calls any event handler attached to the component's OnDrawItem (see page 2439) event. If there is no handler attached, DrawItem fills the rectangle passed in Rect and draws any text associated with the indexed item. The default drawing ignores the State parameter, although State is passed along to attached event handlers.

See Also

ItemHeight (see page 2428)

MeasureItem (see page 2415)

OnDrawItem (see page 2439)

OnMeasureItem (see page 2442)

Style (see page 2436)

30.1.14.5.15 TCustomListBox.GetCount Method

Returns the value of the Count (see page 2427) property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count (see page 2427) property.

See Also

Count (see page 2427)

30.1.14.5.16 TCustomListBox.GetItemData Method

Returns a 32-bit value associated with the specified item in a list box.

Pascal

```
function GetItemData(Index: Integer): TCustomLongData; dynamic;
```

C++

```
__fastcall TCustomLongData GetItemData(int Index);
```

Description

GetItemData retrieves the data associated with a particular item in the list box. This value can be displayed as that list box item, or can be a value (such as a pointer) associated with the list box item.

See Also

SetItemData (see page 2417)

30.1.14.5.17 TCustomListBox.GetItemIndex Method

Returns the value of the ItemIndex property.

Pascal

```
function GetItemIndex: Integer; override;
```

C++

```
virtual __fastcall int GetItemIndex();
```

Description

GetItemIndex is the protected read implementation of the ItemIndex property. It returns the index of the first selected item in the list box.

The first item in the list has index 0, the second item has index 1, and so on. If no item is selected, the value of ItemIndex is -1.

See Also

ItemIndex

30.1.14.5.18 TCustomListBox.GetSelCount Method

Returns the number of selected items.

Pascal

```
function GetSelCount: Integer; override;
```

C++

```
virtual __fastcall int GetSelCount();
```

Description

GetSelCount is the protected read implementation of the SelCount property.

See Also

SelCount

30.1.14.5.19 TCustomListBox.GetSelected Method

Indicates whether a particular item is selected.

Pascal

```
function GetSelected(Index: Integer): Boolean;
```

C++

```
__fastcall Boolean GetSelected(int Index);
```

Description

GetSelected is the protected read implementation of the Selected (see page 2432) property. It returns true if the item specified by Index is selected, false if it is not.

30.1.14.5.20 TCustomListBox.InternalGetItemData Method

Retrieves the data associated with an item in the list box.

Pascal

```
function InternalGetItemData(Index: Integer): TCustomLongData; dynamic;
```

C++

```
__fastcall TCustomLongData InternalGetItemData(int Index);
```

Description

InternalGetItemData is the low-level protected method for fetching the data associated with an item in the list box. It retrieves a reference pointer to the object associated with the specified item, as an integer.

Index is the index of the item whose data is to be fetched.

See Also

GetItemData (see page 2409)

30.1.14.5.21 TCustomListBox.InternalSetItemData Method

Associates an integer with a specified item in the list box.

Pascal

```
procedure InternalSetItemData(Index: Integer; AData: TCustomLongData); dynamic;
```

C++

```
__fastcall InternalSetItemData(int Index, TCustomLongData AData);
```

Description

InternalSetItemData is the low-level protected method for associating data with an item in the list box. It associates the item that has the index specified by Index with the integer specified by AData.

See Also

SetItemData (see page 2417)

30.1.14.5.22 TCustomListBox.ItemAtPos Method

Returns the index of the list box item indicated by the coordinates of a point on the control.

Pascal

```
function ItemAtPos(Pos: TPoint; Existing: Boolean): Integer;
```

C++

```
__fastcall int ItemAtPos(TPoint Pos, Boolean Existing);
```

Description

Use ItemAtPos to detect if an item exists at a particular point in the control.

The Pos (see page 3627) parameter is the point in the control in window coordinates. If Pos (see page 3627) is beyond the last item in the list box, the value of the Existing variable determines the returned value. If Existing is set to true, ItemAtPos returns -1, indicating that no item exists at that point. If Existing is set to false, ItemAtPos returns the index of the last item in the list box plus one.

C++ Examples:

```
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  dynamic_cast<TObject *>(Components[i]));
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
        break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to Form1.ComponentCount-1 do
    ListBox1.Items.InsertObject(
      0, Form1.Components[I].Name,
      Form1.Components[I] as TObject);
end;

procedure TForm1.ListBox1MouseUp(
  Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X,Y: integer);
var
  APoint: TPoint;
  Index: integer;
  TheObject: TControl;
begin
  if Button = mbRight then
  begin
  APoint.X := X;
  APoint.Y := Y;
  Index := ListBox1.ItemAtPos(APoint, True);
  if (ListBox1.Items.Objects[Index] is TControl) then
  begin
    TheObject :=
      (ListBox1.Items.Objects[Index] as TControl);
    StatusBar1.SimpleText := TheObject.Name + ' is at (' +
                             IntToStr(TheObject.Left) + ', ' +
                             IntToStr(TheObject.Top) + ') ';
  end
  else
    Beep;
  end;
end;
```

See Also

ItemIndex

Items (see page 2429)

30.1.14.5.23 TCustomListBox.ItemRect Method

Returns the rectangle that surrounds the item specified in the Item parameter.

Pascal

```
function ItemRect(Index: Integer): TRect;
```

C++

```
__fastcall TRect ItemRect(int Index);
```

Description

Use ItemRect to get the coordinates of a particular item in the list box. For example, ItemRect is used internally by the ItemAtPos (see page 2411) method.

The Item parameter is the ItemIndex of the item whose position is queried.

C++ Examples:

```
/*
This example uses a list box on a form. When the user clicks
one of the items in the list box, a beep occurs if the mouse
```

```
is in the first half of the item. A similar technique can be
used to detect "hot" regions in an owner-draw list box:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Hello");
  ListBox1->Items->Add("New");
  ListBox1->Items->Add("World");
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  TRect ItemBounds;
  if (Button == mbLeft)
  {
    ItemBounds = ListBox1->ItemRect(ListBox1->ItemIndex);
    if (Y > ItemBounds.Top &&
        Y < ItemBounds.Bottom &&
        X > ItemBounds.Left &&
        X < (ItemBounds.Left + ItemBounds.Right)/2)
      MessageBeep(0);
  }
}
```

Delphi Examples:

```
{
This example uses a list box on a form. When the user clicks
one of the items in the list box, a beep occurs if the mouse
is in the first half of the item. A similar technique can be
used to detect "hot" regions in an owner-draw list box:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  with ListBox1 do
  begin
    Items.Add('Hello');
    Items.Add('New');
    Items.Add('World');
  end;
end;

procedure TForm1.ListBox1MouseUp (Sender: TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
var
  ListBoxItem: TRect;
begin
  if Button = mbLeft then
  begin
    ListBoxItem := ListBox1.ItemRect(ListBox1.ItemIndex);
    if (Y > ListBoxItem.Top) and
       (Y < ListBoxItem.Bottom) and
       (X > ListBoxItem.Left) and
       (X < (ListBoxItem.Left + ListBoxItem.Right) div 2) then
      Beep;
  end;
 end;
```

See Also

ItemAtPos (see page 2411)

ItemIndex

30.1.14.5.24 TCustomListBox.KeyPress Method

Responds when the user presses a key.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the list box receives a key-press message (WM_CHAR). It generates an OnKeyPress event, and then, if the AutoComplete (see page 2423) property is true, updates ItemIndex to reflect the next character that the user typed.

The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

See Also

OnKeyPress

30.1.14.5.25 TCustomListBox.LoadRecreateItems Method

Assigns new strings to the list.

Pascal

```
procedure LoadRecreateItems(RecreateItems: TStrings); virtual;
```

C++

```
virtual __fastcall LoadRecreateItems(TStrings RecreateItems);
```

Description

LoadRecreateItems is used to set the strings from the list to the strings of the `RecreateItems` parameter.

See Also

Items (see page 2429)

Assign

30.1.14.5.26 TCustomListBox.MeasureItem Method

Generates an OnMeasureItem (see page 2442) event.

Pascal

```
procedure MeasureItem(Index: Integer; var Height: Integer); virtual;
```

C++

```
virtual __fastcall MeasureItem(int Index, int Height);
```

Description

MeasureItem is called whenever a visual aspect of a variable owner-draw list box changes and the value of the Style (see page 2436) property is csOwnerDrawVariable.

Before drawing its items, a variable owner-draw list box calls MeasureItem once for each visible item, passing the index of the item to measure, and the default height of the item. Since Height is passed by reference (a var parameter), MeasureItem can

increase or reduce the height of each item as needed.

By default, the MeasureItem method does nothing except call any event handler attached to the OnMeasureItem (see page 2442) event. Override MeasureItem to change or add functionality to the default behavior.

See Also

DrawItem (see page 2408)

ItemHeight (see page 2428)

ItemIndex

OnDrawItem (see page 2439)

OnMeasureItem (see page 2442)

Style (see page 2436)

30.1.14.5.27 TCustomListBox.ResetContent Method

Removes all of the items from a list box.

Pascal

```
procedure ResetContent; dynamic;
```

C++

```
__fastcall ResetContent();
```

Description

ResetContent is used to remove the items from the list box. ResetContent is used, for example, by the Clear (see page 2403) method.

See Also

Clear (see page 2403)

30.1.14.5.28 TCustomListBox.SaveRecreateItems Method

Saves the items from a list box.

Pascal

```
procedure SaveRecreateItems(RecreateItems: TStrings); virtual;
```

C++

```
virtual __fastcall SaveRecreateItems(TStrings RecreateItems);
```

Description

SaveRecreateItems is used to save the values and the properties of the items in a TStrings variable.

See Also

Items (see page 2429)

Assign

30.1.14.5.29 TCustomListBox.SelectAll Method

Selects the entire block of text in the list box.

Pascal

```
procedure SelectAll; override;
```

C++

```
virtual __fastcall SelectAll();
```

Description

Use SelectAll to select the entire block of text in the list box. To select only part of the text, use the SelStart and SelLength properties.

30.1.14.5.30 TCustomListBox.SetItemData Method

Sets the 32-bit value associated with the specified item in a list box.

Pascal

```
procedure SetItemData(Index: Integer; AData: TCustomLongData); dynamic;
```

C++

```
__fastcall SetItemData(int Index, TCustomLongData AData);
```

Description

Use SetItemData to set the data associated with a particular item in a list box. This value can be displayed as that list box item, or it can be a value (such as a pointer) represented by the list box item.

See Also

GetItemData (see page 2409)

30.1.14.5.31 TCustomListBox.SetItemIndex Method

Gives focus to the item with a specified index.

Pascal

```
procedure SetItemIndex(const Value: Integer); override;
```

C++

```
virtual __fastcall SetItemIndex(const int Value);
```

Description

SetItemIndex is the protected write implementation for the ItemIndex property. Value is the index of the item that should receive focus.

See Also

ItemIndex

30.1.14.5.32 TCustomListBox.SetMultiSelect Method

Sets the MultiSelect property.

Pascal

```
procedure SetMultiSelect(Value: Boolean); override;
```

C++

```
virtual __fastcall SetMultiSelect(Boolean Value);
```

Description

SetMultiSelect is the protected write implementation for the MultiSelect property. Value is true if the list box should allow more than one item to be selected, false if only one item can be selected at a time.

See Also

MultiSelect

30.1.14.5.33 TCustomListBox.WndProc Method

Handles Windows messages.

Pascal

```
procedure WndProc(var Message: TMessage); override;
```

C++

```
virtual __fastcall WndProc(TMessage Message);
```

Description

WndProc overrides the inherited WndProc to provide specialized responses to messages that allows the list box to handle auto drag mode itself. Then it calls the inherited WndProc to handle all other messages.

When overriding the WndProc method, be sure to call the inherited WndProc at the end to dispatch any other messages.

C++ Examples:

```
/*
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
*/
class TMyListBoxDescendant : public TListBox
{
__published:    // IDE-managed Components
    void __fastcall SubClassWndProc(Messages::TMessage &Message);
    void __fastcall ToggleSubClass(bool On);
    void __fastcall OnDrawItemProc(
      TWinControl *Control, int Index, const TRect &Rect,
      TOwnerDrawState State);
private:    // User declarations
public:     // User declarations
    __fastcall TMyListBoxDescendant(TComponent* Owner);
};

TForm1 *Form1;
TMyListBoxDescendant *MyListBoxDescendant1;
Graphics::TBitmap *bitmap0;

const WM_STYLEMESSAGE = WM_USER + 2000;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

__fastcall TMyListBoxDescendant::TMyListBoxDescendant(TComponent* Owner)
    : TListBox(Owner)
{
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  PostMessage(
    MyListBoxDescendant1->Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
}

void __fastcall TForm1::SubClassRadioGroup1Click(TObject *Sender)
{
  MyListBoxDescendant1->ToggleSubClass(SubClassRadioGroup1->ItemIndex == 0);
}

void __fastcall TMyListBoxDescendant::SubClassWndProc(Messages::TMessage &Message)
{
  if (Message.Msg == WM_STYLEMESSAGE)
    Style = (TListBoxStyle)Message.WParam;
  else
    WndProc(Message);
}

void __fastcall TMyListBoxDescendant::ToggleSubClass(bool On)
{
  if (On)
    WindowProc = SubClassWndProc;
  else
    WindowProc = WndProc;
}

#include <memory>       //for STL auto_ptr class

void __fastcall TMyListBoxDescendant::OnDrawItemProc(TWinControl *Control, int Index,
      const TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *bitmap; // temporary variable for item's bitmap
  int Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *canvas = dynamic_cast<TListBox *>(Control)->Canvas;
  canvas->FillRect(Rect); // clear the rectangle
  bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap)
  {
    canvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, bitmap->Width, bitmap->Height),
      bitmap, Bounds(0, 0, bitmap->Width, bitmap->Height), clRed); // render bitmap
    Offset += bitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  canvas->TextOut(Rect.Left + Offset, Rect.Top, dynamic_cast<TListBox
*>(Control)->Items->Strings[Index]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  MyListBoxDescendant1 = new TMyListBoxDescendant(Form1); // The owner cleans this up.
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Parent = Form1;
  MyListBoxDescendant1->Visible = True;
  MyListBoxDescendant1->Left =
    SubClassRadioGroup1->Left + SubClassRadioGroup1->Width + 30;;
  MyListBoxDescendant1->Top = SubClassRadioGroup1->Top;
  MyListBoxDescendant1->Height = SubClassRadioGroup1->Height;
  MyListBoxDescendant1->OnDrawItem =
    MyListBoxDescendant1->OnDrawItemProc;

  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  ImageList1->GetBitmap(0, bitmap0);
  MyListBoxDescendant1->Items->AddObject("Butterfly", bitmap0);

  SubClassRadioGroup1->Items->Add("SubClassWndProc");
  SubClassRadioGroup1->Items->Add("WndProc");
  SubClassRadioGroup1->ItemIndex = 2;
}
```

Delphi Examples:

```
{
This example shows how to use the WndProc method and the
WindowProc property to subclass a custom control's window
procedure.  This example subclasses the window procedure of
a TListBox descendant to respond to a user-defined message
called WM_STYLEMESSAGE. The subclassed window procedure can
be turned on or off by pressing a radio button.
}

type
  TMyListBoxDescendant = class(TlistBox)
    procedure SubClassWndProc(var Message: TMessage);
    procedure ToggleSubClass(On: Boolean);
    procedure OnDrawItemProc(
      Control: TWinControl;
      Index: Integer;
      Rect:TRect;
      State: TOwnerDrawState);
  end;
  TForm1 = class(TForm)
    SubClassRadioGroup1: TRadioGroup;
    Button1: TButton;
    ImageList1: TImageList;
    Button2: TButton;
    procedure SubClassRadioGroup1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyListBoxDescendant1: TMyListBoxDescendant;
  bitmap0: TBitmap;

implementation

{$R *.dfm}

const WM_STYLEMESSAGE = WM_USER + 2000;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbOwnerDrawFixed),
    0);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  PostMessage(
    MyListBoxDescendant1.Handle,
    WM_STYLEMESSAGE,
    Integer(lbStandard),
    0);
end;

procedure TMyListBoxDescendant.SubClassWndProc(var Message: TMessage);
begin
  if (Message.Msg = WM_STYLEMESSAGE) then
    Style:= TListBoxStyle(Message.WParam)
  else
    WndProc(Message);
end;

procedure TMyListBoxDescendant.ToggleSubClass(On: Boolean);
begin
  if On then
    WindowProc := SubClassWndProc
  else
    WindowProc := WndProc;
end;

procedure TForm1.SubClassRadioGroup1Click(Sender: TObject);
begin
  MyListBoxDescendant1.ToggleSubClass(
    SubClassRadioGroup1.ItemIndex = 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MyListBoxDescendant1:= TMyListBoxDescendant.Create(self);
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Parent:= Form1;
  MyListBoxDescendant1.Visible:= True;
  MyListBoxDescendant1.Left:=
    SubClassRadioGroup1.Left + SubClassRadioGroup1.Width + 30;;
  MyListBoxDescendant1.Top:= SubClassRadioGroup1.Top;
  MyListBoxDescendant1.Height:= SubClassRadioGroup1.Height;
  MyListBoxDescendant1.OnDrawItem:=
    MyListBoxDescendant1.OnDrawItemProc;

  bitmap0 := TBitmap.Create;
  ImageList1.GetBitmap(0, bitmap0);
  MyListBoxDescendant1.Items.AddObject('Butterfly', bitmap0);

  SubClassRadioGroup1.Items.Add('SubClassWndProc');
  SubClassRadioGroup1.Items.Add('WndProc');
  SubClassRadioGroup1.ItemIndex := 2;
end;

procedure TMyListBoxDescendant.OnDrawItemProc(
  Control: TWinControl;
  Index: Integer;
```

```
    Rect:TRect;
    State: TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    { draw on control canvas, not on the form }
    with (Control as TListBox).Canvas do
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

WndProc

30.1.14.6 TCustomListBox Properties

The properties of the TCustomListBox class are listed here.

Protected Properties

	Name	Description
	BorderStyle (see page 2423)	Determines whether the list box has a border.
	Columns (see page 2426)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	ExtendedSelect (see page 2427)	Determines if the user can select a sequential range of items in the list box.
	IntegralHeight (see page 2428)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2428)	Specifies the height in pixels of an item in an owner-draw list box.
	ParentColor (see page 2431)	Determines where a control looks for its color information.
	Sorted (see page 2434)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2436)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabWidth (see page 2437)	Specifies the size of the tabs in the list box.

Public Properties

	Name	Description
	AutoComplete (see page 2423)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2423)	Specifies the delay between a key press and an attempt to autocomplete the field.
R	Canvas (see page 2424)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2439) event of an owner-draw list box.
	Count (see page 2427)	Indicates the number of items in the list box.
	Items (see page 2429)	Contains the strings that appear in the list box.
	ScrollWidth (see page 2432)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	Selected (see page 2432)	Indicates whether a particular item is selected.
	TopIndex (see page 2437)	Specifies the index number of the item that appears at the top of the list box.

Published Properties

	Name	Description
	TabStop (see page 2436)	Determines if the user can tab to a control.

30.1.14.6.1 TCustomListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

30.1.14.6.2 TCustomListBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2423) is true.

See Also

AutoComplete (see page 2423)

30.1.14.6.3 TCustomListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D

30.1.14.6.4 TCustomListBox.Canvas Property

Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2439) event of an owner-draw list box.

Pascal

```
property Canvas: TCanvas;
```

C++

```
__property TCanvas Canvas;
```

Description

Canvas is only useful for applications that control the drawing of list box items, that is, for owner-draw list boxes. Canvas is used to draw graphical images that can be used as items in a list box.

By default a TControlCanvas object is created for the Canvas property in the list box constructor. To create an owner-draw list box that uses a canvas, set the Style (see page 2436) property accordingly.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
```

```
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = (dynamic_cast<TListBox *>(Control))->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
      pBitmap,
      Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}
/*
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
*/
{
  TPoint APoint = Point(X, Y);
  int Index = ListBox1->ItemAtPos(APoint, True);
  TListBox *myListBox;
  TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
  if (String(ClassRef->ClassName()) == "TListBox")
  {
    myListBox = dynamic_cast<TListBox *>(ListBox1->Items->Objects[Index]);
    // make sure we are using the TControls canvas and not the from's!
    myListBox->Canvas->MoveTo(myListBox->ClientRect.Left, myListBox->ClientRect.Top);
    myListBox->Canvas->LineTo(myListBox->ClientRect.Right, myListBox->ClientRect.Bottom);
    myListBox = NULL;
  }
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
  ImageList1.GetBitmap(2, bitmap2);
```

```
    ListBox1.Items.AddObject('Butterfly', bitmap2);
  end;

  procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
  TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    Bitmap := TBitmap.Create;
    with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
  {
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  }
    if (ListBox1.Items.Objects[Index] is TListBox) then
    begin
      myListBox:= (ListBox1.Items.Objects[Index] as TListBox);
      with myListBox.ClientRect do
      begin // make sure we are using the TControl's canvas and not the form's!
        myListBox.Canvas.MoveTo(Left, Top);
        myListBox.Canvas.LineTo(Right, Bottom);
      end;
    myListBox:= nil;
    end;
```

See Also

ItemHeight (see page 2428)

OnDrawItem (see page 2439)

Style (see page 2436)

TCanvas

TControlCanvas

30.1.14.6.5 TCustomListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
  property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 2428)

Width

30.1.14.6.6 TCustomListBox.Count Property

Indicates the number of items in the list box.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count to determine the number of items in the list box. You can only set Count when the Style (see page 2436) property is lbVirtual or lbVirtualOwnerDraw, in which case you must set this property.

See Also

Style (see page 2436)

30.1.14.6.7 TCustomListBox.ExtendedSelect Property

Determines if the user can select a sequential range of items in the list box.

Pascal

```
property ExtendedSelect: Boolean;
```

C++

```
__property Boolean ExtendedSelect;
```

Description

ExtendedSelect works with the MultiSelect property. If ExtendedSelect is true and MultiSelect is true, the user can select an item then hold down the Shift key and select another and all the items in between the two selected items also become selected.

If the user doesn't hold down the Shift or Ctrl key while selecting a second item, the first selected item becomes unselected— in other words, the user must use the Ctrl key to select multiple noncontiguous items, or the Shift key to select a range of items.

If ExtendedSelect is false and MultiSelect is true, the user can select multiple items without using the Shift or Ctrl key, but they can't select a range of items in one operation.

If MultiSelect is false, the setting of ExtendedSelect has no effect as the user will not be able to select more than one item at a time in the list box.

See Also

MultiSelect

30.1.14.6.8 TCustomListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style (see page 2436) property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight (see page 2428) property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style (see page 2436) property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 2428)

Items (see page 2429)

Style (see page 2436)

Using TDBListBox and TDBComboBox

30.1.14.6.9 TCustomListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style (see page 2436) property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight (see page 2428) property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style (see page 2436) property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 2408)

IntegralHeight (see page 2428)

Items (see page 2429)

MeasureItem (see page 2415)

OnDrawItem (see page 2439)

OnMeasureItem (see page 2442)

Style (see page 2436)

Using TDBListBox and TDBComboBox

30.1.14.6.10 TCustomListBox.Items Property

Contains the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to add, insert, delete and move items. By default, the items in a list box are of type TStrings. Use this item type to access its methods or properties to manipulate the items in the list.

For example, the following code adds the text in the edit box to the list box as an item:

```
ListBox1.Items.Add(Edit1.Text); {Delphi }
ListBox1->Items->Add(Edit1->Text); // C++
```

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
```

```
          String("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
      fclose(F);
    }
  }
}
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
```

```
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

See Also

Clear (see page 2403)

ItemIndex

OnDrawItem (see page 2439)

Selected (see page 2432)

TStrings

30.1.14.6.11 TCustomListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.14.6.12 TCustomListBox.ScrollWidth Property

Specifies the width, in pixels, by which the list box can scroll horizontally.

Pascal

```
property ScrollWidth: Integer;
```

C++

```
__property int ScrollWidth;
```

Description

Use ScrollWidth to get or set the logical width of the list box. When ScrollWidth is greater than the client width of the list box, the list box gets a horizontal scroll bar. When ScrollWidth is less than or equal to ClientWidth, the horizontal scroll bar disappears.

See Also

Columns (see page 2426)

ClientWidth

30.1.14.6.13 TCustomListBox.Selected Property

Indicates whether a particular item is selected.

Pascal

```
property Selected [Index: Integer]: Boolean;
```

C++

```
__property Boolean Selected[int Index];
```

Description

Use Selected to query the state of a particular item in the list box. If the item specified by the Index parameter is selected in the list box, the value of the Selected property is true. If the specified item is not selected, Selected is false. Set Selected to change the currently selected item.

The Index parameter is the item referenced by its position in the list box, with the first item having an Index value of 0.

An item is selected when the user highlights it. More than one item in the list box can be selected by setting the MultiSelect

property to true. In this case, the Selected item is the one that has focus when the list box has input focus.

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          String("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
      fclose(F);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
```

```
        if not FileExists(FileListBox1.Items.Strings[i]) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' not found', mtError, [mbOk], 0);
          Continue;
        end;
        filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
        if (filehandle = -1) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
     finally
     { do something here }
     end;
   end;
end;
```

See Also

ExtendedSelect (see page 2427)

Items (see page 2429)

MultiSelect

SelCount

30.1.14.6.14 TCustomListBox.Sorted Property

Specifies whether the items in a list box are arranged alphabetically.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Use Sorted to sort the items by setting its value to true. If Sorted is false, the items are unsorted. When Sorted is true, items are automatically added or inserted into the list box in alphabetical order.

Note: Sorted has no effect if Listbox's style is lbVirtual or lbVirtualOwnerDraw

C++ Examples:

```
/*
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
```

```
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}

void __fastcall TForm1::AddClick(TObject *Sender)
{
  ListBox1->Items->Add(Edit1->Text);
}

void __fastcall TForm1::SortClick(TObject *Sender)
{
  ListBox1->Sorted = True;
}
```

Delphi Examples:

```
{
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;


procedure TForm1.AddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.SortClick(Sender: TObject);
begin
  ListBox1.Sorted := True;
end;
```

See Also

Items (see page 2429)

Style (see page 2436)

30.1.14.6.15 TCustomListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 2408)

ItemHeight (see page 2428)

Items (see page 2429)

MeasureItem (see page 2415)

OnDrawItem (see page 2439)

OnMeasureItem (see page 2442)

OnData (see page 2438)

OnDataFind (see page 2439)

OnDataObject (see page 2439)

30.1.14.6.16 TCustomListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.14.6.17 TCustomListBox.TabWidth Property

Specifies the size of the tabs in the list box.

Pascal

```
property TabWidth: Integer;
```

C++

```
__property int TabWidth;
```

Description

Set TabWidth to control the number of dialog base units displayed for each tab character that appears in the strings of a list box.

If TabWidth is set to 0, which is the default setting, the tabs default to the width specified by Windows, which is two dialog base units.

30.1.14.6.18 TCustomListBox.TopIndex Property

Specifies the index number of the item that appears at the top of the list box.

Pascal

```
property TopIndex: Integer;
```

C++

```
__property int TopIndex;
```

Description

Use TopIndex property to find or set the first item displayed at the top of the list box. TopIndex can be used, for example, to change the topmost item to a different item in the list.

C++ Examples:

```
/*
The following example uses a list box containing a list of
strings, a button, and an edit box on a form. When the user
runs the application and clicks the button, the third item
in the list becomes the first item, and the index value of
that item appears in the edit box. The index value displayed
is 2, indicating the third item in the list (the first item
in the list has an index value of 0):
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->TopIndex = 2;
  Edit1->Text = IntToStr(ListBox1->TopIndex);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int Number = 1; Number <= 20; Number++)
    ListBox1->Items->Add("Item " + IntToStr(Number));
}
```

Delphi Examples:

```
{
The following example uses a list box containing a list of
strings, a button, and an edit box on a form. When the user
runs the application and clicks the button, the third item
in the list becomes the first item, and the index value of
```

```
  that item appears in the edit box. The index value displayed
  is 2, indicating the third item in the list (the first item
  in the list has an index value of 0):
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    ListBox1.TopIndex := 2;
    Edit1.Text := IntToStr(ListBox1.TopIndex);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    Number: Integer;
  begin
    for Number := 1 to 20 do
      ListBox1.Items.Add('Item ' + IntToStr(Number));
  end;
```

See Also

ItemIndex

Items (see page 2429)

Sorted (see page 2434)

30.1.14.7 TCustomListBox Events

The events of the TCustomListBox class are listed here.

Protected Events

	Name	Description
	OnData (see page 2438)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2439)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2439)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDrawItem (see page 2439)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnMeasureItem (see page 2442)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

30.1.14.7.1 TCustomListBox.OnData Event

Occurs when a virtual list box needs to supply the text of an item.

Pascal

```
property OnData: TLBGetDataEvent;
```

C++

```
__property TLBGetDataEvent OnData;
```

Description

You must supply an OnData event handler to dynamically supply the items in the list box when the Style (see page 2436) property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnDataFind (see page 2439)

OnDataObject (see page 2439)

Style (see page 2436)

Count (see page 2427)

30.1.14.7.2 TCustomListBox.OnDataFind Event

Occurs when a virtual list box needs to identify the index of an item given its text.

Pascal

```
property OnDataFind: TLBFindDataEvent;
```

C++

```
__property TLBFindDataEvent OnDataFind;
```

Description

You must supply an OnDataFind event handler to dynamically provide the mapping from list item to index when the Style (see page 2436) property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnData (see page 2438)

OnDataObject (see page 2439)

Style (see page 2436)

Count (see page 2427)

30.1.14.7.3 TCustomListBox.OnDataObject Event

Occurs when a virtual list box needs to supply the object associated with an item.

Pascal

```
property OnDataObject: TLBGetDataObjectEvent;
```

C++

```
__property TLBGetDataObjectEvent OnDataObject;
```

Description

Supply an OnDataObject event handler to dynamically provide objects associated with the items in a virtual list box.

See Also

OnData (see page 2438)

OnDataFind (see page 2439)

Style (see page 2436)

Count (see page 2427)

30.1.14.7.4 TCustomListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style (see page 2436) values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight (see page 2428) property for fixed owner-draw list boxes or by the response to the OnMeasureItem (see page 2442) event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = (dynamic_cast<TListBox *>(Control))->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
      pBitmap,
      Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
```

```
  example, a list box with the lbOwnerDrawFixed style draws a
  bitmap to the left of each string.
  Select the TListBox and set the ItemHeight property to 32.
  Select the ImageList and set the Layout Height and Width to
  32.  Double click on the ImageList to load the images.  Be
  sure to load in the correct order, logo2, Sunflower,
  butterfly.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyList := TStringList.Create;
    bitmap0 := TBitmap.Create;
    bitmap1 := TBitmap.Create;
    bitmap2 := TBitmap.Create;
  //  MyList.Add('Animal');
  //  MyList.Add('Flowers');
  //  MyList.Add('Butterfly');
  //  ListBox1.Items.AddStrings(MyList);

    ImageList1.GetBitmap(0, bitmap0);
    ListBox1.Items.AddObject('Flowers', bitmap0);
    ImageList1.GetBitmap(1, bitmap1);
    ListBox1.Items.AddObject('Animal', bitmap1);
    ImageList1.GetBitmap(2, bitmap2);
    ListBox1.Items.AddObject('Butterfly', bitmap2);
  end;

  procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
  TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    Bitmap := TBitmap.Create;
    with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DrawItem (see page 2408)

ItemHeight (see page 2428)

MeasureItem (see page 2415)

OnMeasureItem (see page 2442)

Style (see page 2436)

30.1.14.7.5 TCustomListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style (see page 2436) property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent (see page 3012) which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem (see page 2439) event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
      int &Height)
{
  Graphics::TBitmap *bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap && bitmap->Height > Height)
    Height = bitmap->Height;
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
```

```
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}

#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Making these static keeps them around until the project terminates.
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);
  bitmap0->LoadFromFile("../littleB_64.bmp");
//  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Butterfly", bitmap0);
  ImageList1->GetBitmap(3, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
64.
  ImageList1->GetBitmap(4, bitmap2);
  ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
at 64.

}
```

See Also

DrawItem (see page 2408)

ItemHeight (see page 2428)

MeasureItem (see page 2415)

OnDrawItem (see page 2439)

Style (see page 2436)

30.1.15 TCustomMemo Class

TCustomMemo is the base type for memo components, which are multiline edit boxes, including TMemo (see page 2737) and TDBMemo.

Class Hierarchy

Pascal

```
TCustomMemo = class(TCustomEdit);
```

C++

```
class TCustomMemo : public TCustomEdit;
```

File

StdCtrls

Description

TCustomMemo introduces new properties and methods to expand the capabilities of TCustomEdit (see page 2345) for handling multiline edit boxes.

Do not create instances of TCustomMemo. Use TCustomMemo as a base class when declaring control objects that implement a multiline edit control. Properties and methods of TCustomMemo provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TDBMemo

TMemo (see page 2737)

30.1.15.1 TCustomMemo Members

The following tables list the members exposed by TCustomMemo.

Protected Events

	Name	Description
	OnChange (see page 2381)	Occurs when the text for the edit control may have changed.

Protected Methods

	Name	Description
	Change (see page 2348)	Generates an OnChange (see page 2381) event.
V	CreateParams (see page 2447)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2447)	Creates a window based on the window-creation parameter record.
V	CreateWnd (see page 2352)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2354)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 2354)	Sets the MaxLength (see page 2368) property.
V	DoSetTextHint (see page 2355)	Sets the value of TextHint (see page 2380).
V	GetCaretPos (see page 2448)	Returns the value of the CaretPos (see page 2450) property.
V	GetSelLength (see page 2356)	Returns the SelLength (see page 2371) property.
V	GetSelStart (see page 2356)	Returns the SelStart (see page 2374) property.
V	GetSelText (see page 2356)	Returns the SelText (see page 2377) property.
V	KeyPress (see page 2448)	Ensures that return characters are not entered into the text when WantReturns (see page 2454) is false.
V	Loaded (see page 2448)	Initializes the memo control when it is first loaded into memory.
	SetAlignment (see page 2361)	Sets the Alignment (see page 2365) property.
V	SetAutoSize (see page 2361)	Sets the AutoSize (see page 2366) property.
V	SetCaretPos (see page 2449)	Sets the CaretPos (see page 2450) property.
	SetLines (see page 2449)	Sets the Lines (see page 2451) property.
	SetScrollBars (see page 2449)	Sets the ScrollBars (see page 2452) property.
V	SetSelLength (see page 2361)	Sets the SelLength (see page 2371) property.
V	SetSelStart (see page 2362)	Sets the SelStart (see page 2374) property.
	SetWordWrap (see page 2450)	Sets the WordWrap (see page 2455) property.

Protected Properties

	Name	Description
	AutoSelect (see page 2365)	Determines whether all the text in the edit control is automatically selected when the control gets focus.

	AutoSize (see page 2366)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BorderStyle (see page 2367)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2367)	Determines the case of the text within the edit control.
	HideSelection (see page 2368)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	MaxLength (see page 2368)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2369)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2369)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentColor (see page 2370)	Determines where a control looks for its color information.
	PasswordChar (see page 2370)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	ScrollBars (see page 2452)	Determines whether the memo control has scroll bars.
	WantReturns (see page 2454)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2454)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2455)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Public Methods

	Name	Description
V	Clear (see page 2348)	Deletes all text from the edit control.
	ClearSelection (see page 2349)	Removes the selected text from the edit control.
	ClearUndo (see page 2349)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2349)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 2446)	Creates and initializes an instance of TCustomMemo (see page 2443).
	CutToClipboard (see page 2352)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 2354)	Provides message handling for all messages that the control does not fully process itself.
V	Destroy (see page 2446)	Destroys an instance of TCustomMemo (see page 2443).
V	GetControlsAlignment (see page 2355)	Indicates how text is aligned within the control.
V	GetSelTextBuf (see page 2357)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2358)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2359)	Selects all text in the edit control.
	SetSelText (see page 2362)	Sets the SelText (see page 2377) property.
	SetSelTextBuf (see page 2363)	Replaces the selected text with a null-terminated string.
	Undo (see page 2364)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
	Alignment (see page 2365)	Determines how the text is aligned within the text edit control.
R	CanUndo (see page 2367)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 2450)	Indicates the position of the caret in the client area of the memo.
	Lines (see page 2451)	Contains the individual lines of text in the memo control.
	Modified (see page 2369)	Indicates whether the user edited the text of the edit control.
	ReadOnly (see page 2371)	Determines whether the user can change the text of the edit control.
	SelLength (see page 2371)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2374)	Specifies the position of the first selected character in the text.
	SelText (see page 2377)	Specifies the selected portion of the edit control's text.
	Text (see page 2379)	Contains a text string associated with the control.
	TextHint (see page 2380)	A hint or message to be displayed when the Text (see page 2379) property is empty.

Published Properties

	Name	Description
	TabStop (see page 2379)	Determines if the user can tab to a control.

30.1.15.2 TCustomMemo.Create Constructor

Creates and initializes an instance of TCustomMemo (see page 2443).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomMemo(TComponent * AOwner);
```

Description

Call Create to instantiate a TCustomMemo (see page 2443) object.

AOwner is the component, typically the form, that is responsible for freeing the memo object. This becomes the value of the Owner property.

Derived objects that override the Create method should call the inherited Create before making any additional initializations.

See Also

Owner

30.1.15.3 TCustomMemo.Destroy Destructor

Destroys an instance of TCustomMemo (see page 2443).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomMemo();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free checks that the TCustomMemo (see page 2443) reference is not nil, and only then calls Destroy.

Destroy frees the TStrings object that implements the Lines (see page 2451) property before calling the inherited destructor.

See Also

System::TObject::Free (see page 3372)

Lines (see page 2451)

TStrings

30.1.15.4 TCustomMemo Methods

The methods of the TCustomMemo class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2447)	Initializes a window-creation parameter record passed in the Params parameter.
	CreateWindowHandle (see page 2447)	Creates a window based on the window-creation parameter record.
	GetCaretPos (see page 2448)	Returns the value of the CaretPos (see page 2450) property.
	KeyPress (see page 2448)	Ensures that return characters are not entered into the text when WantReturns (see page 2454) is false.
	Loaded (see page 2448)	Initializes the memo control when it is first loaded into memory.
	SetCaretPos (see page 2449)	Sets the CaretPos (see page 2450) property.
	SetLines (see page 2449)	Sets the Lines (see page 2451) property.
	SetScrollBars (see page 2449)	Sets the ScrollBars (see page 2452) property.
	SetWordWrap (see page 2450)	Sets the WordWrap (see page 2455) property.

30.1.15.4.1 TCustomMemo.CreateParams Method

Initializes a window-creation parameter record passed in the Params parameter.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd (see page 2352) method calls CreateParams to initialize the parameters it passes to CreateWindowHandle (see page 2447). TCustomMemo (see page 2443) overrides the inherited method to specify a scrollable multiline edit control. The Params parameter is updated to reflect the current values of the Alignment (see page 2365), ScrollBars (see page 2452), and WordWrap (see page 2455) properties, in addition to the BorderStyle (see page 2367), HideSelection (see page 2368), PasswordChar (see page 2370), and ReadOnly (see page 2371) properties that the inherited method specifies.

See Also

Alignment (see page 2365)

BorderStyle (see page 2367)

CreateWindowHandle (see page 2447)

CreateWnd (see page 2352)

HideSelection (see page 2368)

PasswordChar (see page 2370)

ReadOnly (see page 2371)

ScrollBars (see page 2452)

WordWrap (see page 2455)

30.1.15.4.2 TCustomMemo.CreateWindowHandle Method

Creates a window based on the window-creation parameter record.

Pascal

```
procedure CreateWindowHandle(const Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateWindowHandle(const TCreateParams Params);
```

Description

The CreateWnd (see page 2352) method calls CreateWindowHandle to create the edit control window once it has been specified in the window-creation parameters. CreateWindowHandle creates the window by calling the CreateWindowEx API function, passing parameters from the record passed in the Params parameter. CreateWindowHandle overrides the inherited method to interpret the Caption in Params as the initial text rather than as the window caption.

See Also

CreateParams (see page 2447)

CreateWnd (see page 2352)

30.1.15.4.3 TCustomMemo.GetCaretPos Method

Returns the value of the CaretPos (see page 2450) property.

Pascal

```
function GetCaretPos: TPoint; virtual;
```

C++

```
virtual __fastcall TPoint GetCaretPos();
```

Description

CaretPos (see page 2450) represents the coordinates of the cursor. GetCaretPos is the protected implementation of the CaretPos (see page 2450) property. Override this method to change the way the caret position is computed.

See Also

CaretPos (see page 2450)

30.1.15.4.4 TCustomMemo.KeyPress Method

Ensures that return characters are not entered into the text when WantReturns (see page 2454) is false.

Pascal

```
procedure KeyPress(var Key: Char); override;
```

C++

```
virtual __fastcall KeyPress(Char Key);
```

Description

KeyPress is called automatically when the user presses a key. After calling the inherited KeyPress, which calls the OnKeyPress event handler, KeyPress discards return characters when WantReturns (see page 2454) is false.

See Also

WantReturns (see page 2454)

30.1.15.4.5 TCustomMemo.Loaded Method

Initializes the memo control when it is first loaded into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Loaded overrides the inherited method so that the Modified (see page 2369) property is set to false when the form containing the memo is first loaded into memory. Derived classes can override the Loaded method to perform additional initializations after the memo's form is loaded.

See Also

Modified (see page 2369)

30.1.15.4.6 TCustomMemo.SetCaretPos Method

Sets the CaretPos (see page 2450) property.

Pascal

```
procedure SetCaretPos(const Value: TPoint); virtual;
```

C++

```
virtual __fastcall SetCaretPos(const TPoint Value);
```

Description

SetCaretPos is the property write access method for the CaretPos (see page 2450) property. The CaretPos (see page 2450) property returns the coordinates of the cursor. SetCaretPos sets the cursor position to Value.

See Also

CaretPos (see page 2450)

30.1.15.4.7 TCustomMemo.SetLines Method

Sets the Lines (see page 2451) property.

Pascal

```
procedure SetLines(Value: TStrings);
```

C++

```
__fastcall SetLines(TStrings Value);
```

Description

SetLines is the protected write implementation for the Lines (see page 2451) property.

SetLines is exposed as a protected method so derived classes can customize their implementation of the Lines (see page 2451) property.

See Also

Lines (see page 2451)

30.1.15.4.8 TCustomMemo.SetScrollBars Method

Sets the ScrollBars (see page 2452) property.

Pascal

```
procedure SetScrollBars(Value: TScrollStyle);
```

C++

```
__fastcall SetScrollBars(TScrollStyle Value);
```

Description

SetScrollBars is the property access method for the ScrollBars (see page 2452) property.

SetScrollBars is exposed as a protected method so derived classes can customize their implementation of the ScrollBars (see page 2452) property.

See Also

ScrollBars (see page 2452)

30.1.15.4.9 TCustomMemo.SetWordWrap Method

Sets the WordWrap (see page 2455) property.

Pascal

```
procedure SetWordWrap(Value: Boolean);
```

C++

```
__fastcall SetWordWrap(Boolean Value);
```

Description

SetWordWrap is the property access method for the WordWrap (see page 2455) property.

See Also

WordWrap (see page 2455)

30.1.15.5 TCustomMemo Properties

The properties of the TCustomMemo class are listed here.

Protected Properties

	Name	Description
	ScrollBars (see page 2452)	Determines whether the memo control has scroll bars.
	WantReturns (see page 2454)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2454)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2455)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Public Properties

	Name	Description
	CaretPos (see page 2450)	Indicates the position of the caret in the client area of the memo.
	Lines (see page 2451)	Contains the individual lines of text in the memo control.

30.1.15.5.1 TCustomMemo.CaretPos Property

Indicates the position of the caret in the client area of the memo.

Pascal

```
property CaretPos: TPoint;
```

C++

```
__property TPoint CaretPos;
```

Description

Use CaretPos to determine the coordinates of the cursor. CaretPos defines a location onscreen, in characters, with the origin in the top left corner of the memo. X specifies the horizontal coordinate of the point, Y specifies the vertical coordinate.

To determine the position of the caret in terms of characters of text only (rather than X/Y location), use the SelStart (see page 2374) property.

See Also

ClientRect

ClientOrigin

SelStart (see page 2374)

30.1.15.5.2 TCustomMemo.Lines Property

Contains the individual lines of text in the memo control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to manipulate text in an memo control on a line-by-line basis. Lines is a TStrings object, so the TStrings methods may be used for Lines to perform manipulations such as counting the lines of text, adding new lines, deleting lines, or replacing lines with new text.

To work with all the text at once, use the Text (see page 2379) property. To manipulate individual lines of text, the Lines property works better.

Note: Although Lines is implemented as a TStrings descendant, it does not implement the support for associating objects with the strings in the list.

C++ Examples:

```
/*
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Lines->LoadFromFile("../readme.txt");
  Caption = Memo1->Lines->Strings[4];
}
```

Delphi Examples:

```
{
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
```

```
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.LoadFromFile('readme.txt');
  Caption:= Memo1.Lines[4];
end;
```

See Also

TStrings

SelText (see page 2377)

SetLines (see page 2449)

Text

WordWrap (see page 2455)

30.1.15.5.3 TCustomMemo.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
```

```
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

SetScrollBars (see page 2449)

WordWrap (see page 2455)

Displaying and Editing Text in a Memo Control

30.1.15.5.4 TCustomMemo.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text

WantTabs (see page 2454)

WordWrap (see page 2455)

30.1.15.5.5 TCustomMemo.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text

WantReturns (see page 2454)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

30.1.15.5.6 TCustomMemo.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
```

```
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
```

```
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
```

See Also

Lines (see page 2451)

ScrollBars (see page 2452)

Text

WantReturns (see page 2454)

Displaying and Editing Text in a Memo Control

30.1.16 TCustomStaticText Class

TCustomStaticText is the base class for TStaticText (see page 2945).

Class Hierarchy

Pascal

```
TCustomStaticText = class(TWinControl);
```

C++

```
class TCustomStaticText : public TWinControl;
```

File

StdCtrls

Description

TCustomStaticText is the base class for TStaticText (see page 2945), a windowed control that displays text on a form. This component functions like TLabel (see page 2607), except that it has a window handle.

TCustomStaticText cannot receive focus or contain other controls.

See Also

TLabel (see page 2607)

TStaticText (see page 2945)

30.1.16.1 TCustomStaticText Members

The following tables list the members exposed by TCustomStaticText.

Protected Methods

	Name	Description
	CreateParams (see page 2459)	Initializes a window-creation parameter data structure.
	Loaded (see page 2459)	Initializes the control after it is loaded from a stream.
	Notification (see page 2460)	Responds to notifications that components are being created or destroyed.
	SetAutoSize (see page 2460)	Sets the value of the AutoSize property.

Protected Properties

	Name	Description
	Alignment (see page 2461)	Determines how text is aligned within the component.
	AutoSize (see page 2461)	Determines whether the component is automatically resized to accommodate its text.
	BorderStyle (see page 2462)	Determines the appearance of the control's border.
	FocusControl (see page 2462)	Links the component with another windowed control.
	ShowAccelChar (see page 2462)	Enables accelerator characters.
	Transparent (see page 2463)	Specifies whether the background of the image obscures objects below the image object.

Public Methods

	Name	Description
	Create (see page 2458)	Creates and initializes a TCustomStaticText (see page 2457) instance.

30.1.16.2 TCustomStaticText.Create Constructor

Creates and initializes a TCustomStaticText (see page 2457) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomStaticText(TComponent * AOwner);
```

Description

Create calls the inherited constructor, then initializes the TCustomStaticText (see page 2457) instance

See Also

AutoSize (see page 2461)

ControlStyle

ShowAccelChar (see page 2462)

30.1.16.3 TCustomStaticText Methods

The methods of the TCustomStaticText class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2459)	Initializes a window-creation parameter data structure.
	Loaded (see page 2459)	Initializes the control after it is loaded from a stream.
	Notification (see page 2460)	Responds to notifications that components are being created or destroyed.
	SetAutoSize (see page 2460)	Sets the value of the AutoSize property.

30.1.16.3.1 TCustomStaticText.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

30.1.16.3.2 TCustomStaticText.Loaded Method

Initializes the control after it is loaded from a stream.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call Loaded. The VCL streaming system calls Loaded automatically after the control's form is loaded into memory so that the control can complete any initializations that depend on other objects in the form.

Loaded overrides the inherited method in order to initialize the control from its associated Action. To change the properties the control copies from its action, override the ActionChange method.

See Also

Loaded

Action

ActionChange

30.1.16.3.3 TCustomStaticText.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

30.1.16.3.4 TCustomStaticText.SetAutoSize Method

Sets the value of the AutoSize property.

Pascal

```
procedure SetAutoSize(Value: Boolean); override;
```

C++

```
virtual __fastcall SetAutoSize(Boolean Value);
```

Description

SetAutoSize is a protected method in the TControl class and is used to set the value of the AutoSize property.

See Also

TControl

AutoSize

30.1.16.4 TCustomStaticText Properties

The properties of the TCustomStaticText class are listed here.

Protected Properties

	Name	Description
	Alignment (see page 2461)	Determines how text is aligned within the component.
	AutoSize (see page 2461)	Determines whether the component is automatically resized to accommodate its text.
	BorderStyle (see page 2462)	Determines the appearance of the control's border.
	FocusControl (see page 2462)	Links the component with another windowed control.
	ShowAccelChar (see page 2462)	Enables accelerator characters.
	Transparent (see page 2463)	Specifies whether the background of the image obscures objects below the image object.

30.1.16.4.1 TCustomStaticText.Alignment Property

Determines how text is aligned within the component.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

The possible values of Alignment are

Value	Meaning
taLeftJustify	Text is aligned at the left of the control.
taCenter	Text is centered in the control.
taRightJustify	Text is aligned at the right of the control.

See Also

Font

Caption

30.1.16.4.2 TCustomStaticText.AutoSize Property

Determines whether the component is automatically resized to accommodate its text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

If AutoSize is true, the component is automatically resized— at runtime and design time— whenever the font size or Caption text changes.

See Also

Caption

Font

30.1.16.4.3 TCustomStaticText.BorderStyle Property

Determines the appearance of the control's border.

Pascal

```
property BorderStyle: TStaticBorderStyle;
```

C++

```
__property TStaticBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify the appearance of the border, if any, that appears around the text.

30.1.16.4.4 TCustomStaticText.FocusControl Property

Links the component with another windowed control.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Use the FocusControl property to link the TCustomStaticText (see page 2457) instance with another windowed control. If the text of Caption includes an accelerator key, the component specified by FocusControl receives focus when the accelerator is pressed.

See Also

Caption

FocusControl (see page 2395)

TWinControl

30.1.16.4.5 TCustomStaticText.ShowAccelChar Property

Enables accelerator characters.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

If ShowAccelChar is true, an ampersand in the component's Caption text appears as an underline and makes the character after it into an accelerator key. If ShowAccelChar is false, ampersands appear as ampersands.

See Also

Caption

30.1.16.4.6 TCustomStaticText.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TCustomStaticText (see page 2457) object to show through the background of the image. Set Transparent to false to make the background of the image opaque.

30.1.17 TEdit Class

TEdit is a wrapper for a Windows single-line edit control.

Class Hierarchy

Pascal

```
TEdit = class(TCustomEdit);
```

C++

```
class TEdit : public TCustomEdit;
```

File

StdCtrls

Description

Use a TEdit object to put a standard Windows edit control on a form. Edit controls are used to retrieve text that users type. Edit controls can also display text to the user.

When only displaying text to the user, choose an edit control to allow users to select text and copy it to the Clipboard. Choose a label object if the selection capabilities of an edit control are not needed.

TEdit implements the generic behavior introduced in TCustomEdit (see page 2345). TEdit publishes many of the properties inherited from TCustomEdit (see page 2345), but does not introduce any new behavior. For specialized edit controls, use other descendant classes of TCustomEdit (see page 2345) or derive from it.

See Also

TDBEdit

TDBMemo

TInplaceEdit

TLabel (see page 2607)

TMaskEdit

TMemo (see page 2737)

TRichEdit

Displaying Field Component Values in Standard Controls

Edit Controls

30.1.17.1 TEdit Members

The following tables list the members exposed by TEdit.

Protected Methods

	Name	Description
	Change (see page 2348)	Generates an OnChange (see page 2381) event.
V	CreateParams (see page 2351)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2351)	Creates a Windows control to represent the control.
V	CreateWnd (see page 2352)	Creates a Windows control corresponding to the edit control.
V	DestroyWnd (see page 2354)	Destroys the Windows control that corresponds to the edit control.
V	DoSetMaxLength (see page 2354)	Sets the MaxLength (see page 2368) property.
V	DoSetTextHint (see page 2355)	Sets the value of TextHint (see page 2380).
V	GetSelLength (see page 2356)	Returns the SelLength (see page 2371) property.
V	GetSelStart (see page 2356)	Returns the SelStart (see page 2374) property.
V	GetSelText (see page 2356)	Returns the SelText (see page 2377) property.
	SetAlignment (see page 2361)	Sets the Alignment (see page 2365) property.
V	SetAutoSize (see page 2361)	Sets the AutoSize (see page 2366) property.
V	SetSelLength (see page 2361)	Sets the SelLength (see page 2371) property.
V	SetSelStart (see page 2362)	Sets the SelStart (see page 2374) property.

Public Methods

	Name	Description
V	Clear (see page 2348)	Deletes all text from the edit control.
	ClearSelection (see page 2349)	Removes the selected text from the edit control.
	ClearUndo (see page 2349)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2349)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
V	Create (see page 2347)	Creates an instance of TCustomEdit (see page 2345).
	CutToClipboard (see page 2352)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
V	DefaultHandler (see page 2354)	Provides message handling for all messages that the control does not fully process itself.
V	GetControlsAlignment (see page 2355)	Indicates how text is aligned within the control.
V	GetSelTextBuf (see page 2357)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2358)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2359)	Selects all text in the edit control.
	SetSelText (see page 2362)	Sets the SelText (see page 2377) property.
	SetSelTextBuf (see page 2363)	Replaces the selected text with a null-terminated string.
	Undo (see page 2364)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
R	CanUndo (see page 2367)	Indicates whether the edit control contains changes that can be backed out.
	Modified (see page 2369)	Indicates whether the user edited the text of the edit control.

	SelLength (see page 2371)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2374)	Specifies the position of the first selected character in the text.
	SelText (see page 2377)	Specifies the selected portion of the edit control's text.

Published Events

	Name	Description
	OnChange (see page 2501)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2502)	Occurs when the user clicks the control.
	OnContextPopup (see page 2504)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2504)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2505)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2508)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2511)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2511)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2512)	Occurs when a control receives the input focus.
	OnExit (see page 2515)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2519)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2522)	Occurs when a key is pressed.
	OnKeyUp (see page 2523)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2525)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2526)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2528)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2529)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2529)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2531)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2533)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2534)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2467)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2471)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2471)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 2472)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2472)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 2473)	Specifies which edges of the control are beveled.
	BevelInner (see page 2474)	Specifies the cut of the inner bevel.
	BevelKind (see page 2474)	Specifies the control's bevel style.
	BevelOuter (see page 2474)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2475)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2475)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2476)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2476)	Determines the case of the text within the edit control.
	Color (see page 2476)	Specifies the background color of the control.
	Constraints (see page 2477)	Specifies the size constraints for the control.

	Ctl3D (see page 2478)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2478)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2480)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2481)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2481)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2482)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2483)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2484)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2484)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2485)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 2486)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2486)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2486)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2487)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2487)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2488)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2488)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2488)	Determines where a control looks for its font information.
	ParentShowHint (see page 2489)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 2489)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 2490)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2495)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 2495)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2497)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2497)	Determines if the user can tab to a control.
	Text (see page 2498)	Contains a text string associated with the control.
	TextHint (see page 2499)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 2499)	Determines whether the component appears onscreen.

30.1.17.2 TEdit Properties

The properties of the TEdit class are listed here.

Published Properties

	Name	Description
	Align (see page 2467)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2471)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2471)	Specifies how the control is anchored to its parent.
	AutoSelect (see page 2472)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2472)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	BevelEdges (see page 2473)	Specifies which edges of the control are beveled.
	BevelInner (see page 2474)	Specifies the cut of the inner bevel.
	BevelKind (see page 2474)	Specifies the control's bevel style.
	BevelOuter (see page 2474)	Specifies the cut of the outer bevel.

	BevelWidth (see page 2475)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2475)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2476)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2476)	Determines the case of the text within the edit control.
	Color (see page 2476)	Specifies the background color of the control.
	Constraints (see page 2477)	Specifies the size constraints for the control.
	Ctl3D (see page 2478)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2478)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2480)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2481)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2481)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2482)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2483)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2484)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2484)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2485)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	MaxLength (see page 2486)	Specifies the maximum number of characters the user can enter into the edit control.
	NumbersOnly (see page 2486)	Allows only numbers to be typed into the text edit.
	OEMConvert (see page 2486)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2487)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2487)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2488)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2488)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2488)	Determines where a control looks for its font information.
	ParentShowHint (see page 2489)	Determines where a control looks to find out if its Help Hint should be shown.
	PasswordChar (see page 2489)	Indicates the character, if any, to display in place of the actual characters typed in the control.
	PopupMenu (see page 2490)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2495)	Determines whether the user can change the text of the edit control.
	ShowHint (see page 2495)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2497)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2497)	Determines if the user can tab to a control.
	Text (see page 2498)	Contains a text string associated with the control.
	TextHint (see page 2499)	A hint or message to be displayed when the Text property is empty.
	Visible (see page 2499)	Determines whether the component appears onscreen.

30.1.17.2.1 TEdit.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
```

```
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
```

```
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
```

```
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

30.1.17.2.2 TEdit.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 2361)

Text

Displaying and Editing Fields in an Edit Box

30.1.17.2.3 TEdit.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.17.2.4 TEdit.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

Pascal

```
property AutoSelect: Boolean;
```

C++

```
__property Boolean AutoSelect;
```

Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

See Also

SelectAll (see page 2359)

SelLength (see page 2371)

SelStart (see page 2374)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

30.1.17.2.5 TEdit.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

Note: AutoSize only adjusts the height of the edit control. Even with AutoSize set to true, text in the edit control may appear truncated if it extends beyond the width of the control.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit1->AutoSize = true;
  Edit1->Font->Size = 20;
  Edit1->Text = "The edit box is taller now";
}
```

Delphi Examples:

```
{
{
This example uses an edit box and a button on a form. When
the user clicks the button, the font in the edit box
enlarges, and the edit box enlarges also to accommodate the
larger font size.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AutoSize := True;
  Edit1.Font.Size := 20;
  Edit1.Text := 'The edit box is taller now';
end;
```

See Also

BorderStyle (see page 2367)

ClientHeight

ControlStyle

Height

30.1.17.2.6 TEdit.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

30.1.17.2.7 TEdit.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

30.1.17.2.8 TEdit.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

30.1.17.2.9 TEdit.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

30.1.17.2.10 TEdit.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

30.1.17.2.11 TEdit.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.17.2.12 TEdit.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

30.1.17.2.13 TEdit.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

30.1.17.2.14 TEdit.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.17.2.15 TEdit.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the

ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.17.2.16 TEdit.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.17.2.17 TEdit.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the

windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.17.2.18 TEdit.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.17.2.19 TEdit.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.17.2.20 TEdit.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
```

```
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.17.2.21 TEdit.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.17.2.22 TEdit.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.17.2.23 TEdit.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 2371)

SelStart (see page 2374)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

30.1.17.2.24 TEdit.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

30.1.17.2.25 TEdit.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

30.1.17.2.26 TEdit.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

30.1.17.2.27 TEdit.NumbersOnly Property

Allows only numbers to be typed into the text edit.

Pascal

```
property NumbersOnly: Boolean;
```

C++

```
__property Boolean NumbersOnly;
```

Description

Use NumbersOnly to prohibit entry of non-numeric characters in the textfield. Note, however, that a user can paste non-numeric characters in the textfield even when this property is set.

30.1.17.2.28 TEdit.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

30.1.17.2.29 TEdit.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.17.2.30 TEdit.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.17.2.31 TEdit.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.17.2.32 TEdit.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.17.2.33 TEdit.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.17.2.34 TEdit.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.17.2.35 TEdit.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Pascal

```
property PasswordChar: Char;
```

C++

```
__property Char PasswordChar;
```

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

See Also

Text

30.1.17.2.36 TEdit.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
```

```
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
```

```
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
```

```
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
```

```
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.17.2.37 TEdit.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 2349)

GetSelTextBuf (see page 2357)

30.1.17.2.38 TEdit.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.17.2.39 TEdit.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.17.2.40 TEdit.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.17.2.41 TEdit.Text Property

Contains a text string associated with the control.

Pascal

```
property Text;
```

C++

```
__property Text;
```

Description

Use the Text property to read the Text of the control or specify a new string for the Text value. By default, Text is the control name. For edit controls and memos, the Text appears within the control. For combo boxes, the Text is the content of the edit control portion of the combo box.

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

C++ Examples:

```
/*
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Size = Edit1->GetTextLen(); //Get length of string in Edit1
  Size++;                       //Add room for null character
//  Char *Buffer = new Char[Size];  //Creates Buffer dynamic variable
  std::auto_ptr<Char> Buffer(new Char[Size]);
  Edit1->GetTextBuf(Buffer.get(),Size); //Puts Edit1->Text into Buffer
  Edit2->Text = Buffer.get();
}
/*
Note: the same thing could be accomplished more simply as follows:
*/
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Edit2->Text = Edit1->Text;
}
```

Delphi Examples:

```
{
This example requires two edit boxes and a button.  This example
copies the text in an edit box into a null-terminated string, and
puts this string in another edit box when the user clicks the
button on the form.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
  Size: Byte;
begin
  Size := Edit1.GetTextLen;       {Get length of string in Edit1}
  Inc(Size);                      {Add room for null character}
  GetMem(Buffer, Size);           {Creates Buffer dynamic variable}
  Edit1.GetTextBuf(Buffer,Size);  {Puts Edit1.Text into Buffer}
  Edit2.Text := StrPas(Buffer);   {Converts Buffer to a Pascal-style string}
  FreeMem(Buffer, Size);{Frees memory allocated to Buffer}
end;

{
Note: the same thing could be accomplished more simply as follows:
}
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
end;
```

See Also

Caption

Lines (see page 2451)

30.1.17.2.42 TEdit.TextHint Property

A hint or message to be displayed when the Text property is empty.

Pascal

```
property TextHint: string;
```

C++

```
__property UnicodeString TextHint;
```

Description

Use TextHint to cue the user on what kind of entry is expected in the text field. Runtime themes must be enabled.

30.1.17.2.43 TEdit.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
```

```
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.17.3 TEdit Events

The events of the TEdit class are listed here.

Published Events

	Name	Description
	OnChange (see page 2501)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2502)	Occurs when the user clicks the control.
	OnContextPopup (see page 2504)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2504)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2505)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2508)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2511)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2511)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2512)	Occurs when a control receives the input focus.
	OnExit (see page 2515)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2519)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2522)	Occurs when a key is pressed.
	OnKeyUp (see page 2523)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2525)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2526)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2528)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2529)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2529)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2531)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2533)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2534)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.17.3.1 TEdit.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 2348)

Text

30.1.17.3.2 TEdit.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}
```

```
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.17.3.3 TEdit.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.17.3.4 TEdit.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.17.3.5 TEdit.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.17.3.6 TEdit.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.17.3.7 TEdit.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.17.3.8 TEdit.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
```

```
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.17.3.9 TEdit.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.17.3.10 TEdit.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
```

```
    MaskEdit2->Text = "";
    MaskEdit3->Text = "";

    // use the Courier font in the maskedit boxes
    MaskEdit1->Font->Name = "Courier";
    MaskEdit2->Font->Name = "Courier";
    MaskEdit3->Font->Name = "Courier";

    // initialize the mask of each maskedit box
    // according to IP address coding conventions
    MaskEdit1->EditMask = "!099.099.099.099;1; ";
    MaskEdit2->EditMask = "!099.099.099.099;1; ";
    MaskEdit3->EditMask = "!099.099.099.099;1; ";

    MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
    // extract the net and host address from the IP
    String IP = MaskEdit1->Text;
    int net1 = IP.SubString(0, 3).TrimRight().ToInt();
    int net2 = IP.SubString(5, 3).TrimRight().ToInt();
    int host1 = IP.SubString(9, 3).TrimRight().ToInt();
    int host2 = IP.SubString(13, 3).TrimRight().ToInt();

    // a range test that you cannot validate through edit masks
    if (net1 < 0 || net1 > 255 ||
        net2 < 0 || net2 > 255 ||
        host1 < 0 || host1 > 255 ||
        host2 < 0 || host2 > 255)
      throw(Exception("Not a valid IP address."));

    // extract the net and host mask from the subnet mask
    String mask = MaskEdit2->Text;
    int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
    int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
    int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
    int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

    // a range test that you cannot validate through edit masks
    if (netmask1 < 0 || netmask1 > 255 ||
        netmask2 < 0 || netmask2 > 255 ||
        hostmask1 < 0 || hostmask1 > 255 ||
        hostmask2 < 0 || hostmask2 > 255)
      throw(Exception("Not a valid subnet mask."));

    // compute the subnet address using bit-wise AND
    int sub_net1 = net1 & netmask1;
    int sub_net2 = net2 & netmask2;
    int sub_host1 = host1 & hostmask1;
    int sub_host2 = host2 & hostmask2;

    // display the subnet address
    MaskEdit3->Text =
      String(sub_net1) + "." + String(sub_net2) + "." +
      String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;
```

```
begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

30.1.17.3.11 TEdit.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
```

```
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
```

```
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.17.3.12 TEdit.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII

character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.17.3.13 TEdit.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can

respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
```

```
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

30.1.17.3.14 TEdit.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.17.3.15 TEdit.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
```

```
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.17.3.16 TEdit.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.17.3.17 TEdit.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.17.3.18 TEdit.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.17.3.19 TEdit.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.17.3.20 TEdit.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.17.3.21 TEdit.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

30.1.18 TGroupBox Class

TGroupBox represents a Windows group box.

Class Hierarchy

Pascal

```
TGroupBox = class(TCustomGroupBox);
```

C++

```
class TGroupBox : public TCustomGroupBox;
```

File

StdCtrls

Description

The TGroupBox component represents a standard Windows group box, used to group related controls on a form. When another control component is placed within a group box, the group box becomes the parent of that component.

See Also

TRadioGroup

TDBRadioGroup

TControl

Group Boxes and Radio Groups

Grouping Controls

30.1.18.1 TGroupBox Members

The following tables list the members exposed by TGroupBox.

Protected Methods

	Name	Description
	AdjustClientRect (see page 2382)	Adjusts the ClientRect property for idiosyncrasies of the window.
	CreateParams (see page 2383)	Initializes a window-creation parameter record passed in the Params parameter.
	Paint (see page 2383)	Renders the onscreen image of the group box.

Public Methods

	Name	Description
	Create (see page 2382)	Creates and initializes a TCustomGroupBox (see page 2381) instance.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2566)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2566)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 2567)	Occurs when the user clicks the control.
	OnContextPopup (see page 2569)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

	OnDblClick (see page 2570)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2570)	Occurs when another control is docked to the control.
	OnDockOver (see page 2572)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2574)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2577)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2580)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2580)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2581)	Occurs when a control receives the input focus.
	OnExit (see page 2584)	Occurs when the input focus shifts away from one control to another.
	OnGetSiteInfo (see page 2588)	Returns the control's docking information.
	OnMouseActivate (see page 2590)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2591)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2593)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2594)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2594)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2596)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2598)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2599)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2600)	Occurs when the application tries to undock a control that is docked to the windowed control.

Published Properties

	Name	Description
	Align (see page 2537)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2541)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2541)	Specifies the bi-directional mode for the control.
	Caption (see page 2542)	Specifies a text string that identifies the control to the user.
	Color (see page 2544)	Specifies the background color of the control.
	Constraints (see page 2545)	Specifies the size constraints for the control.
	Ctl3D (see page 2546)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2546)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2547)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2549)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2550)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2550)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2551)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2552)	Controls the attributes of text written on or in the control.
	Padding (see page 2553)	Specifies the padding of a control.
	ParentBackground (see page 2553)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2554)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2554)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2555)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2555)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2555)	Determines where a control looks for its font information.
	ParentShowHint (see page 2556)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2556)	Identifies the pop-up menu associated with the control.

	ShowHint (see page 2561)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2563)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2563)	Determines if the user can tab to a control.
	Visible (see page 2564)	Determines whether the component appears onscreen.

30.1.18.2 TGroupBox Properties

The properties of the TGroupBox class are listed here.

Published Properties

	Name	Description
	Align (see page 2537)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2541)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2541)	Specifies the bi-directional mode for the control.
	Caption (see page 2542)	Specifies a text string that identifies the control to the user.
	Color (see page 2544)	Specifies the background color of the control.
	Constraints (see page 2545)	Specifies the size constraints for the control.
	Ctl3D (see page 2546)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DockSite (see page 2546)	Specifies whether the control can be the target of drag-and-dock operations.
	DoubleBuffered (see page 2547)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2549)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2550)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2550)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2551)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2552)	Controls the attributes of text written on or in the control.
	Padding (see page 2553)	Specifies the padding of a control.
	ParentBackground (see page 2553)	Determines whether control uses parent's theme background.
	ParentBiDiMode (see page 2554)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2554)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2555)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2555)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2555)	Determines where a control looks for its font information.
	ParentShowHint (see page 2556)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2556)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2561)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2563)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2563)	Determines if the user can tab to a control.
	Visible (see page 2564)	Determines whether the component appears onscreen.

30.1.18.2.1 TGroupBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
```

```
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
```

```
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
```

```
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

30.1.18.2.2 TGroupBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.18.2.3 TGroupBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.18.2.4 TGroupBox.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
```

```
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

30.1.18.2.5 TGroupBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.18.2.6 TGroupBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.18.2.7 TGroupBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.18.2.8 TGroupBox.DockSite Property

Specifies whether the control can be the target of drag-and-dock operations.

Pascal

```
property DockSite;
```

C++

```
__property DockSite;
```

Description

Set DockSite to true to allow other controls to be docked to this windowed control.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
```

30.1.18.2.9 TGroupBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.18.2.10 TGroupBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.18.2.11 TGroupBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.18.2.12 TGroupBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
```

```
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.18.2.13 TGroupBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.18.2.14 TGroupBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.18.2.15 TGroupBox.Padding Property

Specifies the padding of a control.

Pascal

```
property Padding;
```

C++

```
__property Padding;
```

Description

Use Padding to specify the padding of a control. This value is an instance of the class TPadding.

See Also

TPadding

30.1.18.2.16 TGroupBox.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has

no effect unless XP themes are enabled.

30.1.18.2.17 TGroupBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.18.2.18 TGroupBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.18.2.19 TGroupBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.18.2.20 TGroupBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.18.2.21 TGroupBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.18.2.22 TGroupBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.18.2.23 TGroupBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
```

```
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
```

```
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
```

```
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.18.2.24 TGroupBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.18.2.25 TGroupBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.18.2.26 TGroupBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.18.2.27 TGroupBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.18.3 TGroupBox Events

The events of the TGroupBox class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 2566)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 2566)	Occurs when an object with custom alignment is aligned.
	OnClick (see page 2567)	Occurs when the user clicks the control.
	OnContextPopup (see page 2569)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2570)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDockDrop (see page 2570)	Occurs when another control is docked to the control.
	OnDockOver (see page 2572)	Occurs when another control is dragged over the control.
	OnDragDrop (see page 2574)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2577)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2580)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2580)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2581)	Occurs when a control receives the input focus.
	OnExit (see page 2584)	Occurs when the input focus shifts away from one control to another.

	OnGetSiteInfo (see page 2588)	Returns the control's docking information.
	OnMouseActivate (see page 2590)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2591)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2593)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2594)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2594)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2596)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2598)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2599)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnUnDock (see page 2600)	Occurs when the application tries to undock a control that is docked to the windowed control.

30.1.18.3.1 TGroupBox.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

30.1.18.3.2 TGroupBox.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment

parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

30.1.18.3.3 TGroupBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
```

```
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.18.3.4 TGroupBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.18.3.5 TGroupBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.18.3.6 TGroupBox.OnDockDrop Event

Occurs when another control is docked to the control.

Pascal

```
property OnDockDrop;
```

C++

```
__property OnDockDrop;
```

Description

Write code in the OnDockDrop event to perform actions when another control is docked on the control.

OnDockDrop can only occur if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockOver

30.1.18.3.7 TGroupBox.OnDockOver Event

Occurs when another control is dragged over the control.

Pascal

```
property OnDockOver;
```

C++

```
__property OnDockOver;
```

Description

Write code in the OnDockOver event to indicate whether a dockable control that is dragged over the windowed control can be accepted as a dock client. OnDockOver occurs after the TDragDockObject has been initialized by an OnGetSiteInfo event.

OnDockOver occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
```

```
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnGetSiteInfo

TDragDockObject

30.1.18.3.8 TGroupBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
```

```
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
```

```
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
```

```
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.18.3.9 TGroupBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
```

```
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
```

```
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.
```

```
procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.18.3.10 TGroupBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.18.3.11 TGroupBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.18.3.12 TGroupBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
```

```
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.18.3.13 TGroupBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
```

```
  where the third mask edit box is set to be read only.
  When the application runs, it initializes the mask edit boxes
  to contain the empty string, to use the Courier font and to
  use a particular mask, corresponding to IP address coding convetions.
  The mask edit boxes display the IP address, the subnet mask
  and the subnet address correspondingly. When the user exits the
  subnet mask edit box, the subnet address is automatically calculated
  using bit-wise AND operations and displayed in the third mask edit box.
  An additional test is made to assure that the values in the IP address
  and in the subnet mask are in the interval from 0 to 255 ,
  as this validation test cannot be accomplished
  using the edit mask property alone.
  */
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    // initialize the maskedit boxes to contain the empty string
    MaskEdit1->Text = "";
    MaskEdit2->Text = "";
    MaskEdit3->Text = "";

    // use the Courier font in the maskedit boxes
    MaskEdit1->Font->Name = "Courier";
    MaskEdit2->Font->Name = "Courier";
    MaskEdit3->Font->Name = "Courier";

    // initialize the mask of each maskedit box
    // according to IP address coding conventions
    MaskEdit1->EditMask = "!099.099.099.099;1; ";
    MaskEdit2->EditMask = "!099.099.099.099;1; ";
    MaskEdit3->EditMask = "!099.099.099.099;1; ";

    MaskEdit2->OnExit = MaskEdit2Exit;
  }

  void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
  {
    // extract the net and host address from the IP
    String IP = MaskEdit1->Text;
    int net1 = IP.SubString(0, 3).TrimRight().ToInt();
    int net2 = IP.SubString(5, 3).TrimRight().ToInt();
    int host1 = IP.SubString(9, 3).TrimRight().ToInt();
    int host2 = IP.SubString(13, 3).TrimRight().ToInt();

    // a range test that you cannot validate through edit masks
    if (net1 < 0 || net1 > 255 ||
        net2 < 0 || net2 > 255 ||
        host1 < 0 || host1 > 255 ||
        host2 < 0 || host2 > 255)
      throw(Exception("Not a valid IP address."));

    // extract the net and host mask from the subnet mask
    String mask = MaskEdit2->Text;
    int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
    int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
    int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
    int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

    // a range test that you cannot validate through edit masks
    if (netmask1 < 0 || netmask1 > 255 ||
        netmask2 < 0 || netmask2 > 255 ||
        hostmask1 < 0 || hostmask1 > 255 ||
        hostmask2 < 0 || hostmask2 > 255)
      throw(Exception("Not a valid subnet mask."));

    // compute the subnet address using bit-wise AND
    int sub_net1 = net1 & netmask1;
```

```
    int sub_net2 = net2 & netmask2;
    int sub_host1 = host1 & hostmask1;
    int sub_host2 = host2 & hostmask2;

    // display the subnet address
    MaskEdit3->Text =
      String(sub_net1) + "." + String(sub_net2) + "." +
      String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
```

```
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
```

```
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

30.1.18.3.14 TGroupBox.OnGetSiteInfo Event

Returns the control's docking information.

Pascal

```
property OnGetSiteInfo;
```

C++

```
__property OnGetSiteInfo;
```

Description

Write code in the OnGetSiteInfo event handler to initialize a TDragDockObject with information about where a dragged object can be docked, if at all. OnGetSiteInfo occurs before the OnDockOver event.

OnGetSiteInfo occurs only if DockSite is true.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
```

```
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
  //Make DockManager repaints it's clients.
end;

procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer; State: TDragState;
  var Accept: Boolean);
var
  ARect: TRect;
begin
  Accept := Source.Control is TDockableForm;
  if Accept then
  begin
    //Modify the DockRect to preview dock area.
    ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
    ARect.BottomRight := LeftDockPanel.ClientToScreen(
      Point(Self.ClientWidth div 3, LeftDockPanel.Height));
    Source.DockRect := ARect;
```

```
  end;
end;

procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, False, nil);
end;

procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
  DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
  var CanDock: Boolean);
begin
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock := DockClient is TDockableForm;
end;
```

See Also

DockSite

OnDockDrop

OnDockOver

TDragDockObject

30.1.18.3.15 TGroupBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.18.3.16 TGroupBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
```

```
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
```

```
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.18.3.17 TGroupBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.18.3.18 TGroupBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.18.3.19 TGroupBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.18.3.20 TGroupBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.18.3.21 TGroupBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.18.3.22 TGroupBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

30.1.18.3.23 TGroupBox.OnUnDock Event

Occurs when the application tries to undock a control that is docked to the windowed control.

Pascal

```
property OnUnDock;
```

C++

```
__property OnUnDock;
```

Description

Write an OnUnDock event handler to adjust the windowed control when a dock client is undocked. The OnUnDock event handler can make any necessary adjustments, or it can block the undock attempt by setting the Allow parameter to false.

C++ Examples:

```
/*
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
*/
void __fastcall TConjoinDockHost::FormUnDock(TObject *Sender, TControl *Client,
      TWinControl *NewTarget, bool &Allow)
{
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if (Client->ClassNameIs("TDockableForm"))
    dynamic_cast<TDockableForm *>(Client)->DockSite = True;
  if ((DockClientCount == 2) && (NewTarget != this))        // Self?
    PostMessage(dynamic_cast<TConjoinDockHost *>(this)->Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
}
/*
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
*/

void __fastcall TMainForm::LeftDockPanelDockDrop(TObject *Sender,
      TDragDockObject *Source, int X, int Y)
{
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, True, NULL);
  panel->DockManager->ResetBounds(True);
  //Make DockManager repaints it's clients.
}

void __fastcall TMainForm::LeftDockPanelDockOver(TObject *Sender,
      TDragDockObject *Source, int X, int Y, TDragState State, bool &Accept)
{
  Accept = (dynamic_cast<TDockableForm*>(Source->Control) != NULL);
  if (Accept)
  {
    // Modify the DockRect to preview dock area.
```

```
    Types::TPoint TopLeft = LeftDockPanel->ClientToScreen(Point(0, 0));
    Types::TPoint BottomRight = LeftDockPanel->ClientToScreen(
      Point(this->ClientWidth/3, LeftDockPanel->Height));
    Source->DockRect = Types::TRect(TopLeft, BottomRight);
  }
}

void __fastcall TMainForm::LeftDockPanelUnDock(TObject *Sender,
      TControl *Client, TWinControl *NewTarget, bool &Allow)
{
  //OnUnDock gets called BEFORE the client is undocked, in order to optionally
  //disallow the undock. DockClientCount is never 0 when called from this event.
  TPanel *panel = dynamic_cast<TPanel *>(Sender);
  if (panel->DockClientCount == 1)
    ShowDockPanel(panel, False, NULL);
}

void __fastcall TMainForm::LeftDockPanelGetSiteInfo(TObject *Sender,
      TControl *DockClient, TRect &InfluenceRect, TPoint &MousePos,
      bool &CanDock)
{
  //if CanDock is true, the panel will not automatically draw the preview rect.
  CanDock = DockClient->ClassNameIs("TDockableForm");
  if (!CanDock)
  {
     ShowMessage(L"GetSiteInfo CanDock went bad!");
  }
}
```

Delphi Examples:

```
{
The following example is taken from the docking demo. It
shows how the OnUnDock event handler of the conjoinment
docking site re-enables docking in the control that is
undocked (if it is a dockable form). In addition, when the
next to last docked control is undocked, the conjoinment
docking site sends itself a close message so that the last
docked control is undocked to its old position and size.
}
procedure TConjoinDockHost.FormUnDock(Sender: TObject; Client: TControl;
  NewTarget: TWinControl; var Allow: Boolean);
begin
  // Only 2 dock clients means the host must be destroyed and
  // the remaining window undocked to its old position and size.
  // Recall that OnUnDock gets called before the undocking
  // actually occurs.
  if Client is TDockableForm then
    TDockableForm(Client).DockSite := True;
  if (DockClientCount = 2) and (NewTarget <> Self) then
    PostMessage(Self.Handle, WM_CLOSE, 0, 0);
  UpdateCaption(Client);
end;
{
The following example is taken from the docking demo. It
shows how set permissions to dock dockable objects onto a
docking site.
}

procedure TMainForm.LeftDockPanelDockDrop(Sender: TObject;
  Source: TDragDockObject; X, Y: Integer);
begin
  //OnDockDrop gets called AFTER the client has actually docked,
  //so we check for DockClientCount = 1 before making the dock panel visible.
  if (Sender as TPanel).DockClientCount = 1 then
    ShowDockPanel(Sender as TPanel, True, nil);
  (Sender as TPanel).DockManager.ResetBounds(True);
```

```
    //Make DockManager repaints it's clients.
  end;

  procedure TMainForm.LeftDockPanelDockOver(Sender: TObject;
    Source: TDragDockObject; X, Y: Integer; State: TDragState;
    var Accept: Boolean);
  var
    ARect: TRect;
  begin
    Accept := Source.Control is TDockableForm;
    if Accept then
    begin
      //Modify the DockRect to preview dock area.
      ARect.TopLeft := LeftDockPanel.ClientToScreen(Point(0, 0));
      ARect.BottomRight := LeftDockPanel.ClientToScreen(
        Point(Self.ClientWidth div 3, LeftDockPanel.Height));
      Source.DockRect := ARect;
    end;
  end;

  procedure TMainForm.LeftDockPanelUnDock(Sender: TObject; Client: TControl;
    NewTarget: TWinControl; var Allow: Boolean);
  begin
    //OnUnDock gets called BEFORE the client is undocked, in order to optionally
    //disallow the undock. DockClientCount is never 0 when called from this event.
    if (Sender as TPanel).DockClientCount = 1 then
      ShowDockPanel(Sender as TPanel, False, nil);
  end;

  procedure TMainForm.LeftDockPanelGetSiteInfo(Sender: TObject;
    DockClient: TControl; var InfluenceRect: TRect; MousePos: TPoint;
    var CanDock: Boolean);
  begin
    //if CanDock is true, the panel will not automatically draw the preview rect.
    CanDock := DockClient is TDockableForm;
  end;
```

See Also

DockSite

OnDockDrop

OnDockOver

30.1.19 TImageMargins Class

Margins of image on button.

Class Hierarchy

Pascal

```
  TImageMargins = class(TPersistent);
```

C++

```
  class TImageMargins : public TPersistent;
```

File

StdCtrls

Description

A button image may be positioned using margins. TImageMargins defines the margins of an image on a TButton (see page 2035). The margin values are in the Bottom (see page 2605), Top (see page 2606), Left (see page 2605) and Right (see page 2606) properties.

The OnChange (see page 2607) event is triggered when a margin changes.

The TButton.ImageMargins property contains a TImageMargins instance.

See Also

Bottom (see page 2605)

ImageMargins (see page 2058)

Left (see page 2605)

Right (see page 2606)

Top (see page 2606)

OnChange (see page 2607)

30.1.19.1 TImageMargins Members

The following tables list the members exposed by TImageMargins.

Protected Methods

	Name	Description
	Change (see page 2604)	Called when the image margins change.

Public Events

	Name	Description
	OnChange (see page 2607)	Margin changed.

Public Methods

	Name	Description
	Assign (see page 2604)	Copies the contents of another, similar object.

Published Properties

	Name	Description
	Bottom (see page 2605)	Bottom margin of image on button.
	Left (see page 2605)	Left margin of image on button.
	Right (see page 2606)	Right margin of image on button.
	Top (see page 2606)	Top margin of image on button.

30.1.19.2 TImageMargins Methods

The methods of the TImageMargins class are listed here.

Protected Methods

	Name	Description
	Change (see page 2604)	Called when the image margins change.

Public Methods

	Name	Description
	Assign (see page 2604)	Copies the contents of another, similar object.

30.1.19.2.1 TImageMargins.Assign Method

Copies the contents of another, similar object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy the properties or other attributes of one object from another. The standard form of a call to Assign is

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

which tells the Destination object to copy the contents of the Source object to itself.

Most objects override Assign to handle the assignment of properties from similar objects. When overriding Assign, call the inherited method if the destination object can't handle the assignment of properties from the class of the Source parameter.

If no overridden Assign method can handle the assignment of properties from Source, the method implemented in TPersistent calls the source object's AssignTo method. This allows the source object to handle the assignment. If the Source object is nil (Delphi) or NULL (C++), Assign raises an EConvertError exception.

In general, the statement

```
Destination := Source; {Delphi}
Destination = Source; // C++
```

is not the same as the statement

```
Destination.Assign(Source); {Delphi}
Destination->Assign(Source); // C++
```

The assignment operator makes Destination reference the same object as Source, whereas the Assign method copies the contents of the object referenced by Source into the object referenced by Destination.

Note: The types of some properties are also objects. If these properties have write methods that use Assign to set the value of the property, then in these cases the assignment operator does the same thing as the Assign method.

See Also

AssignTo

EConvertError (see page 3919)

30.1.19.2.2 TImageMargins.Change Method

Called when the image margins change.

Pascal

```
procedure Change; virtual;
```

C++

```
virtual __fastcall Change();
```

Description

Override Change in a derived class to perform actions when the image margins change.

By default, Change calls the OnChange (see page 2607) event handler.

See Also

ImageMargins (see page 2058)

OnChange (see page 2607)

30.1.19.3 TImageMargins Properties

The properties of the TImageMargins class are listed here.

Published Properties

	Name	Description
	Bottom (see page 2605)	Bottom margin of image on button.
	Left (see page 2605)	Left margin of image on button.
	Right (see page 2606)	Right margin of image on button.
	Top (see page 2606)	Top margin of image on button.

30.1.19.3.1 TImageMargins.Bottom Property

Bottom margin of image on button.

Pascal

```
property Bottom: Integer;
```

C++

```
__property int Bottom;
```

Description

A button image may be positioned using margins. Bottom is the bottom margin of an image on a TButton (see page 2035).

See Also

ImageMargins (see page 2058)

Left (see page 2605)

Right (see page 2606)

Top (see page 2606)

30.1.19.3.2 TImageMargins.Left Property

Left margin of image on button.

Pascal

```
property Left: Integer;
```

C++

```
__property int Left;
```

Description

A button image may be positioned using margins. Left is the left margin of an image on a TButton (see page 2035).

See Also

Bottom (see page 2605)

ImageMargins (see page 2058)

Right (see page 2606)

Top (see page 2606)

30.1.19.3.3 TImageMargins.Right Property

Right margin of image on button.

Pascal

```
property Right: Integer;
```

C++

```
__property int Right;
```

Description

A button image may be positioned using margins. Right is the right margin of an image on a TButton (see page 2035).

See Also

Bottom (see page 2605)

ImageMargins (see page 2058)

Left (see page 2605)

Top (see page 2606)

30.1.19.3.4 TImageMargins.Top Property

Top margin of image on button.

Pascal

```
property Top: Integer;
```

C++

```
__property int Top;
```

Description

A button image may be positioned using margins. Top is the top margin of an image on a TButton (see page 2035).

See Also

Bottom (see page 2605)

ImageMargins (see page 2058)

Left (see page 2605)

Right (see page 2606)

30.1.19.4 TImageMargins Events

The events of the TImageMargins class are listed here.

Public Events

	Name	Description
	OnChange (see page 2607)	Margin changed.

30.1.19.4.1 TImageMargins.OnChange Event

Margin changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

OnChange occurs when a margin changes.

See Also

ImageMargins (see page 2058)

30.1.20 TLabel Class

TLabel is a nonwindowed control that displays text on a form.

Class Hierarchy

Pascal

```
TLabel = class(TCustomLabel);
```

C++

```
class TLabel : public TCustomLabel;
```

File

StdCtrls

Description

Use TLabel to add text that the user can't edit to a form. This text can be used to label another control, and can set focus to that control when the user types an accelerator key.

Because TLabel is not a descendant of TWinControl, it does not have its own window and can't receive direct input from the keyboard. To add an object to a form that can respond to keyboard input (other than setting focus to another object when an accelerator key is typed) in addition to displaying text, use TStaticText (see page 2945).

To add an object to a form that displays text that a user can scroll or edit, use TEdit (see page 2463).

See Also

TEdit (see page 2463)

TStaticText (see page 2945)

Labels

Display Controls

30.1.20.1 TLabel Members

The following tables list the members exposed by TLabel.

Protected Methods

	Name	Description
	AdjustBounds (see page 2385)	Resizes the label when the text changes.
	DoDrawText (see page 2386)	Renders the text in the client area.
V	GetLabelText (see page 2386)	Returns the value of the Caption (see page 2392) property.
V	Loaded (see page 2387)	Adjusts the size of the label when it is first loaded. .
V	Notification (see page 2387)	Responds to notifications when objects are about to be created or destroyed.
V	Paint (see page 2387)	Draws the image of the label.
V	SetAutoSize (see page 2388)	Sets the AutoSize (see page 2389) property.

Public Methods

	Name	Description
V	Create (see page 2385)	Creates and initializes an instance of TCustomLabel (see page 2384).

Public Properties

	Name	Description
	Canvas (see page 2390)	Provides the drawing surface used by the graphic control.

Published Events

	Name	Description
	OnClick (see page 2637)	Occurs when the user clicks the control.
	OnContextPopup (see page 2639)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2640)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2640)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2643)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2646)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2647)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2648)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2649)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2651)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2651)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2652)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2654)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2656)	Occurs when the user begins to drag a control with a DragKind of dkDock.

	OnStartDrag (see page 2657)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2610)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2613)	Controls the horizontal placement of the text within the label.
	Anchors (see page 2614)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2614)	Determines whether the size of the label automatically resizes to accommodate the text.
	BiDiMode (see page 2615)	Specifies the bi-directional mode for the control.
	Caption (see page 2616)	Specifies a text string that identifies the control to the user.
	Color (see page 2618)	Specifies the background color of the control.
	Constraints (see page 2619)	Specifies the size constraints for the control.
	DragCursor (see page 2620)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2620)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2621)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EllipsisPosition (see page 2622)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	Enabled (see page 2623)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 2623)	Designates a windowed control associated with the label.
	Font (see page 2623)	Controls the attributes of text written on or in the control.
	GlowSize (see page 2624)	Specifies the radius of the glow around the label.
	Layout (see page 2625)	Specifies the vertical placement of the text within the label.
	ParentBiDiMode (see page 2625)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2626)	Determines where a control looks for its color information.
	ParentFont (see page 2626)	Determines where a control looks for its font information.
	ParentShowHint (see page 2627)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2627)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2632)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 2633)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 2635)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 2635)	Determines whether the component appears onscreen.
	WordWrap (see page 2636)	Specifies whether the label text wraps when it is too long for the width of the label.

30.1.20.2 TLabel Properties

The properties of the TLabel class are listed here.

Published Properties

	Name	Description
	Align (see page 2610)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2613)	Controls the horizontal placement of the text within the label.
	Anchors (see page 2614)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2614)	Determines whether the size of the label automatically resizes to accommodate the text.
	BiDiMode (see page 2615)	Specifies the bi-directional mode for the control.
	Caption (see page 2616)	Specifies a text string that identifies the control to the user.
	Color (see page 2618)	Specifies the background color of the control.
	Constraints (see page 2619)	Specifies the size constraints for the control.
	DragCursor (see page 2620)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2620)	Specifies whether the control is being dragged normally or for docking.

	DragMode (see page 2621)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	EllipsisPosition (see page 2622)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	Enabled (see page 2623)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 2623)	Designates a windowed control associated with the label.
	Font (see page 2623)	Controls the attributes of text written on or in the control.
	GlowSize (see page 2624)	Specifies the radius of the glow around the label.
	Layout (see page 2625)	Specifies the vertical placement of the text within the label.
	ParentBiDiMode (see page 2625)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2626)	Determines where a control looks for its color information.
	ParentFont (see page 2626)	Determines where a control looks for its font information.
	ParentShowHint (see page 2627)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2627)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2632)	Determines how an ampersand in the label text is displayed.
	ShowHint (see page 2633)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Transparent (see page 2635)	Specifies whether controls that sit below the label on a form can be seen through the label.
	Visible (see page 2635)	Determines whether the component appears onscreen.
	WordWrap (see page 2636)	Specifies whether the label text wraps when it is too long for the width of the label.

30.1.20.2.1 TLabel.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
```

```
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
```

```
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

30.1.20.2.2 TLabel.Alignment Property

Controls the horizontal placement of the text within the label.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap property is true and the label includes more than one line of text.

See Also

ClientRect

GetLabelText (see page 2386)

Layout (see page 2396)

WordWrap (see page 2397)

30.1.20.2.3 TLabel.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.20.2.4 TLabel.AutoSize Property

Determines whether the size of the label automatically resizes to accommodate the text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

Use AutoSize to make the label adjust its size automatically so the client area accommodates the height and width of the text. When AutoSize is false, the label is fixed in size. When AutoSize is true, the size of the label readjusts whenever the text changes. The size of the label is also readjusts when the Font property changes.

When WordWrap is true, the width of the label is fixed. If AutoSize is also true, changes to the text cause the label to change in height. When AutoSize is true and WordWrap is false, the font determines the height of the label, and changes to the text cause the label to change in width.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
```

See Also

BoundsRect

GetLabelText (see page 2386)

Font

WordWrap (see page 2397)

Layout (see page 2396)

30.1.20.2.5 TLabel.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.20.2.6 TLabel.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
```

```
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

30.1.20.2.7 TLabel.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.20.2.8 TLabel.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.20.2.9 TLabel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.20.2.10 TLabel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.20.2.11 TLabel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
```

```
    AttachMode: TNodeAttachMode;
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.20.2.12 TLabel.EllipsisPosition Property

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
property EllipsisPosition: TEllipsisPosition;
```

C++

```
__property TEllipsisPosition EllipsisPosition;
```

Description

Set EllipsisPosition to determine how an ellipsis is placed in text that does not fit in its allocated rectangle. This is a TEllipsisPosition constant and its value corresponds to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

See Also

WordWrap (see page 2397)

30.1.20.2.13 TLabel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.20.2.14 TLabel.FocusControl Property

Designates a windowed control associated with the label.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Set FocusControl to the windowed control that should receive focus when the user presses the accelerator key specified by the label. Specify an accelerator key by preceding a character in the label text with an ampersand (&) and setting the ShowAccelChar property to true.

See Also

ShowAccelChar (see page 2396)

GetLabelText (see page 2386)

30.1.20.2.15 TLabel.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.20.2.16 TLabel.GlowSize Property

Specifies the radius of the glow around the label.

Pascal

```
property GlowSize: Integer;
```

C++

```
__property int GlowSize;
```

Description

Set the value of GlowSize to specify the radius of the glow around the label.

Note: This is available only on Windows Vista with the Aero theme enabled.

See Also

Hint

Transparent (see page 2397)

Alignment (see page 2388)

30.1.20.2.17 TLabel.Layout Property

Specifies the vertical placement of the text within the label.

Pascal

```
property Layout: TTextLayout;
```

C++

```
__property TTextLayout Layout;
```

Description

Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout is the vertical analog to the Alignment property.

See Also

Alignment (see page 2388)

ClientRect

30.1.20.2.18 TLabel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.20.2.19 TLabel.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.20.2.20 TLabel.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.20.2.21 TLabel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.20.2.22 TLabel.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
```

```
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
```

```
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
```

```
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.20.2.23 TLabel.ShowAccelChar Property

Determines how an ampersand in the label text is displayed.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

Set ShowAccelChar to true to allow the label to display an underlined accelerator key value. When ShowAccelChar is true, any character preceded by an ampersand (&) appears underlined. If the FocusControl property is set, the windowed control specified by the FocusControl property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is true, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to false to display the label text with all ampersands appearing as ampersands. When ShowAccelChar is false, the value of the FocusControl property is not used.

C++ Examples:

```
/*
This example uses two labels on a form. The first label has
```

```
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label1->ShowAccelChar = true;
  Label1->Caption = "An &Underlined character appears here";
  Label2->ShowAccelChar = false;
  Label2->Caption = "An ampersand (&) appears here";
}
```

Delphi Examples:

```
{
This example uses two labels on a form. The first label has
a caption with an accelerator character in it. The second
label also includes an ampersand, but it does not appear as
an accelerator character.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See Also

FocusControl (see page 2395)

GetLabelText (see page 2386)

30.1.20.2.24 TLabel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
```

```
  to both controls. The last statement enables the Help Hints
  for the entire application.
  */
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    Edit1->Hint = "Enter your name";
    Edit1->ShowHint = true;

    char string[10];
    char index[3];
    for(int i = 1; i <= 10; i++)
    {
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
  {
  This example uses an edit box and a list box on a form.
  Items are added to the list box and a Help Hint is assigned
  to both controls. The last statement enables the Help Hints
  for the entire application.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    I: Integer;
  begin
    Edit1.Hint := 'Enter your name';
    Edit1.ShowHint := True;
    with ListBox1 do
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.20.2.25 TLabel.Transparent Property

Specifies whether controls that sit below the label on a form can be seen through the label.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to true so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is false. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to false.

See Also

Color

ParentColor

30.1.20.2.26 TLabel.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.20.2.27 TLabel.WordWrap Property

Specifies whether the label text wraps when it is too long for the width of the label.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the label to display multiple line of text. When WordWrap is true, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to false to limit the label to a single line. When WordWrap is false, text that is too wide for the label appears truncated.

See Also

ClientWidth

GetLabelText (see page 2386)

30.1.20.3 TLabel Events

The events of the TLabel class are listed here.

Published Events

	Name	Description
	OnClick (see page 2637)	Occurs when the user clicks the control.
	OnContextPopup (see page 2639)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2640)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2640)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2643)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2646)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2647)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2648)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2649)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2651)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2651)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2652)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2654)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2656)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2657)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.20.3.1 TLabel.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.20.3.2 TLabel.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The

OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.20.3.3 TLabel.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.20.3.4 TLabel.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
```

```
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.20.3.5 TLabel.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor

property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
```

```
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
```

```
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.20.3.6 TLabel.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.20.3.7 TLabel.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
```

```
    S := (Sender as TLabel).Name + ' was dropped... and ';
    if Target <> nil then
      S := S + 'accepted!'
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.20.3.8 TLabel.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate

event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.20.3.9 TLabel.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
```

```
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
```

```
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.20.3.10 TLabel.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.20.3.11 TLabel.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.20.3.12 TLabel.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
```

```
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
```

```
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.20.3.13 TLabel.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
```

```
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
```

```
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.20.3.14 TLabel.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.20.3.15 TLabel.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

30.1.21 TListBox Class

TListBox displays a collection of items in a scrollable list.

Class Hierarchy

Pascal

```
TListBox = class(TCustomListBox);
```

C++

```
class TListBox : public TCustomListBox;
```

File

StdCtrls

Description

Use TListBox to display a scrollable list of items that users can select, add, or delete. TListBox is a wrapper for the Windows listbox control. For specialized list boxes, use other descendant classes of TCustomListBox (see page 2398) or derive your own class from TCustomListBox (see page 2398).

TListBox implements the generic behavior introduced in TCustomListBox (see page 2398). TListBox publishes many of the properties inherited from TCustomListBox (see page 2398), but does not introduce any new behavior.

See Also

List Controls

List Boxes and Check-list Boxes

30.1.21.1 TListBox Members

The following tables list the members exposed by TListBox.

Protected Fields

	Name	Description
	FMoving (see page 2401)	Specifies whether the strings of the list box are changing their position.

Protected Methods

	Name	Description
	CreateParams (see page 2404)	Initializes the window-creation parameter record passed in the Params parameter.
	CreateWnd (see page 2405)	Creates a Windows control corresponding to the list box component.
	DeleteString (see page 2406)	Deletes a string from the list box.
	DestroyWnd (see page 2406)	Saves the list box items in a string list before freeing the underlying Windows control.
	DoFindData (see page 2407)	Returns the index of a string in the list box.
	DoGetData (see page 2407)	Returns a string in the list box given its index.
	DoGetDataObject (see page 2408)	Returns the object associated with an item in a virtual list box.
	DragCanceled (see page 2408)	Sends a mouse message canceling the drag-and-drop operation.
	DrawItem (see page 2408)	Generates an OnDrawItem (see page 2439) event.
	GetItemData (see page 2409)	Returns a 32-bit value associated with the specified item in a list box.
	GetItemIndex (see page 2410)	Returns the value of the ItemIndex property.
	GetSelCount (see page 2410)	Returns the number of selected items.
	GetSelected (see page 2410)	Indicates whether a particular item is selected.
	InternalGetItemData (see page 2411)	Retrieves the data associated with an item in the list box.
	InternalSetItemData (see page 2411)	Associates an integer with a specified item in the list box.
	KeyPress (see page 2415)	Responds when the user presses a key.
	LoadRecreateItems (see page 2415)	Assigns new strings to the list.
	MeasureItem (see page 2415)	Generates an OnMeasureItem (see page 2442) event.
	ResetContent (see page 2416)	Removes all of the items from a list box.
	SaveRecreateItems (see page 2416)	Saves the items from a list box.
	SetItemData (see page 2417)	Sets the 32-bit value associated with the specified item in a list box.
	SetItemIndex (see page 2417)	Gives focus to the item with a specified index.
	SetMultiSelect (see page 2417)	Sets the MultiSelect property.

	WndProc (see page 2418)	Handles Windows messages.

Public Methods

	Name	Description
	AddItem (see page 2402)	Adds an item to the list box
	Clear (see page 2403)	Deletes all items from the list box.
	ClearSelection (see page 2403)	Deselects all selected items in the list box, if possible.
	CopySelection (see page 2404)	Copies the selected items to another list-type control
	Create (see page 2400)	Instantiates a list box control.
	DeleteSelected (see page 2406)	Deletes the selected items from the list box
	Destroy (see page 2401)	Destroys the instance of a list box control.
	GetCount (see page 2409)	Returns the value of the Count (see page 2427) property.
	ItemAtPos (see page 2411)	Returns the index of the list box item indicated by the coordinates of a point on the control.
	ItemRect (see page 2413)	Returns the rectangle that surrounds the item specified in the Item parameter.
	SelectAll (see page 2416)	Selects the entire block of text in the list box.

Public Properties

	Name	Description
	Canvas (see page 2424)	Provides a drawing surface when implementing a handler for the OnDrawItem (see page 2439) event of an owner-draw list box.
	Count (see page 2427)	Indicates the number of items in the list box.
	Selected (see page 2432)	Indicates whether a particular item is selected.
	TopIndex (see page 2437)	Specifies the index number of the item that appears at the top of the list box.

Published Events

	Name	Description
	OnClick (see page 2699)	Occurs when the user clicks the control.
	OnContextPopup (see page 2701)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 2701)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2702)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2702)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 2703)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2703)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2706)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2709)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 2711)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2712)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2713)	Occurs when a control receives the input focus.
	OnExit (see page 2715)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2719)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2723)	Occurs when a key is pressed.
	OnKeyUp (see page 2724)	Occurs when the user releases a key that has been pressed.
	OnMeasureItem (see page 2725)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 2727)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2728)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2730)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2731)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2731)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 2733)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2735)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2736)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2662)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2665)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 2666)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2666)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BevelEdges (see page 2666)	Specifies which edges of the control are beveled.
	BevelInner (see page 2667)	Specifies the cut of the inner bevel.
	BevelKind (see page 2667)	Specifies the control's bevel style.
	BevelOuter (see page 2668)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2668)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2668)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2669)	Determines whether the list box has a border.
	Color (see page 2669)	Specifies the background color of the control.
	Columns (see page 2670)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 2671)	Specifies the size constraints for the control.
	Ctl3D (see page 2671)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2672)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2674)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2674)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2674)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2676)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 2676)	Makes list item multi-select operate with Shift and CTRL keys.
	Font (see page 2677)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2678)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2678)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 2679)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2680)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 2680)	Contains the strings that appear in the list box.
	MultiSelect (see page 2683)	Specifies whether the user can select more than one item.
	ParentBiDiMode (see page 2684)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2684)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2685)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2685)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2685)	Determines where a control looks for its font information.
	ParentShowHint (see page 2686)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2686)	Identifies the pop-up menu associated with the control.
	ScrollWidth (see page 2691)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	ShowHint (see page 2692)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2693)	Specifies whether the items in a list box are arranged alphabetically.
	Style (see page 2695)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 2695)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2696)	Determines if the user can tab to a control.

	TabWidth (see page 2696)	Specifies the size of the tabs in the list box.
	Visible (see page 2697)	Determines whether the component appears onscreen.

30.1.21.2 TListBox Properties

The properties of the TListBox class are listed here.

Published Properties

	Name	Description
	Align (see page 2662)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2665)	Specifies how the control is anchored to its parent.
	AutoComplete (see page 2666)	Determines whether the user can give focus to items by typing in the list.
	AutoCompleteDelay (see page 2666)	Specifies the delay between a key press and an attempt to autocomplete the field.
	BevelEdges (see page 2666)	Specifies which edges of the control are beveled.
	BevelInner (see page 2667)	Specifies the cut of the inner bevel.
	BevelKind (see page 2667)	Specifies the control's bevel style.
	BevelOuter (see page 2668)	Specifies the cut of the outer bevel.
	BevelWidth (see page 2668)	Specifies the width of the inner and outer bevels.
	BiDiMode (see page 2668)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2669)	Determines whether the list box has a border.
	Color (see page 2669)	Specifies the background color of the control.
	Columns (see page 2670)	Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.
	Constraints (see page 2671)	Specifies the size constraints for the control.
	Ctl3D (see page 2671)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2672)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2674)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2674)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2674)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2676)	Controls whether the control responds to mouse, keyboard, and timer events.
	ExtendedSelect (see page 2676)	Makes list item multi-select operate with Shift and CTRL keys.
	Font (see page 2677)	Controls the attributes of text written on or in the control.
	ImeMode (see page 2678)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2678)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	IntegralHeight (see page 2679)	Determines whether the list box displays the partial items.
	ItemHeight (see page 2680)	Specifies the height in pixels of an item in an owner-draw list box.
	Items (see page 2680)	Contains the strings that appear in the list box.
	MultiSelect (see page 2683)	Specifies whether the user can select more than one item.
	ParentBiDiMode (see page 2684)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2684)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2685)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2685)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2685)	Determines where a control looks for its font information.
	ParentShowHint (see page 2686)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2686)	Identifies the pop-up menu associated with the control.
	ScrollWidth (see page 2691)	Specifies the width, in pixels, by which the list box can scroll horizontally.
	ShowHint (see page 2692)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Sorted (see page 2693)	Specifies whether the items in a list box are arranged alphabetically.

	Style (see page 2695)	Determines whether the list box is standard or owner-draw and whether it is virtual.
	TabOrder (see page 2695)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2696)	Determines if the user can tab to a control.
	TabWidth (see page 2696)	Specifies the size of the tabs in the list box.
	Visible (see page 2697)	Determines whether the component appears onscreen.

30.1.21.2.1 TListBox.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);
```

```
  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
```

```
  into resizable panes.  All objects on the form are created
  dynamically in the OnCreate event handler of the form. To
  run this example, add ExtCtrls and FileCtrl to the uses
  clause of the form's implementation section.
  Note that when creating a splitter dynamically at runtime,
  it is important to set its position to the appropriate side
  of the control it will resize.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
```

```
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

30.1.21.2.2 TListBox.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.21.2.3 TListBox.AutoComplete Property

Determines whether the user can give focus to items by typing in the list.

Pascal

```
property AutoComplete: Boolean;
```

C++

```
__property Boolean AutoComplete;
```

Description

Use AutoComplete to specify whether the list box automatically updates the ItemIndex property in response to user keystrokes. When AutoComplete is true, the item with focus changes to the first item that begins with the sequence of characters that the user typed. As the user types more characters, ItemIndex may change to a later item because the string to match becomes longer.

See Also

ItemIndex

30.1.21.2.4 TListBox.AutoCompleteDelay Property

Specifies the delay between a key press and an attempt to autocomplete the field.

Pascal

```
property AutoCompleteDelay: Cardinal;
```

C++

```
__property unsigned AutoCompleteDelay;
```

Description

Specifies the delay in milliseconds between a key press and an attempt to autocomplete the field. Autocomplete will position to matching list values as you type, sometimes saving you keystrokes.

AutoCompleteDelay is only in effect when the AutoComplete (see page 2666) is true.

See Also

AutoComplete (see page 2666)

30.1.21.2.5 TListBox.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

30.1.21.2.6 TListBox.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

30.1.21.2.7 TListBox.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

30.1.21.2.8 TListBox.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

30.1.21.2.9 TListBox.BevelWidth Property

Specifies the width of the inner and outer bevels.

Pascal

```
property BevelWidth;
```

C++

```
__property BevelWidth;
```

Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

See Also

BevelEdges

BevelInner

BevelKind

BevelOuter

30.1.21.2.10 TListBox.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.21.2.11 TListBox.BorderStyle Property

Determines whether the list box has a border.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to change the appearance of the outer boundary of the list box control. A list box can have a single border (bsSingle) or no visible border (bsNone).

See Also

Ctl3D

30.1.21.2.12 TListBox.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the

Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.21.2.13 TListBox.Columns Property

Specifies the number of columns, in a multi-column list box, that are visible without having to scroll.

Pascal

```
property Columns: Integer;
```

C++

```
__property int Columns;
```

Description

Use Columns to specify the number of columns, in a multi-column list box, that are visible without having to use the horizontal scroll bar.

Multi-column list boxes have a horizontal scroll bar that allows users to view multiple columns as they wrap. The default value for Columns is 0, meaning that the list box is not multi-column. That is, users can scroll only vertically and the list of items will not wrap.

For Columns values greater than 0, multiple columns accommodate the items as they wrap beyond the bottom of the list box. The Columns property specifies the number of columns that are visible without having to horizontally scroll the list box.

The width of each column depends upon both the Width property and the number of Columns.

See Also

ItemHeight (see page 2428)

Width

30.1.21.2.14 TListBox.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.21.2.15 TListBox.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.21.2.16 TListBox.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
```

```
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.21.2.17 TListBox.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.21.2.18 TListBox.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.21.2.19 TListBox.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.21.2.20 TListBox.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.21.2.21 TListBox.ExtendedSelect Property

Makes list item multi-select operate with Shift and CTRL keys.

Pascal

```
property ExtendedSelect: Boolean;
```

C++

```
__property Boolean ExtendedSelect;
```

Description

ExtendedSelect only has effect when the MultiSelect (see page 2683) is true.

When ExtendedSelect is true, the Shift key (in conjunction with the left mouse button, or arrow keys) is used to select a contiguous sequence of list items. And the Ctrl key is used (in conjunction with the left mouse buttton) to select non-contiguous list items.

When ExtendedSelect is false, Multi-Select operates very differently. The Shift and Ctrl keys are not recognised. List items are always selected individually in any sequence. Reselecting a selected list item will deselect it. Use the arrow and spacebar keys, or the left mouse button to make selections.

See Also

MultiSelect (see page 2683)

30.1.21.2.22 TListBox.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
```

```
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.21.2.23 TListBox.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

30.1.21.2.24 TListBox.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

30.1.21.2.25 TListBox.IntegralHeight Property

Determines whether the list box displays the partial items.

Pascal

```
property IntegralHeight: Boolean;
```

C++

```
__property Boolean IntegralHeight;
```

Description

IntegralHeight controls the height of the list box itself.

When IntegralHeight is true, and the Style property is lsOwnerDrawFixed, the list box height will always be a multiple of the ItemHeight property value. In this case, items will never be partially displayed.

If IntegralHeight is false, the list box height can be set to any value, but the last item may not be fully displayed.

If the Style property value of the list box is lbOwnerDrawVariable, setting the IntegralHeight property to true has no effect.

See Also

ItemHeight (see page 2428)

Items (see page 2429)

Style (see page 2436)

Using TDBListBox and TDBComboBox

30.1.21.2.26 TListBox.ItemHeight Property

Specifies the height in pixels of an item in an owner-draw list box.

Pascal

```
property ItemHeight: Integer;
```

C++

```
__property int ItemHeight;
```

Description

Use ItemHeight to control the height of an item in a list box when the Style property is lbOwnerDrawFixed. The ItemHeight property can also affect the height of the list box itself. When the IntegralHeight property is true, the height of the list box is always an increment of the ItemHeight value.

If the Style property is lbStandard or lbOwnerDrawVariable, the value of ItemHeight is ignored.

See Also

DrawItem (see page 2408)

IntegralHeight (see page 2428)

Items (see page 2429)

MeasureItem (see page 2415)

OnDrawItem (see page 2439)

OnMeasureItem (see page 2442)

Style (see page 2436)

Using TDBListBox and TDBComboBox

30.1.21.2.27 TListBox.Items Property

Contains the strings that appear in the list box.

Pascal

```
property Items: TStrings;
```

C++

```
__property TStrings Items;
```

Description

Use Items to add, insert, delete and move items. By default, the items in a list box are of type TStrings. Use this item type to access its methods or properties to manipulate the items in the list.

For example, the following code adds the text in the edit box to the list box as an item:

```
ListBox1.Items.Add(Edit1.Text); {Delphi }
ListBox1->Items->Add(Edit1->Text); // C++
```

C++ Examples:

```
/*
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
```

```
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Continue will continue listing
file sizes in a multiple selection that includes the deleted
file.  Break will stop.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
*/
#include <stdio.h>     // for FILE, fopen, fstat, fileno, fread and fclose
#include <sys\stat.h>  // for fstat and the stat struct

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < FileListBox1->Items->Count; i++)
  {
    if (FileListBox1->Selected[i])
    {
      if (!FileExists(FileListBox1->Items->Strings[i]))
      {
        MessageBeep(0);
        if (Application->MessageBox(
          String("File " + FileListBox1->Items->Strings[i] + "not found").c_str(),
          NULL,
          MB_OKCANCEL) == IDCANCEL)
          break;
        else
          continue;
      }
      FILE *F = fopen(AnsiString(FileListBox1->Items->Strings[i]).c_str(),"r");
      struct stat statbuf;
      fstat(fileno(F), &statbuf);
      ListBox1->Items->Add(
          FileListBox1->Items->Strings[i] + ": " + IntToStr(int(statbuf.st_size)));
      fclose(F);
    }
  }
}
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
```

```
  ExtendedSelect properties on the FileListBox.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: File;
    i, filehandle: Integer;
  begin
    for i := 0 to (FileListBox1.Items.Count - 1) do begin
    try
      if FileListBox1.Selected[i] then
      begin
        if not FileExists(FileListBox1.Items.Strings[i]) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' not found', mtError, [mbOk], 0);
          Continue;
        end;
        filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
        if (filehandle = -1) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
     finally
     { do something here }
     end;
    end;
  end;
  {
  This example uses a list box and a button on a form. Use the
  Items property editor in the Object Inspector to enter a
  list of strings in the list box. When the application runs
  and the button is clicked, the strings in the list box
  become uppercase.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    I: Integer;
  begin
    for I := 0 to ListBox1.Items.Count -1 do
      ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    Listbox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;
```

See Also

Clear (see page 2403)

ItemIndex

OnDrawItem (see page 2439)

Selected (see page 2432)

TStrings

30.1.21.2.28 TListBox.MultiSelect Property

Specifies whether the user can select more than one item.

Pascal

```
property MultiSelect;
```

C++

```
__property MultiSelect;
```

Description

Use MultiSelect to specify whether the user can select more than one item. When MultiSelect is true, the user can select multiple items in the control, and the SelCount property indicates the number of selected items. When MultiSelect is false, the user can only select one item, which is indicated by the ItemIndex property.

Note: To implement the MultiSelect property in a TCustomMultiSelectListControl descendant, override the protected SetMultiSelect method.

C++ Examples:

```
/*
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
selects an item and clicks the button, the selected item in
the list box is moved to the top of the list box:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->MultiSelect = false;
  Button1->Caption = "Move to Top";
  for (int i = 1; i <= 10; i++)
    ListBox1->Items->Add("Item " + IntToStr(i));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ListBox1->ItemIndex >= 0 && ListBox1->ItemIndex <= 9)
    ListBox1->Items->Move(ListBox1->ItemIndex, 0);
}
```

Delphi Examples:

```
{
This example uses a list box and a button on a form. The
list box contains items when the form appears. When the user
clicks the button, the selected item in the list box is
moved to the top of the list box:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (ListBox1.ItemIndex in [0..9]) then
    ListBox1.Items.Move(ListBox1.ItemIndex, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
var
  I: Integer;
begin
  ListBox1.MultiSelect := False;
  Button1.Caption := 'Move to Top';
  for I := 1 to 10 do
    ListBox1.Items.Add('Item ' + IntToStr(I));
end;
```

See Also

SelCount

ItemIndex

30.1.21.2.29 TListBox.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.21.2.30 TListBox.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.21.2.31 TListBox.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.21.2.32 TListBox.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.21.2.33 TListBox.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own

Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.21.2.34 TListBox.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.21.2.35 TListBox.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
```

```
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
```

```
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.21.2.36 TListBox.ScrollWidth Property

Specifies the width, in pixels, by which the list box can scroll horizontally.

Pascal

```
property ScrollWidth: Integer;
```

C++

```
__property int ScrollWidth;
```

Description

Use ScrollWidth to get or set the logical width of the list box. When ScrollWidth is greater than the client width of the list box, the list box gets a horizontal scroll bar. When ScrollWidth is less than or equal to ClientWidth, the horizontal scroll bar disappears.

See Also

Columns (see page 2426)

ClientWidth

30.1.21.2.37 TListBox.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
```

```
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.21.2.38 TListBox.Sorted Property

Specifies whether the items in a list box are arranged alphabetically.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Use Sorted to sort the items by setting its value to true. If Sorted is false, the items are unsorted. When Sorted is true, items are automatically added or inserted into the list box in alphabetical order.

Note: Sorted has no effect if Listbox's style is lbVirtual or lbVirtualOwnerDraw

C++ Examples:

```
/*
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
}

void __fastcall TForm1::AddClick(TObject *Sender)
{
  ListBox1->Items->Add(Edit1->Text);
}

void __fastcall TForm1::SortClick(TObject *Sender)
{
  ListBox1->Sorted = True;
}
```

Delphi Examples:

```
{
This example uses an edit box, a list box, and two buttons
on a form. The buttons are named Add and Sort. When the user
clicks the Add button, the text in the edit box is added to
the list in the list box. Before the user clicks the sort
button, any new items are added to the end of the list.
After the user clicks the Sort button, the list in the list
box is sorted and remains sorted, even if additional strings
are added.  If the user clicks the sort button again, the
list box remains sorted, but any new strings are again added
to the end of the list.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;


procedure TForm1.AddClick(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.SortClick(Sender: TObject);
begin
  ListBox1.Sorted := True;
end;
```

See Also

Items (see page 2429)

Style (see page 2436)

30.1.21.2.39 TListBox.Style Property

Determines whether the list box is standard or owner-draw and whether it is virtual.

Pascal

```
property Style: TListBoxStyle;
```

C++

```
__property TListBoxStyle Style;
```

Description

Use Style to specify whether the list box is a standard list box that displays text strings, or an owner-draw list box that can display graphical images. Owner-draw list boxes let you display items in some nonstandard way. In this case, you must write the code to paint items in the list box.

In addition, Style controls whether the list box is virtual, which means that items are supplied dynamically using event handlers. By default, Style is lbStandard, meaning that the list box is not virtual and it displays each item as a string.

See Also

DrawItem (see page 2408)

ItemHeight (see page 2428)

Items (see page 2429)

MeasureItem (see page 2415)

OnDrawItem (see page 2439)

OnMeasureItem (see page 2442)

OnData (see page 2438)

OnDataFind (see page 2439)

OnDataObject (see page 2439)

30.1.21.2.40 TListBox.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.21.2.41 TListBox.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.21.2.42 TListBox.TabWidth Property

Specifies the size of the tabs in the list box.

Pascal

```
property TabWidth: Integer;
```

C++

```
__property int TabWidth;
```

Description

Set TabWidth to control the number of dialog base units displayed for each tab character that appears in the strings of a list box.

If TabWidth is set to 0, which is the default setting, the tabs default to the width specified by Windows, which is two dialog base units.

30.1.21.2.43 TListBox.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.21.3 TListBox Events

The events of the TListBox class are listed here.

Published Events

	Name	Description
	OnClick (see page 2699)	Occurs when the user clicks the control.
	OnContextPopup (see page 2701)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnData (see page 2701)	Occurs when a virtual list box needs to supply the text of an item.
	OnDataFind (see page 2702)	Occurs when a virtual list box needs to identify the index of an item given its text.
	OnDataObject (see page 2702)	Occurs when a virtual list box needs to supply the object associated with an item.
	OnDblClick (see page 2703)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2703)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2706)	Occurs when the user drags an object over a control.
	OnDrawItem (see page 2709)	Occurs when an item in an owner-draw list box needs to be redisplayed.
	OnEndDock (see page 2711)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2712)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2713)	Occurs when a control receives the input focus.
	OnExit (see page 2715)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2719)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2723)	Occurs when a key is pressed.
	OnKeyUp (see page 2724)	Occurs when the user releases a key that has been pressed.

	OnMeasureItem (see page 2725)	Occurs when the application needs to redisplay an item in a variable height owner-draw list box.
	OnMouseActivate (see page 2727)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2728)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2730)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2731)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2731)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2733)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2735)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2736)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.21.3.1 TListBox.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.21.3.2 TListBox.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.21.3.3 TListBox.OnData Event

Occurs when a virtual list box needs to supply the text of an item.

Pascal

```
property OnData: TLBGetDataEvent;
```

C++

```
__property TLBGetDataEvent OnData;
```

Description

You must supply an OnData event handler to dynamically supply the items in the list box when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnDataFind (see page 2439)

OnDataObject (see page 2439)

Style (see page 2436)

Count (see page 2427)

30.1.21.3.4 TListBox.OnDataFind Event

Occurs when a virtual list box needs to identify the index of an item given its text.

Pascal

```
property OnDataFind: TLBFindDataEvent;
```

C++

```
__property TLBFindDataEvent OnDataFind;
```

Description

You must supply an OnDataFind event handler to dynamically provide the mapping from list item to index when the Style property is lbVirtual or lbVirtualOwnerDraw.

See Also

OnData (see page 2438)

OnDataObject (see page 2439)

Style (see page 2436)

Count (see page 2427)

30.1.21.3.5 TListBox.OnDataObject Event

Occurs when a virtual list box needs to supply the object associated with an item.

Pascal

```
property OnDataObject: TLBGetDataObjectEvent;
```

C++

```
__property TLBGetDataObjectEvent OnDataObject;
```

Description

Supply an OnDataObject event handler to dynamically provide objects associated with the items in a virtual list box.

See Also

OnData (see page 2438)

OnDataFind (see page 2439)

Style (see page 2436)

Count (see page 2427)

30.1.21.3.6 TListBox.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.21.3.7 TListBox.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
```

```
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
```

```
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.21.3.8 TListBox.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}
```

```
void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
```

```
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.21.3.9 TListBox.OnDrawItem Event

Occurs when an item in an owner-draw list box needs to be redisplayed.

Pascal

```
property OnDrawItem: TDrawItemEvent;
```

C++

```
__property TDrawItemEvent OnDrawItem;
```

Description

Use OnDrawItem to write a handler for drawing of the items in list boxes with the Style values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

The size of the rectangle that contains the item is determined either by the ItemHeight property for fixed owner-draw list boxes or by the response to the OnMeasureItem event for variable owner-draw list boxes.

C++ Examples:

```
/*
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
*/
#include <memory>       //for STL auto_ptr class
```

```
Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);

  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Flowers", bitmap0);
  ImageList1->GetBitmap(1, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1);
  ImageList1->GetBitmap(2, bitmap2);
  ListBox1->Items->AddObject("Butterfly", bitmap2);
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = (dynamic_cast<TListBox *>(Control))->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(
      Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
      pBitmap,
      Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}
```

Delphi Examples:

```
{
Here is a typical handler for an OnDrawItem event. In the
example, a list box with the lbOwnerDrawFixed style draws a
bitmap to the left of each string.
Select the TListBox and set the ItemHeight property to 32.
Select the ImageList and set the Layout Height and Width to
32.  Double click on the ImageList to load the images.  Be
sure to load in the correct order, logo2, Sunflower,
butterfly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  MyList := TStringList.Create;
  bitmap0 := TBitmap.Create;
  bitmap1 := TBitmap.Create;
  bitmap2 := TBitmap.Create;
//  MyList.Add('Animal');
//  MyList.Add('Flowers');
//  MyList.Add('Butterfly');
//  ListBox1.Items.AddStrings(MyList);

  ImageList1.GetBitmap(0, bitmap0);
  ListBox1.Items.AddObject('Flowers', bitmap0);
  ImageList1.GetBitmap(1, bitmap1);
  ListBox1.Items.AddObject('Animal', bitmap1);
```

```
    ImageList1.GetBitmap(2, bitmap2);
    ListBox1.Items.AddObject('Butterfly', bitmap2);
  end;

  procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer; Rect:TRect; State:
  TOwnerDrawState);
  var
    Bitmap: TBitmap;      { temporary variable for the item's bitmap }
    Offset: Integer;      { text offset width }
  begin
    Bitmap := TBitmap.Create;
    with (Control as TListBox).Canvas do  { draw on control canvas, not on the form }
    begin
      FillRect(Rect);       { clear the rectangle }
      Offset := 2;          { provide default offset }
      Bitmap := TBitmap((Control as TListBox).Items.Objects[Index]);  { get the bitmap }
      if Bitmap <> nil then
      begin
        BrushCopy(
          Bounds(Rect.Left + Offset, Rect.Top, Bitmap.Width, Bitmap.Height),
          Bitmap,
          Bounds(0, 0, Bitmap.Width, Bitmap.Height),
          clRed);  {render bitmap}
        Offset := Bitmap.width + 6;    { add four pixels between bitmap and text}
      end;
      TextOut(Rect.Left + Offset, Rect.Top, (Control as TListBox).Items[Index])  { display the
  text }
    end;
  end;
```

See Also

DrawItem (see page 2408)

ItemHeight (see page 2428)

MeasureItem (see page 2415)

OnMeasureItem (see page 2442)

Style (see page 2436)

30.1.21.3.10 TListBox.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.21.3.11 TListBox.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.21.3.12 TListBox.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.21.3.13 TListBox.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
```

```
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';
```

```
    { use the Courier font in the maskedit boxes }
    MaskEdit1.Font.Name := 'Courier';
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
```

```
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

30.1.21.3.14 TListBox.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
```

```
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
```

```
    if Key = VK_F1 then
    begin
      Form1.Color := FormColor;
      if (ComboBox1.Text = 'KeyUp') then
        ComboBox1.Text := 'KeyUp2'
      else
        ComboBox1.Text := 'KeyUp';
      WriteLn(ComboBox1.Text);
    end;
  end;
  {
  The following code aborts a print job if the user presses
  Esc. Note that you should set KeyPreview to true to ensure
  that the OnKeyDown event handler of Form1 is called.
  }

  uses Printers;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    I, X, Y: Integer;
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.21.3.15 TListBox.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
```

```
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.21.3.16 TListBox.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
```

```
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

30.1.21.3.17 TListBox.OnMeasureItem Event

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.

Pascal

```
property OnMeasureItem: TMeasureItemEvent;
```

C++

```
__property TMeasureItemEvent OnMeasureItem;
```

Description

Use OnMeasureItem to write a handler to measure items to be drawn in a list box with a Style property value of lbOwnerDrawVariable.

OnMeasureItem is of type TMeasureItemEvent which contains three parameters describing the item to measure:

Control, specifying the control containing the item.

Index, specifying the index of the item in the control.

Height, specifying the height of the item.

The Index parameter identifies the position of the item in the list box.

The Height parameter should specify the height in pixels that the given item will occupy in the control. The Height parameter is passed by reference (a var parameter), which initially contains the default height of the item or the height of the item text in the control's font. The handler can set Height to a value appropriate to the contents of the item, such as the height of a graphical image to be displayed within the item.

After the OnMeasureItem event occurs, the OnDrawItem event occurs, rendering the item with the measured size.

C++ Examples:

```
/*
Here is a typical handler for an OnMeasureItem event. The
example assumes that a variable owner-draw list box already
has bitmaps associated with each of its strings. It sets the
height of the item to the height of the associated bitmap if
that height is greater than the default height
(ListBox1.ItemHeight).  Note that TWinControl can also be
used in place of TWidgetControl on Windows.
*/
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
      int &Height)
{
  Graphics::TBitmap *bitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (bitmap && bitmap->Height > Height)
    Height = bitmap->Height;
}

void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
      TRect &Rect, TOwnerDrawState State)
{
  Graphics::TBitmap *pBitmap; // temporary variable for item's bitmap
  int     Offset = 2;   // default text offset width
  // note that we draw on the listbox's canvas, not on the form
  TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
  pCanvas->FillRect(Rect); // clear the rectangle
  pBitmap = dynamic_cast<Graphics::TBitmap *>((dynamic_cast<TListBox
*>(Control))->Items->Objects[Index]);
  if (pBitmap)
  {
    pCanvas->BrushCopy(Bounds(Rect.Left + Offset, Rect.Top, pBitmap->Width, pBitmap->Height),
pBitmap, Bounds(0, 0, pBitmap->Width, pBitmap->Height), clRed); // render bitmap
    Offset += pBitmap->Width + 4;   // add four pixels between bitmap and text
  }
  // display the text
```

```
  pCanvas->TextOut(Rect.Left + Offset, Rect.Top, (dynamic_cast<TListBox
*>(Control))->Items->Strings[Index]);
}

#include <memory>       //for STL auto_ptr class

Graphics::TBitmap *bitmap0, *bitmap1, *bitmap2;
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Making these static keeps them around until the project terminates.
  static std::auto_ptr<Graphics::TBitmap> _bitmap0Cleaner(bitmap0 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap1Cleaner(bitmap1 = new Graphics::TBitmap);
  static std::auto_ptr<Graphics::TBitmap> _bitmap2Cleaner(bitmap2 = new Graphics::TBitmap);
  bitmap0->LoadFromFile("../littleB_64.bmp");
//  ImageList1->GetBitmap(0, bitmap0);
  ListBox1->Items->AddObject("Butterfly", bitmap0);
  ImageList1->GetBitmap(3, bitmap1);
  ListBox1->Items->AddObject("Animal", bitmap1); // no bitmap, see if it leaves this height at
64.
  ImageList1->GetBitmap(4, bitmap2);
  ListBox1->Items->AddObject("Sunflower", bitmap2); // no bitmap, see if it leaves this height
at 64.

}
```

See Also

DrawItem (see page 2408)

ItemHeight (see page 2428)

MeasureItem (see page 2415)

OnDrawItem (see page 2439)

Style (see page 2436)

30.1.21.3.18 TListBox.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.21.3.19 TListBox.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
```

```
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.21.3.20 TListBox.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.21.3.21 TListBox.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.21.3.22 TListBox.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.21.3.23 TListBox.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.21.3.24 TListBox.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.21.3.25 TListBox.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

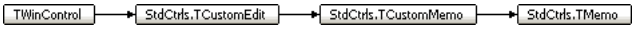
OnDragOver

OnDragDrop

30.1.22 TMemo Class

TMemo is a wrapper for a Windows multiline edit control.

Class Hierarchy



Pascal

```
TMemo = class(TCustomMemo);
```

C++

```
class TMemo : public TCustomMemo;
```

File

StdCtrls

Description

Use TMemo to put a standard Windows multiline edit control on a form. Multiline edit boxes allow the user to enter more than one line of text. They are appropriate for representing lengthy information.

TMemo implements the generic behavior introduced in TCustomMemo (see page 2443). TMemo publishes many of the properties inherited from TCustomMemo (see page 2443), but does not introduce any new behavior.

Note: Under Win 9x, there is a limit of 64k of text for this control.

See Also

TDBMemo

TEdit (see page 2463)

TRichEdit

Displaying and Editing Text in a Memo Control

Adding Scroll Bars at Runtime

Setting Text Alignment

Memo and Rich Edit Controls

Edit Controls

30.1.22.1 TMemo Members

The following tables list the members exposed by TMemo.

Protected Methods

	Name	Description
	Change (see page 2348)	Generates an OnChange (see page 2381) event.
V	CreateParams (see page 2351)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateParams (see page 2447)	Initializes a window-creation parameter record passed in the Params parameter.
V	CreateWindowHandle (see page 2351)	Creates a Windows control to represent the control.
V	CreateWindowHandle (see page 2447)	Creates a window based on the window-creation parameter record.
V	CreateWnd (see page 2352)	Creates a Windows control corresponding to the edit control.

	DestroyWnd (see page 2354)	Destroys the Windows control that corresponds to the edit control.
	DoSetMaxLength (see page 2354)	Sets the MaxLength (see page 2368) property.
	DoSetTextHint (see page 2355)	Sets the value of TextHint (see page 2380).
	GetCaretPos (see page 2448)	Returns the value of the CaretPos (see page 2450) property.
	GetSelLength (see page 2356)	Returns the SelLength (see page 2371) property.
	GetSelStart (see page 2356)	Returns the SelStart (see page 2374) property.
	GetSelText (see page 2356)	Returns the SelText (see page 2377) property.
	KeyPress (see page 2448)	Ensures that return characters are not entered into the text when WantReturns (see page 2454) is false.
	Loaded (see page 2448)	Initializes the memo control when it is first loaded into memory.
	SetAlignment (see page 2361)	Sets the Alignment (see page 2365) property.
	SetAutoSize (see page 2361)	Sets the AutoSize (see page 2366) property.
	SetCaretPos (see page 2449)	Sets the CaretPos (see page 2450) property.
	SetLines (see page 2449)	Sets the Lines (see page 2451) property.
	SetScrollBars (see page 2449)	Sets the ScrollBars (see page 2452) property.
	SetSelLength (see page 2361)	Sets the SelLength (see page 2371) property.
	SetSelStart (see page 2362)	Sets the SelStart (see page 2374) property.
	SetWordWrap (see page 2450)	Sets the WordWrap (see page 2455) property.

Protected Properties

	Name	Description
	AutoSelect (see page 2365)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
	AutoSize (see page 2366)	Determines whether the height of the edit control automatically resizes to accommodate the text.
	NumbersOnly (see page 2369)	Allows only numbers to be typed into the text edit.
	PasswordChar (see page 2370)	Indicates the character, if any, to display in place of the actual characters typed in the control.

Public Methods

	Name	Description
	Clear (see page 2348)	Deletes all text from the edit control.
	ClearSelection (see page 2349)	Removes the selected text from the edit control.
	ClearUndo (see page 2349)	Clears the undo buffer so that no changes to the text can be backed out.
	CopyToClipboard (see page 2349)	Copies the selected text in the edit control to the Clipboard in CF_TEXT format.
	Create (see page 2347)	Creates an instance of TCustomEdit (see page 2345).
	Create (see page 2446)	Creates and initializes an instance of TCustomMemo (see page 2443).
	CutToClipboard (see page 2352)	Copies the selected text to the Clipboard in CF_TEXT format and then deletes the selection.
	DefaultHandler (see page 2354)	Provides message handling for all messages that the control does not fully process itself.
	Destroy (see page 2446)	Destroys an instance of TCustomMemo (see page 2443).
	GetControlsAlignment (see page 2355)	Indicates how text is aligned within the control.
	GetSelTextBuf (see page 2357)	Copies the selected text into a buffer and returns the number of characters copied.
	PasteFromClipboard (see page 2358)	Pastes the contents of the Clipboard into edit control, replacing the current selection.
	SelectAll (see page 2359)	Selects all text in the edit control.
	SetSelText (see page 2362)	Sets the SelText (see page 2377) property.
	SetSelTextBuf (see page 2363)	Replaces the selected text with a null-terminated string.
	Undo (see page 2364)	Backs out all changes in the undo buffer.

Public Properties

	Name	Description
	CanUndo (see page 2367)	Indicates whether the edit control contains changes that can be backed out.
	CaretPos (see page 2450)	Indicates the position of the caret in the client area of the memo.
	Modified (see page 2369)	Indicates whether the user edited the text of the edit control.

	SelLength (see page 2371)	Specifies the number of characters (bytes) that are selected.
	SelStart (see page 2374)	Specifies the position of the first selected character in the text.
	SelText (see page 2377)	Specifies the selected portion of the edit control's text.
	Text (see page 2379)	Contains a text string associated with the control.
	TextHint (see page 2380)	A hint or message to be displayed when the Text (see page 2379) property is empty.

Published Events

	Name	Description
	OnChange (see page 2777)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2778)	Occurs when the user clicks the control.
	OnContextPopup (see page 2780)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2780)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2781)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2784)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2787)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2787)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2788)	Occurs when a control receives the input focus.
	OnExit (see page 2791)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2795)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2798)	Occurs when a key is pressed.
	OnKeyUp (see page 2799)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2801)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2802)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2804)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2805)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2805)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2807)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2809)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2810)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2741)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2745)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2745)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 2746)	Specifies which edges of the control are beveled.
	BevelInner (see page 2746)	Specifies the cut of the inner bevel.
	BevelKind (see page 2747)	Specifies the control's bevel style.
	BevelOuter (see page 2747)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2747)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2748)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2748)	Determines the case of the text within the edit control.
	Color (see page 2749)	Specifies the background color of the control.
	Constraints (see page 2749)	Specifies the size constraints for the control.
	Ctl3D (see page 2750)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

	DoubleBuffered (see page 2750)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2752)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2753)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2753)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2755)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2755)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2756)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2757)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2757)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Lines (see page 2758)	Contains the individual lines of text in the memo control.
	MaxLength (see page 2759)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 2759)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2760)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2760)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2761)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2761)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2761)	Determines where a control looks for its font information.
	ParentShowHint (see page 2762)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2762)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2767)	Determines whether the user can change the text of the edit control.
	ScrollBars (see page 2768)	Determines whether the memo control has scroll bars.
	ShowHint (see page 2770)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2771)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2772)	Determines if the user can tab to a control.
	Visible (see page 2772)	Determines whether the component appears onscreen.
	WantReturns (see page 2774)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2774)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2775)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

30.1.22.2 TMemo Properties

The properties of the TMemo class are listed here.

Published Properties

	Name	Description
	Align (see page 2741)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2745)	Determines how the text is aligned within the text edit control.
	Anchors (see page 2745)	Specifies how the control is anchored to its parent.
	BevelEdges (see page 2746)	Specifies which edges of the control are beveled.
	BevelInner (see page 2746)	Specifies the cut of the inner bevel.
	BevelKind (see page 2747)	Specifies the control's bevel style.
	BevelOuter (see page 2747)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2747)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2748)	Determines whether the edit control has a single line border around the client area.
	CharCase (see page 2748)	Determines the case of the text within the edit control.
	Color (see page 2749)	Specifies the background color of the control.

	Constraints (see page 2749)	Specifies the size constraints for the control.
	Ctl3D (see page 2750)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2750)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2752)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2753)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2753)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2755)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2755)	Controls the attributes of text written on or in the control.
	HideSelection (see page 2756)	Determines whether the visual indication of the selected text remains when focus shifts to another control.
	ImeMode (see page 2757)	Determines the behavior of the input method editor (IME).
	ImeName (see page 2757)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
	Lines (see page 2758)	Contains the individual lines of text in the memo control.
	MaxLength (see page 2759)	Specifies the maximum number of characters the user can enter into the edit control.
	OEMConvert (see page 2759)	Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.
	ParentBiDiMode (see page 2760)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2760)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2761)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2761)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2761)	Determines where a control looks for its font information.
	ParentShowHint (see page 2762)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2762)	Identifies the pop-up menu associated with the control.
	ReadOnly (see page 2767)	Determines whether the user can change the text of the edit control.
	ScrollBars (see page 2768)	Determines whether the memo control has scroll bars.
	ShowHint (see page 2770)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2771)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2772)	Determines if the user can tab to a control.
	Visible (see page 2772)	Determines whether the component appears onscreen.
	WantReturns (see page 2774)	Determines whether the user can insert return characters into the text.
	WantTabs (see page 2774)	Determines whether the user can insert tab characters into the text.
	WordWrap (see page 2775)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

30.1.22.2.1 TMemo.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The

value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
```

```
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
```

```
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

30.1.22.2.2 TMemo.Alignment Property

Determines how the text is aligned within the text edit control.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

Use Alignment to change the way the text is formatted in the text edit control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control.
taRightJustify	Align text to the right side of the control.

See Also

SetAlignment (see page 2361)

Text

Displaying and Editing Fields in an Edit Box

30.1.22.2.3 TMemo.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.22.2.4 TMemo.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

30.1.22.2.5 TMemo.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

30.1.22.2.6 TMemo.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

30.1.22.2.7 TMemo.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

30.1.22.2.8 TMemo.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.22.2.9 TMemo.BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Pascal

```
property BorderStyle: TBorderStyle;
```

C++

```
__property TBorderStyle BorderStyle;
```

Description

Use BorderStyle to affect the sharpness with which the client area of the edit control stands out. BorderStyle can have a value of either bsSingle or bsNone. If BorderStyle is bsSingle, the edit control has a single-line border around the client area. If BorderStyle is bsNone, there will be no border.

30.1.22.2.10 TMemo.CharCase Property

Determines the case of the text within the edit control.

Pascal

```
property CharCase: TEditCharCase;
```

C++

```
__property TEditCharCase CharCase;
```

Description

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

See Also

Text

30.1.22.2.11 TMemo.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.22.2.12 TMemo.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.22.2.13 TMemo.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.22.2.14 TMemo.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
```

```
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.22.2.15 TMemo.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.22.2.16 TMemo.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.22.2.17 TMemo.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
```

```
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.22.2.18 TMemo.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.22.2.19 TMemo.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
```

```
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.22.2.20 TMemo.HideSelection Property

Determines whether the visual indication of the selected text remains when focus shifts to another control.

Pascal

```
property HideSelection: Boolean;
```

C++

```
__property Boolean HideSelection;
```

Description

Set HideSelection to false to provide visual feedback of the selected portion of the text even when the edit control does not have focus. Set HideSelection to true to show the selection only when the edit control has focus. HideSelection does not affect the actual value of the selection, only the visual indication. Always setting HideSelection to false can make forms with many edit controls look too busy.

See Also

SelLength (see page 2371)

SelStart (see page 2374)

SelText (see page 2377)

SetSelTextBuf (see page 2363)

30.1.22.2.21 TMemo.ImeMode Property

Determines the behavior of the input method editor (IME).

Pascal

```
property ImeMode;
```

C++

```
__property ImeMode;
```

Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

Note: The value of ImeMode only takes effect when the control receives focus. To change the value of ImeMode when the control already has input focus, call the SetIme method.

See Also

ImeName

ResetIme

ResetImeComposition

SetIme

SetImeCompositionWindow

30.1.22.2.22 TMemo.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

Pascal

```
property ImeName;
```

C++

```
__property ImeName;
```

Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

Note: The value of ImeName only takes effect when the control receives focus. To change the value of ImeName after the control has input focus, call the SetIme method.

See Also

ImeMode

ResetIme

SetIme

DefaultIme

Imes

30.1.22.2.23 TMemo.Lines Property

Contains the individual lines of text in the memo control.

Pascal

```
property Lines: TStrings;
```

C++

```
__property TStrings Lines;
```

Description

Use Lines to manipulate text in an memo control on a line-by-line basis. Lines is a TStrings object, so the TStrings methods may be used for Lines to perform manipulations such as counting the lines of text, adding new lines, deleting lines, or replacing lines with new text.

To work with all the text at once, use the Text property. To manipulate individual lines of text, the Lines property works better.

Note: Although Lines is implemented as a TStrings descendant, it does not implement the support for associating objects with the strings in the list.

C++ Examples:

```
/*
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Memo1->Lines->LoadFromFile("../readme.txt");
  Caption = Memo1->Lines->Strings[4];
}
```

Delphi Examples:

```
{
This example uses a button and a memo control on a form.
When the user clicks the button, the content of the specified
file is loaded into the memo, and the fifth line of the file
is written across the top of the form.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.LoadFromFile('readme.txt');
  Caption:= Memo1.Lines[4];
end;
```

See Also

TStrings

SelText (see page 2377)

SetLines (see page 2449)

Text

WordWrap (see page 2455)

30.1.22.2.24 TMemo.MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the edit control. A value of 0 indicates that there is no application-defined limit on the length.

Use MaxLength to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered into an edit control.

Note: The unit of MaxLength is characters (not bytes) if the application is using themes.

See Also

Text

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

30.1.22.2.25 TMemo.OEMConvert Property

Determines whether characters typed in the edit control are converted from ANSI to OEM and then back to ANSI.

Pascal

```
property OEMConvert: Boolean;
```

C++

```
__property Boolean OEMConvert;
```

Description

To ensure that any characters typed in the edit control can be unambiguously converted to the OEM character set, set OEMConvert to true. This causes all characters to be converted from ANSI to OEM and then back to ANSI as they are typed. Set OEMConvert to false to avoid the overhead of this conversion when it does not matter whether the text can be unambiguously mapped to an OEM string.

OEMConvert is most useful for edit controls used to enter file names when the application does not use Unicode file names.

See Also

Text

30.1.22.2.26 TMemo.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.22.2.27 TMemo.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.22.2.28 TMemo.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.22.2.29 TMemo.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.22.2.30 TMemo.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.22.2.31 TMemo.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.22.2.32 TMemo.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
```

```
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
```

```
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
```

```
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.22.2.33 TMemo.ReadOnly Property

Determines whether the user can change the text of the edit control.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

See Also

CopyToClipboard (see page 2349)

GetSelTextBuf (see page 2357)

30.1.22.2.34 TMemo.ScrollBars Property

Determines whether the memo control has scroll bars.

Pascal

```
property ScrollBars: TScrollStyle;
```

C++

```
__property TScrollStyle ScrollBars;
```

Description

Use ScrollBars to give a multiline edit control horizontal or vertical scroll bars. ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	The control has a scroll bar on both the bottom and right edges.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
```

```
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
```

```
  RichEdit1.WordWrap := False;
end;
```

See Also

SetScrollBars (see page 2449)

WordWrap (see page 2455)

Displaying and Editing Text in a Memo Control

30.1.22.2.35 TMemo.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
```

```
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.22.2.36 TMemo.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the

control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.22.2.37 TMemo.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.22.2.38 TMemo.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
```

```
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.22.2.39 TMemo.WantReturns Property

Determines whether the user can insert return characters into the text.

Pascal

```
property WantReturns: Boolean;
```

C++

```
__property Boolean WantReturns;
```

Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

Note: If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

See Also

Text

WantTabs (see page 2454)

WordWrap (see page 2455)

30.1.22.2.40 TMemo.WantTabs Property

Determines whether the user can insert tab characters into the text.

Pascal

```
property WantTabs: Boolean;
```

C++

```
__property Boolean WantTabs;
```

Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

Note: If WantTabs is true, users can tab into the edit control, but they can't tab out.

See Also

Text

WantReturns (see page 2454)

Displaying and Editing Text in a Memo Control

Displaying and Editing Text in a Rich Edit Memo Control

30.1.22.2.41 TMemo.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

Note: There should be no use for a horizontal scroll bar if WordWrap is true.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
```

```
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
```

```
end;
```

See Also

Lines (see page 2451)

ScrollBars (see page 2452)

Text

WantReturns (see page 2454)

Displaying and Editing Text in a Memo Control

30.1.22.3 TMemo Events

The events of the TMemo class are listed here.

Published Events

	Name	Description
	OnChange (see page 2777)	Occurs when the text for the edit control may have changed.
	OnClick (see page 2778)	Occurs when the user clicks the control.
	OnContextPopup (see page 2780)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2780)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2781)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2784)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2787)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2787)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2788)	Occurs when a control receives the input focus.
	OnExit (see page 2791)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2795)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2798)	Occurs when a key is pressed.
	OnKeyUp (see page 2799)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2801)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2802)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2804)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2805)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2805)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2807)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2809)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2810)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.22.3.1 TMemo.OnChange Event

Occurs when the text for the edit control may have changed.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

See Also

Change (see page 2348)

Text

30.1.22.3.2 TMemo.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
```

```
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.22.3.3 TMemo.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.22.3.4 TMemo.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.22.3.5 TMemo.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
```

```
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.22.3.6 TMemo.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
```

```
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.22.3.7 TMemo.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.22.3.8 TMemo.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
```

```
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.22.3.9 TMemo.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
```

```
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.22.3.10 TMemo.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
```

```
    MaskEdit2->Text = "";
    MaskEdit3->Text = "";

    // use the Courier font in the maskedit boxes
    MaskEdit1->Font->Name = "Courier";
    MaskEdit2->Font->Name = "Courier";
    MaskEdit3->Font->Name = "Courier";

    // initialize the mask of each maskedit box
    // according to IP address coding conventions
    MaskEdit1->EditMask = "!099.099.099.099;1; ";
    MaskEdit2->EditMask = "!099.099.099.099;1; ";
    MaskEdit3->EditMask = "!099.099.099.099;1; ";

    MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
    // extract the net and host address from the IP
    String IP = MaskEdit1->Text;
    int net1 = IP.SubString(0, 3).TrimRight().ToInt();
    int net2 = IP.SubString(5, 3).TrimRight().ToInt();
    int host1 = IP.SubString(9, 3).TrimRight().ToInt();
    int host2 = IP.SubString(13, 3).TrimRight().ToInt();

    // a range test that you cannot validate through edit masks
    if (net1 < 0 || net1 > 255 ||
        net2 < 0 || net2 > 255 ||
        host1 < 0 || host1 > 255 ||
        host2 < 0 || host2 > 255)
      throw(Exception("Not a valid IP address."));

    // extract the net and host mask from the subnet mask
    String mask = MaskEdit2->Text;
    int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
    int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
    int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
    int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

    // a range test that you cannot validate through edit masks
    if (netmask1 < 0 || netmask1 > 255 ||
        netmask2 < 0 || netmask2 > 255 ||
        hostmask1 < 0 || hostmask1 > 255 ||
        hostmask2 < 0 || hostmask2 > 255)
      throw(Exception("Not a valid subnet mask."));

    // compute the subnet address using bit-wise AND
    int sub_net1 = net1 & netmask1;
    int sub_net2 = net2 & netmask2;
    int sub_host1 = host1 & hostmask1;
    int sub_host2 = host2 & hostmask2;

    // display the subnet address
    MaskEdit3->Text =
      String(sub_net1) + "." + String(sub_net2) + "." +
      String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
```

```
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;
```

```
begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((net1 < 0) Or (net1 > 255) Or
      (net2 < 0) Or (net2 > 255) Or
      (host1 < 0) Or (host1 > 255) Or
      (host2 < 0) Or (host2 > 255)) then
    raise EArgumentException.Create('Not a valid IP address.');

  // extract the net and host mask from the subnet mask
  mask := MaskEdit2.Text;
  netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
  netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
  hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
  hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

  // a range test that you cannot validate through edit masks
  if ((netmask1 < 0) Or (netmask1 > 255) Or
      (netmask2 < 0) Or (netmask2 > 255) Or
      (hostmask1 < 0) Or (hostmask1 > 255) Or
      (hostmask2 < 0) Or (hostmask2 > 255)) then
    raise EArgumentException.Create('Not a valid subnet mask.');

  // compute the subnet address using bit-wise AND
  sub_net1 := net1 And netmask1;
  sub_net2 := net2 And netmask2;
  sub_host1 := host1 And hostmask1;
  sub_host2 := host2 And hostmask2;

  // display the subnet address
  MaskEdit3.Text :=
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
```

```
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

30.1.22.3.11 TMemo.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
```

```
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
```

```
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
```

```
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
  MessageDlg('Printing aborted', mtInformation, [mbOK],0);
  end;
end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.22.3.12 TMemo.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII

character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.22.3.13 TMemo.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can

respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
```

```
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

30.1.22.3.14 TMemo.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.22.3.15 TMemo.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
```

```
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.22.3.16 TMemo.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.22.3.17 TMemo.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.22.3.18 TMemo.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
```

```
    Form1->Canvas->Rectangle(StartX, StartY, X, Y);
    StatusBar1->Panels->Items[0]->Text = "";
    StatusBar1->Panels->Items[1]->Text = "";
    StatusBar1->Panels->Items[2]->Text = "";
    StatusBar1->Panels->Items[3]->Text = "";
  }

  // Use this code as the OnMouseMove event handler of the form:
  void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
  {
    if (Shift.Contains(ssLeft)) // make sure button is down
    {
      if (Y > StartY)
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
      }
      else
      {
        StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
        StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
      }
      if (X > StartX)
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
      }
      else
      {
        StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
        StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
      }
    }
  }
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.22.3.19 TMemo.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.22.3.20 TMemo.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.22.3.21 TMemo.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

30.1.23 TPushButtonActionLink Class

TPushButtonActionLink is an action link designed specifically for button control clients.

Class Hierarchy

Pascal

```
TPushButtonActionLink = class(TButtonActionLink);
```

C++

```
class TPushButtonActionLink : public TButtonActionLink;
```

File

StdCtrls

Description

TPushButtonActionLink is designed to specifically link actions to button control clients. TPushButtonActionLink inherits from its ancestors support for linking actions with the properties of components, generic controls, and windowed controls. TPushButtonActionLink overrides methods to set the client's property that corresponds to the action's ImageIndex property.

See Also

TButton (see page 2035)

TButtonActionLink (see page 2106)

TPushButtonActionLinkClass (see page 3013)

30.1.23.1 TPushButtonActionLink Members

The following tables list the members exposed by TPushButtonActionLink.

Protected Fields

	Name	Description
	FClient (see page 2107)	Specifies a client control.

Protected Methods

	Name	Description
	AssignClient (see page 2108)	Assigns a client button that is linked to the associated action.
	IsCheckedLinked (see page 2108)	Indicates whether the checked state for the action and the client are linked.
	IsImageIndexLinked (see page 2812)	Indicates whether the image index for the action and the client are linked.
	SetChecked (see page 2108)	Sets the checked state of a button control if the client is linked to the action for this property.
	SetImageIndex (see page 2812)	Introduces an interface for setting the image index for a client.

30.1.23.2 TPushButtonActionLink Methods

The methods of the TPushButtonActionLink class are listed here.

Protected Methods

	Name	Description
	IsImageIndexLinked (see page 2812)	Indicates whether the image index for the action and the client are linked.
	SetImageIndex (see page 2812)	Introduces an interface for setting the image index for a client.

30.1.23.2.1 TPushButtonActionLink.IsImageIndexLinked Method

Indicates whether the image index for the action and the client are linked.

Pascal

```
function IsImageIndexLinked: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsImageIndexLinked();
```

Description

TActionLink descendants call IsImageIndexLinked to determine whether they need to propagate changes to the ImageIndex property of the action so that they are reflected by the client object. When IsImageIndexLinked returns true, the action controls the ImageIndex property: Changes to the ImageIndex property of the action are reflected by changes to the corresponding property of the client object. When IsImageIndexLinked returns false, the client object does not reflect the ImageIndex property of its action, either because it does not have a corresponding property, or, more commonly, because it manages the value of its corresponding property independently.

As implemented in TActionLink, IsImageIndexLinked merely checks that the Action property specifies a descendant of TCustomAction (that is, it is an action that has an ImageIndex property). Descendant classes override IsImageIndexLinked to determine whether the ImageIndex property is linked with the appropriate client property.

See Also

ImageIndex

ImageIndex

SetImageIndex

Action

30.1.23.2.2 TPushButtonActionLink.SetImageIndex Method

Introduces an interface for setting the image index for a client.

Pascal

```
procedure SetImageIndex(Value: Integer); override;
```

C++

```
virtual __fastcall SetImageIndex(int Value);
```

Description

The associated action calls SetImageIndex when its ImageIndex property changes so that the action link can propagate the new ImageIndex value to the client object.

Value specifies the new value of the ImageIndex property.

As implemented in TActionLink, SetImageIndex does nothing. Descendant classes override SetImageIndex to set the client property that corresponds to the action's ImageIndex property if the IsImageIndexLinked method returns true.

See Also

ImageIndex

ImageIndex

Action

IsImageIndexLinked

30.1.24 TRadioButton Class

TRadioButton is a wrapper for a system radio button.

Class Hierarchy

Pascal

```
TRadioButton = class(TButtonControl);
```

C++

```
class TRadioButton : public TButtonControl;
```

File

StdCtrls

Description

Use TRadioButton to add a radio button to a form. Radio buttons present a set of mutually exclusive options to the user- that is, only one radio button in a set can be selected at a time. When the user selects a radio button, the previously selected radio button becomes unselected. Radio buttons are frequently grouped in a radio group box (TGroupBox (see page 2535)). Add the group box to the form first, then get the radio buttons from the Component palette and put them into the group box. Or you can add a TRadioGroup, a group that already has radio buttons in it, so that you do not have to get the buttons from the Tool Palette.

By default, all radio buttons that are directly contained in the same windowed control container, such as a TRadioGroup or TPanel, are grouped. For example, two radio buttons on a form can be checked at the same time only if they are contained in separate containers, such as two different group boxes.

See Also

TCheckBox (see page 2116)

TPanel

TRadioGroup

Buttons and Similar Controls

Group Boxes and Radio Groups

Radio Buttons

30.1.24.1 TRadioButton Members

The following tables list the members exposed by TRadioButton.

Protected Methods

	Name	Description
	ActionChange (see page 2110)	Updates the button control to reflect changes in its associated action.
	CreateParams (see page 2816)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2816)	Creates the underlying window.
	GetActionLinkClass (see page 2111)	Returns the appropriate action link class that is associated with button controls.
	GetChecked (see page 2817)	Returns whether the button control is checked.
	SetChecked (see page 2817)	Sets the Checked property.
	WndProc (see page 2112)	Handles Windows messages.

Protected Properties

	Name	Description
	ClicksDisabled (see page 2115)	Specifies whether the button control accepts mouse clicks.

Public Methods

	Name	Description
	Create (see page 2815)	Creates an instance of TRadioButton (see page 2813).
	GetControlsAlignment (see page 2817)	Returns the alignment of any text in the control.

Published Events

	Name	Description
	OnClick (see page 2848)	Occurs when the user clicks the control.
	OnContextPopup (see page 2849)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2850)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2850)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2853)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2857)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2857)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2858)	Occurs when a control receives the input focus.
	OnExit (see page 2861)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2865)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2868)	Occurs when a key is pressed.
	OnKeyUp (see page 2869)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2871)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2872)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2874)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2874)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2875)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2877)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2879)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2880)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Action (see page 2818)	Designates the action associated with the control.
	Align (see page 2820)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2823)	Determines the position of the caption relative to the radio button.
	Anchors (see page 2823)	Specifies how the control is anchored to its parent.

	BiDiMode (see page 2824)	Specifies the bi-directional mode for the control.
	Caption (see page 2824)	Specifies a text string that identifies the control to the user.
	Checked (see page 2827)	Determines whether the option represented by the radio button is selected.
	Color (see page 2827)	Specifies the background color of the control.
	Constraints (see page 2828)	Specifies the size constraints for the control.
	Ctl3D (see page 2829)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2829)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2831)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2831)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2832)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2833)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2834)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2835)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2835)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2836)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2836)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2837)	Determines where a control looks for its font information.
	ParentShowHint (see page 2837)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2838)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2843)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2844)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2845)	Determines if the user can tab to a control.
	Visible (see page 2845)	Determines whether the component appears onscreen.
	WordWrap (see page 2847)	Specifies whether the button text wraps to fit the width of the control.

30.1.24.2 TRadioButton.Create Constructor

Creates an instance of TRadioButton (see page 2813).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TRadioButton(TComponent * AOwner);
```

Description

Call Create to instantiate a radio button at runtime. Radio buttons added to forms at design time are created automatically.

AOwner is the component, typically a form, that becomes the value of the Owner property. The Owner is responsible for freeing the radio button object.

See Also

Owner

30.1.24.3 TRadioButton Methods

The methods of the TRadioButton class are listed here.

Protected Methods

	Name	Description
	CreateParams (see page 2816)	Initializes a window-creation parameter data structure.
	CreateWnd (see page 2816)	Creates the underlying window.
	GetChecked (see page 2817)	Returns whether the button control is checked.
	SetChecked (see page 2817)	Sets the Checked property.

Public Methods

	Name	Description
	GetControlsAlignment (see page 2817)	Returns the alignment of any text in the control.

30.1.24.3.1 TRadioButton.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

30.1.24.3.2 TRadioButton.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

30.1.24.3.3 TRadioButton.GetChecked Method

Returns whether the button control is checked.

Pascal

```
function GetChecked: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetChecked();
```

Description

GetChecked is the read implementation of the Checked property.

See Also

Checked (see page 2113)

30.1.24.3.4 TRadioButton.GetControlsAlignment Method

Returns the alignment of any text in the control.

Pascal

```
function GetControlsAlignment: TAlignment; override;
```

C++

```
virtual __fastcall TAlignment GetControlsAlignment();
```

Description

Use GetControlsAlignment to determine the orientation of text in controls that do not have an Alignment (see page 2823) property.

GetControlsAlignment overrides the inherited method, returning taRightJustify if UseRightToLeftAlignment returns false. If UseRightToLeftAlignment returns true, GetControlsAlignment returns the opposite of the Alignment (see page 2823) property.

30.1.24.3.5 TRadioButton.SetChecked Method

Sets the Checked property.

Pascal

```
procedure SetChecked(Value: Boolean); override;
```

C++

```
virtual __fastcall SetChecked(Boolean Value);
```

Description

SetChecked is the write implementer for the Checked property.

See Also

Checked (see page 2113)

30.1.24.4 TRadioButton Properties

The properties of the TRadioButton class are listed here.

Published Properties

	Name	Description
	Action (see page 2818)	Designates the action associated with the control.
	Align (see page 2820)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2823)	Determines the position of the caption relative to the radio button.
	Anchors (see page 2823)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2824)	Specifies the bi-directional mode for the control.
	Caption (see page 2824)	Specifies a text string that identifies the control to the user.
	Checked (see page 2827)	Determines whether the option represented by the radio button is selected.
	Color (see page 2827)	Specifies the background color of the control.
	Constraints (see page 2828)	Specifies the size constraints for the control.
	Ctl3D (see page 2829)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2829)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2831)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2831)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2832)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2833)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 2834)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2835)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2835)	Determines where a control looks for its color information.
	ParentCtl3D (see page 2836)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2836)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2837)	Determines where a control looks for its font information.
	ParentShowHint (see page 2837)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2838)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 2843)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2844)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2845)	Determines if the user can tab to a control.
	Visible (see page 2845)	Determines whether the component appears onscreen.
	WordWrap (see page 2847)	Specifies whether the button text wraps to fit the width of the control.

30.1.24.4.1 TRadioButton.Action Property

Designates the action associated with the control.

Pascal

```
property Action;
```

C++

```
__property Action;
```

Description

Action is the action object that is associated with the control. Actions allow an application to centralize the response to user commands. When a control is associated with an action, the action determines the appropriate properties and events of the control (such as whether the control is enabled or how it responds to an OnClick event).

To create actions at design time, place an action list component on a form or data module. Double click the action list to bring up the action list editor. Add actions in the editor using its context menu. Once the actions have been added using the action list editor, they appear in the drop-down list for the Action property in the Object Inspector.

C++ Examples:

```
/*
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
*/
void __fastcall TForm1::ChangeColor(TObject *Sender)
{
  // randomly change the color of the form
  Color = Random(0xFFFF);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // allocate memory for the Action property
  Action = new TBasicAction(this);

  // set the OnExecute event handler
  Action->OnExecute = ChangeColor;

  // initialize the random number generator
  Randomize();
}
```

Delphi Examples:

```
{
This example shows that OnClick is the default event of a form.
When the form is created, the OnExecute event of its Action property
is initialized to some user-defined procedure, which randomly changes
the color of the form.
}
procedure TForm1.ChangeColor(Sender: TObject);
begin
  { randomly change the color of the form }
  Color := Random($FFFF);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { allocate memory for the Action property }
  Action := TBasicAction.Create(Self);

  { set the OnExecute event handler }
  Action.OnExecute := ChangeColor;

  { initialize the random number generator }
  Randomize;
end;
```

30.1.24.4.2 TRadioButton.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
```

```
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

30.1.24.4.3 TRadioButton.Alignment Property

Determines the position of the caption relative to the radio button.

Pascal

```
property Alignment: TLeftRight;
```

C++

```
__property TLeftRight Alignment;
```

Description

Set Alignment to determine whether the caption of the radio button should appear to the left of the radio button (taLeftJustify) or to the right of the radio button (taRightJustify).

Note: Do not confuse the Alignment property with the alignment of the caption in the text area of the radio button. The caption always appears left aligned. Alignment only controls on which side of the button the text area appears.

See Also

Caption

30.1.24.4.4 TRadioButton.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.24.4.5 TRadioButton.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.24.4.6 TRadioButton.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
```

```
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
```

```
    for I := 0 to High(ButtonCaptions) do
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

30.1.24.4.7 TRadioButton.Checked Property

Determines whether the option represented by the radio button is selected.

Pascal

```
property Checked: Boolean;
```

C++

```
__property Boolean Checked;
```

Description

Read (see page 3634) Checked to determine whether the radio button is selected. Set Checked to true to select the radio button and deselect all other radio buttons in the same container. Set Checked to false to deselect the radio button, leaving no radio button in the group selected.

See Also

TRadioGroup

30.1.24.4.8 TRadioButton.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.24.4.9 TRadioButton.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.24.4.10 TRadioButton.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.24.4.11 TRadioButton.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
```

```
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.24.4.12 TRadioButton.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.24.4.13 TRadioButton.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.24.4.14 TRadioButton.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
```

```
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.24.4.15 TRadioButton.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.24.4.16 TRadioButton.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
```

```
  TMemo on a form.  During form creation, a list of font names
  is loaded into the combo box. When the combo box is clicked,
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  begin
    for i := 0 to Screen.Fonts.Count - 1 do
     ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
  //  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
    RichEdit1.Lines.Text := 'How is it to be you?';
  end;

  procedure TForm1.ComboBox1Click(Sender: TObject);
  begin
    RichEdit1.Font.Name := ComboBox1.Text;
  end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.24.4.17 TRadioButton.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.24.4.18 TRadioButton.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.24.4.19 TRadioButton.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.24.4.20 TRadioButton.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.24.4.21 TRadioButton.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.24.4.22 TRadioButton.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.24.4.23 TRadioButton.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.24.4.24 TRadioButton.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.24.4.25 TRadioButton.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.24.4.26 TRadioButton.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.24.4.27 TRadioButton.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.24.4.28 TRadioButton.WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Pascal

```
property WordWrap: Boolean;
```

C++

```
__property Boolean WordWrap;
```

Description

Set WordWrap to true to allow the display of multiple lines of text. When WordWrap is true, text that is too wide for the control wraps at the right margin.

Set WordWrap to false to limit the text to a single line. When WordWrap is false, text that is too wide for the control appears truncated.

See Also

ClientWidth

30.1.24.5 TRadioButton Events

The events of the TRadioButton class are listed here.

Published Events

	Name	Description
	OnClick (see page 2848)	Occurs when the user clicks the control.
	OnContextPopup (see page 2849)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2850)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2850)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2853)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2857)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2857)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2858)	Occurs when a control receives the input focus.
	OnExit (see page 2861)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2865)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2868)	Occurs when a key is pressed.
	OnKeyUp (see page 2869)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 2871)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2872)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2874)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2874)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2875)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2877)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2879)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2880)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.24.5.1 TRadioButton.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
```

```
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.24.5.2 TRadioButton.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.24.5.3 TRadioButton.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.24.5.4 TRadioButton.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
```

```
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
```

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.24.5.5 TRadioButton.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
```

```
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
```

```
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.24.5.6 TRadioButton.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.24.5.7 TRadioButton.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
```

```
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.24.5.8 TRadioButton.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
```

```
  applied to the active control. Clicking the button sets the
  ActiveControl to the button.  That is why we need to save
  the ActiveControl in a shared OnEnter procedure.  In the rich
  edit, only the text selected converts.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  begin
     FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
     FontDialog1.Execute;
  end;

  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    myActiveControl := ActiveControl;
  end;

  procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
  begin
    if myActiveControl is TEdit then
      with myActiveControl as TEdit do
         Font.Assign(TFontDialog(Sender).Font)
    else if myActiveControl is TRichEdit then
      with myActiveControl as TRichEdit do
        SelAttributes.Assign(TFontDialog(Sender).Font)
    else
      Beep;
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.24.5.9 TRadioButton.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
```

```
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
  sub_net1, sub_net2, sub_host1, sub_host2: Integer;

  IP, mask: String;

begin
  // extract the net and host address from the IP
  IP := MaskEdit1.Text;
  net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
  net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
  host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
  host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

  // a range test that you cannot validate through edit masks
```

```
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

30.1.24.5.10 TRadioButton.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.24.5.11 TRadioButton.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.24.5.12 TRadioButton.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
```

```
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

30.1.24.5.13 TRadioButton.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control.

However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.24.5.14 TRadioButton.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
```

```
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
```

```
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.24.5.15 TRadioButton.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.24.5.16 TRadioButton.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.24.5.17 TRadioButton.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}
```

```
// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
```

```
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.24.5.18 TRadioButton.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
```

```
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;
```

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.24.5.19 TRadioButton.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.24.5.20 TRadioButton.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

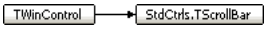
OnDragOver

OnDragDrop

30.1.25 TScrollBar Class

TScrollBar is a Windows scroll bar, which is used to scroll the contents of a window, form, or control.

Class Hierarchy



Pascal

```
TScrollBar = class(TWinControl);
```

C++

```
class TScrollBar : public TWinControl;
```

File

StdCtrls

Description

Use TScrollBar to add a free-standing scroll bar to a form. Many controls have properties that add scroll bars which are an integral part of the control (TControlScrollBar objects). TScrollBar allows controls that do not have integrated scroll bars or groupings of controls to be scrolled when the user manipulates the TScrollBar object.

When the bi-directional mode of a scroll bar is changed, its coordinate system is reversed. Position (see page 2912) zero in left-to-right mode is the left side, and position zero in right-to-left mode is the right side.

See Also

TScrollingWinControl

TTrackBar

TUpDown

Scroll Bars

Specialized Input Controls

30.1.25.1 TScrollBar Members

The following tables list the members exposed by TScrollBar.

Protected Methods

	Name	Description
	Change (see page 2883)	Called when the Position (see page 2912) property changes.
V	CreateParams (see page 2884)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 2884)	Creates the underlying window.
	Scroll (see page 2885)	Triggers an `OnScroll` (see page 2942) event with the given `ScrollCode` parameter and updates the `ScrollPos` variable that is passed.

Public Methods

	Name	Description
V	Create (see page 2883)	Creates and initializes an instance of TScrollBar (see page 2881).
	SetParams (see page 2885)	Sets the Position (see page 2912), Max (see page 2900), and Min (see page 2901) property values all at once.

Published Events

	Name	Description
	OnChange (see page 2920)	Occurs immediately after the value of Position (see page 2912) changes.
	OnContextPopup (see page 2920)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2921)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2924)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2927)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2928)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2929)	Occurs when a control receives the input focus.
	OnExit (see page 2931)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2935)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2939)	Occurs when a key is pressed.
	OnKeyUp (see page 2940)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 2942)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2942)	Occurs when the user moves the mouse outside of a control.
	OnScroll (see page 2942)	Occurs when the user scrolls the scroll bar with the mouse or keyboard.
	OnStartDock (see page 2944)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2944)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2886)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2890)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2890)	Specifies the bi-directional mode for the control.
	Constraints (see page 2891)	Specifies the size constraints for the control.
	Ctl3D (see page 2892)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2892)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2894)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2894)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2895)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2896)	Controls whether the control responds to mouse, keyboard, and timer events.
	Kind (see page 2897)	Specifies whether the scroll bar is horizontal or vertical.
	LargeChange (see page 2898)	Determines how much Position (see page 2912) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.
	Max (see page 2900)	Specifies the maximum position represented by the scroll bar.
	Min (see page 2901)	Specifies the minimum position represented by the scroll bar.
	PageSize (see page 2903)	Specifies the size of the thumb tab.
	ParentBiDiMode (see page 2905)	Specifies whether the control uses its parent's BiDiMode.
	ParentCtl3D (see page 2905)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2906)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 2906)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2907)	Identifies the pop-up menu associated with the control.
	Position (see page 2912)	Indicates the current position of the scroll bar.
	ShowHint (see page 2914)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SmallChange (see page 2915)	Determines how much Position (see page 2912) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.
	TabOrder (see page 2917)	Indicates the position of the control in its parent's tab order.

	TabStop (see page 2918)	Determines if the user can tab to a control.
	Visible (see page 2918)	Determines whether the component appears onscreen.

30.1.25.2 TScrollBar.Create Constructor

Creates and initializes an instance of TScrollBar (see page 2881).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScrollBar(TComponent * AOwner);
```

Description

Call Create to create an instance of TScrollBar (see page 2881) at runtime. Scroll (see page 2885) bars placed on a form at design time are created automatically.

AOwner is the component that is responsible for freeing the scroll bar. This becomes the value of the Owner property.

See Also

Owner

30.1.25.3 TScrollBar Methods

The methods of the TScrollBar class are listed here.

Protected Methods

	Name	Description
	Change (see page 2883)	Called when the Position (see page 2912) property changes.
V	CreateParams (see page 2884)	Initializes a window-creation parameter data structure.
V	CreateWnd (see page 2884)	Creates the underlying window.
	Scroll (see page 2885)	Triggers an `OnScroll` (see page 2942) event with the given `ScrollCode` parameter and updates the `ScrollPos` variable that is passed.

Public Methods

	Name	Description
	SetParams (see page 2885)	Sets the Position (see page 2912), Max (see page 2900), and Min (see page 2901) property values all at once.

30.1.25.3.1 TScrollBar.Change Method

Called when the Position (see page 2912) property changes.

Pascal

```
procedure Change; dynamic;
```

C++

```
__fastcall Change();
```

Description

Override Change in a derived class to perform actions when the Position (see page 2912) of the scroll bar changes.

By default, Change calls the OnChange (see page 2920) event handler.

See Also

OnChange (see page 2920)

Position (see page 2912)

30.1.25.3.2 TScrollBar.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

30.1.25.3.3 TScrollBar.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

30.1.25.3.4 TScrollBar.Scroll Method

Triggers an `OnScroll` (see page 2942) event with the given `ScrollCode` parameter and updates the `ScrollPos` variable that is passed.

Pascal

```
procedure Scroll(ScrollCode: TScrollCode; var ScrollPos: Integer); dynamic;
```

C++

```
__fastcall Scroll(TScrollCode ScrollCode, int ScrollPos);
```

Description

Scroll is a protected routine in the TScrollBar (see page 2881) class that is used to trigger an `OnScroll` (see page 2942) event, with the given `ScrollCode` parameter.

Scroll also updates the value of the `ScrollPos` variable that is passed.

The following table gives the possible values for the `ScrollCode` parameter.

Value	Meaning
scLineUp	User clicked the top or left scroll arrow or pressed the Up or Left arrow key.
scLineDown	User clicked the bottom or right scroll arrow or pressed the Down or Right arrow key.
scPageUp	User clicked the area to the left of the thumb tab or pressed the PgUp key.
scPageDown	User clicked the area to the right of the thumb tab or pressed the PgDn key.
scPosition	User positioned the thumb tab and released it.
scTrack	User is moving the thumb tab.
scTop	User moved the thumb tab to the top or leftmost on the scroll bar.
scBottom	User moved the thumb tab to the bottom or rightmost on the scroll bar.
scEndScroll	User finished moving the thumb tab on the scroll bar.

See Also

OnScroll (see page 2942)

TScrollCode (see page 3004)

30.1.25.3.5 TScrollBar.SetParams Method

Sets the Position (see page 2912), Max (see page 2900), and Min (see page 2901) property values all at once.

Pascal

```
procedure SetParams(APosition: Integer; AMin: Integer; AMax: Integer);
```

C++

```
__fastcall SetParams(int APosition, int AMin, int AMax);
```

Description

Use the SetParams method to set the Position (see page 2912), Min (see page 2901), and Max (see page 2900) property values all at once, preventing repaints that would occur if each property value were set separately.

See Also

Max (see page 2900)

Min (see page 2901)

Position (see page 2912)

30.1.25.4 TScrollBar Properties

The properties of the TScrollBar class are listed here.

Published Properties

	Name	Description
	Align (see page 2886)	Determines how the control aligns within its container (parent control).
	Anchors (see page 2890)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 2890)	Specifies the bi-directional mode for the control.
	Constraints (see page 2891)	Specifies the size constraints for the control.
	Ctl3D (see page 2892)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DoubleBuffered (see page 2892)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2894)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2894)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2895)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2896)	Controls whether the control responds to mouse, keyboard, and timer events.
	Kind (see page 2897)	Specifies whether the scroll bar is horizontal or vertical.
	LargeChange (see page 2898)	Determines how much Position (see page 2912) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.
	Max (see page 2900)	Specifies the maximum position represented by the scroll bar.
	Min (see page 2901)	Specifies the minimum position represented by the scroll bar.
	PageSize (see page 2903)	Specifies the size of the thumb tab.
	ParentBiDiMode (see page 2905)	Specifies whether the control uses its parent's BiDiMode.
	ParentCtl3D (see page 2905)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 2906)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 2906)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2907)	Identifies the pop-up menu associated with the control.
	Position (see page 2912)	Indicates the current position of the scroll bar.
	ShowHint (see page 2914)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	SmallChange (see page 2915)	Determines how much Position (see page 2912) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.
	TabOrder (see page 2917)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2918)	Determines if the user can tab to a control.
	Visible (see page 2918)	Determines whether the component appears onscreen.

30.1.25.4.1 TScrollBar.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
```

```
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
```

```
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
```

```
        Brush.Color := colorPalette[i];
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

30.1.25.4.2 TScrollBar.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.25.4.3 TScrollBar.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.25.4.4 TScrollBar.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.25.4.5 TScrollBar.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

30.1.25.4.6 TScrollBar.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
```

```
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.25.4.7 TScrollBar.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.25.4.8 TScrollBar.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.25.4.9 TScrollBar.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.25.4.10 TScrollBar.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.25.4.11 TScrollBar.Kind Property

Specifies whether the scroll bar is horizontal or vertical.

Pascal

```
property Kind: TScrollBarKind;
```

C++

```
__property TScrollBarKind Kind;
```

Description

Set Kind to indicate the orientation of the scroll bar. These are the possible values:

Value	Meaning
sbHorizontal	Scroll (see page 2885) bar is horizontal
sbVertical	Scroll (see page 2885) bar is vertical

C++ Examples:

```
/*
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("Vertical");
  RadioGroup1->Items->Add("Horizontal");
  RadioGroup1->ItemIndex = 2;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Vertical")
    ScrollBar1->Kind = sbVertical;
  else if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "Horizontal")
    ScrollBar1->Kind = sbHorizontal;
}
```

Delphi Examples:

```
{
This example uses a radio group box and a scroll bar on a
form. When the user selects one of the radio buttons, the
scroll bar changes orientation accordingly.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('Vertical');
  RadioGroup1.Items.Add('Horizontal');
  RadioGroup1.ItemIndex := 2;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Vertical' then
```

```
    ScrollBar1.Kind := sbVertical;
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'Horizontal' then
    ScrollBar1.Kind := sbHorizontal;
end;
```

30.1.25.4.12 TScrollBar.LargeChange Property

Determines how much Position (see page 2912) changes when the user clicks the scroll bar on either side of the thumb tab or presses PgUp or PgDn.

Pascal

```
property LargeChange: TScrollBarInc;
```

C++

```
__property TScrollBarInc LargeChange;
```

Description

Set LargeChange to specify how much to change the value of Position (see page 2912) when the user clicks on the side of the thumb tab (that is, on the thumb tab track, but not on the thumb tab itself). The Max (see page 2900) and Min (see page 2901) properties define the total range over which Position (see page 2912) can vary.

For example, if Max (see page 2900) is 30000 and Min (see page 2901) is 0, the scroll box can assume 30,000 positions. If the LargeChange property setting is 10000 and the Position (see page 2912) property is 0, the user can click the scroll bar track three times after the thumb tab before it moves all the way to the end of the scroll bar (30000 / 10000 = 3).

LargeChange should correspond to the portion of this range that the user sees in one page or screen, expressed in the logical units used by the Max (see page 2900), Min (see page 2901), and Position (see page 2912) properties.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
```

```
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

Max (see page 2900)

Min (see page 2901)

Position (see page 2912)

SmallChange (see page 2915)

PageSize (see page 2903)

30.1.25.4.13 TScrollBar.Max Property

Specifies the maximum position represented by the scroll bar.

Pascal

```
property Max: Integer;
```

C++

```
__property int Max;
```

Description

Set Max to the maximum value the Position (see page 2912) property can take. The Max and Min (see page 2901) properties define the total range over which Position (see page 2912) can vary.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 2898)

Min (see page 2901)

Position (see page 2912)

SetParams (see page 2885)

SmallChange (see page 2915)

30.1.25.4.14 TScrollBar.Min Property

Specifies the minimum position represented by the scroll bar.

Pascal

```
property Min: Integer;
```

C++

```
__property int Min;
```

Description

Set Min to the minimum value the Position (see page 2912) property can take. The Min and Max (see page 2900) properties define the total range over which Position (see page 2912) can vary.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
```

```
  to automatically change the active page by one tab or one
  row respectively.  The OnScroll event is used to make the
  actual change in the page control, while the OnChange event
  of the page control ensures that the scroll bar is kept in
  sync with changes made directly to the page control.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  const
    colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
  clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
  begin
    { First, initialize the page control }
    PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
    PageControl1.ActivePage := PageControl1.Pages[0];
    for i := 0 to PageControl1.PageCount - 1 do
      PageControl1.Pages[i].Brush.Color:= colorPalette[i];
    { set up the scrollbar to reflect the structure of the page control}
    with ScrollBar1 do
    begin
      Max := PageControl1.PageCount - 1;
      Min := 0;
      SmallChange := 1;
      { set LargeChange to a single row of tabs }
      LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
    end;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    with Sender as TPageControl do
      ScrollBar1.Position := ActivePage.PageIndex;
  end;

  procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  begin
    PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
  end;
```

See Also

LargeChange (see page 2898)

Max (see page 2900)

Position (see page 2912)

SetParams (see page 2885)

SmallChange (see page 2915)

30.1.25.4.15 TScrollBar.PageSize Property

Specifies the size of the thumb tab.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

PageSize is the size of the thumb tab, measured in the same units as Position (see page 2912), Min (see page 2901), and Max (see page 2900) (not pixels).

C++ Examples:

```
/*
The following code sets the thumb tab of a scroll bar so
that it represents the proportion of the scrolling range
that is visible:
*/
void __fastcall SetThumbTab()
{
  // Get the minimum (default) size of the thumb tab
  int MinHeight = GetSystemMetrics(SM_CYVTHUMB);
  // Get the size of the scrollbar track
  int TrackHeight = Form1->ScrollBar1->ClientHeight - 2*GetSystemMetrics(SM_CYVSCROLL);
  // set PageSize to represent the visible portion of the scrolling range:
  Form1->ScrollBar1->PageSize = TrackHeight/(Form1->ScrollBar1->Max - Form1->ScrollBar1->Min +
1);
  // Don't let the thumb tab get too small
  if (Form1->ScrollBar1->PageSize < MinHeight)
  Form1->ScrollBar1->PageSize = MinHeight;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const AnsiString Path = "../overview.rtf";
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile(Path);
  SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RichEdit1Change(TObject *Sender)
{
 SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
 SetThumbTab();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RichEdit1EndDrag(TObject *Sender, TObject *Target,
      int X, int Y)
{
 SetThumbTab();
}
```

Delphi Examples:

```
{
The following code sets the thumb tab of a scroll bar so
that it represents the proportion of the scrolling range
that is visible:
}
procedure SetThumbTab(wincontrol: TWinControl);
var
  TrackHeight: Integer; { the size of the scroll bar track }
  MinHeight: Integer; { the default size of the thumb tab }
begin
  MinHeight := GetSystemMetrics(SM_CYVTHUMB); { save default size }
  with Form1.ScrollBar1 do
  begin
    TrackHeight := wincontrol.ClientHeight - 2 * GetSystemMetrics(SM_CYVSCROLL);
```

```
      PageSize := TrackHeight div (Max - Min + 1);
      if PageSize < MinHeight then PageSize := MinHeight;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const
    Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    SetThumbTab(RichEdit1);
  end;
  procedure TForm1.RichEdit1Change(Sender: TObject);
  begin
    SetThumbTab(RichEdit1);
  end;
```

See Also

Min (see page 2901)

Max (see page 2900)

ClientHeight

LargeChange (see page 2898)

30.1.25.4.16 TScrollBar.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.25.4.17 TScrollBar.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

30.1.25.4.18 TScrollBar.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.25.4.19 TScrollBar.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.25.4.20 TScrollBar.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
```

```
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
```

```
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
```

```
    Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
```

```
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.25.4.21 TScrollBar.Position Property

Indicates the current position of the scroll bar.

Pascal

```
property Position: Integer;
```

C++

```
__property int Position;
```

Description

Read (see page 3634) Position to determine the current position of the thumb tab. This value can be used to determine how to scroll any components controlled by the scroll bar. When the user scrolls the scroll bar, the value of Position changes. Set Position to programmatically move the thumb tab of the scroll bar.

The number of possible positions on the scroll bar is determined by the difference between the Max (see page 2900) property and the Min (see page 2901) property. If Position has the same value as Min (see page 2901), the thumb tab appears at the far left of a horizontal scroll bar or the top of a vertical scroll bar. If Position has the same value as Max (see page 2900), the thumb tab appears at the far right of a horizontal scroll bar or the bottom of a vertical scroll bar.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
```

```
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

Max (see page 2900)

Min (see page 2901)

SetParams (see page 2885)

30.1.25.4.22 TScrollBar.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
```

```
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.25.4.23 TScrollBar.SmallChange Property

Determines how much Position (see page 2912) changes when the user clicks the arrow buttons on the scroll bar or presses the arrow keys on the keyboard.

Pascal

```
property SmallChange: TScrollBarInc;
```

C++

```
__property TScrollBarInc SmallChange;
```

Description

Set SmallChange to specify how much to change the value of Position (see page 2912) when the user clicks on the arrow buttons. The Max (see page 2900) and Min (see page 2901) properties define the total range over which Position (see page 2912) can vary.

For example, if Max (see page 2900) is 100 and Min (see page 2901) is 0, the scroll box can assume 100 positions. If the SmallChange property setting is 5 and the Position (see page 2912) property is 0, the user can click the down or right arrow button 20 times before the thumb tab is moved all the way to the end of the scroll bar.

SmallChange should correspond to the portion of this range that the user sees as one line or one item, expressed in the logical units used by the Max (see page 2900), Min (see page 2901), and Position (see page 2912) properties.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
```

```
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
```

```
    begin
      Max := PageControl1.PageCount - 1;
      Min := 0;
      SmallChange := 1;
      { set LargeChange to a single row of tabs }
      LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
    end;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    with Sender as TPageControl do
      ScrollBar1.Position := ActivePage.PageIndex;
  end;

  procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  begin
    PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
  end;
```

See Also

LargeChange (see page 2898)

Max (see page 2900)

Min (see page 2901)

Position (see page 2912)

30.1.25.4.24 TScrollBar.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.25.4.25 TScrollBar.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.25.4.26 TScrollBar.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.25.5 TScrollBar Events

The events of the TScrollBar class are listed here.

Published Events

	Name	Description
	OnChange (see page 2920)	Occurs immediately after the value of Position (see page 2912) changes.
	OnContextPopup (see page 2920)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDragDrop (see page 2921)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2924)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2927)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2928)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 2929)	Occurs when a control receives the input focus.
	OnExit (see page 2931)	Occurs when the input focus shifts away from one control to another.
	OnKeyDown (see page 2935)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 2939)	Occurs when a key is pressed.
	OnKeyUp (see page 2940)	Occurs when the user releases a key that has been pressed.
	OnMouseEnter (see page 2942)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2942)	Occurs when the user moves the mouse outside of a control.
	OnScroll (see page 2942)	Occurs when the user scrolls the scroll bar with the mouse or keyboard.
	OnStartDock (see page 2944)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2944)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.25.5.1 TScrollBar.OnChange Event

Occurs immediately after the value of Position (see page 2912) changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3707) an OnChange event handler to control another object or group of objects using TScrollBar (see page 2881). Adjust the objects controlled by the scroll bar to reflect the new value of Position (see page 2912).

OnChange occurs whether the value of Position (see page 2912) is changed by the user or programmatically. If the value of Position (see page 2912) is changed by the user, OnChange occurs immediately after the OnScroll (see page 2942) event.

See Also

OnScroll (see page 2942)

Position (see page 2912)

30.1.25.5.2 TScrollBar.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.25.5.3 TScrollBar.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)
```

```
{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
```

```
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
```

```
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.25.5.4 TScrollBar.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
```

```
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
```

```
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.25.5.5 TScrollBar.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.25.5.6 TScrollBar.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.25.5.7 TScrollBar.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
```

```
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
```

```
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

30.1.25.5.8 TScrollBar.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK

button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
```

```
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    { initialize the maskedit boxes to contain the empty string }
    MaskEdit1.Text := '';
    MaskEdit2.Text := '';
    MaskEdit3.Text := '';

    { use the Courier font in the maskedit boxes }
    MaskEdit1.Font.Name := 'Courier';
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
```

```
    IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
    IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

30.1.25.5.9 TScrollBar.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys

that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
```

```
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
```

```
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
  Memo1 : TMemo;
  r: TRect;
begin
  Memo1 := TMemo.Create(Form1);
  Memo1.Parent := Form1;
  Memo1.Visible := True;
  Memo1.Width := 700;
  if (OpenDialog1.Execute) then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

    Printer.BeginDoc;
    X := 200;
    Y := 200;
    for I := 0 to 140 do
      if (not Printer.Aborted) then
      begin
        Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
        Y := Y + 80;
        if (Y > (Printer.PageHeight - 300)) then
        begin
          Y := 200;
          Printer.NewPage;
          Sleep(1000);  // to give you time to abort!
        end;
      end;
    if (not Printer.Aborted) then Printer.EndDoc;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   KeyPreview := True;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if (Key=VK_ESCAPE) and Printer.Printing then
  begin
  Printer.Abort;
```

```
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

30.1.25.5.10 TScrollBar.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
```

```
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

30.1.25.5.11 TScrollBar.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
```

```
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

30.1.25.5.12 TScrollBar.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.25.5.13 TScrollBar.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.25.5.14 TScrollBar.OnScroll Event

Occurs when the user scrolls the scroll bar with the mouse or keyboard.

Pascal

```
property OnScroll: TScrollEvent;
```

C++

```
__property TScrollEvent OnScroll;
```

Description

Write (see page 3707) an OnScroll to alter the way the scrollbar adjusts its position when the user scrolls the scroll bar.

C++ Examples:

```
/*
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  const TColor colorPalette[12] = {clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};
  // First, initialize the page control
  PageControl1->TabWidth = PageControl1->ClientWidth/4 - 1;
  PageControl1->ActivePage = PageControl1->Pages[0];
  for (int i = 0; i < PageControl1->PageCount; i++)
    PageControl1->Pages[i]->Brush->Color = colorPalette[i];
  /* set up the scrollbar to reflect
     the structure of the page control */
  ScrollBar1->Max = PageControl1->PageCount - 1;
  ScrollBar1->Min = 0;
  ScrollBar1->SmallChange = 1;
  // set LargeChange to a single row of tabs
  ScrollBar1->LargeChange = PageControl1->ClientWidth / PageControl1->TabWidth;
}

void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  ScrollBar1->Position = (dynamic_cast<TPageControl *>(Sender))->ActivePage->PageIndex;
}

void __fastcall TForm1::ScrollBar1Scroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{
  PageControl1->ActivePage = PageControl1->Pages[ScrollPos];
}
```

Delphi Examples:

```
{
This example shows how to use a scroll bar to navigate the
pages of a page control.  To run this example, place a
scroll bar and a page control on a form.  Right click the
page control and add 12 pages to it.  Add the following code
to the OnCreate event of the form, the OnChange event of the
page control, and the OnScroll event of the scroll bar.
Note that the LargeChange and SmallChange properties are set
to automatically change the active page by one tab or one
row respectively.  The OnScroll event is used to make the
actual change in the page control, while the OnChange event
of the page control ensures that the scroll bar is kept in
sync with changes made directly to the page control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
const
  colorpalette: Array[0..11] of TColor = (clRed, clGreen, clYellow, clBlue, clWhite,
clFuchsia, clTeal, clNavy, clMaroon, clLime, clOlive, clPurple);
begin
  { First, initialize the page control }
  PageControl1.TabWidth := PageControl1.ClientWidth div 4 - 1;
  PageControl1.ActivePage := PageControl1.Pages[0];
  for i := 0 to PageControl1.PageCount - 1 do
    PageControl1.Pages[i].Brush.Color:= colorPalette[i];
  { set up the scrollbar to reflect the structure of the page control}
  with ScrollBar1 do
  begin
    Max := PageControl1.PageCount - 1;
    Min := 0;
    SmallChange := 1;
    { set LargeChange to a single row of tabs }
    LargeChange := PageControl1.ClientWidth div PageControl1.TabWidth;
  end;
end;
```

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  with Sender as TPageControl do
    ScrollBar1.Position := ActivePage.PageIndex;
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  PageControl1.ActivePage := PageControl1.Pages[ScrollPos];
end;
```

See Also

LargeChange (see page 2898)

OnChange (see page 2920)

Position (see page 2912)

SmallChange (see page 2915)

30.1.25.5.15 TScrollBar.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.25.5.16 TScrollBar.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse

button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

30.1.26 TStaticText Class

TStaticText is a windowed control that displays text on a form.

Class Hierarchy

Pascal

```
TStaticText = class(TCustomStaticText);
```

C++

```
class TStaticText : public TCustomStaticText;
```

File

StdCtrls

Description

The TStaticText component functions like TLabel (see page 2607), except that it descends from TWinControl and therefore has a window handle. Use TStaticText instead of TLabel (see page 2607) when the component's accelerator key must belong

to a windowed control—for example, on an ActiveX property page.

See Also

TCustomStaticText (see page 2457)

TLabel (see page 2607)

TWinControl

Labels

Display Controls

30.1.26.1 TStaticText Members

The following tables list the members exposed by TStaticText.

Protected Methods

	Name	Description
	CreateParams (see page 2459)	Initializes a window-creation parameter data structure.
	Loaded (see page 2459)	Initializes the control after it is loaded from a stream.
	Notification (see page 2460)	Responds to notifications that components are being created or destroyed.
	SetAutoSize (see page 2460)	Sets the value of the AutoSize property.

Public Methods

	Name	Description
	Create (see page 2458)	Creates and initializes a TCustomStaticText (see page 2457) instance.

Published Events

	Name	Description
	OnClick (see page 2978)	Occurs when the user clicks the control.
	OnContextPopup (see page 2980)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2980)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2981)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2984)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2987)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2987)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2988)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2989)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2991)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 2992)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2992)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2994)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2996)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2997)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 2948)	Determines how the control aligns within its container (parent control).

	Alignment (see page 2951)	Determines how text is aligned within the component.
	Anchors (see page 2952)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2952)	Determines whether the component is automatically resized to accommodate its text.
	BevelEdges (see page 2953)	Specifies which edges of the control are beveled.
	BevelInner (see page 2953)	Specifies the cut of the inner bevel.
	BevelKind (see page 2954)	Specifies the control's bevel style.
	BevelOuter (see page 2954)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2954)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2955)	Determines the appearance of the control's border.
	Caption (see page 2955)	Specifies a text string that identifies the control to the user.
	Color (see page 2958)	Specifies the background color of the control.
	Constraints (see page 2959)	Specifies the size constraints for the control.
	DoubleBuffered (see page 2959)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2961)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2961)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2962)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2963)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 2964)	Links the component with another windowed control.
	Font (see page 2964)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2965)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2966)	Determines where a control looks for its color information.
	ParentDoubleBuffered (see page 2966)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2966)	Determines where a control looks for its font information.
	ParentShowHint (see page 2967)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2968)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2973)	Enables accelerator characters.
	ShowHint (see page 2973)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2974)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2975)	Determines if the user can tab to a control.
	Transparent (see page 2975)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 2976)	Determines whether the component appears onscreen.

30.1.26.2 TStaticText Properties

The properties of the TStaticText class are listed here.

Published Properties

	Name	Description
	Align (see page 2948)	Determines how the control aligns within its container (parent control).
	Alignment (see page 2951)	Determines how text is aligned within the component.
	Anchors (see page 2952)	Specifies how the control is anchored to its parent.
	AutoSize (see page 2952)	Determines whether the component is automatically resized to accommodate its text.
	BevelEdges (see page 2953)	Specifies which edges of the control are beveled.
	BevelInner (see page 2953)	Specifies the cut of the inner bevel.
	BevelKind (see page 2954)	Specifies the control's bevel style.
	BevelOuter (see page 2954)	Specifies the cut of the outer bevel.
	BiDiMode (see page 2954)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 2955)	Determines the appearance of the control's border.

	Caption (see page 2955)	Specifies a text string that identifies the control to the user.
	Color (see page 2958)	Specifies the background color of the control.
	Constraints (see page 2959)	Specifies the size constraints for the control.
	DoubleBuffered (see page 2959)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 2961)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 2961)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 2962)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 2963)	Controls whether the control responds to mouse, keyboard, and timer events.
	FocusControl (see page 2964)	Links the component with another windowed control.
	Font (see page 2964)	Controls the attributes of text written on or in the control.
	ParentBiDiMode (see page 2965)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 2966)	Determines where a control looks for its color information.
	ParentDoubleBuffered (see page 2966)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 2966)	Determines where a control looks for its font information.
	ParentShowHint (see page 2967)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 2968)	Identifies the pop-up menu associated with the control.
	ShowAccelChar (see page 2973)	Enables accelerator characters.
	ShowHint (see page 2973)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabOrder (see page 2974)	Indicates the position of the control in its parent's tab order.
	TabStop (see page 2975)	Determines if the user can tab to a control.
	Transparent (see page 2975)	Specifies whether the background of the image obscures objects below the image object.
	Visible (see page 2976)	Determines whether the component appears onscreen.

30.1.26.2.1 TStaticText.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of

the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
```

```
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
    clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
    clNavy, clMaroon, clLime, clOlive, clPurple);
var
  i: Integer;
begin
  PageControl1 := TPageControl.Create(Self);
  PageControl1.Parent := Self;
  PageControl1.Align := alClient;
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i]:= TTabSheet.Create(PageControl1);
    with pts[i] do
    begin
      PageControl := PageControl1;
      Name := 'ts' + TabTitles[i];
      Caption := TabTitles[i];
      Brush.Color := colorPalette[i];
   end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

30.1.26.2.2 TStaticText.Alignment Property

Determines how text is aligned within the component.

Pascal

```
property Alignment: TAlignment;
```

C++

```
__property TAlignment Alignment;
```

Description

The possible values of Alignment are

Value	Meaning
taLeftJustify	Text is aligned at the left of the control.
taCenter	Text is centered in the control.
taRightJustify	Text is aligned at the right of the control.

See Also

Font

Caption

30.1.26.2.3 TStaticText.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

30.1.26.2.4 TStaticText.AutoSize Property

Determines whether the component is automatically resized to accommodate its text.

Pascal

```
property AutoSize: Boolean;
```

C++

```
__property Boolean AutoSize;
```

Description

If AutoSize is true, the component is automatically resized— at runtime and design time— whenever the font size or Caption text changes.

See Also

Caption

Font

30.1.26.2.5 TStaticText.BevelEdges Property

Specifies which edges of the control are beveled.

Pascal

```
property BevelEdges;
```

C++

```
__property BevelEdges;
```

Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

See Also

BevelInner

BevelOuter

BevelKind

30.1.26.2.6 TStaticText.BevelInner Property

Specifies the cut of the inner bevel.

Pascal

```
property BevelInner;
```

C++

```
__property BevelInner;
```

Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

See Also

BevelEdges

BevelOuter

BevelKind

BevelWidth

BorderWidth

30.1.26.2.7 TStaticText.BevelKind Property

Specifies the control's bevel style.

Pascal

```
property BevelKind;
```

C++

```
__property BevelKind;
```

Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

See Also

BevelEdges

BevelOuter

BevelInner

BevelWidth

30.1.26.2.8 TStaticText.BevelOuter Property

Specifies the cut of the outer bevel.

Pascal

```
property BevelOuter;
```

C++

```
__property BevelOuter;
```

Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

See Also

BevelEdges

BevelInner

BevelKind

BevelWidth

30.1.26.2.9 TStaticText.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

30.1.26.2.10 TStaticText.BorderStyle Property

Determines the appearance of the control's border.

Pascal

```
property BorderStyle: TStaticBorderStyle;
```

C++

```
__property TStaticBorderStyle BorderStyle;
```

Description

Use BorderStyle to specify the appearance of the border, if any, that appears around the text.

30.1.26.2.11 TStaticText.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is

used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());
```

```
  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
  for I := 0 to High(ButtonCaptions) do
    begin
      with TToolButton.Create(ToolBar) do
      begin
        Parent := ToolBar;
        Caption := ButtonCaptions[I];
        if (ButtonCaptions[I] = '|') then
          Style := tbsSeparator
        else
          Style := tbsButton;
      end;
    end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ToolBar1 := TToolBar.Create(Self);
  ToolBar1.Parent := Self;
  AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
  ToolBar1.ShowCaptions := True;
  ToolBar1.Height := 40;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  I: Integer;
begin
  for I := ToolBar1.ButtonCount - 1 downto 0 do
    ToolBar1.Buttons[I].Free;
  ToolBar1.Free;
end;
```

30.1.26.2.12 TStaticText.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
```

```
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

30.1.26.2.13 TStaticText.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

30.1.26.2.14 TStaticText.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

```
// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DockSite

DockManager

PaintWindow

30.1.26.2.15 TStaticText.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

30.1.26.2.16 TStaticText.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

30.1.26.2.17 TStaticText.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
    else if htNowhere in HT then AttachMode := naAdd
    else if htOnIndent in HT then AttachMode := naInsert;
    TreeView1.Selected.MoveTo(AnItem, AttachMode);
  end;
end;

procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TTreeView;
end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

30.1.26.2.18 TStaticText.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled

controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

30.1.26.2.19 TStaticText.FocusControl Property

Links the component with another windowed control.

Pascal

```
property FocusControl: TWinControl;
```

C++

```
__property TWinControl * FocusControl;
```

Description

Use the FocusControl property to link the TCustomStaticText instance with another windowed control. If the text of Caption includes an accelerator key, the component specified by FocusControl receives focus when the accelerator is pressed.

See Also

Caption

FocusControl (see page 2395)

TWinControl

30.1.26.2.20 TStaticText.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
```

```
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  */
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    for (int i = 0; i < Screen->Fonts->Count; i++)
     ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
  //  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
  }

  void __fastcall TForm1::ComboBox1Change(TObject *Sender)
  {
    RichEdit1->Font->Name = ComboBox1->Text;
  }
```

Delphi Examples:

```
  {
  To use this example, place a TComboBox and TRichEdit or
  TMemo on a form.  During form creation, a list of font names
  is loaded into the combo box. When the combo box is clicked,
  the memo or Rich Edit control font is set to the
  corresponding font name in the combo box.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i : Integer;
  begin
    for i := 0 to Screen.Fonts.Count - 1 do
     ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
  //  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
    RichEdit1.Lines.Text := 'How is it to be you?';
  end;

  procedure TForm1.ComboBox1Click(Sender: TObject);
  begin
    RichEdit1.Font.Name := ComboBox1.Text;
  end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

30.1.26.2.21 TStaticText.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

30.1.26.2.22 TStaticText.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

30.1.26.2.23 TStaticText.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

30.1.26.2.24 TStaticText.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

30.1.26.2.25 TStaticText.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

30.1.26.2.26 TStaticText.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
```

```
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
```

```
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
```

```
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
```

```
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

30.1.26.2.27 TStaticText.ShowAccelChar Property

Enables accelerator characters.

Pascal

```
property ShowAccelChar: Boolean;
```

C++

```
__property Boolean ShowAccelChar;
```

Description

If ShowAccelChar is true, an ampersand in the component's Caption text appears as an underline and makes the character after it into an accelerator key. If ShowAccelChar is false, ampersands appear as ampersands.

See Also

Caption

30.1.26.2.28 TStaticText.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
```

```
      strcpy(string, "Item");
      itoa(i, index, 10);
      strcat(string, index);
      ListBox1->Items->Add(string);
    }
    ListBox1->Hint = "Select an item";
    ListBox1->ShowHint = true;
    Application->ShowHint = true;
  }
  /*
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  */
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

30.1.26.2.29 TStaticText.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

30.1.26.2.30 TStaticText.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

30.1.26.2.31 TStaticText.Transparent Property

Specifies whether the background of the image obscures objects below the image object.

Pascal

```
property Transparent: Boolean;
```

C++

```
__property Boolean Transparent;
```

Description

Set Transparent to true to allow objects behind the TCustomStaticText object to show through the background of the image. Set Transparent to false to make the background of the image opaque.

30.1.26.2.32 TStaticText.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
```

```
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

30.1.26.3 TStaticText Events

The events of the TStaticText class are listed here.

Published Events

	Name	Description
	OnClick (see page 2978)	Occurs when the user clicks the control.
	OnContextPopup (see page 2980)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 2980)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 2981)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 2984)	Occurs when the user drags an object over a control.
	OnEndDock (see page 2987)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 2987)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnMouseActivate (see page 2988)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 2989)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 2991)	Occurs when the user moves the mouse into a control.

	OnMouseLeave (see page 2992)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 2992)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 2994)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 2996)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 2997)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

30.1.26.3.1 TStaticText.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

30.1.26.3.2 TStaticText.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

30.1.26.3.3 TStaticText.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

30.1.26.3.4 TStaticText.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
```

```
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
```

```
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

30.1.26.3.5 TStaticText.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

30.1.26.3.6 TStaticText.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

30.1.26.3.7 TStaticText.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
```

```
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

30.1.26.3.8 TStaticText.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate.

At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.26.3.9 TStaticText.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
```

```
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
```

```
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

30.1.26.3.10 TStaticText.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

30.1.26.3.11 TStaticText.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

30.1.26.3.12 TStaticText.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
```

```
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

30.1.26.3.13 TStaticText.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

30.1.26.3.14 TStaticText.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

30.1.26.3.15 TStaticText.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

30.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCheckBoxState (see page 2999)	TCheckBoxState indicates the state of a check box control.
	TComboBoxStyle (see page 3000)	TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls (see page 2033) unit.
	TEditCharCase (see page 3000)	TEditCharCase indicates how a control forces the case of characters, if at all.
	TEllipsisPosition (see page 3001)	Specifies how ellipsis is placed in text not fitting in allocated rectangle.
	TImageAlignment (see page 3002)	Alignment of image on TButton (see page 2035).
	TListBoxStyle (see page 3003)	TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.
	TScrollCode (see page 3004)	TScrollCode indicates what action the user performed on a scroll bar.
	TScrollStyle (see page 3005)	Identifies a set of scrollbars by their orientation.
	TStaticBorderStyle (see page 3006)	TStaticBorderStyle determines the appearance of the border on a static text control.
	TTextLayout (see page 3007)	TTextLayout indicates where text appears within the client area of a control.

Records

	Name	Description
	TSelection (see page 3006)	TSelection definese the start and position of a selection.

30.2.1 StdCtrls.TCheckBoxState Enumeration

TCheckBoxState indicates the state of a check box control.

Pascal

```
TCheckBoxState = (
  cbUnchecked,
  cbChecked,
  cbGrayed
);
```

C++

```
enum TCheckBoxState {
  cbUnchecked,
  cbChecked,
  cbGrayed
};
```

File

StdCtrls

Description

Following are the possible values of TCheckBoxState:

Value	Meaning
cbUnchecked	The check box has no check mark, indicating that the user has not selected the option.
cbChecked	The check box has a check mark in it, indicating that the user has selected the option.

cbGrayed	The check box has a check mark in it, but it is grayed.

30.2.2 StdCtrls.TComboBoxStyle Enumeration

TComboBoxStyle is the type for the Style property of combo box controls that are defined in the StdCtrls (see page 2033) unit.

Pascal

```
TComboBoxStyle = (
  csDropDown,
  csSimple,
  csDropDownList,
  csOwnerDrawFixed,
  csOwnerDrawVariable
);
```

C++

```
enum TComboBoxStyle {
  csDropDown,
  csSimple,
  csDropDownList,
  csOwnerDrawFixed,
  csOwnerDrawVariable
};
```

File

StdCtrls

Description

These are the values in the TComboBoxStyle type:

Value	Meaning
csDropDown	Creates a drop-down list with an edit box for manually entered text. All items are strings of the same height.
csSimple	Creates an edit box with a fixed list (list box) underneath. The length of the list is determined by the Height of the combo box.
csDropDownList	Creates a drop-down list with no edit box; the user cannot enter text manually. All items are strings of the same height.
csOwnerDrawFixed	Creates an owner-draw drop-down list with no edit box. Each item in the list is the height specified by the ItemHeight property.
csOwnerDrawVariable	Creates an owner-draw drop-down list with no edit box. List items can have varying heights.

30.2.3 StdCtrls.TEditCharCase Enumeration

TEditCharCase indicates how a control forces the case of characters, if at all.

Pascal

```
TEditCharCase = (
  ecNormal,
  ecUpperCase,
```

```
  ecLowerCase
);
```

C++

```
enum TEditCharCase {
  ecNormal,
  ecUpperCase,
  ecLowerCase
};
```

File

StdCtrls

Description

TEditCharCase specifies what case is given to text typed into a control. It can have one of the following values:

Value	Meaning
ecLowerCase	The text is converted to lowercase.
ecNormal	The text appears in mixed case. It is not forced into any case.
ecUpperCase	The text is converted to uppercase.

30.2.4 StdCtrls.TEllipsisPosition Enumeration

Specifies how ellipsis is placed in text not fitting in allocated rectangle.

Pascal

```
TEllipsisPosition = (
  epNone,
  epPathEllipsis,
  epEndEllipsis,
  epWordEllipsis
);
```

C++

```
enum TEllipsisPosition {
  epNone,
  epPathEllipsis,
  epEndEllipsis,
  epWordEllipsis
};
```

File

StdCtrls

Description

TEllipsisPosition specifies how an ellipsis is placed in text that does not fit in its allocated rectangle. The constants here correspond to constants specifying ellipsis placement in the **dwDTFormat** parameter of the Windows **DrawText** and **DrawTextEx** functions.

The following table lists values for TEllipsisPosition:

Value	Meaning
epNone	An ellipsis is not inserted.

epPathEllipsis	Replaces text in the middle with an ellipsis so that the resulting text fits in the rectangle specified. If the text contains backslash (\) characters, treat the text as a path and attempt to preserve as much text as possible after the last backslash. Corresponds to a value of DT_PATH_ELLIPSIS in the **dwDTFormat** parameter.
epEndEllipsis	Replaces text at the end with an ellipsis so that the resulting text fits in the rectangle specified. Any word not fitting in the rectangle is truncated without adding an ellipsis. Corresponds to a value of DT_END_ELLIPSIS in the **dwDTFormat** parameter.
epWordEllipsis	Truncates any word that doesn't fit in the rectangle and adds ellipses. Corresponds to a value of DT_WORD_ELLIPSIS in the **dwDTFormat** parameter.

30.2.5 StdCtrls.TImageAlignment Enumeration

Alignment of image on TButton (see page 2035).

Pascal

```
TImageAlignment = (
  iaLeft,
  iaRight,
  iaTop,
  iaBottom,
  iaCenter
);
```

C++

```
enum TImageAlignment {
  iaLeft,
  iaRight,
  iaTop,
  iaBottom,
  iaCenter
};
```

File

StdCtrls

Description

TImageAlignment indicates the alignment of an image on a TButton (see page 2035), which is the value of its ImageAlignment property. Following are the possible values of TImageAlignment:

Value	Meaning
iaLeft	Left aligned.
iaRight	Right aligned.
iaTop	Top aligned.
iaBottom	Bottom aligned.
iaCenter	Center aligned.

See Also

TButton (see page 2035)

ImageAlignment (see page 2057)

30.2.6 StdCtrls.TListBoxStyle Enumeration

TListBoxStyle specifies the way a listbox control gets its data and how it is drawn.

Pascal

```
TListBoxStyle = (
  lbStandard,
  lbOwnerDrawFixed,
  lbOwnerDrawVariable,
  lbVirtual,
  lbVirtualOwnerDraw
);
```

C++

```
enum TListBoxStyle {
  lbStandard,
  lbOwnerDrawFixed,
  lbOwnerDrawVariable,
  lbVirtual,
  lbVirtualOwnerDraw
};
```

File

StdCtrls

Description

The following table lists the values for the TListBoxStyle type:

Value	Meaning
lbStandard	All items are strings, with each item the same height.
lbOwnerDrawFixed	The list box is owner-drawn, but each item in the list box is the height specified by the ItemHeight property. Each time an item is displayed in an lbOwnerDrawFixed list box, the OnDrawItem event occurs. The event handler for OnDrawItem draws the specified item. The ItemHeight property determines the height of each of the items.
lbOwnerDrawVariable	The list box is owner-drawn, and items in the list box can be of varying heights. Each time an item is displayed in an lbOwnerDrawVariable list box, two events occur. The first is the OnMeasureItem event. The code for the OnMeasureItem handler can set the height of each item. Then the OnDrawItem event occurs. The code for the OnDrawItem handler draws each item in the list box using the size specified by the OnMeasureItem handler.
lbVirtual	The list box is virtual, but all items are strings with the same height. You must indicate the number of items in the list box by setting the Count property. You must supply the items in the list box using an OnData event handler. If the items have associated objects, you must supply them using an OnDataObject event handler. In addition, you must supply an OnDataFind event handler to provide the mapping from the strings in the list box to their indexes.
lbVirtualOwnerDraw	The list box is virtual and owner-drawn. All items are the same height. As with lbOwnerDrawFixed, the item height is specified by the ItemHeight property, and you must draw items using an OnDrawItem event handler. As with lbVirtual, you must set the Count property and supply OnData and OnDataFind event handlers to provide the mapping between items and indexes. If items are associated with objects, you must also supply an OnDataObject event handler.

30.2.7 StdCtrls.TScrollCode Enumeration

TScrollCode indicates what action the user performed on a scroll bar.

Pascal

```
TScrollCode = (
  scLineUp,
  scLineDown,
  scPageUp,
  scPageDown,
  scPosition,
  scTrack,
  scTop,
  scBottom,
  scEndScroll
);
```

C++

```
enum TScrollCode {
  scLineUp,
  scLineDown,
  scPageUp,
  scPageDown,
  scPosition,
  scTrack,
  scTop,
  scBottom,
  scEndScroll
};
```

File

StdCtrls

Description

TScrollCode describes the type of scrolling action requested. It can be any of the following values:

Value	Meaning
scLineUp	User clicked the top or left scroll arrow or pressed the Up or Left arrow key.
scLineDown	User clicked the bottom or right scroll arrow or pressed the Down or Right arrow key.
scPageUp	User clicked the area to the left of the thumb tab or pressed the PgUp key.
scPageDown	User clicked the area to the right of the thumb tab or pressed the PgDn key.
scPosition	User positioned the thumb tab and released it.
scTrack	User is moving the thumb tab.
scTop	User moved the thumb tab to the top or far left on the scroll bar.
scBottom	User moved the thumb tab to the bottom or far right on the scroll bar.
scEndScroll	User finished moving the thumb tab on the scroll bar.

30.2.8 StdCtrls.TScrollStyle Enumeration

Identifies a set of scrollbars by their orientation.

Pascal

```
TScrollStyle = (
  ssNone,
  ssHorizontal,
  ssVertical,
  ssBoth
);
```

C++

```
enum TScrollStyle {
  ssNone,
  ssHorizontal,
  ssVertical,
  ssBoth
};
```

File

StdCtrls

Description

TScrollStyle can have one of the following values:

Value	Meaning
ssNone	No scrollbars are indicated.
ssHorizontal	Only a horizontal scrollbar.
ssVertical	Only a vertical scrollbar.
ssBoth	Both a horizontal and a vertical scrollbar.

C++ Examples:

```
/*
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  RichEdit1->Lines->Clear();
  for (int i = 0; i < 500; i++)
  {
    RichEdit1->Text = RichEdit1->Text + AnsiString("This is sentence ") + IntToStr(i) + ".  ";
    if ((i % 2) == 0) {
      RichEdit1->Text = RichEdit1->Text + AnsiString("\n");
    }
  }
  RichEdit1->SelStart = 0; // move text cursor to top of edit
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
```

```
  RichEdit1->HideScrollBars = CheckBox1->Checked;
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  TScrollStyle ScrollBarA[4] = {ssBoth, ssHorizontal, ssNone, ssVertical };
  RichEdit1->ScrollBars = ScrollBarA[RadioGroup1->ItemIndex];
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Caption = "Scroll Bars";
  RadioGroup1->Items->Clear();
  RadioGroup1->Items->Add("ssBoth");
  RadioGroup1->Items->Add("ssHorizontal");
  RadioGroup1->Items->Add("ssNone");
  RadioGroup1->Items->Add("ssVertical");
  CheckBox1->Caption = "Hide Scroll Bars";
  CheckBox1->Checked = true;
  Button1->Caption = "Load Text";
  RichEdit1->Text = "Just a little text.";
  RichEdit1->WordWrap = False;
}
```

30.2.9 StdCtrls.TSelection Record

TSelection definese the start and position of a selection.

Pascal

```
TSelection = record
  EndPos: Integer;
  StartPos: Integer;
end;
```

C++

```
struct TSelection {
  int EndPos;
  int StartPos;
};
```

File

StdCtrls

Description

TSelection is used internally in Delphi to hold the start and end position of a selection.

30.2.10 StdCtrls.TStaticBorderStyle Enumeration

TStaticBorderStyle determines the appearance of the border on a static text control.

Pascal

```
TStaticBorderStyle = (
  sbsNone,
  sbsSingle,
  sbsSunken
);
```

C++

```
enum TStaticBorderStyle {
  sbsNone,
  sbsSingle,
  sbsSunken
};
```

File

StdCtrls

Description

These are the values in the TStaticBorderStyle type:

Value	Meaning
sbsNone	No visible border.
sbsSingle	Single-line border.
sbsSunken	Raised border.

30.2.11 StdCtrls.TTextLayout Enumeration

TTextLayout indicates where text appears within the client area of a control.

Pascal

```
TTextLayout = (
  tlTop,
  tlCenter,
  tlBottom
);
```

C++

```
enum TTextLayout {
  tlTop,
  tlCenter,
  tlBottom
};
```

File

StdCtrls

Description

The following table describes the possible values:

Value	Meaning
tlTop	The text appears at the top of the its control.
tlCenter	The text is vertically centered in the control.
tlBottom	The text appears along the bottom of the control.

30.3 Types

The following table lists types in this documentation.

Types

Name	Description
TButtonActionLinkClass (see page 3008)	TButtonActionLinkClass defines the metaclass for TButtonActionLink (see page 2106).
TCustomComboBoxStringsClass (see page 3008)	TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 2339).
TDrawItemEvent (see page 3009)	TDrawItemEvent is the type of an OnDrawItem event handler.
TFNDrawText (see page 3010)	TFNDrawText represents the procedure type for the events that specify the details of the Text property of a control.
TLBFindDataEvent (see page 3010)	TLBFindDataEvent is the type of the list box OnDataFind event handler.
TLBGetDataEvent (see page 3011)	TLBGetDataEvent is the type of the list box OnData event handler.
TLBGetDataObjectEvent (see page 3011)	TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.
TMeasureItemEvent (see page 3012)	TMeasureItemEvent is the type for the OnMeasureItem event handler.
TOwnerDrawState (see page 3012)	Indicates state information that can influence how an item is drawn.
TPushButtonActionLinkClass (see page 3013)	TPushButtonActionLinkClass defines the metaclass for TPushButtonActionLink (see page 2811).
TScrollEvent (see page 3013)	TScrollEvent is the type of an OnScroll event handler.

30.3.1 StdCtrls.TButtonActionLinkClass Type

TButtonActionLinkClass defines the metaclass for TButtonActionLink (see page 2106).

Pascal

```
TButtonActionLinkClass = class of TButtonActionLink;
```

C++

```
TButtonActionLink TButtonActionLinkClass;
```

File

StdCtrls

Description

TButtonActionLinkClass is the metaclass for TButtonActionLink (see page 2106). Its value is the class reference for TButtonActionLink (see page 2106) or for one of its descendants.

See Also

TButtonActionLink (see page 2106)

Class References

30.3.2 StdCtrls.TCustomComboBoxStringsClass Type

TCustomComboBoxStringsClass describes a descendant of TCustomComboBoxStrings (see page 2339).

Pascal

```
TCustomComboBoxStringsClass = class of TCustomComboBoxStrings;
```

C++

```
TCustomComboBoxStrings TCustomComboBoxStringsClass;
```

File

StdCtrls

Description

TCustomComboBoxStringsClass is the type for a class reference of TCustomComboBoxStrings (see page 2339) or any of its descendants. It is a pointer to the runtime type information (RTTI) for the class type.

See Also

Class References

30.3.3 StdCtrls.TDrawItemEvent Type

TDrawItemEvent is the type of an OnDrawItem event handler.

Pascal

```
TDrawItemEvent = procedure (Control: TWinControl; Index: Integer; Rect: TRect; State:
TOwnerDrawState) of object;
```

C++

```
(Control: TWinControl; Index: Integer; Rect: TRect; State: TOwnerDrawState) (
TDrawItemEvent)();
```

File

StdCtrls

Description

TDrawItemEvent defines the type of method procedure that you must define in order to implement the OnDrawItem event of a visual component such as TListBox (see page 2657).

OnDrawItem is used to write a handler for drawing of the items in list boxes with the TListBoxStyle (see page 3003) Style values lbOwnerDrawFixed, lbOwnerDrawVariable, or lbVirtualOwnerDraw. OnDrawItem occurs when the list box needs to display an item. OnDrawItem occurs only for owner-draw list boxes.

Control	is a reference to the component invoking the OnDrawEvent.
Index	is the index of the item in the list box.
Rect	defines the rectangle in which the user drawing must be made.
State	defines state of the item to be drawn.

See Also

OnDrawItem (see page 2709)

Style (see page 2695)

TListBoxStyle (see page 3003)

TOwnerDrawState

Procedural types

Creating events

30.3.4 StdCtrls.TFNDrawText Type

TFNDrawText represents the procedure type for the events that specify the details of the Text property of a control.

Pascal

```
TFNDrawText = procedure (DC: HDC; const Text: UnicodeString; var TextRect: TRect; TextFlags:
Cardinal) of object;
```

C++

```
(DC: HDC; const Text: UnicodeString; var TextRect: TRect; TextFlags: Cardinal) (
TFNDrawText)();
```

File

StdCtrls

Description

A variable of TFNDrawText type references an event that occurs when creating a control and specifies the details of the Text property.

See Also

Text

30.3.5 StdCtrls.TLBFindDataEvent Type

TLBFindDataEvent is the type of the list box OnDataFind event handler.

Pascal

```
TLBFindDataEvent = function (Control: TWinControl; FindString: string): Integer of object;
```

C++

```
(Control: TWinControl; FindString: string): Integer ( TLBFindDataEvent)();
```

File

StdCtrls

Description

Control is the virtual list box whose item index is needed.

FindString is the text of the item whose index is needed.

The event handler returns the index of the item identified by FindString. This return value must be in the range from 0 to one less than the value of the Count property.

See Also

Procedural types

Creating events

30.3.6 StdCtrls.TLBGetDataEvent Type

TLBGetDataEvent is the type of the list box OnData event handler.

Pascal

```
TLBGetDataEvent = procedure (Control: TWinControl; Index: Integer; var Data: string) of object;
```

C++

```
(Control: TWinControl; Index: Integer; var Data: string) ( TLBGetDataEvent)();
```

File

StdCtrls

Description

Control is the virtual list box whose item is needed.

Index is the index of the desired item. Index must be less than the value of the Count property.

Data returns the string that is the value of the specified item.

See Also

Procedural types

Creating events

30.3.7 StdCtrls.TLBGetDataObjectEvent Type

TLBGetDataObjectEvent is the type of the list box OnDataObject event handler.

Pascal

```
TLBGetDataObjectEvent = procedure (Control: TWinControl; Index: Integer; var DataObject:
TObject) of object;
```

C++

```
(Control: TWinControl; Index: Integer; var DataObject: TObject) ( TLBGetDataObjectEvent)();
```

File

StdCtrls

Description

Control is the virtual list box that contains the item associated with the object.

Index is the index of the desired item. Index must be less than the value of the Count property.

DataObject returns the object associated with the specified item.

See Also

Procedural types

Creating events

30.3.8 StdCtrls.TMeasureItemEvent Type

TMeasureItemEvent is the type for the OnMeasureItem event handler.

Pascal

```
TMeasureItemEvent = procedure (Control: TWinControl; Index: Integer; var Height: Integer) of object;
```

C++

```
(Control: TWinControl; Index: Integer; var Height: Integer) ( TMeasureItemEvent)();
```

File

StdCtrls

Description

TMeasureItemEvent is the type for event handlers on custom-drawn controls that allow different items in a list to have different heights.

Control is a reference to the control that contains the item.

Index is the index of the item in the Items property.

Height is the default height of the item, in pixels. The event handler can reset Height to the appropriate value.

See Also

Procedural types

Creating events

30.3.9 StdCtrls.TOwnerDrawState Type

Indicates state information that can influence how an item is drawn.

Pascal

```
TOwnerDrawState = Windows.TOwnerDrawState;
```

C++

```
Windows.TOwnerDrawState TOwnerDrawState;
```

File

StdCtrls

Description

TOwnerDrawState is used by event handlers in owner-draw controls to indicate the state of an item about to be drawn. It is a set that includes zero or more of the following:

Value	Meaning
odSelected	The item is selected.
odGrayed	The item should appear grayed.
odDisabled	The item is disabled.

odChecked	The item should appear checked. (This does not apply to all objects)
odFocused	Keyboard input is directed to the item.
odDefault	The item is the default item (This does not apply to all objects)
odHotLight	The item is hot-lighted because under the mouse (only if the operating system supports hot-lighting)
odInactive	The item and its associated menu are inactive
odNoAccel	Any underline of an accelerator character should be skipped when drawing the control
odNoFocusRect	Do not display visual cues that indicate focus
odReserved1	Not currently used.
odReserved2	Not currently used.
odComboBoxEdit	The drawing takes place in the edit box of a combo box control.

30.3.10 StdCtrls.TPushButtonActionLinkClass Type

TPushButtonActionLinkClass defines the metaclass for TPushButtonActionLink (see page 2811).

Pascal

```
TPushButtonActionLinkClass = class of TPushButtonActionLink;
```

C++

```
TPushButtonActionLink TPushButtonActionLinkClass;
```

File

StdCtrls

Description

TPushButtonActionLinkClass is the metaclass for TPushButtonActionLink (see page 2811). Its value is the class reference for TPushButtonActionLink (see page 2811) or for one of its descendants.

See Also

TPushButtonActionLink (see page 2811)

Class References

30.3.11 StdCtrls.TScrollEvent Type

TScrollEvent is the type of an OnScroll event handler.

Pascal

```
TScrollEvent = procedure (Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer) of
object;
```

C++

```
(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer) ( TScrollEvent)();
```

File

StdCtrls

Description

TScrollEvent defines the type of method procedure that you must define in order to implement the OnScroll event of a visual component such as TScrollBar (see page 2881).

The OnScroll event occurs when the user scrolls a scroll bar with the mouse or keyboard.

Sender is a reference to the component invoking the OnScroll event.

ScrollCode defines the user scroll action that was performed.

ScrollPos provides the new scroll position.

See Also

OnScroll (see page 2942)

TScrollCode (see page 3004)

Procedural types

Creating events

31 StdStyleActnCtrls Namespace

This is namespace StdStyleActnCtrls.

Topics

Name	Description
Classes (see page 3016)	The following table lists classes in this documentation.
Variables (see page 3019)	The following table lists variables in this documentation.

Classes

	Name	Description
	TStandardStyleActionBars (see page 3016)	This is class StdStyleActnCtrls.TStandardStyleActionBars.

Variables

Name	Description
StandardStyle (see page 3019)	This is variable StdStyleActnCtrls.StandardStyle.

31.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TStandardStyleActionBars (see page 3016)	This is class StdStyleActnCtrls.TStandardStyleActionBars.

31.1.1 TStandardStyleActionBars Class

Class Hierarchy

Pascal

```
TStandardStyleActionBars = class(TActionBarStyleEx);
```

C++

```
class TStandardStyleActionBars : public TActionBarStyleEx;
```

File

StdStyleActnCtrls

Description

This is class StdStyleActnCtrls.TStandardStyleActionBars.

31.1.1.1 TStandardStyleActionBars Members

The following tables list the members exposed by TStandardStyleActionBars.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 3017)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 3017)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 3017)	Returns the control class type on an action bar item.
	GetPopupClass (see page 3018)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 3018)	This is GetScrollBtnClass, a member of class TStandardStyleActionBars.
	GetStyleName (see page 3018)	Returns the style associated with action manager.

31.1.1.2 TStandardStyleActionBars Methods

The methods of the TStandardStyleActionBars class are listed here.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 3017)	Returns the base class for ActionBand menus for customizing action bands.

	GetColorMapClass (see page 3017)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 3017)	Returns the control class type on an action bar item.
	GetPopupClass (see page 3018)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 3018)	This is GetScrollBtnClass, a member of class TStandardStyleActionBars.
	GetStyleName (see page 3018)	Returns the style associated with action manager.

31.1.1.2.1 TStandardStyleActionBars.GetAddRemoveItemClass Method

Returns the base class for ActionBand menus for customizing action bands.

Pascal

```
function GetAddRemoveItemClass(ActionBar: TCustomActionBar): TCustomAddRemoveItemClass;
override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass(TCustomActionBar ActionBar);
```

Description

Call GetAddRemoveItemClass to customize action bands.

See Also

TCustomAddRemoveItem

31.1.1.2.2 TStandardStyleActionBars.GetColorMapClass Method

Returns the class type of the colormap for an action bar.

Pascal

```
function GetColorMapClass(ActionBar: TCustomActionBar): TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetColorMapClass(TCustomActionBar ActionBar);
```

Description

Returns the class type TCustomActionBarColorMap, which is a colormap associated with an action bar. Override this method to customize the appearance of your applications.

31.1.1.2.3 TStandardStyleActionBars.GetControlClass Method

Returns the control class type on an action bar item.

Pascal

```
function GetControlClass(ActionBar: TCustomActionBar; AnItem: TActionClientItem):
TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TCustomActionBar ActionBar,
TActionClientItem AnItem);
```

Description

Returns the control class type on an item in the action bar, such as a button or menu item.

31.1.1.2.4 TStandardStyleActionBars.GetPopupClass Method

Returns the popup class added to a menu action band.

Pascal

```
function GetPopupClass(ActionBar: TCustomActionBar): TCustomPopupClass; override;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass(TCustomActionBar ActionBar);
```

Description

Call GetPopupClass to find the popup menu associated with a menu action band. Override GetPopupClass to change which popup class is associated with a menu action band depending on the state of the control.

See Also

TCustomActionPopupMenu

31.1.1.2.5 TStandardStyleActionBars.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; override;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TStandardStyleActionBars.

31.1.1.2.6 TStandardStyleActionBars.GetStyleName Method

Returns the style associated with action manager.

Pascal

```
function GetStyleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetStyleName();
```

Description

Returns the value of the Style property.

31.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
StandardStyle (see page 3019)	This is variable StdStyleActnCtrls.StandardStyle.

31.2.1 StdStyleActnCtrls.StandardStyle Variable

Pascal

```
StandardStyle: TStandardStyleActionBars;
```

C++

```
TStandardStyleActionBars StandardStyle;
```

File

StdStyleActnCtrls

Description

This is variable StdStyleActnCtrls.StandardStyle.

32 StrUtils Namespace

This is namespace StrUtils.

Topics

Name	Description
Functions (see page 3024)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3055)	The following table lists structs, records, enums in this documentation.
Types (see page 3056)	The following table lists types in this documentation.
Variables (see page 3058)	The following table lists variables in this documentation.
Constants (see page 3060)	The following table lists constants in this documentation.

Constants

Name	Description
WordDelimiters (see page 3060)	This is constant StrUtils.WordDelimiters.

Functions

	Name	Description
	AnsiContainsStr (see page 3025)	Indicates whether one string is a (case-sensitive) substring of another.
	AnsiContainsText (see page 3026)	Indicates whether one string is a (case-insensitive) substring of another.
	AnsiEndsStr (see page 3026)	Indicates whether one string is a (case-sensitive) suffix of another.
	AnsiEndsText (see page 3027)	Indicates whether one string is a (case-insensitive) suffix of another.
	AnsiIndexStr (see page 3028)	Provides the index of a specified string in an array of strings.
	AnsiIndexText (see page 3028)	Provides the index of a specified string in an array of strings.
	AnsiLeftStr (see page 3029)	Returns the substring of a specified length that appears at the start of a string.
	AnsiMatchStr (see page 3029)	Indicates whether an array of strings contains an exact match to a specified string.
	AnsiMatchText (see page 3030)	Indicates whether an array of strings contains a case-insensitive match to a specified string.
	AnsiMidStr (see page 3030)	Returns the substring of a specified length that appears at a specified position in a string.
	AnsiReplaceStr (see page 3031)	Replaces all occurrences of a substring within a string, using case-sensitive search.
	AnsiReplaceText (see page 3032)	Replaces all case-insensitive matches of a substring with another string.
	AnsiResemblesText (see page 3032)	Indicates whether two strings are similar.
	AnsiReverseString (see page 3033)	Returns reversed string.
	AnsiRightStr (see page 3033)	Returns the substring of a specified length that appears at the end of a string.
	AnsiStartsStr (see page 3034)	Indicates whether one string is a (case-sensitive) prefix of another.

	AnsiStartsText (see page 3034)	Indicates whether one string is a (case-insensitive) prefix of another.
	ContainsStr (see page 3035)	Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	ContainsText (see page 3035)	Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	DecodeSoundexInt (see page 3036)	Converts an integer representation of a Soundex (see page 3048) encoding into the corresponding phonetic string.
	DecodeSoundexWord (see page 3036)	Converts a Word representation of a Soundex (see page 3048) encoding into the corresponding phonetic string.
	DupeString (see page 3037)	Returns the concatenation of a string with itself a specified number of repeats.
	EndsStr (see page 3037)	Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.
	EndsText (see page 3038)	Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.
	IfThen (see page 3038)	Conditionally returns one of two specified values.
	IndexStr (see page 3039)	IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	IndexText (see page 3039)	IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	LeftBStr (see page 3040)	Returns the substring of a specified number of bytes that appears at the start of a string.
	LeftStr (see page 3040)	Returns the substring of a specified length that appears at the start of a string.
	LeftStr (see page 3040)	Returns the substring of a specified length that appears at the start of a string.
	MatchStr (see page 3041)	MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MatchText (see page 3041)	MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MidBStr (see page 3042)	Returns the substring of a specified number of bytes that appears at a specified position in a string.
	MidStr (see page 3043)	Returns the substring of a specified length that appears at a specified position in a string.
	MidStr (see page 3043)	Returns the substring of a specified length that appears at a specified position in a string.
	PosEx (see page 3043)	Returns the index value of a substring.
	RandomFrom (see page 3044)	Returns a randomly selected element from an array.
	ReplaceStr (see page 3044)	Replaces all occurrences of a substring within a string, using case-sensitive search.
	ReplaceText (see page 3045)	ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.
	ResemblesText (see page 3046)	ResemblesText uses a Soundex (see page 3048) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 3048) method, and false otherwise.
	ReverseString (see page 3046)	Returns the reverse of a specified string.
	RightBStr (see page 3046)	Returns the substring of a specified number of bytes that appears at the end of a string.
	RightStr (see page 3047)	Returns the substring of a specified length that appears at the end of a string.
	RightStr (see page 3047)	Returns the substring of a specified length that appears at the end of a string.
	SearchBuf (see page 3048)	Locates a substring within a text buffer.
	SearchBuf (see page 3048)	Locates a substring within a text buffer.
	Soundex (see page 3048)	Converts a string into its Soundex representation.
	SoundexCompare (see page 3049)	Compares the Soundex (see page 3048) representations of two strings.
	SoundexInt (see page 3050)	Converts a string into an integer that represents its phonetic value.
	SoundexProc (see page 3050)	Indicates whether two strings are similar.
	SoundexSimilar (see page 3051)	Indicates whether two strings are similar.
	SoundexWord (see page 3052)	Converts a string into a Word that represents its phonetic value.

	StartsStr (see page 3052)	Determines whether the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StartsText (see page 3053)	StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StuffString (see page 3053)	Inserts a substring into a specified position of a string, replacing the current characters.

Structs, Records, Enums

	Name	Description
	TStringSeachOption (see page 3055)	TStringSeachOption is an enumeration that describes the behavior of string searches.

Types

Name	Description
TCompareTextProc (see page 3056)	TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.
TSoundexIntLength (see page 3056)	TSoundexIntLength The maximum calculation length for the SoundexInt (see page 3050) function. The integer value can be from 1 to 8.
TSoundexLength (see page 3057)	TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.
TStringSearchOptions (see page 3057)	TStringSearchOptions is set of the enumeration TStringSeachOption (see page 3055).

Variables

Name	Description
AnsiResemblesProc (see page 3058)	Controls the algorithm used by AnsiResemblesText (see page 3032) to determine when two strings are similar.
ResemblesProc (see page 3058)	Controls the algorithm used by AnsiResemblesText (see page 3032) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 3058).

32.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiContainsStr (see page 3025)	Indicates whether one string is a (case-sensitive) substring of another.
	AnsiContainsText (see page 3026)	Indicates whether one string is a (case-insensitive) substring of another.
	AnsiEndsStr (see page 3026)	Indicates whether one string is a (case-sensitive) suffix of another.
	AnsiEndsText (see page 3027)	Indicates whether one string is a (case-insensitive) suffix of another.
	AnsiIndexStr (see page 3028)	Provides the index of a specified string in an array of strings.
	AnsiIndexText (see page 3028)	Provides the index of a specified string in an array of strings.
	AnsiLeftStr (see page 3029)	Returns the substring of a specified length that appears at the start of a string.
	AnsiMatchStr (see page 3029)	Indicates whether an array of strings contains an exact match to a specified string.
	AnsiMatchText (see page 3030)	Indicates whether an array of strings contains a case-insensitive match to a specified string.
	AnsiMidStr (see page 3030)	Returns the substring of a specified length that appears at a specified position in a string.
	AnsiReplaceStr (see page 3031)	Replaces all occurrences of a substring within a string, using case-sensitive search.
	AnsiReplaceText (see page 3032)	Replaces all case-insensitive matches of a substring with another string.
	AnsiResemblesText (see page 3032)	Indicates whether two strings are similar.
	AnsiReverseString (see page 3033)	Returns reversed string.
	AnsiRightStr (see page 3033)	Returns the substring of a specified length that appears at the end of a string.
	AnsiStartsStr (see page 3034)	Indicates whether one string is a (case-sensitive) prefix of another.
	AnsiStartsText (see page 3034)	Indicates whether one string is a (case-insensitive) prefix of another.
	ContainsStr (see page 3035)	Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	ContainsText (see page 3035)	Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.
	DecodeSoundexInt (see page 3036)	Converts an integer representation of a Soundex (see page 3048) encoding into the corresponding phonetic string.
	DecodeSoundexWord (see page 3036)	Converts a Word representation of a Soundex (see page 3048) encoding into the corresponding phonetic string.
	DupeString (see page 3037)	Returns the concatenation of a string with itself a specified number of repeats.
	EndsStr (see page 3037)	Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.
	EndsText (see page 3038)	Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.
	IfThen (see page 3038)	Conditionally returns one of two specified values.
	IndexStr (see page 3039)	IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	IndexText (see page 3039)	IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.
	LeftBStr (see page 3040)	Returns the substring of a specified number of bytes that appears at the start of a string.
	LeftStr (see page 3040)	Returns the substring of a specified length that appears at the start of a string.
	LeftStr (see page 3040)	Returns the substring of a specified length that appears at the start of a string.
	MatchStr (see page 3041)	MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

	MatchText (see page 3041)	MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.
	MidBStr (see page 3042)	Returns the substring of a specified number of bytes that appears at a specified position in a string.
	MidStr (see page 3043)	Returns the substring of a specified length that appears at a specified position in a string.
	MidStr (see page 3043)	Returns the substring of a specified length that appears at a specified position in a string.
	PosEx (see page 3043)	Returns the index value of a substring.
	RandomFrom (see page 3044)	Returns a randomly selected element from an array.
	ReplaceStr (see page 3044)	Replaces all occurrences of a substring within a string, using case-sensitive search.
	ReplaceText (see page 3045)	ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.
	ResemblesText (see page 3046)	ResemblesText uses a Soundex (see page 3048) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 3048) method, and false otherwise.
	ReverseString (see page 3046)	Returns the reverse of a specified string.
	RightBStr (see page 3046)	Returns the substring of a specified number of bytes that appears at the end of a string.
	RightStr (see page 3047)	Returns the substring of a specified length that appears at the end of a string.
	RightStr (see page 3047)	Returns the substring of a specified length that appears at the end of a string.
	SearchBuf (see page 3048)	Locates a substring within a text buffer.
	SearchBuf (see page 3048)	Locates a substring within a text buffer.
	Soundex (see page 3048)	Converts a string into its Soundex representation.
	SoundexCompare (see page 3049)	Compares the Soundex (see page 3048) representations of two strings.
	SoundexInt (see page 3050)	Converts a string into an integer that represents its phonetic value.
	SoundexProc (see page 3050)	Indicates whether two strings are similar.
	SoundexSimilar (see page 3051)	Indicates whether two strings are similar.
	SoundexWord (see page 3052)	Converts a string into a Word that represents its phonetic value.
	StartsStr (see page 3052)	Determines whether the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StartsText (see page 3053)	StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.
	StuffString (see page 3053)	Inserts a substring into a specified position of a string, replacing the current characters.

32.1.1 StrUtils.AnsiContainsStr Function

Indicates whether one string is a (case-sensitive) substring of another.

Pascal

```
function AnsiContainsStr(const AText: string; const ASubText: string): Boolean; overload;
```

C++

```
Boolean AnsiContainsStr(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

AnsiContainsStr returns true if the string specified by ASubText appears as a substring of the string specified by AText.

Comparison is performed case-sensitively.

See Also

AnsiResemblesText (see page 3032)

AnsiStartsStr (see page 3034)

AnsiEndsStr (see page 3026)

AnsiCompareStr (see page 4050)

Pos (see page 3627)

StringReplace (see page 4253)

AnsiContainsText (see page 3026)

32.1.2 StrUtils.AnsiContainsText Function

Indicates whether one string is a (case-insensitive) substring of another.

Pascal

```
function AnsiContainsText(const AText: string; const ASubText: string): Boolean; overload;
```

C++

```
Boolean AnsiContainsText(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

AnsiContainsText returns true if the string specified by ASubText appears as a substring of the string specified by AText. Comparison is performed case-insensitively and based on the current system locale.

See Also

AnsiResemblesText (see page 3032)

AnsiStartsText (see page 3034)

AnsiEndsText (see page 3027)

AnsiCompareText (see page 4051)

Pos (see page 3627)

StringReplace (see page 4253)

AnsiContainsStr (see page 3025)

32.1.3 StrUtils.AnsiEndsStr Function

Indicates whether one string is a (case-sensitive) suffix of another.

Pascal

```
function AnsiEndsStr(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiEndsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiEndsStr returns true if the string specified by ASubText appears as the trailing part of the string specified by AText. Comparison is performed case-sensitively.

See Also

AnsiStartsStr (see page 3034)

AnsiContainsStr (see page 3025)

Pos (see page 3627)

RightStr (see page 3047)

AnsiEndsText (see page 3027)

32.1.4 StrUtils.AnsiEndsText Function

Indicates whether one string is a (case-insensitive) suffix of another.

Pascal

```
function AnsiEndsText(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiEndsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiEndsText returns true if the string specified by ASubText appears as the trailing part of the string specified by AText. Comparison is performed case-insensitively and based on the current system locale.

See Also

AnsiStartsText (see page 3034)

AnsiContainsText (see page 3026)

Pos (see page 3627)

RightStr (see page 3047)

32.1.5 StrUtils.AnsiIndexStr Function

Provides the index of a specified string in an array of strings.

Pascal

```
function AnsiIndexStr(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int AnsiIndexStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

AnsiIndexStr returns the 0-offset index of the first string it finds in AValues that matches AText case-sensitively. If the string specified by AText does not have an exact match in AValues, AnsiIndexStr returns –1.

See Also

AnsiMatchStr (see page 3029)

RandomFrom (see page 3044)

AnsiIndexText (see page 3028)

32.1.6 StrUtils.AnsiIndexText Function

Provides the index of a specified string in an array of strings.

Pascal

```
function AnsiIndexText(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int AnsiIndexText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

AnsiIndexText returns the 0-offset index of the first string it finds in AValues that matches AText case-insensitively. If the string specified by AText does not have a (possibly case-insensitive) match in AValues, AnsiIndexText returns –1. Comparisons are based on the current system locale.

See Also

AnsiMatchText (see page 3030)

RandomFrom (see page 3044)

AnsiIndexStr (see page 3028)

32.1.7 StrUtils.AnsiLeftStr Function

Returns the substring of a specified length that appears at the start of a string.

Pascal

```
function AnsiLeftStr(const AText: string; const ACount: Integer): string; overload;
```

C++

```
UnicodeString AnsiLeftStr(const UnicodeString AText, const int ACount);
```

File

StrUtils

Description

AnsiLeftStr returns the leading characters of AText up to a length of ACount characters.

See Also

Copy (see page 3547)

TrimLeft (see page 4301)

AnsiStartsText (see page 3034)

AnsiStartsStr (see page 3034)

LeftStr (see page 3040)

LeftBStr (see page 3040)

32.1.8 StrUtils.AnsiMatchStr Function

Indicates whether an array of strings contains an exact match to a specified string.

Pascal

```
function AnsiMatchStr(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean AnsiMatchStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

Call AnsiMatchStr to determine whether the string specified by AText matches one of the strings in AValues. Comparisons are case-sensitive.

See Also

AnsiIndexStr (see page 3028)

AnsiSameStr (see page 4055)

AnsiMatchText (see page 3030)

32.1.9 StrUtils.AnsiMatchText Function

Indicates whether an array of strings contains a case-insensitive match to a specified string.

Pascal

```
function AnsiMatchText(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean AnsiMatchText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

Call AnsiMatchText to determine whether the string specified by AText matches one of the strings in AValues. Comparisons are case-insensitive and use the current system locale.

AText is the string to locate.

AValues is the array of strings to search.

Note: AValues_Size is the index of the last element in AValues (one less than the number of elements).

AnsiMatchText returns true if AText (or a string that matches except for case) appears in AValues, false if it does not appear in the array.

See Also

AnsiIndexText (see page 3028)

AnsiSameText (see page 4055)

AnsiMatchStr (see page 3029)

32.1.10 StrUtils.AnsiMidStr Function

Returns the substring of a specified length that appears at a specified position in a string.

Pascal

```
function AnsiMidStr(const AText: string; const AStart: Integer; const ACount: Integer):
string; overload;
```

C++

```
UnicodeString AnsiMidStr(const UnicodeString AText, const int AStart, const int ACount);
```

File

StrUtils

Description

AnsiMidStr returns a substring Count characters at AText[AStart].

If AStart is larger than the length of AText, AnsiMidStr returns an empty string.

If Count specifies more characters than are available, only the characters from AText[AStart] to the end of AText are returned.

See Also

Copy (see page 3547)

LeftStr (see page 3040)

RightStr (see page 3047)

MidStr (see page 3043)

MidBStr (see page 3042)

32.1.11 StrUtils.AnsiReplaceStr Function

Replaces all occurrences of a substring within a string, using case-sensitive search.

Pascal

```
function AnsiReplaceStr(const AText: string; const AFromText: string; const AToText: string):
string; overload;
```

C++

```
UnicodeString AnsiReplaceStr(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

Use AnsiReplaceStr to replace all occurrences of the substring `AFromText` with the string `AToText`, within the string `AText`. AnsiReplaceStr returns the string obtained after replacement.

Use ReplaceText (see page 3045) to do a case-insensitive replacement.

Note: The `Ansi` prefix in the function name appears for backward compatibility reasons only. Both the parameters and the return value are of type `string`, which is an alias for `UnicodeString` (see page 3381). To do the replacement in an `AnsiString` context, use the ReplaceStr (see page 3044) function. Also, to do the replacement in a `WideString` (see page 3467) context, use the WideReplaceStr function.

Note: Recursive replacement of substrings is not supported. This means that if the substitution of `AToText` results in a new match for `AFromText`, that match is not replaced.

ReplaceStr (see page 3044) is the same function as AnsiReplaceStr. In writing your application, we recommend that you use ReplaceStr (see page 3044), to avoid any confusion caused by the `Ansi` prefix.

The StringReplace (see page 4253) function is a more general replacement routine, giving case-sensitivity options, as well as the option to decide whether to replace all occurrences or the first occurrence of the substring in the string.

See Also

ReplaceStr

WideReplaceStr (see page 6879)

StringReplace (see page 4253)

ReplaceText (see page 3045)

PosEx (see page 3043)

StuffString (see page 3053)

32.1.12 StrUtils.AnsiReplaceText Function

Replaces all case-insensitive matches of a substring with another string.

Pascal

```
function AnsiReplaceText(const AText: string; const AFromText: string; const AToText: string):
string; overload;
```

C++

```
UnicodeString AnsiReplaceText(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

Call AnsiReplaceText to scan the string specified by AText and replace all occurrences of the string specified by AFromText with the string specified by AToText. Occurrences of AFromText are identified by a case-insensitive comparison that uses the current system locale. The identification of substrings is not recursive: if the substitution of AToText results in a new match for AFromText, that match is not replaced.

See Also

AnsiContainsText (see page 3026)

AnsiReplaceStr (see page 3031)

StringReplace (see page 4253)

SearchBuf (see page 3048)

StuffString (see page 3053)

32.1.13 StrUtils.AnsiResemblesText Function

Indicates whether two strings are similar.

Pascal

```
function AnsiResemblesText(const AText: string; const AOther: string): Boolean; overload;
```

C++

```
Boolean AnsiResemblesText(const UnicodeString AText, const UnicodeString AOther);
```

File

StrUtils

Description

AnsiResemblesText returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar.

The algorithm that controls whether AnsiResemblesText considers two strings similar is specified by the AnsiResemblesProc (see page 3058) variable. By default, AnsiResemblesProc (see page 3058) is set to SoundexProc (see page 3050), which converts both strings to a phonetic representation using a Soundex (see page 3048) algorithm and compares up to 4 characters of the resulting phonetic representation. By changing the function assigned to AnsiResemblesProc (see page 3058), you can change the comparison made by AnsiResemblesText.

See Also

AnsiResemblesProc (see page 3058)

SoundexProc (see page 3050)

32.1.14 StrUtils.AnsiReverseString Function

Returns reversed string.

Pascal

```
function AnsiReverseString(const AText: string): string; overload;
```

C++

```
UnicodeString AnsiReverseString(const UnicodeString AText);
```

File

StrUtils

Description

AnsiReverseString returns AText with its characters reversed.

32.1.15 StrUtils.AnsiRightStr Function

Returns the substring of a specified length that appears at the end of a string.

Pascal

```
function AnsiRightStr(const AText: string; const ACount: Integer): string; overload;
```

C++

```
UnicodeString AnsiRightStr(const UnicodeString AText, const int ACount);
```

File

StrUtils

Description

AnsiRightStr returns the trailing characters of AText up to a length of ACount characters. Thus, for example, if AText is the string "Programmer" and ACount is 7, AnsiRightStr returns the string "grammer".

See Also

Copy (see page 3547)

TrimRight (see page 4302)

AnsiEndsText (see page 3027)

AnsiEndsStr (see page 3026)

RightStr (see page 3047)

RightBStr (see page 3046)

32.1.16 StrUtils.AnsiStartsStr Function

Indicates whether one string is a (case-sensitive) prefix of another.

Pascal

```
function AnsiStartsStr(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiStartsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiStartsStr returns true if the string specified by ASubText appears as the leading part of the string specified by AText. Comparison is performed case-sensitively.

See Also

AnsiEndsStr (see page 3026)

AnsiContainsStr (see page 3025)

Pos (see page 3627)

LeftStr (see page 3040)

AnsiStartsText (see page 3034)

32.1.17 StrUtils.AnsiStartsText Function

Indicates whether one string is a (case-insensitive) prefix of another.

Pascal

```
function AnsiStartsText(const ASubText: string; const AText: string): Boolean; overload;
```

C++

```
Boolean AnsiStartsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

AnsiStartsText returns true if the string specified by ASubText appears as the leading part of the string specified by AText. Comparison is performed case-insensitively and uses the current system locale.

See Also

AnsiEndsText (see page 3027)

AnsiContainsText (see page 3026)

Pos (see page 3627)

LeftStr (see page 3040)

AnsiStartsStr (see page 3034)

32.1.18 StrUtils.ContainsStr Function

Performs a case sensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.

Pascal

```
function ContainsStr(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean ContainsStr(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

For a case insensitive search, use the ContainsText (see page 3035) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ContainsText (see page 3035)

32.1.19 StrUtils.ContainsText Function

Performs a case insensitive search for substring ASubText in string AText. Returns true if one or more instances of ASubText exist in AText.

Pascal

```
function ContainsText(const AText: string; const ASubText: string): Boolean;
```

C++

```
Boolean ContainsText(const UnicodeString AText, const UnicodeString ASubText);
```

File

StrUtils

Description

For a case sensitive search, use the ContainsStr (see page 3035) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ContainsStr (see page 3035)

32.1.20 StrUtils.DecodeSoundexInt Function

Converts an integer representation of a Soundex (see page 3048) encoding into the corresponding phonetic string.

Pascal

```
function DecodeSoundexInt(AValue: Integer): string;
```

C++

```
UnicodeString DecodeSoundexInt(int AValue);
```

File

StrUtils

Description

Call DecodeSoundexInt to convert the integer representation of a string returned by the SoundexInt (see page 3050) function into the corresponding Soundex (see page 3048) string representation.

A Soundex (see page 3048) representation of a string is a phonetic encoding where each character of the Soundex (see page 3048) representation represents a family of similar phonemes.

See Also

SoundexInt (see page 3050)

DecodeSoundexWord (see page 3036)

Soundex (see page 3048)

32.1.21 StrUtils.DecodeSoundexWord Function

Converts a Word representation of a Soundex (see page 3048) encoding into the corresponding phonetic string.

Pascal

```
function DecodeSoundexWord(AValue: Word): string;
```

C++

```
UnicodeString DecodeSoundexWord(Word AValue);
```

File

StrUtils

Description

Call DecodeSoundexWord to convert the unsigned 16-bit representation of a string returned by the SoundexWord (see page 3052) function into the corresponding Soundex (see page 3048) string representation.

A Soundex (see page 3048) representation of a string is a phonetic encoding where each character of the Soundex (see page 3048) representation represents a family of similar phonemes.

See Also

SoundexWord (see page 3052)

DecodeSoundexInt (see page 3036)

Soundex (see page 3048)

32.1.22 StrUtils.DupeString Function

Returns the concatenation of a string with itself a specified number of repeats.

Pascal

```
function DupeString(const AText: string; ACount: Integer): string; overload;
```

C++

```
UnicodeString DupeString(const UnicodeString AText, int ACount);
```

File

StrUtils

Description

DupeString returns a string containing ACount repeats of the string specified by AText. For example,

S := DupeString('Ha', 5);

sets S to the string 'HaHaHaHaHa'.

See Also

StringOfChar (see page 3677)

StringReplace (see page 4253)

32.1.23 StrUtils.EndsStr Function

Determines if string AText ends with substring ASubText using a case sensitive string comparison. Returns true if AText ends with ASubText.

Pascal

```
function EndsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean EndsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case insensitive comparison, use the EndsText (see page 3038) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

EndsText (see page 3038)

32.1.24 StrUtils.EndsText Function

Determines if string AText ends with substring ASubText using a case insensitive string comparison. Returns true if AText ends with ASubText.

Pascal

```
function EndsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean EndsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case sensitive comparison, use the EndsStr (see page 3037) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

EndsStr (see page 3037)

32.1.25 StrUtils.IfThen Function

Conditionally returns one of two specified values.

Pascal

```
function IfThen(AValue: Boolean; const ATrue: string; AFalse: string = ''): string; overload;
```

C++

```
UnicodeString IfThen(Boolean AValue, const UnicodeString ATrue, UnicodeString AFalse = '');
```

File

StrUtils

Description

IfThen checks the expression passed as AValue and returns ATrue if it evaluates to true, or AFalse if it evaluates to false. In Delphi, if the AFalse parameter is omitted, IfThen returns 0 or an empty string when AValue evaluates to False.

32.1.26 StrUtils.IndexStr Function

IndexStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.

Pascal

```
function IndexStr(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int IndexStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case insensitive comparison, use the IndexText (see page 3039) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

IndexText (see page 3039)

32.1.27 StrUtils.IndexText Function

IndexText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns the zero-based index of the first match in the array, or -1 if no match is found.

Pascal

```
function IndexText(const AText: string; const AValues: array of string): Integer; overload;
```

C++

```
int IndexText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case sensitive comparison, use the IndexStr (see page 3039) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

IndexStr (see page 3039)

32.1.28 StrUtils.LeftBStr Function

Returns the substring of a specified number of bytes that appears at the start of a string.

Pascal

```
function LeftBStr(const AText: AnsiString; const AByteCount: Integer): AnsiString;
```

C++

```
AnsiString LeftBStr(const AnsiString AText, const int AByteCount);
```

File

StrUtils

Description

LeftBStr returns the leading bytes of AText up to a length of AByteCount bytes.

Note: LeftBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use LeftStr (see page 3040).

See Also

Copy (see page 3547)

TrimLeft (see page 4301)

AnsiStartsText (see page 3034)

AnsiStartsStr (see page 3034)

AnsiLeftStr (see page 3029)

LeftStr (see page 3040)

32.1.29 StrUtils.LeftStr Function

Returns the substring of a specified length that appears at the start of a string.

Pascal

```
function LeftStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
function LeftStr(const AText: WideString; const ACount: Integer): WideString; overload;
```

C++

```
AnsiString LeftStr(const AnsiString AText, const int ACount);
BSTR LeftStr(const BSTR AText, const int ACount);
```

File

StrUtils

Description

LeftStr returns the leading characters of AText up to a length of ACount characters.

Note: If AText is an AnsiString and the current locale uses multi-byte characters, LeftStr may return more than ACount bytes. To return a specific number of bytes, use LeftBStr (see page 3040).

See Also

Copy (see page 3547)

TrimLeft (see page 4301)

AnsiStartsText (see page 3034)

AnsiStartsStr (see page 3034)

AnsiLeftStr (see page 3029)

LeftBStr (see page 3040)

32.1.30 StrUtils.MatchStr Function

MatchStr determines if any of the strings in the array AValues match the string specified by AText using a case sensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

Pascal

```
function MatchStr(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean MatchStr(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case insensitive match, use the MatchText (see page 3041) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

MatchText (see page 3041)

32.1.31 StrUtils.MatchText Function

MatchText determines if any of the strings in the array AValues match the string specified by AText using a case insensitive comparison. It returns true if at least one of the strings in the array match, or false if none of the strings match.

Pascal

```
function MatchText(const AText: string; const AValues: array of string): Boolean; overload;
```

C++

```
Boolean MatchText(const UnicodeString AText, const array of string AValues);
```

File

StrUtils

Description

For a case sensitive comparison, use the MatchStr (see page 3041) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

MatchStr (see page 3041)

32.1.32 StrUtils.MidBStr Function

Returns the substring of a specified number of bytes that appears at a specified position in a string.

Pascal

```
function MidBStr(const AText: AnsiString; const AByteStart: Integer; const AByteCount:
Integer): AnsiString;
```

C++

```
AnsiString MidBStr(const AnsiString AText, const int AByteStart, const int AByteCount);
```

File

StrUtils

Description

MidBStr returns a substring AByteCount bytes beginning at AByteStart bytes.

If AByteStart is larger than the length of AText in bytes, MidBStr returns an empty string.

If AByteCount specifies more bytes than are available, only the available bytes are returned.

Note: MidBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use MidStr (see page 3043).

See Also

Copy (see page 3547)

LeftStr (see page 3040)

RightStr (see page 3047)

AnsiMidStr (see page 3030)

MidStr (see page 3043)

32.1.33 StrUtils.MidStr Function

Returns the substring of a specified length that appears at a specified position in a string.

Pascal

```
function MidStr(const AText: AnsiString; const AStart: Integer; const ACount: Integer):
AnsiString; overload;
function MidStr(const AText: WideString; const AStart: Integer; const ACount: Integer):
WideString; overload;
```

C++

```
AnsiString MidStr(const AnsiString AText, const int AStart, const int ACount);
BSTR MidStr(const BSTR AText, const int AStart, const int ACount);
```

File

StrUtils

Description

MidStr returns a substring Count characters at AText[AStart].

If AStart is larger than the length of AText, MidStr returns an empty string.

If Count specifies more characters than are available, only the characters from AText[AStart] to the end of AText are returned.

Note: If AText is an AnsiString and the current locale uses multi-byte characters, MidStr may return more than ACount bytes. To return a specific number of bytes, use MidBStr (see page 3042).

See Also

Copy (see page 3547)

LeftStr (see page 3040)

RightStr (see page 3047)

AnsiMidStr (see page 3030)

MidBStr (see page 3042)

32.1.34 StrUtils.PosEx Function

Returns the index value of a substring.

Pascal

```
function PosEx(const SubStr: string; const S: string; Offset: Integer = 1): Integer; overload;
```

C++

```
int PosEx(const UnicodeString SubStr, const UnicodeString S, int Offset = 1);
```

File

StrUtils

Description

PosEx returns the index of SubStr in S, beginning the search at Offset. If Offset is 1 (default), PosEx is equivalent to Pos (see

page 3627).

PosEx returns 0 if SubStr is not found, if Offset is greater than the length of S, or if Offset is less than 1.

See Also

Pos (see page 3627)

32.1.35 StrUtils.RandomFrom Function

Returns a randomly selected element from an array.

Pascal

```
function RandomFrom(const AValues: array of string): string; overload;
```

C++

```
UnicodeString RandomFrom(const array of string AValues);
```

File

StrUtils

Description

RandomFrom returns a randomly selected element from the array passed as the AValues argument. In C++, AValues_Size is the index of the last array element (one less than the number of elements.)

To initialize the random number generator, add a single call Randomize (see page 3633) or assign a value to the RandSeed (see page 3834) variable before making any calls to RandomFrom.

See Also

Randomize (see page 3633)

RandSeed (see page 3834)

32.1.36 StrUtils.ReplaceStr Function

Replaces all occurrences of a substring within a string, using case-sensitive search.

Pascal

```
function ReplaceStr(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
UnicodeString ReplaceStr(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

Use ReplaceStr to replace all occurrences of the substring `AFromText` with the string `AToText`, within the string `AText`. ReplaceStr returns the string obtained after replacement.

Use ReplaceText (see page 3045) to do a case-insensitive replacement.

Note: Both the parameters and the return value are of type `string`, which is an alias for `UnicodeString` (see page 3381). To do the replacement in an `AnsiString` context, use the ReplaceStr function. Also, to do the replacement in a `WideString` (see page 3467) context, use the WideReplaceStr function.

Note: Recursive replacement of substrings is not supported. This means that if the substitution of `AToText` results in a new match for `AFromText`, that match is not replaced.

ReplaceStr is the same function as AnsiReplaceStr (see page 3031).

The StringReplace (see page 4253) function is a more general replacement routine, giving case-sensitivity options, as well as the option to decide whether to replace all occurrences or the first occurrence of the substring in the string.

See Also

ReplaceStr

WideReplaceStr (see page 6879)

StringReplace (see page 4253)

ReplaceText (see page 3045)

PosEx (see page 3043)

StuffString (see page 3053)

32.1.37 StrUtils.ReplaceText Function

ReplaceText replaces all instances of string AFromText to string AToText in the source string AText and returns this value as the result. The replacement is case insensitive.

Pascal

```
function ReplaceText(const AText: string; const AFromText: string; const AToText: string):
string;
```

C++

```
UnicodeString ReplaceText(const UnicodeString AText, const UnicodeString AFromText, const
UnicodeString AToText);
```

File

StrUtils

Description

For a case sensitive replacement, use the ReplaceStr (see page 3044) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

ReplaceStr (see page 3044)

32.1.38 StrUtils.ResemblesText Function

ResemblesText uses a Soundex (see page 3048) method to determine if string AText and string AOther are similar. The result is True if the two strings a similar according to the Soundex (see page 3048) method, and false otherwise.

Pascal

```
function ResemblesText(const AText: string; const AOther: string): Boolean; overload;
```

C++

```
Boolean ResemblesText(const UnicodeString AText, const UnicodeString AOther);
```

File

StrUtils

See Also

Commonly Used Long String Routines

String Types

String Constants

AnsiResemblesText (see page 3032)

32.1.39 StrUtils.ReverseString Function

Returns the reverse of a specified string.

Pascal

```
function ReverseString(const AText: string): string; overload;
```

C++

```
UnicodeString ReverseString(const UnicodeString AText);
```

File

StrUtils

Description

ReverseString returns the string specified by AText with the characters in reverse order.

See Also

DupeString (see page 3037)

32.1.40 StrUtils.RightBStr Function

Returns the substring of a specified number of bytes that appears at the end of a string.

Pascal

```
function RightBStr(const AText: AnsiString; const AByteCount: Integer): AnsiString;
```

C++

```
AnsiString RightBStr(const AnsiString AText, const int AByteCount);
```

File

StrUtils

Description

RightBStr returns the trailing bytes of AText up to a length of AByteCount bytes.

Note: RightBStr assumes all characters are one byte long, and does not behave correctly if AText contains multi-byte characters. To handle all strings correctly, use RightStr (see page 3047).

See Also

Copy (see page 3547)

TrimRight (see page 4302)

AnsiEndsText (see page 3027)

AnsiEndsStr (see page 3026)

AnsiRightStr (see page 3033)

32.1.41 StrUtils.RightStr Function

Returns the substring of a specified length that appears at the end of a string.

Pascal

```
function RightStr(const AText: AnsiString; const ACount: Integer): AnsiString; overload;
function RightStr(const AText: WideString; const ACount: Integer): WideString; overload;
```

C++

```
AnsiString RightStr(const AnsiString AText, const int ACount);
BSTR RightStr(const BSTR AText, const int ACount);
```

File

StrUtils

Description

RightStr returns the trailing characters of AText up to a length of ACount characters. Thus, for example, if AText is the string "Programmer" and ACount is 7, RightStr returns the string "grammer".

Note: If AText is an AnsiString and the current locale uses multi-byte characters, RightStr may return more than ACount bytes. To return a specific number of bytes, use RightBStr (see page 3046).

See Also

Copy (see page 3547)

TrimRight (see page 4302)

AnsiEndsText (see page 3027)

AnsiEndsStr (see page 3026)

AnsiRightStr (see page 3033)

32.1.42 StrUtils.SearchBuf Function

Locates a substring within a text buffer.

Pascal

```
function SearchBuf(Buf: PAnsiChar; BufLen: Integer; SelStart: Integer; SelLength: Integer;
SearchString: AnsiString; Options: TStringSearchOptions = [soDown]): PAnsiChar; overload;
function SearchBuf(Buf: PWideChar; BufLen: Integer; SelStart: Integer; SelLength: Integer;
SearchString: UnicodeString; Options: TStringSearchOptions = [soDown]): PWideChar; overload;
```

C++

```
PAnsiChar SearchBuf(PAnsiChar Buf, int BufLen, int SelStart, int SelLength, AnsiString
SearchString, TStringSearchOptions Options = [soDown]);
PWideChar SearchBuf(PWideChar Buf, int BufLen, int SelStart, int SelLength, UnicodeString
SearchString, TStringSearchOptions Options = [soDown]);
```

File

StrUtils

Description

Call SearchBuf to search for a specified search string within a text buffer.

Buf is the text buffer to search.

BufLen is the length, in bytes, of Buf.

SelStart is the first character of the search when Options indicates a backward search (does not include soDown). The first character in Buf has position 0.

SelLength is the number of characters after SelStart that the search begins when Options indicates a forward search (includes soDown).

SearchString is the string to find in Buf.

Options determines whether the search runs forward (soDown) or backward from SelStart or SelStart+SelLength, whether the search is case sensitive (soMatchCase), and whether the matching string must be a whole word (soWholeWord).

If SearchBuf finds a match, it returns a pointer to the first character of the matching string in Buf. If it does not find a match, SearchBuf returns nil (Delphi) or NULL (C++).

See Also

TFindDialog

32.1.43 StrUtils.Soundex Function

Converts a string into its Soundex representation.

Pascal

```
function Soundex(const AText: string; ALength: TSoundexLength = 4): string;
```

C++

```
UnicodeString Soundex(const UnicodeString AText, TSoundexLength ALength = 4);
```

File

StrUtils

Description

Soundex converts the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It returns up to the first ALength Soundex codes in the resulting string representation.

See Also

SoundexWord (see page 3052)

SoundexInt (see page 3050)

SoundexSimilar (see page 3051)

SoundexCompare (see page 3049)

SoundexProc (see page 3050)

32.1.44 StrUtils.SoundexCompare Function

Compares the Soundex (see page 3048) representations of two strings.

Pascal

```
function SoundexCompare(const AText: string; const AOther: string; ALength: TSoundexLength =
4): Integer;
```

C++

```
int SoundexCompare(const UnicodeString AText, const UnicodeString AOther, TSoundexLength
ALength = 4);
```

File

StrUtils

Description

SoundexCompare uses a Soundex (see page 3048) algorithm to convert the strings specified by AText and AOther into phonetic representations where each character of the resulting string represents one of six families of similar phonemes. It then compares the first ALength characters of the resulting representations by calling AnsiCompareStr (see page 4050).

SoundexCompare returns 0 if the first ALength characters of the phonetic representations are identical, a value less then 0 if the representation of AText is less than the representation of AOther, and a value greater than 0 if the representation of AText is greater than the representation of AOther.

See Also

SoundexSimilar (see page 3051)

SoundexWord (see page 3052)

SoundexInt (see page 3050)

AnsiCompareStr (see page 4050)

Soundex (see page 3048)

SoundexProc (see page 3050)

32.1.45 StrUtils.SoundexInt Function

Converts a string into an integer that represents its phonetic value.

Pascal

```
function SoundexInt(const AText: string; ALength: TSoundexIntLength = 4): Integer;
```

C++

```
int SoundexInt(const UnicodeString AText, TSoundexIntLength ALength = 4);
```

File

StrUtils

Description

SoundexInt uses a Soundex (see page 3048) algorithm to convert the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then returns an integer that uniquely represents the first ALength Soundex (see page 3048) codes in that string representation.

Note: The TSoundexIntLengthLength type represents an integer between 1 and 8. This is because the size of the integer return value limits the number of Soundex (see page 3048) codes that can be used in the final representation.

See Also

SoundexSimilar (see page 3051)

SoundexWord (see page 3052)

DecodeSoundexInt (see page 3036)

SoundexCompare (see page 3049)

Soundex (see page 3048)

SoundexProc (see page 3050)

32.1.46 StrUtils.SoundexProc Function

Indicates whether two strings are similar.

Pascal

```
function SoundexProc(const AText: string; const AOther: string): Boolean;
```

C++

```
Boolean SoundexProc(const UnicodeString AText, const UnicodeString AOther);
```

File

StrUtils

Description

SoundexProc returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar. SoundexProc is the default value of the AnsiResemblesProc (see page 3058) variable, which provides the underlying implementation of AnsiResemblesText (see page 3032).

SoundexProc uses a Soundex (see page 3048) algorithm to convert both strings into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then compares up to 4 digits in the resulting representation, returning true if they are all the same and false if they are not.

Note: SoundexProc is the same as the SoundexSimilar (see page 3051) function using an ALength value of 4.

See Also

AnsiResemblesProc (see page 3058)

SoundexSimilar (see page 3051)

SoundexCompare (see page 3049)

SoundexWord (see page 3052)

SoundexInt (see page 3050)

Soundex (see page 3048)

AnsiResemblesText (see page 3032)

32.1.47 StrUtils.SoundexSimilar Function

Indicates whether two strings are similar.

Pascal

```
function SoundexSimilar(const AText: string; const AOther: string; ALength: TSoundexLength =
4): Boolean;
```

C++

```
Boolean SoundexSimilar(const UnicodeString AText, const UnicodeString AOther, TSoundexLength
ALength = 4);
```

File

StrUtils

Description

SoundexSimilar returns true if the string specified by AText is "similar" to the string specified by AOther, or false if the two strings are not sufficiently similar.

SoundexSimilar uses a Soundex (see page 3048) algorithm to convert both strings into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then compares up to ALength digits in the resulting representation, returning true if they are all the same and false if they are not.

See Also

SoundexCompare (see page 3049)

SoundexWord (see page 3052)

SoundexInt (see page 3050)

Soundex (see page 3048)

SoundexProc (see page 3050)

32.1.48 StrUtils.SoundexWord Function

Converts a string into a Word that represents its phonetic value.

Pascal

```
function SoundexWord(const AText: string): Word;
```

C++

```
Word SoundexWord(const UnicodeString AText);
```

File

StrUtils

Description

SoundexWord uses a Soundex (see page 3048) algorithm to convert the string specified by AText into a phonetic representation where each character of the resulting string represents one of six families of similar phonemes. It then returns an unsigned 16-bit value that uniquely represents the first 4 Soundex (see page 3048) codes in that string representation.

See Also

SoundexSimilar (see page 3051)

SoundexInt (see page 3050)

DecodeSoundexWord (see page 3036)

SoundexCompare (see page 3049)

Soundex (see page 3048)

SoundexProc (see page 3050)

32.1.49 StrUtils.StartsStr Function

Determines whether the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.

Pascal

```
function StartsStr(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean StartsStr(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case insensitive comparison, use the StartsText (see page 3053) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

StartsText (see page 3053)

32.1.50 StrUtils.StartsText Function

StartsText determines if the substring ASubText begins the string AText using a case insensitive algorithm. If ASubText matches the begining of AText, the result is true, otherwise it is false.

Pascal

```
function StartsText(const ASubText: string; const AText: string): Boolean;
```

C++

```
Boolean StartsText(const UnicodeString ASubText, const UnicodeString AText);
```

File

StrUtils

Description

For a case sensitive comparison, use the StartsStr (see page 3052) routine.

See Also

Commonly Used Long String Routines

String Types

String Constants

StartsStr (see page 3052)

32.1.51 StrUtils.StuffString Function

Inserts a substring into a specified position of a string, replacing the current characters.

Pascal

```
function StuffString(const AText: string; AStart: Cardinal; ALength: Cardinal; const ASubText:
string): string; overload;
```

C++

```
UnicodeString StuffString(const UnicodeString AText, unsigned AStart, unsigned ALength, const
UnicodeString ASubText);
```

File

StrUtils

Description

StuffString returns a new string that is constructed by replacing the characters of AText specified by AStart and ALength with the string specified by ASubText.

AText is the string to which ASubText is added. It is not modified by StuffString.

AStart is the index of the first character in AText to be replaced, where 1 identifies the first character, 2 identifies the second character, and so on.

ALength is the number of characters in AText that are replaced by ASubText. If ALength is 0, ASubText is inserted into AText without removing any characters. If AText is negative, the characters preceding the insertion point are duplicated after the occurrence of ASubText in the result string.

ASubText is the substring to insert into AText, replacing the characters specified by AStart and ALength.

See Also

AnsiReplaceStr (see page 3031)

AnsiReplaceText (see page 3032)

StringReplace (see page 4253)

32.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TStringSeachOption (see page 3055)	TStringSeachOption is an enumeration that describes the behavior of string searches.

32.2.1 StrUtils.TStringSeachOption Enumeration

TStringSeachOption is an enumeration that describes the behavior of string searches.

Pascal

```
TStringSeachOption = (
  soDown,
  soMatchCase,
  soWholeWord
);
```

C++

```
enum TStringSeachOption {
  soDown,
  soMatchCase,
  soWholeWord
};
```

File

StrUtils

Description

Note the unusual spelling. TStringSeachOption can have one of the following values:

Value	Meaning
soDown	A forward search is performed through the target text from the specified start point to the end of the search buffer. If soDown is not specified a reverse search is performed from the start point to the begining of the search buffer.
soMatchCase	Performs a case sensitive search.
soWholeWord	Match whole words, not pieces of words. Whole words are determined using standard identifier parser rules.

See Also

Commonly Used Long String Routines

String Types

String Constants

TStringSearchOptions (see page 3057)

32.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCompareTextProc (see page 3056)	TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.
TSoundexIntLength (see page 3056)	TSoundexIntLength The maximum calculation length for the SoundexInt (see page 3050) function. The integer value can be from 1 to 8.
TSoundexLength (see page 3057)	TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.
TStringSearchOptions (see page 3057)	TStringSearchOptions is set of the enumeration TStringSeachOption (see page 3055).

32.3.1 StrUtils.TCompareTextProc Type

TCompareTextProc is a function pointer which specifies a function to compare two string values, AText, and AOther, and return true if these two values are equivilant.

Pascal

```
TCompareTextProc = function (const AText, AOther: string): Boolean;
```

C++

```
(const AText, AOther: string): Boolean ( TCompareTextProc)();
```

File

StrUtils

See Also

TStringSeachOption (see page 3055)

32.3.2 StrUtils.TSoundexIntLength Type

TSoundexIntLength The maximum calculation length for the SoundexInt (see page 3050) function. The integer value can be from 1 to 8.

Pascal

```
TSoundexIntLength = 1..8;
```

C++

```
1..8 TSoundexIntLength;
```

File

StrUtils

Description

This value determines how many characters to use when generating the soundex string from the source string. There is a maximum of 8 characters due to the maximum capacity of the Integer type.

32.3.3 StrUtils.TSoundexLength Type

TSoundexLength The maximum calculation length of the SoundEx function. This value determines how many characters to use when generating the soundex string form the source string.

Pascal

```
TSoundexLength = 1..MaxInt;
```

C++

```
1..MaxInt TSoundexLength;
```

File

StrUtils

See Also

TSoundexIntLength (see page 3056)

32.3.4 StrUtils.TStringSearchOptions Type

TStringSearchOptions is set of the enumeration TStringSeachOption (see page 3055).

Pascal

```
TStringSearchOptions = set of TStringSeachOption;
```

C++

```
TStringSeachOption TStringSearchOptions;
```

File

StrUtils

Description

Note the unusual spelling of the enumeration.

See Also

TStringSeachOption (see page 3055)

32.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AnsiResemblesProc (see page 3058)	Controls the algorithm used by AnsiResemblesText (see page 3032) to determine when two strings are similar.
ResemblesProc (see page 3058)	Controls the algorithm used by AnsiResemblesText (see page 3032) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 3058).

32.4.1 StrUtils.AnsiResemblesProc Variable

Controls the algorithm used by AnsiResemblesText (see page 3032) to determine when two strings are similar.

Pascal

```
AnsiResemblesProc: TCompareTextProc = SoundexProc;
```

C++

```
TCompareTextProc AnsiResemblesProc = SoundexProc;
```

File

StrUtils

Description

The AnsiResemblesText (see page 3032) function uses the function assigned to AnsiResemblesProc to determine when two strings should be considered similar. By default, AnsiResemblesProc is set to SoundexProc (see page 3050).

To make AnsiResemblesText (see page 3032) use a different algorithm, you can change AnsiResemblesProc to another function. For example, you can change the number of phonemes compared by using a function that assigns a different length to SoundexSimilar (see page 3051) than then default value of 4. Alternately, you can create a function that uses a different algorithm entirely.

See Also

AnsiResemblesText (see page 3032)

SoundexSimilar (see page 3051)

32.4.2 StrUtils.ResemblesProc Variable

Controls the algorithm used by AnsiResemblesText (see page 3032) to determine when two strings are similar. It is the same as AnsiResemblesProc (see page 3058).

Pascal

```
ResemblesProc: TCompareTextProc = SoundexProc;
```

C++

```
TCompareTextProc ResemblesProc = SoundexProc;
```

File

StrUtils

Description

The AnsiResemblesText (see page 3032) function uses the function assigned to ResemblesProc to determine when two strings should be considered similar. By default, ResemblesProc is set to SoundexProc (see page 3050).

To make AnsiResemblesText (see page 3032) use a different algorithm, you can change ResemblesProc to another function. For example, you can change the number of phonemes compared by using a function that assigns a different length to SoundexSimilar (see page 3051) than then default value of 4. Alternately, you can create a function that uses a different algorithm entirely.

See Also

AnsiResemblesText (see page 3032)

AnsiResemblesProc (see page 3058)

SoundexSimilar (see page 3051)

32.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
WordDelimiters (see page 3060)	This is constant StrUtils.WordDelimiters.

32.5.1 StrUtils.WordDelimiters Constant

Pascal

```
WordDelimiters: set of AnsiChar = [#0..#255] - ['a'..'z','A'..'Z','1'..'9','0'];
```

C++

```
set of AnsiChar WordDelimiters = [#0..#255] - ['a'..'z','A'..'Z','1'..'9','0'];
```

File

StrUtils

Description

This is constant StrUtils.WordDelimiters.

33 SvcMgr Namespace

This is namespace SvcMgr.

Topics

Name	Description
Classes (see page 3063)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3102)	The following table lists structs, records, enums in this documentation.
Types (see page 3106)	The following table lists types in this documentation.
Variables (see page 3109)	The following table lists variables in this documentation.
Constants (see page 3110)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDependencies (see page 3063)	TDependencies is a container for TDependency (see page 3066) objects.
	TDependency (see page 3066)	TDependency object represents a dependent service or load ordering group used by TService (see page 3071) objects.
	TEventLogger (see page 3068)	This is class SvcMgr.TEventLogger.
	TService (see page 3071)	TService encapsulates a Windows NT service.
	TServiceApplication (see page 3089)	TServiceApplication encapsulates a Windows NT service application.
	TServiceThread (see page 3095)	TServiceThread represents the thread associated with a specific service (TService (see page 3071) object).

Constants

Name	Description
CM_SERVICE_CONTROL_CODE (see page 3110)	This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

Structs, Records, Enums

	Name	Description
	TCurrentStatus (see page 3102)	TCurrentStatus indicates the state of a service.
	TErrorSeverity (see page 3103)	TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.
	TServiceType (see page 3103)	TServiceType indicates the type of a service.
	TStartType (see page 3104)	TStartType indicates how a service starts up.

Types

Name	Description
TContinueEvent (see page 3106)	TContinueEvent is the type for event handlers that respond when the Service Control manager resumes a service.
TPauseEvent (see page 3106)	TPauseEvent is the type for event handlers that respond when the Service Control manager pauses a service.
TServiceController (see page 3107)	TServiceController is the type for a registered service's handler function.
TServiceEvent (see page 3107)	TServiceEvent is the procedure type used for service events that do not require parameters.
TStartEvent (see page 3108)	TStartEvent is the type for event handlers that respond when the Service Control manager starts a service.
TStopEvent (see page 3108)	TStopEvent is the type for event handlers that respond when the Service Control manager stops a service.

Variables

Name	Description
Application (see page 3109)	Represents a service Application for a service project.

33.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDependencies (see page 3063)	TDependencies is a container for TDependency (see page 3066) objects.
	TDependency (see page 3066)	TDependency object represents a dependent service or load ordering group used by TService (see page 3071) objects.
	TEventLogger (see page 3068)	This is class SvcMgr.TEventLogger.
	TService (see page 3071)	TService encapsulates a Windows NT service.
	TServiceApplication (see page 3089)	TServiceApplication encapsulates a Windows NT service application.
	TServiceThread (see page 3095)	TServiceThread represents the thread associated with a specific service (TService (see page 3071) object).

33.1.1 TDependencies Class

TDependencies is a container for TDependency (see page 3066) objects.

Class Hierarchy

Pascal

```
TDependencies = class(TCollection);
```

C++

```
class TDependencies : public TCollection;
```

File

SvcMgr

Description

TDependencies is a collection of TDependency (see page 3066) objects, each of which represents a dependent service or load ordering group for a service (TService (see page 3071) instance). TDependencies objects are the implementation of the Dependencies property of TService (see page 3071).

See Also

TDependency (see page 3066)

TService (see page 3071)

33.1.1.1 TDependencies Members

The following tables list the members exposed by TDependencies.

Protected Methods

	Name	Description
	GetOwner (see page 3064)	Returns the owner of an object.

Public Methods

	Name	Description
	Create (see page 3064)	Creates an instance of a TDependencies (see page 3063) object.

Public Properties

	Name	Description
	Items (see page 3065)	Lists the dependency objects in the dependencies collection.

33.1.1.2 TDependencies.Create Constructor

Creates an instance of a TDependencies (see page 3063) object.

Pascal

```
constructor Create(Owner: TPersistent);
```

C++

```
__fastcall TDependencies(TPersistent * Owner);
```

Description

Applications should not need to call the constructor directly. Instances of TService (see page 3071) create the TDependencies (see page 3063) object from their own constructors. The Owner parameter is the TService (see page 3071) instance that uses the TDependencies (see page 3063) object to implement its Dependencies property.

Create sets the TDependency (see page 3066) owner to the Owner parameter and calls the inherited Create.

See Also

Create

TService (see page 3071)

33.1.1.3 TDependencies Methods

The methods of the TDependencies class are listed here.

Protected Methods

	Name	Description
	GetOwner (see page 3064)	Returns the owner of an object.

33.1.1.3.1 TDependencies.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

33.1.1.4 TDependencies Properties

The properties of the TDependencies class are listed here.

Public Properties

	Name	Description
	Items (see page 3065)	Lists the dependency objects in the dependencies collection.

33.1.1.4.1 TDependencies.Items Property

Lists the dependency objects in the dependencies collection.

Pascal

```
property Items [Index: Integer]: TDependency;
```

C++

```
__property TDependency Items[int Index];
```

Description

The value of the Index parameter corresponds to the Index property of each TDependency (see page 3066) object in the collection. It represents the position of the dependency object in the dependencies collection.

Note: In Delphi, Items is the default property of TDependencies (see page 3063). This means that you can omit the property names. For example

Note: Service1.Dependencies.Items[i]

Note: can be written

Note: Service1.Dependencies[i]

Note: In C++, Items can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

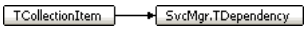
See Also

Index

33.1.2 TDependency Class

TDependency object represents a dependent service or load ordering group used by TService (see page 3071) objects.

Class Hierarchy



Pascal

```
TDependency = class(TCollectionItem);
```

C++

```
class TDependency : public TCollectionItem;
```

File

SvcMgr

Description

A TDependency object represents a new service or load ordering group that must start before the service represented by a TService (see page 3071) object. Collections of dependency objects are TDependencies (see page 3063) objects. TService (see page 3071) has Dependencies and LoadGroup properties that use dependency collections. Each dependency has a Name (see page 3068) property that identifies the service or load ordering group, and a boolean IsGroup (see page 3067) property to indicate whether the dependency represents a load ordering group.

See Also

TDependencies (see page 3063)

TService (see page 3071)

Service Name Properties

33.1.2.1 TDependency Members

The following tables list the members exposed by TDependency.

Protected Methods

	Name	Description
	GetDisplayName (see page 3067)	Returns the name of the collection item as it appears in the collection editor.

Published Properties

	Name	Description
	IsGroup (see page 3067)	Indicates whether the dependency represents a load ordering group for a service (TService (see page 3071) object).
	Name (see page 3068)	Identifies the dependent service or load ordering group used by a TService (see page 3071) object.

33.1.2.2 TDependency Methods

The methods of the TDependency class are listed here.

Protected Methods

	Name	Description
	GetDisplayName (see page 3067)	Returns the name of the collection item as it appears in the collection editor.

33.1.2.2.1 TDependency.GetDisplayName Method

Returns the name of the collection item as it appears in the collection editor.

Pascal

```
function GetDisplayName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the protected read implementation of the DisplayName property.

See Also

DisplayName

GetNamePath

33.1.2.3 TDependency Properties

The properties of the TDependency class are listed here.

Published Properties

	Name	Description
	IsGroup (see page 3067)	Indicates whether the dependency represents a load ordering group for a service (TService (see page 3071) object).
	Name (see page 3068)	Identifies the dependent service or load ordering group used by a TService (see page 3071) object.

33.1.2.3.1 TDependency.IsGroup Property

Indicates whether the dependency represents a load ordering group for a service (TService (see page 3071) object).

Pascal

```
property IsGroup: Boolean;
```

C++

```
__property Boolean IsGroup;
```

Description

IsGroup indicates whether the dependency represents a load ordering group.

Note: : Because services and service groups share the same name space, group names must be prefixed by the SC_GROUP_IDENTIFIER character to distinguish them from service names. If IsGroup is set to true, SC_GROUP_IDENTIFIER will automatically be prefixed to the Dependency name (Name (see page 3068) property). DisplayName is not modified.

See Also

Name (see page 3068)

LoadGroup (see page 3080)

33.1.2.3.2 TDependency.Name Property

Identifies the dependent service or load ordering group used by a TService (see page 3071) object.

Pascal

```
property Name: String;
```

C++

```
__property UnicodeString Name;
```

Description

Name is the name identifying the service or load ordering group. If Name is the name of a service, Name reflects the actual ServiceStartName of a TService (see page 3071) instance.

The property editor for the Dependencies property of TService (see page 3071) technically uses the dependency's DisplayName to identify the dependency service in its collection list. However, Name and DisplayName are nearly always the same.

Dependency objects that are load ordering groups are prefixed by SC_GROUP_IDENTIFIER. Setting the IsGroup (see page 3067) property handles prefixing the Name automatically.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

DisplayName

LoadGroup (see page 3080)

TService (see page 3071)

33.1.3 TEventLogger Class

Class Hierarchy

Pascal

```
TEventLogger = class(TObject);
```

C++

```
class TEventLogger : public TObject;
```

File

SvcMgr

Description

This is class SvcMgr.TEventLogger.

33.1.3.1 TEventLogger Members

The following tables list the members exposed by TEventLogger.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3069)	This is Create, a member of class TEventLogger.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3070)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	LogMessage (see page 3070)	This is LogMessage, a member of class TEventLogger.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

33.1.3.2 TEventLogger.Create Constructor

Pascal

```
constructor Create(Name: String);
```

C++

```
__fastcall TEventLogger(UnicodeString Name);
```

Description

This is Create, a member of class TEventLogger.

33.1.3.3 TEventLogger.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TEventLogger();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

33.1.3.4 TEventLogger Methods

The methods of the TEventLogger class are listed here.

Public Methods

	Name	Description
	LogMessage (see page 3070)	This is LogMessage, a member of class TEventLogger.

33.1.3.4.1 TEventLogger.LogMessage Method

Pascal

```
procedure LogMessage(Message: String; EventType: DWord = 1; Category: Word = 0; ID: DWord = 0);
```

C++

```
__fastcall LogMessage(UnicodeString Message, DWord EventType = 1, Word Category = 0, DWord ID
= 0);
```

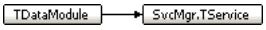
Description

This is LogMessage, a member of class TEventLogger.

33.1.4 TService Class

TService encapsulates a Windows NT service.

Class Hierarchy



Pascal

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TService = class(TDataModule);
```

C++

```
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TService : public TDataModule;
```

File

SvcMgr

Description

TService encapsulates a Windows NT service in an NT service application. A Win32 service is accessed via the Service Control Manager and can be started automatically at system boot, by a user through the Services control panel applet, or by a Win32-based application that uses the service functions. Services can execute even when no user is logged on to the system. Windows NT also supports a driver service that does not interact with the SCM.

See Also

TServiceApplication (see page 3089)

Service Applications

Service Name Properties

Service Threads

33.1.4.1 TService Members

The following tables list the members exposed by TService.

Protected Methods

	Name	Description
	Controller (see page 3074)	This is Controller, a member of class TService.
V	DoContinue (see page 3074)	This is DoContinue, a member of class TService.
V	DoCustomControl (see page 3074)	This is DoCustomControl, a member of class TService.
V	DoInterrogate (see page 3074)	This is DoInterrogate, a member of class TService.
V	DoPause (see page 3075)	This is DoPause, a member of class TService.
V	DoShutdown (see page 3075)	This is DoShutdown, a member of class TService.
V	DoStart (see page 3075)	This is DoStart, a member of class TService.
V	DoStop (see page 3075)	This is DoStop, a member of class TService.
	Main (see page 3077)	This is Main, a member of class TService.

Public Methods

	Name	Description
	CreateNew (see page 3073)	Instantiates and initializes a TService (see page 3071) object.
	Destroy (see page 3073)	Disposes of a TService (see page 3071) instance.
	GetServiceController (see page 3076)	Returns the service's registered handler function.
	LogMessage (see page 3076)	Sends an error message to the event log.
	ReportStatus (see page 3077)	Updates the Service Control Manager's status information for the calling service.

Public Properties

	Name	Description
	ErrCode (see page 3079)	Contains the service specific error code returned on starting or stopping the service.
	Param (see page 3081)	Lists the parameters to the service.
	ParamCount (see page 3081)	Indicates the number of startup parameters for the service.
	ServiceThread (see page 3082)	Identifies the dedicated thread for the service.
	Status (see page 3083)	Indicates the current state of the service.
	Terminated (see page 3084)	Indicates whether the service thread has terminated.
	Win32ErrCode (see page 3085)	Indicates the Win32 error code reported by the service if a service specific code is not defined.

Published Events

	Name	Description
	AfterInstall (see page 3085)	Occurs immediately after the service is registered with the Windows Service Control manager.
	AfterUninstall (see page 3086)	Occurs immediately after the service is removed from the Service Control manager's database.
	BeforeInstall (see page 3086)	Occurs before the service is first registered with the Windows Service Control manager.
	BeforeUninstall (see page 3087)	Occurs immediately before the service is removed from the Service Control manager's database.
	OnContinue (see page 3087)	Occurs when the Service Control manager resumes the service after it has been paused.
	OnExecute (see page 3088)	Occurs when the thread associated with the service starts up.
	OnPause (see page 3088)	Occurs when the Service Control manager pauses the service temporarily.
	OnShutdown (see page 3088)	Occurs when the system running the service application is about to shut down.
	OnStart (see page 3089)	OnStartup occurs when the service first starts up, before the OnExecute (see page 3088) event.
	OnStop (see page 3089)	Occurs when the Service Control manager stops the service.

Published Properties

	Name	Description
	AllowPause (see page 3078)	Indicates whether clients are allowed to pause the service.
	AllowStop (see page 3078)	Indicates whether clients are allowed to stop the service.
	Dependencies (see page 3078)	Lists the services or load ordering groups on which the service depends.
	DisplayName (see page 3079)	Identifies the service in the Windows Service Control Manager.
	ErrorSeverity (see page 3080)	Indicates the severity of service startup failure.
	Interactive (see page 3080)	Indicates whether the service is interactive with the Windows desktop.
	LoadGroup (see page 3080)	Names of the load ordering group that includes the service.
	Password (see page 3081)	Specifies the password, optionally used depending upon the ServiceStartName (see page 3082), for accessing the service.
	ServiceStartName (see page 3082)	ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.
	ServiceType (see page 3083)	Identifies the type of the service as Win32, device driver, or file system driver.
	StartType (see page 3083)	Indicates when or how the service starts.
	TagID (see page 3084)	Specifies a unique tag for the service within its LoadGroup (see page 3080).
	WaitHint (see page 3084)	Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

33.1.4.2 TService.CreateNew Constructor

Instantiates and initializes a TService (see page 3071) object.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TService(TComponent * AOwner, int Dummy = 0);
```

Description

Call CreateNew to instantiate a service object at runtime. Services added at design time are created automatically. The AOwner parameter specifies the TServiceApplication (see page 3089) object that is responsible for ultimately freeing the service. The Dummy parameter is not used. Use CreateNew rather than the Create constructor to instantiate a new service object at runtime. CreateNew does not attempt to stream in an associated DFM file that indicates any components added to the service at design time.

See Also

Owner

Create (see page 3064)

TServiceApplication (see page 3089)

33.1.4.3 TService.Destroy Destructor

Disposes of a TService (see page 3071) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TService();
```

Description

Destroy frees the memory associated with the objects for the Params and Dependencies (see page 3078) properties that were created in the constructor, and for the EventLogger object created in the LogMessage (see page 3076) method. Then it calls the inherited Destroy.

See Also

System::TObject::Free (see page 3372)

LogMessage (see page 3076)

33.1.4.4 TService Methods

The methods of the TService class are listed here.

Protected Methods

	Name	Description
	Controller (see page 3074)	This is Controller, a member of class TService.
	DoContinue (see page 3074)	This is DoContinue, a member of class TService.

	DoCustomControl (see page 3074)	This is DoCustomControl, a member of class TService.
	DoInterrogate (see page 3074)	This is DoInterrogate, a member of class TService.
	DoPause (see page 3075)	This is DoPause, a member of class TService.
	DoShutdown (see page 3075)	This is DoShutdown, a member of class TService.
	DoStart (see page 3075)	This is DoStart, a member of class TService.
	DoStop (see page 3075)	This is DoStop, a member of class TService.
	Main (see page 3077)	This is Main, a member of class TService.

Public Methods

	Name	Description
	GetServiceController (see page 3076)	Returns the service's registered handler function.
	LogMessage (see page 3076)	Sends an error message to the event log.
	ReportStatus (see page 3077)	Updates the Service Control Manager's status information for the calling service.

33.1.4.4.1 TService.Controller Method

Pascal

```
procedure Controller(CtrlCode: DWord);
```

C++

```
__fastcall Controller(DWord CtrlCode);
```

Description

This is Controller, a member of class TService.

33.1.4.4.2 TService.DoContinue Method

Pascal

```
function DoContinue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoContinue();
```

Description

This is DoContinue, a member of class TService.

33.1.4.4.3 TService.DoCustomControl Method

Pascal

```
function DoCustomControl(CtrlCode: DWord): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoCustomControl(DWord CtrlCode);
```

Description

This is DoCustomControl, a member of class TService.

33.1.4.4.4 TService.DoInterrogate Method

Pascal

```
procedure DoInterrogate; virtual;
```

C++

```
virtual __fastcall DoInterrogate();
```

Description

This is DoInterrogate, a member of class TService.

33.1.4.4.5 TService.DoPause Method

Pascal

```
function DoPause: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoPause();
```

Description

This is DoPause, a member of class TService.

33.1.4.4.6 TService.DoShutdown Method

Pascal

```
procedure DoShutdown; virtual;
```

C++

```
virtual __fastcall DoShutdown();
```

Description

This is DoShutdown, a member of class TService.

33.1.4.4.7 TService.DoStart Method

Pascal

```
procedure DoStart; virtual;
```

C++

```
virtual __fastcall DoStart();
```

Description

This is DoStart, a member of class TService.

33.1.4.4.8 TService.DoStop Method

Pascal

```
function DoStop: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoStop();
```

Description

This is DoStop, a member of class TService.

33.1.4.4.9 TService.GetServiceController Method

Returns the service's registered handler function.

Pascal

```
function GetServiceController: TServiceController; virtual; abstract;
```

C++

```
virtual __fastcall TServiceController GetServiceController() = 0;
```

Description

GetServiceController returns the handler for the service. You do not need to call this function directly. The service object's main function calls it automatically.

Note: Each service in the service application has a main function for that service. When a request is made to start the service, the main thread of the service calls the RegisterServiceCtrlHandler function to register this control handler function. TService (see page 3071) automatically calls RegisterServiceCtrlHandler and returns this handler function. Because this function cannot be a class method (member function), the Service application wizard generates code that creates a non-member handler, (ServiceController) for your service object. This routine returns the service object's Handler (Controller (see page 3074)).

33.1.4.4.10 TService.LogMessage Method

Sends an error message to the event log.

Pascal

```
procedure LogMessage(Message: String; EventType: DWord = 1; Category: Integer = 0; ID: Integer
= 0);
```

C++

```
__fastcall LogMessage(UnicodeString Message, DWord EventType = 1, int Category = 0, int ID =
0);
```

Description

LogMessage sends an error message to the event log when an error or exception occurs. By default the EventType is EVENTLOG_ERROR_TYPE (usually indicating a loss of functionality or data) and the category and ID are zero. For example, if a service cannot be loaded as the system boots, it can log an error event, and the category and ID are zero.

The EventType specifies the type of event being logged. This parameter can be one of the following values.

Value	Meaning
EVENTLOG_WARNING_TYPE	Warning event
EVENTLOG_INFORMATION_TYPE	Information event
EVENTLOG_AUDIT_SUCCESS	Success Audit event
EVENTLOG_AUDIT_FAILURE	Failure Audit event
EVENTLOG_ERROR_TYPE	Error (see page 3560) event

The Category parameter specifies the event category, which is source-specific information and can have any value.

The ID parameter specifies the event identifier, which is the message that goes with this event as an entry in the message file associated with the event source.

33.1.4.4.11 TService.Main Method

Pascal

```
procedure Main(Argc: DWord; Argv: PLPSTR);
```

C++

```
__fastcall Main(DWord Argc, PLPSTR Argv);
```

Description

This is Main, a member of class TService.

33.1.4.4.12 TService.ReportStatus Method

Updates the Service Control Manager's status information for the calling service.

Pascal

```
procedure ReportStatus;
```

C++

```
__fastcall ReportStatus();
```

Description

ReportStatus sends current status information about the calling service to the Service Control Manager.

See Also

LogMessage (see page 3076)

Status (see page 3083)

WaitHint (see page 3084)

33.1.4.5 TService Properties

The properties of the TService class are listed here.

Public Properties

	Name	Description
	ErrCode (see page 3079)	Contains the service specific error code returned on starting or stopping the service.
R	Param (see page 3081)	Lists the parameters to the service.
R	ParamCount (see page 3081)	Indicates the number of startup parameters for the service.
R	ServiceThread (see page 3082)	Identifies the dedicated thread for the service.
	Status (see page 3083)	Indicates the current state of the service.
R	Terminated (see page 3084)	Indicates whether the service thread has terminated.
	Win32ErrCode (see page 3085)	Indicates the Win32 error code reported by the service if a service specific code is not defined.

Published Properties

	Name	Description
	AllowPause (see page 3078)	Indicates whether clients are allowed to pause the service.
	AllowStop (see page 3078)	Indicates whether clients are allowed to stop the service.
	Dependencies (see page 3078)	Lists the services or load ordering groups on which the service depends.
	DisplayName (see page 3079)	Identifies the service in the Windows Service Control Manager.

	ErrorSeverity (see page 3080)	Indicates the severity of service startup failure.
	Interactive (see page 3080)	Indicates whether the service is interactive with the Windows desktop.
	LoadGroup (see page 3080)	Names of the load ordering group that includes the service.
	Password (see page 3081)	Specifies the password, optionally used depending upon the ServiceStartName (see page 3082), for accessing the service.
	ServiceStartName (see page 3082)	ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.
	ServiceType (see page 3083)	Identifies the type of the service as Win32, device driver, or file system driver.
	StartType (see page 3083)	Indicates when or how the service starts.
	TagID (see page 3084)	Specifies a unique tag for the service within its LoadGroup (see page 3080).
	WaitHint (see page 3084)	Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

33.1.4.5.1 TService.AllowPause Property

Indicates whether clients are allowed to pause the service.

Pascal

```
property AllowPause: Boolean;
```

C++

```
__property Boolean AllowPause;
```

Description

AllowPause indicates whether clients of the service can pause and resume the service by calling the Windows API ControlService function. When AllowPause is true, an OnPause (see page 3088) event occurs before the service pauses, and an OnContinue (see page 3087) event occurs when the service resumes.

See Also

OnPause (see page 3088)

OnContinue (see page 3087)

33.1.4.5.2 TService.AllowStop Property

Indicates whether clients are allowed to stop the service.

Pascal

```
property AllowStop: Boolean;
```

C++

```
__property Boolean AllowStop;
```

Description

AllowStop indicates whether clients of the service can stop it by calling the Windows API ControlService function. When AllowStop is true, an OnStop (see page 3089) event occurs before the service stops.

See Also

OnStop (see page 3089)

33.1.4.5.3 TService.Dependencies Property

Lists the services or load ordering groups on which the service depends.

Pascal

```
property Dependencies: TDependencies;
```

C++

```
__property TDependencies Dependencies;
```

Description

Dependencies is a container for TDependency (see page 3066) objects, which are dependent services or load ordering groups that must start before this service. If the container is empty, the service has no dependencies.

Dependency on a service means that this service can only run if the service it depends on is running. Dependency on a group means that this service can run if at least one member of the group is running after an attempt to start all members of the group.

See Also

TDependency (see page 3066)

LoadGroup (see page 3080)

33.1.4.5.4 TService.DisplayName Property

Identifies the service in the Windows Service Control Manager.

Pascal

```
property DisplayName: String;
```

C++

```
__property UnicodeString DisplayName;
```

Description

DisplayName contains a string, with a maximum length of 256 characters, that is the name of the service as displayed in the Service Control Manager. User interface programs use DisplayName to identify the service. DisplayName is case-preserved in the Service Control Manager, however display name comparisons are always case-insensitive.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Name

ServiceStartName (see page 3082)

33.1.4.5.5 TService.ErrCode Property

Contains the service specific error code returned on starting or stopping the service.

Pascal

```
property ErrCode: DWord;
```

C++

```
__property DWord ErrCode;
```

Description

ErrCode specifies a service specific error code that the service returns when an error occurs while the service is starting or stopping. If ErrCode is not defined, the Win32ErrCode (see page 3085) property is used instead.

See Also

ReportStatus (see page 3077)

Win32ErrCode (see page 3085)

33.1.4.5.6 TService.ErrorSeverity Property

Indicates the severity of service startup failure.

Pascal

```
property ErrorSeverity: TErrorSeverity;
```

C++

```
__property TErrorSeverity ErrorSeverity;
```

Description

ErrorSeverity indicates the severity if this service fails to start during startup of the service application.

See Also

ErrCode (see page 3079)

ReportStatus (see page 3077)

Win32ErrCode (see page 3085)

33.1.4.5.7 TService.Interactive Property

Indicates whether the service is interactive with the Windows desktop.

Pascal

```
property Interactive: Boolean;
```

C++

```
__property Boolean Interactive;
```

Description

Interactive indicates whether the service can interact with the Windows desktop. Interactive only has an effect when ServiceType (see page 3083) is stWin32. For more information about creating interactive NT services, see the Microsoft Help.

See Also

ServiceType (see page 3083)

33.1.4.5.8 TService.LoadGroup Property

Names of the load ordering group that includes the service.

Pascal

```
property LoadGroup: String;
```

C++

```
__property UnicodeString LoadGroup;
```

Description

LoadGroup is the name of the load-ordering group that includes the service. This is used by other services that have

dependencies on the load ordering group (See Dependencies (see page 3078)). The registry lists load-ordering groups at HKEY_LOCAL_MACHINE\System (see page 3159)\CurrentControlSet\Control\ServiceGroupOrder. The order in which services are loaded depends on the order of the load groups in the Registry and the dependencies listed in service Dependencies (see page 3078) properties.

Because services and service groups share the same name space, the SC_GROUP_IDENTIFIER character must prefix group names to distinguish them from service names. This is automatically prefixed if the TDependency (see page 3066)'s IsGroup property is true.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Dependencies (see page 3078)

TagID (see page 3084)

33.1.4.5.9 TService.Param Property

Lists the parameters to the service.

Pascal

```
property Param [Index: Integer]: String;
```

C++

```
__property UnicodeString Param[int Index];
```

Description

Param is an indexed property that contains the parameters to the service. These parameters are typed into the Startup Parameters input window in the Service Control Manager.

See Also

ParamCount (see page 3081)

33.1.4.5.10 TService.ParamCount Property

Indicates the number of startup parameters for the service.

Pascal

```
property ParamCount: Integer;
```

C++

```
__property int ParamCount;
```

Description

ParamCount is the number of parameters for the service. These parameters are typed into the Startup Parameters input window in the Service Control Manager.

See Also

Param (see page 3081)

33.1.4.5.11 TService.Password Property

Specifies the password, optionally used depending upon the ServiceStartName (see page 3082), for accessing the service.

Pascal

```
property Password: String;
```

C++

```
__property UnicodeString Password;
```

Description

Password is the password for the account name specified by the ServiceStartName (see page 3082) property if the ServiceType (see page 3083) is stWin32.

If Password is nil (Delphi) or NULL (C++), or if it points to an empty string, the service has no password. If ServiceType (see page 3083) is stDevice or stFileSystem, this parameter is ignored.

See Also

ServiceStartName (see page 3082)

33.1.4.5.12 TService.ServiceStartName Property

ServiceStartName is the name used in the Service Control Manager Startup dialog to start the service.

Pascal

```
property ServiceStartName: String;
```

C++

```
__property UnicodeString ServiceStartName;
```

Description

ServiceStartName is the name used to start the service.

If ServiceType (see page 3083) is stWin32, ServiceStartName is the account name (in the form "DomainName\Username") which the service process is logged on as when it runs. If the account belongs to the built-in domain, ".\Username" can be specified. If the service application includes more than one service, leave ServiceStartName blank, and leave the Password (see page 3081) property blank.

If the service type is not stWin32, ServiceStartName is the driver object name (\FileSytems\Rdr or \Driver\Xns) which the input and output system uses to load the device driver. With device drivers or file systems drivers, you can leave ServiceStartName blank and the I/O system will create a default name based on the service name.

Note: A dependency can be a service. If this is true, do not confuse the various name-related properties of dependency and service objects. For more information, see Service name properties.

See Also

Password (see page 3081)

33.1.4.5.13 TService.ServiceThread Property

Identifies the dedicated thread for the service.

Pascal

```
property ServiceThread: TServiceThread;
```

C++

```
__property TServiceThread ServiceThread;
```

Description

ServiceThread is the instance of the TServiceThread (see page 3095) class that is the dedicated thread for the service. Each service in the service application has a dedicated thread that contains a loop in its Execute method for processing service requests. Often a service will spawn a new thread, derived from TThread, to free the service thread to continue processing service requests.

See Also

ProcessRequests (see page 3100)

TThread

33.1.4.5.14 TService.ServiceType Property

Identifies the type of the service as Win32, device driver, or file system driver.

Pascal

```
property ServiceType: TServiceType;
```

C++

```
__property TServiceType ServiceType;
```

Description

ServiceType indicates the type of service. ServiceType can be stWin32 (a Win32 service – the default), stDevice (a device driver), or stFileSystem (a file system driver).

See Also

Interactive (see page 3080)

33.1.4.5.15 TService.StartType Property

Indicates when or how the service starts.

Pascal

```
property StartType: TStartType;
```

C++

```
__property TStartType StartType;
```

Description

StartType specifies how and why the service starts

33.1.4.5.16 TService.Status Property

Indicates the current state of the service.

Pascal

```
property Status: TCurrentStatus;
```

C++

```
__property TCurrentStatus Status;
```

Description

Status indicates the current state of the service. The status information is displayed in the Service Control Manager.

See Also

ReportStatus (see page 3077)

33.1.4.5.17 TService.TagID Property

Specifies a unique tag for the service within its LoadGroup (see page 3080).

Pascal

```
property TagID: DWord;
```

C++

```
__property DWord TagID;
```

Description

TagID is the unique tag value for this service in its LoadGroup (see page 3080). If no tag is requested, this property is 0. Tags are only evaluated when the ServiceType (see page 3083) is not stWin32 and the StartType (see page 3083) is stBoot or stSystem.

See Also

Dependencies (see page 3078)

LoadGroup (see page 3080)

33.1.4.5.18 TService.Terminated Property

Indicates whether the service thread has terminated.

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

Terminated indicates whether the dedicated thread that the service is running in has terminated.

See Also

Terminated

Terminate

33.1.4.5.19 TService.WaitHint Property

Specifies the estimated time, in milliseconds, between a call from the Windows Service Control Manager and the service reporting its status.

Pascal

```
property WaitHint: Integer;
```

C++

```
__property int WaitHint;
```

Description

WaitHint specifies the amount of time, in milliseconds, that can elapse between a pending start, stop, pause, or continue

operation and the service setting its status with a call to the Windows API. Normally this is handled automatically. If the service performs a lengthy operation of more than WaitHint time in one of its event handlers, you should call ReportStatus (see page 3077) method to update to the Service Control Manager.

See Also

ReportStatus (see page 3077)

33.1.4.5.20 TService.Win32ErrCode Property

Indicates the Win32 error code reported by the service if a service specific code is not defined.

Pascal

```
property Win32ErrCode: DWord;
```

C++

```
__property DWord Win32ErrCode;
```

Description

Win32ErrCode specifies a Win32 error code that the service uses to report an error that occurs when it is starting or stopping. If an error code specific to the service is defined, this property is automatically set to ERROR_SERVICE_SPECIFIC_ERROR and the service specific error is reported instead.

See Also

ErrCode (see page 3079)

ReportStatus (see page 3077)

33.1.4.6 TService Events

The events of the TService class are listed here.

Published Events

	Name	Description
	AfterInstall (see page 3085)	Occurs immediately after the service is registered with the Windows Service Control manager.
	AfterUninstall (see page 3086)	Occurs immediately after the service is removed from the Service Control manager's database.
	BeforeInstall (see page 3086)	Occurs before the service is first registered with the Windows Service Control manager.
	BeforeUninstall (see page 3087)	Occurs immediately before the service is removed from the Service Control manager's database.
	OnContinue (see page 3087)	Occurs when the Service Control manager resumes the service after it has been paused.
	OnExecute (see page 3088)	Occurs when the thread associated with the service starts up.
	OnPause (see page 3088)	Occurs when the Service Control manager pauses the service temporarily.
	OnShutdown (see page 3088)	Occurs when the system running the service application is about to shut down.
	OnStart (see page 3089)	OnStartup occurs when the service first starts up, before the OnExecute (see page 3088) event.
	OnStop (see page 3089)	Occurs when the Service Control manager stops the service.

33.1.4.6.1 TService.AfterInstall Event

Occurs immediately after the service is registered with the Windows Service Control manager.

Pascal

```
property AfterInstall: TServiceEvent;
```

C++

```
__property TServiceEvent AfterInstall;
```

Description

Write (see page 3707) an AfterInstall event handler to take specific action immediately after the service is registered. Install services by running the service application with the INSTALL switch.

See Also

AfterUninstall (see page 3086)

OnStart (see page 3089)

OnExecute (see page 3088)

OnCreate

BeforeInstall (see page 3086)

33.1.4.6.2 TService.AfterUninstall Event

Occurs immediately after the service is removed from the Service Control manager's database.

Pascal

```
property AfterUninstall: TServiceEvent;
```

C++

```
__property TServiceEvent AfterUninstall;
```

Description

Write (see page 3707) an AfterUninstall event handler to take specific action immediately after the service is removed. Uninstall services by running the service application with the UNINSTALL switch.

See Also

AfterInstall (see page 3085)

OnStop (see page 3089)

BeforeUninstall (see page 3087)

Destroy

OnShutdown (see page 3088)

33.1.4.6.3 TService.BeforeInstall Event

Occurs before the service is first registered with the Windows Service Control manager.

Pascal

```
property BeforeInstall: TServiceEvent;
```

C++

```
__property TServiceEvent BeforeInstall;
```

Description

Write (see page 3707) a BeforeInstall event handler to take specific action immediately before the service is first registered. Install services by running the service application with the INSTALL switch.

See Also

AfterInstall (see page 3085)

OnStart (see page 3089)

OnExecute (see page 3088)

OnCreate

BeforeUninstall (see page 3087)

33.1.4.6.4 TService.BeforeUninstall Event

Occurs immediately before the service is removed from the Service Control manager's database.

Pascal

```
property BeforeUninstall: TServiceEvent;
```

C++

```
__property TServiceEvent BeforeUninstall;
```

Description

Write (see page 3707) a BeforeUninstall event handler to take specific action immediately before the service is removed. Uninstall services by running the service application with the UNINSTALL switch.

See Also

AfterUninstall (see page 3086)

OnStop (see page 3089)

BeforeInstall (see page 3086)

Destroy

OnShutdown (see page 3088)

33.1.4.6.5 TService.OnContinue Event

Occurs when the Service Control manager resumes the service after it has been paused.

Pascal

```
property OnContinue: TContinueEvent;
```

C++

```
__property TContinueEvent OnContinue;
```

Description

Write (see page 3707) an OnContinue event handler to take specific action when the Service Control manager resumes the service after it has been paused.

See Also

OnPause (see page 3088)

33.1.4.6.6 TService.OnExecute Event

Occurs when the thread associated with the service starts up.

Pascal

```
property OnExecute: TServiceEvent;
```

C++

```
__property TServiceEvent OnExecute;
```

Description

If you are not spawning a new thread to handle individual service requests in an OnStart (see page 3089) event handler, this is where you implement the service. When the OnExecute event handler finishes, the service thread terminates. Most OnExecute event handlers contain a loop that calls the service thread's ProcessRequests method so that other service requests are not locked out.

See Also

ProcessRequests (see page 3100)

OnStart (see page 3089)

33.1.4.6.7 TService.OnPause Event

Occurs when the Service Control manager pauses the service temporarily.

Pascal

```
property OnPause: TPauseEvent;
```

C++

```
__property TPauseEvent OnPause;
```

Description

Write (see page 3707) an OnPause event handler to take specific action when the service is temporarily paused.

See Also

OnContinue (see page 3087)

33.1.4.6.8 TService.OnShutdown Event

Occurs when the system running the service application is about to shut down.

Pascal

```
property OnShutdown: TServiceEvent;
```

C++

```
__property TServiceEvent OnShutdown;
```

Description

Write (see page 3707) an OnShutdown event handler to take specific action just before the service shuts down. The service application terminates immediately after OnShutdown.

See Also

OnStop (see page 3089)

OnStart (see page 3089)

OnPause (see page 3088)

33.1.4.6.9 TService.OnStart Event

OnStartup occurs when the service first starts up, before the OnExecute (see page 3088) event.

Pascal

```
property OnStart: TStartEvent;
```

C++

```
__property TStartEvent OnStart;
```

Description

This event should be used to initialize the service. For example, if each service request is handled in a separate thread (a good idea if handling the request takes much time) the thread for a request is spawned in an OnStart event handler.

See Also

OnStop (see page 3089)

OnContinue (see page 3087)

OnExecute (see page 3088)

33.1.4.6.10 TService.OnStop Event

Occurs when the Service Control manager stops the service.

Pascal

```
property OnStop: TStopEvent;
```

C++

```
__property TStopEvent OnStop;
```

Description

Write (see page 3707) an OnStop event handler to take specific action when the service stops.

See Also

OnShutdown (see page 3088)

OnStart (see page 3089)

OnPause (see page 3088)

33.1.5 TServiceApplication Class

TServiceApplication encapsulates a Windows NT service application.

Class Hierarchy

Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
TServiceApplication = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
[SecurityPermission(SecurityAction.Demand, Unrestricted=True)]
class TServiceApplication : public TComponent;
```

File

SvcMgr

Description

TServiceApplication encapsulates a Windows NT application type known as a service. TServiceApplication provides the fundamental behavior of this type of application. Each Service project automatically declares an Application (see page 3109) variable of type TServiceApplication as the instance of the application. The Application (see page 3109) variable is added to a project by selecting File | New (see page 3619) from the menu of the IDE main window and selecting Service Application (see page 3109) in the New (see page 3619) Items dialog.

TServiceApplication contains TService (see page 3071) objects, each of which encapsulates a Windows NT service. The service application object provides methods to create service objects and to install, register, dispatch, and uninstall services.

When the service application is run using the /INSTALL switch, it registers the services it contains. When run using the /UNINSTALL switch, it unregisters its services. Unless the application is run with the /SILENT switch, the service application shows a confirmation message after successfully installing or uninstalling its services.

Note: Do not add the Forms or Httpapp units to the uses list of the project (Delphi) or includ the unit header for these units in the project source (C++). Those units declare conflicting Application (see page 3109) variables of types TApplication and TWebApplication, respectively.

See Also

Application (see page 3109)

TService (see page 3071)

Service Applications

Service Name\ Properties

33.1.5.1 TServiceApplication Members

The following tables list the members exposed by TServiceApplication.

Protected Methods

	Name	Description
	DispatchServiceMain (see page 3092)	This is DispatchServiceMain, a member of class TServiceApplication.
	DoHandleException (see page 3092)	This is DoHandleException, a member of class TServiceApplication.
	Hook (see page 3093)	This is Hook, a member of class TServiceApplication.
	RegisterServices (see page 3093)	This is RegisterServices, a member of class TServiceApplication.

Public Methods

	Name	Description
	Create (see page 3091)	Instantiates a TServiceApplication (see page 3089) object.
	CreateForm (see page 3092)	Creates the service object or objects for the service application.

	Destroy (see page 3091)	Disposes of an instance of a TServiceApplication (see page 3089) object.
	Initialize (see page 3093)	Sets the initial state of the application and hides any associated MainForm.
	Installing (see page 3093)	This is Installing, a member of class TServiceApplication.
	Run (see page 3094)	Installs, registers, and adds services to its component list, and creates a thread for the service application.

Public Properties

	Name	Description
	DelayInitialize (see page 3094)	This is DelayInitialize, a member of class TServiceApplication.
	ServiceCount (see page 3094)	Indicates the number of services in the service application.
	Title (see page 3095)	Indicates the name of the title application.

33.1.5.2 TServiceApplication.Create Constructor

Instantiates a TServiceApplication (see page 3089) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TServiceApplication(TComponent * AOwner);
```

Description

Applications do not need to call the constructor directly. TServiceApplication (see page 3089) is instantiated by selecting File | New (see page 3619) from the menu of the IDE main window and selecting Service Application (see page 3109) in the New (see page 3619) Items dialog. An Application (see page 3109) variable of type TServiceApplication (see page 3089) is declared and initialized as the instance of the application.

33.1.5.3 TServiceApplication.Destroy Destructor

Disposes of an instance of a TServiceApplication (see page 3089) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TServiceApplication();
```

Description

You do not need to call the destructor directly. TServiceApplication (see page 3089) is destroyed when your application closes.

The destructor frees the memory associated with sub-objects.

See Also

System::TObject::Free (see page 3372)

33.1.5.4 TServiceApplication Methods

The methods of the TServiceApplication class are listed here.

Protected Methods

	Name	Description
	DispatchServiceMain (see page 3092)	This is DispatchServiceMain, a member of class TServiceApplication.
	DoHandleException (see page 3092)	This is DoHandleException, a member of class TServiceApplication.
	Hook (see page 3093)	This is Hook, a member of class TServiceApplication.
	RegisterServices (see page 3093)	This is RegisterServices, a member of class TServiceApplication.

Public Methods

	Name	Description
	CreateForm (see page 3092)	Creates the service object or objects for the service application.
	Initialize (see page 3093)	Sets the initial state of the application and hides any associated MainForm.
	Installing (see page 3093)	This is Installing, a member of class TServiceApplication.
	Run (see page 3094)	Installs, registers, and adds services to its component list, and creates a thread for the service application.

33.1.5.4.1 TServiceApplication.CreateForm Method

Creates the service object or objects for the service application.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

C++

```
virtual __fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Do not call CreateForm in an application. The module manager calls CreateForm internally to register the service objects of the service application.

CreateForm instantiates a TService (see page 3071) object for each service contained in the service application. CreateForm is called automatically, for the first service, when the service application is first created. Subsequently, CreateForm is called for each new service added to the application. To add a new service, select File | New (see page 3619) from the menu of the IDE main window and choose Service in the New (see page 3619) Items dialog.

33.1.5.4.2 TServiceApplication.DispatchServiceMain Method

Pascal

```
procedure DispatchServiceMain(Argc: DWord; Argv: PLPSTR);
```

C++

```
__fastcall DispatchServiceMain(DWord Argc, PLPSTR Argv);
```

Description

This is DispatchServiceMain, a member of class TServiceApplication.

33.1.5.4.3 TServiceApplication.DoHandleException Method

Pascal

```
procedure DoHandleException(E: Exception); dynamic;
```

C++

```
__fastcall DoHandleException(Exception E);
```

Description

This is DoHandleException, a member of class TServiceApplication.

33.1.5.4.4 TServiceApplication.Hook Method

Pascal

```
function Hook(var Message: TMessage): Boolean;
```

C++

```
__fastcall Boolean Hook(TMessage Message);
```

Description

This is Hook, a member of class TServiceApplication.

33.1.5.4.5 TServiceApplication.Initialize Method

Sets the initial state of the application and hides any associated MainForm.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

Initialize sets the initial state of the application to hide any main form that may be associated with the application. Service applications are rarely interactive and do not typically have a form associated with them. If a form has been added to the application, Initialize hides it on start. Initialize then follows the standard (default) VCL application initialization process.

33.1.5.4.6 TServiceApplication.Installing Method

Pascal

```
function Installing: Boolean;
```

C++

```
__fastcall Boolean Installing();
```

Description

This is Installing, a member of class TServiceApplication.

33.1.5.4.7 TServiceApplication.RegisterServices Method

Pascal

```
procedure RegisterServices(Install: Boolean; Silent: Boolean);
```

C++

```
__fastcall RegisterServices(Boolean Install, Boolean Silent);
```

Description

This is RegisterServices, a member of class TServiceApplication.

33.1.5.4.8 TServiceApplication.Run Method

Installs, registers, and adds services to its component list, and creates a thread for the service application.

Pascal

```
procedure Run; virtual;
```

C++

```
virtual __fastcall Run();
```

Description

Run handles installing and uninstalling services, depending on the command line switches. When the service application is run using the /INSTALL switch, it registers the services it contains. When run using the /UNINSTALL switch, it unregisters its services. Unless the application is run with the /SILENT switch, the service application shows a confirmation message after successfully installing or uninstalling its services.

If not called with the /INSTALL or /UNINSTALL switch, Run creates a service application thread that responds to Windows messages from clients. Note that this thread is for the application and differs from the dedicated TServiceThread (see page 3095) instances associated with individual services in the application.

33.1.5.5 TServiceApplication Properties

The properties of the TServiceApplication class are listed here.

Public Properties

	Name	Description
	DelayInitialize (see page 3094)	This is DelayInitialize, a member of class TServiceApplication.
R	ServiceCount (see page 3094)	Indicates the number of services in the service application.
	Title (see page 3095)	Indicates the name of the title application.

33.1.5.5.1 TServiceApplication.DelayInitialize Property

Pascal

```
property DelayInitialize: Boolean;
```

C++

```
__property Boolean DelayInitialize;
```

Description

This is DelayInitialize, a member of class TServiceApplication.

33.1.5.5.2 TServiceApplication.ServiceCount Property

Indicates the number of services in the service application.

Pascal

```
property ServiceCount: Integer;
```

C++

```
__property int ServiceCount;
```

Description

ServiceCount is the number of services in the service application. Each service is encapsulated by a TService (see page 3071) instance and has its own dedicated service thread.

Note: If the Service application requires a user name and password, the application must implement only a single service.

See Also

TService (see page 3071)

33.1.5.5.3 TServiceApplication.Title Property

Indicates the name of the title application.

Pascal

```
property Title: string;
```

C++

```
__property UnicodeString Title;
```

Description

Title is the name of the service application. This text string appears below the icon of the application when it is minimized.

33.1.6 TServiceThread Class

TServiceThread represents the thread associated with a specific service (TService (see page 3071) object).

Class Hierarchy

Pascal

```
TServiceThread = class(TThread);
```

C++

```
class TServiceThread : public TThread;
```

File

SvcMgr

Description

TServiceThread is the thread associated with a service. Each service application has one or more service objects, each of which has its own dedicated thread. TServiceThread instances represent those threads.

Note: Service threads.

See Also

TService (see page 3071)

TServiceApplication (see page 3089)

Service Threads

33.1.6.1 TServiceThread Members

The following tables list the members exposed by TServiceThread.

Protected Methods

	Name	Description
	Execute (see page 3097)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Methods

	Name	Description
	Create (see page 3096)	Instantiates and initializes a TServiceThread (see page 3095) object.
	ProcessRequests (see page 3100)	Allows clients to invoke the service associated with the service thread.

33.1.6.2 TServiceThread.Create Constructor

Instantiates and initializes a TServiceThread (see page 3095) object.

Pascal

```
constructor Create(Service: TService);
```

C++

```
__fastcall TServiceThread(TService Service);
```

Description

Applications do not need to call the constructor directly. Instances of TService (see page 3071) create the TServiceThread (see page 3095) object on startup. The Service parameter is the TService (see page 3071) instance for which the TServiceThread (see page 3095) is the dedicated thread.

The constructor sets the FreeOnTerminate property to true and then calls the base class constructor so that the thread object is created with its Suspended property set to false.

See Also

FreeOnTerminate

Suspended

33.1.6.3 TServiceThread Methods

The methods of the TServiceThread class are listed here.

Protected Methods

	Name	Description
	Execute (see page 3097)	Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Public Methods

	Name	Description
	ProcessRequests (see page 3100)	Allows clients to invoke the service associated with the service thread.

33.1.6.3.1 TServiceThread.Execute Method

Provides an abstract or pure virtual method to contain the code which executes when the thread is run.

Pascal

```
procedure Execute; override;
```

C++

```
virtual __fastcall Execute();
```

Description

Override Execute and insert the code that should be executed when the thread runs. Execute is responsible for checking the value of the Terminated property to determine if the thread needs to exit.

A thread executes when Create is called if CreateSuspended set to false, or when Resume is first called after the thread is created if CreateSuspended set to true.

Note: Do not use the properties and methods of other objects directly in the Execute method of a thread. Instead, separate the use of other objects into a separate procedure call, and call that procedure by passing it as a parameter to the Synchronize method.

C++ Examples:

```
/*
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes. To
create the extension choose the
File | New | Other | Thread Object menu.
*/
bool mythreadRunning;
bool youthreadRunning;

class TMyThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TMyThread(bool suspended);
};

__fastcall TMyThread::TMyThread(bool suspended)
    : TThread(suspended)
{
}

class TYouThread : public TThread
{
__published:    // IDE-managed Components
private:    // User declarations
protected:  // User declarations
    void __fastcall Execute();
public:     // User declarations
    __fastcall TYouThread(bool suspended);
};

__fastcall TYouThread::TYouThread(bool suspended)
```

```
    : TThread(suspended)
{
}

void __fastcall TMyThread::Execute()
{
  Form1->Memo1->Lines->Add("Process has been running for this many seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo1->Lines->Add("Low priority process " + IntToStr(I));
    Sleep(1000);
  }
  mythreadRunning = FALSE;
}

void __fastcall TYouThread::Execute()
{
  Form1->Memo3->Lines->Add("Second low priority process has been running for this many
seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo3->Lines->Add("Second low priority process " + IntToStr(I));
    Sleep(1000);
  }
  youthreadRunning = FALSE;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMyThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (mythreadRunning == FALSE)
  {
    mythreadRunning = TRUE;
    SecondProcess = new TMyThread(True); // create suspended - secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  Form1->Memo2->Lines->Add("Do some work in the main process for 10 seconds:");
  for (int I = 0; I <= 10; I++)
  {
    Form1->Memo2->Lines->Add("Main process " + IntToStr(I));
    Sleep(1000);
  }
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  TYouThread *SecondProcess; // TMyThread is a custom descendant of TThread
  if (youthreadRunning == FALSE)
  {
    youthreadRunning = TRUE;
    SecondProcess = new TYouThread(True); // create suspended - secondprocess does not run yet
    SecondProcess->FreeOnTerminate = True; // don't need to cleanup after terminate
    SecondProcess->Priority = tpLower;  // set the priority to lower than normal
    SecondProcess->Resume(); // now run the thread
  }
  else
    MessageDlg("This thread is still running.  You are going to hurt yourself!",
```

```
      mtInformation, TMsgDlgButtons() << mbOK, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  mythreadRunning = FALSE;
  youthreadRunning = FALSE;
}
```

Delphi Examples:

```
{
This example shows how to create a thread and start it
running at a lower priority than the main execution thread.
Set the thread's FreeOnTerminate property is True, so that
there is no need to free the thread when it finishes.  The
Execute procedure must be overridden in the extension class
of TThread or an "Abstract Error" will result. To create the
extension choose the File | New | Other | Thread Object menu.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (mythreadRunning = FALSE) then
  begin
    mythreadRunning:= TRUE;
    MyProcess := TMyThread.Create(True); { create suspended - secondprocess does not run yet }
    MyProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    MyProcess.Priority := tpLower;  // set the priority to lower than normal
    MyProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  if mythreadRunning then MyProcess.Terminate();
end;

procedure TMyThread.Execute;
var I: Integer;
begin
  Form1.Memo1.Lines.Add('Process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo1.Lines.Add('Low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
  mythreadRunning:= FALSE;
end;

procedure TYouThread.Execute;
var I: Integer;
begin
  Form1.Memo3.Lines.Add('Second low priority process has been running for this many seconds:');
  for I := 0 to 10 do
  begin
    if Terminated then break;
    Form1.Memo3.Lines.Add('Second low priority process ' + InttoStr(I));
    Sleep(1000);
  end;
    youthreadRunning:= FALSE;
end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
var   I: Integer;
begin
  Form1.Memo2.Lines.Add('Do some work in the main process for 10 seconds:');
  for I := 0 to 10 do
  begin
    Form1.Memo2.Lines.Add('Main process ' + InttoStr(I));
    Sleep(1000);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if (youthreadRunning = FALSE) then
  begin
    youthreadRunning:= TRUE;
    YouProcess := TYouThread.Create(True); { create suspended – secondprocess does not run yet
}
    YouProcess.FreeOnTerminate := True; { don't need to cleanup after terminate }
    YouProcess.Priority := tpLower;  // set the priority to lower than normal
    YouProcess.Resume; { now run the thread }
  end
  else
    MessageDlg('This thread is still running.  You are going to hurt yourself!',
      mtInformation, [mbOk], 0);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if youthreadRunning then YouProcess.Terminate();
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  mythreadRunning:= FALSE;
  youthreadRunning:= FALSE;
end;
```

See Also

Create

Terminated

Synchronize

Checking for Termination by Other Threads

Waiting for a Thread to Finish Executing

Writing Clean-up Code

Writing the Thread Function

33.1.6.3.2 TServiceThread.ProcessRequests Method

Allows clients to invoke the service associated with the service thread.

Pascal

```
procedure ProcessRequests(WaitForMessage: Boolean);
```

C++

```
__fastcall ProcessRequests(Boolean WaitForMessage);
```

Description

TServiceThread (see page 3095)'s ProcessRequests is called in the Execute (see page 3097) method after any associated service's OnExecute event handler has finished executing. For more information about optimizing service execution by spawning individual threads for each service request, see the Developer's Guide.

See Also

OnExecute (see page 3088)

33.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCurrentStatus (see page 3102)	TCurrentStatus indicates the state of a service.
	TErrorSeverity (see page 3103)	TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.
	TServiceType (see page 3103)	TServiceType indicates the type of a service.
	TStartType (see page 3104)	TStartType indicates how a service starts up.

33.2.1 SvcMgr.TCurrentStatus Enumeration

TCurrentStatus indicates the state of a service.

Pascal

```
TCurrentStatus = (
  csStopped,
  csStartPending,
  csStopPending,
  csRunning,
  csContinuePending,
  csPausePending,
  csPaused
);
```

C++

```
enum TCurrentStatus {
  csStopped,
  csStartPending,
  csStopPending,
  csRunning,
  csContinuePending,
  csPausePending,
  csPaused
};
```

File

SvcMgr

Description

TCurrentStatus can have one of the following values:

Value	Meaning
csStopped	The service is not running.
csStartPending	The service is starting.
csStopPending	The service is stopping.
csRunning	The service is running.

csContinuePending	The service continue is pending.
csPausePending	The service pause is pending.
csPaused	The service is paused

33.2.2 SvcMgr.TErrorSeverity Enumeration

TErrorSeverity indicates the severity of a failure and implies the corresponding response by a service startup program.

Pascal

```
TErrorSeverity = (
  esIgnore,
  esNormal,
  esSevere,
  esCritical
);
```

C++

```
enum TErrorSeverity {
  esIgnore,
  esNormal,
  esSevere,
  esCritical
};
```

File

SvcMgr

Description

TErrorSeverity can have one of the following values:

Value	What the startup program does
esIgnore	Logs the error and continues.
esNormal	Logs the error and displays a message, but continues.
esSevere	Logs the error but continues only if the last-known-good configuration is being started; otherwise the last-known-good configuration is started instead.
esCritical	Logs the error and starts up the last-known-good configuration instead. If this is the last-known-good configuration, the startup operation fails.

33.2.3 SvcMgr.TServiceType Enumeration

TServiceType indicates the type of a service.

Pascal

```
TServiceType = (
  stWin32,
  stDevice,
  stFileSystem
);
```

C++

```
enum TServiceType {
  stWin32,
  stDevice,
  stFileSystem
};
```

File

SvcMgr

Description

TServiceType can have one of the following values:

Value	Meaning
stWin32	a Win32 service.
stDevice	a device driver.
stFileSystem	a file system driver.

33.2.4 SvcMgr.TStartType Enumeration

TStartType indicates how a service starts up.

Pascal

```
TStartType = (
  stBoot,
  stSystem,
  stAuto,
  stManual,
  stDisabled
);
```

C++

```
enum TStartType {
  stBoot,
  stSystem,
  stAuto,
  stManual,
  stDisabled
};
```

File

SvcMgr

Description

The following table lists the possible values of TStartType:

StartType value	Service is started
stBoot(used only when ServiceType is not stWin32)	by the operating system loader
stSystem(used only when ServiceType is not stWin32)	after boot systems are initialized
stAuto	automatically during system startup

stManual	when a process calls the Windows StartService function
stDisabled	the service can't be started except by an administrator

33.3 Types

The following table lists types in this documentation.

Types

Name	Description
TContinueEvent (see page 3106)	TContinueEvent is the type for event handlers that respond when the Service Control manager resumes a service.
TPauseEvent (see page 3106)	TPauseEvent is the type for event handlers that respond when the Service Control manager pauses a service.
TServiceController (see page 3107)	TServiceController is the type for a registered service's handler function.
TServiceEvent (see page 3107)	TServiceEvent is the procedure type used for service events that do not require parameters.
TStartEvent (see page 3108)	TStartEvent is the type for event handlers that respond when the Service Control manager starts a service.
TStopEvent (see page 3108)	TStopEvent is the type for event handlers that respond when the Service Control manager stops a service.

33.3.1 SvcMgr.TContinueEvent Type

TContinueEvent is the type for event handlers that respond when the Service Control manager resumes a service.

Pascal

```
TContinueEvent = procedure (Sender: TService; var Continued: Boolean) of object;
```

C++

```
(Sender: TService; var Continued: Boolean) ( TContinueEvent)();
```

File

SvcMgr

Description

Sender is the service that the Service Control manager resumed.

Set Continued to true when the service resumes.

See Also

Procedural types

Creating events

33.3.2 SvcMgr.TPauseEvent Type

TPauseEvent is the type for event handlers that respond when the Service Control manager pauses a service.

Pascal

```
TPauseEvent = procedure (Sender: TService; var Paused: Boolean) of object;
```

C++

```
(Sender: TService; var Paused: Boolean) ( TPauseEvent)();
```

File

SvcMgr

Description

Sender is the service that the Service Control manager paused.

Set Paused to true if the event handler pauses the service.

See Also

Procedural types

Creating events

33.3.3 SvcMgr.TServiceController Type

TServiceController is the type for a registered service's handler function.

Pascal

```
TServiceController = procedure (CtrlCode: DWord);
```

C++

```
(CtrlCode: DWord) ( TServiceController)();
```

File

SvcMgr

Description

TServiceController is the type of the service handler function that the Service application wizard generates.

33.3.4 SvcMgr.TServiceEvent Type

TServiceEvent is the procedure type used for service events that do not require parameters.

Pascal

```
TServiceEvent = procedure (Sender: TService) of object;
```

C++

```
(Sender: TService) ( TServiceEvent)();
```

File

SvcMgr

Description

TServiceEvent specifies a method used for service events that have no event-specific parameters. TServiceEvent is used to execute a process that occurs typically during installation, uninstallation, or shutdown. It is also the event type for the main service execution. The Sender parameter is the service component receiving the event notification.

See Also

Procedural types

Creating events

33.3.5 SvcMgr.TStartEvent Type

TStartEvent is the type for event handlers that respond when the Service Control manager starts a service.

Pascal

```
TStartEvent = procedure (Sender: TService; var Started: Boolean) of object;
```

C++

```
(Sender: TService; var Started: Boolean) ( TStartEvent)();
```

File

SvcMgr

Description

Sender is the service that the Service Control manager started.

Set Started to true if the event handler starts the service..

See Also

Procedural types

Creating events

33.3.6 SvcMgr.TStopEvent Type

TStopEvent is the type for event handlers that respond when the Service Control manager stops a service.

Pascal

```
TStopEvent = procedure (Sender: TService; var Stopped: Boolean) of object;
```

C++

```
(Sender: TService; var Stopped: Boolean) ( TStopEvent)();
```

File

SvcMgr

Description

Sender is the service that the Service Control manager stopped.

Set Stopped to true if the event handler stops the service..

See Also

Procedural types

Creating events

33.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 3109)	Represents a service Application for a service project.

33.4.1 SvcMgr.Application Variable

Represents a service Application for a service project.

Pascal

```
Application: TServiceApplication = nil;
```

C++

```
TServiceApplication Application = nil;
```

File

SvcMgr

Description

By default, when a new service Application project is created, the Forms designer constructs an Application object and assigns it to Application. If the project represents a service Application, Application is initialized to a TServiceApplication (see page 3089) object.

33.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_SERVICE_CONTROL_CODE (see page 3110)	This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

33.5.1 SvcMgr.CM_SERVICE_CONTROL_CODE Constant

Pascal

```
CM_SERVICE_CONTROL_CODE = WM_USER + 1;
```

C++

```
CM_SERVICE_CONTROL_CODE = WM_USER + 1;
```

File

SvcMgr

Description

This is constant SvcMgr.CM_SERVICE_CONTROL_CODE.

34 SyncObjs Namespace

This is namespace SyncObjs.

Topics

Name	Description
Classes (see page 3112)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 3156)	The following table lists structs, records, enums in this documentation.
Types (see page 3157)	The following table lists types in this documentation.

Classes

	Name	Description
	TConditionVariableCS (see page 3112)	TConditionVariableCS allows the use of condition variables in combination with critical sections.
	TConditionVariableHelper (see page 3117)	TConditionVariableHelper is a class helper for the `TRTLConditionVariable` record.
	TConditionVariableMutex (see page 3120)	TConditionVariableMutex allows the use of condition variables in combination with mutexes.
	TCriticalSection (see page 3125)	TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.
	TCriticalSectionHelper (see page 3130)	TCriticalSectionHelper is a record helper for the `TRTLCriticalSection` record.
	TEvent (see page 3133)	TEvent represents an external event.
	THandleObject (see page 3137)	THandleObject is the base class for all thread synchronization objects that use a handle.
	TMutex (see page 3143)	TMutex is a synchronization primitive used to guard a shared resource.
	TSemaphore (see page 3147)	TSemaphore is a synchronization primitive that manages the concurrent execution of threads, in the context of sharing a common resource.
	TSimpleEvent (see page 3152)	TSimpleEvent represents an unnamed manual event object.
	TSynchroObject (see page 3152)	TSynchroObject is the base class for all thread synchronization objects.

Structs, Records, Enums

	Name	Description
	TWaitResult (see page 3156)	TWaitResult indicates the reason an event object stopped waiting for its signal.

Types

Name	Description
THandleObjectArray (see page 3157)	THandleObjectArray represents an array of THandleObject (see page 3137).

34.1 Classes

The following table lists classes in this documentation.

Class Helpers

Name	Description
TConditionVariableHelper (see page 3117)	TConditionVariableHelper is a class helper for the `TRTLConditionVariable` record.
TCriticalSectionHelper (see page 3130)	TCriticalSectionHelper is a record helper for the `TRTLCriticalSection` record.

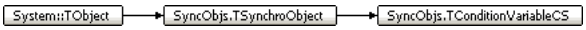
Classes

	Name	Description
	TConditionVariableCS (see page 3112)	TConditionVariableCS allows the use of condition variables in combination with critical sections.
	TConditionVariableMutex (see page 3120)	TConditionVariableMutex allows the use of condition variables in combination with mutexes.
	TCriticalSection (see page 3125)	TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.
	TEvent (see page 3133)	TEvent represents an external event.
	THandleObject (see page 3137)	THandleObject is the base class for all thread synchronization objects that use a handle.
	TMutex (see page 3143)	TMutex is a synchronization primitive used to guard a shared resource.
	TSemaphore (see page 3147)	TSemaphore is a synchronization primitive that manages the concurrent execution of threads, in the context of sharing a common resource.
	TSimpleEvent (see page 3152)	TSimpleEvent represents an unnamed manual event object.
	TSynchroObject (see page 3152)	TSynchroObject is the base class for all thread synchronization objects.

34.1.1 TConditionVariableCS Class

TConditionVariableCS allows the use of condition variables in combination with critical sections.

Class Hierarchy



Pascal

```
TConditionVariableCS = class(TSynchroObject);
```

C++

```
class TConditionVariableCS : public TSynchroObject;
```

File

SyncObjs

Description

Use TConditionVariableCS in multi-threaded applications in which a thread is required to sleep, inside its critical section, until a specified condition becomes true. During this waiting period, other threads are able to run and are supposed to notify the sleeping thread when the condition becomes true.

When the condition variable is awakened, the critical section is automatically reacquired.

To create a condition variable, instantiate the TConditionVariableCS class. Then, in the executing code of each thread, use a

combination of WaitFor (see page 3116) and the Release (see page 3115) or ReleaseAll (see page 3116) methods in order to achieve the communication between threads.

See Also

Create (see page 3114)

WaitFor (see page 3116)

Release (see page 3115)

ReleaseAll (see page 3116)

TCriticalSection (see page 3125)

TMutex (see page 3143)

TSemaphore (see page 3147)

TEvent (see page 3133)

TThread

THandleObject (see page 3137)

TConditionVariableMutex (see page 3120)

TConditionVariableHelper (see page 3117)

TCriticalSectionHelper (see page 3130)

34.1.1.1 TConditionVariableCS Members

The following tables list the members exposed by TConditionVariableCS.

Protected Fields

	Name	Description
	FConditionVariable (see page 3115)	This is FConditionVariable, a member of class TConditionVariableCS.

Public Methods

	Name	Description
	Acquire (see page 3115)	Raises an exception because a condition variable cannot be acquired.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3114)	Has no effect.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3115)	Wakes up a single thread sleeping on the condition variable.
	ReleaseAll (see page 3116)	Wakes up all threads sleeping on the condition variable.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
	WaitFor (see page 3116)	This is the overview for the WaitFor method overload.

34.1.1.2 TConditionVariableCS.Create Constructor

Has no effect.

Pascal

```
constructor Create;
```

C++

```
__fastcall TConditionVariableCS();
```

Description

The Create method is empty and is only provided for code compatibility reasons.

See Also

Acquire (see page 3115)

Release (see page 3115)

ReleaseAll (see page 3116)

WaitFor (see page 3116)

34.1.1.3 TConditionVariableCS Fields

The fields of the TConditionVariableCS class are listed here.

Protected Fields

	Name	Description
	FConditionVariable (see page 3115)	This is FConditionVariable, a member of class TConditionVariableCS.

34.1.1.3.1 TConditionVariableCS.FConditionVariable Field

Pascal

```
FConditionVariable: TRTLConditionVariable;
```

C++

```
TRTLConditionVariable FConditionVariable;
```

Description

This is FConditionVariable, a member of class TConditionVariableCS.

34.1.1.4 TConditionVariableCS Methods

The methods of the TConditionVariableCS class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3115)	Raises an exception because a condition variable cannot be acquired.
	Release (see page 3115)	Wakes up a single thread sleeping on the condition variable.
	ReleaseAll (see page 3116)	Wakes up all threads sleeping on the condition variable.
	WaitFor (see page 3116)	This is the overview for the WaitFor method overload.

34.1.1.4.1 TConditionVariableCS.Acquire Method

Raises an exception because a condition variable cannot be acquired.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

The Acquire method raises an exception because condition variables cannot be acquired. Acquire is only provided for code compatibility reasons.

See Also

Create (see page 3114)

Release (see page 3115)

ReleaseAll (see page 3116)

WaitFor (see page 3116)

34.1.1.4.2 TConditionVariableCS.Release Method

Wakes up a single thread sleeping on the condition variable.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release to wake up a single thread sleeping on the condition variable.

Note: To reduce the number of context switches, make a call to Release after releasing the critical section.

See Also

ReleaseAll (see page 3116)

WaitFor (see page 3116)

34.1.1.4.3 TConditionVariableCS.ReleaseAll Method

Wakes up all threads sleeping on the condition variable.

Pascal

```
procedure ReleaseAll;
```

C++

```
__fastcall ReleaseAll();
```

Description

Call ReleaseAll to wake up all threads sleeping on the condition variable.

Note: To reduce the number of context switches, make a call to ReleaseAll after releasing the critical section.

See Also

Release (see page 3115)

WaitFor (see page 3116)

34.1.1.4.4 WaitFor Method

This is the overview for the WaitFor method overload.

Overload List

	Name	Description
	TConditionVariableCS.WaitFor (TCriticalSection, LongWord) (see page 3116)	Sleeps on the condition variable and releases the given critical section.
	TConditionVariableCS.WaitFor (TRTLCriticalSection, LongWord) (see page 3116)	Sleeps on the condition variable and releases the given critical section.

34.1.1.4.4.1 TConditionVariableCS.WaitFor Method (TCriticalSection, LongWord)

Sleeps on the condition variable and releases the given critical section.

Pascal

```
function WaitFor(CriticalSection: TCriticalSection; TimeOut: LongWord = INFINITE):
TWaitResult; overload;
function WaitFor(var CriticalSection: TRTLCriticalSection; TimeOut: LongWord = INFINITE):
TWaitResult; overload;
```

C++

```
__fastcall TWaitResult WaitFor(TCriticalSection CriticalSection, LongWord TimeOut = INFINITE);
__fastcall TWaitResult WaitFor(TRTLCriticalSection CriticalSection, LongWord TimeOut =
INFINITE);
```

Description

Call WaitFor to sleep on the condition variable and to release the critical section specified through the `CriticalSection` parameter.

The `TimeOut` parameter sets the time-out interval. When this interval elapses, the critical section is reacquired, even if the condition variable has not been awakened by other threads.

The result returned by the WaitFor function is of TWaitResult (see page 3156) type and can have one of the following values:

Value	Meaning
wrSignaled	The condition variable has been awakened.
wrTimeout	The time-out interval has elapsed without the condition variable being awakened.
wrAbandoned	The condition variable has been destroyed before the time-out interval has elapsed.
wrError	An error occured while waiting. Check the `LastError` property for an error code giving more information.

Condition variables might suddenly wake up, without making an explicit call to the Release (see page 3115) or ReleaseAll (see page 3116) methods. Also, other threads might manage to run before the thread that sleeps on the condition variable is awakened. Therefore, make the call to WaitFor inside a loop. To avoid the above mentioned errors, the loop should test that the desired condition becomes true even after the condition variable is awakened.

See Also

Release (see page 3115)

ReleaseAll (see page 3116)

34.1.2 TConditionVariableHelper Class Helper

TConditionVariableHelper is a class helper for the `TRTLConditionVariable` record.

Class Hierarchy

SyncObjs.TConditionVariableHelper

Pascal

```
TConditionVariableHelper = class helper for TRTLConditionVariable;
```

C++

```
class TConditionVariableHelper;
```

File

SyncObjs

Description

TConditionVariableHelper provides several operating-system-dependent helper routines for the `TRTLConditionVariable` record.

See Also

SleepCS (see page 3119)

Wake (see page 3120)

WakeAll (see page 3120)

TCriticalSection (see page 3125)

TMutex (see page 3143)

TSemaphore (see page 3147)

TEvent (see page 3133)

TThread

THandleObject (see page 3137)

TConditionVariableCS (see page 3112)

TConditionVariableMutex (see page 3120)

TCriticalSectionHelper (see page 3130)

34.1.2.1 TConditionVariableHelper Members

The following tables list the members exposed by TConditionVariableHelper.

Public Methods

	Name	Description
	Create (see page 3118)	Initializes the condition variable.
	Free (see page 3119)	Has no effect.
	SleepCS (see page 3119)	Sleeps on the condition variable and releases the given critical section.
	Wake (see page 3120)	Wakes up a single thread sleeping on the condition variable.
	WakeAll (see page 3120)	Wakes up all threads sleeping on the condition variable.

34.1.2.2 TConditionVariableHelper Methods

The methods of the TConditionVariableHelper class are listed here.

Public Methods

	Name	Description
	Create (see page 3118)	Initializes the condition variable.
	Free (see page 3119)	Has no effect.
	SleepCS (see page 3119)	Sleeps on the condition variable and releases the given critical section.
	Wake (see page 3120)	Wakes up a single thread sleeping on the condition variable.
	WakeAll (see page 3120)	Wakes up all threads sleeping on the condition variable.

34.1.2.2.1 TConditionVariableHelper.Create Method

Initializes the condition variable.

Pascal

```
class function Create: TRTLConditionVariable; static;
```

C++

```
static __fastcall TRTLConditionVariable Create();
```

Description

Call Create to initialize the condition variable.

Note: Condition variables are shared by threads only inside the same process. They cannot be shared between processes.

See Also

Free (see page 3119)

SleepCS (see page 3119)

Wake (see page 3120)

WakeAll (see page 3120)

34.1.2.2.2 TConditionVariableHelper.Free Method

Has no effect.

Pascal

```
procedure Free;
```

C++

```
__fastcall Free();
```

Description

The Free method is empty and is only provided for code compatibility reasons.

See Also

Create (see page 3118)

34.1.2.2.3 TConditionVariableHelper.SleepCS Method

Sleeps on the condition variable and releases the given critical section.

Pascal

```
function SleepCS(var CriticalSection: TRTLCriticalSection; dwMilliseconds: DWORD): Boolean;
```

C++

```
__fastcall Boolean SleepCS(TRTLCriticalSection CriticalSection, DWORD dwMilliseconds);
```

Description

Call SleepCS to sleep on the condition variable and to release the critical section specified through the `CriticalSection` parameter.

The `dwMilliseconds` parameter sets the time-out interval. When this interval elapses, the critical section is reacquired, even if the condition variable has not been awakened by other threads.

Condition variables might suddenly wake up without making an explicit call to the Wake (see page 3120) or to the WakeAll (see page 3120) method. Also, other threads might run before the thread that sleeps on the condition variable is awakened. Therefore, make the call to SleepCS inside a loop. To avoid the above mentioned errors, the loop should test that the desired condition becomes true even after the condition variable has awakened.

See Also

Wake (see page 3120)

WakeAll (see page 3120)

34.1.2.2.4 TConditionVariableHelper.Wake Method

Wakes up a single thread sleeping on the condition variable.

Pascal

```
procedure Wake;
```

C++

```
__fastcall Wake();
```

Description

Call Wake to wake up a single thread sleeping on the condition variable.

Note: To reduce the number of context switches, make a call to Wake after releasing the critical section.

See Also

SleepCS (see page 3119)

WakeAll (see page 3120)

34.1.2.2.5 TConditionVariableHelper.WakeAll Method

Wakes up all threads sleeping on the condition variable.

Pascal

```
procedure WakeAll;
```

C++

```
__fastcall WakeAll();
```

Description

Call Wake (see page 3120) to wake up all threads sleeping on the condition variable.

Note: To reduce the number of context switches, make a call to WakeAll after releasing the critical section.

See Also

SleepCS (see page 3119)

Wake (see page 3120)

34.1.3 TConditionVariableMutex Class

TConditionVariableMutex allows the use of condition variables in combination with mutexes.

Class Hierarchy

Pascal

```
TConditionVariableMutex = class(TSynchroObject);
```

C++

```
class TConditionVariableMutex : public TSynchroObject;
```

File

SyncObjs

Description

Use TConditionVariableMutex in multi-threaded applications in which a thread is required to release a mutex until a specified condition becomes true. During this waiting period, other threads are able to run and are supposed to notify the sleeping thread when the condition becomes true.

When the condition variable is awakened, the mutex is automatically reacquired.

To create a condition variable, simply instantiate the TConditionVariableMutex class. Then, in the executing code of each thread, use a combination of WaitFor (see page 3124) and the Release (see page 3124) or ReleaseAll (see page 3124) methods in order to achieve the communication between threads.

See Also

Create (see page 3122)

Destroy (see page 3123)

WaitFor (see page 3124)

Release (see page 3124)

ReleaseAll (see page 3124)

TCriticalSection (see page 3125)

TMutex (see page 3143)

TSemaphore (see page 3147)

TEvent (see page 3133)

TThread

THandleObject (see page 3137)

TConditionVariableCS (see page 3112)

TConditionVariableHelper (see page 3117)

TCriticalSectionHelper (see page 3130)

34.1.3.1 TConditionVariableMutex Members

The following tables list the members exposed by TConditionVariableMutex.

Public Methods

	Name	Description
	Acquire (see page 3123)	Raises an exception, since a condition variable cannot be acquired.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3122)	Initializes the mutex condition variable.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3123)	Destroys the mutex condition variable.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 3124)	Wakes up a single thread sleeping on the mutex condition variable.
	ReleaseAll (see page 3124)	Wakes up all threads sleeping on the mutex condition variable.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
	WaitFor (see page 3124)	Sleeps on the condition variable and releases the specified mutex.

34.1.3.2 TConditionVariableMutex.Create Constructor

Initializes the mutex condition variable.

Pascal

```
constructor Create;
```

C++

```
__fastcall TConditionVariableMutex();
```

Description

Call Create to initialize the mutex condition variable.

See Also

Acquire (see page 3123)

Destroy (see page 3123)

Release (see page 3124)

ReleaseAll (see page 3124)

WaitFor (see page 3124)

34.1.3.3 TConditionVariableMutex.Destroy Destructor

Destroys the mutex condition variable.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TConditionVariableMutex();
```

Description

Call Destroy to destroy the mutex condition variable.

See Also

Create (see page 3122)

Acquire (see page 3123)

Release (see page 3124)

ReleaseAll (see page 3124)

WaitFor (see page 3124)

34.1.3.4 TConditionVariableMutex Methods

The methods of the TConditionVariableMutex class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3123)	Raises an exception, since a condition variable cannot be acquired.
	Release (see page 3124)	Wakes up a single thread sleeping on the mutex condition variable.
	ReleaseAll (see page 3124)	Wakes up all threads sleeping on the mutex condition variable.
	WaitFor (see page 3124)	Sleeps on the condition variable and releases the specified mutex.

34.1.3.4.1 TConditionVariableMutex.Acquire Method

Raises an exception, since a condition variable cannot be acquired.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

The Acquire method raises an exception, since condition variables cannot be acquired. Acquire is only provided for code compatibility reasons.

See Also

Create (see page 3122)

Destroy (see page 3123)

Release (see page 3124)

ReleaseAll (see page 3124)

WaitFor (see page 3124)

34.1.3.4.2 TConditionVariableMutex.Release Method

Wakes up a single thread sleeping on the mutex condition variable.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release to wake up a single thread sleeping on the mutex condition variable.

Note: To reduce the number of context switches, make a call to Release after releasing the mutex.

See Also

ReleaseAll (see page 3124)

WaitFor (see page 3124)

34.1.3.4.3 TConditionVariableMutex.ReleaseAll Method

Wakes up all threads sleeping on the mutex condition variable.

Pascal

```
procedure ReleaseAll;
```

C++

```
__fastcall ReleaseAll();
```

Description

Call Release (see page 3124) to wake up all threads sleeping on the mutex condition variable.

Note: To reduce the number of context switches, make a call to ReleaseAll after releasing the mutex.

See Also

Release (see page 3124)

WaitFor (see page 3124)

34.1.3.4.4 TConditionVariableMutex.WaitFor Method

Sleeps on the condition variable and releases the specified mutex.

Pascal

```
function WaitFor(AExternalMutex: TMutex; TimeOut: LongWord = INFINITE): TWaitResult;
```

C++

```
__fastcall TWaitResult WaitFor(TMutex AExternalMutex, LongWord TimeOut = INFINITE);
```

Description

Call WaitFor to sleep on the condition variable and to release the mutex specified through the `AExternalMutex` parameter.

The `TimeOut` parameter sets the time-out interval. When this interval elapses, the mutex is reacquired, even if the condition variable has not been awakened by other threads.

The result returned by the WaitFor function is of TWaitResult (see page 3156) type and can have one of the following values:

Value	Meaning
wrSignaled	The condition variable has been awakened.
wrTimeout	The time-out interval has elapsed without the condition variable being awakened.
wrAbandoned	The condition variable had been destroyed before the time-out interval has elapsed.
wrError	An error occured while waiting. Check the `LastError` property for an error code giving more information.

Condition variables might suddenly wake up, without making an explicit call to the Release (see page 3124) or to the ReleaseAll (see page 3124) method. Also, other threads might run before the thread that sleeps on the condition variable is awakened. Therefore, make the call to WaitFor inside a loop. To avoid the above mentioned errors, the loop should test that the desired condition becomes true even after the condition variable has awakened.

See Also

Release (see page 3124)

ReleaseAll (see page 3124)

34.1.4 TCriticalSection Class

TCriticalSection allows a thread in a multi-threaded application to temporarily block other threads from accessing a block of code.

Class Hierarchy

Pascal

```
TCriticalSection = class(TSynchroObject);
```

C++

```
class TCriticalSection : public TSynchroObject;
```

File

SyncObjs

Description

Use TCriticalSection to safeguard operations that may be disrupted if another thread executes certain sections of code before the operation is completed. Critical sections work like gates that allow only a single thread to enter at a time. Because they block the execution of other threads, overusing critical sections can seriously interfere with performance.

Critical sections must be global in scope so that they are available to all threads. In every thread, any action that could interfere with the operations safeguarded by the critical section should only take place after calling the Acquire (see page 3128) or Enter (see page 3128) method. Threads that do not use the critical section for relevant operations can introduce bugs.

See Also

TThread

Using Critical Sections

34.1.4.1 TCriticalSection Members

The following tables list the members exposed by TCriticalSection.

Protected Fields

	Name	Description
	FSection (see page 3128)	This is FSection, a member of class TCriticalSection.

Public Methods

	Name	Description
	Acquire (see page 3128)	Binds the critical section to the calling thread.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3127)	Creates an instance of TCriticalSection (see page 3125).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3127)	Destroys an instance of TCriticalSection (see page 3125).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Enter (see page 3128)	Blocks other threads when the calling thread enters a thread-sensitive section.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	Leave (see page 3129)	Allows other threads to use the critical section.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3129)	Allows other threads to use the critical section.

	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	TryEnter (see page 3130)	Tries to enter a critical section.
	UnitName (see page 3380)	Returns a string containing the unit name.

34.1.4.2 TCriticalSection.Create Constructor

Creates an instance of TCriticalSection (see page 3125).

Pascal

```
constructor Create;
```

C++

```
__fastcall TCriticalSection();
```

Description

Call Create to create and initialize an instance of TCriticalSection (see page 3125). A TCriticalSection (see page 3125) object can be created for a single object that contains thread-sensitive code, or as a global object to be shared by several threads.

See Also

System::TObject::Create (see page 3355)

34.1.4.3 TCriticalSection.Destroy Destructor

Destroys an instance of TCriticalSection (see page 3125).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCriticalSection();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCriticalSection (see page 3125) object is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

34.1.4.4 TCriticalSection Fields

The fields of the TCriticalSection class are listed here.

Protected Fields

	Name	Description
	FSection (see page 3128)	This is FSection, a member of class TCriticalSection.

34.1.4.4.1 TCriticalSection.FSection Field

Pascal

```
FSection: TRTLCriticalSection;
```

C++

```
TRTLCriticalSection FSection;
```

Description

This is FSection, a member of class TCriticalSection.

34.1.4.5 TCriticalSection Methods

The methods of the TCriticalSection class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3128)	Binds the critical section to the calling thread.
	Enter (see page 3128)	Blocks other threads when the calling thread enters a thread-sensitive section.
	Leave (see page 3129)	Allows other threads to use the critical section.
	Release (see page 3129)	Allows other threads to use the critical section.
	TryEnter (see page 3130)	Tries to enter a critical section.

34.1.4.5.1 TCriticalSection.Acquire Method

Binds the critical section to the calling thread.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire to block all other threads from acquiring this critical section until the Release (see page 3129) or Leave (see page 3129) method is called. Acquire does the same thing as the Enter (see page 3128) method.

See Also

Enter (see page 3128)

Leave (see page 3129)

Release (see page 3129)

Using Critical Sections

34.1.4.5.2 TCriticalSection.Enter Method

Blocks other threads when the calling thread enters a thread-sensitive section.

Pascal

```
procedure Enter;
```

C++

```
__fastcall Enter();
```

Description

Call Enter to block all other threads from entering code protected by this critical section until the Leave (see page 3129) or Release (see page 3129) method is called. Enter calls the Acquire (see page 3128) method to bind the critical section to the calling thread.

See Also

Acquire (see page 3128)

Leave (see page 3129)

Release (see page 3129)

34.1.4.5.3 TCriticalSection.Leave Method

Allows other threads to use the critical section.

Pascal

```
procedure Leave;
```

C++

```
__fastcall Leave();
```

Description

Call Leave at the end of a thread-sensitive section of code. Each call to Leave should be balanced by a call to Enter (see page 3128) or Acquire (see page 3128) at the beginning of the thread-sensitive section. Leave calls Release (see page 3129) to allow other threads to acquire the critical section.

See Also

Acquire (see page 3128)

Enter (see page 3128)

Release (see page 3129)

34.1.4.5.4 TCriticalSection.Release Method

Allows other threads to use the critical section.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release when the thread no longer needs to block other threads from executing code that is protected by this critical section. Each call to Release should be balance by an earlier call to Acquire (see page 3128) or Enter (see page 3128) that blocked other threads from executing. Release does the same thing as the Leave (see page 3129) method.

See Also

Acquire (see page 3128)

Enter (see page 3128)

Leave (see page 3129)

Using Critical Sections

34.1.4.5.5 TCriticalSection.TryEnter Method

Tries to enter a critical section.

Pascal

```
function TryEnter: Boolean;
```

C++

```
__fastcall Boolean TryEnter();
```

Description

Call TryEnter to block all the other threads from entering the code that is protected by this critical section until the Leave (see page 3129) or Release (see page 3129) methods are called.

TryEnter does not wait for other threads to leave the critical section. True is returned on success; False is returned if the section is acquired by another thread.

See Also

Acquire (see page 3128)

Leave (see page 3129)

Release (see page 3129)

34.1.5 TCriticalSectionHelper Class Helper

TCriticalSectionHelper is a record helper for the `TRTLCriticalSection` record.

Class Hierarchy

SyncObjs.TCriticalSectionHelper

Pascal

```
TCriticalSectionHelper = class helper for TRTLCriticalSection;
```

C++

```
class TCriticalSectionHelper;
```

File

SyncObjs

Description

TCriticalSectionHelper provides several operating-system-dependent helper routines for the `TRTLCriticalSection` record.

See Also

TCriticalSection (see page 3125)

TMutex (see page 3143)

TSemaphore (see page 3147)

TEvent (see page 3133)

TThread

THandleObject (see page 3137)

TConditionVariableCS (see page 3112)

TConditionVariableMutex (see page 3120)

TConditionVariableHelper (see page 3117)

34.1.5.1 TCriticalSectionHelper Members

The following tables list the members exposed by TCriticalSectionHelper.

Public Methods

	Name	Description
	Destroy (see page 3131)	Deletes the critical section from memory.
	Enter (see page 3132)	Waits until the critical section is free and then locks it.
	Free (see page 3132)	Deletes the critical section.
	Initialize (see page 3132)	Initializes the critical section.
	Leave (see page 3133)	Releases the lock on the critical section.
	TryEnter (see page 3133)	Attempts to acquire a lock on the critical section.

34.1.5.2 TCriticalSectionHelper Methods

The methods of the TCriticalSectionHelper class are listed here.

Public Methods

	Name	Description
	Destroy (see page 3131)	Deletes the critical section from memory.
	Enter (see page 3132)	Waits until the critical section is free and then locks it.
	Free (see page 3132)	Deletes the critical section.
	Initialize (see page 3132)	Initializes the critical section.
	Leave (see page 3133)	Releases the lock on the critical section.
	TryEnter (see page 3133)	Attempts to acquire a lock on the critical section.

34.1.5.2.1 TCriticalSectionHelper.Destroy Method

Deletes the critical section from memory.

Pascal

```
procedure Destroy;
```

C++

```
__fastcall Destroy();
```

Description

Call Destroy to delete the critical section from memory and to release all the resources used by the critical section.

See Also

Free (see page 3132)

Initialize (see page 3132)

34.1.5.2.2 TCriticalSectionHelper.Enter Method

Waits until the critical section is free and then locks it.

Pascal

```
procedure Enter;
```

C++

```
__fastcall Enter();
```

Description

Call Enter to grant ownership of the critical section to the calling thread.

If the critical section is not currently free, then the calling thread is added to the waiting queue of the critical section.

See Also

TryEnter (see page 3133)

Leave (see page 3133)

34.1.5.2.3 TCriticalSectionHelper.Free Method

Deletes the critical section.

Pascal

```
procedure Free;
```

C++

```
__fastcall Free();
```

Description

Call Free to delete the critical section.

Free is an alias for the Destroy (see page 3131) method. You can use either of these methods, with the same effect.

See Also

Destroy (see page 3131)

Initialize (see page 3132)

34.1.5.2.4 TCriticalSectionHelper.Initialize Method

Initializes the critical section.

Pascal

```
procedure Initialize;
```

C++

```
__fastcall Initialize();
```

Description

Call Initialize to initialize the critical section.

See Also

Free (see page 3132)

Destroy (see page 3131)

34.1.5.2.5 TCriticalSectionHelper.Leave Method

Releases the lock on the critical section.

Pascal

```
procedure Leave;
```

C++

```
__fastcall Leave();
```

Description

Call Leave to release the lock on the critical section. This lock is acquired by using the Enter (see page 3132) method.

See Also

Enter (see page 3132)

34.1.5.2.6 TCriticalSectionHelper.TryEnter Method

Attempts to acquire a lock on the critical section.

Pascal

```
function TryEnter: Boolean;
```

C++

```
__fastcall Boolean TryEnter();
```

Description

Call TryEnter when attempting to lock the critical section.

If successful, the TryEnter method grants ownership of the critical section to the calling thread and returns True. Otherwise, it returns False and no ownership is granted.

See Also

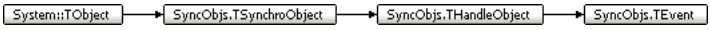
Enter (see page 3132)

Leave (see page 3133)

34.1.6 TEvent Class

TEvent represents an external event.

Class Hierarchy



Pascal

```
TEvent = class(THandleObject);
```

C++

```
class TEvent : public THandleObject;
```

File

SyncObjs

Description

Use TEvent to signal that an event has occurred or a state has been reached.

In a multi-threaded application, use TEvent to allow one thread to signal to other threads that an event has occurred.

The handle of a TEvent object can also be used to communicate with other processes, so that an application can coordinate the timing of events with other applications. For example, use the handle of a TEvent object to wait until another process is ready to transfer information.

In a single-threaded application, use TEvent to coordinate between sections of code that respond to asynchronous events such as system events.

See Also

TThread

Waiting for a Task to Be Completed

34.1.6.1 TEvent Members

The following tables list the members exposed by TEvent.

Protected Fields

	Name	Description
	FHandle (see page 3140)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3140)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3140)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
	Acquire (see page 3154)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3135)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3139)	Destroys an instance of THandleObject (see page 3137).
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.

	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3154)	Provides the interface for a method that releases the synchronization object for use by other threads.
	ResetEvent (see page 3136)	Turns off the signal of the event object.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetEvent (see page 3137)	Turns on the signal of the event object.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
	WaitFor (see page 3141)	Waits until this THandleObject (see page 3137) object enters into the signaled state.
	WaitForMultiple (see page 3141)	Waits until a list of THandleObject (see page 3137) objects enters into the signaled state.

Public Properties

	Name	Description
	Handle (see page 3142)	Specifies the handle of the Windows object used by THandleObject (see page 3137).
	LastError (see page 3142)	Indicates the Windows error code obtained from Handle (see page 3142).

34.1.6.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TEvent.Create (Boolean) (see page 3135)	Instantiates a TEvent (see page 3133) object to represent an event object.
	TEvent.Create (PSecurityAttributes, Boolean, Boolean, string, Boolean) (see page 3135)	Instantiates a TEvent (see page 3133) object to represent an event object.

34.1.6.2.1 TEvent.Create Constructor (Boolean)

Instantiates a TEvent (see page 3133) object to represent an event object.

Pascal

```
constructor Create(EventAttributes: PSecurityAttributes; ManualReset: Boolean; InitialState:
Boolean; const Name: string; UseCOMWait: Boolean = False); overload;
constructor Create(UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TEvent(PSecurityAttributes EventAttributes, Boolean ManualReset, Boolean
InitialState, const UnicodeString Name, Boolean UseCOMWait = False);
```

```
__fastcall TEvent(Boolean UseCOMWait = False);
```

Description

Call Create to create a TEvent (see page 3133) object. Create can generate a new event object or can provide access to an existing named event object.

Set `EventAttributes` parameter to specify the access rights and inheritance capabilities of the new event object. Calling Create with EventAttributes set to nil (Delphi) or NULL (C++) creates an event object with a default set of security attributes. If the TEvent (see page 3133) object is created to access an existing named event object, only the bInheritHandle field of EventAttributes is used, as the access rights were determined when the event object was created.

Set `ManualReset` to specify whether the signal of the TEvent (see page 3133) object can only be turned off by a call to the ResetEvent (see page 3136) method, or whether it is automatically reset when a single waiting thread is released. When ManualReset is true, the TEvent (see page 3133) signal stays set until the ResetEvent (see page 3136) method turns it off. When ManualReset is false, the signal is automatically turned off when only a single thread waits on the signal and that thread is released. If the TEvent (see page 3133) object is created to access an existing named event object, the ManualReset parameter is ignored.

Set `InitialState` to indicate whether the TEvent (see page 3133) object should be created with the signal set, or turned off. When InitialState is true, the TEvent (see page 3133) object is created with the signal set. If the TEvent (see page 3133) object is created to access an existing named event object, the InitialState parameter is ignored.

Set `Name` to provide a name for a new event object or to specify an existing named event object. If no other thread or process will need to access the event object to wait for its signal, Name can be left blank. Name can be a string of up to 260 characters, not including the backslash character (\). If Name is used to specify an existing event object, the value must match the name of the existing event in a case-sensitive comparison. If Name matches the name of an existing semaphore, mutex, or file-mapping object, the TEvent (see page 3133) object will be created with Handle (see page 3142) set to 0 and all method calls will fail.

Set `UseCOMWait` to **True** to ensure that when a thread is blocked and waiting for the object, any STA COM calls can be made back into this thread.

See Also

Handle (see page 3142)

ResetEvent (see page 3136)

34.1.6.3 TEvent Methods

The methods of the TEvent class are listed here.

Public Methods

	Name	Description
	ResetEvent (see page 3136)	Turns off the signal of the event object.
	SetEvent (see page 3137)	Turns on the signal of the event object.

34.1.6.3.1 TEvent.ResetEvent Method

Turns off the signal of the event object.

Pascal

```
procedure ResetEvent;
```

C++

```
__fastcall ResetEvent();
```

Description

Call ResetEvent to signal that the event has not occurred or the desired state was not achieved. Other threads or processes that wait for the event will wait until the signal is turned on again by the SetEvent (see page 3137) method, the event object is destroyed, or the waiting process times out.

See Also

SetEvent (see page 3137)

WaitFor (see page 3141)

Waiting for a Task to Be Completed

34.1.6.3.2 TEvent.SetEvent Method

Turns on the signal of the event object.

Pascal

```
procedure SetEvent;
```

C++

```
__fastcall SetEvent();
```

Description

Call SetEvent to signal that the event has occurred or the desired state was achieved. Other threads or processes that wait for the event will resume execution with a return value indicating that the signal was set.

See Also

ResetEvent (see page 3136)

WaitFor (see page 3141)

Waiting for a Task to Be Completed

34.1.7 THandleObject Class

THandleObject is the base class for all thread synchronization objects that use a handle.

Class Hierarchy

Pascal

```
THandleObject = class(TSynchroObject);
```

C++

```
class THandleObject : public TSynchroObject;
```

File

SyncObjs

Description

Use THandleObject as a base class when defining an object that uses a handle when coordinating the execution of different threads in a multi-threaded application. Handled synchronization objects include event objects that allow threads to signal when

critical events have occurred.

Do not create instances of THandleObject. THandleObject is a base class that introduces properties for managing the handle of a synchronization object but does not provide a mechanism for acquiring that handle. Descendants of THandleObject must acquire the appropriate type of handle.

See Also

TThread

TEvent (see page 3133)

34.1.7.1 THandleObject Members

The following tables list the members exposed by THandleObject.

Protected Fields

	Name	Description
	FHandle (see page 3140)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3140)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3140)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
	Acquire (see page 3154)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3139)	Instantiates a THandleObject (see page 3137) object.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3139)	Destroys an instance of THandleObject (see page 3137).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.

	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3154)	Provides the interface for a method that releases the synchronization object for use by other threads.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
	WaitFor (see page 3141)	Waits until this THandleObject (see page 3137) object enters into the signaled state.
	WaitForMultiple (see page 3141)	Waits until a list of THandleObject (see page 3137) objects enters into the signaled state.

Public Properties

	Name	Description
	Handle (see page 3142)	Specifies the handle of the Windows object used by THandleObject (see page 3137).
	LastError (see page 3142)	Indicates the Windows error code obtained from Handle (see page 3142).

34.1.7.2 THandleObject.Create Constructor

Instantiates a THandleObject (see page 3137) object.

Pascal

```
constructor Create(UseCOMWait: Boolean = False);
```

C++

```
__fastcall THandleObject(Boolean UseCOMWait = False);
```

Description

Call Create to create a THandleObject (see page 3137) object.

Set the `UseCOMWait` parameter to ensure that when a thread is blocked and waiting for the object, any STA COM calls can be made back into this thread.

See Also

Handle (see page 3142)

34.1.7.3 THandleObject.Destroy Destructor

Destroys an instance of THandleObject (see page 3137).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THandleObject();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the THandleObject (see page 3137) reference is not nil and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

Handle (see page 3142)

34.1.7.4 THandleObject Fields

The fields of the THandleObject class are listed here.

Protected Fields

	Name	Description
	FHandle (see page 3140)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3140)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3140)	This is FUseCOMWait, a member of class THandleObject.

34.1.7.4.1 THandleObject.FHandle Field

Pascal

```
FHandle: THandle;
```

C++

```
THandle FHandle;
```

Description

This is FHandle, a member of class THandleObject.

34.1.7.4.2 THandleObject.FLastError Field

Pascal

```
FLastError: Integer;
```

C++

```
int FLastError;
```

Description

This is FLastError, a member of class THandleObject.

34.1.7.4.3 THandleObject.FUseCOMWait Field

Pascal

```
FUseCOMWait: Boolean;
```

C++

```
Boolean FUseCOMWait;
```

Description

This is FUseCOMWait, a member of class THandleObject.

34.1.7.5 THandleObject Methods

The methods of the THandleObject class are listed here.

Public Methods

	Name	Description
	WaitFor (see page 3141)	Waits until this THandleObject (see page 3137) object enters into the signaled state.
	WaitForMultiple (see page 3141)	Waits until a list of THandleObject (see page 3137) objects enters into the signaled state.

34.1.7.5.1 THandleObject.WaitFor Method

Waits until this THandleObject (see page 3137) object enters into the signaled state.

Pascal

```
function WaitFor(Timeout: LongWord): TWaitResult; virtual;
```

C++

```
virtual __fastcall TWaitResult WaitFor(LongWord Timeout);
```

Description

Call WaitFor to wait until a THandleObject (see page 3137) object enters into the signaled state. If the signaled state is not reached after `TimeOut` milliseconds, WaitFor returns anyway.

Use INFINITE (see page 3848) value for the `TimeOut` parameter to force WaitFor to wait an infinite amount of time.

The function returns a result of TWaitResult (see page 3156) type that can be used to detect how the operation finished.

See Also

WaitForMultiple (see page 3141)

LastError (see page 3142)

Waiting for a Task to Be Completed

34.1.7.5.2 THandleObject.WaitForMultiple Method

Waits until a list of THandleObject (see page 3137) objects enters into the signaled state.

Pascal

```
class function WaitForMultiple(const HandleObjs: THandleObjectArray; Timeout: LongWord; AAll:
Boolean; out SignaledObj: THandleObject; UseCOMWait: Boolean = False; Len: Integer = 0):
TWaitResult;
```

C++

```
__fastcall TWaitResult WaitForMultiple(const THandleObjectArray HandleObjs, LongWord Timeout,
Boolean AAll, THandleObject SignaledObj, Boolean UseCOMWait = False, int Len = 0);
```

Description

Call WaitForMultiple to wait until a list of THandleObject (see page 3137) objects enters into the signaled state. If the signaled state is not reached after `TimeOut` milliseconds, WaitForMultiple returns anyway.

The `HandleObjs` parameter contains the list of objects that are waited on.

If the `AAll` parameter is true, the call will finish only after all objects become signaled.

The first signaled object is passed into the `SignaledObj` parameter. This only happens if the function returns wrSignaled and `AAll` is False.

Set the `UseCOMWait` parameter to ensure that when a thread is blocked and waiting for the object, any STA COM calls can be made back into this thread.

Len contains the number of objects that are monitored in the given list.

The function returns a result of TWaitResult (see page 3156) type, which can be used to detect how the operation finished.

See Also

WaitFor (see page 3141)

LastError (see page 3142)

Waiting for a Task to Be Completed

34.1.7.6 THandleObject Properties

The properties of the THandleObject class are listed here.

Public Properties

	Name	Description
	Handle (see page 3142)	Specifies the handle of the Windows object used by THandleObject (see page 3137).
	LastError (see page 3142)	Indicates the Windows error code obtained from Handle (see page 3142).

34.1.7.6.1 THandleObject.Handle Property

Specifies the handle of the Windows object used by THandleObject (see page 3137).

Pascal

```
property Handle: THandle;
```

C++

```
__property THandle Handle;
```

Description

Read (see page 3634) Handle to obtain the Windows handle for the synchronization object. Use Handle when making Windows API calls that require a handle to the synchronization object.

THandleObject (see page 3137) provides no mechanism for setting the Handle property. Descendants of THandleObject (see page 3137) set the Handle property to the handle for whatever Windows synchronization object they represent.

See Also

Destroy (see page 3139)

34.1.7.6.2 THandleObject.LastError Property

Indicates the Windows error code obtained from Handle (see page 3142).

Pascal

```
property LastError: Integer;
```

C++

```
__property int LastError;
```

Description

Read (see page 3634) LastError when an operation of the THandleObject (see page 3137) fails. LastError returns the Windows error code that represents the reason for the failure. If no operation failed, LastError is 0.

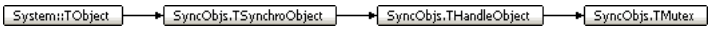
See Also

Handle (see page 3142)

34.1.8 TMutex Class

TMutex is a synchronization primitive used to guard a shared resource.

Class Hierarchy



Pascal

```
TMutex = class(THandleObject);
```

C++

```
class TMutex : public THandleObject;
```

File

SyncObjs

Description

Use TMutex to synchronize two or more threads and allow them to safely access a shared resource. A mutex creates a queue of threads that are waiting on a resource and allows only one thread to access that resource at any given moment.

A mutex is basically equivalent to a binary semaphore.

See Also

Create (see page 3145)

Acquire (see page 3146)

Release (see page 3146)

TCriticalSection (see page 3125)

TSemaphore (see page 3147)

TEvent (see page 3133)

TThread

THandleObject (see page 3137)

TConditionVariableCS (see page 3112)

TConditionVariableMutex (see page 3120)

TConditionVariableHelper (see page 3117)

TCriticalSectionHelper (see page 3130)

34.1.8.1 TMutex Members

The following tables list the members exposed by TMutex.

Protected Fields

	Name	Description
	FHandle (see page 3140)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3140)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3140)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
V	Acquire (see page 3146)	Attempts to acquire a lock on the mutex.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3145)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3139)	Destroys an instance of THandleObject (see page 3137).
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 3146)	Releases the exclusive lock on the mutex.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
V	WaitFor (see page 3141)	Waits until this THandleObject (see page 3137) object enters into the signaled state.
	WaitForMultiple (see page 3141)	Waits until a list of THandleObject (see page 3137) objects enters into the signaled state.

Public Properties

	Name	Description
R	Handle (see page 3142)	Specifies the handle of the Windows object used by THandleObject (see page 3137).
R	LastError (see page 3142)	Indicates the Windows error code obtained from Handle (see page 3142).

34.1.8.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TMutex.Create (Boolean) (see page 3145)	Instantiates a TMutex (see page 3143) object.
	TMutex.Create (LongWord, Boolean, string, Boolean) (see page 3145)	Instantiates a TMutex (see page 3143) object.
	TMutex.Create (PSecurityAttributes, Boolean, string, Boolean) (see page 3145)	Instantiates a TMutex (see page 3143) object.

34.1.8.2.1 TMutex.Create Constructor (Boolean)

Instantiates a TMutex (see page 3143) object.

Pascal

```
constructor Create(UseCOMWait: Boolean = False); overload;
constructor Create(MutexAttributes: PSecurityAttributes; InitialOwner: Boolean; const Name:
string; UseCOMWait: Boolean = False); overload;
constructor Create(DesiredAccess: LongWord; InheritHandle: Boolean; const Name: string;
UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TMutex(Boolean UseCOMWait = False);
__fastcall TMutex(PSecurityAttributes MutexAttributes, Boolean InitialOwner, const
UnicodeString Name, Boolean UseCOMWait = False);
__fastcall TMutex(LongWord DesiredAccess, Boolean InheritHandle, const UnicodeString Name,
Boolean UseCOMWait = False);
```

Description

Call Create to create a TMutex (see page 3143) object.

The first version of the constructor creates an unnamed mutex with the default security descriptor, and it establishes that the thread that created the mutex is not its initial owner. Set the `UseCOMWait` parameter to ensure that when a thread is blocked and waiting for the object, any STA COM calls can be made back into this thread.

The second version of the constructor allows opening an existing mutex, identified by its name. The parameters have the following meaning:

Parameter	Meaning
DesiredAccess	The type of access to the mutex. The constructor fails if the requested type of access is not permitted. It can be any combination of the following constants: **MUTEX_ALL_ACCESS**, **MUTEX_MODIFY_STATE**, **SYNCHRONIZE**, **_DELETE**, **READ_CONTROL**, **WRITE_DAC**, **WRITE_OWNER**. Any combination must include the **SYNCHRONIZE** access right, since this is specific to a mutex.
InheritHandle	Set this parameter for the child processes to inherit the mutex handle.
Name	The name of the mutex to open. Names are case-sensitive.
UseCOMWait	Set this parameter to ensure that when a thread is blocked and waiting for the object, any STA COM calls can be made back into this thread.

The third version of the constructor creates a named mutex with several attributes, given through the following list of parameters:

Parameter	Meaning
MutexAttributes	The security attributes of the mutex. `MutexAttributes` is of a pointer type to a **_SECURITY_ATTRIBUTES** record, with the following fields: **nLength**, **lpSecurityDescriptor**, **InheritHandle**. The **nLength** field should always be equal to the size, in bytes, of the **_SECURITY_ATTRIBUTES** record. **lpSecurityDescriptor** is a pointer to the security descriptor of the mutex. Finally, **InheritHandle** is True if the handle to the mutex is to be inherited by child processes.
InitialOwner	Set this for the thread that created the mutex to be considered the initial owner of the mutex.
Name	The name of the mutex. Names are case-sensitive.
UseCOMWait	Set this parameter to ensure that when a thread is blocked and waiting for the object, any STA COM calls can be made back into this thread.

See Also

Acquire (see page 3146)

Release (see page 3146)

34.1.8.3 TMutex Methods

The methods of the TMutex class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3146)	Attempts to acquire a lock on the mutex.
	Release (see page 3146)	Releases the exclusive lock on the mutex.

34.1.8.3.1 TMutex.Acquire Method

Attempts to acquire a lock on the mutex.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use a shared resource.

The Acquire method waits until the mutex becomes free, and then acquires an exclusive lock. This blocks all other threads from acquiring the mutex. Later, calling the Release (see page 3146) method will make the mutex available for other threads to acquire.

See Also

Release (see page 3146)

Acquire (see page 3151)

34.1.8.3.2 TMutex.Release Method

Releases the exclusive lock on the mutex.

Pascal

```
procedure Release; override;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that has finished using a shared resource.

The Release method frees the exclusive lock and allows one of the blocked threads to acquire the mutex.

See Also

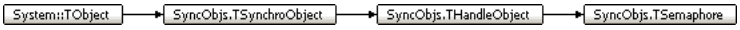
Acquire (see page 3146)

Release (see page 3151)

34.1.9 TSemaphore Class

TSemaphore is a synchronization primitive that manages the concurrent execution of threads, in the context of sharing a common resource.

Class Hierarchy



Pascal

```
TSemaphore = class(THandleObject);
```

C++

```
class TSemaphore : public THandleObject;
```

File

SyncObjs

Description

Use TSemaphore in multi-threaded applications that require safe access to a shared resource.

To each semaphore there is an associated counter variable. This variable gives the number of units of the shared resource that are currently free. The counter is decremented whenever a thread locks the semaphore and incremented when a thread releases the semaphore.

If a thread tries to lock the semaphore and the counter value is nil, then the thread adds to the semaphore waiting queue. The first thread in the semaphore queue locks the semaphore as soon as the counter value is nonzero.

When instantiating the TSemaphore class, the number of initial free units of the shared resource can be specified. This is actually the initial value of the semaphore counter variable.

Also, the number of available units of the shared resource can be specified through the `AMaximumCount` parameter of the TSemaphore class constructor.

Note: If the number of available units of the shared resource is one, then TSemaphore is called a binary semaphore and is basically equivalent to a TMutex (see page 3143).

See Also

Create (see page 3149)

Acquire (see page 3151)

Release (see page 3151)

TCriticalSection (see page 3125)

TMutex (see page 3143)

TEvent (see page 3133)

TThread

THandleObject (see page 3137)

TConditionVariableCS (see page 3112)

TConditionVariableMutex (see page 3120)

TConditionVariableHelper (see page 3117)

TCriticalSectionHelper (see page 3130)

34.1.9.1 TSemaphore Members

The following tables list the members exposed by TSemaphore.

Protected Fields

	Name	Description
	FHandle (see page 3140)	This is FHandle, a member of class THandleObject.
	FLastError (see page 3140)	This is FLastError, a member of class THandleObject.
	FUseCOMWait (see page 3140)	This is FUseCOMWait, a member of class THandleObject.

Public Methods

	Name	Description
V	Acquire (see page 3151)	Attempts to acquire a lock on the semaphore.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3149)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3139)	Destroys an instance of THandleObject (see page 3137).
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	Release (see page 3151)	This is the overview for the Release method overload.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
V	WaitFor (see page 3141)	Waits until this THandleObject (see page 3137) object enters into the signaled state.
	WaitForMultiple (see page 3141)	Waits until a list of THandleObject (see page 3137) objects enters into the signaled state.

Public Properties

	Name	Description
R	Handle (see page 3142)	Specifies the handle of the Windows object used by THandleObject (see page 3137).
R	LastError (see page 3142)	Indicates the Windows error code obtained from Handle (see page 3142).

34.1.9.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TSemaphore.Create (Boolean) (see page 3149)	Instantiates a TSemaphore (see page 3147) object.
	TSemaphore.Create (LongWord, Boolean, string, Boolean) (see page 3149)	Instantiates a TSemaphore (see page 3147) object.
	TSemaphore.Create (PSecurityAttributes, Integer, Integer, string, Boolean) (see page 3149)	Instantiates a TSemaphore (see page 3147) object.

34.1.9.2.1 TSemaphore.Create Constructor (Boolean)

Instantiates a TSemaphore (see page 3147) object.

Pascal

```
constructor Create(UseCOMWait: Boolean = False); overload;
constructor Create(SemaphoreAttributes: PSecurityAttributes; AInitialCount: Integer;
AMaximumCount: Integer; const Name: string; UseCOMWait: Boolean = False); overload;
constructor Create(DesiredAccess: LongWord; InheritHandle: Boolean; const Name: string;
UseCOMWait: Boolean = False); overload;
```

C++

```
__fastcall TSemaphore(Boolean UseCOMWait = False);
__fastcall TSemaphore(PSecurityAttributes SemaphoreAttributes, int AInitialCount, int
AMaximumCount, const UnicodeString Name, Boolean UseCOMWait = False);
__fastcall TSemaphore(LongWord DesiredAccess, Boolean InheritHandle, const UnicodeString Name,
```

```
Boolean UseCOMWait = False);
```

Description

Call Create to create a TSemaphore (see page 3147) object.

The first version of the constructor creates an unnamed binary semaphore with an initial counter value of one. Also, the semaphore is created with the default security descriptor. This means that the child processes of the current process cannot inherit the handle to this semaphore. This is basically equivalent to creating an unnamed TMutex (see page 3143) object. Set the `UseCOMWait` parameter to ensure that when a thread is blocked and waiting for the object, any STA (single-threaded apartment) COM calls can still be made back into this thread.

The second version of the constructor allows opening an existing semaphore, identified by its name. The parameters have the following meaning:

Parameter	Meaning
DesiredAccess	The type of access to the semaphore. The constructor fails if the requested type of access is not permitted. It can be any combination of the following constants: **SEMAPHORE_ALL_ACCESS**, **SEMAPHORE_MODIFY_STATE**, **SYNCHRONIZE**, **_DELETE**, **READ_CONTROL**, **WRITE_DAC**, **WRITE_OWNER.**
InheritHandle	Set this paremeter for the child processes to inherit the handle.
Name	The name of the semaphore to open. Names are case-sensitive.
UseCOMWait	Set this parameter in order to ensure that when a thread is blocked and waiting for the object, any STA COM calls can still be made back into this thread.

The third version of the constructor creates a named semaphore with several attributes, given through the following list of parameters:

Parameter	Meaning
SemaphoreAttributes	The security attributes of the semaphore. `SemaphoreAttributes` is of a pointer type to a **_SECURITY_ATTRIBUTES** record, with the following fields: **nLength**, **lpSecurityDescriptor**, **InheritHandle**. The **nLength** field should always be equal to the size, in bytes, of the **_SECURITY_ATTRIBUTES** record. **lpSecurityDescriptor** is a pointer to the security descriptor of the semaphore. Finally, **InheritHandle** is True if the handle to the semaphore is to be inherited by child processes.
AInitialCount	The initial value of the semaphore counter variable, giving the initial number of free units of the shared resource.
AMaximumCount	The maximum value of the semaphore counter variable, giving the number of available units of the shared resource.
Name	The name of the semaphore. Names are case-sensitive.
UseCOMWait	Set this parameter in order to ensure that when a thread is blocked and waiting for the object, any STA COM calls can still be made back into this thread.

See Also

Acquire (see page 3151)

Release (see page 3151)

34.1.9.3 TSemaphore Methods

The methods of the TSemaphore class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3151)	Attempts to acquire a lock on the semaphore.
	Release (see page 3151)	This is the overview for the Release method overload.

34.1.9.3.1 TSemaphore.Acquire Method

Attempts to acquire a lock on the semaphore.

Pascal

```
procedure Acquire; override;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use some of a shared resource's units.

The Acquire method waits until the semaphore becomes free, that is, the number of free units of the resource is nonzero. Then the Acquire method acquires a lock on the semaphore.

If the semaphore is not free, then the calling thread is added to the semaphore waiting queue.

See Also

Release (see page 3151)

Acquire (see page 3146)

34.1.9.3.2 Release Method

This is the overview for the Release method overload.

Overload List

	Name	Description
	TSemaphore.Release () (see page 3151)	Releases one lock or several locks on the semaphore.
	TSemaphore.Release (Integer) (see page 3151)	Releases one lock or several locks on the semaphore.

34.1.9.3.2.1 TSemaphore.Release Method ()

Releases one lock or several locks on the semaphore.

Pascal

```
procedure Release; override; overload;
function Release(AReleaseCount: Integer): Integer; overload; reintroduce;
```

C++

```
virtual __fastcall Release();
__fastcall int Release(int AReleaseCount);
```

Description

Call Release from a thread that has finished using one or more units of a shared resource.

The Release method releases one or several locks on the semaphore. If several locks are to be freed, they are specified through

the `AReleaseCount` parameter.

If the number of free units of the shared resource plus the value of the `AReleaseCount` parameter exceeds the number of available units of the shared resource, specified through the `AMaximumCount` parameter of the constructor, then the semaphore counter variable is left unchanged and the Release function raises an error.

See Also

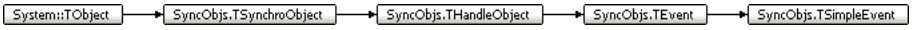
Acquire (see page 3151)

Release (see page 3146)

34.1.10 TSimpleEvent Class

TSimpleEvent represents an unnamed manual event object.

Class Hierarchy



Pascal

```
TSimpleEvent = class(TEvent);
```

C++

```
class TSimpleEvent : public TEvent;
```

File

SyncObjs

Description

Use TSimpleEvent to coordinate between threads that all have access to the same simple event object. TSimpleEvent can also be used in a single-threaded application to coordinate sections of code that respond to different asynchronous events such as system messages or user actions.

Because TSimpleEvent represents an unnamed event object, a single TSimpleEvent object must be within the scope of all code that uses it. To coordinate with code that is out of scope, or with other applications, use a named TEvent (see page 3133) object.

See Also

TThread

TEvent (see page 3133)

Waiting for a Task to Be Completed

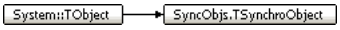
Procedural types

Creating events

34.1.11 TSynchroObject Class

TSynchroObject is the base class for all thread synchronization objects.

Class Hierarchy



Pascal

```
TSynchroObject = class(TObject);
```

C++

```
class TSynchroObject : public TObject;
```

File

SyncObjs

Description

Use TSynchroObject as a base class when defining an object that coordinates the execution of different threads in a multi-threaded application. Synchronization objects include critical sections that temporarily block the execution of other threads which may interfere with the successful completion of a section of code, and event objects that allow threads to signal when critical events have occurred.

See Also

TCriticalSection (see page 3125)

TEvent (see page 3133)

TThread

34.1.11.1 TSynchroObject Members

The following tables list the members exposed by TSynchroObject.

Public Methods

	Name	Description
	Acquire (see page 3154)	Provides the interface for a method that devotes the synchronization object to a thread.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.

	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Release (see page 3154)	Provides the interface for a method that releases the synchronization object for use by other threads.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

34.1.11.2 TSynchroObject Methods

The methods of the TSynchroObject class are listed here.

Public Methods

	Name	Description
	Acquire (see page 3154)	Provides the interface for a method that devotes the synchronization object to a thread.
	Release (see page 3154)	Provides the interface for a method that releases the synchronization object for use by other threads.

34.1.11.2.1 TSynchroObject.Acquire Method

Provides the interface for a method that devotes the synchronization object to a thread.

Pascal

```
procedure Acquire; virtual;
```

C++

```
virtual __fastcall Acquire();
```

Description

Call Acquire from the thread that needs to use the synchronization object. The Acquire method for TSynchroObject (see page 3152) does nothing other than provide the interface for a method that can start the synchronization object working on behalf of the calling thread.

See Also

Release (see page 3154)

34.1.11.2.2 TSynchroObject.Release Method

Provides the interface for a method that releases the synchronization object for use by other threads.

Pascal

```
procedure Release; virtual;
```

C++

```
virtual __fastcall Release();
```

Description

Call Release from a thread that is finished using the synchronization object. The Release method for TSynchroObject (see page 3152) does nothing other than provide the interface for a method that can unbind the synchronization object from the calling thread.

See Also

Acquire (see page 3154)

34.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWaitResult (see page 3156)	TWaitResult indicates the reason an event object stopped waiting for its signal.

34.2.1 SyncObjs.TWaitResult Enumeration

TWaitResult indicates the reason an event object stopped waiting for its signal.

Pascal

```
TWaitResult = (
  wrSignaled,
  wrTimeout,
  wrAbandoned,
  wrError,
  wrIOCompletion
);
```

C++

```
enum TWaitResult {
  wrSignaled,
  wrTimeout,
  wrAbandoned,
  wrError,
  wrIOCompletion
};
```

File

SyncObjs

Description

TWaitResult includes the following values:

Value	Meaning
wrSignaled	The signal of the event object was set.
wrTimeOut	The time specified by the TimeOut parameter elapsed without the signal being set.
wrAbandoned	The event object was destroyed before the TimeOut period elapsed.
wrError	An error occurred while waiting. Check the LastError property for an error code giving more information.

34.3 Types

The following table lists types in this documentation.

Types

Name	Description
THandleObjectArray (see page 3157)	THandleObjectArray represents an array of THandleObject (see page 3137).

34.3.1 SyncObjs.THandleObjectArray Type

THandleObjectArray represents an array of THandleObject (see page 3137).

Pascal

```
THandleObjectArray = array of THandleObject;
```

C++

```
array of THandleObject THandleObjectArray;
```

File

SyncObjs

Description

THandleObjectArray is used by synchronization objects to store and manage objects of type THandleObject (see page 3137).

See Also

THandleObject (see page 3137)

35 System Namespace

This is namespace System.

Topics

Name	Description
Classes (see page 3174)	The following table lists classes in this documentation.
Interfaces (see page 3493)	The following table lists interfaces in this documentation.
Functions (see page 3506)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 3712)	The following table lists structs, records, enums in this documentation.
Types (see page 3750)	The following table lists types in this documentation.
Variables (see page 3806)	The following table lists variables in this documentation.
Constants (see page 3842)	The following table lists constants in this documentation.
Macros (see page 3892)	The following table lists macros in this documentation.

Classes

	Name	Description
	AnsiStringBase (see page 3174)	System::AnsiStringBase is the base class for the C++ analog of the Delphi long string type.
	AnsiStringT (see page 3207)	System::AnsiStringT is the C++ analog for the Delphi long string type.
	Currency (see page 3235)	System::Currency is the C++ implementation of the Delphi System::Currency data type.
	CurrencyBase (see page 3255)	This is class System::CurrencyBase.
	DelphiInterface (see page 3256)	System::DelphiInterface is a C++ implementation of the Delphi interface.
	DynamicArray (see page 3261)	System::DynamicArray is the C++ analog for the Delphi dynamic array type.
	DynArrayException (see page 3271)	This is class System::DynArrayException.
	DynArrayNullData (see page 3272)	This is class System::DynArrayNullData.
	DynArrayOutOfRange (see page 3272)	This is class System::DynArrayOutOfRange.
	OleVariant (see page 3274)	System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.
	SmallString (see page 3307)	This is class System::SmallString.
	SmallStringBase (see page 3310)	This is class System::SmallStringBase.
	sPointer (see page 3311)	This is class System::sPointer.
	TAggregatedObject (see page 3313)	System::TAggregatedObject provides the functionality for an inner object of an aggregate by implementing the IInterface (see page 3502) methods to delegate to the controlling IInterface (see page 3502).
	TClassHelperBase (see page 3318)	This is class System.TClassHelperBase.
	TContainedObject (see page 3320)	System::TContainedObject implements the IInterface (see page 3502) interface as an inner object in an aggregate.

	TDateTime (see page 3323)	System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.
	TDateTimeBase (see page 3341)	This is class System::TDateTimeBase.
	TInterfacedObject (see page 3342)	System::TInterfacedObject is a base for simple classes that need a basic IInterface (see page 3502) implementation.
	TMetaClass (see page 3348)	System::TMetaClass is the C++ representation of the Delphi class-reference type.
	TMonitor (see page 3349)	TMonitor provides methods for synchronizing the access of several threads to a single object.
	TObject (see page 3354)	System::TObject is the ultimate ancestor of all objects and components.
	UnicodeString (see page 3381)	System::UnicodeString is the C++ analog for the Delphi UnicodeString type.
	Variant (see page 3421)	System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.
	WideString (see page 3467)	System::WideString represents strings of 16-bit characters.

Constants

Name	Description
CompilerVersion (see page 3844)	Designates the version number of the Delphi compiler.
CPUi386 (see page 3844)	This is constant System.CPUi386.
CPUi486 (see page 3845)	This is constant System.CPUi486.
CPUPentium (see page 3845)	This is constant System.CPUPentium.
E_NOINTERFACE (see page 3845)	This is constant System.E_NOINTERFACE.
E_NOTIMPL (see page 3846)	This is constant System.E_NOTIMPL.
E_UNEXPECTED (see page 3846)	This is constant System.E_UNEXPECTED.
fmClosed (see page 3846)	This is constant System.fmClosed.
fmInOut (see page 3847)	This is constant System.fmInOut.
fmInput (see page 3847)	This is constant System.fmInput.
fmOutput (see page 3847)	This is constant System.fmOutput.
hfFieldSize (see page 3848)	This is constant System.hfFieldSize.
hfMonitorOffset (see page 3848)	This is constant System.hfMonitorOffset.
INFINITE (see page 3848)	This is constant System.INFINITE.
MaxInt (see page 3849)	The maximum value of the Integer data type.
MaxLongint (see page 3849)	The maximum value of the Longint data type.
NumSmallBlockTypes (see page 3849)	NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.
opAdd (see page 3850)	opAdd defines an addition operation.
opAnd (see page 3850)	opAnd defines a logical and operation.
opCmpEQ (see page 3851)	opCmpEQ defines an equality comparison operator.
opCmpGE (see page 3851)	opCmpGE defines a greater than or equal comparison operator.
opCmpGT (see page 3852)	opCmpGT defines a greater than comparison operator.
opCmpLE (see page 3852)	opCmpLE defines a less than or equal comparison operator.
opCmpLT (see page 3853)	opCmpLT defines a less than comparison operator.
opCmpNE (see page 3853)	opCmpNE defines an inequality comparison operator.
opCompare (see page 3854)	opCompare defines a comparison operation.
opDivide (see page 3854)	opDivide defines a divide operation.
opIntDivide (see page 3855)	opIntDivide defines an integer divide operation.
opModulus (see page 3855)	opModulus defines a modulus operation.
opMultiply (see page 3855)	opMultiply defines a multiply operation.
opNegate (see page 3856)	opNegate defines a negate operation.
opNot (see page 3856)	opNot defines a binary negation operation.
opOr (see page 3857)	opOr defines a logical or operation.
opShiftLeft (see page 3857)	opShiftLeft defines a bitwise shift left operation.
opShiftRight (see page 3858)	opShiftRight defines a bitwise shift right operation.
opSubtract (see page 3858)	opSubtract defines a subtraction operation.
opXor (see page 3859)	opXor defines a logical exclusive or operation.
RTLVersion (see page 3859)	Designates the version number of the Delphi runtime library.
S_FALSE (see page 3859)	This is constant System.S_FALSE.

S_OK (see page 3860)	This is constant System.S_OK.
sLineBreak (see page 3860)	Represents the character(s) used to generate a line feed.
tfCRLF (see page 3861)	This is constant System.tfCRLF.
varAny (see page 3861)	Represents a Variant (see page 3421) that can hold any value.
varArray (see page 3861)	Represents a Variant (see page 3421) that holds dynamic array data.
varBoolean (see page 3862)	Represents a Variant (see page 3421) that holds WordBool data.
varByRef (see page 3862)	Represents a Variant (see page 3421) that points to another variable.
varByte (see page 3863)	Represents a Variant (see page 3421) that holds byte data.
varCurrency (see page 3863)	Represents a Variant (see page 3421) that holds a monetary floating point value.
varDate (see page 3864)	Represents a Variant (see page 3421) that holds a TDateTime (see page 3323) value.
varDispatch (see page 3864)	Represents a Variant (see page 3421) that has been assigned an IDispatch (see page 3494) value.
varDouble (see page 3865)	Represents a Variant (see page 3421) that holds a floating point value.
varEmpty (see page 3865)	Represents a Variant (see page 3421) that has not yet been assigned a value.
varError (see page 3866)	Represents a Variant (see page 3421) that holds a 32 bit error code.
varInt64 (see page 3867)	Represents a Variant (see page 3421) that holds Int64 data.
varInteger (see page 3867)	Represents a Variant (see page 3421) that holds Integer data.
varLongWord (see page 3868)	Represents a Variant (see page 3421) that holds LongWord data.
varNull (see page 3868)	Represents a Variant (see page 3421) that holds a null value.
varOleStr (see page 3869)	Represents a Variant (see page 3421) that has been assigned a COM interface type string value.
varShortInt (see page 3869)	Represents a Variant (see page 3421) that holds ShortInt data.
varSingle (see page 3870)	Represents a Variant (see page 3421) that holds byte data.
varSmallint (see page 3870)	Represents a Variant (see page 3421) that holds Smallint data.
varStrArg (see page 3871)	Represents a Variant (see page 3421) that has holds a COM compatible string value.
varString (see page 3871)	Represents a Variant (see page 3421) that holds string data.
varTypeMask (see page 3872)	Defines a mask for filtering out the meta data part of a Variant (see page 3421) data type.
varUInt64 (see page 3872)	This is constant System.varUInt64.
varUnknown (see page 3873)	Represents a Variant (see page 3421) that holds an unknown value.
varUString (see page 3873)	This is constant System.varUString.
varVariant (see page 3873)	Represents a Variant (see page 3421) data type.
varWord (see page 3874)	Represents a Variant (see page 3421) that holds Word data.
vmtAddRef (see page 3874)	This is constant System.vmtAddRef.
vmtAfterConstruction (see page 3875)	This is constant System.vmtAfterConstruction.
vmtAutoTable (see page 3875)	This is constant System.vmtAutoTable.
vmtBeforeDestruction (see page 3875)	This is constant System.vmtBeforeDestruction.
vmtClassName (see page 3876)	This is constant System.vmtClassName.
vmtCreateObject (see page 3876)	This is constant System.vmtCreateObject.
vmtDefaultHandler (see page 3876)	This is constant System.vmtDefaultHandler.
vmtDestroy (see page 3877)	This is constant System.vmtDestroy.
vmtDispatch (see page 3877)	This is constant System.vmtDispatch.
vmtDynamicTable (see page 3877)	This is constant System.vmtDynamicTable.
vmtEquals (see page 3878)	This is constant System.vmtEquals.
vmtFieldTable (see page 3878)	This is constant System.vmtFieldTable.
vmtFreeInstance (see page 3878)	This is constant System.vmtFreeInstance.
vmtGetHashCode (see page 3879)	This is constant System.vmtGetHashCode.
vmtInitTable (see page 3879)	This is constant System.vmtInitTable.
vmtInstanceSize (see page 3879)	This is constant System.vmtInstanceSize.
vmtIntfTable (see page 3880)	This is constant System.vmtIntfTable.
vmtMethodTable (see page 3880)	This is constant System.vmtMethodTable.
vmtNewInstance (see page 3880)	This is constant System.vmtNewInstance.
vmtParent (see page 3881)	This is constant System.vmtParent.

vmtQueryInterface (see page 3881)	This is constant System.vmtQueryInterface.
vmtRelease (see page 3881)	This is constant System.vmtRelease.
vmtSafeCallException (see page 3882)	This is constant System.vmtSafeCallException.
vmtSelfPtr (see page 3882)	This is constant System.vmtSelfPtr.
vmtToString (see page 3882)	This is constant System.vmtToString.
vmtTypeInfo (see page 3883)	This is constant System.vmtTypeInfo.
vtAnsiString (see page 3883)	vtAnsiString defines an AnsiString TVarData (see page 3740) VType value.
vtBoolean (see page 3883)	vtBoolean defines a Boolean TVarData (see page 3740) VType value.
vtChar (see page 3884)	vtChar defines a Char TVarData (see page 3740) VType value.
vtClass (see page 3884)	vtClass defines a TClass (see page 3785) TVarData (see page 3740) VType value.
vtCurrency (see page 3885)	vtCurrency defines a Currency (see page 3235) TVarData (see page 3740) VType value.
vtExtended (see page 3885)	vtExtended defines an Extended TVarData (see page 3740) VType value.
vtInt64 (see page 3886)	vtInt64 defines an Int64 TVarData (see page 3740) VType value.
vtInteger (see page 3886)	vtInteger defines an Integer TVarData (see page 3740) VType value.
vtInterface (see page 3887)	vtInterface defines an Interface pointer TVarData (see page 3740) VType value.
vtObject (see page 3887)	vtObject defines a TObject (see page 3354) TVarData (see page 3740) VType value.
vtPChar (see page 3888)	vtPChar defines a PChar TVarData (see page 3740) VType value.
vtPointer (see page 3888)	vtPointer defines a Pointer TVarData (see page 3740) VType value.
vtPWideChar (see page 3889)	vtPWideChar defines a PWideChar TVarData (see page 3740) VType value.
vtString (see page 3889)	vtString defines a String TVarData (see page 3740) VType value.
vtUnicodeString (see page 3890)	This is constant System.vtUnicodeString.
vtVariant (see page 3890)	vtString (see page 3889) defines a Variant (see page 3421) TVarData (see page 3740) VType value.
vtWideChar (see page 3890)	vtWideChar defines a WideChar TVarData (see page 3740) VType value.
vtWideString (see page 3891)	vtWideString defines a WideString (see page 3467) TVarData (see page 3740) VType value.

Functions

	Name	Description
	Abs (see page 3511)	Returns an absolute value.
	AcquireExceptionObject (see page 3512)	Retains access to exception object.
	AddModuleUnloadProc (see page 3514)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	AddModuleUnloadProc (see page 3514)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	Addr (see page 3514)	Returns a pointer to a specified object.
	AllocMem (see page 3515)	AllocMem allocates a memory block and initializes each byte to zero.
	AnsiToUtf8 (see page 3517)	Converts a string encoded in Ansi to UTF-8.
	Append (see page 3520)	Prepares an existing file for adding text to the end.
	ArcTan (see page 3521)	Calculates the arctangent of a given number.
	Assert (see page 3522)	Tests whether a Boolean expression is true.
	Assert (see page 3522)	Tests whether a Boolean expression is true.
	Assigned (see page 3524)	Tests for a nil (unassigned) pointer or procedural variable.
	AssignFile (see page 3525)	Associates the name of an external file with a file variable.
	AttemptToUseSharedMemoryManager (see page 3527)	Attempts to use a shared memory manager.
	BeginThread (see page 3527)	Spawns a separate thread of execution.
	BlockRead (see page 3528)	Reads one or more records from an open file into a variable.
	BlockRead (see page 3528)	Reads one or more records from an open file into a variable.
	BlockWrite (see page 3530)	Writes one or more records from a variable to an open file.
	BlockWrite (see page 3530)	Writes one or more records from a variable to an open file.
	Break (see page 3531)	Causes the flow of control to exit a for, while, or repeat statement.
	ChDir (see page 3534)	Changes the current directory.
	ChDir (see page 3534)	Changes the current directory.

	CheckSafecallResult (see page 3536)	This is function System::CheckSafecallResult.
	Chr (see page 3536)	Returns the character for a specified ASCII value.
	Close (see page 3538)	Terminates the association between a file variable and an external file.
	CloseFile (see page 3539)	Terminates the association between file variable and an external disk file.
	CompToCurrency (see page 3542)	Converts a **Comp** value to a Currency (see page 3235) value.
	CompToDouble (see page 3543)	Converts a Comp value to a double value.
	Concat (see page 3545)	Concatenates two or more strings into one.
	Concat (see page 3545)	Concatenates two or more strings into one.
	Concat (see page 3545)	Concatenates two or more strings into one.
	Continue (see page 3546)	Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.
	Copy (see page 3547)	Returns a substring of a string or a segment of a dynamic array.
	Cos (see page 3549)	Calculates the cosine of an angle.
	CurrencyToComp (see page 3550)	Converts a Currency (see page 3235) value to a Comp value.
	Dec (see page 3550)	Decrements a variable by 1 or N.
	Dec (see page 3550)	Decrements a variable by 1 or N.
	Delete (see page 3552)	Removes a substring from a string.
	Dispose (see page 3553)	Releases memory allocated for a dynamic variable.
	DoubleToComp (see page 3555)	Converts a double value to a Comp.
	DynArrayClear (see page 3555)	This is function System.DynArrayClear.
	DynArrayDim (see page 3555)	This is function System.DynArrayDim.
	DynArraySetLength (see page 3556)	This is function System.DynArraySetLength.
	EndThread (see page 3556)	Terminates the execution of a thread.
	EnumModules (see page 3556)	Executes a callback for all the modules in an application.
	EnumModules (see page 3556)	Executes a callback for all the modules in an application.
	EnumResourceModules (see page 3557)	Executes a callback for all resource modules in an application.
	EnumResourceModules (see page 3557)	Executes a callback for all resource modules in an application.
	Eof (see page 3558)	Tests whether the file position is at the end of a file.
	Eof (see page 3558)	Tests whether the file position is at the end of a file.
	Eof (see page 3558)	Tests whether the file position is at the end of a file.
	Eoln (see page 3559)	Tests whether the file pointer is at the end of a line.
	Eoln (see page 3559)	Tests whether the file pointer is at the end of a line.
	Erase (see page 3560)	Deletes an external file.
	Error (see page 3560)	Error is used to generate a run-time exception.
	ExceptAddr (see page 3562)	Returns the address at which the current exception was raised.
	ExceptObject (see page 3562)	Returns current exception object.
	Exclude (see page 3564)	Removes an element from a Delphi set or removes an event handler in .NET applications.
	Exit (see page 3565)	Exits from the current procedure.
	Exp (see page 3567)	Returns the exponential of X.
	Exp (see page 3567)	Returns the exponential of X.
	FilePos (see page 3567)	Returns the current file position.
	FileSize (see page 3569)	Returns the number of records in a file.
	FillChar (see page 3570)	Fills contiguous bytes with a specified value.
	Finalize (see page 3572)	Uninitializes a dynamically allocated variable.
	Finalize (see page 3572)	Uninitializes a dynamically allocated variable.
	FindClassHInstance (see page 3573)	Returns the instance handle for the module in which a class type is defined.
	FindDynaInst (see page 3573)	This is function System::FindDynaInst.
	FindHInstance (see page 3574)	Returns the instance handle of the module that contains a specified address.
	FindResourceHInstance (see page 3574)	Returns the instance handle of the resource module associated with a specified `HInstance`.
	Flush (see page 3575)	Empties the buffer of a text file opened for output.
	FPower10 (see page 3576)	This is function System.FPower10.
	Frac (see page 3576)	Returns the fractional part of a real number.
	FreeMem (see page 3577)	FreeMem frees a memory block.

	FreeMem (see page 3577)	FreeMem frees a memory block.
	FreeMemory (see page 3579)	FreeMemory frees a memory block.
	Get8087CW (see page 3580)	Returns the value of the 8087 code word.
	GetDir (see page 3580)	Returns the current directory.
	GetHeapStatus (see page 3581)	Returns the current status of the memory manager.
	GetLastError (see page 3582)	Returns the last error reported by an operating system API call.
	GetMem (see page 3584)	GetMem allocates a memory block.
	GetMemory (see page 3585)	GetMemory allocates a memory block.
	GetMemoryManager (see page 3587)	Returns the entry points of the currently installed memory manager.
	GetMemoryManager (see page 3587)	Returns the entry points of the currently installed memory manager.
	GetMemoryManagerState (see page 3591)	Fetches state information for the Memory Manager.
	GetMemoryMap (see page 3592)	Fetches the memory map.
	GetMinimumBlockAlignment (see page 3592)	Gets the minimum memory block alignment.
	GetModuleFileName (see page 3593)	Returns the fully qualified name for a module, given its handle.
	GetVariantManager (see page 3594)	Returns the entry points of the routines that define variant behavior.
	Halt (see page 3594)	Initiates abnormal termination of a program.
	Halt (see page 3594)	Initiates abnormal termination of a program.
	Hi (see page 3595)	Returns the high-order byte of X as an unsigned value.
	High (see page 3596)	Returns the highest value in the range of an argument.
	Inc (see page 3597)	Increments an ordinal value by one or N.
	Inc (see page 3597)	Increments an ordinal value by one or N.
	Include (see page 3598)	Adds an element to a Delphi set or adds an event handler in .NET applications.
	Initialize (see page 3599)	Initializes a dynamically allocated variable.
	Initialize (see page 3599)	Initializes a dynamically allocated variable.
	Insert (see page 3601)	Inserts a substring into a string beginning at a specified point.
	Int (see page 3602)	Returns the integer part of a real number.
	IOResult (see page 3602)	Returns the status of the last I/O operation performed.
	IsMemoryManagerSet (see page 3604)	Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 3660) procedure.
	IsVariantManagerSet (see page 3606)	Indicates whether variant support is implemented in an application.
	Length (see page 3607)	Returns the number of characters in a string or elements in an array.
	Ln (see page 3608)	Returns the natural log of a real expression.
	Lo (see page 3609)	Returns the low order Byte of argument X.
	LoadResourceModule (see page 3609)	This is function System.LoadResourceModule.
	LoadResourceString (see page 3610)	This is function System::LoadResourceString.
	LoadResString (see page 3610)	This is function System.LoadResString.
	Low (see page 3610)	Returns the lowest value in a range.
	Mark (see page 3612)	This is function System.Mark.
	MemoryBarrier (see page 3612)	This is function System.MemoryBarrier.
	MkDir (see page 3612)	Creates a new subdirectory.
	MkDir (see page 3612)	Creates a new subdirectory.
	MonitorEnter (see page 3614)	Prohibits the access of all other threads but the calling one to the specified object.
	MonitorExit (see page 3614)	Releases a lock of the calling thread on the specified object.
	MonitorPulse (see page 3615)	Notifies the next thread in the waiting queue that it will be able to lock the specified object as soon as the calling thread releases it.
	MonitorPulseAll (see page 3615)	Signals all threads in the waiting queue that either one of them will be able to lock the specified object as soon as the calling thread releases it.
	MonitorTryEnter (see page 3616)	Attempts to lock the specified object, so that only the calling thread is able to access it.
	MonitorWait (see page 3616)	Drops the lock on the specified object and puts the calling thread in the waiting queue for a specified amount of time.
	Move (see page 3617)	Copies bytes from a source to a destination.
	MoveChars (see page 3618)	Copies bytes from a source to a destination.
	New (see page 3619)	Creates a new dynamic variable and sets P to point to it.
	Odd (see page 3621)	Returns true if argument is an odd number.
	OleStrToString (see page 3622)	Copies data received from a COM interface to a string.

	OleStrToStrVar (see page 3622)	Copies a string from its COM representation to an existing Delphi string.
	OleStrToStrVar (see page 3622)	Copies a string from its COM representation to an existing Delphi string.
	Ord (see page 3623)	Returns the ordinal value of an ordinal-type expression.
	ParamCount (see page 3623)	Returns the number of parameters passed on the command line.
	ParamStr (see page 3625)	Returns a specified parameter from the command-line.
	Pi (see page 3626)	Returns 3.1415926535897932385.
	Pos (see page 3627)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3627)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3627)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pred (see page 3628)	Returns the predecessor of the argument.
	Ptr (see page 3630)	Converts a specified address to a pointer.
	PUCS4Chars (see page 3632)	Converts a UCS4 string to a null-terminated array of UCS4 characters.
	RaiseList (see page 3632)	This is function System.RaiseList.
	Random (see page 3632)	Generates random numbers within a specified range.
	Random (see page 3632)	Generates random numbers within a specified range.
	Randomize (see page 3633)	Initializes the random number generator with a random value.
	Read (see page 3634)	Read reads data from a file.
	Read (see page 3634)	Read reads data from a file.
	Read (see page 3634)	Read reads data from a file.
	Read (see page 3634)	Read reads data from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReallocMem (see page 3637)	ReallocMem reallocates a memory block.
	ReallocMemory (see page 3638)	ReallocMemory resizes a memory block.
	RegisterExpectedMemoryLeak (see page 3640)	Registers an expected memory leak.
	RegisterModule (see page 3640)	Registers the specified module.
	Release (see page 3641)	This is function System.Release.
	ReleaseExceptionObject (see page 3641)	Releases exception object acquired by AcquireExceptionObject (see page 3512).
	RemoveModuleUnloadProc (see page 3641)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	RemoveModuleUnloadProc (see page 3641)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	Rename (see page 3642)	Changes the name of an external file.
	Rename (see page 3642)	Changes the name of an external file.
	Reset (see page 3643)	Opens an existing file.
	Reset (see page 3643)	Opens an existing file.
	Reset (see page 3643)	Opens an existing file.
	Reset (see page 3643)	Opens an existing file.
	Rewrite (see page 3645)	Creates a new file and opens it.
	Rewrite (see page 3645)	Creates a new file and opens it.
	RmDir (see page 3647)	Deletes an empty subdirectory.
	RmDir (see page 3647)	Deletes an empty subdirectory.
	Round (see page 3648)	Returns the value of X rounded to the nearest whole number.
	RunError (see page 3649)	Stops execution and generates a runtime error.
	RunError (see page 3649)	Stops execution and generates a runtime error.
	Seek (see page 3650)	Moves the current position of a file to a specified component.
	SeekEof (see page 3651)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEof (see page 3651)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEoln (see page 3652)	Returns the end-of-line status of a file, ignoring whitespace.
	SeekEoln (see page 3652)	Returns the end-of-line status of a file, ignoring whitespace.

	Set8087CW (see page 3653)	Sets both the control word in the floating point unit and the variable NoErrMsg (see page 3832) declared in the system unit.
	SetCodePage (see page 3655)	This is function System.SetCodePage.
	SetInOutRes (see page 3655)	This is function System.SetInOutRes.
	SetLastError (see page 3656)	SetLastError sets the last API error code.
	SetLength (see page 3657)	Sets the length of a string or dynamic-array variable.
	SetLineBreakStyle (see page 3659)	Determines the end-of-line and end-of-file conventions for text file I/O.
	SetMemoryManager (see page 3660)	Sets entry points of the memory manager.
	SetMemoryManager (see page 3660)	Sets entry points of the memory manager.
	SetMinimumBlockAlignment (see page 3664)	Sets the minimum memory block alignment.
	SetMultiByteConversionCodePage (see page 3665)	Sets the code page used internally whenever multibyte characters are converted to/from WideChar.
	SetRaiseList (see page 3665)	This is function System.SetRaiseList.
	SetString (see page 3665)	Sets the contents and length of the given string.
	SetTextBuf (see page 3667)	Assigns an I/O buffer to a text file.
	SetTextBuf (see page 3667)	Assigns an I/O buffer to a text file.
	SetVariantManager (see page 3668)	Sets the entry points of routines that define variant behavior.
	ShareMemoryManager (see page 3668)	Shares the module's memory manager.
	Sin (see page 3669)	Returns the sine of the angle in radians.
	SizeOf (see page 3670)	Returns the number of bytes occupied by a variable or type.
	Slice (see page 3671)	Returns a sub-section of an array.
	Sqr (see page 3674)	Returns the square of a number.
	Sqr (see page 3674)	Returns the square of a number.
	Sqrt (see page 3675)	Returns the square root of X.
	Str (see page 3676)	Formats a string and returns it to a variable.
	Str (see page 3676)	Formats a string and returns it to a variable.
	StringCodePage (see page 3676)	Returns the code page associated with the string.
	StringCodePage (see page 3676)	Returns the code page associated with the string.
	StringElementSize (see page 3677)	Returns the number of bytes that represent a character in the string.
	StringElementSize (see page 3677)	Returns the number of bytes that represent a character in the string.
	StringOfChar (see page 3677)	Returns a string with a specified number of repeating characters.
	StringOfChar (see page 3677)	Returns a string with a specified number of repeating characters.
	StringRefCount (see page 3678)	Returns the reference count of a string.
	StringRefCount (see page 3678)	Returns the reference count of a string.
	StringToOleStr (see page 3678)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToOleStr (see page 3678)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToWideChar (see page 3679)	Returns a UNICODE string from an AnsiString.
	Succ (see page 3680)	Returns the successor of an argument.
	Swap (see page 3680)	Exchanges high order byte with the low order byte of an integer or word.
	SysAllocMem (see page 3681)	This is function System.SysAllocMem.
	SysFreeMem (see page 3681)	Frees the memory pointed to by a specified pointer.
	SysGetMem (see page 3682)	Allocates a specified number of bytes and returns a pointer to them.
	SysReallocMem (see page 3683)	Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.
	SysRegisterExpectedMemoryLeak (see page 3683)	This is function System.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 3684)	This is function System.SysUnregisterExpectedMemoryLeak.
	TextStart (see page 3684)	This is function System.TextStart.
	Trunc (see page 3684)	Truncates a real number to an integer.
	Truncate (see page 3685)	Deletes all the records after the current file position.
	UCS4StringToUnicodeString (see page 3686)	Converts an UCS-4 encoded string to an Unicode string.
	UCS4StringToWideString (see page 3687)	Converts a string of 4-byte Unicode characters into a WideString (see page 3467) value.
	UnicodeStringToUCS4String (see page 3689)	Converts an Unicode string to its UCS-4 encoded representation.
	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.

	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.
	UniqueString (see page 3693)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3693)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3693)	Ensures that a given string has a reference count of one.
	UnregisterExpectedMemoryLeak (see page 3694)	Unregisters an expected memory leak.
	UnregisterModule (see page 3694)	Unregisters the specified module.
	UpCase (see page 3695)	Converts a character to uppercase.
	UpCase (see page 3695)	Converts a character to uppercase.
	UTF8Decode (see page 3696)	Converts a UTF8 string to a Unicode string (WideString (see page 3467)).
	UTF8Encode (see page 3696)	Converts a WideString (see page 3467) value to a UTF8 string.
	UTF8Encode (see page 3696)	Converts a WideString (see page 3467) value to a UTF8 string.
	UTF8Encode (see page 3696)	Converts a WideString (see page 3467) value to a UTF8 string.
	UTF8EncodeToShortString (see page 3697)	Converts a string to its UTF-8 encoded representation.
	UTF8EncodeToShortString (see page 3697)	Converts a string to its UTF-8 encoded representation.
	UTF8EncodeToShortString (see page 3697)	Converts a string to its UTF-8 encoded representation.
	Utf8ToAnsi (see page 3698)	Converts a UTF8 string to a string of Ansi characters.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	Utf8ToUnicode (see page 3699)	Converts a string of UTF-8 characters to a Unicode string.
	Utf8ToUnicode (see page 3699)	Converts a string of UTF-8 characters to a Unicode string.
	UTF8ToUnicodeString (see page 3700)	Converts an UTF-8 encoded string to an Unicode string.
	UTF8ToUnicodeString (see page 3700)	Converts an UTF-8 encoded string to an Unicode string.
	UTF8ToUnicodeString (see page 3700)	Converts an UTF-8 encoded string to an Unicode string.
	UTF8ToWideString (see page 3700)	Converts an UTF-8 encoded string to a WideString (see page 3467).
	Val (see page 3701)	Converts a string to a numeric representation.
	VarArrayRedim (see page 3702)	Resizes a Variant (see page 3421) array.
	VarClear (see page 3704)	Empties a Variant (see page 3421), so that it is unassigned.
	WideCharLenToString (see page 3705)	Converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)**.
	WideCharLenToStrVar (see page 3706)	WideCharLenToStrVar converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)** or **AnsiString**.
	WideCharLenToStrVar (see page 3706)	WideCharLenToStrVar converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)** or **AnsiString**.
	WideCharToString (see page 3706)	Converts null-terminated **WideString (see page 3467)** string to **UnicodeString (see page 3381)**.
	WideCharToStrVar (see page 3707)	Converts **WideString (see page 3467)** string to **UnicodeString (see page 3381)**.
	WideStringToUCS4String (see page 3707)	Converts a WideString (see page 3467) value to a string of 4-byte Unicode characters.
	Write (see page 3707)	Writes to either a typed file or a text file
	Write (see page 3707)	Writes to either a typed file or a text file
	Write (see page 3707)	Writes to either a typed file or a text file
	Write (see page 3707)	Writes to either a typed file or a text file
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	YieldProcessor (see page 3711)	This is function System.YieldProcessor.

Interfaces

	Name	Description
	IComparable (see page 3493)	IComparable is the generic interface used to compare two objects.
	IDispatch (see page 3494)	IDispatch is the interface used to support Automation.
	IEnumerable (see page 3497)	IEnumerable is the generic interface for enumerable containers.

	IEnumerator (see page 3498)	IEnumerator is the generic interface for container enumerators.
	IEquatable (see page 3500)	IEquatable is the generic interface used to check the equality of two objects.
	IInterface (see page 3502)	IInterface is the base class for all interfaces defined in Delphi code.
	IInvokable (see page 3504)	IInvokable is the base class for invokable interfaces.

Macros

Name	Description
USE (see page 3892)	This is macro System::USE.
USEADDITIONALFILES (see page 3892)	This is macro System::USEADDITIONALFILES.
USEASM (see page 3893)	This is macro System::USEASM.
USEDATAMODULE (see page 3893)	This is macro System::USEDATAMODULE.
USEDATAMODULENS (see page 3893)	This is macro System::USEDATAMODULENS.
USEDEF (see page 3894)	This is macro System::USEDEF.
USEFILE (see page 3894)	This is macro System::USEFILE.
USEFORM (see page 3895)	This is macro System::USEFORM.
USEFORMNS (see page 3895)	This is macro System::USEFORMNS.
USEFORMRES (see page 3895)	This is macro System::USEFORMRES.
USEIDL (see page 3896)	This is macro System::USEIDL.
USELIB (see page 3896)	This is macro System::USELIB.
USEOBJ (see page 3896)	This is macro System::USEOBJ.
USEPACKAGE (see page 3897)	This is macro System::USEPACKAGE.
USERC (see page 3897)	This is macro System::USERC.
USERES (see page 3897)	This is macro System::USERES.
USETLB (see page 3898)	This is macro System::USETLB.
USEUNIT (see page 3898)	This is macro System::USEUNIT.

Structs, Records, Enums

	Name	Description
	PackageInfoTable (see page 3713)	A variable of type PackageInfoTable provides information on initializing or finalizing a series of package units.
	PackageUnitEntry (see page 3713)	A variable of type PackageUnitEntry provides information on initializing or finalizing a package unit.
	TCallDesc (see page 3714)	TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 3758) is a pointer to this descriptior.
	TChunkStatus (see page 3714)	TChunkStatus enumerates the possible states of a chunk of memory.
	TCVModInfo (see page 3715)	This is record System.TCVModInfo.
	TDispatchMessage (see page 3716)	This is record System.TDispatchMessage.
	TDispDesc (see page 3716)	This is record System.TDispDesc.
	TDynArrayTypeInfo (see page 3716)	This is record System.TDynArrayTypeInfo.
	TFileRec (see page 3717)	TFileRec is the internal format for typed and untyped files in the Delphi language.
	TGUID (see page 3718)	TGUID is a structured form of the value that uniquely identifies an interface.
	THeapBlock (see page 3720)	This is record System::THeapBlock.
	THeapStatus (see page 3720)	System::THeapStatus represents information about the global heap.
	TInitContext (see page 3721)	This is record System.TInitContext.
	TInterfaceEntry (see page 3722)	TInterfaceEntry encapsulates the information needed to dispatch interface calls.
	TInterfaceTable (see page 3723)	TInterfaceTable lists the interfaces that a class supports, and PInterfaceTable (see page 3766) points to this list.
	TLibModule (see page 3723)	System::TLibModule describes a module that is compiled with the application.
	TMemoryManager (see page 3724)	System::TMemoryManager defines the memory block entry points.
	TMemoryManagerEx (see page 3725)	System::TMemoryManagerEx defines extended memory block entry points.
	TMemoryManagerState (see page 3730)	System::TMemoryManagerState holds statistics about memory allocations.
	TMethod (see page 3731)	TMethod stores Code and Data fields to represent a method.
	TMinimumBlockAlignment (see page 3732)	TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.
	TModuleUnloadRec (see page 3732)	This is record System::TModuleUnloadRec.
	TMonitorSupport (see page 3733)	This is record System.TMonitorSupport.

	TResStringRec (see page 3733)	This is record System::TResStringRec.
	TRuntimeError (see page 3734)	TRuntimeError is an enumeration of internal runtime error codes.
	TSmallBlockTypeState (see page 3736)	System::TSmallBlockTypeState holds statistics about a small memory block.
	TTextLineBreakStyle (see page 3737)	TTextLineBreakStyle indicates the default set of characters used to divide lines in text.
	TTextRec (see page 3738)	TTextRec is the internal format for text files.
	TVarArray (see page 3739)	This is the descriptor type for safe arrays.
	TVarArrayBound (see page 3740)	Gives bound information for one dimension of a safe array.
	TVarData (see page 3740)	TVarData represents Variant (see page 3421) information.
	TVariantManager (see page 3745)	System::TVariantManager defines entry points for variant conversion routines and other procedures that define variant behavior.
	TVarRec (see page 3747)	System::TVarRec is used inside a function with an untyped array parameter.
	TCallDesc (see page 3749)	This is type System::TCallDesc.
	TDispDesc (see page 3749)	This is type System::TDispDesc.

Types

Name	Description
CppLongInt (see page 3752)	This is type System.CppLongInt.
CppULongInt (see page 3753)	This is type System.CppULongInt.
GetPackageInfoTable (see page 3753)	This is type System.GetPackageInfoTable.
HGLOBAL (see page 3753)	This is type System.HGLOBAL.
HINST (see page 3754)	This is type System.HINST.
HMODULE (see page 3754)	This is type System.HMODULE.
HRESULT (see page 3754)	This is type System.HRESULT.
HRSRC (see page 3755)	This is type System.HRSRC.
Int16 (see page 3755)	Alias for `SmallInt`.
Int32 (see page 3755)	Alias for `Integer`.
Int8 (see page 3756)	Alias for `ShortInt`.
IntegerArray (see page 3756)	This is type System.IntegerArray.
IUnknown (see page 3756)	IUnknown is a special base interface.
PackageInfo (see page 3757)	PackageInfo is a pointer to a PackageInfoTable (see page 3713) variable.
PAnsiString (see page 3757)	PAnsiString is a pointer to an `AnsiString` variable.
PBoolean (see page 3758)	PBoolean is a pointer to a `Boolean` variable.
PByte (see page 3758)	PByte is a pointer to a `Byte` variable.
PCallDesc (see page 3758)	This is type System::PCallDesc.
PCardinal (see page 3759)	PCardinal is a pointer to a `Cardinal` variable.
PComp (see page 3759)	PComp is a pointer to a `Comp` variable.
PCurrency (see page 3759)	PCurrency is a pointer to a `Currency (see page 3235)` variable.
PCVModInfo (see page 3760)	This is type System.PCVModInfo.
PDate (see page 3760)	PDate is a pointer to a Date variable.
PDateTime (see page 3761)	PDateTime is a pointer to a TDateTime (see page 3323) variable.
PDispatch (see page 3761)	This is type System.PDispatch.
PDispDesc (see page 3761)	This is type System::PDispDesc.
PDouble (see page 3762)	PDouble is a pointer to a `Double` variable.
PDynArrayTypeInfo (see page 3762)	This is type System.PDynArrayTypeInfo.
PError (see page 3762)	This is type System.PError.
PExtended (see page 3763)	PExtended is a pointer to an `Extended` variable.
PGUID (see page 3763)	PGUID is a pointer to a TGUID (see page 3718) variable.
PInitContext (see page 3765)	This is type System.PInitContext.
PInt64 (see page 3765)	PInt64 is a pointer to an `Int64` variable.
PInteger (see page 3765)	PInteger is a pointer to an `Integer` variable.
PIntegerArray (see page 3766)	PIntegerArray is a pointer to an `IntegerArray (see page 3756)` variable.
PInterfaceEntry (see page 3766)	PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3722) variable.
PInterfaceTable (see page 3766)	PInterfaceTable is a pointer to a TInterfaceTable (see page 3723) variable.

PLibModule (see page 3767)	PLibModule is a pointer to a System::TLibModule (see page 3723) variable.
PLongint (see page 3767)	PLongint is a pointer to a `LongInt` variable.
PLongWord (see page 3768)	PLongWord is a pointer to a `LongWord` variable.
PMemoryManager (see page 3768)	System::PMemoryManager is a pointer to a System::TMemoryManager (see page 3724) variable.
PMemoryManagerEx (see page 3768)	System::PMemoryManagerEx is a pointer to a System::TMemoryManagerEx (see page 3725) variable.
PModuleUnloadRec (see page 3769)	This is type System::PModuleUnloadRec.
PMonitor (see page 3769)	This is type System.PMonitor.
PMonitorSupport (see page 3769)	This is type System.PMonitorSupport.
PointerArray (see page 3770)	This is type System.PointerArray.
POleVariant (see page 3770)	POleVariant is a pointer to a System::OleVariant (see page 3274) variable.
PPAnsiChar (see page 3770)	This is type System.PPAnsiChar.
PPChar (see page 3771)	This is type System.PPChar.
PPCharArray (see page 3771)	This is type System.PPCharArray.
PPDispatch (see page 3771)	This is type System.PPDispatch.
PPMonitor (see page 3772)	This is type System.PPMonitor.
PPointer (see page 3772)	PPointer is a pointer to a `Pointer` variable.
PPointerArray (see page 3772)	PPointerArray is a pointer to a PointerArray (see page 3770) variable.
PPUnknown (see page 3773)	This is type System.PPUnknown.
PPWideChar (see page 3773)	This is type System.PPWideChar.
PRawByteString (see page 3773)	This is type System.PRawByteString.
PResStringRec (see page 3774)	This is type System::PResStringRec.
PShortInt (see page 3774)	PShortInt is a pointer to a `ShortInt` variable.
PShortString (see page 3774)	PShortString is a pointer to a `ShortString` variable.
PSingle (see page 3775)	PSingle is a pointer to a `Single` variable.
PSmallBlockTypeState (see page 3775)	This is type System::PSmallBlockTypeState.
PSmallInt (see page 3775)	PSmallInt is a pointer to a `SmallInt` variable.
PString (see page 3776)	PString is a pointer to a `String` variable.
PTextBuf (see page 3776)	This is type System.PTextBuf.
PUCS2Char (see page 3776)	This is type System.PUCS2Char.
PUCS4Char (see page 3777)	This is type System.PUCS4Char.
PUCS4CharArray (see page 3777)	This is type System.PUCS4CharArray.
PUInt64 (see page 3777)	This is type System.PUInt64.
PUnicodeString (see page 3778)	This is type System.PUnicodeString.
PUnitEntryTable (see page 3778)	PUnitEntryTable is a pointer to a UnitEntryTable (see page 3804) variable.
PUnknown (see page 3778)	PUnknown is a pointer to an IUnknown (see page 3756) variable.
PUTF8String (see page 3779)	This is type System.PUTF8String.
PVarArray (see page 3779)	Pointer type to a TVarArray (see page 3739).
PVarArrayBoundArray (see page 3779)	This is type System.PVarArrayBoundArray.
PVarArrayCoorArray (see page 3780)	Pointer type to a TVarArrayCoorArray (see page 3799).
PVarData (see page 3780)	This is type System.PVarData.
PVariant (see page 3780)	PVariant is a pointer to a System::Variant (see page 3421) variable.
PVariantManager (see page 3781)	This is type System::PVariantManager.
PVarRec (see page 3781)	System::PVarRec is a pointer to a System::TVarRec (see page 3747) variable.
PWideString (see page 3782)	PWideString is a pointer to a `WideString` (see page 3467) variable.
PWord (see page 3782)	PWord is a pointer to a `Word` variable.
PWordBool (see page 3782)	PWordBool is a pointer to a `WordBool` variable.
RawByteString (see page 3783)	RawByteString enables the passing of string data of any code page without doing any codepage conversions.
ResourceString (see page 3783)	This is type System::ResourceString.
TAssertErrorProc (see page 3783)	This is type System.TAssertErrorProc.
TBooleanDynArray (see page 3784)	This is type System::TBooleanDynArray.
TBoundArray (see page 3784)	This is type System.TBoundArray.
TByteDynArray (see page 3784)	This is type System::TByteDynArray.

TCardinalDynArray (see page 3785)	This is type System::TCardinalDynArray.
TClass (see page 3785)	TClass defines the metaclass for System::TObject (see page 3354).
TClassHelperBaseClass (see page 3786)	This is type System.TClassHelperBaseClass.
TDate (see page 3786)	TDate is used to represent a date.
TDateTime (see page 3787)	TDateTime represents a date-and-time value in the Delphi language.
TDLLProc (see page 3788)	This is type System.TDLLProc.
TDLLProcEx (see page 3788)	This is type System.TDLLProcEx.
TDoubleDynArray (see page 3788)	This is type System::TDoubleDynArray.
TEnumModuleFunc (see page 3789)	This is type System.TEnumModuleFunc.
TEnumModuleFuncLW (see page 3789)	This is type System.TEnumModuleFuncLW.
TFileIOFunc (see page 3789)	This is type System.TFileIOFunc.
THandle (see page 3790)	THandle is a general purpose data reference (handle).
TInt64DynArray (see page 3790)	This is type System::TInt64DynArray.
TIntegerDynArray (see page 3790)	This is type System::TIntegerDynArray.
TInterfacedClass (see page 3791)	TInterfacedClass defines the metaclass for System::TInterfacedObject (see page 3342).
TLongWordDynArray (see page 3791)	This is type System::TLongWordDynArray.
TMemoryMap (see page 3791)	TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 3714) values.
TModuleUnloadProc (see page 3792)	TModuleUnloadProc defines a procedure that can be called during package unload.
TModuleUnloadProcLW (see page 3793)	TModuleUnloadProcLW dehines a procedure that can be called during package unload.
TPCharArray (see page 3793)	This is type System.TPCharArray.
TResourceHandle (see page 3793)	This is type System.TResourceHandle.
TSafeCallErrorProc (see page 3794)	This is type System.TSafeCallErrorProc.
TShortIntDynArray (see page 3794)	This is type System::TShortIntDynArray.
TSingleDynArray (see page 3794)	This is type System::TSingleDynArray.
TSmallBlockTypeStates (see page 3795)	System::TSmallBlockTypeStates is an array of small memory block information records.
TSmallIntDynArray (see page 3795)	This is type System::TSmallIntDynArray.
TStringDynArray (see page 3796)	This is type System::TStringDynArray.
TSystemThreadEndProc (see page 3796)	This is type System.TSystemThreadEndProc.
TSystemThreadFuncProc (see page 3796)	This is type System.TSystemThreadFuncProc.
TTextBuf (see page 3797)	This is type System.TTextBuf.
TTextIOFunc (see page 3797)	This is type System.TTextIOFunc.
TThreadFunc (see page 3797)	System::TThreadFunc defines a new thread of execution.
TTime (see page 3798)	TTime is used to represent a time.
TUCS4CharArray (see page 3798)	This is type System.TUCS4CharArray.
TVarArrayBoundArray (see page 3798)	Gives bound information for all the dimensions of a safe array.
TVarArrayCoorArray (see page 3799)	Gives the position of an element in a safe array.
TVarOp (see page 3799)	TVarOp defines a Variant (see page 3421) operation.
TVarType (see page 3801)	This is type System.TVarType.
TWideStringDynArray (see page 3801)	This is type System::TWideStringDynArray.
TWordDynArray (see page 3801)	This is type System::TWordDynArray.
UCS2Char (see page 3802)	This is type System.UCS2Char.
UCS4Char (see page 3802)	UCS4Char is used for working with 4–byte Unicode characters.
UCS4String (see page 3802)	UCS4String is used for working with 4–byte Unicode characters.
UInt16 (see page 3803)	Alias for `Word`.
UInt32 (see page 3803)	Alias for `Cardinal`.
UInt8 (see page 3804)	Alias for `Byte`.
UnitEntryTable (see page 3804)	UnitEntryTable is an array type with entries of type PackageUnitEntry (see page 3713).
UTF8String (see page 3804)	UTF8String represents UTF-8 encoded Unicode strings.

Variables

Name	Description
AbstractErrorProc (see page 3807)	AbstractErrorProc defines a callback routine for abstract method call errors.
AllocMemCount (see page 3807)	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemSize (see page 3809)	Represents the total size of allocated memory blocks.
AssertErrorProc (see page 3810)	Points to the assertion error-handler.
CmdLine (see page 3812)	Points to the command-line arguments specified when the application is invoked.
CmdShow (see page 3812)	Specifies the flags for displaying the application's main window.
CPUCount (see page 3813)	This is variable System.CPUCount.
DebugHook (see page 3813)	DebugHook is set if an application is running under the IDE debugger.
Default8087CW (see page 3813)	Specifies the Default 8087 control word.
DefaultSystemCodePage (see page 3815)	This is variable System.DefaultSystemCodePage.
DefaultTextLineBreakStyle (see page 3815)	Specifies the characters that are used by default to separate lines in text.
DefaultUnicodeCodePage (see page 3816)	This is variable System.DefaultUnicodeCodePage.
DispCallByIDProc (see page 3816)	This is variable System.DispCallByIDProc.
DLLProc (see page 3816)	Points to a procedure invoked by a DLL entry point.
envp (see page 3817)	This is variable System.envp.
ErrorAddr (see page 3817)	Contains the address of a statement causing a runtime error.
ErrorProc (see page 3818)	Points to the RTL runtime error handler.
ErrOutput (see page 3820)	ErrOutput specifies a write-only Text file associated with the process's error output file.
ExceptClsProc (see page 3820)	ExceptClsProc defines a pointer to a function called during exception handling.
ExceptionAcquired (see page 3821)	This is variable System.ExceptionAcquired.
ExceptionClass (see page 3821)	This is variable System.ExceptionClass.
ExceptObjProc (see page 3821)	ExceptObjProc defines a pointer to a function called during exception handling.
ExceptProc (see page 3822)	Points to the lowest-level exception handler.
ExitCode (see page 3823)	Contains the application's exit code.
ExitProc (see page 3823)	Points to a program's exit procedure (provided for backward compatibility).
ExitProcessProc (see page 3824)	Specifies the last procedure to execute before the application shuts down.
FileMode (see page 3824)	Indicates the access mode on typed and untyped files opened by the Reset (see page 3643) procedure.
HeapAllocFlags (see page 3825)	Indicates how the memory manager obtains memory from the operating system.
HPrevInst (see page 3826)	This is variable System.HPrevInst.
InitProc (see page 3826)	InitProc is the last installed initialization procedure.
Input (see page 3827)	Specifies a read-only Text file associated with the processes standard Input file.
IsConsole (see page 3828)	Indicates whether the module was compiled as a console application.
IsLibrary (see page 3828)	Indicates whether the module is a shared library.
IsMultiThread (see page 3829)	Indicates whether the application spawned additional threads using BeginThread (see page 3527) or TThread objects.
JITEnable (see page 3829)	Controls when the just-in-time debugger is called.
LibModuleList (see page 3830)	Lists module records for enumeration procedures.
MainInstance (see page 3830)	Indicates the instance handle for the main executable.
MainThreadID (see page 3831)	Indicates the instance handle for the current module's main thread of execution.
ModuleUnloadList (see page 3831)	This is variable System.ModuleUnloadList.
MonitorSupport (see page 3831)	This is variable System.MonitorSupport.
NeverSleepOnMMThreadContention (see page 3832)	Dictates memory manager behavior during a thread contention.
NoErrMsg (see page 3832)	Controls whether the application displays an error message when a runtime error occurs.
Output (see page 3833)	Specifies a write-only Text file associated with the process's standard output file.
RaiseExceptionProc (see page 3833)	Points to RaiseException Windows function.
RaiseExceptObjProc (see page 3834)	This is variable System.RaiseExceptObjProc.
RandSeed (see page 3834)	RandSeed stores the built-in random number generator's seed.
ReportMemoryLeaksOnShutdown (see page 3835)	Report memory leaks on shutdown.
RTLUnwindProc (see page 3835)	This is variable System.RTLUnwindProc.

SafeCallErrorProc (see page 3836)	SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.
SystemThreadEndProc (see page 3836)	This is variable System.SystemThreadEndProc.
SystemThreadFuncProc (see page 3837)	This is variable System.SystemThreadFuncProc.
Test8086 (see page 3837)	This is variable System.Test8086.
Test8087 (see page 3837)	This is variable System.Test8087.
TestFDIV (see page 3838)	This is variable System.TestFDIV.
UTF8CompareLocale (see page 3838)	This is variable System.UTF8CompareLocale.
VarAddRefProc (see page 3838)	The VarAddRefProc variable is used internally.
VarClearProc (see page 3839)	The VarClearProc variable is used internally.
VarCopyProc (see page 3839)	The VarCopyProc variable is used internally.
VarToLStrProc (see page 3840)	The VarToLStrProc variable is used internally.
VarToWStrProc (see page 3840)	The VarToWStrProc variable is used internally.

35.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	AnsiStringBase (see page 3174)	System::AnsiStringBase is the base class for the C++ analog of the Delphi long string type.
	Currency (see page 3235)	System::Currency is the C++ implementation of the Delphi System::Currency data type.
	CurrencyBase (see page 3255)	This is class System::CurrencyBase.
	DynArrayException (see page 3271)	This is class System::DynArrayException.
	DynArrayNullData (see page 3272)	This is class System::DynArrayNullData.
	DynArrayOutOfRange (see page 3272)	This is class System::DynArrayOutOfRange.
	OleVariant (see page 3274)	System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.
	TAggregatedObject (see page 3313)	System::TAggregatedObject provides the functionality for an inner object of an aggregate by implementing the IInterface (see page 3502) methods to delegate to the controlling IInterface (see page 3502).
	TClassHelperBase (see page 3318)	This is class System.TClassHelperBase.
	TContainedObject (see page 3320)	System::TContainedObject implements the IInterface (see page 3502) interface as an inner object in an aggregate.
	TDateTime (see page 3323)	System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.
	TDateTimeBase (see page 3341)	This is class System::TDateTimeBase.
	TInterfacedObject (see page 3342)	System::TInterfacedObject is a base for simple classes that need a basic IInterface (see page 3502) implementation.
	TMetaClass (see page 3348)	System::TMetaClass is the C++ representation of the Delphi class-reference type.
	TObject (see page 3354)	System::TObject is the ultimate ancestor of all objects and components.
	UnicodeString (see page 3381)	System::UnicodeString is the C++ analog for the Delphi UnicodeString type.
	Variant (see page 3421)	System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.
	WideString (see page 3467)	System::WideString represents strings of 16-bit characters.

Records

	Name	Description
	TMonitor (see page 3349)	TMonitor provides methods for synchronizing the access of several threads to a single object.

Templates

	Name	Description
	AnsiStringT (see page 3207)	System::AnsiStringT is the C++ analog for the Delphi long string type.
	DelphiInterface (see page 3256)	System::DelphiInterface is a C++ implementation of the Delphi interface.
	DynamicArray (see page 3261)	System::DynamicArray is the C++ analog for the Delphi dynamic array type.
	SmallString (see page 3307)	This is class System::SmallString.
	SmallStringBase (see page 3310)	This is class System::SmallStringBase.
	sPointer (see page 3311)	This is class System::sPointer.

35.1.1 AnsiStringBase Class

System::AnsiStringBase is the base class for the C++ analog of the Delphi long string type.

Class Hierarchy

Pascal

```
AnsiStringBase = class;
```

C++

```
class AnsiStringBase;
```

File

dstring.h

Description

System::AnsiStringBase exists to provide a place for methods/members that are common to all instantiations of the template class AnsiStringT (see page 3207)<CodePage (see page 3188)>. AnsiString itself is a typedef that is equivalent to AnsiStringT (see page 3207)<0>. You should never deal directly with System::AnsiStringBase.

System::AnsiStringT (see page 3207) is a template class that you use when handling a Delphi AnsiString that has a code page specified with it.

Delphi uses several string types. One important long string type is commonly known as System::AnsiString. Support for this type includes the following features:

- strings can be as large as available memory
- efficient use of memory is ensured through shared references
- routines and operators evaluate strings based on the current locale

Despite its name, System::AnsiStringBase is not restricted to the ANSI character set, and can use any character set supported by the current locale definition, including multi-byte and Unicode character sets.

System::AnsiStringBase variables that have not been assigned an initial value contain a zero-length string.

To use the C++ streaming operators (**<<** and **>>**) with System::AnsiStringBase, you must `iostream` before including `system.hpp` and use `AnsiStringT::c_str()` to return the internal string representation. The following example demonstrates the usage of AnsiString with stream operators:

```
#include <iostream>
#include <system.hpp>
int main() {
  AnsiString HelloStr = "hello";
  std::cout << HelloStr.c_str();
}
```

Note: Delphi also supports System::AnsiString, but implements it as a primitive type rather than a class.

UnicodeString (see page 3381) has become the default string type. By default, variables declared as type String are System::UnicodeString (see page 3381).

See Also

How to Handle Delphi AnsiString Code Page Specification in C++

System::WideString (see page 3467)

System::AnsiStringBase::[] (see page 3205)

System::AnsiStringBase::= (see page 3206)

System::AnsiStringBase::== (see page 3206)

System::AnsiStringBase::!= (see page 3204)

System::AnsiStringBase::> (see page 3207)

System::AnsiStringBase::>= (see page 3207)

System::AnsiStringBase::< (see page 3205)

System::AnsiStringBase::<= (see page 3206)

System::AnsiStringBase::+ (see page 3205)

System::AnsiStringT::+= (see page 3232)

35.1.1.1 AnsiStringBase Members

The following tables list the members exposed by AnsiStringBase.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3180)	System::AnsiStringBase::TStringFloatFormat is an enumerated list of formatting codes for float functions.
	TStringMbcsByteType (see page 3181)	Enumerated list of byte types in MCBS string.

Structures

	Name	Description
	StrRec (see page 3180)	This is record System::AnsiStringBase::StrRec.

Protected Data Members

	Name	Description
	Data (see page 3182)	This is Data, a member of class AnsiStringBase.

Protected Methods

	Name	Description
	_AnsiCat (see page 3184)	This is the overview for the _AnsiCat method overload.
	_AnsiFromPWChar (see page 3184)	This is _AnsiFromPWChar, a member of class AnsiStringBase.
	AnsiCompare (see page 3185)	Compares the System::AnsiStringBase::AnsiStringBase (see page 3174) to another specified string (case-sensitively).
	AnsiCompareIC (see page 3185)	Compares the System::AnsiStringBase::AnsiStringBase (see page 3174) to another specified string (case insensitively).
	AnsiStringBase (see page 3178)	This is the overview for the AnsiStringBase constructor overload.
	GetRec (see page 3193)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3200)	This is ThrowIfOutOfRange, a member of class AnsiStringBase.

Protected Operators

	Name	Description
	!= (see page 3204)	Compares two strings.
	< (see page 3205)	Compares two strings.
	<= (see page 3206)	Compares two strings.
	= (see page 3206)	Assigns the string rhs to this System::AnsiStringBase (see page 3174).
	== (see page 3206)	Compares the System::AnsiStringBase (see page 3174) with a specified string, returning true if they are the same.
	> (see page 3207)	Compares two strings.
	>= (see page 3207)	Compares two strings.

Public Methods

	Name	Description
	~AnsiStringBase (see page 3178)	Frees an instance of System::AnsiStringBase (see page 3174).

	AnsiLastChar (see page 3186)	Returns a pointer to the last full character in the System::AnsiStringBase::AnsiStringBase (see page 3174).
	AnsiPos (see page 3186)	Returns the index at which the specified substring begins.
	BytesOf (see page 3186)	Get string's data in bytes.
	ByteType (see page 3187)	Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase (see page 3174).
	cat_vprintf (see page 3187)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringBase::AnsiStringBase (see page 3174).
	CodePage (see page 3188)	Gets code page of a System::AnsiStringBase (see page 3174).
	CurrToStr (see page 3188)	Converts a currency value to its string representation.
	CurrToStrF (see page 3188)	Converts a currency value to its string representation using the indicated float format convention.
	Delete (see page 3189)	Removes a specified number of characters from the string.
	ElementSize (see page 3189)	Gets element size of data in a System::AnsiStringBase (see page 3174).
	EnsureAnsi (see page 3189)	Ensure string's data is ANSI data.
	FloatToStrF (see page 3190)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 3190)	Formats output using a resourced format string.
	Format (see page 3191)	System::AnsiStringBase::Formats a string assembled from an System::AnsiStringBase (see page 3174), format, and a series of array arguments, args.
	FormatFloat (see page 3191)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3193)	System::AnsiStringBase::Inserts a specified string into the System::AnsiStringBase (see page 3174).
	IntToHex (see page 3194)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3194)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 3194)	Returns true if the System::AnsiStringBase (see page 3174) is empty.
	IsLeadByte (see page 3195)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 3195)	Returns true if the specified character is '\'.
	IsTrailByte (see page 3195)	Returns true if the specified byte is of type mbTrailByte.
	LastChar (see page 3196)	Returns a pointer to the last character in a string.
	LastDelimiter (see page 3196)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 3196)	Returns the length of the System::AnsiStringBase (see page 3174).
	LoadStr (see page 3196)	This is the overview for the LoadStr method overload.
	LoadString (see page 3197)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 3198)	Returns a lower-case version of System::AnsiStringBase (see page 3174).
	Pos (see page 3198)	Returns the index at which a specified substring begins.
	RefCount (see page 3198)	Gets reference count of System::AnsiStringBase (see page 3174).
	SetLength (see page 3199)	Changes the length of the System::AnsiStringBase (see page 3174).
	StringOfChar (see page 3199)	Returns a string with the specified number of characters.
	SubString (see page 3199)	Returns a specified substring of the System::AnsiStringBase (see page 3174).
	swap (see page 3200)	Swap string pointers.
	ToDouble (see page 3200)	Converts the string to a floating-point value.
	ToInt (see page 3200)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3201)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3201)	Returns a new System::AnsiStringBase (see page 3174) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3201)	Returns a new System::AnsiStringBase (see page 3174) without leading spaces or control characters.
	TrimRight (see page 3202)	Returns a new System::AnsiStringBase (see page 3174) without trailing spaces or control characters.
	Unique (see page 3202)	Makes the System::AnsiStringBase (see page 3174) unique.
	UpperCase (see page 3202)	Returns an upper-case version of this System::AnsiStringBase (see page 3174).
	vprintf (see page 3202)	Sets the value of the System::AnsiStringBase (see page 3174) given a format string and its arguments.

	WideChar (see page 3203)	Converts the System::AnsiStringBase (see page 3174) to a wide-character array.
	WideCharBufSize (see page 3203)	Returns the size of the buffer required by the WideChar (see page 3203) function.

Public Operators

	Name	Description
	[] (see page 3204)	This is the overview for the [] operator overload.
	+ (see page 3205)	Concatenates two System::AnsiStringBase (see page 3174) values.

35.1.1.2 AnsiStringBase::~AnsiStringBase Destructor

Frees an instance of System::AnsiStringBase (see page 3174).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~AnsiStringBase();
```

Description

System::AnsiStringBase (see page 3174)::~AnsiStringBase (see page 3174) is the destructor for System::AnsiStringBase (see page 3174).

35.1.1.3 AnsiStringBase Constructor

This is the overview for the AnsiStringBase constructor overload.

Overload List

	Name	Description
	AnsiStringBase::AnsiStringBase () (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (AnsiStringBase&) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (char*, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (char*, int, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (char16_t*, int, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (char32_t*, int, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (double, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (UnicodeString &, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (wchar_t*, int, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).

	AnsiStringBase::AnsiStringBase (wchar_t, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).
	AnsiStringBase::AnsiStringBase (WideString &, int) (see page 3179)	Creates an instance of System::AnsiStringBase (see page 3174).

35.1.1.3.1 AnsiStringBase::AnsiStringBase Constructor ()

Creates an instance of System::AnsiStringBase (see page 3174).

Pascal

```
constructor Create;
constructor Create(const src: AnsiStringBase&);
constructor Create(const src: char*; codePage: Integer);
constructor Create(const src: char*; byteLen: Integer; codePage: Integer);
constructor Create(const src: wchar_t*; numwchar: Integer; codePage: Integer);
constructor Create(const src: char16_t*; numChar16: Integer; codePage: Integer);
constructor Create(const src: char32_t*; numChar32: Integer; codePage: Integer);
constructor Create(src: double; codePage: Integer);
constructor Create(src: wchar_t; codePage: Integer);
constructor Create(const src: WideString &; codePage: Integer);
constructor Create(const src: UnicodeString &; codePage: Integer);
```

C++

```
__fastcall AnsiStringBase();
__fastcall AnsiStringBase(const AnsiStringBase& src);
__fastcall AnsiStringBase(const char* src, int codePage);
__fastcall AnsiStringBase(const char* src, int byteLen, int codePage);
__fastcall AnsiStringBase(const wchar_t* src, int numwchar, int codePage);
__fastcall AnsiStringBase(const char16_t* src, int numChar16, int codePage);
__fastcall AnsiStringBase(const char32_t* src, int numChar32, int codePage);
__fastcall AnsiStringBase(double src, int codePage);
__fastcall AnsiStringBase(wchar_t src, int codePage);
__fastcall AnsiStringBase(const WideString & src, int codePage);
__fastcall AnsiStringBase(const UnicodeString & src, int codePage);
```

Description

System::AnsiStringBase::AnsiStringBase (see page 3174) is the constructor for System::AnsiStringBase (see page 3174).

If the constructor is given no arguments, the resulting System::AnsiStringBase (see page 3174) instance is an empty string. If the constructor is passed a single character, the resulting System::AnsiStringBase (see page 3174) instance has character length one, and contains that character. Other versions of the constructor copy the string passed as an argument, converting characters to `char` if necessary. When constructing a System::AnsiStringBase (see page 3174) instance from an array of `char` or `wchar_t`, the source string should be null terminated, unless an argument is given that indicates the number of characters in the array. For some overloads of the constructor, you can pass the code page you want for the string. In another overload, you can pass a floating point number to be converted to a string.

35.1.1.4 AnsiStringBase Structures

The structures of the AnsiStringBase class are listed here.

Structures

	Name	Description
	StrRec (see page 3180)	This is record System::AnsiStringBase::StrRec.

35.1.1.4.1 System::AnsiStringBase::StrRec Structure

Pascal

```
StrRec = record
  codePage: unsigned short;
  elemSize: unsigned short;
  refCnt: Integer;
  length: Integer;
end;
```

C++

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

File

dstring.h

Description

This is record System::AnsiStringBase::StrRec.

35.1.1.5 AnsiStringBase Enumerations

The enumerations of the AnsiStringBase class are listed here.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3180)	System::AnsiStringBase::TStringFloatFormat is an enumerated list of formatting codes for float functions.
	TStringMbcsByteType (see page 3181)	Enumerated list of byte types in MCBS string.

35.1.1.5.1 System::AnsiStringBase::TStringFloatFormat Enumeration

System::AnsiStringBase::TStringFloatFormat is an enumerated list of formatting codes for float functions.

Pascal

```
TStringFloatFormat = (
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
);
```

C++

```
enum TStringFloatFormat {
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
};
```

File

dstring.h

Description

System::AnsiStringBase::TStringFloatFormat defines an enumerated list of formatting codes for the float functions.

Format (see page 3191)	Defines
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 3235) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, and DecimalSeparator global variables, all of which are initialized from the Currency (see page 3235) Format (see page 3191) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18.

See Also

System::AnsiStringBase::FloatToStrF (see page 3190)

35.1.1.5.2 System::AnsiStringBase::TStringMbcsByteType Enumeration

Enumerated list of byte types in MCBS string.

Pascal

```
TStringMbcsByteType = (
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
);
```

C++

```
enum TStringMbcsByteType {
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
};
```

File

dstring.h

Description

System::AnsiStringBase::TStringMbcsByteType defines an enumerated list of byte types in multibyte character system (MBCS) string.

Constant	Meaning
mbSingleByte	Single byte character.
mbLeadByte	Lead byte of a multi-byte character.
mbTrailByte	Trailing byte of a multi-byte character.

See Also

System::AnsiStringBase::ByteType (see page 3187)

35.1.1.6 AnsiStringBase Data Members

The data members of the AnsiStringBase class are listed here.

Protected Data Members

	Name	Description
	Data (see page 3182)	This is Data, a member of class AnsiStringBase.

35.1.1.6.1 AnsiStringBase::Data Data Member

Pascal

```
Data: PChar;
```

C++

```
char * Data;
```

Description

This is Data, a member of class AnsiStringBase.

35.1.1.7 AnsiStringBase Methods

The methods of the AnsiStringBase class are listed here.

Protected Methods

	Name	Description
	_AnsiCat (see page 3184)	This is the overview for the _AnsiCat method overload.
	_AnsiFromPWChar (see page 3184)	This is _AnsiFromPWChar, a member of class AnsiStringBase.
	AnsiCompare (see page 3185)	Compares the System::AnsiStringBase::AnsiStringBase (see page 3174) to another specified string (case-sensitively).
	AnsiCompareIC (see page 3185)	Compares the System::AnsiStringBase::AnsiStringBase (see page 3174) to another specified string (case insensitively).
	GetRec (see page 3193)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3200)	This is ThrowIfOutOfRange, a member of class AnsiStringBase.

Public Methods

	Name	Description
	AnsiLastChar (see page 3186)	Returns a pointer to the last full character in the System::AnsiStringBase::AnsiStringBase (see page 3174).
	AnsiPos (see page 3186)	Returns the index at which the specified substring begins.
	BytesOf (see page 3186)	Get string's data in bytes.
	ByteType (see page 3187)	Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase (see page 3174).
	cat_vprintf (see page 3187)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringBase::AnsiStringBase (see page 3174).
	CodePage (see page 3188)	Gets code page of a System::AnsiStringBase (see page 3174).
	CurrToStr (see page 3188)	Converts a currency value to its string representation.
	CurrToStrF (see page 3188)	Converts a currency value to its string representation using the indicated float format convention.
	Delete (see page 3189)	Removes a specified number of characters from the string.
	ElementSize (see page 3189)	Gets element size of data in a System::AnsiStringBase (see page 3174).
	EnsureAnsi (see page 3189)	Ensure string's data is ANSI data.
	FloatToStrF (see page 3190)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 3190)	Formats output using a resourced format string.
	Format (see page 3191)	System::AnsiStringBase::Formats a string assembled from an System::AnsiStringBase (see page 3174), format, and a series of array arguments, args.
	FormatFloat (see page 3191)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3193)	System::AnsiStringBase::Inserts a specified string into the System::AnsiStringBase (see page 3174).
	IntToHex (see page 3194)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3194)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 3194)	Returns true if the System::AnsiStringBase (see page 3174) is empty.
	IsLeadByte (see page 3195)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 3195)	Returns true if the specified character is '\'.
	IsTrailByte (see page 3195)	Returns true if the specified byte is of type mbTrailByte.
	LastChar (see page 3196)	Returns a pointer to the last character in a string.
	LastDelimiter (see page 3196)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 3196)	Returns the length of the System::AnsiStringBase (see page 3174).
	LoadStr (see page 3196)	This is the overview for the LoadStr method overload.
	LoadString (see page 3197)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 3198)	Returns a lower-case version of System::AnsiStringBase (see page 3174).
	Pos (see page 3198)	Returns the index at which a specified substring begins.
	RefCount (see page 3198)	Gets reference count of System::AnsiStringBase (see page 3174).
	SetLength (see page 3199)	Changes the length of the System::AnsiStringBase (see page 3174).
	StringOfChar (see page 3199)	Returns a string with the specified number of characters.
	SubString (see page 3199)	Returns a specified substring of the System::AnsiStringBase (see page 3174).
	swap (see page 3200)	Swap string pointers.
	ToDouble (see page 3200)	Converts the string to a floating-point value.
	ToInt (see page 3200)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3201)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3201)	Returns a new System::AnsiStringBase (see page 3174) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3201)	Returns a new System::AnsiStringBase (see page 3174) without leading spaces or control characters.
	TrimRight (see page 3202)	Returns a new System::AnsiStringBase (see page 3174) without trailing spaces or control characters.
	Unique (see page 3202)	Makes the System::AnsiStringBase (see page 3174) unique.

	UpperCase (see page 3202)	Returns an upper-case version of this System::AnsiStringBase (see page 3174).
	vprintf (see page 3202)	Sets the value of the System::AnsiStringBase (see page 3174) given a format string and its arguments.
	WideChar (see page 3203)	Converts the System::AnsiStringBase (see page 3174) to a wide-character array.
	WideCharBufSize (see page 3203)	Returns the size of the buffer required by the WideChar (see page 3203) function.

35.1.1.7.1 _AnsiCat Method

This is the overview for the _AnsiCat method overload.

Overload List

	Name	Description
	AnsiStringBase::_AnsiCat (AnsiStringBase&, AnsiStringBase&) (see page 3184)	This is _AnsiCat, a member of class AnsiStringBase.
	AnsiStringBase::_AnsiCat (AnsiStringBase&, AnsiStringBase&, AnsiStringBase&) (see page 3184)	This is _AnsiCat, a member of class AnsiStringBase.

35.1.1.7.1.1 AnsiStringBase::_AnsiCat Method (AnsiStringBase&, AnsiStringBase&)

Pascal

```
procedure _AnsiCat(var dst: AnsiStringBase; const src: AnsiStringBase&); static;
```

C++

```
static __fastcall void _AnsiCat(AnsiStringBase& dst, const AnsiStringBase& src);
```

Description

This is _AnsiCat, a member of class AnsiStringBase.

35.1.1.7.1.2 AnsiStringBase::_AnsiCat Method (AnsiStringBase&, AnsiStringBase&, AnsiStringBase&)

Pascal

```
procedure _AnsiCat(var dst: AnsiStringBase; const src1: AnsiStringBase&; const src2:
AnsiStringBase&); static;
```

C++

```
static __fastcall void _AnsiCat(AnsiStringBase& dst, const AnsiStringBase& src1, const
AnsiStringBase& src2);
```

Description

This is _AnsiCat, a member of class AnsiStringBase.

35.1.1.7.2 AnsiStringBase::_AnsiFromPWChar Method

Pascal

```
procedure _AnsiFromPWChar(var dst: AnsiStringBase; const src: wchar_t*; len: Integer;
codePage: Integer); static;
```

C++

```
static __fastcall void _AnsiFromPWChar(AnsiStringBase& dst, const wchar_t* src, int len, int codePage);
```

Description

This is _AnsiFromPWChar, a member of class AnsiStringBase.

35.1.1.7.3 AnsiStringBase::AnsiCompare Method

Compares the System::AnsiStringBase::AnsiStringBase (see page 3174) to another specified string (case-sensitively).

Pascal

```
function AnsiCompare(const rhs: AnsiStringBase&): Integer;
```

C++

```
__fastcall int AnsiCompare(const AnsiStringBase& rhs) const;
```

Description

Compares this System::AnsiStringBase::AnsiStringBase (see page 3174) to rhs, with case sensitivity. The compare operation is controlled by the current Windows locale and is not necessarily the same as using the System::AnsiStringBase (see page 3174) comparison operators. System::AnsiStringBase::AnsiCompare is based on a locale-specific collating order. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

35.1.1.7.4 AnsiStringBase::AnsiCompareIC Method

Compares the System::AnsiStringBase::AnsiStringBase (see page 3174) to another specified string (case insensitively).

Pascal

```
function AnsiCompareIC(const rhs: AnsiStringBase&): Integer;
```

C++

```
__fastcall int AnsiCompareIC(const AnsiStringBase& rhs) const;
```

Description

System::AnsiStringBase::AnsiCompareIC compares this System::AnsiStringBase::AnsiStringBase (see page 3174) to rhs, without case sensitivity. The compare operation is controlled by the current Windows locale. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

35.1.1.7.5 AnsiStringBase::AnsiLastChar Method

Returns a pointer to the last full character in the System::AnsiStringBase::AnsiStringBase (see page 3174).

Pascal

```
function AnsiLastChar: char*;
```

C++

```
__fastcall char* AnsiLastChar() const;
```

Description

Returns a pointer to the last full character in the System::AnsiStringBase::AnsiStringBase (see page 3174). This function supports multibyte characters.

See Also

System::AnsiStringBase::ByteType (see page 3187)

35.1.1.7.6 AnsiStringBase::AnsiPos Method

Returns the index at which the specified substring begins.

Pascal

```
function AnsiPos(const subStr: AnsiStringBase&): Integer;
```

C++

```
__fastcall int AnsiPos(const AnsiStringBase& subStr) const;
```

Description

Returns the index in the System::AnsiStringBase::AnsiStringBase (see page 3174) at which the substring subStr begins, where 1 is the first character in the string, 2 is the second character, and so on. If the substring is not contained in the System::AnsiStringBase::AnsiStringBase (see page 3174), Pos (see page 3198) returns 0. This member function is the same as Pos (see page 3198) but supports multibyte character strings.

Note: See ByteType (see page 3187) for more information about multibyte characters.

See Also

System::AnsiStringBase::ByteType (see page 3187)

35.1.1.7.7 AnsiStringBase::BytesOf Method

Get string's data in bytes.

Pascal

```
function BytesOf: DynamicArray<System::Byte>;
```

C++

```
__fastcall DynamicArray<System::Byte> BytesOf() const;
```

Description

System::AnsiStringBase::BytesOf gets a byte array copy of a System::AnsiStringBase (see page 3174) string's data. The data is copied as is, with no encoding or decoding.

See Also

System::UnicodeString::BytesOf (see page 3391)

BytesOf (see page 4068)

35.1.1.7.8 AnsiStringBase::ByteType Method

Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase (see page 3174).

Pascal

```
function ByteType(index: Integer): TStringMbcsByteType;
```

C++

```
__fastcall TStringMbcsByteType ByteType(int index) const;
```

Description

System::AnsiStringBase::ByteType indicates what kind of byte exists at the byte indicated by index (counting from 1) in the System::AnsiStringBase (see page 3174). These are the possible return values.

Constant	Meaning
mbSingleByte	Single byte character.
mbLeadByte	Lead byte of a multi-byte character.
mbTrailByte	Trailing byte of a multi-byte character.

Some locales, including all Western locales, do not use multi-byte characters. Under those locales, System::AnsiStringBase::ByteType always returns mbSingleByte.

Note: No checking is done to ensure that Index is less than the length of S. It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

LeadBytes (see page 4354)

StrByteType (see page 4242)

35.1.1.7.9 AnsiStringBase::cat_vprintf Method

Appends the result of a given format string and its arguments to the end of the System::AnsiStringBase::AnsiStringBase (see page 3174).

Pascal

```
function cat_vprintf(codePage: Integer; const format: char*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(int codePage, const char* format, va_list);
```

Description

Use cat_System::AnsiStringBase::cat_vprintf to append to the value of the System::AnsiStringBase::AnsiStringBase (see page 3174) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the number of bytes appended to the System::AnsiStringBase::AnsiStringBase (see page 3174).

See Also

System::AnsiStringBase::Format (see page 3191)

System::AnsiStringT::cat_printf (see page 3216)

System::AnsiStringT::cat_sprintf (see page 3216)

35.1.1.7.10 AnsiStringBase::CodePage Method

Gets code page of a System::AnsiStringBase (see page 3174).

Pascal

```
function CodePage: unsigned short;
```

C++

```
unsigned short CodePage() const;
```

Description

System::AnsiStringBase::CodePage gets the code page of a System::AnsiStringBase (see page 3174) instance. If the instance is empty, it returns the system default code page, System::DefaultSystemCodePage (see page 3815).

See Also

System::AnsiStringBase::GetRec (see page 3193)

35.1.1.7.11 AnsiStringBase::CurrToStr Method

Converts a currency value to its string representation.

Pascal

```
function CurrToStr(value: Currency; codePage: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase CurrToStr(Currency value, int codePage);
```

Description

System::AnsiStringBase::CurrToStr converts the currency value given by value to its string representation. The conversion using the sffGeneral floating-point conversion rules. For further details, see System::AnsiStringBase::CurrToStrF (see page 3188).

See Also

System::AnsiStringBase::CurrToStrF (see page 3188)

35.1.1.7.12 AnsiStringBase::CurrToStrF Method

Converts a currency value to its string representation using the indicated float format convention.

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer; codePage:
Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase CurrToStrF(Currency value, TStringFloatFormat format, int
digits, int codePage);
```

Description

System::AnsiStringBase::CurrToStrF converts the currency value given by value to its string representation using the indicated float format convention. A call to System::AnsiStringBase::CurrToStrF corresponds to a call to FloatToStrF (see page 3190) with an implied precision of 19 digits.

35.1.1.7.13 AnsiStringBase::Delete Method

Removes a specified number of characters from the string.

Pascal

```
function Delete(index: Integer; count: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& Delete(int index, int count);
```

Description

System::AnsiStringBase::Delete modifies the System::AnsiStringBase::AnsiStringBase (see page 3174) to remove count characters from the string beginning with the character at index, where 1 is the index of the first character. It returns the resulting modified string (*this).

If index is larger than the length of the System::AnsiStringBase::AnsiStringBase (see page 3174) or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, System::AnsiStringBase::Delete removes the rest of the string. If count is less than 0, no characters are deleted.

35.1.1.7.14 AnsiStringBase::ElementSize Method

Gets element size of data in a System::AnsiStringBase (see page 3174).

Pascal

```
function ElementSize: unsigned short;
```

C++

```
unsigned short ElementSize() const;
```

Description

System::AnsiStringBase::ElementSize gets the element size in bytes of the data in a System::AnsiStringBase (see page 3174) instance. If the instance is empty, this returns 1.

See Also

System::AnsiStringBase::Data (see page 3182)

System::AnsiStringBase::GetRec (see page 3193)

35.1.1.7.15 AnsiStringBase::EnsureAnsi Method

Ensure string's data is ANSI data.

Pascal

```
function EnsureAnsi: AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& EnsureAnsi();
```

Description

If the string's data is Unicode, i.e., its element length is 2, System::AnsiStringBase::EnsureAnsi converts the Unicode to ANSI using the current locale.

Note: In some cases, the conversion to an ANSI string may fail.

35.1.1.7.16 AnsiStringBase::FloatToStrF Method

Returns the string representation of a floating-point value.

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer; codePage: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits, int codePage);
```

Description

The format parameter controls the format of the resulting string. The precision parameter specifies the precision of the given value. It should be 7 or less for values of type float, 15 or less for values of type double, and 18 or less for values of type Extended. The meaning of the digits parameter depends on the particular format selected.

The possible values of the format parameter, and the meaning of each, are described below.

Value	Meaning
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 3235) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, and DecimalSeparator global variables, all of which are initialized from the Currency (see page 3235) Format (see page 3191) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18.

For all formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables.

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

35.1.1.7.17 AnsiStringBase::FmtLoadStr Method

Formats output using a resourced format string.

Pascal

```
function FmtLoadStr(ident: Integer; const args: TVarRec *; size: Integer; codePage: Integer):
AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase FmtLoadStr(int ident, const TVarRec * args, int size, int
codePage);
```

Description

System::AnsiStringBase::FmtLoadStr loads a string from a program's resource string table and uses that string, plus the args array, as a parameter to Format (see page 3191). Ident is the string resource ID of the desired format string.

See Also

System::AnsiStringBase::LoadString (see page 3197)

System::AnsiStringBase::LoadStr (see page 3197)

35.1.1.7.18 AnsiStringBase::Format Method

System::AnsiStringBase::Formats a string assembled from an System::AnsiStringBase (see page 3174), format, and a series of array arguments, args.

Pascal

```
function Format(const format: AnsiStringBase&; const args: TVarRec *; size: Integer; codePage:
Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase Format(const AnsiStringBase& format, const TVarRec * args,
int size, int codePage);
```

Description

Use System::AnsiStringBase::Format to obtain an System::AnsiStringBase (see page 3174) whose value is the result of applying an array of arguments (args) to a given a format string (format). The last parameter (size) is the index of the last element in args (one less than the number of arguments).

Note that System::AnsiStringBase::Format is a static method. The returned value is a newly allocated System::AnsiStringBase (see page 3174), even if the method is called from an existing System::AnsiStringBase (see page 3174) instance.

For information on how System::AnsiStringBase::Format assembles a final string from its format string and arguments, see System::AnsiStringBase::Format Strings.

See Also

System::AnsiStringBase::printf

System::AnsiStringBase::sprintf

35.1.1.7.19 AnsiStringBase::FormatFloat Method

Formats the floating-point value given by value using the format string given by format.

Pascal

```
function FormatFloat(const format: AnsiStringBase&; const value: long double&; codePage:
Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase FormatFloat(const AnsiStringBase& format, const long double&
```

```
value, int codePage);
```

Description

The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first dot character ('.') in the format string determines the location of the decimal separator in the formatted value; any additional dot characters are ignored. The actual character used as the decimal separator in the output string is determined by the DecimalSeparator global variable, whose default value is specified in the Number Format (see page 3191) of the International section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more comma characters (','), the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of comma characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as the thousand separator in the output is determined by the ThousandSeparator global variable, whose default value is specified in the Number Format (see page 3191) of the International section in the Windows Control Panel.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain from one to three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If no formatting is specified for positive values, positive values are formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 3190) with the sffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The following table shows some sample formats and the results produced when the formats are applied to different values:

Format (see page 3191) string-1234-12340.50

Format (see page 3191) string-	1234	-1234	0.5	0
1234	-1234	0.5	0	
0	1234	-1234	1	
0.00	1234.00	-1234.00	0.50	
#	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	
#,##0.00;;Zero	1,234.00	-1,234.00	0.50	
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	
#.###E-0	1.234E3	-1.234E3	5E-1	

35.1.1.7.20 GetRec Method

This is the overview for the GetRec method overload.

Overload List

	Name	Description
	AnsiStringBase::GetRec () (see page 3193)	Gets a struct describing a System::AnsiStringBase (see page 3174).
	AnsiStringBase::GetRec () (see page 3193)	Gets a struct describing a System::AnsiStringBase (see page 3174).

35.1.1.7.20.1 AnsiStringBase::GetRec Method ()

Gets a struct describing a System::AnsiStringBase (see page 3174).

Pascal

```
function GetRec: const StrRec&;
function GetRec: StrRec&;
```

C++

```
const StrRec& GetRec() const;
StrRec& GetRec();
```

Description

System::AnsiStringBase::GetRec gets a struct that describes the contents of a System::AnsiStringBase (see page 3174) instance. The struct has this format:

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

35.1.1.7.21 AnsiStringBase::Insert Method

System::AnsiStringBase::Inserts a specified string into the System::AnsiStringBase (see page 3174).

Pascal

```
function Insert(const str: AnsiStringBase&; index: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& Insert(const AnsiStringBase& str, int index);
```

Description

System::AnsiStringBase::Insert inserts the string str into this System::AnsiStringBase (see page 3174) beginning at the position index, where 1 is the first position in the string. It returns the resulting modified string (*this).

If index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to Length (see page 3196)(), turning the operation into an append.

If the str parameter is an empty string, System::AnsiStringBase::Insert does nothing.

System::AnsiStringBase::Insert throws an EOutOfMemory exception if the method is unable to allocate enough memory to accommodate the new, longer, string.

Note: The index parameter specifies the character, not the byte, when working with a multi-byte character system.

35.1.1.7.22 AnsiStringBase::IntToHex Method

Converts a number into a string containing the number's hexadecimal (base 16) representation.

Pascal

```
function IntToHex(value: Integer; digits: Integer; codePage: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase IntToHex(int value, int digits, int codePage);
```

Description

Value is the number to convert. Digits indicates the minimum number of hexadecimal digits.

35.1.1.7.23 AnsiStringBase::IsDelimiter Method

Returns true if the specified character matches any character in the delimiters string.

Pascal

```
function IsDelimiter(const delimiters: AnsiStringBase&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const AnsiStringBase& delimiters, int index) const;
```

Description

System::AnsiStringBase::IsDelimiter returns true if the byte indicated by index (counting from 1) is a single-byte character contained in delimiters. If the byte is part of a multi-byte character or not in delimiters, System::AnsiStringBase::IsDelimiter returns false.

35.1.1.7.24 AnsiStringBase::IsEmpty Method

Returns true if the System::AnsiStringBase (see page 3174) is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::AnsiStringBase::IsEmpty returns true if this System::AnsiStringBase (see page 3174) is empty, false otherwise.

35.1.1.7.25 AnsiStringBase::IsLeadByte Method

Returns true if the specified byte is a lead byte.

Pascal

```
function IsLeadByte(index: Integer): bool;
```

C++

```
__fastcall bool IsLeadByte(int index) const;
```

Description

System::AnsiStringBase::IsLeadByte returns true if the byte indicated by index (counting from 0) is the lead byte in a multi-byte character. If the byte is a trailing byte, or not part of a multi-byte character, System::AnsiStringBase::IsLeadByte returns false.

35.1.1.7.26 AnsiStringBase::IsPathDelimiter Method

Returns true if the specified character is '\'.

Pascal

```
function IsPathDelimiter(index: Integer): bool;
```

C++

```
__fastcall bool IsPathDelimiter(int index) const;
```

Description

System::AnsiStringBase::IsPathDelimiter returns true if the byte indicated by index (counting from 1) is a single-byte character containing the path delimiter ('\' on Windows, '/' on Linux).

See Also

System::AnsiStringBase::ByteType (see page 3187)

35.1.1.7.27 AnsiStringBase::IsTrailByte Method

Returns true if the specified byte is of type mbTrailByte.

Pascal

```
function IsTrailByte(index: Integer): bool;
```

C++

```
__fastcall bool IsTrailByte(int index) const;
```

Description

System::AnsiStringBase::IsTrailByte returns true if the byte indicated by index (counting from 0) is the trailing byte in a multi-byte character. If the byte is a lead byte, or not part of a multi-byte character, System::AnsiStringBase::IsTrailByte returns false.

35.1.1.7.28 AnsiStringBase::LastChar Method

Returns a pointer to the last character in a string.

Pascal

```
function LastChar: char*;
```

C++

```
__fastcall char* LastChar() const;
```

Description

Call System::AnsiStringBase::LastChar to obtain a pointer to the last character in a string. This function supports multi-byte character sets (MBCS).

See Also

System::AnsiStringBase::AnsiLastChar (see page 3186)

35.1.1.7.29 AnsiStringBase::LastDelimiter Method

Returns the index of the rightmost character that matches any character in the delimiters string.

Pascal

```
function LastDelimiter(const delimiters: AnsiStringBase&): Integer;
```

C++

```
__fastcall int LastDelimiter(const AnsiStringBase& delimiters) const;
```

Description

System::AnsiStringBase::LastDelimiter returns the byte index in this System::AnsiStringBase (see page 3174) of the rightmost whole character that matches any character in delimiters except NULL The System::AnsiStringBase (see page 3174) may contain multibyte characters; delimiters must contain only single byte non-NULL characters.

See Also

System::AnsiStringBase::ByteType (see page 3187)

35.1.1.7.30 AnsiStringBase::Length Method

Returns the length of the System::AnsiStringBase (see page 3174).

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::AnsiStringBase::Length returns the number of bytes in this System::AnsiStringBase (see page 3174).

35.1.1.7.31 LoadStr Method

This is the overview for the LoadStr method overload.

Overload List

	Name	Description
	AnsiStringBase::LoadStr (HINSTANCE, int, int) (see page 3197)	Loads a string from the compiled resources of a specified module.
	AnsiStringBase::LoadStr (int, int) (see page 3197)	Loads a string from the compiled resources of a specified module.

35.1.1.7.31.1 AnsiStringBase::LoadStr Method (HINSTANCE, int, int)

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadStr(ident: Integer; codePage: Integer): AnsiStringBase; static;
function LoadStr(hInstance: HINSTANCE; ident: Integer; codePage: Integer): AnsiStringBase;
static;
```

C++

```
static __fastcall AnsiStringBase LoadStr(int ident, int codePage);
static __fastcall AnsiStringBase LoadStr(HINSTANCE hInstance, int ident, int codePage);
```

Description

System::AnsiStringBase::LoadStr loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource. If hInstance is not given, System::AnsiStringBase::LoadStr loads the string from the current module (executable or shared library)

If the string resource does not exist, System::AnsiStringBase::LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiStringBase::LoadString (see page 3197)

System::AnsiStringBase::FmtLoadStr (see page 3190)

35.1.1.7.32 AnsiStringBase::LoadString Method

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadString(hInstance: HINSTANCE; ident: Integer; codePage: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& LoadString(HINSTANCE hInstance, int ident, int codePage);
```

Description

System::AnsiStringBase::LoadString loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource.

If the string resource does not exist, System::AnsiStringBase::LoadString returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiStringBase::LoadStr (see page 3197)

System::AnsiStringBase::FmtLoadStr (see page 3190)

35.1.1.7.33 AnsiStringBase::LowerCase Method

Returns a lower-case version of System::AnsiStringBase (see page 3174).

Pascal

```
function LowerCase(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase LowerCase(int codePage) const;
```

Description

System::AnsiStringBase::LowerCase returns a new System::AnsiStringBase (see page 3174) that contains all characters in this string, converted to lower case. The conversion uses the current locale . The original System::AnsiStringBase (see page 3174) remains unchanged.

35.1.1.7.34 AnsiStringBase::Pos Method

Returns the index at which a specified substring begins.

Pascal

```
function Pos(const subStr: AnsiStringBase&): Integer;
```

C++

```
__fastcall int Pos(const AnsiStringBase& subStr) const;
```

Description

System::AnsiStringBase::Pos returns the index in the System::AnsiStringBase (see page 3174) at which the substring subStr begins, where 1 is the index of the first character, 2 is the index of the second character, and so on. If the substring is not contained in the System::AnsiStringBase (see page 3174), System::AnsiStringBase::Pos returns 0.

35.1.1.7.35 AnsiStringBase::RefCount Method

Gets reference count of System::AnsiStringBase (see page 3174).

Pascal

```
function RefCount: Integer;
```

C++

```
int RefCount() const;
```

Description

System::AnsiStringBase::RefCount gets the reference count of a System::AnsiStringBase (see page 3174) instance. If the instance is empty, it returns 0.

See Also

System::AnsiStringBase::GetRec (see page 3193)

35.1.1.7.36 AnsiStringBase::SetLength Method

Changes the length of the System::AnsiStringBase (see page 3174).

Pascal

```
function SetLength(newLength: Integer; codePage: Integer): AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& SetLength(int newLength, int codePage);
```

Description

System::AnsiStringBase::SetLength changes the string's length to newLength, truncating if necessary.

System::AnsiStringBase::SetLength reallocates the underlying array of characters that stores the value of the System::AnsiStringBase (see page 3174). Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate the System::AnsiStringBase (see page 3174), an EOutOfMemory exception is thrown. Following a call to System::AnsiStringBase::SetLength, the System::AnsiStringBase (see page 3174) is guaranteed not to be a shared reference.

System::AnsiStringBase::SetLength returns the modified System::AnsiStringBase (see page 3174) object (*this)

35.1.1.7.37 AnsiStringBase::StringOfChar Method

Returns a string with the specified number of characters.

Pascal

```
function StringOfChar(ch: char; count: Integer): AnsiStringBase; static;
```

C++

```
static __fastcall AnsiStringBase StringOfChar(char ch, int count);
```

Description

System::AnsiStringBase::StringOfChar returns a string containing count characters with the character value given by ch. For example,

System::AnsiStringBase (see page 3174) s = System::AnsiStringBase::System::AnsiStringBase::StringOfChar('A', 10);

sets s to the string "AAAAAAAAAA".

35.1.1.7.38 AnsiStringBase::SubString Method

Returns a specified substring of the System::AnsiStringBase (see page 3174).

Pascal

```
function SubString(index: Integer; count: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase SubString(int index, int count) const;
```

Description

System::AnsiStringBase::SubString returns a new System::AnsiStringBase (see page 3174) that is a substring of this System::AnsiStringBase (see page 3174). The substring contains count characters beginning at index.

35.1.1.7.39 AnsiStringBase::swap Method

Swap string pointers.

Pascal

```
function swap(var other: AnsiStringBase): AnsiStringBase&;
```

C++

```
AnsiStringBase& swap(AnsiStringBase& other);
```

Description

System::AnsiStringBase::swap swaps two System::AnsiStringBase (see page 3174) strings. Pointers to the data are swapped, and the data is not copied. This makes the swap operation efficient for sorting.

35.1.1.7.40 AnsiStringBase::ThrowIfOutOfRange Method

Pascal

```
procedure ThrowIfOutOfRange(idx: Integer); cdecl;
```

C++

```
__cdecl void ThrowIfOutOfRange(int idx) const;
```

Description

This is ThrowIfOutOfRange, a member of class AnsiStringBase.

35.1.1.7.41 AnsiStringBase::ToDouble Method

Converts the string to a floating-point value.

Pascal

```
function ToDouble: double;
```

C++

```
__fastcall double ToDouble() const;
```

Description

System::AnsiStringBase::ToDouble converts the string to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks in the string are ignored. The DecimalSeparator global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, an EConvertError exception is thrown.

35.1.1.7.42 AnsiStringBase::ToInt Method

Converts a string to an integer, throwing an exception if the string is invalid.

Pascal

```
function ToInt: Integer;
```

C++

```
__fastcall int ToInt() const;
```

Description

System::AnsiStringBase::ToInt converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, an EConvertError exception is thrown.

See Also

System::AnsiStringBase::ToIntDef (see page 3201)

35.1.1.7.43 AnsiStringBase::ToIntDef Method

Converts the string to an integer, returning a default value if the string is invalid.

Pascal

```
function ToIntDef(defaultValue: Integer): Integer;
```

C++

```
__fastcall int ToIntDef(int defaultValue) const;
```

Description

System::AnsiStringBase::ToIntDef converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, the value given by defaultValue is returned.

See Also

System::AnsiStringBase::ToInt (see page 3200)

35.1.1.7.44 AnsiStringBase::Trim Method

Returns a new System::AnsiStringBase (see page 3174) without leading spaces, trailing spaces, or control characters.

Pascal

```
function Trim(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase Trim(int codePage) const;
```

Description

System::AnsiStringBase::Trim returns a new System::AnsiStringBase (see page 3174) removing leading and trailing spaces and control characters. Use System::AnsiStringBase::Trim to remove blank space before and after the first printing character.

35.1.1.7.45 AnsiStringBase::TrimLeft Method

Returns a new System::AnsiStringBase (see page 3174) without leading spaces or control characters.

Pascal

```
function TrimLeft(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase TrimLeft(int codePage) const;
```

Description

System::AnsiStringBase::TrimLeft returns a new System::AnsiStringBase (see page 3174) removing leading spaces and control characters. Use System::AnsiStringBase::TrimLeft to remove blank space before the first printing character.

35.1.1.7.46 AnsiStringBase::TrimRight Method

Returns a new System::AnsiStringBase (see page 3174) without trailing spaces or control characters.

Pascal

```
function TrimRight(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase TrimRight(int codePage) const;
```

Description

System::AnsiStringBase::TrimRight returns a new System::AnsiStringBase (see page 3174) removing trailing spaces and control characters. Use the System::AnsiStringBase::TrimRight function to remove blank space after the last printing character.

35.1.1.7.47 AnsiStringBase::Unique Method

Makes the System::AnsiStringBase (see page 3174) unique.

Pascal

```
function Unique: AnsiStringBase&;
```

C++

```
__fastcall AnsiStringBase& Unique();
```

Description

Use this function to make a string unique (refcnt == 1). System::AnsiStringBase::Unique sets the reference count on the string for the System::AnsiStringBase (see page 3174) object to 1 so that it is the only object referencing that string.

35.1.1.7.48 AnsiStringBase::UpperCase Method

Returns an upper-case version of this System::AnsiStringBase (see page 3174).

Pascal

```
function UpperCase(codePage: Integer): AnsiStringBase;
```

C++

```
__fastcall AnsiStringBase UpperCase(int codePage) const;
```

Description

System::AnsiStringBase::UpperCase returns a new System::AnsiStringBase (see page 3174) that contains all characters in this string, converted to upper case. The conversion uses the current locale . The original System::AnsiStringBase (see page 3174) remains unchanged.

35.1.1.7.49 AnsiStringBase::vprintf Method

Sets the value of the System::AnsiStringBase (see page 3174) given a format string and its arguments.

Pascal

```
function vprintf(codePage: Integer; const format: char*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(int codePage, const char* format, va_list);
```

Description

Use System::AnsiStringBase::vprintf to set to the value of the System::AnsiStringBase (see page 3174) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the length of the final formatted string.

See Also

System::AnsiStringBase::Format (see page 3191)

System::AnsiStringBase::sprintf

System::AnsiStringBase::printf

System::AnsiStringBase::cat_vprintf (see page 3187)

35.1.1.7.50 AnsiStringBase::WideChar Method

Converts the System::AnsiStringBase (see page 3174) to a wide-character array.

Pascal

```
function WideChar(var dest: wchar_t; destSize: Integer; codePage: Integer): wchar_t*;
```

C++

```
__fastcall wchar_t* WideChar(wchar_t* dest, int destSize, int codePage) const;
```

Description

System::AnsiStringBase::WideChar converts the System::AnsiStringBase (see page 3174) to an array of wide characters, and returns the resulting wide-character array.

dest is a buffer the caller allocates to hold the return value. Use the System::AnsiStringBase::WideCharBufSize (see page 3203) method to determine the size needed to hold the entire string value.

destSize is the size, in bytes, of dest.

System::AnsiStringBase::WideChar returns a pointer to the wide-character array that is written to dest.

See Also

System::AnsiStringBase::WideCharBufSize (see page 3203)

35.1.1.7.51 AnsiStringBase::WideCharBufSize Method

Returns the size of the buffer required by the WideChar (see page 3203) function.

Pascal

```
function WideCharBufSize(codePage: Integer): Integer;
```

C++

```
__fastcall int WideCharBufSize(int codePage) const;
```

Description

System::AnsiStringBase::WideCharBufSize returns an integer value indicating the size of the buffer required by the WideChar (see page 3203) method in order to convert this string.

See Also

System::AnsiStringBase::WideChar (see page 3203)

35.1.1.8 AnsiStringBase Operators

The operators of the AnsiStringBase class are listed here.

Protected Operators

	Name	Description
	!= (see page 3204)	Compares two strings.
	< (see page 3205)	Compares two strings.
	<= (see page 3206)	Compares two strings.
	= (see page 3206)	Assigns the string rhs to this System::AnsiStringBase (see page 3174).
	== (see page 3206)	Compares the System::AnsiStringBase (see page 3174) with a specified string, returning true if they are the same.
	> (see page 3207)	Compares two strings.
	>= (see page 3207)	Compares two strings.

Public Operators

	Name	Description
	[] (see page 3204)	This is the overview for the [] operator overload.
	+ (see page 3205)	Concatenates two System::AnsiStringBase (see page 3174) values.

35.1.1.8.1 AnsiStringBase::!= Operator

Compares two strings.

Pascal

```
operator !=(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator !=(const AnsiStringBase& rhs) const;
```

Description

Returns true if the strings are not the same, false otherwise.

Note: This System::AnsiStringBase (see page 3174)::!= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3185) or AnsiCompareIC (see page 3185) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3185)

System::AnsiStringBase::AnsiCompareIC (see page 3185)

35.1.1.8.2 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	AnsiStringBase::[] (int) (see page 3205)	Returns the character at a specified index in the string.
	AnsiStringBase::[] (int) (see page 3205)	Returns the character at a specified index in the string.

35.1.1.8.2.1 AnsiStringBase::[] Operator (int)

Returns the character at a specified index in the string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall char operator [](const int idx) const;
__fastcall char& operator [](const int idx);
```

Description

This System::AnsiStringBase (see page 3174)::[] returns the byte in the string at index value idx. The System::AnsiStringBase (see page 3174)::[] assumes a base index of 1.

Note: If the string contains multi-byte characters, the System::AnsiStringBase (see page 3174)::[] may return a byte value that represents the wrong character and/or a partial character. For character-level access, use the SubString (see page 3199) method.

See Also

System::AnsiStringBase::SubString (see page 3199)

35.1.1.8.3 AnsiStringBase::+ Operator

Concatenates two System::AnsiStringBase (see page 3174) values.

Pascal

```
operator +(const rhs: AnsiStringBase&);
```

C++

```
__fastcall AnsiStringBase operator +(const AnsiStringBase& rhs) const;
```

Description

The friend function returns an System::AnsiStringBase (see page 3174) that is the concatenation of strings lhs and rhs.

The method returns an System::AnsiStringBase (see page 3174) that is the concatenation of this System::AnsiStringBase (see page 3174) and rhs.

35.1.1.8.4 AnsiStringBase::< Operator

Compares two strings.

Pascal

```
operator <(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator <(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase::AnsiString is less than rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3174)::< performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3185) or AnsiCompareIC (see page 3185) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3185)

System::AnsiStringBase::AnsiCompareIC (see page 3185)

35.1.1.8.5 AnsiStringBase::<= Operator

Compares two strings.

Pascal

```
operator <=(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator <=(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase::AnsiString is less than or equal to rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3174)::<= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3185) or AnsiCompareIC (see page 3185) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3185)

System::AnsiStringBase::AnsiCompareIC (see page 3185)

35.1.1.8.6 AnsiStringBase::= Operator

Assigns the string rhs to this System::AnsiStringBase (see page 3174).

Pascal

```
operator =(const rhs: AnsiStringBase&);
```

C++

```
__fastcall AnsiStringBase& operator =(const AnsiStringBase& rhs);
```

Description

Use this System::AnsiStringBase (see page 3174)::= to assign the value of the System::AnsiStringBase (see page 3174) from another System::AnsiStringBase (see page 3174).

35.1.1.8.7 AnsiStringBase::== Operator

Compares the System::AnsiStringBase (see page 3174) with a specified string, returning true if they are the same.

Pascal

```
operator ==(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator ==(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase (see page 3174) is the same as rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3174)::== performs a simple byte-level numeric comparison. For locale

support, or to ignore case, use the AnsiCompare (see page 3185) or AnsiCompareIC (see page 3185) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3185)

System::AnsiStringBase::AnsiCompareIC (see page 3185)

35.1.1.8.8 AnsiStringBase::> Operator

Compares two strings.

Pascal

```
operator >(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator >(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase (see page 3174) is greater than rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3174)::> performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3185) or AnsiCompareIC (see page 3185) methods.

See Also

System::AnsiStringBase::AnsiCompare (see page 3185)

System::AnsiStringBase::AnsiCompareIC (see page 3185)

35.1.1.8.9 AnsiStringBase::>= Operator

Compares two strings.

Pascal

```
operator >=(const rhs: AnsiStringBase&);
```

C++

```
__fastcall bool operator >=(const AnsiStringBase& rhs) const;
```

Description

Returns true if this System::AnsiStringBase (see page 3174) is greater than or equal to rhs, false otherwise.

Note: This System::AnsiStringBase (see page 3174)::>= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3185) or AnsiCompareIC (see page 3185) methods.

See Also

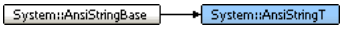
System::AnsiStringBase::AnsiCompare (see page 3185)

System::AnsiStringBase::AnsiCompareIC (see page 3185)

35.1.2 AnsiStringT Template

System::AnsiStringT is the C++ analog for the Delphi long string type.

Class Hierarchy



Pascal

```
AnsiStringT<unsigned short CP> = class(AnsiStringBase);
```

C++

```
template <unsigned short CP>
class AnsiStringT : public AnsiStringBase;
```

File

dstring.h

Description

System::AnsiStringT exists to provide a place for methods/members that are common to all instantiations of the template class AnsiStringT<CodePage (see page 3217)>. AnsiString itself is a typedef that is equivalent to AnsiStringT<0>. You should never deal directly with System::AnsiStringT.

System::AnsiStringT is a template class that you use when handling a Delphi AnsiString that has a code page specified with it.

Delphi uses several string types. One important long string type is commonly known as System::AnsiString. Support for this type includes the following features:

- strings can be as large as available memory
- efficient use of memory is ensured through shared references
- routines and operators evaluate strings based on the current locale

System::AnsiStringT is a template class that holds code for each specific code page. Despite its name, System::AnsiStringT is not restricted to the ANSI character set, and can use any character set supported by the current locale definition, including multi-byte or Unicode character sets.

System::AnsiStringT variables that have not been assigned an initial value contain a zero-length string.

To use the C++ streaming operators (**<<** and **>>**) with System::AnsiStringT, you must `iostream` before including `system.hpp` and use `AnsiStringT::c_str (see page 3216)()` to return the internal string representation. The following example demonstrates the usage of AnsiString with stream operators:

```
#include <iostream>
#include <system.hpp>
int main() {
  AnsiString HelloStr = "hello";
  std::cout << HelloStr.c_str();
}
```

Note: Delphi also supports System::AnsiString, but implements it as a primitive type rather than a class. By default, variables declared as type String are System::UnicodeString (see page 3381).

See Also

How to Handle Delphi AnsiString Code Page Specification in C++

System::WideString (see page 3467)

System::AnsiStringT::[]

System::AnsiStringT::= (see page 3233)

System::AnsiStringT::== (see page 3234)

System::AnsiStringT::!= (see page 3232)

System::AnsiStringT::> (see page 3234)

System::AnsiStringT::>= (see page 3234)

System::AnsiStringT::< (see page 3233)

System::AnsiStringT::<= (see page 3233)

System::AnsiStringT::+ (see page 3232)

System::AnsiStringT::+= (see page 3232)

35.1.2.1 AnsiStringT Members

The following tables list the members exposed by AnsiStringT.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3180)	System::AnsiStringBase::TStringFloatFormat is an enumerated list of formatting codes for float functions.
	TStringMbcsByteType (see page 3181)	Enumerated list of byte types in MCBS string.

Structures

	Name	Description
	StrRec (see page 3180)	This is record System::AnsiStringBase::StrRec.

Protected Data Members

	Name	Description
	Data (see page 3182)	This is Data, a member of class AnsiStringBase.

Protected Methods

	Name	Description
	_AnsiCat (see page 3184)	This is the overview for the _AnsiCat method overload.
	_AnsiFromPWChar (see page 3184)	This is _AnsiFromPWChar, a member of class AnsiStringBase.
	AnsiStringBase (see page 3178)	This is the overview for the AnsiStringBase constructor overload.
	GetRec (see page 3193)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3200)	This is ThrowIfOutOfRange, a member of class AnsiStringBase.

Public Methods

	Name	Description
	~AnsiStringBase (see page 3178)	Frees an instance of System::AnsiStringBase (see page 3174).
	~AnsiStringT (see page 3211)	This is ~AnsiStringT, a member of class AnsiStringT.
	AnsiCompare (see page 3214)	Compares the System::AnsiStringT (see page 3207) to another specified string (case-sensitively).
	AnsiCompareIC (see page 3215)	Compares the System::AnsiStringT (see page 3207) to another specified string (case insensitively).
	AnsiLastChar (see page 3186)	Returns a pointer to the last full character in the System::AnsiStringBase::AnsiStringBase (see page 3174).
	AnsiPos (see page 3215)	Returns the index at which the specified substring begins.
	AnsiStringT (see page 3211)	This is the overview for the AnsiStringT constructor overload.
	BytesOf (see page 3186)	Get string's data in bytes.
	ByteType (see page 3187)	Indicates the kind of byte that occupies a specified position in the System::AnsiStringBase (see page 3174).
	c_str (see page 3216)	Returns the System::AnsiStringT (see page 3207) converted to a null-terminated character array.
	cat_printf (see page 3216)	Appends to the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.

	cat_sprintf (see page 3216)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringT (see page 3207).
	cat_vprintf (see page 3217)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringT (see page 3207).
	CodePage (see page 3217)	Gets code page of a System::AnsiStringT (see page 3207).
	CurrToStr (see page 3218)	Converts a currency value to its string representation.
	CurrToStrF (see page 3218)	Converts a currency value to its string representation using the indicated float format convention.
	data (see page 3218)	Returns a pointer to the underlying string System::AnsiStringT::data.
	Delete (see page 3219)	Removes a specified number of characters from the string.
	ElementSize (see page 3219)	Gets element size of data (see page 3218) in a System::AnsiStringT (see page 3207).
	EnsureAnsi (see page 3220)	Ensure string's data (see page 3218) is ANSI data (see page 3218).
	FloatToStrF (see page 3220)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 3221)	Formats output using a resourced format string.
	Format (see page 3221)	System::AnsiStringT::Formats a string assembled from an System::AnsiStringT (see page 3207), format, and a series of array arguments, args.
	FormatFloat (see page 3222)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3223)	System::AnsiStringT::Inserts a specified string into the System::AnsiStringT (see page 3207).
	IntToHex (see page 3224)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3224)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 3194)	Returns true if the System::AnsiStringBase (see page 3174) is empty.
	IsLeadByte (see page 3195)	Returns true if the specified byte is a lead byte.
	IsPathDelimiter (see page 3195)	Returns true if the specified character is '\'.
	IsTrailByte (see page 3195)	Returns true if the specified byte is of type mbTrailByte.
	LastChar (see page 3196)	Returns a pointer to the last character in a string.
	LastDelimiter (see page 3224)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 3196)	Returns the length of the System::AnsiStringBase (see page 3174).
	LoadStr (see page 3196)	This is the overview for the LoadStr method overload.
	LoadStr (see page 3224)	This is the overview for the LoadStr method overload.
	LoadString (see page 3225)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 3226)	Returns a lower-case version of System::AnsiStringT (see page 3207).
	Pos (see page 3226)	Returns the index at which a specified substring begins.
	printf (see page 3226)	Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.
	RefCount (see page 3227)	Gets reference count of System::AnsiStringT (see page 3207).
	SetLength (see page 3227)	Changes the length of the System::AnsiStringT (see page 3207).
	sprintf (see page 3227)	Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.
	StringOfChar (see page 3228)	Returns a string with the specified number of characters.
	SubString (see page 3228)	Returns a specified substring of the System::AnsiStringT (see page 3207).
	swap (see page 3228)	Swap string pointers.
	ToDouble (see page 3200)	Converts the string to a floating-point value.
	ToInt (see page 3200)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3201)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3229)	Returns a new System::AnsiStringT (see page 3207) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3229)	Returns a new System::AnsiStringT (see page 3207) without leading spaces or control characters.
	TrimRight (see page 3229)	Returns a new System::AnsiStringT (see page 3207) without trailing spaces or control characters.
	Unique (see page 3229)	Makes the System::AnsiStringT (see page 3207) unique.
	UpperCase (see page 3230)	Returns an upper-case version of this System::AnsiStringT (see page 3207).

	vprintf (see page 3230)	Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.
	WideChar (see page 3230)	Converts the System::AnsiStringT (see page 3207) to a wide-character array.
	WideCharBufSize (see page 3231)	Returns the size of the buffer required by the WideChar (see page 3230) function.

Public Operators

	Name	Description
	!= (see page 3232)	Compares two strings.
	[] (see page 3204)	This is the overview for the [] operator overload.
	+ (see page 3232)	Concatenates two System::AnsiStringT (see page 3207) values.
	+= (see page 3232)	Append (see page 3520) an System::AnsiStringT (see page 3207).
	< (see page 3233)	Compares two strings.
	<= (see page 3233)	Compares two strings.
	= (see page 3233)	Assigns the string rhs to this System::AnsiStringT (see page 3207).
	== (see page 3234)	Compares the System::AnsiStringT (see page 3207) with a specified string, returning true if they are the same.
	> (see page 3234)	Compares two strings.
	>= (see page 3234)	Compares two strings.

35.1.2.2 AnsiStringT::~AnsiStringT Destructor

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~AnsiStringT();
```

Description

This is ~AnsiStringT, a member of class AnsiStringT.

35.1.2.3 AnsiStringT Constructor

This is the overview for the AnsiStringT constructor overload.

Overload List

	Name	Description
	AnsiStringT::AnsiStringT () (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (__int64) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (AnsiStringBase&) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (AnsiStringT&) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (AnsiStringT<OTHER_CP>&) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (char) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).

	AnsiStringT::AnsiStringT (char*) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (char*, int) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (char16_t*, int) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (char32_t*, int) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (double) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (int) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (long) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (short) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (UnicodeString &) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (unsigned __int64) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (unsigned int) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (unsigned long) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (unsigned short) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (wchar_t) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (wchar_t*, int) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).
	AnsiStringT::AnsiStringT (WideString &) (see page 3212)	Creates an instance of System::AnsiStringT (see page 3207).

35.1.2.3.1 AnsiStringT::AnsiStringT Constructor ()

Creates an instance of System::AnsiStringT (see page 3207).

Pascal

```
constructor Create;
constructor Create(const src: char*);
constructor Create(const src: AnsiStringT&);
constructor Create(const src: AnsiStringBase&);
constructor Create<unsigned short OTHER_CP>(const src: AnsiStringT<OTHER_CP>&);
constructor Create(const src: char*; byteLen: Integer);
constructor Create(const src: wchar_t*; numwchar: Integer = -1);
constructor Create(const src: char16_t*; numChar16: Integer = -1);
constructor Create(const src: char32_t*; numChar32: Integer = -1);
constructor Create(src: char);
constructor Create(src: wchar_t);
constructor Create(src: short);
```

```
constructor Create(src: unsigned short);
constructor Create(src: Integer);
constructor Create(src: unsigned int);
constructor Create(src: long);
constructor Create(src: unsigned long);
constructor Create(src: __int64);
constructor Create(src: unsigned __int64);
constructor Create(src: double);
constructor Create(const src: WideString &);
constructor Create(const src: UnicodeString &);
```

C++

```
__fastcall AnsiStringT();
__fastcall AnsiStringT(const char* src);
__fastcall AnsiStringT(const AnsiStringT& src);
__fastcall AnsiStringT(const AnsiStringBase& src);
template <unsigned short OTHER_CP> AnsiStringT<CP>(const AnsiStringT<OTHER_CP>& src);
__fastcall AnsiStringT(const char* src, int byteLen);
__fastcall AnsiStringT(const wchar_t* src, int numwchar = -1);
__fastcall AnsiStringT(const char16_t* src, int numChar16 = -1);
__fastcall AnsiStringT(const char32_t* src, int numChar32 = -1);
__fastcall AnsiStringT(char src);
__fastcall AnsiStringT(wchar_t src);
__fastcall AnsiStringT(short src);
__fastcall AnsiStringT(unsigned short src);
__fastcall AnsiStringT(int src);
__fastcall AnsiStringT(unsigned int src);
__fastcall AnsiStringT(long src);
__fastcall AnsiStringT(unsigned long src);
__fastcall AnsiStringT(__int64 src);
__fastcall AnsiStringT(unsigned __int64 src);
__fastcall AnsiStringT(double src);
__fastcall AnsiStringT(const WideString & src);
__fastcall AnsiStringT(const UnicodeString & src);
```

Description

System::AnsiStringT::AnsiStringT (see page 3207) is the constructor for System::AnsiStringT (see page 3207).

If the constructor is given no arguments, the resulting System::AnsiStringT (see page 3207) instance is an empty string. If the constructor is passed a single character, the resulting System::AnsiStringT (see page 3207) instance has character length one, and contains that character. Other versions of the constructor copy the string passed as an argument, converting characters to `char` if necessary. When constructing a System::AnsiStringT (see page 3207) instance from an array of `char` or `wchar_t`, the source string should be null terminated, unless an argument is given that indicates the number of characters in the array. For some overloads of the constructor, you can pass the code page you want for the string. In another overload, you can pass a floating point number to be converted to a string.

35.1.2.4 AnsiStringT Methods

The methods of the AnsiStringT class are listed here.

Public Methods

	Name	Description
	AnsiCompare (see page 3214)	Compares the System::AnsiStringT (see page 3207) to another specified string (case-sensitively).
	AnsiCompareIC (see page 3215)	Compares the System::AnsiStringT (see page 3207) to another specified string (case insensitively).
	AnsiPos (see page 3215)	Returns the index at which the specified substring begins.
	c_str (see page 3216)	Returns the System::AnsiStringT (see page 3207) converted to a null-terminated character array.

	cat_printf (see page 3216)	Appends to the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.
	cat_sprintf (see page 3216)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringT (see page 3207).
	cat_vprintf (see page 3217)	Appends the result of a given format string and its arguments to the end of the System::AnsiStringT (see page 3207).
	CodePage (see page 3217)	Gets code page of a System::AnsiStringT (see page 3207).
	CurrToStr (see page 3218)	Converts a currency value to its string representation.
	CurrToStrF (see page 3218)	Converts a currency value to its string representation using the indicated float format convention.
	data (see page 3218)	Returns a pointer to the underlying string System::AnsiStringT::data.
	Delete (see page 3219)	Removes a specified number of characters from the string.
	ElementSize (see page 3219)	Gets element size of data (see page 3218) in a System::AnsiStringT (see page 3207).
	EnsureAnsi (see page 3220)	Ensure string's data (see page 3218) is ANSI data (see page 3218).
	FloatToStrF (see page 3220)	Returns the string representation of a floating-point value.
	FmtLoadStr (see page 3221)	Formats output using a resourced format string.
	Format (see page 3221)	System::AnsiStringT::Formats a string assembled from an System::AnsiStringT (see page 3207), format, and a series of array arguments, args.
	FormatFloat (see page 3222)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3223)	System::AnsiStringT::Inserts a specified string into the System::AnsiStringT (see page 3207).
	IntToHex (see page 3224)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3224)	Returns true if the specified character matches any character in the delimiters string.
	LastDelimiter (see page 3224)	Returns the index of the rightmost character that matches any character in the delimiters string.
	LoadStr (see page 3224)	This is the overview for the LoadStr method overload.
	LoadString (see page 3225)	Loads a string from the compiled resources of a specified module.
	LowerCase (see page 3226)	Returns a lower-case version of System::AnsiStringT (see page 3207).
	Pos (see page 3226)	Returns the index at which a specified substring begins.
	printf (see page 3226)	Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.
	RefCount (see page 3227)	Gets reference count of System::AnsiStringT (see page 3207).
	SetLength (see page 3227)	Changes the length of the System::AnsiStringT (see page 3207).
	sprintf (see page 3227)	Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.
	StringOfChar (see page 3228)	Returns a string with the specified number of characters.
	SubString (see page 3228)	Returns a specified substring of the System::AnsiStringT (see page 3207).
	swap (see page 3228)	Swap string pointers.
	Trim (see page 3229)	Returns a new System::AnsiStringT (see page 3207) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3229)	Returns a new System::AnsiStringT (see page 3207) without leading spaces or control characters.
	TrimRight (see page 3229)	Returns a new System::AnsiStringT (see page 3207) without trailing spaces or control characters.
	Unique (see page 3229)	Makes the System::AnsiStringT (see page 3207) unique.
	UpperCase (see page 3230)	Returns an upper-case version of this System::AnsiStringT (see page 3207).
	vprintf (see page 3230)	Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.
	WideChar (see page 3230)	Converts the System::AnsiStringT (see page 3207) to a wide-character array.
	WideCharBufSize (see page 3231)	Returns the size of the buffer required by the WideChar (see page 3230) function.

35.1.2.4.1 AnsiStringT::AnsiCompare Method

Compares the System::AnsiStringT (see page 3207) to another specified string (case-sensitively).

Pascal

```
function AnsiCompare(const rhs: AnsiStringT&): Integer;
```

C++

```
__fastcall int AnsiCompare(const AnsiStringT& rhs) const;
```

Description

Compares this System::AnsiStringT (see page 3207) to rhs, with case sensitivity. The compare operation is controlled by the current Windows locale and is not necessarily the same as using the System::AnsiStringT (see page 3207) comparison operators. System::AnsiStringT::AnsiCompare is based on a locale-specific collating order. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

35.1.2.4.2 AnsiStringT::AnsiCompareIC Method

Compares the System::AnsiStringT (see page 3207) to another specified string (case insensitively).

Pascal

```
function AnsiCompareIC(const rhs: AnsiStringT&): Integer;
```

C++

```
__fastcall int AnsiCompareIC(const AnsiStringT& rhs) const;
```

Description

System::AnsiStringT::AnsiCompareIC compares this System::AnsiStringT (see page 3207) to rhs, without case sensitivity. The compare operation is controlled by the current Windows locale. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

35.1.2.4.3 AnsiStringT::AnsiPos Method

Returns the index at which the specified substring begins.

Pascal

```
function AnsiPos(const subStr: AnsiStringT&): Integer;
```

C++

```
__fastcall int AnsiPos(const AnsiStringT& subStr) const;
```

Description

Returns the index in the System::AnsiStringT (see page 3207) at which the substring subStr begins, where 1 is the first character in the string, 2 is the second character, and so on. If the substring is not contained in the System::AnsiStringT (see page 3207), Pos (see page 3226) returns 0. This member function is the same as Pos (see page 3226) but supports multibyte character strings.

Note: See ByteType (see page 3187) for more information about multibyte characters.

See Also

System::AnsiStringBase::ByteType (see page 3187)

35.1.2.4.4 AnsiStringT::c_str Method

Returns the System::AnsiStringT (see page 3207) converted to a null-terminated character array.

Pascal

```
function c_str: char*;
```

C++

```
__fastcall char* c_str() const;
```

Description

System::AnsiStringT::c_str returns a pointer to a null-terminated character array containing the same characters as the System::AnsiStringT (see page 3207). If the AnsiString is unassigned, c_str returns an empty string (“”).

Usually, the value returned by System::AnsiStringT::c_str points to the internal character array referenced by the data (see page 3218) property. This pointer is valid until the string is next modified (for example when the SetLength (see page 3227) method is called or the AnsiString goes out of scope). However, if the internal array is NULL, c_str() returns a pointer to a string literal.

The System::AnsiStringT::c_str method is intended primarily for reading the value of the AnsiString. To modify the string’s value, use the [] operator or System::AnsiStringT (see page 3207) methods such as Insert (see page 3223) and Delete (see page 3219).

35.1.2.4.5 AnsiStringT::cat_printf Method

Appends to the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.

Pascal

```
function cat_printf(const format: char*; : ...): Integer; cdecl;
```

C++

```
__cdecl int cat_printf(const char* format, ...);
```

Description

Use cat_System::AnsiStringT::cat_printf to modify the value of the System::AnsiStringT (see page 3207) by appending the result of using a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the number of bytes appended to the System::AnsiStringT (see page 3207).

See Also

System::AnsiStringT::Format (see page 3221)

System::AnsiStringT::cat_sprintf (see page 3216)

System::AnsiStringT::cat_vprintf (see page 3217)

35.1.2.4.6 AnsiStringT::cat_sprintf Method

Appends the result of a given format string and its arguments to the end of the System::AnsiStringT (see page 3207).

Pascal

```
function cat_sprintf(const format: char*; : ...): AnsiStringT&; cdecl;
```

C++

```
__cdecl AnsiStringT& cat_sprintf(const char* format, ...);
```

Description

Use cat_System::AnsiStringT::cat_sprintf to append to the value of the System::AnsiStringT (see page 3207) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

For example, the following code

```
AnsiString Salutation = "Mr."
AnsiString s = "";
s.cat_sprintf("Dear %s ", Salutation);
s.cat_sprintf("%s,", "Smith");
```

results in an System::AnsiStringT (see page 3207) with the value

```
Dear Mr. Smith,
```

This method returns a reference to the System::AnsiStringT (see page 3207) (*this) with the resulting value.

See Also

System::AnsiStringT::Format (see page 3221)

System::AnsiStringT::cat_printf (see page 3216)

System::AnsiStringT::cat_vprintf (see page 3217)

35.1.2.4.7 AnsiStringT::cat_vprintf Method

Appends the result of a given format string and its arguments to the end of the System::AnsiStringT (see page 3207).

Pascal

```
function cat_vprintf(const format: char*; list: va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(const char* format, va_list list);
```

Description

Use cat_System::AnsiStringT::cat_vprintf to append to the value of the System::AnsiStringT (see page 3207) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the number of bytes appended to the System::AnsiStringT (see page 3207).

See Also

System::AnsiStringT::Format (see page 3221)

System::AnsiStringT::cat_printf (see page 3216)

System::AnsiStringT::cat_sprintf (see page 3216)

35.1.2.4.8 AnsiStringT::CodePage Method

Gets code page of a System::AnsiStringT (see page 3207).

Pascal

```
function CodePage: unsigned short;
```

C++

```
unsigned short CodePage() const;
```

Description

System::AnsiStringT::CodePage gets the code page of a System::AnsiStringT (see page 3207) instance. If the instance is empty, it returns the system default code page, System::DefaultSystemCodePage (see page 3815).

See Also

System::AnsiStringBase::GetRec (see page 3193)

35.1.2.4.9 AnsiStringT::CurrToStr Method

Converts a currency value to its string representation.

Pascal

```
function CurrToStr(value: Currency): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT CurrToStr(Currency value);
```

Description

System::AnsiStringT::CurrToStr converts the currency value given by value to its string representation. The conversion using the sffGeneral floating-point conversion rules. For further details, see System::AnsiStringT::CurrToStrF (see page 3218).

See Also

System::AnsiStringT::CurrToStrF (see page 3218)

35.1.2.4.10 AnsiStringT::CurrToStrF Method

Converts a currency value to its string representation using the indicated float format convention.

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer):
AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT CurrToStrF(Currency value, TStringFloatFormat format, int
digits);
```

Description

System::AnsiStringT::CurrToStrF converts the currency value given by value to its string representation using the indicated float format convention. A call to System::AnsiStringT::CurrToStrF corresponds to a call to FloatToStrF (see page 3220) with an implied precision of 19 digits.

35.1.2.4.11 AnsiStringT::data Method

Returns a pointer to the underlying string System::AnsiStringT::data.

Pascal

```
function data: const void*;
```

C++

```
__fastcall const void* data() const;
```

Description

System::AnsiStringT::data returns a pointer to a null-terminated character array that the System::AnsiStringT (see page 3207) uses to store its value. This is the same value as the one returned by the System::AnsiStringT::c_str (see page 3216) method, except that System::AnsiStringT::data returns a NULL pointer when the value is an empty string.

Note: Because System::AnsiStringT::data returns NULL instead of an empty string, it is useful for C++ functions that expect either a valid (non-empty) char* or NULL. The following example assumes that TCHAR mapping is not set to wchar_t:

```
AnsiString x;
    ShellExecute(0, static_cast<const char*>(x.data()), "File.txt", NULL, NULL, SW_SHOW);
```

If TCHAR mapping is set to wchar_t, ShellExecute becomes the wide version, ShellExecuteW. If you want to use the Ansi version of ShellExecute when TCHAR mapping is set to wchar_t, you should specify ShellExecuteA.

See Also

System::AnsiStringT::c_str (see page 3216)

TCHAR Mapping

35.1.2.4.12 AnsiStringT::Delete Method

Removes a specified number of characters from the string.

Pascal

```
function Delete(index: Integer; count: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& Delete(int index, int count);
```

Description

System::AnsiStringT::Delete modifies the System::AnsiStringT (see page 3207) to remove count characters from the string beginning with the character at index, where 1 is the index of the first character. It returns the resulting modified string (*this).

If index is larger than the length of the System::AnsiStringT (see page 3207) or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, System::AnsiStringT::Delete removes the rest of the string. If count is less than 0, no characters are deleted.

35.1.2.4.13 AnsiStringT::ElementSize Method

Gets element size of data (see page 3218) in a System::AnsiStringT (see page 3207).

Pascal

```
function ElementSize: unsigned short;
```

C++

```
unsigned short ElementSize() const;
```

Description

System::AnsiStringT::ElementSize gets the element size in bytes of the data (see page 3218) in a System::AnsiStringT (see page 3207) instance. If the instance is empty, this returns 1.

See Also

System::AnsiStringT::data (see page 3218)

System::AnsiStringBase::GetRec (see page 3193)

35.1.2.4.14 AnsiStringT::EnsureAnsi Method

Ensure string's data (see page 3218) is ANSI data (see page 3218).

Pascal

```
function EnsureAnsi: AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& EnsureAnsi();
```

Description

If the string's data (see page 3218) is Unicode, i.e., its element length is 2, System::AnsiStringT::EnsureAnsi converts the Unicode to ANSI using the current locale.

Note: In some cases, the conversion to an ANSI string may fail.

35.1.2.4.15 AnsiStringT::FloatToStrF Method

Returns the string representation of a floating-point value.

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits);
```

Description

The format parameter controls the format of the resulting string. The precision parameter specifies the precision of the given value. It should be 7 or less for values of type float, 15 or less for values of type double, and 18 or less for values of type Extended. The meaning of the digits parameter depends on the particular format selected.

The possible values of the format parameter, and the meaning of each, are described below.

Value	Meaning
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).

sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 3235) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, and DecimalSeparator global variables, all of which are initialized from the Currency (see page 3235) Format (see page 3221) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18.

For all formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables.

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

35.1.2.4.16 AnsiStringT::FmtLoadStr Method

Formats output using a resourced format string.

Pascal

```
function FmtLoadStr(ident: Integer; const args: TVarRec *; size: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT FmtLoadStr(int ident, const TVarRec * args, int size);
```

Description

System::AnsiStringT::FmtLoadStr loads a string from a program's resource string table and uses that string, plus the args array, as a parameter to Format (see page 3221). Ident is the string resource ID of the desired format string.

See Also

System::AnsiStringT::LoadString (see page 3225)

System::AnsiStringT::LoadStr (see page 3225)

35.1.2.4.17 AnsiStringT::Format Method

System::AnsiStringT::Formats a string assembled from an System::AnsiStringT (see page 3207), format, and a series of array arguments, args.

Pascal

```
function Format(const format: AnsiStringT&; const args: TVarRec *; size: Integer):
AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT Format(const AnsiStringT& format, const TVarRec * args, int
size);
```

Description

Use System::AnsiStringT::Format to obtain an System::AnsiStringT (see page 3207) whose value is the result of applying an array of arguments (args) to a given a format string (format). The last parameter (size) is the index of the last element in args (one less than the number of arguments).

Note that System::AnsiStringT::Format is a static method. The returned value is a newly allocated System::AnsiStringT (see page 3207), even if the method is called from an existing System::AnsiStringT (see page 3207) instance.

For information on how System::AnsiStringT::Format assembles a final string from its format string and arguments, see System::AnsiStringT::Format Strings.

See Also

System::AnsiStringT::printf (see page 3226)

System::AnsiStringT::sprintf (see page 3227)

35.1.2.4.18 AnsiStringT::FormatFloat Method

Formats the floating-point value given by value using the format string given by format.

Pascal

```
function FormatFloat(const format: AnsiStringT&; const value: long double&): AnsiStringT;
static;
```

C++

```
static __fastcall AnsiStringT FormatFloat(const AnsiStringT& format, const long double& value);
```

Description

The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first dot character ('.') in the format string determines the location of the decimal separator in the formatted value; any additional dot characters are ignored. The actual character used as the decimal separator in the output string is determined by the DecimalSeparator global variable, whose default value is specified in the Number Format (see page 3221) of the International section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more comma characters (','), the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of comma characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as the thousand separator in the output is determined by the ThousandSeparator global variable, whose default value is specified in the Number Format (see page 3221) of the International section in the Windows Control Panel.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain from one to three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If no formatting is specified for positive values, positive values are formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 3220) with the sffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The following table shows some sample formats and the results produced when the formats are applied to different values:

Format (see page 3221) string-1234-12340.50

Format (see page 3221) string-	**1234**	**-1234**	**0.5**	**0**
1234	-1234	0.5	0	
0	1234	-1234	1	
0.00	1234.00	-1234.00	0.50	
#	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	
#,##0.00;;Zero	1,234.00	-1,234.00	0.50	
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	
#.###E-0	1.234E3	-1.234E3	5E-1	

35.1.2.4.19 AnsiStringT::Insert Method

System::AnsiStringT::Inserts a specified string into the System::AnsiStringT (see page 3207).

Pascal

```
function Insert(const str: AnsiStringT&; index: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& Insert(const AnsiStringT& str, int index);
```

Description

System::AnsiStringT::Insert inserts the string str into this System::AnsiStringT (see page 3207) beginning at the position index, where 1 is the first position in the string. It returns the resulting modified string (*this).

If index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to Length (see page 3607)(), turning the operation into an append.

If the str parameter is an empty string, System::AnsiStringT::Insert does nothing.

System::AnsiStringT::Insert throws an EOutOfMemory exception if the method is unable to allocate enough memory to accommodate the new, longer, string.

Note: The index parameter specifies the character, not the byte, when working with a multi-byte character system.

35.1.2.4.20 AnsiStringT::IntToHex Method

Converts a number into a string containing the number's hexadecimal (base 16) representation.

Pascal

```
function IntToHex(value: Integer; digits: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT IntToHex(int value, int digits);
```

Description

Value is the number to convert. Digits indicates the minimum number of hexadecimal digits.

35.1.2.4.21 AnsiStringT::IsDelimiter Method

Returns true if the specified character matches any character in the delimiters string.

Pascal

```
function IsDelimiter(const delimiters: AnsiStringT&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const AnsiStringT& delimiters, int index) const;
```

Description

System::AnsiStringT::IsDelimiter returns true if the byte indicated by index (counting from 1) is a single-byte character contained in delimiters. If the byte is part of a multi-byte character or not in delimiters, System::AnsiStringT::IsDelimiter returns false.

35.1.2.4.22 AnsiStringT::LastDelimiter Method

Returns the index of the rightmost character that matches any character in the delimiters string.

Pascal

```
function LastDelimiter(const delimiters: AnsiStringT&): Integer;
```

C++

```
__fastcall int LastDelimiter(const AnsiStringT& delimiters) const;
```

Description

System::AnsiStringT::LastDelimiter returns the byte index in this System::AnsiStringT (see page 3207) of the rightmost whole character that matches any character in delimiters except NULL The System::AnsiStringT (see page 3207) may contain multibyte characters; delimiters must contain only single byte non-NULL characters.

See Also

System::AnsiStringBase::ByteType (see page 3187)

35.1.2.4.23 LoadStr Method

This is the overview for the LoadStr method overload.

Overload List

	Name	Description
	AnsiStringT::LoadStr (HINSTANCE, int) (see page 3225)	Loads a string from the compiled resources of a specified module.
	AnsiStringT::LoadStr (int) (see page 3225)	Loads a string from the compiled resources of a specified module.

35.1.2.4.23.1 AnsiStringT::LoadStr Method (HINSTANCE, int)

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadStr(ident: Integer): AnsiStringT; static;
function LoadStr(hInstance: HINSTANCE; ident: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT LoadStr(int ident);
static __fastcall AnsiStringT LoadStr(HINSTANCE hInstance, int ident);
```

Description

System::AnsiStringT::LoadStr loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource. If hInstance is not given, System::AnsiStringT::LoadStr loads the string from the current module (executable or shared library)

If the string resource does not exist, System::AnsiStringT::LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiStringT::LoadString (see page 3225)

System::AnsiStringT::FmtLoadStr (see page 3221)

35.1.2.4.24 AnsiStringT::LoadString Method

Loads a string from the compiled resources of a specified module.

Pascal

```
function LoadString(hInstance: HINSTANCE; ident: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& LoadString(HINSTANCE hInstance, int ident);
```

Description

System::AnsiStringT::LoadString loads a specified string resource.

ident is the resource identifier of the string.

hInstance is the instance handle of the module that contains the string resource.

If the string resource does not exist, System::AnsiStringT::LoadString returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::AnsiStringT::LoadStr (see page 3225)

System::AnsiStringT::FmtLoadStr (see page 3221)

35.1.2.4.25 AnsiStringT::LowerCase Method

Returns a lower-case version of System::AnsiStringT (see page 3207).

Pascal

```
function LowerCase: AnsiStringT;
```

C++

```
__fastcall AnsiStringT LowerCase() const;
```

Description

System::AnsiStringT::LowerCase returns a new System::AnsiStringT (see page 3207) that contains all characters in this string, converted to lower case. The conversion uses the current locale . The original System::AnsiStringT (see page 3207) remains unchanged.

35.1.2.4.26 AnsiStringT::Pos Method

Returns the index at which a specified substring begins.

Pascal

```
function Pos(const subStr: AnsiStringT&): Integer;
```

C++

```
__fastcall int Pos(const AnsiStringT& subStr) const;
```

Description

System::AnsiStringT::Pos returns the index in the System::AnsiStringT (see page 3207) at which the substring subStr begins, where 1 is the index of the first character, 2 is the index of the second character, and so on. If the substring is not contained in the System::AnsiStringT (see page 3207), System::AnsiStringT::Pos returns 0.

35.1.2.4.27 AnsiStringT::printf Method

Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.

Pascal

```
function printf(const format: char*; : ...): Integer; cdecl;
```

C++

```
__cdecl int printf(const char* format, ...);
```

Description

Use System::AnsiStringT::printf to set the value of the System::AnsiStringT (see page 3207) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the length of the final formatted string.

See Also

System::AnsiStringT::Format (see page 3221)

System::AnsiStringT::sprintf (see page 3227)

System::AnsiStringT::cat_printf (see page 3216)

System::AnsiStringT::vprintf (see page 3230)

35.1.2.4.28 AnsiStringT::RefCount Method

Gets reference count of System::AnsiStringT (see page 3207).

Pascal

```
function RefCount: Integer;
```

C++

```
int RefCount() const;
```

Description

System::AnsiStringT::RefCount gets the reference count of a System::AnsiStringT (see page 3207) instance. If the instance is empty, it returns 0.

See Also

System::AnsiStringBase::GetRec (see page 3193)

35.1.2.4.29 AnsiStringT::SetLength Method

Changes the length of the System::AnsiStringT (see page 3207).

Pascal

```
function SetLength(newLength: Integer): AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& SetLength(int newLength);
```

Description

System::AnsiStringT::SetLength changes the string's length to newLength, truncating if necessary.

System::AnsiStringT::SetLength reallocates the underlying array of characters that stores the value of the System::AnsiStringT (see page 3207). Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate the System::AnsiStringT (see page 3207), an EOutOfMemory exception is thrown. Following a call to System::AnsiStringT::SetLength, the System::AnsiStringT (see page 3207) is guaranteed not to be a shared reference.

System::AnsiStringT::SetLength returns the modified System::AnsiStringT (see page 3207) object (*this)

35.1.2.4.30 AnsiStringT::sprintf Method

Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.

Pascal

```
function sprintf(const format: char*; : ...): AnsiStringT&; cdecl;
```

C++

```
__cdecl AnsiStringT& sprintf(const char* format, ...);
```

Description

Use System::AnsiStringT::sprintf to set to the value of the System::AnsiStringT (see page 3207) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

This method returns a reference to the System::AnsiStringT (see page 3207) (*this) with the resulting value.

See Also

System::AnsiStringT::Format (see page 3221)

System::AnsiStringT::cat_sprintf (see page 3216)

System::AnsiStringT::printf (see page 3226)

System::AnsiStringT::vprintf (see page 3230)

35.1.2.4.31 AnsiStringT::StringOfChar Method

Returns a string with the specified number of characters.

Pascal

```
function StringOfChar(ch: char; count: Integer): AnsiStringT; static;
```

C++

```
static __fastcall AnsiStringT StringOfChar(char ch, int count);
```

Description

System::AnsiStringT::StringOfChar returns a string containing count characters with the character value given by ch. For example,

System::AnsiStringT (see page 3207) s = System::AnsiStringT::System::AnsiStringT::StringOfChar('A', 10);

sets s to the string "AAAAAAAAAA".

35.1.2.4.32 AnsiStringT::SubString Method

Returns a specified substring of the System::AnsiStringT (see page 3207).

Pascal

```
function SubString(index: Integer; count: Integer): AnsiStringT;
```

C++

```
__fastcall AnsiStringT SubString(int index, int count) const;
```

Description

System::AnsiStringT::SubString returns a new System::AnsiStringT (see page 3207) that is a substring of this System::AnsiStringT (see page 3207). The substring contains count characters beginning at index.

35.1.2.4.33 AnsiStringT::swap Method

Swap string pointers.

Pascal

```
function swap(var other: AnsiStringT): AnsiStringT&;
```

C++

```
AnsiStringT& swap(AnsiStringT& other);
```

Description

System::AnsiStringT::swap swaps two System::AnsiStringT (see page 3207) strings. Pointers to the data (see page 3218) are swapped, and the data (see page 3218) is not copied. This makes the swap operation efficient for sorting.

35.1.2.4.34 AnsiStringT::Trim Method

Returns a new System::AnsiStringT (see page 3207) without leading spaces, trailing spaces, or control characters.

Pascal

```
function Trim: AnsiStringT;
```

C++

```
__fastcall AnsiStringT Trim() const;
```

Description

System::AnsiStringT::Trim returns a new System::AnsiStringT (see page 3207) removing leading and trailing spaces and control characters. Use System::AnsiStringT::Trim to remove blank space before and after the first printing character.

35.1.2.4.35 AnsiStringT::TrimLeft Method

Returns a new System::AnsiStringT (see page 3207) without leading spaces or control characters.

Pascal

```
function TrimLeft: AnsiStringT;
```

C++

```
__fastcall AnsiStringT TrimLeft() const;
```

Description

System::AnsiStringT::TrimLeft returns a new System::AnsiStringT (see page 3207) removing leading spaces and control characters. Use System::AnsiStringT::TrimLeft to remove blank space before the first printing character.

35.1.2.4.36 AnsiStringT::TrimRight Method

Returns a new System::AnsiStringT (see page 3207) without trailing spaces or control characters.

Pascal

```
function TrimRight: AnsiStringT;
```

C++

```
__fastcall AnsiStringT TrimRight() const;
```

Description

System::AnsiStringT::TrimRight returns a new System::AnsiStringT (see page 3207) removing trailing spaces and control characters. Use the System::AnsiStringT::TrimRight function to remove blank space after the last printing character.

35.1.2.4.37 AnsiStringT::Unique Method

Makes the System::AnsiStringT (see page 3207) unique.

Pascal

```
function Unique: AnsiStringT&;
```

C++

```
__fastcall AnsiStringT& Unique();
```

Description

Use this function to make a string unique (refcnt == 1). System::AnsiStringT::Unique sets the reference count on the string for the System::AnsiStringT (see page 3207) object to 1 so that it is the only object referencing that string.

35.1.2.4.38 AnsiStringT::UpperCase Method

Returns an upper-case version of this System::AnsiStringT (see page 3207).

Pascal

```
function UpperCase: AnsiStringT;
```

C++

```
__fastcall AnsiStringT UpperCase() const;
```

Description

System::AnsiStringT::UpperCase returns a new System::AnsiStringT (see page 3207) that contains all characters in this string, converted to upper case. The conversion uses the current locale . The original System::AnsiStringT (see page 3207) remains unchanged.

35.1.2.4.39 AnsiStringT::vprintf Method

Sets the value of the System::AnsiStringT (see page 3207) given a format string and its arguments.

Pascal

```
function vprintf(const format: char*; list: va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(const char* format, va_list list);
```

Description

Use System::AnsiStringT::vprintf to set to the value of the System::AnsiStringT (see page 3207) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the length of the final formatted string.

See Also

System::AnsiStringT::Format (see page 3221)

System::AnsiStringT::sprintf (see page 3227)

System::AnsiStringT::printf (see page 3226)

System::AnsiStringT::cat_vprintf (see page 3217)

35.1.2.4.40 AnsiStringT::WideChar Method

Converts the System::AnsiStringT (see page 3207) to a wide-character array.

Pascal

```
function WideChar(var dest: wchar_t; destSize: Integer): wchar_t*;
```

C++

```
__fastcall wchar_t* WideChar(wchar_t* dest, int destSize) const;
```

Description

System::AnsiStringT::WideChar converts the System::AnsiStringT (see page 3207) to an array of wide characters, and returns the resulting wide-character array.

dest is a buffer the caller allocates to hold the return value. Use the System::AnsiStringT::WideCharBufSize (see page 3231) method to determine the size needed to hold the entire string value.

destSize is the size, in bytes, of dest.

System::AnsiStringT::WideChar returns a pointer to the wide-character array that is written to dest.

See Also

System::AnsiStringT::WideCharBufSize (see page 3231)

35.1.2.4.41 AnsiStringT::WideCharBufSize Method

Returns the size of the buffer required by the WideChar (see page 3230) function.

Pascal

```
function WideCharBufSize: Integer;
```

C++

```
__fastcall int WideCharBufSize() const;
```

Description

System::AnsiStringT::WideCharBufSize returns an integer value indicating the size of the buffer required by the WideChar (see page 3230) method in order to convert this string.

See Also

System::AnsiStringT::WideChar (see page 3230)

35.1.2.5 AnsiStringT Operators

The operators of the AnsiStringT class are listed here.

Public Operators

	Name	Description
	!= (see page 3232)	Compares two strings.
	+ (see page 3232)	Concatenates two System::AnsiStringT (see page 3207) values.
	+= (see page 3232)	Append (see page 3520) an System::AnsiStringT (see page 3207).
	< (see page 3233)	Compares two strings.
	<= (see page 3233)	Compares two strings.
	= (see page 3233)	Assigns the string rhs to this System::AnsiStringT (see page 3207).
	== (see page 3234)	Compares the System::AnsiStringT (see page 3207) with a specified string, returning true if they are the same.
	> (see page 3234)	Compares two strings.
	>= (see page 3234)	Compares two strings.

35.1.2.5.1 AnsiStringT::!= Operator

Compares two strings.

Pascal

```
operator !=(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator !=(const AnsiStringT& rhs) const;
```

Description

Returns true if the strings are not the same, false otherwise.

Note: This System::AnsiStringT (see page 3207)::!= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3214) or AnsiCompareIC (see page 3215) methods.

See Also

System::AnsiStringT::AnsiCompare (see page 3214)

System::AnsiStringT::AnsiCompareIC (see page 3215)

35.1.2.5.2 AnsiStringT::+ Operator

Concatenates two System::AnsiStringT (see page 3207) values.

Pascal

```
operator +(const rhs: AnsiStringT&);
```

C++

```
__fastcall AnsiStringT operator +(const AnsiStringT& rhs) const;
```

Description

The friend function returns an System::AnsiStringT (see page 3207) that is the concatenation of strings lhs and rhs.

The method returns an System::AnsiStringT (see page 3207) that is the concatenation of this System::AnsiStringT (see page 3207) and rhs.

35.1.2.5.3 AnsiStringT::+= Operator

Append (see page 3520) an System::AnsiStringT (see page 3207).

Pascal

```
operator +=(const rhs: AnsiStringT&);
```

C++

```
__fastcall AnsiStringT& operator +=(const AnsiStringT& rhs);
```

Description

The System::AnsiStringT (see page 3207)::+= concatenates this System::AnsiStringT (see page 3207) and rhs, assigning the result to this System::AnsiStringT (see page 3207).

The System::AnsiStringT (see page 3207)::+=() is not supported on a property with getter/setter functions.

35.1.2.5.4 AnsiStringT::< Operator

Compares two strings.

Pascal

```
operator <(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator <(const AnsiStringT& rhs) const;
```

Description

Returns true if this System::AnsiStringT::AnsiString is less than rhs, false otherwise.

Note: This System::AnsiStringT (see page 3207)::< performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3214) or AnsiCompareIC (see page 3215) methods.

See Also

System::AnsiStringT::AnsiCompare (see page 3214)

System::AnsiStringT::AnsiCompareIC (see page 3215)

35.1.2.5.5 AnsiStringT::<= Operator

Compares two strings.

Pascal

```
operator <=(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator <=(const AnsiStringT& rhs) const;
```

Description

Returns true if this System::AnsiStringT::AnsiString is less than or equal to rhs, false otherwise.

Note: This System::AnsiStringT (see page 3207)::<= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3214) or AnsiCompareIC (see page 3215) methods.

See Also

System::AnsiStringT::AnsiCompare (see page 3214)

System::AnsiStringT::AnsiCompareIC (see page 3215)

35.1.2.5.6 AnsiStringT::= Operator

Assigns the string rhs to this System::AnsiStringT (see page 3207).

Pascal

```
operator =(const rhs: AnsiStringT&);
```

C++

```
__fastcall AnsiStringT& operator =(const AnsiStringT& rhs);
```

Description

Use this System::AnsiStringT (see page 3207)::= to assign the value of the System::AnsiStringT (see page 3207) from another System::AnsiStringT (see page 3207).

35.1.2.5.7 AnsiStringT::== Operator

Compares the System::AnsiStringT (see page 3207) with a specified string, returning true if they are the same.

Pascal

```
operator ==(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator ==(const AnsiStringT& rhs) const;
```

Description

Returns true if this System::AnsiStringT (see page 3207) is the same as rhs, false otherwise.

Note: This System::AnsiStringT (see page 3207)::== performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3214) or AnsiCompareIC (see page 3215) methods.

See Also

System::AnsiStringT::AnsiCompare (see page 3214)

System::AnsiStringT::AnsiCompareIC (see page 3215)

35.1.2.5.8 AnsiStringT::> Operator

Compares two strings.

Pascal

```
operator >(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator >(const AnsiStringT& rhs) const;
```

Description

Returns true if this System::AnsiStringT (see page 3207) is greater than rhs, false otherwise.

Note: This System::AnsiStringT (see page 3207)::> performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3214) or AnsiCompareIC (see page 3215) methods.

See Also

System::AnsiStringT::AnsiCompare (see page 3214)

System::AnsiStringT::AnsiCompareIC (see page 3215)

35.1.2.5.9 AnsiStringT::>= Operator

Compares two strings.

Pascal

```
operator >=(const rhs: AnsiStringT&);
```

C++

```
__fastcall bool operator >=(const AnsiStringT& rhs) const;
```

Description

Returns true if this System::AnsiStringT (see page 3207) is greater than or equal to rhs, false otherwise.

Note: This System::AnsiStringT (see page 3207)::>= performs a simple byte-level numeric comparison. For locale support, or to ignore case, use the AnsiCompare (see page 3214) or AnsiCompareIC (see page 3215) methods.

See Also

System::AnsiStringT::AnsiCompare (see page 3214)

System::AnsiStringT::AnsiCompareIC (see page 3215)

35.1.3 Currency Class

System::Currency is the C++ implementation of the Delphi System::Currency data type.

Class Hierarchy

Pascal

```
Currency = class(CurrencyBase);
```

C++

```
class Currency : public CurrencyBase;
```

File

syscurr.h

Description

System::Currency is a C++ class that implements a Delphi intrinsic type, also called System::Currency. Use System::Currency to hold monetary values.

Note: A System::Currency object is implemented as a 64-bit signed integer. Avoid operations that might cause integer overflow.

Note: Use only the operators defined by System::Currency. The compiler ignores any operators you overload yourself. Use the System::Currency constructors to create System::Currency objects from pointers to other System::Currency objects, and also from numeric and string values..

Note: To utilize the C++ streaming operators (<< and >>) with System::Currency, you must use #include <iostream> before #include <syscurr.h>. This occurs automatically if you include vcl.h, clx.h, or System.hpp. The following stream operators are defined:

```
ostream& operator << (ostream& os, const Currency& arg);
istream& operator >> (istream& is, Currency& arg);
```

See Also

System::CurrencyBase (see page 3255)

35.1.3.1 Currency Members

The following tables list the members exposed by Currency.

Public Data Members

	Name	Description
	Val (see page 3256)	This is Val, a member of class CurrencyBase.

Public Friends

Name	Description
Currency __fastcall operator -(double lhs, const Currency& rhs) (see page 3253)	This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).
Currency __fastcall operator -(int lhs, const Currency& rhs) (see page 3253)	This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).
Currency __fastcall operator *(double lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).
Currency __fastcall operator *(int lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).
Currency __fastcall operator /(double lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).
Currency __fastcall operator /(int lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).
Currency __fastcall operator +(double lhs, const Currency& rhs) (see page 3255)	This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).
Currency __fastcall operator +(int lhs, const Currency& rhs) (see page 3255)	This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

Public Methods

	Name	Description
	Currency (see page 3237)	This is the overview for the Currency constructor overload.

Public Operators

	Name	Description
	- (see page 3238)	This is the overview for the - operator overload.
	-- (see page 3239)	This is the overview for the -- operator overload.
	! (see page 3240)	Returns a new System::Currency::Currency (see page 3235) object that is the logical negation of this System::Currency::Currency (see page 3235) object.
	!= (see page 3240)	This is the overview for the != operator overload.
	% (see page 3241)	Returns a new System::Currency::Currency (see page 3235) object with the value that is the remainder of this System::Currency::Currency (see page 3235) object divided by a specified integer.
	%= (see page 3242)	Assigns this System::Currency::Currency (see page 3235) object the remainder of its initial value divided by integer rhs.
	* (see page 3242)	This is the overview for the * operator overload.
	*= (see page 3243)	Assigns this System::Currency::Currency (see page 3235) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 3235) object rhs.
	/ (see page 3243)	This is the overview for the / operator overload.
	/= (see page 3244)	Assigns this System::Currency::Currency (see page 3235) object the result of its initial value divided by the value of System::Currency::Currency (see page 3235) object rhs.
	+ (see page 3244)	This is the overview for the + operator overload.
	++ (see page 3245)	This is the overview for the ++ operator overload.
	+= (see page 3246)	Increments the value of this System::Currency::Currency (see page 3235) object by the value of rhs.
	< (see page 3247)	This is the overview for the < operator overload.
	<= (see page 3247)	This is the overview for the <= operator overload.
	-= (see page 3249)	Decrements the value of this System::Currency::Currency (see page 3235) object by the value of rhs.
	= (see page 3248)	This is the overview for the = operator overload.
	== (see page 3249)	This is the overview for the == operator overload.
	> (see page 3250)	This is the overview for the > operator overload.
	>= (see page 3251)	This is the overview for the >= operator overload.
	double (see page 3252)	Converts the System::Currency::Currency (see page 3235) object to a double value.
	int (see page 3252)	Converts the System::Currency::Currency (see page 3235) object to an integer value

	System::String (see page 3252)	Converts the System::Currency::System::String object to a String that displays its value.

35.1.3.2 Currency Constructor

This is the overview for the Currency constructor overload.

Overload List

	Name	Description
	Currency::Currency () (see page 3237)	Creates and initializes a System::Currency::Currency (see page 3235) object.
	Currency::Currency (Currency&) (see page 3237)	Creates and initializes a System::Currency::Currency (see page 3235) object.
	Currency::Currency (CurrencyBase&) (see page 3237)	Creates and initializes a System::Currency::Currency (see page 3235) object.
	Currency::Currency (double) (see page 3237)	Creates and initializes a System::Currency::Currency (see page 3235) object.
	Currency::Currency (int) (see page 3237)	Creates and initializes a System::Currency::Currency (see page 3235) object.
	Currency::Currency (System::String&) (see page 3237)	Creates and initializes a System::Currency::Currency (see page 3235) object.

35.1.3.2.1 Currency::Currency Constructor ()

Creates and initializes a System::Currency::Currency (see page 3235) object.

Pascal

```
constructor Create;
constructor Create(val: double);
constructor Create(val: Integer);
constructor Create(const src: CurrencyBase&);
constructor Create(const src: Currency&);
constructor Create(const src: System::String&);
```

C++

```
__fastcall Currency();
__fastcall Currency(double val);
__fastcall Currency(int val);
__fastcall Currency(const CurrencyBase& src);
__fastcall Currency(const Currency& src);
__fastcall Currency(const System::String& src);
```

Description

Use System::Currency::Currency (see page 3235) to instantiate a System::Currency::Currency (see page 3235) object.

When called with no parameters, the resulting object has a value of 0.

When called with a System::Currency::Currency (see page 3235) object as the src parameter, the resulting object is a copy of src (the copy constructor).

When called with a System::Currency::CurrencyBase (see page 3255) object as the src parameter, the resulting object has the same value as the src parameter. System::Currency::CurrencyBase (see page 3255) is the base class for the System::Currency::Currency (see page 3235) type. It has a single member, a public Val (see page 3701) field that

represents the value of the object.

When called with an AnsiString object as the src parameter, the value of the resulting object is the parsed value of the AnsiString. The AnsiString must represent a numeric value within the range valid for System::Currency::Currency (see page 3235) (-922337203685477.5808 to 922337203685477.5807).

When called with a double (see page 3252) or integer as the val parameter, the value of the resulting object is the value of the val parameter.

35.1.3.3 Currency Operators

The operators of the Currency class are listed here.

Public Operators

	Name	Description
	- (see page 3238)	This is the overview for the - operator overload.
	-- (see page 3239)	This is the overview for the -- operator overload.
	! (see page 3240)	Returns a new System::Currency::Currency (see page 3235) object that is the logical negation of this System::Currency::Currency (see page 3235) object.
	!= (see page 3240)	This is the overview for the != operator overload.
	% (see page 3241)	Returns a new System::Currency::Currency (see page 3235) object with the value that is the remainder of this System::Currency::Currency (see page 3235) object divided by a specified integer.
	%= (see page 3242)	Assigns this System::Currency::Currency (see page 3235) object the remainder of its initial value divided by integer rhs.
	* (see page 3242)	This is the overview for the * operator overload.
	*= (see page 3243)	Assigns this System::Currency::Currency (see page 3235) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 3235) object rhs.
	/ (see page 3243)	This is the overview for the / operator overload.
	/= (see page 3244)	Assigns this System::Currency::Currency (see page 3235) object the result of its initial value divided by the value of System::Currency::Currency (see page 3235) object rhs.
	+ (see page 3244)	This is the overview for the + operator overload.
	++ (see page 3245)	This is the overview for the ++ operator overload.
	+= (see page 3246)	Increments the value of this System::Currency::Currency (see page 3235) object by the value of rhs.
	< (see page 3247)	This is the overview for the < operator overload.
	<= (see page 3247)	This is the overview for the <= operator overload.
	= (see page 3248)	This is the overview for the = operator overload.
	-= (see page 3249)	Decrements the value of this System::Currency::Currency (see page 3235) object by the value of rhs.
	== (see page 3249)	This is the overview for the == operator overload.
	> (see page 3250)	This is the overview for the > operator overload.
	>= (see page 3251)	This is the overview for the >= operator overload.
	double (see page 3252)	Converts the System::Currency::Currency (see page 3235) object to a double value.
	int (see page 3252)	Converts the System::Currency::Currency (see page 3235) object to an integer value
	System::String (see page 3252)	Converts the System::Currency::System::String object to a String that displays its value.

35.1.3.3.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	Currency::- () (see page 3239)	Returns a new System::Currency (see page 3235) object that is the negative value of this System::Currency (see page 3235) object.
	Currency::- (Currency&) (see page 3239)	Returns a new System::Currency (see page 3235) object that is the negative value of this System::Currency (see page 3235) object.
	Currency::- (double) (see page 3239)	Returns a new System::Currency (see page 3235) object that is the negative value of this System::Currency (see page 3235) object.
	Currency::- (int) (see page 3239)	Returns a new System::Currency (see page 3235) object that is the negative value of this System::Currency (see page 3235) object.

35.1.3.3.1.1 Currency::- Operator ()

Returns a new System::Currency (see page 3235) object that is the negative value of this System::Currency (see page 3235) object.

Pascal

```
operator -(const rhs: Currency&);
operator -(rhs: Integer);
operator -(rhs: double);
operator -;
```

C++

```
__fastcall Currency operator -(const Currency& rhs) const;
__fastcall Currency operator -(int rhs) const;
__fastcall Currency operator -(double rhs) const;
__fastcall Currency operator -() const;
```

Description

Use the unary – operator to change the sign of the System::Currency (see page 3235) object (effectively multiplying it by –1).

See Also

System::Currency::! (see page 3240)

35.1.3.3.2 -- Operator

This is the overview for the -- operator overload.

Overload List

	Name	Description
	Currency::-- () (see page 3240)	Decrements the value of the System::Currency::Currency (see page 3235) object by a specified integer value.
	Currency::-- (int) (see page 3240)	Decrements the value of the System::Currency::Currency (see page 3235) object by a specified integer value.

35.1.3.3.2.1 Currency::-- Operator ()

Decrements the value of the System::Currency::Currency (see page 3235) object by a specified integer value.

Pascal

```
operator --;
operator --(: Integer);
```

C++

```
Currency& operator --();
Currency operator --(int);
```

Description

Use the unary System::Currency (see page 3235)::-- to decrement the value of the System::Currency::Currency (see page 3235) object by 1.

Use the binary System::Currency (see page 3235)::-- to decrement the value by the integer specified as an operand.

See Also

System::Currency::++ (see page 3246)

System::Currency::-= (see page 3249)

System::Currency::- (see page 3239)

35.1.3.3.3 Currency::! Operator

Returns a new System::Currency::Currency (see page 3235) object that is the logical negation of this System::Currency::Currency (see page 3235) object.

Pascal

```
operator !;
```

C++

```
__fastcall Currency operator !() const;
```

Description

If the value of the System::Currency::Currency (see page 3235) object is nonzero, the System::Currency (see page 3235)::! returns 0. If the value of the System::Currency::Currency (see page 3235) object is 0, the System::Currency (see page 3235)::! returns 1.

See Also

System::Currency::!= (see page 3241)

System::Currency::- (see page 3239)

35.1.3.3.4 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	Currency::!= (Currency&) (see page 3241)	Tests for inequality between the System::Currency::Currency (see page 3235) object's value and a specified value.

	Currency::!= (double) (see page 3241)	Tests for inequality between the System::Currency::Currency (see page 3235) object's value and a specified value.
	Currency::!= (int) (see page 3241)	Tests for inequality between the System::Currency::Currency (see page 3235) object's value and a specified value.

35.1.3.3.4.1 Currency::!= Operator (Currency&)

Tests for inequality between the System::Currency::Currency (see page 3235) object's value and a specified value.

Pascal

```
operator !=(const rhs: Currency&);
operator !=(rhs: Integer);
operator !=(rhs: double);
```

C++

```
__fastcall bool operator !=(const Currency& rhs) const;
__fastcall bool operator !=(int rhs) const;
__fastcall bool operator !=(double rhs) const;
```

Description

The System::Currency (see page 3235)::!= compares the value of the System::Currency::Currency (see page 3235) object with the value specified by the rhs operand. If the values differ, the System::Currency (see page 3235)::!= returns true. If the values are the same, the System::Currency (see page 3235)::!= returns false.

See Also

System::Currency::= (see page 3248)

System::Currency::> (see page 3250)

System::Currency::>= (see page 3251)

System::Currency::< (see page 3247)

System::Currency::<= (see page 3248)

35.1.3.3.5 Currency::% Operator

Returns a new System::Currency::Currency (see page 3235) object with the value that is the remainder of this System::Currency::Currency (see page 3235) object divided by a specified integer.

Pascal

```
operator %(rhs: Integer);
```

C++

```
__fastcall Currency operator %(int rhs) const;
```

Description

Use the System::Currency (see page 3235)::% to obtain the remainder from dividing the value of this System::Currency::Currency (see page 3235) object by the integer specified by the rhs operand.

See Also

System::Currency::%= (see page 3242)

System::Currency::/ (see page 3243)

35.1.3.3.6 Currency::%= Operator

Assigns this System::Currency::Currency (see page 3235) object the remainder of its initial value divided by integer rhs.

Pascal

```
operator %=(rhs: Integer);
```

C++

```
__fastcall Currency& operator %=(int rhs);
```

Description

Use the System::Currency (see page 3235)::%= to obtain the remainder from dividing this System::Currency::Currency (see page 3235) object's value by the Value of the System::Currency::Currency (see page 3235) object specified by rhs.

See Also

System::Currency::= (see page 3248)

System::Currency::/= (see page 3244)

System::Currency::% (see page 3241)

35.1.3.3.7 * Operator

This is the overview for the * operator overload.

Overload List

	Name	Description
	Currency::* (Currency&) (see page 3242)	Multiplies the value of the System::Currency::Currency (see page 3235) object by a specified value.
	Currency::* (double) (see page 3242)	Multiplies the value of the System::Currency::Currency (see page 3235) object by a specified value.
	Currency::* (int) (see page 3242)	Multiplies the value of the System::Currency::Currency (see page 3235) object by a specified value.

35.1.3.3.7.1 Currency::* Operator (Currency&)

Multiplies the value of the System::Currency::Currency (see page 3235) object by a specified value.

Pascal

```
operator *(const rhs: Currency&);
operator *(rhs: Integer);
operator *(rhs: double);
```

C++

```
__fastcall Currency operator *(const Currency& rhs) const;
__fastcall Currency operator *(int rhs) const;
__fastcall Currency operator *(double rhs) const;
```

Description

Use the System::Currency (see page 3235)::* to multiply the value of the System::Currency::Currency (see page 3235) object by the value specified by the rhs operand.

Note that there are two related friend System::Currency (see page 3235)::*s:

```
friend Currency __fastcall operator *(int lhs, const Currency& rhs);
```

multiples the values of integer lhs and System::Currency::Currency (see page 3235) object rhs and returns a new System::Currency::Currency (see page 3235) object with the resulting value.

```
friend Currency __fastcall operator *(double lhs, const Currency& rhs);
```

multiples the values of double (see page 3252) lhs and System::Currency::Currency (see page 3235) object rhs and returns a new System::Currency::Currency (see page 3235) object with the resulting value.

See Also

System::Currency::*= (see page 3243)

System::Currency::/ (see page 3243)

35.1.3.3.8 Currency::*= Operator

Assigns this System::Currency::Currency (see page 3235) object the result of its initial value multiplied by the value of System::Currency::Currency (see page 3235) object rhs.

Pascal

```
operator *=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator *=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3235)::*= to multiply this System::Currency::Currency (see page 3235) object's value by the Value of the System::Currency::Currency (see page 3235) object specified by rhs.

See Also

System::Currency::= (see page 3248)

System::Currency::/= (see page 3244)

System::Currency::* (see page 3242)

35.1.3.3.9 / Operator

This is the overview for the / operator overload.

Overload List

	Name	Description
	Currency::/ (Currency&) (see page 3243)	Divides the value of the System::Currency::Currency (see page 3235) object by a specified value.
	Currency::/ (double) (see page 3243)	Divides the value of the System::Currency::Currency (see page 3235) object by a specified value.
	Currency::/ (int) (see page 3243)	Divides the value of the System::Currency::Currency (see page 3235) object by a specified value.

35.1.3.3.9.1 Currency::/ Operator (Currency&)

Divides the value of the System::Currency::Currency (see page 3235) object by a specified value.

Pascal

```
operator /(const rhs: Currency&);
operator /(rhs: Integer);
operator /(rhs: double);
```

C++

```
__fastcall Currency operator /(const Currency& rhs) const;
__fastcall Currency operator /(int rhs) const;
__fastcall Currency operator /(double rhs) const;
```

Description

Use the System::Currency (see page 3235)::/ to divide the value of the System::Currency::Currency (see page 3235) object by the value specified by the rhs operand.

Note that there are two related friend System::Currency (see page 3235)::/s:

```
friend Currency __fastcall operator /(int lhs, const Currency& rhs);
```

divides the value of int (see page 3252) lhs by the value of System::Currency::Currency (see page 3235) object rhs and returns a new System::Currency::Currency (see page 3235) object with the resulting value.

```
friend Currency __fastcalloperator /(double lhs, const Currency& rhs);
```

divides the value of double (see page 3252) lhs by the value of System::Currency::Currency (see page 3235) object rhs and returns a System::Currency::Currency (see page 3235) object with the resulting value.

See Also

System::Currency::/= (see page 3244)

System::Currency::* (see page 3242)

System::Currency::% (see page 3241)

35.1.3.3.10 Currency::/= Operator

Assigns this System::Currency::Currency (see page 3235) object the result of its initial value divided by the value of System::Currency::Currency (see page 3235) object rhs.

Pascal

```
operator /=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator /=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3235)::/= to divide this System::Currency::Currency (see page 3235) object's value by the Value of the System::Currency::Currency (see page 3235) object specified by rhs.

See Also

System::Currency::= (see page 3248)

System::Currency::*= (see page 3243)

System::Currency::/ (see page 3243)

35.1.3.3.11 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	Currency::+ (Currency&) (see page 3245)	Adds a specified value to the System::Currency::Currency (see page 3235) object.
	Currency::+ (double) (see page 3245)	Adds a specified value to the System::Currency::Currency (see page 3235) object.
	Currency::+ (int) (see page 3245)	Adds a specified value to the System::Currency::Currency (see page 3235) object.

35.1.3.3.11.1 Currency::+ Operator (Currency&)

Adds a specified value to the System::Currency::Currency (see page 3235) object.

Pascal

```
operator +(const rhs: Currency&);
operator +(rhs: Integer);
operator +(rhs: double);
```

C++

```
__fastcall Currency operator +(const Currency& rhs) const;
__fastcall Currency operator +(int rhs) const;
__fastcall Currency operator +(double rhs) const;
```

Description

Use the System::Currency (see page 3235)::+ to add the value specified by the rhs operand to the System::Currency::Currency (see page 3235) object. If the rhs operand is another System::Currency::Currency (see page 3235) object, the System::Currency (see page 3235)::+ adds its value to the System::Currency::Currency (see page 3235) object. If it is an integer or double (see page 3252) value, the System::Currency (see page 3235)::+ adds that value to the value of the System::Currency::Currency (see page 3235) object.

Note that there are two related friend System::Currency (see page 3235)::+s:

```
friend Currency __fastcall operator +(int lhs, const Currency& rhs);
```

returns a new System::Currency::Currency (see page 3235) object that is the sum of int (see page 3252) lhs and System::Currency::Currency (see page 3235) object rhs.

```
friend Currency __fastcall operator +(double lhs, const Currency& rhs);
```

returns a new System::Currency::Currency (see page 3235) object that is the sum of double (see page 3252) lhs and System::Currency::Currency (see page 3235) object rhs.

See Also

System::Currency::+= (see page 3246)

System::Currency::++ (see page 3246)

System::Currency::- (see page 3239)

35.1.3.3.12 ++ Operator

This is the overview for the ++ operator overload.

Overload List

	Name	Description
	Currency::++ () (see page 3246)	Increments the value of the System::Currency::Currency (see page 3235) object by a specified integer value.
	Currency::++ (int) (see page 3246)	Increments the value of the System::Currency::Currency (see page 3235) object by a specified integer value.

35.1.3.3.12.1 Currency::++ Operator ()

Increments the value of the System::Currency::Currency (see page 3235) object by a specified integer value.

Pascal

```
operator ++;
operator ++(: Integer);
```

C++

```
Currency& operator ++();
Currency operator ++(int);
```

Description

Use the unary System::Currency (see page 3235)::++ to increment the value of the System::Currency::Currency (see page 3235) object by 1.

Use the binary System::Currency (see page 3235)::++ to increment the value by the integer specified as an operand.

See Also

System::Currency::-- (see page 3240)

System::Currency::+= (see page 3246)

System::Currency::+ (see page 3245)

35.1.3.3.13 Currency::+= Operator

Increments the value of this System::Currency::Currency (see page 3235) object by the value of rhs.

Pascal

```
operator +=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator +=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3235)::+= to add the value of another System::Currency::Currency (see page 3235) object to this one.

See Also

System::Currency::= (see page 3248)

System::Currency::-= (see page 3249)

System::Currency::+ (see page 3245)

System::Currency::++ (see page 3246)

35.1.3.3.14 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	Currency::< (Currency&) (see page 3247)	Tests whether this System::Currency::Currency (see page 3235) value is less than a specified value.
	Currency::< (double) (see page 3247)	Tests whether this System::Currency::Currency (see page 3235) value is less than a specified value.
	Currency::< (int) (see page 3247)	Tests whether this System::Currency::Currency (see page 3235) value is less than a specified value.

35.1.3.3.14.1 Currency::< Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3235) value is less than a specified value.

Pascal

```
operator <(const rhs: Currency&);
operator <(rhs: Integer);
operator <(rhs: double);
```

C++

```
__fastcall bool operator <(const Currency& rhs) const;
__fastcall bool operator <(int rhs) const;
__fastcall bool operator <(double rhs) const;
```

Description

The < System::Currency (see page 3235)::< compares the value of the System::Currency::Currency (see page 3235) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3235) object has the lesser value, false otherwise.

See Also

System::Currency::!= (see page 3241)

System::Currency::> (see page 3250)

System::Currency::>= (see page 3251)

System::Currency::== (see page 3250)

System::Currency::<= (see page 3248)

35.1.3.3.15 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	Currency::<= (Currency&) (see page 3248)	Tests whether this System::Currency::Currency (see page 3235) value is less than or equal to a specified value.

	Currency::<= (double) (see page 3248)	Tests whether this System::Currency::Currency (see page 3235) value is less than or equal to a specified value.
	Currency::<= (int) (see page 3248)	Tests whether this System::Currency::Currency (see page 3235) value is less than or equal to a specified value.

35.1.3.3.15.1 Currency::<= Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3235) value is less than or equal to a specified value.

Pascal

```
operator <=(const rhs: Currency&);
operator <=(rhs: Integer);
operator <=(rhs: double);
```

C++

```
__fastcall bool operator <=(const Currency& rhs) const;
__fastcall bool operator <=(int rhs) const;
__fastcall bool operator <=(double rhs) const;
```

Description

The <= System::Currency (see page 3235)::<= compares the value of the System::Currency::Currency (see page 3235) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3235) object has a value less than or equal to that of rhs, false otherwise.

See Also

System::Currency::!= (see page 3241)

System::Currency::> (see page 3250)

System::Currency::>= (see page 3251)

System::Currency::< (see page 3247)

System::Currency::== (see page 3250)

35.1.3.3.16 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	Currency::= (Currency&) (see page 3248)	Assigns the value of the System::Currency::Currency (see page 3235) object.
	Currency::= (CurrencyBase&) (see page 3248)	Assigns the value of the System::Currency::Currency (see page 3235) object.
	Currency::= (double) (see page 3248)	Assigns the value of the System::Currency::Currency (see page 3235) object.
	Currency::= (int) (see page 3248)	Assigns the value of the System::Currency::Currency (see page 3235) object.

35.1.3.3.16.1 Currency::= Operator (Currency&)

Assigns the value of the System::Currency::Currency (see page 3235) object.

Pascal

```
operator =(rhs: double);
operator =(rhs: Integer);
operator =(const rhs: CurrencyBase&);
operator =(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator =(double rhs);
__fastcall Currency& operator =(int rhs);
__fastcall Currency& operator =(const CurrencyBase& rhs);
__fastcall Currency& operator =(const Currency& rhs);
```

Description

The assignment System::Currency (see page 3235)::= assigns the value of the rhs operand to the System::Currency::Currency (see page 3235) object. The assignment System::Currency (see page 3235)::= can take

a double (see page 3252) or integer value, which is assigned to the System::Currency::Currency (see page 3235) object.

a System::Currency::CurrencyBase (see page 3255) or System::Currency::Currency (see page 3235) object, whose value is assigned to the System::Currency::Currency (see page 3235) object.

35.1.3.3.17 Currency::-= Operator

Decrements the value of this System::Currency::Currency (see page 3235) object by the value of rhs.

Pascal

```
operator -=(const rhs: Currency&);
```

C++

```
__fastcall Currency& operator -=(const Currency& rhs);
```

Description

Use the System::Currency (see page 3235)::-= to subtract the value of another System::Currency::Currency (see page 3235) object from this one.

See Also

System::Currency::= (see page 3248)

System::Currency::+= (see page 3246)

System::Currency::- (see page 3239)

System::Currency::-- (see page 3240)

35.1.3.3.18 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	Currency::== (Currency&) (see page 3250)	Tests for equality between the System::Currency::Currency (see page 3235) object's value and a specified value.
	Currency::== (double) (see page 3250)	Tests for equality between the System::Currency::Currency (see page 3235) object's value and a specified value.

	Currency::== (int) (see page 3250)	Tests for equality between the System::Currency::Currency (see page 3235) object's value and a specified value.

35.1.3.3.18.1 Currency::== Operator (Currency&)

Tests for equality between the System::Currency::Currency (see page 3235) object's value and a specified value.

Pascal

```
operator ==(const rhs: Currency&);
operator ==(rhs: Integer);
operator ==(rhs: double);
```

C++

```
__fastcall bool operator ==(const Currency& rhs) const;
__fastcall bool operator ==(int rhs) const;
__fastcall bool operator ==(double rhs) const;
```

Description

The == System::Currency (see page 3235)::== compares the value of the System::Currency::Currency (see page 3235) object with the value specified by the rhs operand. If the two values are the same, the System::Currency (see page 3235)::== returns true. If the values differ, the System::Currency (see page 3235)::== returns false.

See Also

System::Currency::!= (see page 3241)

System::Currency::> (see page 3250)

System::Currency::>= (see page 3251)

System::Currency::< (see page 3247)

System::Currency::<= (see page 3248)

35.1.3.3.19 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	Currency::> (Currency&) (see page 3250)	Tests whether this System::Currency::Currency (see page 3235) value is greater than a specified value.
	Currency::> (double) (see page 3250)	Tests whether this System::Currency::Currency (see page 3235) value is greater than a specified value.
	Currency::> (int) (see page 3250)	Tests whether this System::Currency::Currency (see page 3235) value is greater than a specified value.

35.1.3.3.19.1 Currency::> Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3235) value is greater than a specified value.

Pascal

```
operator >(const rhs: Currency&);
operator >(rhs: Integer);
operator >(rhs: double);
```

C++

```
__fastcall bool operator >(const Currency& rhs) const;
__fastcall bool operator >(int rhs) const;
__fastcall bool operator >(double rhs) const;
```

Description

The System::Currency (see page 3235)::> compares the value of the System::Currency::Currency (see page 3235) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3235) object has a greater value than rhs, false otherwise.

See Also

System::Currency::!= (see page 3241)

System::Currency::== (see page 3250)

System::Currency::>= (see page 3251)

System::Currency::< (see page 3247)

System::Currency::<= (see page 3248)

35.1.3.3.20 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	Currency::>= (Currency&) (see page 3251)	Tests whether this System::Currency::Currency (see page 3235) value is greater than or equal to a specified value.
	Currency::>= (double) (see page 3251)	Tests whether this System::Currency::Currency (see page 3235) value is greater than or equal to a specified value.
	Currency::>= (int) (see page 3251)	Tests whether this System::Currency::Currency (see page 3235) value is greater than or equal to a specified value.

35.1.3.3.20.1 Currency::>= Operator (Currency&)

Tests whether this System::Currency::Currency (see page 3235) value is greater than or equal to a specified value.

Pascal

```
operator >=(const rhs: Currency&);
operator >=(rhs: Integer);
operator >=(rhs: double);
```

C++

```
__fastcall bool operator >=(const Currency& rhs) const;
__fastcall bool operator >=(int rhs) const;
__fastcall bool operator >=(double rhs) const;
```

Description

The >= System::Currency (see page 3235)::>= compares the value of the System::Currency::Currency (see page 3235) object with the value specified by the rhs operand. It returns true if this System::Currency::Currency (see page 3235) object's value is greater than or equal to the value of rhs, false otherwise.

See Also

System::Currency::!= (see page 3241)

System::Currency::> (see page 3250)

System::Currency::== (see page 3250)

System::Currency::< (see page 3247)

System::Currency::<= (see page 3248)

35.1.3.3.21 Currency::double Operator

Converts the System::Currency::Currency (see page 3235) object to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

double is a conversion System::Currency::double that converts a System::Currency::Currency (see page 3235) object to a double.

See Also

System::Currency::int (see page 3252)

System::Currency::System::String (see page 3252)

35.1.3.3.22 Currency::int Operator

Converts the System::Currency::Currency (see page 3235) object to an integer value

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

int is a conversion System::Currency::int that converts a System::Currency::Currency (see page 3235) object to an int.

See Also

System::Currency::double (see page 3252)

System::Currency::System::String (see page 3252)

35.1.3.3.23 Currency::System::String Operator

Converts the System::Currency::System::String object to a String that displays its value.

Pascal

```
operator System::String;
```

C++

```
__fastcall operator System::String() const;
```

Description

String is a conversion System::Currency::System::String that converts the currency value into its string representation. The conversion uses the ffGeneral floating point conversion rules. For further details, see the description of the CurrToStrF and FloatToStrF methods of the AnsiString class.

See Also

System::Currency::int (see page 3252)

System::Currency::double (see page 3252)

35.1.3.4 Currency Friends

The friends of the Currency class are listed here.

Public Friends

Name	Description
Currency __fastcall operator -(double lhs, const Currency& rhs) (see page 3253)	This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).
Currency __fastcall operator -(int lhs, const Currency& rhs) (see page 3253)	This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).
Currency __fastcall operator *(double lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).
Currency __fastcall operator *(int lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).
Currency __fastcall operator /(double lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).
Currency __fastcall operator /(int lhs, const Currency& rhs) (see page 3254)	This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).
Currency __fastcall operator +(double lhs, const Currency& rhs) (see page 3255)	This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).
Currency __fastcall operator +(int lhs, const Currency& rhs) (see page 3255)	This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

35.1.3.4.1 friend Currency __fastcall operator -(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator -(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator -(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator -(double lhs, const Currency& rhs).

35.1.3.4.2 friend Currency __fastcall operator -(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator -(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator -(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator -(int lhs, const Currency& rhs).

35.1.3.4.3 friend Currency __fastcall operator *(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator *(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator *(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator *(double lhs, const Currency& rhs).

35.1.3.4.4 friend Currency __fastcall operator *(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator *(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator *(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator *(int lhs, const Currency& rhs).

35.1.3.4.5 friend Currency __fastcall operator /(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator /(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator /(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator /(double lhs, const Currency& rhs).

35.1.3.4.6 friend Currency __fastcall operator /(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator /(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator /(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator /(int lhs, const Currency& rhs).

35.1.3.4.7 friend Currency __fastcall operator +(double lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator +(double lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator +(double lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator +(double lhs, const Currency& rhs).

35.1.3.4.8 friend Currency __fastcall operator +(int lhs, const Currency& rhs) Friend

Pascal

```
Currency __fastcall operator +(int lhs, const Currency& rhs)
```

C++

```
friend Currency __fastcall operator +(int lhs, const Currency& rhs);
```

Description

This is friend Currency __fastcall operator +(int lhs, const Currency& rhs).

35.1.4 CurrencyBase Class

Class Hierarchy

Pascal

```
CurrencyBase = class;
```

C++

```
class CurrencyBase;
```

File

syscurr.h

Description

This is class System::CurrencyBase.

35.1.4.1 CurrencyBase Members

The following tables list the members exposed by CurrencyBase.

Public Data Members

	Name	Description
	Val (see page 3256)	This is Val, a member of class CurrencyBase.

35.1.4.2 CurrencyBase Data Members

The data members of the CurrencyBase class are listed here.

Public Data Members

	Name	Description
	Val (see page 3256)	This is Val, a member of class CurrencyBase.

35.1.4.2.1 CurrencyBase::Val Data Member

Pascal

```
Val: __int64;
```

C++

```
__int64 Val;
```

Description

This is Val, a member of class CurrencyBase.

35.1.5 DelphiInterface Template

System::DelphiInterface is a C++ implementation of the Delphi interface.

Class Hierarchy

Pascal

```
DelphiInterface<class T> = class;
```

C++

```
template <class T>
class DelphiInterface;
```

File

systobj.h

Description

A Delphi interface is an abstract collection of properties, methods, and events with no implementation. It resembles a C++ class with only pure virtual methods. However a Delphi interface type supports built-in reference counting.

System::DelphiInterface is a C++ template that defines types compatible with Delphi interfaces, System::DelphiInterface defines reference counting logic in its initialization constructor, copy constructor, assignment operator, and destructor.

Note: Other operators that provide access to the underlying interface pointer do not handle reference counting. You may need to explicitly call AddRef or Release (see page 3258) to ensure proper reference counting.

System::DelphiInterface is a template:

```
template <class T> class RTL_DELPHIRETURN DelphiInterface;
```

where the T parameter is an underlying interface class or COM interface. In the object framework, the resulting System::DelphiInterface class is usually given a name such as _di_IUnknown, where IUnknown (see page 3756) is the name of the corresponding interface. Such interfaces are documented under the interface name (such as IFormDesigner).

See Also

System::Variant (see page 3421)

35.1.5.1 DelphiInterface Members

The following tables list the members exposed by DelphiInterface.

Public Methods

	Name	Description
	~DelphiInterface (see page 3257)	Frees an instance of System::DelphiInterface::DelphiInterface (see page 3256).
	DelphiInterface (see page 3257)	This is the overview for the DelphiInterface constructor overload.
	Release (see page 3258)	This is Release, a member of class DelphiInterface.

Public Operators

	Name	Description
	! (see page 3259)	Indicates whether the System::DelphiInterface::DelphiInterface (see page 3256) object represents a NULL interface.
	& (see page 3259)	Returns a pointer to the underlying interface pointer.
	* (see page 3259)	Returns the underlying interface pointer.
	= (see page 3260)	This is the overview for the = operator overload.
	-> (see page 3260)	Provides access to the properties and methods on the underlying interface.
	T* (see page 3261)	This is T*, a member of class DelphiInterface.

35.1.5.2 DelphiInterface::~DelphiInterface Destructor

Frees an instance of System::DelphiInterface::DelphiInterface (see page 3256).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~DelphiInterface<T>();
```

Description

Before freeing the System::DelphiInterface::DelphiInterface (see page 3256) instance, System::DelphiInterface (see page 3256)::~DelphiInterface (see page 3256) calls the Release (see page 3258) method of the underlying interface.

35.1.5.3 DelphiInterface Constructor

This is the overview for the DelphiInterface constructor overload.

Overload List

	Name	Description
	DelphiInterface::DelphiInterface () (see page 3258)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 3256).

	DelphiInterface::DelphiInterface (DelphiInterface<T> &) (see page 3258)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 3256).
	DelphiInterface::DelphiInterface (T*) (see page 3258)	Creates an instance of System::DelphiInterface::DelphiInterface (see page 3256).

35.1.5.3.1 DelphiInterface::DelphiInterface Constructor ()

Creates an instance of System::DelphiInterface::DelphiInterface (see page 3256).

Pascal

```
constructor Create;
constructor Create(const rhs: DelphiInterface<T> &);
constructor Create(var rhs: T);
```

C++

```
__fastcall DelphiInterface<T>();
__fastcall DelphiInterface<T>(const DelphiInterface<T> & rhs);
__fastcall DelphiInterface<T>(T* rhs);
```

Description

The System::DelphiInterface::DelphiInterface (see page 3256) constructor is overloaded to create a System::DelphiInterface::DelphiInterface (see page 3256) in one of three ways:

When used with no parameters, System::DelphiInterface::DelphiInterface (see page 3256) creates a NULL interface.

When used with another System::DelphiInterface::DelphiInterface (see page 3256) object (the copy constructor), System::DelphiInterface::DelphiInterface (see page 3256) creates a new copy of the interface specified by rhs, incrementing the reference count of the underlying interface.

When used with an interface pointer of the underlying type, System::DelphiInterface::DelphiInterface (see page 3256) wraps the underlying interface specified by rhs and increments its reference count.

35.1.5.4 DelphiInterface Methods

The methods of the DelphiInterface class are listed here.

Public Methods

	Name	Description
	Release (see page 3258)	This is Release, a member of class DelphiInterface.

35.1.5.4.1 DelphiInterface::Release Method

Pascal

```
function Release: Integer;
```

C++

```
int Release();
```

Description

This is Release, a member of class DelphiInterface.

35.1.5.5 DelphiInterface Operators

The operators of the DelphiInterface class are listed here.

Public Operators

	Name	Description
	! (see page 3259)	Indicates whether the System::DelphiInterface::DelphiInterface (see page 3256) object represents a NULL interface.
	& (see page 3259)	Returns a pointer to the underlying interface pointer.
	* (see page 3259)	Returns the underlying interface pointer.
	= (see page 3260)	This is the overview for the = operator overload.
	-> (see page 3260)	Provides access to the properties and methods on the underlying interface.
	T* (see page 3261)	This is T*, a member of class DelphiInterface.

35.1.5.5.1 DelphiInterface::! Operator

Indicates whether the System::DelphiInterface::DelphiInterface (see page 3256) object represents a NULL interface.

Pascal

```
operator !;
```

C++

```
bool operator !() const;
```

Description

The ! System::DelphiInterface (see page 3256)::! returns true if the System::DelphiInterface::DelphiInterface (see page 3256) object represents a NULL interface, and false otherwise.

35.1.5.5.2 DelphiInterface::& Operator

Returns a pointer to the underlying interface pointer.

Pascal

```
operator &;
```

C++

```
__fastcall T** operator &();
```

Description

Use the & System::DelphiInterface (see page 3256)::& to access the underlying interface pointer. Care should be used with this System::DelphiInterface (see page 3256)::&, because it provides no reference counting support. For example, when using this System::DelphiInterface (see page 3256)::& to programmatically replace the underlying interface, you must first call Release (see page 3258) to decrement the reference count of the current interface and, depending on whether you are using this as an additional interface reference, you may need to increment the reference count on the interface being added.

35.1.5.5.3 DelphiInterface::* Operator

Returns the underlying interface pointer.

Pascal

```
operator *;
```

C++

```
__fastcall T& operator *();
```

Description

Use the & System::DelphiInterface (see page 3256)::* to access the underlying interface pointer. Care should be used with this System::DelphiInterface (see page 3256)::*, because it provides no reference counting support. For example, you may need to increment the reference count on the returned value.

35.1.5.5.4 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	DelphiInterface::= (DelphiInterface<T>&) (see page 3260)	Assigns a value to the System::DelphiInterface::DelphiInterface (see page 3256).
	DelphiInterface::= (T *) (see page 3260)	Assigns a value to the System::DelphiInterface::DelphiInterface (see page 3256).

35.1.5.5.4.1 DelphiInterface::= Operator (DelphiInterface<T>&)

Assigns a value to the System::DelphiInterface::DelphiInterface (see page 3256).

Pascal

```
operator =(var rhs: DelphiInterface<T>);
operator =(var rhs: T);
```

C++

```
__fastcall DelphiInterface<T>& operator =(DelphiInterface<T>& rhs);
__fastcall DelphiInterface<T>& operator =(T * rhs);
```

Description

The assignment System::DelphiInterface (see page 3256)::= assigns a value to the underlying interface from an interface pointer of the underlying type or from another System::DelphiInterface::DelphiInterface (see page 3256) object that wraps the same interface type.

This System::DelphiInterface (see page 3256)::= manages all reference counting issues. That is, it decrements the reference count on the current interface (if any) before making the assignment and increments the reference count on the newly assigned interface (if any).

35.1.5.5.5 DelphiInterface::-> Operator

Provides access to the properties and methods on the underlying interface.

Pascal

```
operator ->;
```

C++

```
__fastcall T* operator ->() const;
```

Description

Use the System::DelphiInterface (see page 3256)::-> to access the properties and methods of the underlying interface. If the

underlying interface is a dual interface, the System::DelphiInterface (see page 3256)::-> makes a vtable call.

35.1.5.5.6 DelphiInterface::T* Operator

Pascal

```
operator T*;
```

C++

```
__fastcall operator T*() const;
```

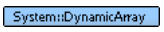
Description

This is T*, a member of class DelphiInterface.

35.1.6 DynamicArray Template

System::DynamicArray is the C++ analog for the Delphi dynamic array type.

Class Hierarchy



Pascal

```
DynamicArray<class T> = class;
```

C++

```
template <class T>
class DynamicArray;
```

File

sysdyn.h

Description

A Dynamic Array is a resizable collection implemented as an array. The System::DynamicArray template is a C++ implementation of Delphi Dynamic Array types. The System::DynamicArray template uses a binary layout compatible with the Delphi Dynamic Array implementation.

```
template <class T> class DELPHIRETURN DynamicArray;
```

You must specify one template parameter:

Parameter	Usage
type	the type of the elements

Length (see page 3270) of dynamic arrays

The size of a System::DynamicArray can be set or obtained from the Length (see page 3270) property:

//C++ example

```
DynamicArray<int> arrayOfInt;
arrayOfInt.Length = 10;
cout << "ArrayLength: " << arrayOfInt.Length << endl;
```

To free a System::DynamicArray, assign it a Length (see page 3270) of zero.

Accessing data

Access elements of a Dynamic Array via the subscript [] operator:

//C++ example

```
void InitArray(DynamicArray<char> &c_array)
{
  c_array[0] = 'A';
  c_array[1] = 'B';
  cout << "Third char is: " << c_array[2];
}
```

Bounds of dynamic arrays

The Low (see page 3271) and High (see page 3270) properties of System::DynamicArrays return the low and high bounds of the array respectively.

//C++ example

```
int TotalArray(const DynamicArray<int>& arrayOfInt)
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

Note: The Low (see page 3271) property always returns 0. The High (see page 3270) property returns Length (see page 3270)-1. So you may also traverse a dynamic array via the following construct:

for (int i=0; i<arrayOfInt. Length (see page 3270); i++)

Assigning, comparing and copying dynamic arrays

Dynamic Arrays are reference counted. When a System::DynamicArray is assigned to another, only the reference is assigned (and the reference count adjusted), the content of the source is not copied. Similarly, when two Dynamic Arrays are compared, only the references are compared, not the contents. To copy the contents of a System::DynamicArray, you must use the Copy (see page 3265) (or CopyRange (see page 3266)) methods.

//C++ example

```
void foo(DynamicArray<int> &i_array)
{
  DynamicArray<int> temp = i_array;
  assert(temp == i_array); // temp and i_array point to same chunk
                           // of data..
  i_array[0] = 20;
  assert(temp[0] == 20); // Temp 'sees' changes to i_array..
  temp = i_array.Copy(); // Temp gets its own copy of data
  temp[0] = 10;
  assert(temp[0] != i_array[0]); // Above assignment did not
                                 // modify i_array.
}
```

Multidimensional dynamic arrays

Dynamic arrays can be multidimensional. Multidimensional dynamic arrays don't have to be rectangular. The Length (see page 3270) property can be used to set each dimension. The following example illustrates.

//C++ example

```
typedef DynamicArray< DynamicArray < AnsiString > > T2DStringArray;
void foo(T2DStringArray &s_array)
{
  SetLength(s_array, 10);
for (int i=0; i<s_array.Length; i++)
  { // Set lengths of second dimensions.(NOTE: non-rectangular)
    SetLength(s_array[i], i+1);
```

```
for (int j=0; j<s_array[i].Length; j++)
    /* */ s_array[i][j] = itoa(i*10+j);
  }
}
```

C++ & Delphi System::DynamicArrays

A dynamic array declared in C++ code can be used in Delphi code and vice versa. The following examples illustrate passing a one-dimensional and two-dimensional array between C++ and Delphi.

C++ System::DynamicArray Example

Delphi System::DynamicArray Example

35.1.6.1 DynamicArray Members

The following tables list the members exposed by DynamicArray.

Protected Methods

	Name	Description
	AllocData (see page 3265)	This is AllocData, a member of class DynamicArray.
	DecRefCount (see page 3266)	This is DecRefCount, a member of class DynamicArray.
	FreeData (see page 3267)	This is FreeData, a member of class DynamicArray.
	IncRefCount (see page 3268)	This is IncRefCount, a member of class DynamicArray.
	SetData (see page 3268)	This is SetData, a member of class DynamicArray.

Public Methods

	Name	Description
	~DynamicArray (see page 3263)	Frees a System::DynamicArray (see page 3261) object.
	Copy (see page 3265)	This is the overview for the Copy method overload.
	CopyRange (see page 3266)	This is the overview for the CopyRange method overload.
	DimCount (see page 3267)	This is DimCount, a member of class DynamicArray.
	DynamicArray (see page 3264)	This is the overview for the DynamicArray constructor overload.
	get_high (see page 3267)	This is get_high, a member of class DynamicArray.
	get_length (see page 3267)	This is get_length, a member of class DynamicArray.
	get_low (see page 3267)	This is get_low, a member of class DynamicArray.
	set_length (see page 3268)	This is set_length, a member of class DynamicArray.

Public Operators

	Name	Description
	[] (see page 3268)	This is the overview for the [] operator overload.
	= (see page 3269)	Assigns data from another Dynamic Array.
	== (see page 3269)	Compares two Dynamic Arrays for equality.

Public Properties

	Name	Description
R	High (see page 3270)	Indicates the high bounds of the array.
	Length (see page 3270)	Represents the number of elements in the System::DynamicArray (see page 3261).
R	Low (see page 3271)	Indicates the low bound of a dynamic array.

35.1.6.2 DynamicArray::~DynamicArray Destructor

Frees a System::DynamicArray (see page 3261) object.

Pascal

```
destructor Destroy;
```

C++

```
~DynamicArray();
```

Description

Do not call the dynamic array destructor. To free a dynamically allocated System::DynamicArray (see page 3261), assign it a Length (see page 3270) of zero. System::DynamicArray (see page 3261) instances on the stack are freed when they go out of scope.

35.1.6.3 DynamicArray Constructor

This is the overview for the DynamicArray constructor overload.

Overload List

	Name	Description
	DynamicArray::DynamicArray () (see page 3264)	Creates and initializes a System::DynamicArray::DynamicArray (see page 3261) object.
	DynamicArray::DynamicArray (DynamicArray<T>&) (see page 3264)	Creates and initializes a System::DynamicArray::DynamicArray (see page 3261) object.

35.1.6.3.1 DynamicArray::DynamicArray Constructor ()

Creates and initializes a System::DynamicArray::DynamicArray (see page 3261) object.

Pascal

```
constructor Create;
constructor Create(const src: DynamicArray<T>&);
```

C++

```
DynamicArray();
DynamicArray(const DynamicArray<T>& src);
```

Description

When called with no parameters, the constructor creates an empty dynamic array. Otherwise, it copies another System::DynamicArray::DynamicArray (see page 3261).

35.1.6.4 DynamicArray Methods

The methods of the DynamicArray class are listed here.

Protected Methods

	Name	Description
	AllocData (see page 3265)	This is AllocData, a member of class DynamicArray.
	DecRefCount (see page 3266)	This is DecRefCount, a member of class DynamicArray.
	FreeData (see page 3267)	This is FreeData, a member of class DynamicArray.
	IncRefCount (see page 3268)	This is IncRefCount, a member of class DynamicArray.

	SetData (see page 3268)	This is SetData, a member of class DynamicArray.

Public Methods

	Name	Description
	Copy (see page 3265)	This is the overview for the Copy method overload.
	CopyRange (see page 3266)	This is the overview for the CopyRange method overload.
	DimCount (see page 3267)	This is DimCount, a member of class DynamicArray.
	get_high (see page 3267)	This is get_high, a member of class DynamicArray.
	get_length (see page 3267)	This is get_length, a member of class DynamicArray.
	get_low (see page 3267)	This is get_low, a member of class DynamicArray.
	set_length (see page 3268)	This is set_length, a member of class DynamicArray.

35.1.6.4.1 DynamicArray::AllocData Method

Pascal

```
function AllocData(count: Integer): T*; static;
```

C++

```
static T* AllocData(int count);
```

Description

This is AllocData, a member of class DynamicArray.

35.1.6.4.2 Copy Method

This is the overview for the Copy method overload.

Overload List

	Name	Description
	DynamicArray::Copy () (see page 3265)	Copies the contents of a System::DynamicArray (see page 3261) to the specified System::DynamicArray (see page 3261).
	DynamicArray::Copy (DynamicArray<T> &) (see page 3265)	Copies the contents of a System::DynamicArray (see page 3261) to the specified System::DynamicArray (see page 3261).

35.1.6.4.2.1 DynamicArray::Copy Method ()

Copies the contents of a System::DynamicArray (see page 3261) to the specified System::DynamicArray (see page 3261).

Pascal

```
function Copy: DynamicArray<T>;
procedure Copy(var dst: DynamicArray<T>);
```

C++

```
DynamicArray<T> Copy() const;
void Copy(DynamicArray<T> & dst) const;
```

Description

Use System::DynamicArray::Copy to make a copy of a dynamic array. Dynamic arrays are reference counted. Therefore assigning one to another simply copies the reference, not the data. The System::DynamicArray::Copy method, on the other hand, makes a separate copy of the dynamic array. The original contents of the destination array are discarded.

The following code copies the contents of i_array into temp; temp and i_array must be of the same System::DynamicArray (see page 3261) type.

```
temp = i_array.Copy();
```

See Also

System::DynamicArray::CopyRange (see page 3266)

35.1.6.4.3 CopyRange Method

This is the overview for the CopyRange method overload.

Overload List

	Name	Description
	DynamicArray::CopyRange (DynamicArray<T> &, int, int) (see page 3266)	Copies a range of the contents of a System::DynamicArray (see page 3261) within a specified range.
	DynamicArray::CopyRange (int, int) (see page 3266)	Copies a range of the contents of a System::DynamicArray (see page 3261) within a specified range.

35.1.6.4.3.1 DynamicArray::CopyRange Method (DynamicArray<T> &, int, int)

Copies a range of the contents of a System::DynamicArray (see page 3261) within a specified range.

Pascal

```
function CopyRange(startIndex: Integer; count: Integer): DynamicArray<T>;
procedure CopyRange(var dst: DynamicArray<T>; startIndex: Integer; count: Integer);
```

C++

```
DynamicArray<T> CopyRange(int startIndex, int count) const;
void CopyRange(DynamicArray<T> & dst, int startIndex, int count) const;
```

Description

Use the System::DynamicArray::CopyRange method to copy a subrange of a dynamic array. The System::DynamicArray::CopyRange method is similar to the Copy (see page 3265) method except that you can specify the startIndex and the number of items to copy from the source object. The size of the destination array will be truncated or expanded to accommodate what has been copied.

newArray = array.System::DynamicArray::CopyRange(3, 5);

35.1.6.4.4 DynamicArray::DecRefCount Method

Pascal

```
procedure DecRefCount;
```

C++

```
void DecRefCount();
```

Description

This is DecRefCount, a member of class DynamicArray.

35.1.6.4.5 DynamicArray::DimCount Method

Pascal

```
function DimCount: Integer; static;
```

C++

```
static int DimCount();
```

Description

This is DimCount, a member of class DynamicArray.

35.1.6.4.6 DynamicArray::FreeData Method

Pascal

```
procedure FreeData;
```

C++

```
void FreeData();
```

Description

This is FreeData, a member of class DynamicArray.

35.1.6.4.7 DynamicArray::get_high Method

Pascal

```
function get_high: Integer;
```

C++

```
int get_high() const;
```

Description

This is get_high, a member of class DynamicArray.

35.1.6.4.8 DynamicArray::get_length Method

Pascal

```
function get_length: Integer;
```

C++

```
int get_length() const;
```

Description

This is get_length, a member of class DynamicArray.

35.1.6.4.9 DynamicArray::get_low Method

Pascal

```
function get_low: Integer;
```

C++

```
int get_low() const;
```

Description

This is get_low, a member of class DynamicArray.

35.1.6.4.10 DynamicArray::IncRefCount Method

Pascal

```
procedure IncRefCount;
```

C++

```
void IncRefCount();
```

Description

This is IncRefCount, a member of class DynamicArray.

35.1.6.4.11 DynamicArray::set_length Method

Pascal

```
procedure set_length(l: Integer);
```

C++

```
void set_length(int l);
```

Description

This is set_length, a member of class DynamicArray.

35.1.6.4.12 DynamicArray::SetData Method

Pascal

```
procedure SetData(var t: T);
```

C++

```
void SetData(T* t);
```

Description

This is SetData, a member of class DynamicArray.

35.1.6.5 DynamicArray Operators

The operators of the DynamicArray class are listed here.

Public Operators

	Name	Description
	[] (see page 3268)	This is the overview for the [] operator overload.
	= (see page 3269)	Assigns data from another Dynamic Array.
	== (see page 3269)	Compares two Dynamic Arrays for equality.

35.1.6.5.1 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	DynamicArray::[] (int) (see page 3269)	Provides access to the data held by the dynamic array.
	DynamicArray::[] (int) (see page 3269)	Provides access to the data held by the dynamic array.

35.1.6.5.1.1 DynamicArray::[] Operator (int)

Provides access to the data held by the dynamic array.

Pascal

```
operator [](index: Integer);
operator [](index: Integer);
```

C++

```
T& operator [](int index);
T operator [](int index) const;
```

Description

The const version of the System::DynamicArray (see page 3261)::[] returns a copy of the data at the specified subscript. The non-const version returns a reference to the data, allowing you to modify the data.

cout << i_array[0]=10;

35.1.6.5.2 DynamicArray::= Operator

Assigns data from another Dynamic Array.

Pascal

```
operator =(const src: DynamicArray<T>&);
```

C++

```
DynamicArray& operator =(const DynamicArray<T>& src);
```

Description

Dynamic Arrays are reference counted. When a System::DynamicArray (see page 3261) is assigned to another, only the reference is assigned (and the reference count adjusted), the contents of the source are not copied. Similarly, when two Dynamic Arrays are compared, only the references are compared, not the contents. To create a separate copy of the contents of a System::DynamicArray (see page 3261), use the Copy (see page 3265) (or CopyRange (see page 3266)) method.

See Also

System::DynamicArray::Copy (see page 3265)

35.1.6.5.3 DynamicArray::== Operator

Compares two Dynamic Arrays for equality.

Pascal

```
operator ==(const other: DynamicArray<T> &);
```

C++

```
bool operator ==(const DynamicArray<T> & other) const;
```

Description

The == System::DynamicArray (see page 3261)::== determines whether two dynamic arrays refer to the same memory. It does not compare the contents of the arrays.

35.1.6.6 DynamicArray Properties

The properties of the DynamicArray class are listed here.

Public Properties

	Name	Description
	High (see page 3270)	Indicates the high bounds of the array.
	Length (see page 3270)	Represents the number of elements in the System::DynamicArray (see page 3261).
	Low (see page 3271)	Indicates the low bound of a dynamic array.

35.1.6.6.1 DynamicArray::High Property

Indicates the high bounds of the array.

Pascal

```
property High: Integer;
```

C++

```
__property int High;
```

Description

Reading the System::DynamicArray::High property returns Length (see page 3270)-1.

The following function returns the sum of the contents of an integer type System::DynamicArray (see page 3261).

int TotalArray(const System::DynamicArray (see page 3261)<int>& arrayOfInt)

```
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

See Also

System::DynamicArray::Length (see page 3270)

System::DynamicArray::Low (see page 3271)

35.1.6.6.2 DynamicArray::Length Property

Represents the number of elements in the System::DynamicArray (see page 3261).

Pascal

```
property Length: Integer;
```

C++

```
__property int Length;
```

Description

Use the System::DynamicArray::Length property to set or get the length of the dynamic array. The following code sets the length of a System::DynamicArray (see page 3261) to 10.

System::DynamicArray (see page 3261)<int> arrayOfInt;

```
arrayOfInt.Length = 10;
cout << "ArrayLength: " << arrayOfInt.Length << endl;
```

SetSystem::DynamicArray::Length(arrayOfInt, 10);

To free a dynamic array, simply set its System::DynamicArray::Length to 0:

arrayOfInt.System::DynamicArray::Length = 0;

See Also

System::DynamicArray::High (see page 3270)

35.1.6.6.3 DynamicArray::Low Property

Indicates the low bound of a dynamic array.

Pascal

```
property Low: Integer;
```

C++

```
__property int Low;
```

Description

The System::DynamicArray::Low property always returns 0. The following function returns the sum of the contents of an integer type System::DynamicArray (see page 3261).

int TotalArray(const System::DynamicArray (see page 3261)<int>& arrayOfInt)

```
{
int total=0;
for (int i=arrayOfInt.Low; i<=arrayOfInt.High; i++)
    total += arrayOfInt[i];
return total;
}
```

See Also

System::DynamicArray::High (see page 3270)

35.1.7 DynArrayException Class

Class Hierarchy

Pascal

```
DynArrayException = class;
```

C++

```
class DynArrayException;
```

File

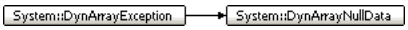
sysdyn.h

Description

This is class System::DynArrayException.

35.1.8 DynArrayNullData Class

Class Hierarchy



Pascal

```
DynArrayNullData = class(DynArrayException);
```

C++

```
class DynArrayNullData : public DynArrayException;
```

File

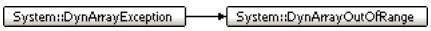
sysdyn.h

Description

This is class System::DynArrayNullData.

35.1.9 DynArrayOutOfRange Class

Class Hierarchy



Pascal

```
DynArrayOutOfRange = class(DynArrayException);
```

C++

```
class DynArrayOutOfRange : public DynArrayException;
```

File

sysdyn.h

Description

This is class System::DynArrayOutOfRange.

35.1.9.1 DynArrayOutOfRange Members

The following tables list the members exposed by DynArrayOutOfRange.

Public Data Members

	Name	Description
	m_Count (see page 3273)	This is m_Count, a member of class DynArrayOutOfRange.
	m_Index (see page 3273)	This is m_Index, a member of class DynArrayOutOfRange.

Public Methods

	Name	Description
	DynArrayOutOfRange (see page 3273)	This is DynArrayOutOfRange, a member of class DynArrayOutOfRange.

35.1.9.2 DynArrayOutOfRange::DynArrayOutOfRange Constructor

Pascal

```
constructor Create(index: Integer; count: Integer);
```

C++

```
DynArrayOutOfRange(int index, int count);
```

Description

This is DynArrayOutOfRange, a member of class DynArrayOutOfRange.

35.1.9.3 DynArrayOutOfRange Data Members

The data members of the DynArrayOutOfRange class are listed here.

Public Data Members

	Name	Description
	m_Count (see page 3273)	This is m_Count, a member of class DynArrayOutOfRange.
	m_Index (see page 3273)	This is m_Index, a member of class DynArrayOutOfRange.

35.1.9.3.1 DynArrayOutOfRange::m_Count Data Member

Pascal

```
m_Count: Integer;
```

C++

```
int m_Count;
```

Description

This is m_Count, a member of class DynArrayOutOfRange.

35.1.9.3.2 DynArrayOutOfRange::m_Index Data Member

Pascal

```
m_Index: Integer;
```

C++

```
int m_Index;
```

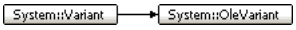
Description

This is m_Index, a member of class DynArrayOutOfRange.

35.1.10 OleVariant Class

System::OleVariant is a C++ implementation of the Delphi intrinsic type System::OleVariant.

Class Hierarchy



Pascal

```
OleVariant = class(Variant);
```

C++

```
class OleVariant : protected Variant;
```

File

sysvari.h

Description

System::OleVariant is a special descendant of the Variant (see page 3421) class that is used to represent COM interfaces or data that is passed over a COM interface. System::OleVariant inherits from Variant (see page 3421) as a protected class so that it can prevent operations on AnsiString (see page 3458) objects, which are not compatible with COM.

The syntax for using System::OleVariant is the same as that for using the Variant (see page 3421) class. Like the Variant (see page 3421) class, System::OleVariant has the following characteristics:

System::OleVariants can contain integer values, real values, string values, boolean values, date-and-time values, and Automation objects. In addition, System::OleVariants can contain arrays of varying size and dimension with elements of any of these types.

The special System::OleVariant value Unassigned is used to indicate that a System::OleVariant has not yet been assigned a value, and the special System::OleVariant value Null is used to indicate unknown or missing data.

A System::OleVariant can be combined with other System::OleVariants and it can be constructed from any of the following data types. The compiler automatically performs the necessary type conversions.

shortWordBool

intByte

floatVariant

doubleWideString

Currencywchar_t*

TDateTimeOle2::IDispatch (see page 3494)*

boolOle2::IUnknown (see page 3756)*

When an System::OleVariant contains an Automation object, the System::OleVariant can be used to get and set properties of the object, and to invoke methods on the object.

System::OleVariant variables are always initialized to be Unassigned when they are first created. This is true whether a System::OleVariant variable is global, local, or part of a structure such as an array.

Note that while System::OleVariants offer great flexibility, they also consume more memory than regular variables, and operations on System::OleVariants are substantially slower than operations on statically typed values.

Note: Use only the operators declared within System::OleVariant. The compiler will ignore any operators you overload yourself.

Note: For binary operators, if one operand is of type System::OleVariant, the other operand is automatically converted to type System::OleVariant.

See Also

System::Variant (see page 3421)

35.1.10.1 OleVariant Members

The following tables list the members exposed by OleVariant.

Public Data Members

	Name	Description
	TVarData::Reserved1 (see page 3284)	This is TVarData::Reserved1, a member of class OleVariant.
	TVarData::Reserved2 (see page 3284)	This is TVarData::Reserved2, a member of class OleVariant.
	TVarData::Reserved3 (see page 3284)	This is TVarData::Reserved3, a member of class OleVariant.
	TVarData::VArray (see page 3284)	This is TVarData::VArray, a member of class OleVariant.
	TVarData::VBoolean (see page 3285)	This is TVarData::VBoolean, a member of class OleVariant.
	TVarData::VByte (see page 3285)	This is TVarData::VByte, a member of class OleVariant.
	TVarData::VCurrency (see page 3285)	This is TVarData::VCurrency, a member of class OleVariant.
	TVarData::VDate (see page 3285)	This is TVarData::VDate, a member of class OleVariant.
	TVarData::VDispatch (see page 3285)	This is TVarData::VDispatch, a member of class OleVariant.
	TVarData::VDouble (see page 3286)	This is TVarData::VDouble, a member of class OleVariant.
	TVarData::VError (see page 3286)	This is TVarData::VError, a member of class OleVariant.
	TVarData::VInteger (see page 3286)	This is TVarData::VInteger, a member of class OleVariant.
	TVarData::VOleStr (see page 3286)	This is TVarData::VOleStr, a member of class OleVariant.
	TVarData::VPointer (see page 3287)	This is TVarData::VPointer, a member of class OleVariant.
	TVarData::VSingle (see page 3287)	This is TVarData::VSingle, a member of class OleVariant.
	TVarData::VSmallint (see page 3287)	This is TVarData::VSmallint, a member of class OleVariant.
	TVarData::VType (see page 3287)	This is TVarData::VType, a member of class OleVariant.
	TVarData::VUnknown (see page 3287)	This is TVarData::VUnknown, a member of class OleVariant.
	Variant::- (see page 3288)	This is Variant::-, a member of class OleVariant.
	Variant::! (see page 3288)	This is Variant::!, a member of class OleVariant.
	Variant::%= (see page 3288)	This is Variant::%=, a member of class OleVariant.
	Variant::&= (see page 3288)	This is Variant::&=, a member of class OleVariant.
	Variant::*= (see page 3289)	This is Variant::*=, a member of class OleVariant.
	Variant::/= (see page 3289)	This is Variant::/=, a member of class OleVariant.
	Variant::^= (see page 3289)	This is Variant::^=, a member of class OleVariant.
	Variant::__int64 (see page 3289)	This is Variant::__int64, a member of class OleVariant.
	Variant::|= (see page 3289)	This is Variant::|=, a member of class OleVariant.
	Variant::+= (see page 3290)	This is Variant::+=, a member of class OleVariant.
	Variant::<<= (see page 3290)	This is Variant::<<=, a member of class OleVariant.
	Variant::-= (see page 3290)	This is Variant::-=, a member of class OleVariant.
	Variant::>>= (see page 3290)	This is Variant::>>=, a member of class OleVariant.
	Variant::AnsiString (see page 3291)	This is Variant::AnsiString, a member of class OleVariant.
	Variant::ArrayDimCount (see page 3291)	This is Variant::ArrayDimCount, a member of class OleVariant.
	Variant::ArrayHighBound (see page 3291)	This is Variant::ArrayHighBound, a member of class OleVariant.
	Variant::ArrayLock (see page 3291)	This is Variant::ArrayLock, a member of class OleVariant.
	Variant::ArrayLowBound (see page 3291)	This is Variant::ArrayLowBound, a member of class OleVariant.
	Variant::ArrayRedim (see page 3292)	This is Variant::ArrayRedim, a member of class OleVariant.
	Variant::ArrayUnlock (see page 3292)	This is Variant::ArrayUnlock, a member of class OleVariant.

	Variant::AsType (see page 3292)	This is Variant::AsType, a member of class OleVariant.
	Variant::bool (see page 3292)	This is Variant::bool, a member of class OleVariant.
	Variant::Byte* (see page 3293)	This is Variant::Byte*, a member of class OleVariant.
	Variant::ChangeType (see page 3293)	This is Variant::ChangeType, a member of class OleVariant.
	Variant::char (see page 3293)	This is Variant::char, a member of class OleVariant.
	Variant::Clear (see page 3293)	This is Variant::Clear, a member of class OleVariant.
	Variant::Currency (see page 3293)	This is Variant::Currency, a member of class OleVariant.
	Variant::Currency* (see page 3294)	This is Variant::Currency*, a member of class OleVariant.
	Variant::double (see page 3294)	This is Variant::double, a member of class OleVariant.
	Variant::double* (see page 3294)	This is Variant::double*, a member of class OleVariant.
	Variant::float (see page 3294)	This is Variant::float, a member of class OleVariant.
	Variant::float* (see page 3295)	This is Variant::float*, a member of class OleVariant.
	Variant::GetElement (see page 3295)	This is Variant::GetElement, a member of class OleVariant.
	Variant::GetError (see page 3295)	This is Variant::GetError, a member of class OleVariant.
	Variant::IDispatch* (see page 3295)	This is Variant::IDispatch*, a member of class OleVariant.
	Variant::IInterface* (see page 3295)	This is Variant::IInterface*, a member of class OleVariant.
	Variant::int (see page 3296)	This is Variant::int, a member of class OleVariant.
	Variant::int* (see page 3296)	This is Variant::int*, a member of class OleVariant.
	Variant::IsArray (see page 3296)	This is Variant::IsArray, a member of class OleVariant.
	Variant::IsEmpty (see page 3296)	This is Variant::IsEmpty, a member of class OleVariant.
	Variant::IsNull (see page 3297)	This is Variant::IsNull, a member of class OleVariant.
	Variant::IUnknown* (see page 3297)	This is Variant::IUnknown*, a member of class OleVariant.
	Variant::long (see page 3297)	This is Variant::long, a member of class OleVariant.
	Variant::long* (see page 3297)	This is Variant::long*, a member of class OleVariant.
	Variant::PutElement (see page 3297)	This is Variant::PutElement, a member of class OleVariant.
	Variant::SetError (see page 3298)	This is Variant::SetError, a member of class OleVariant.
	Variant::short (see page 3298)	This is Variant::short, a member of class OleVariant.
	Variant::short* (see page 3298)	This is Variant::short*, a member of class OleVariant.
	Variant::signed char (see page 3298)	This is Variant::signed char, a member of class OleVariant.
	Variant::signed char* (see page 3299)	This is Variant::signed char*, a member of class OleVariant.
	Variant::TDateTime (see page 3299)	This is Variant::TDateTime, a member of class OleVariant.
	Variant::TDateTime* (see page 3299)	This is Variant::TDateTime*, a member of class OleVariant.
	Variant::Type (see page 3299)	This is Variant::Type, a member of class OleVariant.
	Variant::UnicodeString (see page 3299)	This is Variant::UnicodeString, a member of class OleVariant.
	Variant::unsigned char (see page 3300)	This is Variant::unsigned char, a member of class OleVariant.
	Variant::unsigned int (see page 3300)	This is Variant::unsigned int, a member of class OleVariant.
	Variant::unsigned int* (see page 3300)	This is Variant::unsigned int*, a member of class OleVariant.
	Variant::unsigned long (see page 3300)	This is Variant::unsigned long, a member of class OleVariant.
	Variant::unsigned short (see page 3301)	This is Variant::unsigned short, a member of class OleVariant.
	Variant::unsigned short* (see page 3301)	This is Variant::unsigned short*, a member of class OleVariant.
	Variant::WideString (see page 3301)	This is Variant::WideString, a member of class OleVariant.

Public Methods

	Name	Description
	~OleVariant (see page 3279)	Frees an instance of System::OleVariant (see page 3274).
	~Variant (see page 3424)	Frees a System::Variant::Variant (see page 3421) object.
	ArrayDimCount (see page 3429)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 3429)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 3429)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 3430)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 3430)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 3430)	Unlocks a variant array that was previously locked by ArrayLock (see page 3429).

	AsType (see page 3431)	Casts a System::Variant::Variant (see page 3421) object to the type specified by the argument VarType.
	ChangeType (see page 3431)	Changes the data type of a System::Variant::Variant (see page 3421) object once it has been instantiated.
	Clear (see page 3431)	System::Variant::Clears the value of the System::Variant::Variant (see page 3421).
	Empty (see page 3431)	This is Empty, a member of class Variant.
	GetBaseVariant (see page 3432)	Returns the System::Variant (see page 3421) that is assigned as the value of this System::Variant (see page 3421) when Type (see page 3436) is varSystem::Variant (see page 3421).
	GetElement (see page 3432)	This is the overview for the GetElement method overload.
	GetError (see page 3433)	Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and returns the integer value.
	IsArray (see page 3433)	Queries the VType of the base class TVarData (see page 3740) for the varArray (see page 3861) bit.
	IsEmpty (see page 3433)	Queries the VType of the base class TVarData (see page 3740).
	IsNull (see page 3434)	Queries the VType of the base class TVarData (see page 3740) for the varNull (see page 3868) bit.
	NoParam (see page 3434)	This is NoParam, a member of class Variant.
	OleVariant (see page 3279)	This is the overview for the OleVariant constructor overload.
	PutElement (see page 3434)	This is the overview for the PutElement method overload.
	Sanitized (see page 3435)	This is Sanitized, a member of class Variant.
	SetError (see page 3435)	Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and assigns the integer err as the value of the varError (see page 3866).
	Type (see page 3436)	Returns type code of a System::Variant (see page 3421) instance.
	Variant (see page 3424)	This is the overview for the Variant constructor overload.

Public Operators

	Name	Description
	- (see page 3439)	This is the overview for the - operator overload.
	! (see page 3439)	Returns a new System::Variant::Variant (see page 3421) that is the logical negation of this System::Variant::Variant (see page 3421).
	!= (see page 3301)	This is the overview for the != operator overload.
	!= (see page 3440)	This is the overview for the != operator overload.
	% (see page 3440)	This is the overview for the % operator overload.
	%= (see page 3441)	Returns this System::Variant::Variant (see page 3421) as the remainder from dividing its original value and the System::Variant::Variant (see page 3421) specified by rhs.
	& (see page 3442)	This is the overview for the & operator overload.
	&= (see page 3442)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise and operation.
	* (see page 3443)	This is the overview for the * operator overload.
	*= (see page 3443)	Multiplies this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) specified by rhs.
	/ (see page 3444)	This is the overview for the / operator overload.
	/= (see page 3444)	Divides this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) rhs.
	^ (see page 3445)	This is the overview for the ^ operator overload.
	^= (see page 3445)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise xor operation.
	__int64 (see page 3446)	This is __int64, a member of class Variant.
	| (see page 3446)	This is the overview for the | operator overload.
	|= (see page 3446)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise or operation.
	+ (see page 3447)	This is the overview for the + operator overload.
	+= (see page 3447)	Adds the System::Variant::Variant (see page 3421) specified by rhs to this System::Variant::Variant (see page 3421) object.
	< (see page 3302)	This is the overview for the < operator overload.
	< (see page 3448)	This is the overview for the < operator overload.

	<< (see page 3448)	This is the overview for the << operator overload.
	<<= (see page 3449)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3421).
	<= (see page 3303)	This is the overview for the <= operator overload.
	<= (see page 3449)	This is the overview for the <= operator overload.
	-= (see page 3454)	Subtracts the System::Variant::Variant (see page 3421) specified by rhs from this System::Variant::Variant (see page 3421) object.
	= (see page 3303)	This is the overview for the = operator overload.
	= (see page 3450)	This is the overview for the = operator overload.
	== (see page 3305)	This is the overview for the == operator overload.
	== (see page 3455)	This is the overview for the == operator overload.
	> (see page 3306)	This is the overview for the > operator overload.
	> (see page 3456)	This is the overview for the > operator overload.
	>= (see page 3306)	This is the overview for the >= operator overload.
	>= (see page 3456)	This is the overview for the >= operator overload.
	>> (see page 3457)	This is the overview for the >> operator overload.
	>>= (see page 3458)	Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 3421).
	AnsiString (see page 3458)	Converts the System::Variant::Variant (see page 3421) to an AnsiString object.
	bool (see page 3458)	Converts the System::Variant::Variant (see page 3421) to a boolean value.
	char (see page 3459)	This is char, a member of class Variant.
	Currency (see page 3459)	Converts the System::Variant::Variant (see page 3421) to a Currency object.
	Currency* (see page 3459)	This is Currency*, a member of class Variant.
	double (see page 3460)	Converts the System::Variant::Variant (see page 3421) to a double value.
	double* (see page 3460)	This is double*, a member of class Variant.
	float (see page 3460)	Converts the System::Variant::Variant (see page 3421) to a float value.
	float* (see page 3460)	This is float*, a member of class Variant.
	IDispatch* (see page 3461)	Converts the System::Variant (see page 3421) to an System::Variant::IDispatch (see page 3494)* interface pointer.
	IInterface* (see page 3461)	This is IInterface*, a member of class Variant.
	int (see page 3461)	Converts the System::Variant::Variant (see page 3421) to an integer value.
	int* (see page 3461)	This is int*, a member of class Variant.
	IUnknown* (see page 3462)	Converts the System::Variant (see page 3421) to an System::Variant::IUnknown (see page 3756)* interface pointer.
	long (see page 3462)	This is long, a member of class Variant.
	long* (see page 3462)	This is long*, a member of class Variant.
	short (see page 3462)	Converts the System::Variant::Variant (see page 3421) to a short value.
	short* (see page 3463)	This is short*, a member of class Variant.
	signed char (see page 3463)	This is signed char, a member of class Variant.
	signed char* (see page 3463)	This is signed char*, a member of class Variant.
	TDateTime (see page 3463)	Converts the System::Variant::Variant (see page 3421) to a TDateTime object.
	TDateTime* (see page 3464)	This is TDateTime*, a member of class Variant.
	UnicodeString (see page 3464)	This is UnicodeString, a member of class Variant.
	unsigned char (see page 3464)	Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value.
	unsigned char* (see page 3465)	Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value. And returns a reference.
	unsigned int (see page 3465)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 3465)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 3465)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 3466)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 3466)	Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value.
	unsigned short* (see page 3466)	Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value and returns a reference to the value.
	WideString (see page 3466)	This is WideString, a member of class Variant.

35.1.10.2 OleVariant::~OleVariant Destructor

Frees an instance of System::OleVariant (see page 3274).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~OleVariant();
```

Description

System::OleVariant (see page 3274)::~OleVariant (see page 3274) is the destructor for System::OleVariant (see page 3274) class.

35.1.10.3 OleVariant Constructor

This is the overview for the OleVariant constructor overload.

Overload List

	Name	Description
	OleVariant::OleVariant () (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (__int64) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (__int64*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (AnsiString&) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (bool) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (char) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (char*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (Currency) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (Currency*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (double) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (double*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (float) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (float*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.

	OleVariant::OleVariant (IDispatch* const) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (IInterface* const) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (int) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (int*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (int*, int, Word) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (IUnknown* const) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (long*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (OleVariant&) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (OleVariant*, int) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (short) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (short*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (signed char*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (TDateTime) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (TDateTime*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (UnicodeString&) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (unsigned char) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (unsigned char*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (unsigned int) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (unsigned int*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (unsigned long*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (unsigned short) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (unsigned short*) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (Variant&) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.

	OleVariant::OleVariant (wchar_t* const, bool) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.
	OleVariant::OleVariant (WideString&) (see page 3281)	Creates an instance of System::OleVariant (see page 3274) with a specified value.

35.1.10.3.1 OleVariant::OleVariant Constructor ()

Creates an instance of System::OleVariant (see page 3274) with a specified value.

Pascal

```
constructor Create;
constructor Create(const src: OleVariant&);
constructor Create(const src: Variant&);
constructor Create(const src: char);
constructor Create(const src: unsigned char);
constructor Create(const src: short);
constructor Create(const src: unsigned short);
constructor Create(const src: Integer);
constructor Create(const src: unsigned int);
constructor Create(const src: float);
constructor Create(const src: double);
constructor Create(const src: __int64);
constructor Create(const src: Currency);
constructor Create(const src: TDateTime);
constructor Create(const src: WideString&);
constructor Create(const src: UnicodeString&);
constructor Create(const src: bool);
constructor Create(const src: AnsiString&);
constructor Create(src: wchar_t* const; isBSTR: bool = true);
constructor Create(src: IDispatch* const);
constructor Create(src: IUnknown* const);
constructor Create(src: IInterface* const);
constructor Create(const src: char*);
constructor Create(var src: signed char);
constructor Create(var src: unsigned char);
constructor Create(var src: short);
constructor Create(var src: unsigned short);
constructor Create(var src: Integer);
constructor Create(var src: unsigned int);
constructor Create(var src: long);
constructor Create(var src: unsigned long);
constructor Create(var src: float);
constructor Create(var src: double);
constructor Create(var src: __int64);
constructor Create(var src: Currency);
constructor Create(var src: TDateTime);
constructor Create(const bounds: int*; const boundsSize: Integer; varType: Word);
constructor Create(const values: OleVariant*; const valuesSize: Integer);
```

C++

```
__fastcall OleVariant();
__fastcall OleVariant(const OleVariant& src);
__fastcall OleVariant(const Variant& src);
__fastcall OleVariant(const char src);
__fastcall OleVariant(const unsigned char src);
__fastcall OleVariant(const short src);
__fastcall OleVariant(const unsigned short src);
__fastcall OleVariant(const int src);
__fastcall OleVariant(const unsigned int src);
__fastcall OleVariant(const float src);
__fastcall OleVariant(const double src);
```

```
__fastcall OleVariant(const __int64 src);
__fastcall OleVariant(const Currency src);
__fastcall OleVariant(const TDateTime src);
__fastcall OleVariant(const WideString& src);
__fastcall OleVariant(const UnicodeString& src);
__fastcall OleVariant(const bool src);
__fastcall OleVariant(const AnsiString& src);
__fastcall OleVariant(wchar_t* const src, bool isBSTR = true);
__fastcall OleVariant(IDispatch* const src);
__fastcall OleVariant(IUnknown* const src);
__fastcall OleVariant(IInterface* const src);
__fastcall OleVariant(const char* src);
__fastcall OleVariant(signed char* src);
__fastcall OleVariant(unsigned char* src);
__fastcall OleVariant(short* src);
__fastcall OleVariant(unsigned short* src);
__fastcall OleVariant(int* src);
__fastcall OleVariant(unsigned int* src);
__fastcall OleVariant(long* src);
__fastcall OleVariant(unsigned long* src);
__fastcall OleVariant(float* src);
__fastcall OleVariant(double* src);
__fastcall OleVariant(__int64* src);
__fastcall OleVariant(Currency* src);
__fastcall OleVariant(TDateTime* src);
__fastcall OleVariant(const int* bounds, const int boundsSize, Word varType);
__fastcall OleVariant(const OleVariant* values, const int valuesSize);
```

Description

The System::OleVariant (see page 3274) constructor is overloaded to create an System::OleVariant (see page 3274), converting the src argument as appropriate. If no argument is given, System::OleVariant (see page 3274) creates an System::OleVariant (see page 3274) of type Unassigned.

35.1.10.4 OleVariant Data Members

The data members of the OleVariant class are listed here.

Public Data Members

	Name	Description
	TVarData::Reserved1 (see page 3284)	This is TVarData::Reserved1, a member of class OleVariant.
	TVarData::Reserved2 (see page 3284)	This is TVarData::Reserved2, a member of class OleVariant.
	TVarData::Reserved3 (see page 3284)	This is TVarData::Reserved3, a member of class OleVariant.
	TVarData::VArray (see page 3284)	This is TVarData::VArray, a member of class OleVariant.
	TVarData::VBoolean (see page 3285)	This is TVarData::VBoolean, a member of class OleVariant.
	TVarData::VByte (see page 3285)	This is TVarData::VByte, a member of class OleVariant.
	TVarData::VCurrency (see page 3285)	This is TVarData::VCurrency, a member of class OleVariant.
	TVarData::VDate (see page 3285)	This is TVarData::VDate, a member of class OleVariant.
	TVarData::VDispatch (see page 3285)	This is TVarData::VDispatch, a member of class OleVariant.
	TVarData::VDouble (see page 3286)	This is TVarData::VDouble, a member of class OleVariant.
	TVarData::VError (see page 3286)	This is TVarData::VError, a member of class OleVariant.
	TVarData::VInteger (see page 3286)	This is TVarData::VInteger, a member of class OleVariant.
	TVarData::VOleStr (see page 3286)	This is TVarData::VOleStr, a member of class OleVariant.
	TVarData::VPointer (see page 3287)	This is TVarData::VPointer, a member of class OleVariant.
	TVarData::VSingle (see page 3287)	This is TVarData::VSingle, a member of class OleVariant.
	TVarData::VSmallint (see page 3287)	This is TVarData::VSmallint, a member of class OleVariant.
	TVarData::VType (see page 3287)	This is TVarData::VType, a member of class OleVariant.
	TVarData::VUnknown (see page 3287)	This is TVarData::VUnknown, a member of class OleVariant.
	Variant::- (see page 3288)	This is Variant::-, a member of class OleVariant.

	Variant::! (see page 3288)	This is Variant::!, a member of class OleVariant.
	Variant::%= (see page 3288)	This is Variant::%=, a member of class OleVariant.
	Variant::&= (see page 3288)	This is Variant::&=, a member of class OleVariant.
	Variant::*= (see page 3289)	This is Variant::*=, a member of class OleVariant.
	Variant::/= (see page 3289)	This is Variant::/=, a member of class OleVariant.
	Variant::^= (see page 3289)	This is Variant::^=, a member of class OleVariant.
	Variant::__int64 (see page 3289)	This is Variant::__int64, a member of class OleVariant.
	Variant::|= (see page 3289)	This is Variant::|=, a member of class OleVariant.
	Variant::+= (see page 3290)	This is Variant::+=, a member of class OleVariant.
	Variant::<<= (see page 3290)	This is Variant::<<=, a member of class OleVariant.
	Variant::-= (see page 3290)	This is Variant::-=, a member of class OleVariant.
	Variant::>>= (see page 3290)	This is Variant::>>=, a member of class OleVariant.
	Variant::AnsiString (see page 3291)	This is Variant::AnsiString, a member of class OleVariant.
	Variant::ArrayDimCount (see page 3291)	This is Variant::ArrayDimCount, a member of class OleVariant.
	Variant::ArrayHighBound (see page 3291)	This is Variant::ArrayHighBound, a member of class OleVariant.
	Variant::ArrayLock (see page 3291)	This is Variant::ArrayLock, a member of class OleVariant.
	Variant::ArrayLowBound (see page 3291)	This is Variant::ArrayLowBound, a member of class OleVariant.
	Variant::ArrayRedim (see page 3292)	This is Variant::ArrayRedim, a member of class OleVariant.
	Variant::ArrayUnlock (see page 3292)	This is Variant::ArrayUnlock, a member of class OleVariant.
	Variant::AsType (see page 3292)	This is Variant::AsType, a member of class OleVariant.
	Variant::bool (see page 3292)	This is Variant::bool, a member of class OleVariant.
	Variant::Byte* (see page 3293)	This is Variant::Byte*, a member of class OleVariant.
	Variant::ChangeType (see page 3293)	This is Variant::ChangeType, a member of class OleVariant.
	Variant::char (see page 3293)	This is Variant::char, a member of class OleVariant.
	Variant::Clear (see page 3293)	This is Variant::Clear, a member of class OleVariant.
	Variant::Currency (see page 3293)	This is Variant::Currency, a member of class OleVariant.
	Variant::Currency* (see page 3294)	This is Variant::Currency*, a member of class OleVariant.
	Variant::double (see page 3294)	This is Variant::double, a member of class OleVariant.
	Variant::double* (see page 3294)	This is Variant::double*, a member of class OleVariant.
	Variant::float (see page 3294)	This is Variant::float, a member of class OleVariant.
	Variant::float* (see page 3295)	This is Variant::float*, a member of class OleVariant.
	Variant::GetElement (see page 3295)	This is Variant::GetElement, a member of class OleVariant.
	Variant::GetError (see page 3295)	This is Variant::GetError, a member of class OleVariant.
	Variant::IDispatch* (see page 3295)	This is Variant::IDispatch*, a member of class OleVariant.
	Variant::IInterface* (see page 3295)	This is Variant::IInterface*, a member of class OleVariant.
	Variant::int (see page 3296)	This is Variant::int, a member of class OleVariant.
	Variant::int* (see page 3296)	This is Variant::int*, a member of class OleVariant.
	Variant::IsArray (see page 3296)	This is Variant::IsArray, a member of class OleVariant.
	Variant::IsEmpty (see page 3296)	This is Variant::IsEmpty, a member of class OleVariant.
	Variant::IsNull (see page 3297)	This is Variant::IsNull, a member of class OleVariant.
	Variant::IUnknown* (see page 3297)	This is Variant::IUnknown*, a member of class OleVariant.
	Variant::long (see page 3297)	This is Variant::long, a member of class OleVariant.
	Variant::long* (see page 3297)	This is Variant::long*, a member of class OleVariant.
	Variant::PutElement (see page 3297)	This is Variant::PutElement, a member of class OleVariant.
	Variant::SetError (see page 3298)	This is Variant::SetError, a member of class OleVariant.
	Variant::short (see page 3298)	This is Variant::short, a member of class OleVariant.
	Variant::short* (see page 3298)	This is Variant::short*, a member of class OleVariant.
	Variant::signed char (see page 3298)	This is Variant::signed char, a member of class OleVariant.
	Variant::signed char* (see page 3299)	This is Variant::signed char*, a member of class OleVariant.
	Variant::TDateTime (see page 3299)	This is Variant::TDateTime, a member of class OleVariant.
	Variant::TDateTime* (see page 3299)	This is Variant::TDateTime*, a member of class OleVariant.
	Variant::Type (see page 3299)	This is Variant::Type, a member of class OleVariant.
	Variant::UnicodeString (see page 3299)	This is Variant::UnicodeString, a member of class OleVariant.
	Variant::unsigned char (see page 3300)	This is Variant::unsigned char, a member of class OleVariant.

	Variant::unsigned int (see page 3300)	This is Variant::unsigned int, a member of class OleVariant.
	Variant::unsigned int* (see page 3300)	This is Variant::unsigned int*, a member of class OleVariant.
	Variant::unsigned long (see page 3300)	This is Variant::unsigned long, a member of class OleVariant.
	Variant::unsigned short (see page 3301)	This is Variant::unsigned short, a member of class OleVariant.
	Variant::unsigned short* (see page 3301)	This is Variant::unsigned short*, a member of class OleVariant.
	Variant::WideString (see page 3301)	This is Variant::WideString, a member of class OleVariant.

35.1.10.4.1 OleVariant::TVarData::Reserved1 Data Member

Pascal

```
TVarData::Reserved1: __USING;
```

C++

```
__USING TVarData::Reserved1;
```

Description

This is TVarData::Reserved1, a member of class OleVariant.

35.1.10.4.2 OleVariant::TVarData::Reserved2 Data Member

Pascal

```
TVarData::Reserved2: __USING;
```

C++

```
__USING TVarData::Reserved2;
```

Description

This is TVarData::Reserved2, a member of class OleVariant.

35.1.10.4.3 OleVariant::TVarData::Reserved3 Data Member

Pascal

```
TVarData::Reserved3: __USING;
```

C++

```
__USING TVarData::Reserved3;
```

Description

This is TVarData::Reserved3, a member of class OleVariant.

35.1.10.4.4 OleVariant::TVarData::VArray Data Member

Pascal

```
TVarData::VArray: __USING;
```

C++

```
__USING TVarData::VArray;
```

Description

This is TVarData::VArray, a member of class OleVariant.

35.1.10.4.5 OleVariant::TVarData::VBoolean Data Member

Pascal

```
TVarData::VBoolean: __USING;
```

C++

```
__USING TVarData::VBoolean;
```

Description

This is TVarData::VBoolean, a member of class OleVariant.

35.1.10.4.6 OleVariant::TVarData::VByte Data Member

Pascal

```
TVarData::VByte: __USING;
```

C++

```
__USING TVarData::VByte;
```

Description

This is TVarData::VByte, a member of class OleVariant.

35.1.10.4.7 OleVariant::TVarData::VCurrency Data Member

Pascal

```
TVarData::VCurrency: __USING;
```

C++

```
__USING TVarData::VCurrency;
```

Description

This is TVarData::VCurrency, a member of class OleVariant.

35.1.10.4.8 OleVariant::TVarData::VDate Data Member

Pascal

```
TVarData::VDate: __USING;
```

C++

```
__USING TVarData::VDate;
```

Description

This is TVarData::VDate, a member of class OleVariant.

35.1.10.4.9 OleVariant::TVarData::VDispatch Data Member

Pascal

```
TVarData::VDispatch: __USING;
```

C++

```
__USING TVarData::VDispatch;
```

Description

This is TVarData::VDispatch, a member of class OleVariant.

35.1.10.4.10 OleVariant::TVarData::VDouble Data Member

Pascal

```
TVarData::VDouble: __USING;
```

C++

```
__USING TVarData::VDouble;
```

Description

This is TVarData::VDouble, a member of class OleVariant.

35.1.10.4.11 OleVariant::TVarData::VError Data Member

Pascal

```
TVarData::VError: __USING;
```

C++

```
__USING TVarData::VError;
```

Description

This is TVarData::VError, a member of class OleVariant.

35.1.10.4.12 OleVariant::TVarData::VInteger Data Member

Pascal

```
TVarData::VInteger: __USING;
```

C++

```
__USING TVarData::VInteger;
```

Description

This is TVarData::VInteger, a member of class OleVariant.

35.1.10.4.13 OleVariant::TVarData::VOleStr Data Member

Pascal

```
TVarData::VOleStr: __USING;
```

C++

```
__USING TVarData::VOleStr;
```

Description

This is TVarData::VOleStr, a member of class OleVariant.

35.1.10.4.14 OleVariant::TVarData::VPointer Data Member

Pascal

```
TVarData::VPointer: __USING;
```

C++

```
__USING TVarData::VPointer;
```

Description

This is TVarData::VPointer, a member of class OleVariant.

35.1.10.4.15 OleVariant::TVarData::VSingle Data Member

Pascal

```
TVarData::VSingle: __USING;
```

C++

```
__USING TVarData::VSingle;
```

Description

This is TVarData::VSingle, a member of class OleVariant.

35.1.10.4.16 OleVariant::TVarData::VSmallint Data Member

Pascal

```
TVarData::VSmallint: __USING;
```

C++

```
__USING TVarData::VSmallint;
```

Description

This is TVarData::VSmallint, a member of class OleVariant.

35.1.10.4.17 OleVariant::TVarData::VType Data Member

Pascal

```
TVarData::VType: __USING;
```

C++

```
__USING TVarData::VType;
```

Description

This is TVarData::VType, a member of class OleVariant.

35.1.10.4.18 OleVariant::TVarData::VUnknown Data Member

Pascal

```
TVarData::VUnknown: __USING;
```

C++

```
__USING TVarData::VUnknown;
```

Description

This is TVarData::VUnknown, a member of class OleVariant.

35.1.10.4.19 OleVariant::Variant::- Data Member

Pascal

```
Variant::-: __USING;
```

C++

```
__USING Variant::-;
```

Description

This is Variant::-, a member of class OleVariant.

35.1.10.4.20 OleVariant::Variant::! Data Member

Pascal

```
Variant::!: __USING;
```

C++

```
__USING Variant::!;
```

Description

This is Variant::!, a member of class OleVariant.

35.1.10.4.21 OleVariant::Variant::%= Data Member

Pascal

```
Variant::%=: __USING;
```

C++

```
__USING Variant::%=;
```

Description

This is Variant::%=, a member of class OleVariant.

35.1.10.4.22 OleVariant::Variant::&= Data Member

Pascal

```
Variant::&=: __USING;
```

C++

```
__USING Variant::&=;
```

Description

This is Variant::&=, a member of class OleVariant.

35.1.10.4.23 OleVariant::Variant::*= Data Member

Pascal

```
Variant::*=: __USING;
```

C++

```
__USING Variant::*=;
```

Description

This is Variant::*=, a member of class OleVariant.

35.1.10.4.24 OleVariant::Variant::/= Data Member

Pascal

```
Variant::/=: __USING;
```

C++

```
__USING Variant::/=;
```

Description

This is Variant::/=, a member of class OleVariant.

35.1.10.4.25 OleVariant::Variant::^= Data Member

Pascal

```
Variant::^=: __USING;
```

C++

```
__USING Variant::^=;
```

Description

This is Variant::^=, a member of class OleVariant.

35.1.10.4.26 OleVariant::Variant::__int64 Data Member

Pascal

```
Variant::__int64: __USING;
```

C++

```
__USING Variant::__int64;
```

Description

This is Variant::__int64, a member of class OleVariant.

35.1.10.4.27 OleVariant::Variant::|= Data Member

Pascal

```
Variant::|=: __USING;
```

C++

```
__USING Variant::|=;
```

Description

This is Variant::|=, a member of class OleVariant.

35.1.10.4.28 OleVariant::Variant::+= Data Member

Pascal

```
Variant::+=: __USING;
```

C++

```
__USING Variant::+=;
```

Description

This is Variant::+=, a member of class OleVariant.

35.1.10.4.29 OleVariant::Variant::<<= Data Member

Pascal

```
Variant::<<=: __USING;
```

C++

```
__USING Variant::<<=;
```

Description

This is Variant::<<=, a member of class OleVariant.

35.1.10.4.30 OleVariant::Variant::-= Data Member

Pascal

```
Variant::-=: __USING;
```

C++

```
__USING Variant::-=;
```

Description

This is Variant::-=, a member of class OleVariant.

35.1.10.4.31 OleVariant::Variant::>>= Data Member

Pascal

```
Variant::>>=: __USING;
```

C++

```
__USING Variant::>>=;
```

Description

This is Variant::>>=, a member of class OleVariant.

35.1.10.4.32 OleVariant::Variant::AnsiString Data Member

Pascal

```
Variant::AnsiString: __USING;
```

C++

```
__USING Variant::AnsiString;
```

Description

This is Variant::AnsiString, a member of class OleVariant.

35.1.10.4.33 OleVariant::Variant::ArrayDimCount Data Member

Pascal

```
Variant::ArrayDimCount: __USING;
```

C++

```
__USING Variant::ArrayDimCount;
```

Description

This is Variant::ArrayDimCount, a member of class OleVariant.

35.1.10.4.34 OleVariant::Variant::ArrayHighBound Data Member

Pascal

```
Variant::ArrayHighBound: __USING;
```

C++

```
__USING Variant::ArrayHighBound;
```

Description

This is Variant::ArrayHighBound, a member of class OleVariant.

35.1.10.4.35 OleVariant::Variant::ArrayLock Data Member

Pascal

```
Variant::ArrayLock: __USING;
```

C++

```
__USING Variant::ArrayLock;
```

Description

This is Variant::ArrayLock, a member of class OleVariant.

35.1.10.4.36 OleVariant::Variant::ArrayLowBound Data Member

Pascal

```
Variant::ArrayLowBound: __USING;
```

C++

```
__USING Variant::ArrayLowBound;
```

Description

This is Variant::ArrayLowBound, a member of class OleVariant.

35.1.10.4.37 OleVariant::Variant::ArrayRedim Data Member

Pascal

```
Variant::ArrayRedim: __USING;
```

C++

```
__USING Variant::ArrayRedim;
```

Description

This is Variant::ArrayRedim, a member of class OleVariant.

35.1.10.4.38 OleVariant::Variant::ArrayUnlock Data Member

Pascal

```
Variant::ArrayUnlock: __USING;
```

C++

```
__USING Variant::ArrayUnlock;
```

Description

This is Variant::ArrayUnlock, a member of class OleVariant.

35.1.10.4.39 OleVariant::Variant::AsType Data Member

Pascal

```
Variant::AsType: __USING;
```

C++

```
__USING Variant::AsType;
```

Description

This is Variant::AsType, a member of class OleVariant.

35.1.10.4.40 OleVariant::Variant::bool Data Member

Pascal

```
Variant::bool: __USING;
```

C++

```
__USING Variant::bool;
```

Description

This is Variant::bool, a member of class OleVariant.

35.1.10.4.41 OleVariant::Variant::Byte* Data Member

Pascal

```
Variant::Byte*: __USING;
```

C++

```
__USING Variant::Byte*;
```

Description

This is Variant::Byte*, a member of class OleVariant.

35.1.10.4.42 OleVariant::Variant::ChangeType Data Member

Pascal

```
Variant::ChangeType: __USING;
```

C++

```
__USING Variant::ChangeType;
```

Description

This is Variant::ChangeType, a member of class OleVariant.

35.1.10.4.43 OleVariant::Variant::char Data Member

Pascal

```
Variant::char: __USING;
```

C++

```
__USING Variant::char;
```

Description

This is Variant::char, a member of class OleVariant.

35.1.10.4.44 OleVariant::Variant::Clear Data Member

Pascal

```
Variant::Clear: __USING;
```

C++

```
__USING Variant::Clear;
```

Description

This is Variant::Clear, a member of class OleVariant.

35.1.10.4.45 OleVariant::Variant::Currency Data Member

Pascal

```
Variant::Currency: __USING;
```

C++

```
__USING Variant::Currency;
```

Description

This is Variant::Currency, a member of class OleVariant.

35.1.10.4.46 OleVariant::Variant::Currency* Data Member

Pascal

```
Variant::Currency*: __USING;
```

C++

```
__USING Variant::Currency*;
```

Description

This is Variant::Currency*, a member of class OleVariant.

35.1.10.4.47 OleVariant::Variant::double Data Member

Pascal

```
Variant::double: __USING;
```

C++

```
__USING Variant::double;
```

Description

This is Variant::double, a member of class OleVariant.

35.1.10.4.48 OleVariant::Variant::double* Data Member

Pascal

```
Variant::double*: __USING;
```

C++

```
__USING Variant::double*;
```

Description

This is Variant::double*, a member of class OleVariant.

35.1.10.4.49 OleVariant::Variant::float Data Member

Pascal

```
Variant::float: __USING;
```

C++

```
__USING Variant::float;
```

Description

This is Variant::float, a member of class OleVariant.

35.1.10.4.50 OleVariant::Variant::float* Data Member

Pascal

```
Variant::float*: __USING;
```

C++

```
__USING Variant::float*;
```

Description

This is Variant::float*, a member of class OleVariant.

35.1.10.4.51 OleVariant::Variant::GetElement Data Member

Pascal

```
Variant::GetElement: __USING;
```

C++

```
__USING Variant::GetElement;
```

Description

This is Variant::GetElement, a member of class OleVariant.

35.1.10.4.52 OleVariant::Variant::GetError Data Member

Pascal

```
Variant::GetError: __USING;
```

C++

```
__USING Variant::GetError;
```

Description

This is Variant::GetError, a member of class OleVariant.

35.1.10.4.53 OleVariant::Variant::IDispatch* Data Member

Pascal

```
Variant::IDispatch*: __USING;
```

C++

```
__USING Variant::IDispatch*;
```

Description

This is Variant::IDispatch*, a member of class OleVariant.

35.1.10.4.54 OleVariant::Variant::IInterface* Data Member

Pascal

```
Variant::IInterface*: __USING;
```

C++

```
__USING Variant::IInterface*;
```

Description

This is Variant::IInterface*, a member of class OleVariant.

35.1.10.4.55 OleVariant::Variant::int Data Member

Pascal

```
Variant::int: __USING;
```

C++

```
__USING Variant::int;
```

Description

This is Variant::int, a member of class OleVariant.

35.1.10.4.56 OleVariant::Variant::int* Data Member

Pascal

```
Variant::int*: __USING;
```

C++

```
__USING Variant::int*;
```

Description

This is Variant::int*, a member of class OleVariant.

35.1.10.4.57 OleVariant::Variant::IsArray Data Member

Pascal

```
Variant::IsArray: __USING;
```

C++

```
__USING Variant::IsArray;
```

Description

This is Variant::IsArray, a member of class OleVariant.

35.1.10.4.58 OleVariant::Variant::IsEmpty Data Member

Pascal

```
Variant::IsEmpty: __USING;
```

C++

```
__USING Variant::IsEmpty;
```

Description

This is Variant::IsEmpty, a member of class OleVariant.

35.1.10.4.59 OleVariant::Variant::IsNull Data Member

Pascal

```
Variant::IsNull: __USING;
```

C++

```
__USING Variant::IsNull;
```

Description

This is Variant::IsNull, a member of class OleVariant.

35.1.10.4.60 OleVariant::Variant::IUnknown* Data Member

Pascal

```
Variant::IUnknown*: __USING;
```

C++

```
__USING Variant::IUnknown*;
```

Description

This is Variant::IUnknown*, a member of class OleVariant.

35.1.10.4.61 OleVariant::Variant::long Data Member

Pascal

```
Variant::long: __USING;
```

C++

```
__USING Variant::long;
```

Description

This is Variant::long, a member of class OleVariant.

35.1.10.4.62 OleVariant::Variant::long* Data Member

Pascal

```
Variant::long*: __USING;
```

C++

```
__USING Variant::long*;
```

Description

This is Variant::long*, a member of class OleVariant.

35.1.10.4.63 OleVariant::Variant::PutElement Data Member

Pascal

```
Variant::PutElement: __USING;
```

C++

```
__USING Variant::PutElement;
```

Description

This is Variant::PutElement, a member of class OleVariant.

35.1.10.4.64 OleVariant::Variant::SetError Data Member

Pascal

```
Variant::SetError: __USING;
```

C++

```
__USING Variant::SetError;
```

Description

This is Variant::SetError, a member of class OleVariant.

35.1.10.4.65 OleVariant::Variant::short Data Member

Pascal

```
Variant::short: __USING;
```

C++

```
__USING Variant::short;
```

Description

This is Variant::short, a member of class OleVariant.

35.1.10.4.66 OleVariant::Variant::short* Data Member

Pascal

```
Variant::short*: __USING;
```

C++

```
__USING Variant::short*;
```

Description

This is Variant::short*, a member of class OleVariant.

35.1.10.4.67 OleVariant::Variant::signed char Data Member

Pascal

```
Variant::signed char: __USING;
```

C++

```
__USING Variant::signed char;
```

Description

This is Variant::signed char, a member of class OleVariant.

35.1.10.4.68 OleVariant::Variant::signed char* Data Member

Pascal

```
Variant::signed char*: __USING;
```

C++

```
__USING Variant::signed char*;
```

Description

This is Variant::signed char*, a member of class OleVariant.

35.1.10.4.69 OleVariant::Variant::TDateTime Data Member

Pascal

```
Variant::TDateTime: __USING;
```

C++

```
__USING Variant::TDateTime;
```

Description

This is Variant::TDateTime, a member of class OleVariant.

35.1.10.4.70 OleVariant::Variant::TDateTime* Data Member

Pascal

```
Variant::TDateTime*: __USING;
```

C++

```
__USING Variant::TDateTime*;
```

Description

This is Variant::TDateTime*, a member of class OleVariant.

35.1.10.4.71 OleVariant::Variant::Type Data Member

Pascal

```
Variant::Type: __USING;
```

C++

```
__USING Variant::Type;
```

Description

This is Variant::Type, a member of class OleVariant.

35.1.10.4.72 OleVariant::Variant::UnicodeString Data Member

Pascal

```
Variant::UnicodeString: __USING;
```

C++

```
__USING Variant::UnicodeString;
```

Description

This is Variant::UnicodeString, a member of class OleVariant.

35.1.10.4.73 OleVariant::Variant::unsigned char Data Member

Pascal

```
Variant::unsigned char: __USING;
```

C++

```
__USING Variant::unsigned char;
```

Description

This is Variant::unsigned char, a member of class OleVariant.

35.1.10.4.74 OleVariant::Variant::unsigned int Data Member

Pascal

```
Variant::unsigned int: __USING;
```

C++

```
__USING Variant::unsigned int;
```

Description

This is Variant::unsigned int, a member of class OleVariant.

35.1.10.4.75 OleVariant::Variant::unsigned int* Data Member

Pascal

```
Variant::unsigned int*: __USING;
```

C++

```
__USING Variant::unsigned int*;
```

Description

This is Variant::unsigned int*, a member of class OleVariant.

35.1.10.4.76 OleVariant::Variant::unsigned long Data Member

Pascal

```
Variant::unsigned long: __USING;
```

C++

```
__USING Variant::unsigned long;
```

Description

This is Variant::unsigned long, a member of class OleVariant.

35.1.10.4.77 OleVariant::Variant::unsigned short Data Member

Pascal

```
Variant::unsigned short: __USING;
```

C++

```
__USING Variant::unsigned short;
```

Description

This is Variant::unsigned short, a member of class OleVariant.

35.1.10.4.78 OleVariant::Variant::unsigned short* Data Member

Pascal

```
Variant::unsigned short*: __USING;
```

C++

```
__USING Variant::unsigned short*;
```

Description

This is Variant::unsigned short*, a member of class OleVariant.

35.1.10.4.79 OleVariant::Variant::WideString Data Member

Pascal

```
Variant::WideString: __USING;
```

C++

```
__USING Variant::WideString;
```

Description

This is Variant::WideString, a member of class OleVariant.

35.1.10.5 OleVariant Operators

The operators of the OleVariant class are listed here.

Public Operators

	Name	Description
	!= (see page 3301)	This is the overview for the != operator overload.
	< (see page 3302)	This is the overview for the < operator overload.
	<= (see page 3303)	This is the overview for the <= operator overload.
	= (see page 3303)	This is the overview for the = operator overload.
	== (see page 3305)	This is the overview for the == operator overload.
	> (see page 3306)	This is the overview for the > operator overload.
	>= (see page 3306)	This is the overview for the >= operator overload.

35.1.10.5.1 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	OleVariant::!= (char*) (see page 3302)	This is !=, a member of class OleVariant.
	OleVariant::!= (OleVariant&) (see page 3302)	This is !=, a member of class OleVariant.

35.1.10.5.1.1 OleVariant::!= Operator (OleVariant&)

Pascal

```
operator !=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator !=(const OleVariant& rhs) const;
```

Description

This is !=, a member of class OleVariant.

35.1.10.5.1.2 OleVariant::!= Operator (char*)

Pascal

```
operator !=(const rhs: char*);
```

C++

```
__fastcall bool operator !=(const char* rhs) const;
```

Description

This is !=, a member of class OleVariant.

35.1.10.5.2 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	OleVariant::< (char*) (see page 3303)	This is <, a member of class OleVariant.
	OleVariant::< (OleVariant&) (see page 3302)	This is <, a member of class OleVariant.

35.1.10.5.2.1 OleVariant::< Operator (OleVariant&)

Pascal

```
operator <(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator <(const OleVariant& rhs) const;
```

Description

This is <, a member of class OleVariant.

35.1.10.5.2.2 OleVariant::< Operator (char*)

Pascal

```
operator <(const rhs: char*);
```

C++

```
__fastcall bool operator <(const char* rhs) const;
```

Description

This is <, a member of class OleVariant.

35.1.10.5.3 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	OleVariant::<= (char*) (see page 3303)	This is <=, a member of class OleVariant.
	OleVariant::<= (OleVariant&) (see page 3303)	This is <=, a member of class OleVariant.

35.1.10.5.3.1 OleVariant::<= Operator (OleVariant&)

Pascal

```
operator <=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator <=(const OleVariant& rhs) const;
```

Description

This is <=, a member of class OleVariant.

35.1.10.5.3.2 OleVariant::<= Operator (char*)

Pascal

```
operator <=(const rhs: char*);
```

C++

```
__fastcall bool operator <=(const char* rhs) const;
```

Description

This is <=, a member of class OleVariant.

35.1.10.5.4 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	OleVariant::= (__int64) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (AnsiString&) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).

	OleVariant::= (bool) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (char) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (char*) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (Currency&) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (Currency*) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (double) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (double*) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (float) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (IDispatch*) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (int) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (IUnknown*) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (long) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (OleVariant&) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (short) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (signed char) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (TDateTime&) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (TDateTime*) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (UnicodeString&) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (unsigned char) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (unsigned int) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (unsigned long) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (unsigned short) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (Variant&) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).
	OleVariant::= (WideString&) (see page 3304)	Copies a value into the System::OleVariant (see page 3274).

35.1.10.5.4.1 OleVariant::= Operator (AnsiString&)

Copies a value into the System::OleVariant (see page 3274).

Pascal

```
operator =(const rhs: OleVariant&);
operator =(const rhs: Variant&);
operator =(const rhs: AnsiString&);
operator =(const rhs: char*);
operator =(const rhs: WideString&);
operator =(const rhs: UnicodeString&);
operator =(const rhs: bool);
operator =(const rhs: char);
operator =(const rhs: signed char);
operator =(const rhs: unsigned char);
operator =(const rhs: short);
operator =(const rhs: unsigned short);
operator =(const rhs: Integer);
operator =(const rhs: unsigned int);
operator =(const rhs: long);
operator =(const rhs: unsigned long);
```

```
operator =(const rhs: float);
operator =(const rhs: double);
operator =(const rhs: __int64);
operator =(const rhs: TDateTime&);
operator =(const rhs: Currency&);
operator =(var rhs: double);
operator =(var rhs: TDateTime);
operator =(var rhs: Currency);
operator =(var rhs: IDispatch);
operator =(var rhs: IUnknown);
```

C++

```
__fastcall OleVariant& operator =(const OleVariant& rhs);
__fastcall OleVariant& operator =(const Variant& rhs);
__fastcall OleVariant& operator =(const AnsiString& rhs);
__fastcall OleVariant& operator =(const char* rhs);
__fastcall OleVariant& operator =(const WideString& rhs);
__fastcall OleVariant& operator =(const UnicodeString& rhs);
__fastcall OleVariant& operator =(const bool rhs);
__fastcall OleVariant& operator =(const char rhs);
__fastcall OleVariant& operator =(const signed char rhs);
__fastcall OleVariant& operator =(const unsigned char rhs);
__fastcall OleVariant& operator =(const short rhs);
__fastcall OleVariant& operator =(const unsigned short rhs);
__fastcall OleVariant& operator =(const int rhs);
__fastcall OleVariant& operator =(const unsigned int rhs);
__fastcall OleVariant& operator =(const long rhs);
__fastcall OleVariant& operator =(const unsigned long rhs);
__fastcall OleVariant& operator =(const float rhs);
__fastcall OleVariant& operator =(const double rhs);
__fastcall OleVariant& operator =(const __int64 rhs);
__fastcall OleVariant& operator =(const TDateTime& rhs);
__fastcall OleVariant& operator =(const Currency& rhs);
__fastcall OleVariant& operator =(double* rhs);
__fastcall OleVariant& operator =(TDateTime* rhs);
__fastcall OleVariant& operator =(Currency* rhs);
__fastcall OleVariant& operator =(IDispatch* rhs);
__fastcall OleVariant& operator =(IUnknown* rhs);
```

Description

The assignment operator converts the value of rhs, as appropriate, and assigns it to the System::OleVariant (see page 3274). If it is not possible to perform the conversion, an EVariantError exception is thrown.

35.1.10.5.5 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	OleVariant::== (char*) (see page 3306)	This is ==, a member of class OleVariant.
	OleVariant::== (OleVariant&) (see page 3305)	This is ==, a member of class OleVariant.

35.1.10.5.5.1 OleVariant::== Operator (OleVariant&)

Pascal

```
operator ==(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator ==(const OleVariant& rhs) const;
```

Description

This is ==, a member of class OleVariant.

35.1.10.5.5.2 OleVariant::== Operator (char*)

Pascal

```
operator ==(const rhs: char*);
```

C++

```
__fastcall bool operator ==(const char* rhs) const;
```

Description

This is ==, a member of class OleVariant.

35.1.10.5.6 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	OleVariant::> (char*) (see page 3306)	This is >, a member of class OleVariant.
	OleVariant::> (OleVariant&) (see page 3306)	This is >, a member of class OleVariant.

35.1.10.5.6.1 OleVariant::> Operator (OleVariant&)

Pascal

```
operator >(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator >(const OleVariant& rhs) const;
```

Description

This is >, a member of class OleVariant.

35.1.10.5.6.2 OleVariant::> Operator (char*)

Pascal

```
operator >(const rhs: char*);
```

C++

```
__fastcall bool operator >(const char* rhs) const;
```

Description

This is >, a member of class OleVariant.

35.1.10.5.7 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	OleVariant::>= (char*) (see page 3307)	This is >=, a member of class OleVariant.
	OleVariant::>= (OleVariant&) (see page 3307)	This is >=, a member of class OleVariant.

35.1.10.5.7.1 OleVariant::>= Operator (OleVariant&)

Pascal

```
operator >=(const rhs: OleVariant&);
```

C++

```
__fastcall bool operator >=(const OleVariant& rhs) const;
```

Description

This is >=, a member of class OleVariant.

35.1.10.5.7.2 OleVariant::>= Operator (char*)

Pascal

```
operator >=(const rhs: char*);
```

C++

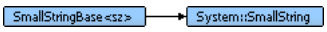
```
__fastcall bool operator >=(const char* rhs) const;
```

Description

This is >=, a member of class OleVariant.

35.1.11 SmallString Template

Class Hierarchy



Pascal

```
SmallString<unsigned char sz> = class(SmallStringBase<sz>);
```

C++

```
template <unsigned char sz>
class SmallString : private SmallStringBase<sz>;
```

File

dstring.h

Description

This is class System::SmallString.

35.1.11.1 SmallString Members

The following tables list the members exposed by SmallString.

Public Methods

	Name	Description
	SmallString (see page 3308)	This is the overview for the SmallString constructor overload.

Public Operators

	Name	Description
	[] (see page 3309)	This is [], a member of class SmallString.
	= (see page 3309)	This is =, a member of class SmallString.
	AnsiString (see page 3310)	This is AnsiString, a member of class SmallString.

35.1.11.2 SmallString Constructor

This is the overview for the SmallString constructor overload.

Overload List

	Name	Description
	SmallString::SmallString () (see page 3308)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (AnsiString&) (see page 3308)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (char*) (see page 3309)	This is SmallString, a member of class SmallString.
	SmallString::SmallString (SmallString&) (see page 3309)	This is SmallString, a member of class SmallString.

35.1.11.2.1 SmallString::SmallString Constructor ()

Pascal

```
constructor Create;
```

C++

```
__fastcall SmallString();
```

Description

This is SmallString, a member of class SmallString.

35.1.11.2.2 SmallString::SmallString Constructor (AnsiString&)

Pascal

```
constructor Create(const src: AnsiString&);
```

C++

```
__fastcall SmallString(const AnsiString& src);
```

Description

This is SmallString, a member of class SmallString.

35.1.11.2.3 SmallString::SmallString Constructor (SmallString&)

Pascal

```
constructor Create(const src: SmallString&);
```

C++

```
__fastcall SmallString(const SmallString& src);
```

Description

This is SmallString, a member of class SmallString.

35.1.11.2.4 SmallString::SmallString Constructor (char*)

Pascal

```
constructor Create(const src: char*);
```

C++

```
__fastcall SmallString(const char* src);
```

Description

This is SmallString, a member of class SmallString.

35.1.11.3 SmallString Operators

The operators of the SmallString class are listed here.

Public Operators

	Name	Description
	[] (see page 3309)	This is [], a member of class SmallString.
	= (see page 3309)	This is =, a member of class SmallString.
	AnsiString (see page 3310)	This is AnsiString, a member of class SmallString.

35.1.11.3.1 SmallString::[] Operator

Pascal

```
operator [](const idx: unsigned char);
```

C++

```
__fastcall char& operator [](const unsigned char idx);
```

Description

This is [], a member of class SmallString.

35.1.11.3.2 SmallString::= Operator

Pascal

```
operator =(const rhs: SmallString&);
```

C++

```
__fastcall SmallString& operator =(const SmallString& rhs);
```

Description

This is =, a member of class SmallString.

35.1.11.3.3 SmallString::AnsiString Operator

Pascal

```
operator AnsiString;
```

C++

```
__fastcall operator AnsiString() const;
```

Description

This is AnsiString, a member of class SmallString.

35.1.12 SmallStringBase Template

Class Hierarchy

Pascal

```
SmallStringBase<unsigned char sz> = class;
```

C++

```
template <unsigned char sz>
class SmallStringBase;
```

File

dstring.h

Description

This is class System::SmallStringBase.

35.1.12.1 SmallStringBase Members

The following tables list the members exposed by SmallStringBase.

Protected Data Members

	Name	Description
	Data (see page 3311)	This is Data, a member of class SmallStringBase.
	Len (see page 3311)	This is Len, a member of class SmallStringBase.

35.1.12.2 SmallStringBase Data Members

The data members of the SmallStringBase class are listed here.

Protected Data Members

	Name	Description
	Data (see page 3311)	This is Data, a member of class SmallStringBase.
	Len (see page 3311)	This is Len, a member of class SmallStringBase.

35.1.12.2.1 SmallStringBase::Data Data Member

Pascal

```
[sz] Data: char;
```

C++

```
char Data[sz];
```

Description

This is Data, a member of class SmallStringBase.

35.1.12.2.2 SmallStringBase::Len Data Member

Pascal

```
Len: unsigned char;
```

C++

```
unsigned char Len;
```

Description

This is Len, a member of class SmallStringBase.

35.1.13 sPointer Template

Class Hierarchy

System::sPointer

Pascal

```
sPointer<typename T> = class;
```

C++

```
template <typename T>
class sPointer;
```

File

sysdyn.h

Description

This is class System::sPointer.

35.1.13.1 sPointer Members

The following tables list the members exposed by sPointer.

Public Methods

	Name	Description
	~sPointer (see page 3312)	This is ~sPointer, a member of class sPointer.
	sPointer (see page 3312)	This is the overview for the sPointer constructor overload.

Public Operators

	Name	Description
	& (see page 3313)	This is &, a member of class sPointer.
	-> (see page 3313)	This is ->, a member of class sPointer.

35.1.13.2 sPointer::~sPointer Destructor

Pascal

```
destructor Destroy;
```

C++

```
~sPointer();
```

Description

This is ~sPointer, a member of class sPointer.

35.1.13.3 sPointer Constructor

This is the overview for the sPointer constructor overload.

Overload List

	Name	Description
	sPointer::sPointer () (see page 3312)	This is sPointer, a member of class sPointer.
	sPointer::sPointer (T*) (see page 3312)	This is sPointer, a member of class sPointer.

35.1.13.3.1 sPointer::sPointer Constructor ()

Pascal

```
constructor Create;
```

C++

```
sPointer();
```

Description

This is sPointer, a member of class sPointer.

35.1.13.3.2 sPointer::sPointer Constructor (T*)

Pascal

```
constructor Create(var p: T);
```

C++

```
sPointer(T* p);
```

Description

This is sPointer, a member of class sPointer.

35.1.13.4 sPointer Operators

The operators of the sPointer class are listed here.

Public Operators

	Name	Description
	& (see page 3313)	This is &, a member of class sPointer.
	-> (see page 3313)	This is ->, a member of class sPointer.

35.1.13.4.1 sPointer::& Operator

Pascal

```
operator &;
```

C++

```
T* operator &();
```

Description

This is &, a member of class sPointer.

35.1.13.4.2 sPointer::-> Operator

Pascal

```
operator ->;
```

C++

```
T* operator ->();
```

Description

This is ->, a member of class sPointer.

35.1.14 TAggregatedObject Class

System::TAggregatedObject provides the functionality for an inner object of an aggregate by implementing the IInterface (see page 3502) methods to delegate to the controlling IInterface (see page 3502).

Class Hierarchy

Pascal

```
TAggregatedObject = class(TObject);
```

C++

```
class TAggregatedObject : public TObject;
```

File

System.pas

Description

An aggregated object is an object composed of several interfaced objects. Each object implements its own behavior and interfaces, but all of the objects share the same reference count, which is that of the controller object. In the container pattern, the controller is the container object.

System::TAggregatedObject does not itself support any interfaces. However, as is typical of an aggregate, it does implement the methods of IInterface (see page 3502), which are used by the objects that descend from it. System::TAggregatedObject, therefore, serves as a base for classes that implement interfaces for creating objects that are part of an aggregate.

System::TAggregatedObject is used as a base for classes that create contained objects and connecting objects. Using System::TAggregatedObject ensures that calls to the IInterface (see page 3502) methods delegate to the controlling IInterface (see page 3502) of the aggregate.

The controlling IInterface (see page 3502) is specified in the constructor for System::TAggregatedObject and is indicated by the System::TAggregatedObject::Controller (see page 3317) property.

See Also

IInterface (see page 3502)

TConnectionPoint

TConnectionPoints

35.1.14.1 TAggregatedObject Members

The following tables list the members exposed by TAggregatedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3316)	Implements the IInterface (see page 3502) System::TAggregatedObject::_AddRef method.
	_Release (see page 3316)	Implements the IInterface (see page 3502) System::TAggregatedObject::_Release method.
	QueryInterface (see page 3317)	Implements the IInterface (see page 3502) System::TAggregatedObject::QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3315)	Instantiates an aggregated object.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Controller (see page 3317)	Provides access to the controlling interface of the aggregate.

35.1.14.2 TAggregatedObject::Create Constructor

Instantiates an aggregated object.

Pascal

```
constructor Create(const Controller: IInterface);
```

C++

```
Create(const IInterface Controller);
```

Description

Call System::TAggregatedObject::Create to instantiate an aggregated object at runtime.

System::TAggregatedObject::Create allocates memory for an aggregated object and sets the System::TAggregatedObject::Controller (see page 3317) property to the System::TAggregatedObject::Controller (see page 3317) parameter passed to it. The System::TAggregatedObject::Controller (see page 3317) is the controlling interface of the aggregate. All other interfaces implemented in the aggregate delegate IInterface (see page 3502) method calls to this System::TAggregatedObject::Controller (see page 3317).

See Also

System::TAggregatedObject::Controller (see page 3317)

35.1.14.3 TAggregatedObject Methods

The methods of the TAggregatedObject class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 3316)	Implements the IInterface (see page 3502) System::TAggregatedObject::_AddRef method.
	_Release (see page 3316)	Implements the IInterface (see page 3502) System::TAggregatedObject::_Release method.
	QueryInterface (see page 3317)	Implements the IInterface (see page 3502) System::TAggregatedObject::QueryInterface method.

35.1.14.3.1 TAggregatedObject::_AddRef Method

Implements the IInterface (see page 3502) System::TAggregatedObject::_AddRef method.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__stdcall Integer _AddRef();
```

Description

Do not call the protected System::TAggregatedObject::_AddRef method directly. System::TAggregatedObject::_AddRef is called through the interface to indicate that another object is using the interfaced object's interface pointer. When the other object is through with the interface, it calls System::TInterfacedObject::_Release (see page 3345) through the same interface. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

See Also

System::TInterfacedObject::RefCount (see page 3347)

System::TAggregatedObject::_Release (see page 3316)

System::TInterfacedObject::_Release (see page 3345)

35.1.14.3.2 TAggregatedObject::_Release Method

Implements the IInterface (see page 3502) System::TAggregatedObject::_Release method.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__stdcall Integer _Release();
```

Description

Do not call the protected _System::TAggregatedObject::_Release method directly. _System::TAggregatedObject::_Release is called through the IInterface (see page 3502) interface to indicate that another object is through with the interfaced object's interface pointer. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

See Also

System::TInterfacedObject::RefCount (see page 3347)

System::TInterfacedObject::_AddRef (see page 3345)

System::TAggregatedObject::_AddRef (see page 3316)

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TObject::BeforeDestruction (see page 3358)

35.1.14.3.3 TAggregatedObject::QueryInterface Method

Implements the IInterface (see page 3502) System::TAggregatedObject::QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

Do not call the protected System::TAggregatedObject::QueryInterface method directly. System::TAggregatedObject::QueryInterface is called through IInterface (see page 3502) to obtain an interface pointer for the interface identified by the IID parameter. If the object supports the requested interface, it is returned in the Obj parameter and System::TAggregatedObject::QueryInterface returns S_OK (see page 3860). If the object does not support the interface, System::TAggregatedObject::QueryInterface returns E_NOINTERFACE (see page 3845).

If System::TAggregatedObject::QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

System::TAggregatedObject::_AddRef (see page 3316)

System::TInterfacedObject::_AddRef (see page 3345)

35.1.14.4 TAggregatedObject Properties

The properties of the TAggregatedObject class are listed here.

Public Properties

	Name	Description
	Controller (see page 3317)	Provides access to the controlling interface of the aggregate.

35.1.14.4.1 TAggregatedObject::Controller Property

Provides access to the controlling interface of the aggregate.

Pascal

```
property Controller: IInterface;
```

C++

```
__property IInterface Controller;
```

Description

System::TAggregatedObject::Controller is the interface reference of the aggregate. System::TAggregatedObject (see page 3313) is used as part of an aggregate that has a single controlling IInterface (see page 3502). The IInterface (see page 3502) methods that are implemented in aggregated objects are delegated to the System::TAggregatedObject::Controller.

When the aggregated object is created, its constructor is passed a controlling IInterface (see page 3502). System::TAggregatedObject::Controller is initialized to this IInterface (see page 3502) in the constructor.

See Also

System::TAggregatedObject::Create (see page 3315)

35.1.15 TClassHelperBase Class

Class Hierarchy

Pascal

```
TClassHelperBase = class(TInterfacedObject, IInterface);
```

C++

```
class TClassHelperBase : public TInterfacedObject, public IInterface;
```

File

System

Description

This is class System.TClassHelperBase.

35.1.15.1 TClassHelperBase Members

The following tables list the members exposed by TClassHelperBase.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Fields

	Name	Description
	FInstance (see page 3320)	This is FInstance, a member of class TClassHelperBase.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Create (see page 3319)	This is _Create, a member of class TClassHelperBase.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

35.1.15.2 TClassHelperBase._Create Constructor

Pascal

```
constructor _Create(Instance: TObject);
```

C++

```
__fastcall TClassHelperBase(TObject * Instance);
```

Description

This is _Create, a member of class TClassHelperBase.

35.1.15.3 TClassHelperBase Fields

The fields of the TClassHelperBase class are listed here.

Protected Fields

	Name	Description
	FInstance (see page 3320)	This is FInstance, a member of class TClassHelperBase.

35.1.15.3.1 TClassHelperBase.FInstance Field

Pascal

```
FInstance: TObject;
```

C++

```
TObject * FInstance;
```

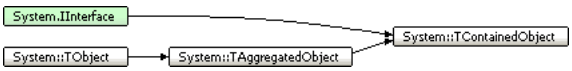
Description

This is FInstance, a member of class TClassHelperBase.

35.1.16 TContainedObject Class

System::TContainedObject implements the IInterface (see page 3502) interface as an inner object in an aggregate.

Class Hierarchy



Pascal

```
TContainedObject = class(TAggregatedObject, IInterface);
```

C++

```
class TContainedObject : public TAggregatedObject, public IInterface;
```

File

System.pas

Description

System::TContainedObject can be used as a base for classes that create inner, or contained, objects of an aggregate. In Delphi, use TComObject or its descendants to instantiate an outer object that supports aggregation.

- The following implementation rules apply to aggregatable (inner or contained) objects, and are supported by System::TContainedObject:
- When an object of its type is created, the creation succeeds only for a requested interface of type IInterface (see page 3502).
- The IInterface (see page 3502) methods _AddRef (see page 3316) and _Release (see page 3316) delegate to the controlling IInterface (see page 3502).
- QueryInterface (see page 3322) is implemented in System::TContainedObject only for interfaces implemented in the contained object.
- The reference count on the inner object is incremented in the containing object.

Note: In Delphi, System::TContainedObject is used as a base class for a connection point that must be an inner object in an aggregate.

Note: An aggregated object is an object that is composed of several interfaced objects. Each object implements its own behavior and interfaces, but all of the objects share the same reference count, which is that of the controller object. In the container pattern, the controller is the container object.

See Also

IInterface (see page 3502)

TComObject

TConnectionPoint

Automation Servers

35.1.16.1 TContainedObject Members

The following tables list the members exposed by TContainedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3316)	Implements the IInterface (see page 3502) System::TAggregatedObject::_AddRef method.
	_Release (see page 3316)	Implements the IInterface (see page 3502) System::TAggregatedObject::_Release method.
V	QueryInterface (see page 3322)	Implements the IInterface (see page 3502) System::TContainedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3315)	Instantiates an aggregated object.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.

	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Controller (see page 3317)	Provides access to the controlling interface of the aggregate.

35.1.16.2 TContainedObject Methods

The methods of the TContainedObject class are listed here.

Protected Methods

	Name	Description
V	QueryInterface (see page 3322)	Implements the IInterface (see page 3502) System::TContainedObject::QueryInterface method.

35.1.16.2.1 TContainedObject::QueryInterface Method

Implements the IInterface (see page 3502) System::TContainedObject::QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

Do not call the protected System::TContainedObject::QueryInterface method directly. System::TContainedObject::QueryInterface is called through IInterface (see page 3502) to obtain an interface pointer for the interface identified by the IID parameter. If the object supports the requested interface, it is returned in the Obj parameter and System::TInterfacedObject::QueryInterface (see page 3347) returns S_OK (see page 3860). If the object does not support the interface, System::TContainedObject::QueryInterface returns E_NOINTERFACE (see page 3845).

If System::TContainedObject::QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

System::TInterfacedObject::QueryInterface (see page 3347)

System::TAggregatedObject::QueryInterface (see page 3317)

QueryInterface (see page 3504)

35.1.17 TDateTime Class

System::TDateTime is the C++ analog for the Delphi System::TDateTime data type.

Class Hierarchy

Pascal

```
TDateTime = class(TDateTimeBase);
```

C++

```
class TDateTime : public TDateTimeBase;
```

File

systdate.h

Description

System::TDateTime implements the Delphi System::TDateTime data type and the date/time runtime library routines that use the System::TDateTime data type.

The System::TDateTime class inherits a val data member declared as a double (see page 3340) that holds the date-time value. The integral part of a System::TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a System::TDateTime value is the time of day.

Following are some examples of System::TDateTime values and their corresponding dates and times:

0 12/30/1899 12:00 am

2.75 1/1/1900 6:00 pm

-1.25 12/29/1899 6:00 am

35065 1/1/1996 12:00 am

To find the fractional number of days between two dates, simply subtract the two values, unless one of the System::TDateTime values is negative. Similarly, to increment a date and time value by a certain fractional number of days, add the fractional number to the date and time value if the System::TDateTime value is positive.

When working with negative System::TDateTime values, computations must handle time portion separately. The fractional part reflects the fraction of a 24-hour day without regard to the sign of the System::TDateTime value. For example, 6:00 am on 12/29/1899 is –1.25, not –1 + 0.25, which would be –0.75. There are no System::TDateTime values between –1 and 0.

Note: Use only the operators declared by System::TDateTime. The compiler ignores any operators you overload yourself.

Warning: It is not possible to use System::TDateTime as a base class for inheriting. To add functionality to System::TDateTime, you must write a wrapper class.

Note: To utilize the C++ streaming operators (<< and >>) with Currency (see page 3235), you must use #include <iostream> before #include <systdate.h>. This occurs automatically if you include vcl.h, clx.h, or System.hpp. The following stream operators are defined:

ostream& operator << (ostream& os, const System::TDateTime& arg);

istream& operator >> (istream& is, System::TDateTime& arg);

35.1.17.1 TDateTime Members

The following tables list the members exposed by TDateTime.

Enumerations

	Name	Description
	TDateTimeFlag (see page 3327)	This is record System::TDateTime::TDateTimeFlag.

Public Data Members

	Name	Description
	Val (see page 3342)	This is Val, a member of class TDateTimeBase.

Public Methods

	Name	Description
	CurrentDate (see page 3328)	Returns the current date as a System::TDateTime::TDateTime (see page 3323) value.
	CurrentDateTime (see page 3328)	Returns the current date and time as a System::TDateTime::TDateTime (see page 3323) value.
	CurrentTime (see page 3328)	Returns the current time as a System::TDateTime::TDateTime (see page 3323) value.
	DateString (see page 3329)	Converts the date of the System::TDateTime::TDateTime (see page 3323) value to a string.
	DateTimeString (see page 3329)	Converts the System::TDateTime::TDateTime (see page 3323) value to a string.
	DayOfWeek (see page 3329)	Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 3323)
	DecodeDate (see page 3330)	Separates the System::TDateTime::TDateTime (see page 3323) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.
	DecodeTime (see page 3330)	Separates the System::TDateTime::TDateTime (see page 3323) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.
	FileDate (see page 3330)	Converts the date/time value to an OS date/time stamp.
	FileDateToDateTime (see page 3331)	Converts an OS timestamp value to System::TDateTime::TDateTime (see page 3323) value.
	FormatString (see page 3331)	Formats the System::TDateTime::TDateTime (see page 3323) object using the format given by format.
	TDateTime (see page 3325)	This is the overview for the TDateTime constructor overload.
	TimeString (see page 3333)	Converts the System::TDateTime::TDateTime (see page 3323) object to an AnsiString.

Public Operators

	Name	Description
	- (see page 3334)	This is the overview for the - operator overload.
	-- (see page 3334)	This is the overview for the -- operator overload.
	!= (see page 3335)	Compares two System::TDateTime::TDateTime (see page 3323) objects for inequality.
	+ (see page 3335)	This is the overview for the + operator overload.
	++ (see page 3336)	This is the overview for the ++ operator overload.
	+= (see page 3336)	This is the overview for the += operator overload.
	< (see page 3337)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is less than another specified System::TDateTime::TDateTime (see page 3323) value.
	<= (see page 3337)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is less than or equal to another specified System::TDateTime::TDateTime (see page 3323) value.
	= (see page 3338)	This is the overview for the = operator overload.

	-= (see page 3338)	This is the overview for the -= operator overload.
	== (see page 3339)	Compares two System::TDateTime::TDateTime (see page 3323) objects for equality.
	> (see page 3339)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is greater than another specified System::TDateTime::TDateTime (see page 3323) value.
	>= (see page 3340)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is greater than or equal to another specified System::TDateTime::TDateTime (see page 3323) value.
	double (see page 3340)	Converts the System::TDateTime::TDateTime (see page 3323) object to a double value.
	int (see page 3340)	Converts the System::TDateTime::TDateTime (see page 3323) object to an integer value.
	System::String (see page 3341)	Converts the System::TDateTime::TDateTime (see page 3323) object to a String.

35.1.17.2 TDateTime Constructor

This is the overview for the TDateTime constructor overload.

Overload List

	Name	Description
	TDateTime::TDateTime () (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (double) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (int) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (System::String&, TDateTimeFlag) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (TDateTime&) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (TDateTimeBase&) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short, unsigned short) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).
	TDateTime::TDateTime (unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short, unsigned short) (see page 3325)	Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).

35.1.17.2.1 TDateTime::TDateTime Constructor ()

Creates and initializes an instance of System::TDateTime::TDateTime (see page 3323).

Pascal

```
constructor Create;
constructor Create(const src: TDateTimeBase&);
```

```
constructor Create(const src: TDateTime&);
constructor Create(const src: double);
constructor Create(const src: Integer);
constructor Create(const src: System::String&; flag: TDateTimeFlag = DateTime);
constructor Create(year: unsigned short; month: unsigned short; day: unsigned short);
constructor Create(hour: unsigned short; min: unsigned short; sec: unsigned short; msec:
unsigned short);
constructor Create(year: unsigned short; month: unsigned short; day: unsigned short; hour:
unsigned short; min: unsigned short; sec: unsigned short; msec: unsigned short);
```

C++

```
__fastcall TDateTime();
__fastcall TDateTime(const TDateTimeBase& src);
__fastcall TDateTime(const TDateTime& src);
__fastcall TDateTime(const double src);
__fastcall TDateTime(const int src);
__fastcall TDateTime(const System::String& src, TDateTimeFlag flag = DateTime);
__fastcall TDateTime(unsigned short year, unsigned short month, unsigned short day);
__fastcall TDateTime(unsigned short hour, unsigned short min, unsigned short sec, unsigned
short msec);
__fastcall TDateTime(unsigned short year, unsigned short month, unsigned short day, unsigned
short hour, unsigned short min, unsigned short sec, unsigned short msec);
```

Description

Use System::TDateTime::TDateTime (see page 3323) to instantiate a System::TDateTime::TDateTime (see page 3323) object.

When called with no arguments, the resulting System::TDateTime::TDateTime (see page 3323) object has a Val (see page 3701) data member of zero. This date and time corresponds to 12/30/1899 12:00 am.

When called with a System::TDateTime::TDateTime (see page 3323) object as the src parameter, the resulting object is a copy of src (the copy constructor).

When called with a System::TDateTime::TDateTimeBase (see page 3341) object as the src parameter, the resulting object has the same value as the src parameter. System::TDateTime::TDateTimeBase (see page 3341) is the base class for the System::TDateTime::TDateTime (see page 3323) type. It has a single member, a public Val (see page 3701) field that represents the value of the object.

When called with a double (see page 3340) or integer as the src parameter, the resulting object has the same value as the src parameter. The integral part of src is the number of days that have passed since 12/30/1899. The fractional part is the time of day, represented as a fraction of 24 hours.

When called with an AnsiString as the src parameter and a System::TDateTime::TDateTimeFlag (see page 3327) as the flag parameter, the resulting object's value is converted from the AnsiString. The flag parameter indicates whether the string contains a date, a time, or both. The default for the flag is DateTime.

When called with three unsigned shorts (as the year, month, and day parameters), the resulting object's value is assembled from the specified year, month, and day, with a time portion of 0 (midnight). The year must be between 1 and 9999. Valid month values are 1 through 12. Valid day values are 1 through 28, 29, 30, or 31, depending on the month value. For example, the possible day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the year value specifies a leap year.

When called with four unsigned shorts (as the hour, min, sec, and msec parameters), the resulting object's value represents a time based on the specified number of hours, minutes, seconds, and milliseconds past midnight. The date portion of the resulting value is 0 (12/30/1899). The hour must be between 0 and 23, the minute must be between 0 and 59, the second must be between 0 and 59, and the millisecond must be between 0 and 999.

Note: When the constructor calls for a conversion that can't be performed (an AnsiString that does not represent a date or time, or invalid year, month, day, hour, min, sec, or msec value), an EConvertError exception is thrown.

35.1.17.3 TDateTime Enumerations

The enumerations of the TDateTime class are listed here.

Enumerations

	Name	Description
	TDateTimeFlag (see page 3327)	This is record System::TDateTime::TDateTimeFlag.

35.1.17.3.1 System::TDateTime::TDateTimeFlag Enumeration

Pascal

```
TDateTimeFlag = (
  Date,
  Time,
  DateTime
);
```

C++

```
enum TDateTimeFlag {
  Date,
  Time,
  DateTime
};
```

File

systdate.h

Description

This is record System::TDateTime::TDateTimeFlag.

35.1.17.4 TDateTime Methods

The methods of the TDateTime class are listed here.

Public Methods

	Name	Description
	CurrentDate (see page 3328)	Returns the current date as a System::TDateTime::TDateTime (see page 3323) value.
	CurrentDateTime (see page 3328)	Returns the current date and time as a System::TDateTime::TDateTime (see page 3323) value.
	CurrentTime (see page 3328)	Returns the current time as a System::TDateTime::TDateTime (see page 3323) value.
	DateString (see page 3329)	Converts the date of the System::TDateTime::TDateTime (see page 3323) value to a string.
	DateTimeString (see page 3329)	Converts the System::TDateTime::TDateTime (see page 3323) value to a string.
	DayOfWeek (see page 3329)	Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 3323)
	DecodeDate (see page 3330)	Separates the System::TDateTime::TDateTime (see page 3323) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.
	DecodeTime (see page 3330)	Separates the System::TDateTime::TDateTime (see page 3323) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.
	FileDate (see page 3330)	Converts the date/time value to an OS date/time stamp.

	FileDateToDateTime (see page 3331)	Converts an OS timestamp value to System::TDateTime::TDateTime (see page 3323) value.
	FormatString (see page 3331)	Formats the System::TDateTime::TDateTime (see page 3323) object using the format given by format.
	TimeString (see page 3333)	Converts the System::TDateTime::TDateTime (see page 3323) object to an AnsiString.

35.1.17.4.1 TDateTime::CurrentDate Method

Returns the current date as a System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
function CurrentDate: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentDate();
```

Description

System::TDateTime::CurrentDate retrieves the current date and returns it as a new System::TDateTime::TDateTime (see page 3323) object with a 0 time value.

See Also

System::TDateTime::CurrentDateTime (see page 3328)

System::TDateTime::CurrentTime (see page 3328)

35.1.17.4.2 TDateTime::CurrentDateTime Method

Returns the current date and time as a System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
function CurrentDateTime: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentDateTime();
```

Description

System::TDateTime::CurrentDateTime retrieves the current date and time, and returns it as a new System::TDateTime::TDateTime (see page 3323) object.

See Also

System::TDateTime::CurrentTime (see page 3328)

System::TDateTime::CurrentDate (see page 3328)

35.1.17.4.3 TDateTime::CurrentTime Method

Returns the current time as a System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
function CurrentTime: TDateTime; static;
```

C++

```
static __fastcall TDateTime CurrentTime();
```

Description

System::TDateTime::CurrentTime retrieves the current time and returns it as a new System::TDateTime::TDateTime (see page 3323) object with a 0 date value.

See Also

System::TDateTime::CurrentDateTime (see page 3328)

System::TDateTime::CurrentDate (see page 3328)

35.1.17.4.4 TDateTime::DateString Method

Converts the date of the System::TDateTime::TDateTime (see page 3323) value to a string.

Pascal

```
function DateString: System::String;
```

C++

```
__fastcall System::String DateString() const;
```

Description

System::TDateTime::DateString returns the value of the System::TDateTime::TDateTime (see page 3323) object as an AnsiString. The conversion uses the format specified by the ShortDateFormat global variable.

See Also

System::TDateTime::DateTimeString (see page 3329)

System::TDateTime::TimeString (see page 3333)

35.1.17.4.5 TDateTime::DateTimeString Method

Converts the System::TDateTime::TDateTime (see page 3323) value to a string.

Pascal

```
function DateTimeString: System::String;
```

C++

```
__fastcall System::String DateTimeString() const;
```

Description

System::TDateTime::DateTimeString returns an AnsiString object containing the date and time value (the Val (see page 3701) data member) of the System::TDateTime::TDateTime (see page 3323) object to a string. System::TDateTime::DateTimeString converts the System::TDateTime::TDateTime (see page 3323) value using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.

See Also

System::TDateTime::DateString (see page 3329)

System::TDateTime::TimeString (see page 3333)

35.1.17.4.6 TDateTime::DayOfWeek Method

Returns the day of the week of the date portion of the System::TDateTime::TDateTime (see page 3323)

Pascal

```
function DayOfWeek: Integer;
```

C++

```
__fastcall int DayOfWeek() const;
```

Description

System::TDateTime::DayOfWeek returns an integer between 1 and 7. Sunday is treated as the first day of the week and Saturday as the seventh.

35.1.17.4.7 TDateTime::DecodeDate Method

Separates the System::TDateTime::TDateTime (see page 3323) value into year, month, and day values and stores these values in the year, month, and day parameters, respectively.

Pascal

```
procedure DecodeDate(var year: unsigned short; var month: unsigned short; var day: unsigned
short);
```

C++

```
__fastcall void DecodeDate(unsigned short* year, unsigned short* month, unsigned short* day)
const;
```

Description

Use System::TDateTime::DecodeDate() when you need to access the year, month, or day of a System::TDateTime::TDateTime (see page 3323) object's date portion.

35.1.17.4.8 TDateTime::DecodeTime Method

Separates the System::TDateTime::TDateTime (see page 3323) value into hour, minute, second, and millisecond values and stores these values in the hour, min, sec, and msec parameters, respectively.

Pascal

```
procedure DecodeTime(var hour: unsigned short; var min: unsigned short; var sec: unsigned
short; var msec: unsigned short);
```

C++

```
__fastcall void DecodeTime(unsigned short* hour, unsigned short* min, unsigned short* sec,
unsigned short* msec) const;
```

Description

Use System::TDateTime::DecodeTime() to access the hour, minute, second, or millisecond values of the System::TDateTime::TDateTime (see page 3323) object's time portion.

35.1.17.4.9 TDateTime::FileDate Method

Converts the date/time value to an OS date/time stamp.

Pascal

```
function FileDate: Integer;
```

C++

```
__fastcall int FileDate() const;
```

Description

Use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 3323) value to an OS date/time stamp. For example, use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 3323) value to a format that can be used as the Age parameter of FileSetDate. You may also need to use System::TDateTime::FileDate to convert a System::TDateTime::TDateTime (see page 3323) so that it is compatible with the Time field of the TSearchRec structure used by the FindFirst and FindNext functions.

See Also

System::TDateTime::FileDateToDateTime (see page 3331)

FileSetDate (see page 4164)

TSearchRec (see page 4330)

35.1.17.4.10 TDateTime::FileDateToDateTime Method

Converts an OS timestamp value to System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
function FileDateToDateTime(fileDate: Integer): TDateTime; static;
```

C++

```
static __fastcall TDateTime FileDateToDateTime(int fileDate);
```

Description

A timestamp is a signed 32-bit integer, used by the OS to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use System::TDateTime::FileDateToDateTime to convert a timestamp to a System::TDateTime::TDateTime (see page 3323) value.

For example, use System::TDateTime::FileDateToDateTime to convert the file timestamp returned by FileGetDate or FileAge. You may also need to use System::TDateTime::FileDateToDateTime to convert the Time field of the TSearchRec struct used by the FindFirst and FindNext functions.

See Also

FileGetDate (see page 4142)

FileAge (see page 4119)

TSearchRec (see page 4330)

35.1.17.4.11 TDateTime::FormatString Method

Formats the System::TDateTime::TDateTime (see page 3323) object using the format given by format.

Pascal

```
function FormatString(const format: System::String&): System::String;
```

C++

```
__fastcall System::String FormatString(const System::String& format) const;
```

Description

The following format specifiers are supported:

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	Displays the date using the format given by the ShortDateFormat global variable.
dddddd	Displays the date using the format given by the LongDateFormat global variable.
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec (see page 3550)) using the strings given by the ShortMonthNames global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
t	Displays the time using the format given by the ShortTimeFormat global variable.
tt	Displays the time using the format given by the LongTimeFormat global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString global variable for any hour before noon, and the contents of the TimePMString global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator global variable.
:	Displays the time separator character given by the TimeSeparator global variable.
'xx'/"xx"	Characters enclosed in single or double (see page 3340) quotes are displayed as is, and do not affect formatting.

If the string given by the format parameter is empty, the System::TDateTime::TDateTime (see page 3323) value is formatted as if a c format specifier had been given.

35.1.17.4.12 TDateTime::TimeString Method

Converts the System::TDateTime::TDateTime (see page 3323) object to an AnsiString.

Pascal

```
function TimeString: System::String;
```

C++

```
__fastcall System::String TimeString() const;
```

Description

System::TDateTime::TimeString returns an AnsiString object containing the time value (the fractional portion of the Val (see page 3701) data member) of the System::TDateTime::TDateTime (see page 3323) object to a string. The conversion uses the format specified by the LongTimeFormat variable. Change the format of how the string is displayed by changing the value of one or more of the date/time formatting variables.

See Also

System::TDateTime::DateString (see page 3329)

System::TDateTime::DateTimeString (see page 3329)

35.1.17.5 TDateTime Operators

The operators of the TDateTime class are listed here.

Public Operators

	Name	Description
	- (see page 3334)	This is the overview for the - operator overload.
	-- (see page 3334)	This is the overview for the -- operator overload.
	!= (see page 3335)	Compares two System::TDateTime::TDateTime (see page 3323) objects for inequality.
	+ (see page 3335)	This is the overview for the + operator overload.
	++ (see page 3336)	This is the overview for the ++ operator overload.
	+= (see page 3336)	This is the overview for the += operator overload.
	< (see page 3337)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is less than another specified System::TDateTime::TDateTime (see page 3323) value.
	<= (see page 3337)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is less than or equal to another specified System::TDateTime::TDateTime (see page 3323) value.
	= (see page 3338)	This is the overview for the = operator overload.
	-= (see page 3338)	This is the overview for the -= operator overload.
	== (see page 3339)	Compares two System::TDateTime::TDateTime (see page 3323) objects for equality.
	> (see page 3339)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is greater than another specified System::TDateTime::TDateTime (see page 3323) value.
	>= (see page 3340)	Tests whether this System::TDateTime::TDateTime (see page 3323) value is greater than or equal to another specified System::TDateTime::TDateTime (see page 3323) value.
	double (see page 3340)	Converts the System::TDateTime::TDateTime (see page 3323) object to a double value.
	int (see page 3340)	Converts the System::TDateTime::TDateTime (see page 3323) object to an integer value.
	System::String (see page 3341)	Converts the System::TDateTime::TDateTime (see page 3323) object to a String.

35.1.17.5.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	TDateTime::- (double) (see page 3334)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::- (int) (see page 3334)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::- (TDateTime&) (see page 3334)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::- (TDateTimeBase&) (see page 3334)	Subtracts a specified value from the System::TDateTime::TDateTime (see page 3323) object.

35.1.17.5.1.1 TDateTime::- Operator (TDateTime&)

Subtracts a specified value from the System::TDateTime::TDateTime (see page 3323) object.

Pascal

```
operator -(const rhs: TDateTimeBase&);
operator -(const rhs: TDateTime&);
operator -(const rhs: double);
operator -(const rhs: Integer);
```

C++

```
__fastcall TDateTime operator -(const TDateTimeBase& rhs) const;
__fastcall TDateTime operator -(const TDateTime& rhs) const;
__fastcall TDateTime operator -(const double rhs) const;
__fastcall TDateTime operator -(const int rhs) const;
```

Description

Use the System::TDateTime (see page 3323)::- to subtract the value specified by the rhs operand from the System::TDateTime::TDateTime (see page 3323) object. The rhs operand represents the number of days (including fractional days) to subtract.

35.1.17.5.2 -- Operator

This is the overview for the -- operator overload.

Overload List

	Name	Description
	TDateTime::-- () (see page 3334)	Decrements the value of the System::TDateTime::TDateTime (see page 3323) object by a specified number of days.
	TDateTime::-- (int) (see page 3334)	Decrements the value of the System::TDateTime::TDateTime (see page 3323) object by a specified number of days.

35.1.17.5.2.1 TDateTime::-- Operator ()

Decrements the value of the System::TDateTime::TDateTime (see page 3323) object by a specified number of days.

Pascal

```
operator --;
operator --(: Integer);
```

C++

```
TDateTime& operator --();
TDateTime operator --(int);
```

Description

Use the unary System::TDateTime (see page 3323)::-- to decrement the value of the System::TDateTime::TDateTime (see page 3323) object by 1 day.

Use the binary System::TDateTime (see page 3323)::-- to decrement the value by the number of days specified by the integer operand.

35.1.17.5.3 TDateTime::!= Operator

Compares two System::TDateTime::TDateTime (see page 3323) objects for inequality.

Pascal

```
operator !=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator !=(const TDateTime& rhs) const;
```

Description

The != System::TDateTime (see page 3323)::!= returns true if the two objects do not have the same value, false otherwise.

35.1.17.5.4 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	TDateTime::+ (double) (see page 3335)	Adds a specified value to the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::+ (int) (see page 3335)	Adds a specified value to the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::+ (TDateTime&) (see page 3335)	Adds a specified value to the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::+ (TDateTimeBase&) (see page 3335)	Adds a specified value to the System::TDateTime::TDateTime (see page 3323) object.

35.1.17.5.4.1 TDateTime::+ Operator (TDateTime&)

Adds a specified value to the System::TDateTime::TDateTime (see page 3323) object.

Pascal

```
operator +(const rhs: TDateTimeBase&);
operator +(const rhs: TDateTime&);
operator +(const rhs: double);
operator +(const rhs: Integer);
```

C++

```
__fastcall TDateTime operator +(const TDateTimeBase& rhs) const;
__fastcall TDateTime operator +(const TDateTime& rhs) const;
__fastcall TDateTime operator +(const double rhs) const;
__fastcall TDateTime operator +(const int rhs) const;
```

Description

Use the + System::TDateTime (see page 3323)::+ to add the value specified by the rhs operand to the System::TDateTime::TDateTime (see page 3323) object. The rhs operand represents the number of days (including fractional days) to add.

35.1.17.5.5 ++ Operator

This is the overview for the ++ operator overload.

Overload List

	Name	Description
	TDateTime::++ () (see page 3336)	Increments the value of the System::TDateTime::TDateTime (see page 3323) object by a specified number of days.
	TDateTime::++ (int) (see page 3336)	Increments the value of the System::TDateTime::TDateTime (see page 3323) object by a specified number of days.

35.1.17.5.5.1 TDateTime::++ Operator ()

Increments the value of the System::TDateTime::TDateTime (see page 3323) object by a specified number of days.

Pascal

```
operator ++;
operator ++(: Integer);
```

C++

```
TDateTime& operator ++();
TDateTime operator ++(int);
```

Description

Use the unary ++ System::TDateTime (see page 3323)::++ to increment the value of the System::TDateTime::TDateTime (see page 3323) object by 1 day.

Use the binary ++ System::TDateTime (see page 3323)::++ to increment the value by the number of days specified by the integer operand.

35.1.17.5.6 += Operator

This is the overview for the += operator overload.

Overload List

	Name	Description
	TDateTime::+= (double) (see page 3337)	Increments the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.
	TDateTime::+= (int) (see page 3337)	Increments the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.

	TDateTime::+= (TDateTime&) (see page 3337)	Increments the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.
	TDateTime::+= (TDateTimeBase&) (see page 3337)	Increments the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.

35.1.17.5.6.1 TDateTime::+= Operator (TDateTime&)

Increments the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.

Pascal

```
operator +=(const rhs: TDateTimeBase&);
operator +=(const rhs: TDateTime&);
operator +=(const rhs: double);
operator +=(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator +=(const TDateTimeBase& rhs);
__fastcall TDateTime& operator +=(const TDateTime& rhs);
__fastcall TDateTime& operator +=(const double rhs);
__fastcall TDateTime& operator +=(const int rhs);
```

Description

Use the += System::TDateTime (see page 3323)::+= to add to the value specified by rhs to this System::TDateTime::TDateTime (see page 3323) value. The rhs operand represents the number of days (including fractional days) to add.

35.1.17.5.7 TDateTime::< Operator

Tests whether this System::TDateTime::TDateTime (see page 3323) value is less than another specified System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
operator <(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator <(const TDateTime& rhs) const;
```

Description

The < System::TDateTime (see page 3323)::< returns true if this System::TDateTime::TDateTime (see page 3323) object has a value (Val (see page 3701) data member) less than that of the object specified by rhs, and false otherwise.

35.1.17.5.8 TDateTime::<= Operator

Tests whether this System::TDateTime::TDateTime (see page 3323) value is less than or equal to another specified System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
operator <=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator <=(const TDateTime& rhs) const;
```

Description

The <= System::TDateTime (see page 3323)::<= returns true if this System::TDateTime::TDateTime (see page 3323) object

has a value (Val (see page 3701) data member) less than or equal to that of the object specified by rhs, and false otherwise.

35.1.17.5.9 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	TDateTime::= (double) (see page 3338)	Assigns the value of the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::= (int) (see page 3338)	Assigns the value of the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::= (TDateTime&) (see page 3338)	Assigns the value of the System::TDateTime::TDateTime (see page 3323) object.
	TDateTime::= (TDateTimeBase&) (see page 3338)	Assigns the value of the System::TDateTime::TDateTime (see page 3323) object.

35.1.17.5.9.1 TDateTime::= Operator (TDateTime&)

Assigns the value of the System::TDateTime::TDateTime (see page 3323) object.

Pascal

```
operator =(const rhs: TDateTimeBase&);
operator =(const rhs: TDateTime&);
operator =(const rhs: double);
operator =(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator =(const TDateTimeBase& rhs);
__fastcall TDateTime& operator =(const TDateTime& rhs);
__fastcall TDateTime& operator =(const double rhs);
__fastcall TDateTime& operator =(const int rhs);
```

Description

The assignment System::TDateTime (see page 3323)::= assigns the value of the rhs operand to the System::TDateTime::TDateTime (see page 3323) object. The assignment System::TDateTime (see page 3323)::= can take

a double (see page 3340) or integer value, which is assigned to the System::TDateTime::TDateTime (see page 3323) object such that the integral part of rhs is the number of days that have passed since 12/30/1899 and the fractional part is the time of day, represented as a fraction of 24 hours.

a System::TDateTime::TDateTimeBase (see page 3341) or System::TDateTime::TDateTime (see page 3323) object, whose value is assigned to the System::TDateTime::TDateTime (see page 3323) object. (Note that System::TDateTime::TDateTimeBase (see page 3341) is the base class for the System::TDateTime::TDateTime (see page 3323) class. It has a single member, a public Val (see page 3701) field that represents the value of the object).

35.1.17.5.10 -= Operator

This is the overview for the -= operator overload.

Overload List

	Name	Description
	TDateTime::-= (double) (see page 3339)	Decrements the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.
	TDateTime::-= (int) (see page 3339)	Decrements the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.
	TDateTime::-= (TDateTime&) (see page 3339)	Decrements the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.
	TDateTime::-= (TDateTimeBase&) (see page 3339)	Decrements the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.

35.1.17.5.10.1 TDateTime::-= Operator (TDateTime&)

Decrements the value of this System::TDateTime::TDateTime (see page 3323) object by a specified value.

Pascal

```
operator -=(const rhs: TDateTimeBase&);
operator -=(const rhs: TDateTime&);
operator -=(const rhs: double);
operator -=(const rhs: Integer);
```

C++

```
__fastcall TDateTime& operator -=(const TDateTimeBase& rhs);
__fastcall TDateTime& operator -=(const TDateTime& rhs);
__fastcall TDateTime& operator -=(const double rhs);
__fastcall TDateTime& operator -=(const int rhs);
```

Description

Use the -= System::TDateTime (see page 3323)::-= to subtract the value specified by rhs from this System::TDateTime::TDateTime (see page 3323) value. The rhs operand represents the number of days (including fractional days) to subtract.

35.1.17.5.11 TDateTime::== Operator

Compares two System::TDateTime::TDateTime (see page 3323) objects for equality.

Pascal

```
operator ==(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator ==(const TDateTime& rhs) const;
```

Description

The == System::TDateTime (see page 3323)::== returns true if the two objects have the same value, false otherwise.

35.1.17.5.12 TDateTime::> Operator

Tests whether this System::TDateTime::TDateTime (see page 3323) value is greater than another specified System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
operator >(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator >(const TDateTime& rhs) const;
```

Description

The > System::TDateTime (see page 3323)::> returns true if this System::TDateTime::TDateTime (see page 3323) object has a value (Val (see page 3701) data member) greater than that of the object specified by rhs, and false otherwise.

35.1.17.5.13 TDateTime::>= Operator

Tests whether this System::TDateTime::TDateTime (see page 3323) value is greater than or equal to another specified System::TDateTime::TDateTime (see page 3323) value.

Pascal

```
operator >=(const rhs: TDateTime&);
```

C++

```
__fastcall bool operator >=(const TDateTime& rhs) const;
```

Description

The >= System::TDateTime (see page 3323)::>= returns true if this System::TDateTime::TDateTime (see page 3323) object has a value (Val (see page 3701) data member) greater than or equal to that of the object specified by rhs, and false otherwise.

35.1.17.5.14 TDateTime::double Operator

Converts the System::TDateTime::TDateTime (see page 3323) object to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

The double conversion System::TDateTime::double returns a double that represents this System::TDateTime::TDateTime (see page 3323) object. The integral part of the double is the number of days that have passed since 12/30/1899. The fractional part is the time of day, represented as a fraction of 24 hours.

See Also

System::TDateTime::TDateTime (see page 3325)

35.1.17.5.15 TDateTime::int Operator

Converts the System::TDateTime::TDateTime (see page 3323) object to an integer value.

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

The int conversion System::TDateTime::int returns an int that is the integer conversion of the date portion of this System::TDateTime::TDateTime (see page 3323) object. The resulting integer is the number of days that have passed since

12/30/1899.

See Also

System::TDateTime::TDateTime (see page 3325)

35.1.17.5.16 TDateTime::System::String Operator

Converts the System::TDateTime::TDateTime (see page 3323) object to a String.

Pascal

```
operator System::String;
```

C++

```
__fastcall operator System::String() const;
```

Description

The String conversion System::TDateTime::System::String returns a String object that is the string conversion of this System::TDateTime (see page 3323) object. The conversion uses the format specified by the LongTimeFormat global variable of the System::TDateTime::TDateTime (see page 3323) formatting variables. Change the format of the string display by changing the values of some of the date and time variables.

See Also

System::TDateTime::DateTimeString (see page 3329)

35.1.18 TDateTimeBase Class

Class Hierarchy

Pascal

```
TDateTimeBase = class;
```

C++

```
class TDateTimeBase;
```

File

systdate.h

Description

This is class System::TDateTimeBase.

35.1.18.1 TDateTimeBase Members

The following tables list the members exposed by TDateTimeBase.

Public Data Members

	Name	Description
	Val (see page 3342)	This is Val, a member of class TDateTimeBase.

35.1.18.2 TDateTimeBase Data Members

The data members of the TDateTimeBase class are listed here.

Public Data Members

	Name	Description
	Val (see page 3342)	This is Val, a member of class TDateTimeBase.

35.1.18.2.1 TDateTimeBase::Val Data Member

Pascal

```
Val: double;
```

C++

```
double Val;
```

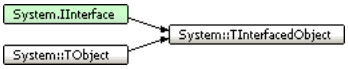
Description

This is Val, a member of class TDateTimeBase.

35.1.19 TInterfacedObject Class

System::TInterfacedObject is a base for simple classes that need a basic IInterface (see page 3502) implementation.

Class Hierarchy



Pascal

```
TInterfacedObject = class(TObject, IInterface);
```

C++

```
class TInterfacedObject : public TObject, public IInterface;
```

File

System.pas

Description

System::TInterfacedObject is a thread-safe base class for Delphi classes that support interfaces. System::TInterfacedObject provides basic reference-counting functionality that makes its descendant classes useful in both Delphi and C++ code.

Note: In Delphi, variables declared to be of type System::TInterfacedObject have dynamic-binding, meaning that an instance of a descendant class can be assigned to that variable at runtime. This is because System::TInterfacedObject implements System::TInterfacedObject::QueryInterface (see page 3347), which is called at runtime to query whether the instantiated object supports the requested interface. If it does, then the descendant object is successfully assigned to the variable. Without System::TInterfacedObject::QueryInterface (see page 3347) support, such an assignment would produce a compile-time error.

Note: In Delphi, a descendant of System::TInterfacedObject does not have a class factories or a CLSID. Thus it cannot be instantiated except by calling a constructor.

Note: System::TInterfacedObject does not implement the methods of the C++ version of IUnknown (see page 3756) or IInterface (see page 3502). This is not a problem when using C++ wrappers for System::TInterfacedObject descendants that are written in Delphi. However, if you are creating C++ descendants of System::TInterfacedObject, you must provide your own implementation of IUnknown (see page 3756). For information on how to do this, see Creating classes that support IUnknown (see page 3756) (supplied with C++ products only).

35.1.19.1 TInterfacedObject Members

The following tables list the members exposed by TInterfacedObject.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

35.1.19.2 TInterfacedObject Data Members

The data members of the TInterfacedObject class are listed here.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

35.1.19.2.1 TInterfacedObject::FRefCount Data Member

Pascal

```
FRefCount: Integer;
```

C++

```
Integer FRefCount;
```

Description

This is FRefCount, a member of class TInterfacedObject.

35.1.19.3 TInterfacedObject Methods

The methods of the TInterfacedObject class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.

35.1.19.3.1 TInterfacedObject::_AddRef Method

Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__stdcall Integer _AddRef();
```

Description

Do not call the protected System::TInterfacedObject::_AddRef method directly. System::TInterfacedObject::_AddRef is called through the interface to indicate that another object is using the interfaced object's interface pointer. When the other object is through with the interface, it calls System::TInterfacedObject::_Release (see page 3345) through the same interface. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

System::TInterfacedObject::_AddRef increments the System::TInterfacedObject::RefCount (see page 3347) property.

See Also

System::TInterfacedObject::RefCount (see page 3347)

System::TInterfacedObject::_Release (see page 3345)

35.1.19.3.2 TInterfacedObject::_Release Method

Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__stdcall Integer _Release();
```

Description

Do not call the protected _System::TInterfacedObject::_Release method directly. _System::TInterfacedObject::_Release is called through the IInterface (see page 3502) interface to indicate that another object is through with the interfaced object's interface pointer. This allows the interfaced object to free itself when it is no longer used (when the reference count goes to zero).

_System::TInterfacedObject::_Release decrements the System::TInterfacedObject::RefCount (see page 3347) property.

See Also

System::TInterfacedObject::RefCount (see page 3347)

System::TInterfacedObject::_AddRef (see page 3345)

System::TInterfacedObject::BeforeDestruction (see page 3346)

35.1.19.3.3 TInterfacedObject::AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; virtual; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject (see page 3342) is allocated, System::TInterfacedObject::RefCount (see page 3347) is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount (see page 3347) to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

35.1.19.3.4 TInterfacedObject::BeforeDestruction Method

Checks that the reference count is zero before allowing the interfaced object to be destroyed.

Pascal

```
procedure BeforeDestruction; virtual; override;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

System::TInterfacedObject::BeforeDestruction is called automatically immediately before the interfaced object's destructor. It checks that the reference count is zero, and, if it is not, raises an exception.

See Also

System::TInterfacedObject::RefCount (see page 3347)

System::TObject::Free (see page 3372)

35.1.19.3.5 TInterfacedObject::NewInstance Method

Allocates memory for each instance of an interfaced object.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
TObject NewInstance();
```

Description

All constructors call System::TInterfacedObject::NewInstance automatically. System::TInterfacedObject::NewInstance calls System::TObject::InstanceSize (see page 3377) to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TInterfacedObject::NewInstance directly.

System::TInterfacedObject (see page 3342) overrides System::TInterfacedObject::NewInstance to increment the reference count when a new instance is created. This prevents any constructor from accidentally deleting the instance due to a

System::TInterfacedObject::RefCount (see page 3347) of zero. The System::TInterfacedObject::AfterConstruction (see page 3345) method decrements the value of System::TInterfacedObject::RefCount (see page 3347) after all constructors have executed.

See Also

System::TInterfacedObject::RefCount (see page 3347)

System::TObject::NewInstance (see page 3379)

System::TObject::InstanceSize (see page 3377)

System::TInterfacedObject::AfterConstruction (see page 3345)

35.1.19.3.6 TInterfacedObject::QueryInterface Method

Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__stdcall HResult QueryInterface(const TGUID IID, out  Obj);
```

Description

Do not call the protected System::TInterfacedObject::QueryInterface method directly. System::TInterfacedObject::QueryInterface is called through IInterface (see page 3502) to obtain an interface pointer for the interface identified by the IID parameter. If the object supports the requested interface, it is returned in the Obj parameter and System::TInterfacedObject::QueryInterface returns S_OK (see page 3860). If the object does not support the interface, System::TInterfacedObject::QueryInterface returns E_NOINTERFACE (see page 3845).

If System::TInterfacedObject::QueryInterface successfully returns an interface pointer, it automatically increments the reference count.

See Also

System::TInterfacedObject::_AddRef (see page 3345)

35.1.19.4 TInterfacedObject Properties

The properties of the TInterfacedObject class are listed here.

Public Properties

	Name	Description
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

35.1.19.4.1 TInterfacedObject::RefCount Property

Indicates the number of interface pointers currently dependent upon the object.

Pascal

```
property RefCount: Integer;
```

C++

```
__property Integer RefCount;
```

Description

System::TInterfacedObject::RefCount defines the lifetime of the object. When System::TInterfacedObject::RefCount is zero, the object is destroyed.

System::TInterfacedObject::RefCount is incremented by calls to the _AddRef (see page 3345) method and decremented by the _Release (see page 3345) method.

Note: If the destructor is called on a System::TInterfacedObject (see page 3342) descendant that is not fully constructed (an exception was raised in the constructor), then System::TInterfacedObject::RefCount will have a value of 1. Otherwise, System::TInterfacedObject::RefCount will be zero.

35.1.20 TMetaClass Class

System::TMetaClass is the C++ representation of the Delphi class-reference type.

Class Hierarchy

Pascal

```
TMetaClass = class;
```

C++

```
class TMetaClass;
```

File

systobj.h

Description

System::TMetaClass allows operations to be performed directly on classes. This contrasts with class types, which can be instantiated as objects and which allow operations to be performed on those instances. The System::TMetaClass for a given class can be acquired by using the __classid keyword extension.

A variable of type System::TMetaClass can be NULL, which indicates that the variable does not currently reference a class.

Every class inherits a method function called ClassType, which returns a reference to the class of an object. The type of the value returned by ClassType is TClass (see page 3785) (System::TMetaClass*), which is a pointer to a System::TMetaClass.

The methods of System::TMetaClass correspond to the static methods of TObject (see page 3354). They have the same functionality but differ in parameters. Whereas the TObject (see page 3354) methods take a TClass (see page 3785) (System::TMetaClass*) as the first parameter, this parameter is implicit in the System::TMetaClass methods.

C++ Examples:

```
/*
The following example allows users to type the name of a
graphics class into an edit control and when a button is
pressed, the name of the default extension for that class is
displayed in another edit control.  In the form's OnCreate
event handler (or a similar place) you must register the
graphics classes so that GetClass can find them.
*/
#include <memory>       //for STL auto_ptr class

TMetaClass *MetaClass;
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TClassFinder> myClassFinder(new TClassFinder(MetaClass, False));
  TGraphicClass mygraphclass = (TGraphicClass) (myClassFinder->GetClass(Edit2->Text));
//  TGraphicClass mygraphclass = (TGraphicClass) GetClass(Edit2->Text);
  Edit1->Text = GraphicExtension(mygraphclass);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  MetaClass = __classid(TIcon); // ico
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(Graphics::TBitmap); // bmp
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TMetafile); // emf
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TShape);  // nographic extension
  RegisterClasses(&MetaClass, 0);
  MetaClass = __classid(TImage); // nographic extension
  RegisterClasses(&MetaClass, 0);
}
```

See Also

System::TObject (see page 3354)

Class References

35.1.21 TMonitor Record

TMonitor provides methods for synchronizing the access of several threads to a single object.

Pascal

```
TMonitor = record
end;
```

C++

```
struct TMonitor {
public:
  struct TWaitingThread {
  };
};
```

File

System

Description

Use the class methods of TMonitor to synchronize the threads' access to resources in a multithreaded application.

See Also

Enter (see page 3350)

Exit (see page 3351)

Wait (see page 3353)

Pulse (see page 3352)

35.1.21.1 TMonitor Members

The following tables list the members exposed by TMonitor.

Public Methods

	Name	Description
	Enter (see page 3350)	This is the overview for the Enter method overload.
	Exit (see page 3351)	Releases a lock of the calling thread on the specified object.
	Pulse (see page 3352)	Notifies the next thread in the waiting queue that it will be able to lock the specified object as soon as the calling thread releases the object.
	PulseAll (see page 3352)	Signals all threads in the waiting queue that one of them will be able to lock the specified object as soon as the calling thread releases the object.
	SetSpinCount (see page 3353)	SetSpinCount sets the number of times that Enter (see page 3350) attempts to lock the specified object.
	TryEnter (see page 3353)	Attempts to lock the specified object, so that only the calling thread is able to access the object.
	Wait (see page 3353)	Drops the lock on the specified object and puts the calling thread in the waiting queue for a specified amount of time.

35.1.21.2 TMonitor Methods

The methods of the TMonitor class are listed here.

Public Methods

	Name	Description
	Enter (see page 3350)	This is the overview for the Enter method overload.
	Exit (see page 3351)	Releases a lock of the calling thread on the specified object.
	Pulse (see page 3352)	Notifies the next thread in the waiting queue that it will be able to lock the specified object as soon as the calling thread releases the object.
	PulseAll (see page 3352)	Signals all threads in the waiting queue that one of them will be able to lock the specified object as soon as the calling thread releases the object.
	SetSpinCount (see page 3353)	SetSpinCount sets the number of times that Enter (see page 3350) attempts to lock the specified object.
	TryEnter (see page 3353)	Attempts to lock the specified object, so that only the calling thread is able to access the object.
	Wait (see page 3353)	Drops the lock on the specified object and puts the calling thread in the waiting queue for a specified amount of time.

35.1.21.2.1 Enter Method

This is the overview for the Enter method overload.

Overload List

	Name	Description
	TMonitor.Enter (TObject) (see page 3350)	Prohibits the access of all other threads but the calling one to the specified object.
	TMonitor.Enter (TObject, Cardinal) (see page 3351)	Prohibits the access of all other threads but the calling one to the specified object.

35.1.21.2.1.1 TMonitor.Enter Method (TObject)

Prohibits the access of all other threads but the calling one to the specified object.

Pascal

```
class procedure Enter(AObject: TObject); overload; static;
```

C++

```
static __fastcall Enter(TObject * AObject);
```

Description

Use the Enter method to prohibit `AObject` from being accessed by threads, other than the calling one.

Before attempting to lock the specified object, Enter tests whether the thread that is currently locking the object, if any, has made an equal number of calls to Enter and to Exit (see page 3351). If not, the calling thread is blocked from accessing the object until the previous thread releases it.

The Enter function attempts to lock the specified object for the amount of time designated by the `Timeout` parameter. If the lock attempt is successful, Enter returns True, otherwise it returns False.

See Also

TryEnter (see page 3353)

Exit (see page 3351)

TMonitor (see page 3349)

35.1.21.2.1.2 TMonitor.Enter Method (TObject, Cardinal)

Prohibits the access of all other threads but the calling one to the specified object.

Pascal

```
class function Enter(AObject: TObject; Timeout: Cardinal): Boolean; overload; static;
```

C++

```
static __fastcall Boolean Enter(TObject * AObject, unsigned Timeout);
```

Description

Use the Enter method to prohibit `AObject` from being accessed by threads, other than the calling one.

Before attempting to lock the specified object, Enter tests whether the thread that is currently locking the object, if any, has made an equal number of calls to Enter and to Exit (see page 3351). If not, the calling thread is blocked from accessing the object until the previous thread releases it.

The Enter function attempts to lock the specified object for the amount of time designated by the `Timeout` parameter. If the lock attempt is successful, Enter returns True, otherwise it returns False.

See Also

TryEnter (see page 3353)

Exit (see page 3351)

TMonitor (see page 3349)

35.1.21.2.2 TMonitor.Exit Method

Releases a lock of the calling thread on the specified object.

Pascal

```
class procedure Exit(AObject: TObject); overload; static;
```

C++

```
static __fastcall Exit(TObject * AObject);
```

Description

Use the Exit method to release a lock of the calling thread on `AObject`, which is also owned by the thread. The object is released from all the locks of the calling thread as soon as the number of the thread's calls to Enter (see page 3350).

See Also

Enter (see page 3350)

TMonitor (see page 3349)

35.1.21.2.3 TMonitor.Pulse Method

Notifies the next thread in the waiting queue that it will be able to lock the specified object as soon as the calling thread releases the object.

Pascal

```
class procedure Pulse(AObject: TObject); overload; static;
```

C++

```
static __fastcall Pulse(TObject * AObject);
```

Description

The Pulse method notifies the next thread in the waiting queue that it can move to the ready queue. Thus, as soon as the calling thread releases the lock on `AObject`, the next thread in the ready queue will lock `AObject`.

See Also

PulseAll (see page 3352)

TMonitor (see page 3349)

35.1.21.2.4 TMonitor.PulseAll Method

Signals all threads in the waiting queue that one of them will be able to lock the specified object as soon as the calling thread releases the object.

Pascal

```
class procedure PulseAll(AObject: TObject); overload; static;
```

C++

```
static __fastcall PulseAll(TObject * AObject);
```

Description

The PulseAll method notifies all threads in the waiting queue that one of them can move to the ready queue. As soon as the calling thread releases the lock on `AObject`, the first thread in the ready queue will lock `AObject`.

See Also

Pulse (see page 3352)

TMonitor (see page 3349)

35.1.21.2.5 TMonitor.SetSpinCount Method

SetSpinCount sets the number of times that Enter (see page 3350) attempts to lock the specified object.

Pascal

```
class procedure SetSpinCount(AObject: TObject; ASpinCount: Integer); static;
```

C++

```
static __fastcall SetSpinCount(TObject * AObject, int ASpinCount);
```

Description

Use SetSpinCount to set the number of attempts `ASpinCount` that Enter (see page 3350) makes trying to lock `AObject`.

See Also

Enter (see page 3350)

TMonitor (see page 3349)

35.1.21.2.6 TMonitor.TryEnter Method

Attempts to lock the specified object, so that only the calling thread is able to access the object.

Pascal

```
class function TryEnter(AObject: TObject): Boolean; overload; static;
```

C++

```
static __fastcall Boolean TryEnter(TObject * AObject);
```

Description

The TryEnter function attempts to lock the object given as parameter, so that only the calling thread can access the object.

TryEnter returns True if it manages to lock the object, or False otherwise.

As opposed to the Enter (see page 3350) method, TryEnter does not block the calling thread if the given object is currently owned by other threads.

See Also

Enter (see page 3350)

TMonitor (see page 3349)

35.1.21.2.7 TMonitor.Wait Method

Drops the lock on the specified object and puts the calling thread in the waiting queue for a specified amount of time.

Pascal

```
class function Wait(AObject: TObject; Timeout: Cardinal): Boolean; overload; static;
```

C++

```
static __fastcall Boolean Wait(TObject * AObject, unsigned Timeout);
```

Description

Use Wait to drop the lock on the specified object and also to put the calling thread in the waiting queue for the amount of time specified in the `Timeout` parameter. Hence, the calling thread is blocked until it is again able to lock `AObject`.

See Also

TMonitor (see page 3349)

35.1.22 TObject Class

System::TObject is the ultimate ancestor of all objects and components.

Class Hierarchy

Pascal

```
TObject = class;
```

C++

```
class TObject;
```

File

System.pas

Description

System::TObject encapsulates fundamental behavior common to objects by introducing methods that:

- Create (see page 3355), maintain and destroy instances of the object by allocating, initializing, and freeing required memory.
- Respond when object instances are created or destroyed. return class-type and instance information on an object and runtime type information (RTTI) about its published properties.
- Support message handling.
- Support interfaces that the object implements.

Use System::TObject as an immediate base class when declaring simple objects that do not need to persist (are not saved and reloaded) and that do not need to be assigned to other objects.

Much of the capability of objects is established by methods that System::TObject introduces. Many of these methods are used internally by IDEs and are not intended for users to call directly. Others are overridden in descendant objects that have more complex behavior.

Although System::TObject is the based object of a component framework, not all objects are components. All component classes are descended from TComponent.

Note: System::TObject is never directly instantiated. Although it does not use programming language features that prevent instantiation, System::TObject is an abstract class.

See Also

TComponent

TPersistent

TObject Branch

TWinControl/TWidgetControl Branch

VCL Overview

35.1.22.1 TObject Members

The following tables list the members exposed by TObject.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

35.1.22.2 TObject::Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
Create();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject (see page 3354) allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

35.1.22.3 TObject::Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; virtual;
```

C++

```
virtual Destroy();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free (see page 3372) instead. Free (see page 3372) verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject (see page 3354) deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free (see page 3372) on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free (see page 3372) provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

35.1.22.4 TObject Methods

The methods of the TObject class are listed here.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.

	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

35.1.22.4.1 TObject::AfterConstruction Method

Responds after the last constructor has executed.

Pascal

```
procedure AfterConstruction; virtual;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

The System::TObject::AfterConstruction method implemented in System::TObject (see page 3354) does nothing. Override this method when creating a class that takes some action after the object is created. For example, TCustomForm overrides System::TObject::AfterConstruction to generate an OnCreate event.

See Also

System::TObject::BeforeDestruction (see page 3358)

OnCreate

System::TObject::Create (see page 3355)

35.1.22.4.2 TObject::BeforeDestruction Method

Responds before the first destructor executes.

Pascal

```
procedure BeforeDestruction; virtual;
```

C++

```
virtual __fastcall BeforeDestruction();
```

Description

System::TObject::BeforeDestruction is called automatically before the object's first destructor executes. Do not call it explicitly in your applications.

The System::TObject::BeforeDestruction method implemented in System::TObject (see page 3354) does nothing. Override this method when creating a class that takes some action before the object is destroyed. For example, TCustomForm overrides System::TObject::BeforeDestruction to generate an OnDestroy event.

Note: System::TObject::BeforeDestruction is not called when the object is destroyed before it is fully constructed. That is, if the object's constructor raises an exception, the destructor is called to dispose of the object, but System::TObject::BeforeDestruction is not called.

See Also

System::TObject::AfterConstruction (see page 3357)

OnDestroy

System::TObject::Destroy (see page 3356)

35.1.22.4.3 TObject::ClassInfo Method

Returns a pointer to the runtime type information (RTTI) table for the object type.

Pascal

```
class function ClassInfo: Pointer;
```

C++

```
Pointer ClassInfo();
```

Description

System::TObject::ClassInfo provides access to the RTTI table for a given object type.

Note: The format of the RTTI table is subject to change. Applications should use the other RTTI methods provided by System::TObject (see page 3354); these provide easy, consistent access to RTTI data.

Note: Some classes do not provide runtime type information. For these classes, System::TObject::ClassInfo returns nil (Delphi) or NULL (C++). All classes descended from TPersistent do provide runtime type information.

Note: The Delphi compiler does not provide RTTI by default. Most classes implemented in Delphi that need to provide RTTI are descended from TPersistent. If your Delphi class needs RTTI but not the full functionality of TPersistent, compile the code with the $M switch.

Note: The C++ compiler provides RTTI by default. This feature can be disabled as project option and selectively re-enabled using the __rtti keyword.

See Also

System::TObject::ClassType (see page 3364)

System::TObject::ClassName (see page 3359)

System::TObject::ClassParent (see page 3361)

System::TObject::FieldAddress (see page 3372)

System::TObject::MethodAddress (see page 3378)

System::TObject::InstanceSize (see page 3377)

System::TObject::MethodName (see page 3379)

System::TObject::InheritsFrom (see page 3376)

35.1.22.4.4 TObject::ClassName Method

Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).

Pascal

```
class function ClassName: string;
```

C++

```
string ClassName();
```

Description

Use System::TObject::ClassName to obtain the class name from an object instance or class reference. This is useful for differentiating object instances that are assigned to a variable that has the type of an ancestor class.

Note: In C++ code, call System::TObject::ClassName as a method to obtain an object's class name. Use the global static function to obtain the class name from a metaclass object.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassRef->ClassName());
    ClassRef = ClassRef->ClassParent();
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
```

```
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolbar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
*/
void AddBands(TCoolBar *CoolBar, TList *Objects)
{
  TControl *pCurrent;
  for (int i = 0; i < Objects->Count; i++)
  {
    pCurrent = reinterpret_cast<TControl *>(Objects->Items[i]);
    // only add windowed controls to the coolbar
    if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
    {
      // Placing the control onto the CoolBar
      // causes the TCoolBar object to create a TCoolBand
      // and place the control within the band.
      pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
      // Get the reference of the TCoolBand just created and assign text.
      String S = pCurrent->ClassName();
      CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
    }
  }
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TCoolBar *CoolBar = new TCoolBar(this); // The Ownder (Form1) will clean this up.
  CoolBar->Parent = this;
  CoolBar->Align = alTop;
  std::auto_ptr<TList> List(new TList);
  try
  {
    TControl *Control;
    Control = new TButton(CoolBar); // The Ownder (CoolBar) will clean this up.
    List->Add(Control);
    Control = new TCheckBox(CoolBar); // The Ownder (CoolBar) will clean this up.
    List->Add(Control);
    Control = new TEdit(CoolBar); // The Ownder (CoolBar) will clean this up.
    List->Add(Control);
    AddBands(CoolBar, List.get());
  }
  catch (...)
  {
    ShowMessage("Some objects could not be added to Coolband");
  }
}
```

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassNameIs (see page 3361)

System::TObject::ClassParent (see page 3361)

System::TObject::ClassType (see page 3364)

35.1.22.4.5 TObject::ClassNameIs Method

Determines whether an object is of a specific type.

Pascal

```
class function ClassNameIs(const Name: string): Boolean;
```

C++

```
Boolean ClassNameIs(const string Name);
```

Description

System::TObject::ClassNameIs determines whether an object instance or class reference has a class name that matches a specified string. This is useful to query objects across modules or shared libraries.

Note: In C++ code, call System::TObject::ClassNameIs as a method to compare an object's class name. Use the global static function to compare the class name from a metaclass object.

See Also

System::TObject::ClassName (see page 3359)

System::TObject::InheritsFrom (see page 3376)

35.1.22.4.6 TObject::ClassParent Method

Returns the type of the immediate ancestor of a class.

Pascal

```
class function ClassParent: TClass;
```

C++

```
TClass ClassParent();
```

Description

System::TObject::ClassParent returns the name of the parent class for an object instance or class reference. For System::TObject (see page 3354), System::TObject::ClassParent returns nil (Delphi) or NULL (C++).

Avoid using System::TObject::ClassParent in application code.

Note: In Delphi code, use the is or as operators instead of System::TObject::ClassParent.

Note: In C++ code, use a dynamic cast or the System::TObject::InheritsFrom (see page 3376) method instead of System::TObject::ClassParent.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassRef->ClassName());
    ClassRef = ClassRef->ClassParent();
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
    for (int i = 0; i < ComponentCount; i++)
      ListBox1->Items->InsertObject(0,
                                    Components[i]->Name,
                                    dynamic_cast<TObject *>(Components[i]));
  }

  void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y)
  {
    if (Button == mbRight)
    {
      TClass ClassRef;
      int Index = ListBox1->ItemAtPos(Point(X,Y), true);
      // only components that are controls have a position
      // make sure the component is a control
      for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
           ClassRef != NULL;
           ClassRef = ClassRef->ClassParent())
        if (String(ClassRef->ClassName()) == "TControl")
        {
          TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
          StatusBar1->SimpleText =
            TheObject->Name + " is at (" +
            IntToStr(TheObject->Left) + ", " +
            IntToStr(TheObject->Top) + ")";
          break;
        }
      if (ClassRef == NULL) // if it wasn't a control
        MessageBeep(0);
    }
  }
```

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassInfo (see page 3358)

System::TObject::ClassType (see page 3364)

System::TObject::InheritsFrom (see page 3376)

35.1.22.4.7 TObject::ClassType Method

Returns the class reference for the object's class.

Pascal

```
function ClassType: TClass;
```

C++

```
__fastcall TClass ClassType();
```

Description

Note: System::TObject::ClassType dynamically determines the type of an object and returns its class reference, or metaclass.

Avoid using System::TObject::ClassType in application code.

Note: In Delphi code, use the is or as operators instead of System::TObject::ClassType.

Note: In C++ code, use a dynamic cast or the System::TObject::InheritsFrom (see page 3376) method instead of System::TObject::ClassType.

C++ Examples:

```
/*
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TClass ClassRef;
  ListBox1->Clear();
  ClassRef = Sender->ClassType();
  while (ClassRef != NULL)
  {
    ListBox1->Items->Add(ClassRef->ClassName());
    ClassRef = ClassRef->ClassParent();
  };
}
/*
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
*/
/*
For the following example, add a button, a status bar, and a
list box to the form. Set the SimplePanel property of the
status bar to true, using the object inspector. Also, populate
the OnMouseUp event handler for the list box.  The following
```

```
code fills a list box with the names of all components on the
form when the user clicks the button. References to the
components themselves are inserted along with the names. The
components are all inserted at the front of the list, so that
the last component added to the form is the first component
in the list.  When the user right-clicks the name of an
object in the list, the component's coordinates are displayed
on the status bar. Note that because we are using the right-
click, the item need not be selected.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ComponentCount; i++)
    ListBox1->Items->InsertObject(0,
                                  Components[i]->Name,
                                  dynamic_cast<TObject *>(Components[i]));
}

void __fastcall TForm1::ListBox1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  if (Button == mbRight)
  {
    TClass ClassRef;
    int Index = ListBox1->ItemAtPos(Point(X,Y), true);
    // only components that are controls have a position
    // make sure the component is a control
    for (ClassRef = ListBox1->Items->Objects[Index]->ClassType();
         ClassRef != NULL;
         ClassRef = ClassRef->ClassParent())
      if (String(ClassRef->ClassName()) == "TControl")
      {
        TControl *TheObject = dynamic_cast<TControl *>(ListBox1->Items->Objects[Index]);
        StatusBar1->SimpleText =
          TheObject->Name + " is at (" +
          IntToStr(TheObject->Left) + ", " +
          IntToStr(TheObject->Top) + ")";
        break;
      }
    if (ClassRef == NULL) // if it wasn't a control
      MessageBeep(0);
  }
}
/*
To run this example, add the example code to a new project.
The example code dynamically creates a TCoolbar and three
TCoolBand objects populated with a windowed control in each
TCoolBand.
*/
void AddBands(TCoolBar *CoolBar, TList *Objects)
{
  TControl *pCurrent;
  for (int i = 0; i < Objects->Count; i++)
  {
    pCurrent = reinterpret_cast<TControl *>(Objects->Items[i]);
    // only add windowed controls to the coolbar
    if (dynamic_cast<TWinControl *>(pCurrent) != NULL)
    {
      // Placing the control onto the CoolBar
      // causes the TCoolBar object to create a TCoolBand
      // and place the control within the band.
      pCurrent->Parent = CoolBar;   // This statement increments CoolBar->Bands->Count.
      // Get the reference of the TCoolBand just created and assign text.
      String S = pCurrent->ClassName();
      CoolBar->Bands->Items[CoolBar->Bands->Count - 1]->Text = S;
    }
```

```
  }
}

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TCoolBar *CoolBar = new TCoolBar(this); // The Ownder (Form1) will clean this up.
  CoolBar->Parent = this;
  CoolBar->Align = alTop;
  std::auto_ptr<TList> List(new TList);
  try
  {
    TControl *Control;
    Control = new TButton(CoolBar); // The Ownder (CoolBar) will clean this up.
    List->Add(Control);
    Control = new TCheckBox(CoolBar); // The Ownder (CoolBar) will clean this up.
    List->Add(Control);
    Control = new TEdit(CoolBar); // The Ownder (CoolBar) will clean this up.
    List->Add(Control);
    AddBands(CoolBar, List.get());
  }
  catch (...)
  {
    ShowMessage("Some objects could not be added to Coolband");
  }
}
```

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
    ClassRef := ClassRef.ClassParent;
  end;
end;
{
The list box contains the following strings after the user clicks the button:
    TButton
    TButtonControl
    TWinControl or TWinControl
    TControl
    TComponent
    TPersistent
    TObject
}
```

See Also

System::TObject::ClassInfo (see page 3358)

System::TObject::ClassName (see page 3359)

System::TObject::ClassParent (see page 3361)

35.1.22.4.8 TObject::CleanupInstance Method

Performs finalization on long strings, variants, and interface variables within a class.

Pascal

```
procedure CleanupInstance;
```

C++

```
__fastcall CleanupInstance();
```

Description

Do not call System::TObject::CleanupInstance directly. System::TObject::CleanupInstance is called automatically when the object instance is destroyed.

System::TObject::CleanupInstance releases all long strings and variants. It sets long strings to empty and variants to Unassigned.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

delete

35.1.22.4.9 TObject::DefaultHandler Method

Provides the interface for a method that processes message records.

Pascal

```
procedure DefaultHandler(var Message); virtual;
```

C++

```
virtual DefaultHandler(var  Message);
```

Description

System::TObject::DefaultHandler is called by System::TObject::Dispatch (see page 3369) when it cannot find a method for a particular message. System::TObject::DefaultHandler provides message handling for all messages for which an object does not have specific handlers. Descendant classes that process messages override System::TObject::DefaultHandler according to the types of messages they handle.

Note: In a Delphi message-handling method, calling inherited results in a call to the ancestor's System::TObject::DefaultHandler method only if that ancestor does not specify a message method for the particular message being handled. Otherwise, calling inherited results in a call to the specific handler for that type of message.

Delphi Examples:

```
{
This example demonstrates the use of message dispatching
on TObject and any descendants. Using the same techniques
a form of message based OOP can be acheived where an object
"implements" a certain interface based on messages.
}

{ Define the object interface messages }
const
  INTF_MESSAGE_LOWER_STR = 1;
```

```
    INTF_MESSAGE_UPPER_STR = 2;

  type
    { String container message that will be passed }
    TStrCntrMessage = packed record
      MessageId : Word;
      MsgText   : String;
    end;

    { String handler object that will serve requests }
    TStringHandler = class
    private
      procedure StrUpCase(var AMessage : TStrCntrMessage);
        message INTF_MESSAGE_UPPER_STR;

      procedure StrLowCase(var AMessage : TStrCntrMessage);
        message INTF_MESSAGE_LOWER_STR;

    public
      procedure DefaultHandler(var Message); override;

    end;

  { TStringHandler }

  procedure TStringHandler.DefaultHandler(var Message);
  var
    MessageId : Word absolute Message;
  begin
    { Display out message then pass control to default handler }
    MessageDlg('Unhandled message ' + IntToStr(MessageId) +
      ' received!', mtInformation, [mbOK], 0);

    inherited;
  end;

  procedure TStringHandler.StrLowCase(
    var AMessage: TStrCntrMessage);
  begin
    AMessage.MsgText := LowerCase(AMessage.MsgText);
  end;

  procedure TStringHandler.StrUpCase(
    var AMessage: TStrCntrMessage);
  begin
    AMessage.MsgText := UpperCase(AMessage.MsgText);
  end;

  { TForm2}

  procedure TForm2.FormCreate(Sender: TObject);
  var
    Hndlr: TStringHandler;
    Msg: TStrCntrMessage;
  begin
    { Create object instance, dispatch a message and free it }
    Hndlr := TStringHandler.Create();

    { Lower-case a string using messaging }
    Msg.MessageId := INTF_MESSAGE_LOWER_STR;
    Msg.MsgText := 'Hello World';

    Hndlr.Dispatch(Msg);

    MessageDlg('Lower-cased string: ' + Msg.MsgText,
      mtInformation, [mbOK], 0);
```

```
  { Upper-case a string using messaging }
  Msg.MessageId := INTF_MESSAGE_UPPER_STR;
  Msg.MsgText := 'Hello World';

  Hndlr.Dispatch(Msg);

  MessageDlg('Upper-cased string: ' + Msg.MsgText,
    mtInformation, [mbOK], 0);

  Hndlr.Free;
end;
```

See Also

System::TObject::Dispatch (see page 3369)

35.1.22.4.10 TObject::Dispatch Method

Calls message-handling methods for the object, based on the contents of the Message parameter.

Pascal

```
procedure Dispatch(var Message); virtual;
```

C++

```
virtual Dispatch(var  Message);
```

Description

Call System::TObject::Dispatch to automatically pass messages to the appropriate message handler.

System::TObject::Dispatch determines whether a message is in the list of message handlers declared for the object. If the object does not handle the message, System::TObject::Dispatch then examines the message-handler list of the ancestor class, and continues checking ancestors until it either finds a specific handler or runs out of ancestors, in which case it calls System::TObject::DefaultHandler (see page 3367).

The only assumption System::TObject::Dispatch makes about the data in Message is that the first two bytes contain a message ID—that is, an integer that determines which message handler System::TObject::Dispatch calls. Although any kind of data can be passed to System::TObject::Dispatch, most System::TObject (see page 3354) descendants expect a message record such as TMessage or a specific data structure type.

Delphi Examples:

```
{
This example demonstrates the use of message dispatching
on TObject and any descendants. Using the same techniques
a form of message based OOP can be acheived where an object
"implements" a certain interface based on messages.
}

{ Define the object interface messages }
const
  INTF_MESSAGE_LOWER_STR = 1;
  INTF_MESSAGE_UPPER_STR = 2;

type
  { String container message that will be passed }
  TStrCntrMessage = packed record
    MessageId : Word;
    MsgText   : String;
  end;

  { String handler object that will serve requests }
```

```
  TStringHandler = class
  private
    procedure StrUpCase(var AMessage : TStrCntrMessage);
      message INTF_MESSAGE_UPPER_STR;

    procedure StrLowCase(var AMessage : TStrCntrMessage);
      message INTF_MESSAGE_LOWER_STR;

  public
    procedure DefaultHandler(var Message); override;

  end;

{ TStringHandler }

procedure TStringHandler.DefaultHandler(var Message);
var
  MessageId : Word absolute Message;
begin
  { Display out message then pass control to default handler }
  MessageDlg('Unhandled message ' + IntToStr(MessageId) +
    ' received!', mtInformation, [mbOK], 0);

  inherited;
end;

procedure TStringHandler.StrLowCase(
  var AMessage: TStrCntrMessage);
begin
  AMessage.MsgText := LowerCase(AMessage.MsgText);
end;

procedure TStringHandler.StrUpCase(
  var AMessage: TStrCntrMessage);
begin
  AMessage.MsgText := UpperCase(AMessage.MsgText);
end;

{ TForm2}

procedure TForm2.FormCreate(Sender: TObject);
var
  Hndlr: TStringHandler;
  Msg: TStrCntrMessage;
begin
  { Create object instance, dispatch a message and free it }
  Hndlr := TStringHandler.Create();

  { Lower-case a string using messaging }
  Msg.MessageId := INTF_MESSAGE_LOWER_STR;
  Msg.MsgText := 'Hello World';

  Hndlr.Dispatch(Msg);

  MessageDlg('Lower-cased string: ' + Msg.MsgText,
    mtInformation, [mbOK], 0);

  { Upper-case a string using messaging }
  Msg.MessageId := INTF_MESSAGE_UPPER_STR;
  Msg.MsgText := 'Hello World';

  Hndlr.Dispatch(Msg);

  MessageDlg('Upper-cased string: ' + Msg.MsgText,
    mtInformation, [mbOK], 0);
```

```
  Hndlr.Free;
end;
```

See Also

System::TObject::DefaultHandler (see page 3367)

TMessage

35.1.22.4.11 TObject::Equals Method

Checks if the current instance and the parameter are equal.

Pascal

```
function Equals(Obj: TObject): Boolean; virtual;
```

C++

```
virtual Boolean Equals(TObject Obj);
```

Description

The function has one parameter, **Obj** of type TObject (see page 3354).

By default, the System::TObject::Equals method shows if the addresses corresponding to the object and the parameter are identical. Returns a boolean value that represents the equality between the two addresses.

Note: System::TObject::Equals is supposed to be overridden in user derived classes, in order to provide consumer objects with an equality determining function.

See Also

System::TObject::ClassType (see page 3364)

System::TObject::ClassName (see page 3359)

System::TObject::ClassParent (see page 3361)

System::TObject::FieldAddress (see page 3372)

System::TObject::MethodAddress (see page 3378)

System::TObject::InstanceSize (see page 3377)

System::TObject::MethodName (see page 3379)

System::TObject::InheritsFrom (see page 3376)

35.1.22.4.12 FieldAddress Method

This is the overview for the FieldAddress method overload.

Overload List

	Name	Description
	TObject::FieldAddress (ShortString) (see page 3372)	Returns the address of a published object field.
	TObject::FieldAddress (string) (see page 3372)	Returns the address of a published object field.

35.1.22.4.12.1 TObject::FieldAddress Method (ShortString)

Returns the address of a published object field.

Pascal

```
function FieldAddress(const Name: ShortString): Pointer; overload;
function FieldAddress(const Name: string): Pointer; overload;
```

C++

```
Pointer FieldAddress(const ShortString Name);
Pointer FieldAddress(const string Name);
```

Description

System::TObject::FieldAddress is used internally by the component streaming system to access a specified published field of an object. System::TObject::FieldAddress returns a pointer to the field if it exists. If the object has no published field by that name, System::TObject::FieldAddress returns nil (Delphi) or NULL (C++).

Programs should access and manipulate fields by using properties instead of System::TObject::FieldAddress.

35.1.22.4.13 TObject::Free Method

Destroys an object and frees its associated memory, if necessary.

Pascal

```
procedure Free;
```

C++

```
__fastcall Free();
```

Description

Use System::TObject::Free to destroy an object. System::TObject::Free automatically calls the destructor if the object reference is not nil. Any object instantiated at runtime that does not have an owner should be destroyed by a call to System::TObject::Free so that it can be properly disposed of and its memory released. Unlike System::TObject::Destroy (see page 3356), System::TObject::Free is successful even if the object is nil; so if the object was never initialized, System::TObject::Free won't result in an error.

When you call System::TObject::Free for a component, it calls System::TObject::Free for all components that it owns—that is, all components in its component list. Since a form owns all the controls and other components that are created on it in design mode, those components are automatically freed when the form is freed. By default, all forms are owned by the Application object; when the application terminates, it frees the Application object, which frees all forms. For objects that are not components, or for components created with a nil owner, be sure to call System::TObject::Free after you are finished with them; otherwise the allocated memory will not be usable until after the application terminates.

Warning: Never explicitly free a component within one of its own event handlers or the event handler of a component it owns or contains. For example, don't free a button, or the form that owns the button, in its OnClick event handler.

To free a form, call its _Release method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains are through executing.

Note: In C++ code, do not use System::TObject::Free to destroy an object. Use the delete keyword.

Delphi Examples:

```
{
This example demonstrates how to add and delete menu items
```

```
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
}

type
  TForm1 = class(TForm)
    AddButton: TButton;
    EditButton: TButton;
    DestroyButton: TButton;
    PopupMenu1: TPopupMenu;
    procedure AddButtonClick(Sender: TObject);
    procedure EditButtonClick(Sender: TObject);
    procedure DestroyButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    procedure PopupMenuItemsClick(Sender: TObject);

  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.AddButtonClick(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
    // The owner (PopupMenu1) will cleanup this menu item.
  NewItem := TMenuItem.Create(PopupMenu1); // create the new item
  index := PopupMenu1.Items.Count;
  PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
  NewItem.Caption := 'Menu Item ' + IntToStr(index);
  NewItem.Tag := index;
  NewItem.OnClick :=
    PopupMenuItemsClick; // assign it an event handler
end;

procedure TForm1.PopupMenuItemsClick(Sender: TObject);
begin
  with Sender as TMenuItem do
  begin
    case Tag of
      0:  ShowMessage('first item clicked');
      1:  ShowMessage('second item clicked');
      2:  ShowMessage('third item clicked');
      3:  ShowMessage('fourth item clicked');
    end;
  end;
end;

{
To edit or destroy an item, grab its pointer via the Items
property.
}
```

```
procedure TForm1.EditButtonClick(Sender: TObject);
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

System::TObject::Destroy (see page 3356)

Release

Owner

FreeAndNil (see page 4201)

35.1.22.4.14 TObject::FreeInstance Method

Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.

Pascal

```
procedure FreeInstance; virtual;
```

C++

```
virtual __fastcall FreeInstance();
```

Description

All destructors call System::TObject::FreeInstance automatically to deallocate memory that was allocated by overriding System::TObject::NewInstance (see page 3379).

Do not call System::TObject::FreeInstance directly. System::TObject::FreeInstance should be overridden if System::TObject::NewInstance (see page 3379) was overridden to change the way the object's instance data was allocated.

Like System::TObject::NewInstance (see page 3379), System::TObject::FreeInstance uses the value returned from System::TObject::InstanceSize (see page 3377) to deallocate the object's memory.

See Also

System::TObject::InstanceSize (see page 3377)

System::TObject::NewInstance (see page 3379)

35.1.22.4.15 TObject::GetHashCode Method

Returns an integer containing the hash code.

Pascal

```
function GetHashCode: Integer; virtual;
```

C++

```
virtual __fastcall Integer GetHashCode();
```

Description

By default, calling System::TObject::GetHashCode on an object, returns an integer representing the virtual address at which the object is stored.

Note: System::TObject::GetHashCode is supposed to be overridden in user derived classes, in order to provide consumer objects with an integer hash code representation.

See Also

System::TObject::ClassType (see page 3364)

System::TObject::ClassName (see page 3359)

System::TObject::ClassParent (see page 3361)

System::TObject::FieldAddress (see page 3372)

System::TObject::MethodAddress (see page 3378)

System::TObject::InstanceSize (see page 3377)

System::TObject::MethodName (see page 3379)

System::TObject::InheritsFrom (see page 3376)

35.1.22.4.16 TObject::GetInterface Method

Retrieves a specified interface.

Pascal

```
function GetInterface(const IID: TGUID; out Obj): Boolean;
```

C++

```
Boolean GetInterface(const TGUID IID, out  Obj);
```

Description

System::TObject::GetInterface retrieves the interface designated by a GUID or type name. The basic implementation of System::TObject::GetInterface uses the GUID specified in the IID parameter. If the specified interface is supported by the class, it is returned in the Obj parameter, and System::TObject::GetInterface has a return value of true. Otherwise Obj contains nil (Delphi) or NULL (C++), and System::TObject::GetInterface returns false.

Note: In Delphi code, IID can be an interface name. The compiler automatically translates this name into the corresponding GUID.

Note: In C++ code, use the templated version of System::TObject::GetInterface to obtain an interface from a DelphiInterface (see page 3256) object.

System::TObject::GetInterface is equivalent to the as operator (Delphi) and dynamic casts (C++), except that System::TObject::GetInterface does not raise an exception if the interface is not supported.

See Also

System::TInterfacedObject::QueryInterface (see page 3347)

35.1.22.4.17 TObject::GetInterfaceEntry Method

Returns the entry for a specific interface implemented in a class.

Pascal

```
class function GetInterfaceEntry(const IID: TGUID): PInterfaceEntry;
```

C++

```
PInterfaceEntry GetInterfaceEntry(const TGUID IID);
```

Description

System::TObject::GetInterfaceEntry returns the class entry for the interface specified by the IID parameter.

Note: In Delphi Code, IID can be an interface name. The compiler replaces this name with the actual GUID.

Note: COM objects can use System::TObject::GetInterfaceEntry to automate dispatch calls to a dual-IDispatch (see page 3494) interface.

35.1.22.4.18 TObject::GetInterfaceTable Method

Returns a pointer to a structure containing all of the interfaces implemented by a given class.

Pascal

```
class function GetInterfaceTable: PInterfaceTable;
```

C++

```
PInterfaceTable GetInterfaceTable();
```

Description

System::TObject::GetInterfaceTable returns the interface entries for the class. This list contains only interfaces implemented by this class, not its ancestors. To find the ancestor list, iteratively call System::TObject::ClassParent (see page 3361) and then call System::TObject::GetInterfaceTable on the value it returns. To find the entry for a specific interface, use the System::TObject::GetInterfaceEntry (see page 3376) method instead.

See Also

System::TObject::GetInterfaceEntry (see page 3376)

System::TObject::ClassParent (see page 3361)

35.1.22.4.19 TObject::InheritsFrom Method

Determines the relationship of two object types.

Pascal

```
class function InheritsFrom(AClass: TClass): Boolean;
```

C++

```
Boolean InheritsFrom(TClass AClass);
```

Description

Use System::TObject::InheritsFrom to determine if a particular class type or object is an instance of a class or one of its descendants. System::TObject::InheritsFrom returns true if the object type specified in the aClass parameter is an ancestor of the object type or the type of the object itself. Otherwise, it returns false.

Note: System::TObject::InheritsFrom is similar to the Delphi is operator, but applies to class references.

Note: In C++ code, a non-static version of System::TObject::InheritsFrom is provided. This call is useful in determining whether a descendant class method or property can be used, given a variable of a base class. For example, use System::TObject::InheritsFrom to determine whether the Sender parameter in an event handler is of a particular class type or one of its descendants.

See Also

System::TObject::ClassParent (see page 3361)

System::TObject::ClassNameIs (see page 3361)

35.1.22.4.20 TObject::InitInstance Method

Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

Pascal

```
class function InitInstance(Instance: Pointer): TObject;
```

C++

```
TObject InitInstance(Pointer Instance);
```

Description

There should be no reason to call System::TObject::InitInstance directly. System::TObject::InitInstance is called by System::TObject::NewInstance (see page 3379) when an object is created. When overriding System::TObject::NewInstance (see page 3379), be sure to call System::TObject::InitInstance as the last statement.

System::TObject::InitInstance is not virtual, so you cannot override it: Instead, initialize any data for an object in the constructor.

See Also

System::TObject::Create (see page 3355)

System::TObject::NewInstance (see page 3379)

35.1.22.4.21 TObject::InstanceSize Method

Returns the size in bytes of each instance of the object type.

Pascal

```
class function InstanceSize: Longint;
```

C++

```
Longint InstanceSize();
```

Description

System::TObject::InstanceSize indicates how many bytes of memory are required for a class's instance data. System::TObject::InstanceSize is called from methods that allocate and deallocate memory. System::TObject::InstanceSize is not a virtual method, so it cannot be overridden. System::TObject::InstanceSize should be called only when implementing a custom version of System::TObject::NewInstance (see page 3379).

See Also

System::TObject::FreeInstance (see page 3374)

System::TObject::NewInstance (see page 3379)

35.1.22.4.22 MethodAddress Method

This is the overview for the MethodAddress method overload.

Overload List

	Name	Description
	TObject::MethodAddress (ShortString) (see page 3378)	Returns the address of a class method by name.
	TObject::MethodAddress (string) (see page 3378)	Returns the address of a class method by name.

35.1.22.4.22.1 TObject::MethodAddress Method (ShortString)

Returns the address of a class method by name.

Pascal

```
class function MethodAddress(const Name: ShortString): Pointer; overload;
class function MethodAddress(const Name: string): Pointer; overload;
```

C++

```
Pointer MethodAddress(const ShortString Name);
Pointer MethodAddress(const string Name);
```

Description

There are situations where it is useful to be able to invoke an object method without the need to hard code the method name in advance. Call System::TObject::MethodAddress to dynamically retrieve the address of such a method by specifying the method **Name** as a string.

An easy way to invoke the method is to define a procedure or function data type, such as :

type TProc = procedure of object;

Assign the object name, and the System::TObject::MethodAddress to a TMethod (see page 3731) variable, such as :

MethodVar.Data := Pointer(ObjectInstanceName);

MethodVar.Code := ObjectInstanceName.System::TObject::MethodAddress('MethodNameString');

And pass this in a call to a variable of the procedure or function type :

Proc := TProc(MethodVar);

Proc;

See Also

TMethod (see page 3731)

35.1.22.4.23 TObject::MethodName Method

Returns the name of a class method by address.

Pascal

```
class function MethodName(Address: Pointer): string;
```

C++

```
string MethodName(Pointer Address);
```

Description

There are situations where it is useful to be able to invoke an object method without the need to hard code the method name in advance. Call System::TObject::MethodAddress (see page 3378) to dynamically retrieve the address of such a method by specifying the method name as a string.

System::TObject::MethodName is the converse to this process - by supplying a method **Address**, the name of the method is returned as a string.

35.1.22.4.24 TObject::NewInstance Method

Allocates memory for an instance of an object type and returns a pointer to that new instance.

Pascal

```
class function NewInstance: TObject; virtual;
```

C++

```
virtual TObject NewInstance();
```

Description

All constructors call System::TObject::NewInstance automatically. System::TObject::NewInstance calls System::TObject::InstanceSize (see page 3377) to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TObject::NewInstance directly.

Override System::TObject::NewInstance only for special memory allocation requirements. For example, when allocating a large number of identical objects that all need to be in memory at the same time, you could allocate a single block of memory for the entire group, then override System::TObject::NewInstance to use part of that larger block for each instance.

If you override System::TObject::NewInstance to allocate memory, you may need to override System::TObject::FreeInstance (see page 3374) to deallocate the memory.

Note: By default System::TObject::NewInstance calls System::TObject::InitInstance (see page 3377).

See Also

System::TObject::FreeInstance (see page 3374)

System::TObject::InstanceSize (see page 3377)

System::TObject::InitInstance (see page 3377)

35.1.22.4.25 TObject::SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; virtual;
```

C++

```
virtual HResult SafeCallException(TObject ExceptObject, Pointer ExceptAddr);
```

Description

System::TObject::SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur.

As implemented in System::TObject (see page 3354), System::TObject::SafeCallException simply returns E_UNEXPECTED (see page 3846). This is the appropriate response for classes that do no support interfaces

35.1.22.4.26 TObject::ToString Method

Returns a string containing the class name.

Pascal

```
function ToString: string; virtual;
```

C++

```
virtual __fastcall string ToString();
```

Description

By default, the System::TObject::ToString returns a string containing the class name of the instance that is being called.

Example

Calling System::TObject::ToString on a TButton instance returns a string containing "TButton".

Note: System::TObject::ToString is intended to be overridden in user derived classes, in order to provide consumer objects with a string representation.

See Also

System::TObject::ClassType (see page 3364)

System::TObject::ClassName (see page 3359)

System::TObject::ClassParent (see page 3361)

System::TObject::FieldAddress (see page 3372)

System::TObject::MethodAddress (see page 3378)

System::TObject::InstanceSize (see page 3377)

System::TObject::MethodName (see page 3379)

System::TObject::InheritsFrom (see page 3376)

35.1.22.4.27 TObject::UnitName Method

Returns a string containing the unit name.

Pascal

```
class function UnitName: string;
```

C++

```
string UnitName();
```

Description

By default, calling System::TObject::UnitName on an object, returns a string representing the unit name where it is contained.

Example

Calling System::TObject::UnitName on a TButton instance returns a string containing "StdCtrls (see page 2033)".

See Also

System::TObject::ClassType (see page 3364)

System::TObject::ClassName (see page 3359)

System::TObject::ClassParent (see page 3361)

System::TObject::FieldAddress (see page 3372)

System::TObject::MethodAddress (see page 3378)

System::TObject::InstanceSize (see page 3377)

System::TObject::MethodName (see page 3379)

System::TObject::InheritsFrom (see page 3376)

35.1.23 UnicodeString Class

System::UnicodeString is the C++ analog for the Delphi UnicodeString type.

Class Hierarchy

Pascal

```
UnicodeString = class;
```

C++

```
class UnicodeString;
```

File

ustring.h

Description

Delphi utilizes several string types. System::UnicodeString can contain both Unicode and ANSI strings. Support for this type includes the following features:

- strings as large as available memory;
- efficient use of memory through shared references;
- routines and operators that evaluate strings based on the current locale.

Despite its name, System::UnicodeString can represent both ANSI character set strings and Unicode strings.

System::UnicodeString variables that have not been assigned an initial value contain a zero-length string.

Note: Delphi also supports System::UnicodeString, but implements it as a primitive type rather than a class. By default, variables declared as type String are System::UnicodeString.

Note: To be compatible with Delphi, System::UnicodeString indexing is 1-based.

See Also

System::AnsiStringBase (see page 3174)

System::WideString (see page 3467)

System::UnicodeString::[] (see page 3414)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

System::UnicodeString::+= (see page 3416)

35.1.23.1 UnicodeString Members

The following tables list the members exposed by UnicodeString.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3388)	System::UnicodeString::TStringFloatFormat is an enumerated list of formatting codes for float functions.
	TStringLeadCharType (see page 3389)	Enumerated list of element types in a Unicode string.

Structures

	Name	Description
	StrRec (see page 3387)	Provides information about the Unicode string.

Protected Methods

	Name	Description
	GetRec (see page 3401)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3410)	Throws exception if index is out of range in System::UnicodeString (see page 3381).

Public Friends

Name	Description
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) (see page 3420)	This is friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs).
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) (see page 3420)	This is friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs).

Public Methods

	Name	Description
	~UnicodeString (see page 3384)	Frees an instance of System::UnicodeString (see page 3381).
	BytesOf (see page 3391)	Get string's data (see page 3396) in bytes.
	ByteType (see page 3391)	Indicates the kind of byte that occupies specified position in System::UnicodeString (see page 3381).
	c_str (see page 3392)	Returns a pointer to the underlying string System::UnicodeString::data (see page 3396) as const wchar_t*.
	cat_printf (see page 3392)	Appends to value of System::UnicodeString (see page 3381), given format string and its arguments.
	cat_sprintf (see page 3393)	Appends result of given format string and its arguments to end of System::UnicodeString (see page 3381).
	cat_vprintf (see page 3394)	Appends the result of given format string and its arguments to end of System::UnicodeString (see page 3381).
	CodePage (see page 3394)	Gets code page of a System::UnicodeString (see page 3381).
	Compare (see page 3394)	Compares the System::UnicodeString (see page 3381) to another specified string case-sensitively.
	CompareIC (see page 3395)	Compares System::UnicodeString (see page 3381) to another specified string case insensitively.
	CurrToStr (see page 3395)	Converts a currency value to its string representation.
	CurrToStrF (see page 3396)	Converts currency value to its string representation using indicated float format convention.
	data (see page 3396)	Returns a pointer to the underlying string System::UnicodeString::data.
	Delete (see page 3396)	Removes specified number of characters from string.
	ElementSize (see page 3397)	Gets element size of data (see page 3396) in a System::UnicodeString (see page 3381).
	EnsureUnicode (see page 3397)	Ensure that string contains Unicode data (see page 3396).
	FloatToStrF (see page 3397)	Returns string representation of floating-point value.
	FmtLoadStr (see page 3398)	Formats output using a resourced format string.
	Format (see page 3399)	Formats string assembled from System::UnicodeString (see page 3381), format, and array arguments.
	FormatFloat (see page 3399)	Formats floating-point value given by `value` using format string given by `format`.
	Insert (see page 3401)	Inserts specified string into System::UnicodeString (see page 3381).
	IntToHex (see page 3402)	Converts number into string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3402)	Returns true if the specified character matches any character in `delimiters` string.
	IsEmpty (see page 3402)	Indicates if System::UnicodeString (see page 3381) is empty.
	IsLeadSurrogate (see page 3403)	Tests if element is lead surrogate.
	IsPathDelimiter (see page 3403)	Indicates if specified character is a path delimiter.
	IsTrailSurrogate (see page 3403)	Tests if element is trail surrogate.
	LastChar (see page 3404)	Returns pointer to last character in System::UnicodeString (see page 3381).
	LastDelimiter (see page 3404)	Returns index of rightmost character that matches any character in `delimiters` string.
	Length (see page 3404)	Returns length in characters of System::UnicodeString (see page 3381).
	LoadStr (see page 3405)	This is the overview for the LoadStr method overload.
	LoadString (see page 3405)	Loads string from compiled resources of specified module.
	LowerCase (see page 3406)	Returns lower-case version of System::UnicodeString (see page 3381).
	Pos (see page 3406)	Returns character index at which specified substring begins.
	printf (see page 3406)	Sets value of System::UnicodeString (see page 3381) instance, given format string and its arguments.
	RefCount (see page 3407)	Gets reference count of System::UnicodeString (see page 3381).
	SetLength (see page 3407)	Changes character length of System::UnicodeString (see page 3381).
	sprintf (see page 3408)	Sets value of System::UnicodeString (see page 3381), given format string and its arguments.
	StringOfChar (see page 3408)	Returns string with specified number of characters.
	SubString (see page 3408)	Returns specified substring of System::UnicodeString (see page 3381).
	swap (see page 3409)	Swaps System::UnicodeString (see page 3381) with another string.

	t_str (see page 3409)	Returns System::UnicodeString::UnicodeString (see page 3381) data (see page 3396) as const char*.
	ToDouble (see page 3410)	Converts string to floating-point value.
	ToInt (see page 3410)	Converts string to integer, throwing exception if string is invalid.
	ToIntDef (see page 3411)	Converts string to integer, returning default value if string is invalid.
	Trim (see page 3411)	Returns new System::UnicodeString (see page 3381) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3411)	Returns new System::UnicodeString (see page 3381) without leading spaces or control characters.
	TrimRight (see page 3412)	Returns new System::UnicodeString (see page 3381) without trailing spaces or control characters.
	UnicodeString (see page 3384)	This is the overview for the UnicodeString constructor overload.
	Unique (see page 3412)	Makes System::UnicodeString (see page 3381) instance unique.
	UpperCase (see page 3412)	Returns upper-case version of System::UnicodeString (see page 3381).
	vprintf (see page 3413)	Sets value of System::UnicodeString (see page 3381) given format string and its arguments.
	w_str (see page 3413)	Returns System::UnicodeString::UnicodeString (see page 3381) data (see page 3396).

Public Operators

	Name	Description
	!= (see page 3414)	Compares two strings.
	[] (see page 3414)	This is the overview for the [] operator overload.
	+ (see page 3415)	Concatenates two System::UnicodeString (see page 3381) values.
	+= (see page 3416)	Append (see page 3520) System::UnicodeString (see page 3381).
	< (see page 3416)	Compares two strings.
	<= (see page 3417)	Compares two strings.
	= (see page 3418)	Assigns string `rhs` to System::UnicodeString (see page 3381).
	== (see page 3418)	Compares System::UnicodeString (see page 3381) with specified string.
	> (see page 3419)	Compares two strings.
	>= (see page 3419)	Compares two strings.

35.1.23.2 UnicodeString::~UnicodeString Destructor

Frees an instance of System::UnicodeString (see page 3381).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~UnicodeString();
```

Description

System::UnicodeString (see page 3381)::~UnicodeString (see page 3381) is the destructor for System::UnicodeString (see page 3381).

35.1.23.3 UnicodeString Constructor

This is the overview for the UnicodeString constructor overload.

Overload List

	Name	Description
	UnicodeString::UnicodeString () (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (__int64) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (AnsiStringT<codePage> &) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (char) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (char*) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (char*, int) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (char16_t*, int) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (char32_t*, int) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (double) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (int) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (long) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (short) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (UnicodeString&) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (unsigned __int64) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (unsigned int) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (unsigned long) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (unsigned short) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (wchar_t) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (wchar_t*) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (wchar_t*, int) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).
	UnicodeString::UnicodeString (WideString &) (see page 3386)	Creates instance of System::UnicodeString (see page 3381).

35.1.23.3.1 UnicodeString::UnicodeString Constructor ()

Creates instance of System::UnicodeString (see page 3381).

Pascal

```
constructor Create;
constructor Create(const src: char*);
constructor Create(const src: UnicodeString&);
constructor Create(const src: wchar_t*; len: Integer);
constructor Create(const src: char*; len: Integer);
constructor Create(const src: wchar_t*);
constructor Create(const src: char16_t*; numChar16: Integer = -1);
constructor Create(const src: char32_t*; numChar32: Integer = -1);
constructor Create(src: char);
constructor Create(src: wchar_t);
constructor Create(src: short);
constructor Create(src: unsigned short);
constructor Create(src: Integer);
constructor Create(src: unsigned int);
constructor Create(src: long);
constructor Create(src: unsigned long);
constructor Create(src: __int64);
constructor Create(src: unsigned __int64);
constructor Create(src: double);
constructor Create(const src: WideString &);
constructor Create<unsigned short codePage>(const src: AnsiStringT<codePage> &);
```

C++

```
__fastcall UnicodeString();
__fastcall UnicodeString(const char* src);
__fastcall UnicodeString(const UnicodeString& src);
__fastcall UnicodeString(const wchar_t* src, int len);
__fastcall UnicodeString(const char* src, int len);
__fastcall UnicodeString(const wchar_t* src);
__fastcall UnicodeString(const char16_t* src, int numChar16 = -1);
__fastcall UnicodeString(const char32_t* src, int numChar32 = -1);
__fastcall UnicodeString(char src);
__fastcall UnicodeString(wchar_t src);
__fastcall UnicodeString(short src);
__fastcall UnicodeString(unsigned short src);
__fastcall UnicodeString(int src);
__fastcall UnicodeString(unsigned int src);
__fastcall UnicodeString(long src);
__fastcall UnicodeString(unsigned long src);
__fastcall UnicodeString(__int64 src);
__fastcall UnicodeString(unsigned __int64 src);
__fastcall UnicodeString(double src);
__fastcall UnicodeString(const WideString & src);
template <unsigned short codePage> __fastcall UnicodeString(const AnsiStringT<codePage> & src);
```

Description

Call System::UnicodeString::UnicodeString (see page 3381) to create and initialize a System::UnicodeString (see page 3381) instance. If the constructor is given no arguments, the resulting System::UnicodeString (see page 3381) instance is an empty string. If the constructor is passed a single character, the resulting System::UnicodeString (see page 3381) instance has character length one, and contains that character. All other versions of the constructor copy the string passed as an argument, converting characters to `wchar_t` if necessary. When constructing a System::UnicodeString (see page 3381) instance from an array of `char` or `wchar_t`, the `src` string should be null terminated, unless a second argument is given that indicates the number of characters in the array.

35.1.23.4 UnicodeString Structures

The structures of the UnicodeString class are listed here.

Structures

	Name	Description
	StrRec (see page 3387)	Provides information about the Unicode string.

35.1.23.4.1 System::UnicodeString::StrRec Structure

Provides information about the Unicode string.

Pascal

```
StrRec = record
  codePage: unsigned short;
  elemSize: unsigned short;
  refCnt: Integer;
  length: Integer;
end;
```

C++

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

File

ustring.h

Description

s

Use System::UnicodeString::StrRec to retrieve information about the Unicode string. The fields of System::UnicodeString::StrRec and their meaning are given in the following table.

Field	Meaning
codePage	The code page of the Unicode string
elemSize	The number of bytes per character of the Unicode string
refCnt	The number of references to the Unicode string
length	The number of characters of the Unicode string

See Also

System::UnicodeString (see page 3381)

35.1.23.5 UnicodeString Enumerations

The enumerations of the UnicodeString class are listed here.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3388)	System::UnicodeString::TStringFloatFormat is an enumerated list of formatting codes for float functions.
	TStringLeadCharType (see page 3389)	Enumerated list of element types in a Unicode string.

35.1.23.5.1 System::UnicodeString::TStringFloatFormat Enumeration

System::UnicodeString::TStringFloatFormat is an enumerated list of formatting codes for float functions.

Pascal

```
TStringFloatFormat = (
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
);
```

C++

```
enum TStringFloatFormat {
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
};
```

File

ustring.h

Description

System::UnicodeString::TStringFloatFormat defines an enumerated list of formatting codes for the float functions.

Format (see page 3399)	Defines
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.

sffCurrency	Currency (see page 3235) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, and DecimalSeparator global variables, all of which are initialized from the Currency (see page 3235) Format (see page 3399) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18.

See Also

System::UnicodeString::FloatToStrF (see page 3397)

35.1.23.5.2 System::UnicodeString::TStringLeadCharType Enumeration

Enumerated list of element types in a Unicode string.

Pascal

```
TStringLeadCharType = (
  ctNotLeadChar,
  ctbLeadSurrogate,
  ctTrailSurrogate
);
```

C++

```
enum TStringLeadCharType {
  ctNotLeadChar,
  ctbLeadSurrogate,
  ctTrailSurrogate
};
```

File

ustring.h

Description

System::UnicodeString::TStringLeadCharType defines an enumerated list of element types in a Unicode string.

Constant	Meaning
ctNotLeadChar	Not a lead element.
ctbLeadSurrogate	Lead surrogate pair element.
ctTrailSurrogate	Trailing surrogate pair element.

See Also

System::UnicodeString::ByteType (see page 3391)

35.1.23.6 UnicodeString Methods

The methods of the UnicodeString class are listed here.

Protected Methods

	Name	Description
	GetRec (see page 3401)	This is the overview for the GetRec method overload.
	ThrowIfOutOfRange (see page 3410)	Throws exception if index is out of range in System::UnicodeString (see page 3381).

Public Methods

	Name	Description
	BytesOf (see page 3391)	Get string's data (see page 3396) in bytes.
	ByteType (see page 3391)	Indicates the kind of byte that occupies specified position in System::UnicodeString (see page 3381).
	c_str (see page 3392)	Returns a pointer to the underlying string System::UnicodeString::data (see page 3396) as const wchar_t*.
	cat_printf (see page 3392)	Appends to value of System::UnicodeString (see page 3381), given format string and its arguments.
	cat_sprintf (see page 3393)	Appends result of given format string and its arguments to end of System::UnicodeString (see page 3381).
	cat_vprintf (see page 3394)	Appends the result of given format string and its arguments to end of System::UnicodeString (see page 3381).
	CodePage (see page 3394)	Gets code page of a System::UnicodeString (see page 3381).
	Compare (see page 3394)	Compares the System::UnicodeString (see page 3381) to another specified string case-sensitively.
	CompareIC (see page 3395)	Compares System::UnicodeString (see page 3381) to another specified string case insensitively.
	CurrToStr (see page 3395)	Converts a currency value to its string representation.
	CurrToStrF (see page 3396)	Converts currency value to its string representation using indicated float format convention.
	data (see page 3396)	Returns a pointer to the underlying string System::UnicodeString::data.
	Delete (see page 3396)	Removes specified number of characters from string.
	ElementSize (see page 3397)	Gets element size of data (see page 3396) in a System::UnicodeString (see page 3381).
	EnsureUnicode (see page 3397)	Ensure that string contains Unicode data (see page 3396).
	FloatToStrF (see page 3397)	Returns string representation of floating-point value.
	FmtLoadStr (see page 3398)	Formats output using a resourced format string.
	Format (see page 3399)	Formats string assembled from System::UnicodeString (see page 3381), format, and array arguments.
	FormatFloat (see page 3399)	Formats floating-point value given by `value` using format string given by `format`.
	Insert (see page 3401)	Inserts specified string into System::UnicodeString (see page 3381).
	IntToHex (see page 3402)	Converts number into string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3402)	Returns true if the specified character matches any character in `delimiters` string.
	IsEmpty (see page 3402)	Indicates if System::UnicodeString (see page 3381) is empty.
	IsLeadSurrogate (see page 3403)	Tests if element is lead surrogate.
	IsPathDelimiter (see page 3403)	Indicates if specified character is a path delimiter.
	IsTrailSurrogate (see page 3403)	Tests if element is trail surrogate.
	LastChar (see page 3404)	Returns pointer to last character in System::UnicodeString (see page 3381).
	LastDelimiter (see page 3404)	Returns index of rightmost character that matches any character in `delimiters` string.
	Length (see page 3404)	Returns length in characters of System::UnicodeString (see page 3381).
	LoadStr (see page 3405)	This is the overview for the LoadStr method overload.
	LoadString (see page 3405)	Loads string from compiled resources of specified module.
	LowerCase (see page 3406)	Returns lower-case version of System::UnicodeString (see page 3381).
	Pos (see page 3406)	Returns character index at which specified substring begins.
	printf (see page 3406)	Sets value of System::UnicodeString (see page 3381) instance, given format string and its arguments.
	RefCount (see page 3407)	Gets reference count of System::UnicodeString (see page 3381).
	SetLength (see page 3407)	Changes character length of System::UnicodeString (see page 3381).
	sprintf (see page 3408)	Sets value of System::UnicodeString (see page 3381), given format string and its arguments.
	StringOfChar (see page 3408)	Returns string with specified number of characters.
	SubString (see page 3408)	Returns specified substring of System::UnicodeString (see page 3381).
	swap (see page 3409)	Swaps System::UnicodeString (see page 3381) with another string.

	t_str (see page 3409)	Returns System::UnicodeString::UnicodeString (see page 3381) data (see page 3396) as const char*.
	ToDouble (see page 3410)	Converts string to floating-point value.
	ToInt (see page 3410)	Converts string to integer, throwing exception if string is invalid.
	ToIntDef (see page 3411)	Converts string to integer, returning default value if string is invalid.
	Trim (see page 3411)	Returns new System::UnicodeString (see page 3381) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3411)	Returns new System::UnicodeString (see page 3381) without leading spaces or control characters.
	TrimRight (see page 3412)	Returns new System::UnicodeString (see page 3381) without trailing spaces or control characters.
	Unique (see page 3412)	Makes System::UnicodeString (see page 3381) instance unique.
	UpperCase (see page 3412)	Returns upper-case version of System::UnicodeString (see page 3381).
	vprintf (see page 3413)	Sets value of System::UnicodeString (see page 3381) given format string and its arguments.
	w_str (see page 3413)	Returns System::UnicodeString::UnicodeString (see page 3381) data (see page 3396).

35.1.23.6.1 UnicodeString::BytesOf Method

Get string's data (see page 3396) in bytes.

Pascal

```
function BytesOf: DynamicArray<System::Byte>;
```

C++

```
__fastcall DynamicArray<System::Byte> BytesOf() const;
```

Description

System::UnicodeString::BytesOf gets a byte array copy of a System::UnicodeString (see page 3381)'s data (see page 3396). The data (see page 3396) is copied as is, with no encoding or decoding.

See Also

System::AnsiStringBase::BytesOf (see page 3186)

BytesOf (see page 4068)

35.1.23.6.2 UnicodeString::ByteType Method

Indicates the kind of byte that occupies specified position in System::UnicodeString (see page 3381).

Pascal

```
function ByteType(index: Integer): TStringLeadCharType;
```

C++

```
__fastcall TStringLeadCharType ByteType(int index) const;
```

Description

System::UnicodeString::ByteType indicates what kind of byte exists at the byte indicated by `index` (counting from 1) in the System::UnicodeString (see page 3381). Note that **index** is an element index into the string, not a character or byte index. These are the possible return values.

Constant	Meaning
ctNotLeadChar	Not a lead character.
ctbLeadSurrogate	Lead surrogate pair character.

ctTrailSurrogate	Trailing surrogate pair character.

Note: No checking is done to ensure that `index` is less than the length of the string. It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

LeadBytes (see page 4354)

StrByteType (see page 4242)

35.1.23.6.3 UnicodeString::c_str Method

Returns a pointer to the underlying string System::UnicodeString::data (see page 3396) as const wchar_t*.

Pascal

```
function c_str: wchar_t*;
```

C++

```
__fastcall wchar_t* c_str() const;
```

Description

System::UnicodeString::c_str returns a **wchar_t** pointer to the location in memory where the value of the System::UnicodeString (see page 3381) object is stored. If the System::UnicodeString (see page 3381) is unassigned, System::UnicodeString::c_str returns a **wchar_t** pointer to the empty string (“”).

Usually, the value returned by System::UnicodeString::c_str points to the internal character array referenced by the System::UnicodeString::data (see page 3396) function. This pointer is valid until the System::UnicodeString (see page 3381) is next modified (for example, when the System::UnicodeString::SetLength (see page 3407) method is called or the System::UnicodeString (see page 3381) goes out of scope). However, if the internal array is NULL, System::UnicodeString::c_str returns a **wchar_t** pointer to the empty string (“”).

The System::UnicodeString::c_str method is intended primarily for reading the value of the System::UnicodeString (see page 3381). To modify the System::UnicodeString (see page 3381)’s value, use the [] operator or System::UnicodeString (see page 3381) methods such as System::UnicodeString::Insert (see page 3401) and System::UnicodeString::Delete (see page 3396).

The System::UnicodeString::t_str (see page 3409) method returns **const char*** for backward compatiblity with the VCL and to preserve narrow data (see page 3396).

See Also

System::UnicodeString::data (see page 3396)

System::UnicodeString::Delete (see page 3396)

System::UnicodeString::Insert (see page 3401)

System::UnicodeString::SetLength (see page 3407)

System::UnicodeString::t_str (see page 3409)

35.1.23.6.4 UnicodeString::cat_printf Method

Appends to value of System::UnicodeString (see page 3381), given format string and its arguments.

Pascal

```
function cat_printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int cat_printf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::cat_printf to modify the value of the System::UnicodeString (see page 3381) by appending the result of using a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the number of characters appended to the System::UnicodeString (see page 3381).

See Also

System::UnicodeString::cat_sprintf (see page 3393)

System::UnicodeString::cat_vprintf (see page 3394)

System::UnicodeString::Format (see page 3399)

System::UnicodeString::printf (see page 3406)

35.1.23.6.5 UnicodeString::cat_sprintf Method

Appends result of given format string and its arguments to end of System::UnicodeString (see page 3381).

Pascal

```
function cat_sprintf(const format: wchar_t*; : ...): UnicodeString&; cdecl;
```

C++

```
__cdecl UnicodeString& cat_sprintf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::cat_sprintf to append to the value of the System::UnicodeString (see page 3381) object, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

For example, the following code

```
UnicodeString Salutation = "Mr."
UnicodeString s = "";
s.cat_sprintf("Dear %s ", Salutation);
s.cat_sprintf("%s,", "Smith");
```

results in an System::UnicodeString (see page 3381) object with the value

```
Dear Mr. Smith,
```

This method returns a reference to the System::UnicodeString (see page 3381) object (*this) with the resulting value.

See Also

System::UnicodeString::cat_printf (see page 3392)

System::UnicodeString::cat_vprintf (see page 3394)

System::UnicodeString::Format (see page 3399)

System::UnicodeString::sprintf (see page 3408)

35.1.23.6.6 UnicodeString::cat_vprintf Method

Appends the result of given format string and its arguments to end of System::UnicodeString (see page 3381).

Pascal

```
function cat_vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(const wchar_t* format, va_list);
```

Description

Use System::UnicodeString::cat_vprintf to append to the value of the System::UnicodeString (see page 3381) object, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using `va_start`.

This method returns the number of characters appended to the System::UnicodeString (see page 3381) object.

See Also

System::UnicodeString::Format (see page 3399)

System::UnicodeString::cat_printf (see page 3392)

System::UnicodeString::cat_sprintf (see page 3393)

System::UnicodeString::vprintf (see page 3413)

35.1.23.6.7 UnicodeString::CodePage Method

Gets code page of a System::UnicodeString (see page 3381).

Pascal

```
function CodePage: unsigned short;
```

C++

```
unsigned short CodePage() const;
```

Description

System::UnicodeString::CodePage gets the code page of a System::UnicodeString (see page 3381) instance. If the instance is empty, it returns the system default Unicode code page.

See Also

System::UnicodeString::GetRec (see page 3401)

35.1.23.6.8 UnicodeString::Compare Method

Compares the System::UnicodeString (see page 3381) to another specified string case-sensitively.

Pascal

```
function Compare(const rhs: UnicodeString&): Integer;
```

C++

```
__fastcall int Compare(const UnicodeString& rhs) const;
```

Description

Compares a System::UnicodeString (see page 3381) instance to `rhs`, with case sensitivity. Use

System::UnicodeString::CompareIC (see page 3395) to compare without case sensitivity. The compare operation should be the same as using the System::UnicodeString (see page 3381) comparison operators. System::UnicodeString::Compare is based on a locale-specific collating order. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

See Also

System::UnicodeString::CompareIC (see page 3395)

35.1.23.6.9 UnicodeString::CompareIC Method

Compares System::UnicodeString (see page 3381) to another specified string case insensitively.

Pascal

```
function CompareIC(const rhs: UnicodeString&): Integer;
```

C++

```
__fastcall int CompareIC(const UnicodeString& rhs) const;
```

Description

System::UnicodeString::CompareIC compares a System::UnicodeString (see page 3381) instance to `rhs`, without case sensitivity. Use System::UnicodeString::Compare (see page 3394) to compare with case sensitivity. The compare operation is controlled by the current Windows locale. The return value is shown below:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

See Also

System::UnicodeString::Compare (see page 3394)

35.1.23.6.10 UnicodeString::CurrToStr Method

Converts a currency value to its string representation.

Pascal

```
function CurrToStr(value: Currency): String; static;
```

C++

```
static __fastcall UnicodeString CurrToStr(Currency value);
```

Description

System::UnicodeString::CurrToStr converts the currency value given by `value` to its string representation. The conversion uses the sffGeneral floating-point conversion rules (see System::UnicodeString::FloatToStrF (see page 3397)). For further details, see System::UnicodeString::CurrToStrF (see page 3396).

See Also

System::UnicodeString::CurrToStrF (see page 3396)

System::UnicodeString::FloatToStrF (see page 3397)

35.1.23.6.11 UnicodeString::CurrToStrF Method

Converts currency value to its string representation using indicated float format convention.

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer): String;
static;
```

C++

```
static __fastcall UnicodeString CurrToStrF(Currency value, TStringFloatFormat format, int
digits);
```

Description

System::UnicodeString::CurrToStrF converts the currency value given by `value` to its string representation using the indicated float format convention. A call to System::UnicodeString::CurrToStrF corresponds to a call to System::UnicodeString::FloatToStrF (see page 3397) with an implied precision of 19 digits.

See Also

System::UnicodeString::FloatToStrF (see page 3397)

35.1.23.6.12 UnicodeString::data Method

Returns a pointer to the underlying string System::UnicodeString::data.

Pascal

```
function data: const void*;
```

C++

```
__fastcall const void* data() const;
```

Description

System::UnicodeString::data returns a **wchar_t** pointer to the System::UnicodeString (see page 3381) object that it uses to store its value. This is the same value as the one returned by the System::UnicodeString::c_str (see page 3392) method, except that System::UnicodeString::c_str (see page 3392) returns a pointer to an empty string when the string is unassigned.

See Also

System::UnicodeString::c_str (see page 3392)

35.1.23.6.13 UnicodeString::Delete Method

Removes specified number of characters from string.

Pascal

```
function Delete(index: Integer; count: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& Delete(int index, int count);
```

Description

System::UnicodeString::Delete modifies the System::UnicodeString (see page 3381) object to remove `count` characters from the string beginning with the character at `index`, where 1 is the index of the first character. It returns the resulting modified string (*this).

If `index` is larger than the character length of the System::UnicodeString (see page 3381) object or less than 1, no characters are deleted.

If `count` specifies more characters than remain starting at the `index`, System::UnicodeString::Delete removes the rest of the string. If `count` is less than 0, no characters are deleted.

35.1.23.6.14 UnicodeString::ElementSize Method

Gets element size of data (see page 3396) in a System::UnicodeString (see page 3381).

Pascal

```
function ElementSize: unsigned short;
```

C++

```
unsigned short ElementSize() const;
```

Description

System::UnicodeString::ElementSize gets the element size in bytes of the data (see page 3396) in a System::UnicodeString (see page 3381) instance. If the instance is empty, this returns 2.

See Also

System::UnicodeString::data (see page 3396)

System::UnicodeString::GetRec (see page 3401)

35.1.23.6.15 UnicodeString::EnsureUnicode Method

Ensure that string contains Unicode data (see page 3396).

Pascal

```
function EnsureUnicode: UnicodeString&;
```

C++

```
__fastcall UnicodeString& EnsureUnicode();
```

Description

If the string data (see page 3396) in the System::UnicodeString (see page 3381) instance is not Unicode, System::UnicodeString::EnsureUnicode converts it to Unicode.

35.1.23.6.16 UnicodeString::FloatToStrF Method

Returns string representation of floating-point value.

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer): String; static;
```

C++

```
static __fastcall UnicodeString FloatToStrF(long double value, TStringFloatFormat format, int
```

```
precision, int digits);
```

Description

The `format` parameter controls the format of the resulting string. The `precision` parameter specifies the precision of the given value. It should be 7 or less for values of type float, 15 or less for values of type double, and 18 or less for values of type Extended. The meaning of the `digits` parameter depends on the particular format selected.

The possible values of the format parameter, and the meaning of each, are described below.

Value	Meaning
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the `digits` parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 3235) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the `CurrencyString`, `CurrencyFormat`, `NegCurrFormat`, `ThousandSeparator`, and `DecimalSeparator` global variables, all of which are initialized from the Currency (see page 3235) Format (see page 3399) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the `Digits` parameter--it must be between 0 and 18.

For all formats, the actual characters used as decimal and thousand separators are obtained from the `DecimalSeparator` and `ThousandSeparator` global variables.

If the given value is NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

35.1.23.6.17 UnicodeString::FmtLoadStr Method

Formats output using a resourced format string.

Pascal

```
function FmtLoadStr(ident: Integer; const args: TVarRec *; size: Integer): String; static;
```

C++

```
static __fastcall UnicodeString FmtLoadStr(int ident, const TVarRec * args, int size);
```

Description

System::UnicodeString::FmtLoadStr loads a string from a program's resource string table and uses that string, plus the `args` array, as a parameter to the System::UnicodeString::Format (see page 3399) function. `Ident` is the string resource ID of the desired format string.

See Also

System::UnicodeString::Format (see page 3399)

System::UnicodeString::LoadStr (see page 3405)

System::UnicodeString::LoadString (see page 3405)

35.1.23.6.18 UnicodeString::Format Method

Formats string assembled from System::UnicodeString (see page 3381), format, and array arguments.

Pascal

```
function Format(const format: UnicodeString&; const args: TVarRec *; size: Integer): String;
static;
```

C++

```
static __fastcall UnicodeString Format(const UnicodeString& format, const TVarRec * args, int
size);
```

Description

Use System::UnicodeString::Format to obtain a System::UnicodeString (see page 3381) object whose value is the result of applying an array of arguments (`args`) to a given standard C++ format specifier string (`format`). The last parameter (`size`) is the 1-based index of the last element in `args` (one less than the number of arguments).

Note that System::UnicodeString::Format is a static method. The returned value is a newly allocated System::UnicodeString (see page 3381), even if the method is called with an existing System::UnicodeString (see page 3381) instance.

See Also

System::UnicodeString::printf (see page 3406)

System::UnicodeString::sprintf (see page 3408)

35.1.23.6.19 UnicodeString::FormatFloat Method

Formats floating-point value given by `value` using format string given by `format`.

Pascal

```
function FormatFloat(const format: UnicodeString&; const value: long double&): String; static;
```

C++

```
static __fastcall UnicodeString FormatFloat(const UnicodeString& format, const long double&
value);
```

Description

The following format specifiers are supported in the `format` string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.

.	Decimal point. The first dot character ('.') in the format string determines the location of the decimal separator in the formatted value; any additional dot characters are ignored. The actual character used as the decimal separator in the output string is determined by the `DecimalSeparator` global variable, whose default value is specified in the Number Format (see page 3399) of the International section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more comma characters (','), the output has thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of comma characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as the thousand separator in the output is determined by the `ThousandSeparator` global variable, whose default value is specified in the Number Format (see page 3399) of the International section in the Windows Control Panel.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain from one to three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If no formatting is specified for positive values, positive values are formatted using general floating-point formatting with 15 significant digits, corresponding to a call to System::UnicodeString::FloatToStrF (see page 3397) with the sffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The following table shows some sample formats and the results produced when the formats are applied to different values:

Format (see page 3399) string-1234-12340.50

Format (see page 3399) string-	**1234**	**-1234**	**0.5**	**0**
1234	-1234	0.5	0	
0	1234	-1234	1	
0.00	1234.00	-1234.00	0.50	
#	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	

#,##0.00;;Zero	1,234.00	-1,234.00	0.50	
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	
#.###E-0	1.234E3	-1.234E3	5E-1	

See Also

System::UnicodeString::FloatToStrF (see page 3397)

35.1.23.6.20 GetRec Method

This is the overview for the GetRec method overload.

Overload List

	Name	Description
	UnicodeString::GetRec () (see page 3401)	Gets a struct describing a System::UnicodeString (see page 3381).
	UnicodeString::GetRec () (see page 3401)	Gets a struct describing a System::UnicodeString (see page 3381).

35.1.23.6.20.1 UnicodeString::GetRec Method ()

Gets a struct describing a System::UnicodeString (see page 3381).

Pascal

```
function GetRec: const StrRec&;
function GetRec: StrRec&;
```

C++

```
const StrRec& GetRec() const;
StrRec& GetRec();
```

Description

System::UnicodeString::GetRec gets a struct that describes the contents of a System::UnicodeString (see page 3381) instance. The struct has this format:

```
struct StrRec {
  unsigned short codePage;
  unsigned short elemSize;
  int refCnt;
  int length;
};
```

35.1.23.6.21 UnicodeString::Insert Method

Inserts specified string into System::UnicodeString (see page 3381).

Pascal

```
function Insert(const str: UnicodeString&; index: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& Insert(const UnicodeString& str, int index);
```

Description

System::UnicodeString::Insert inserts the string `str` into this System::UnicodeString (see page 3381) object beginning at the position `index`, where 1 is the first position in the string. It returns the resulting modified string (*this).

If `index` is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to System::UnicodeString::Length (see page 3404), turning the operation into an append.

If the `str` parameter is an empty string, System::UnicodeString::Insert does nothing.

System::UnicodeString::Insert throws an EOutOfMemory exception if the method is unable to allocate enough memory to accommodate the new, longer string.

Note: The `index` parameter always specifies the character, not the byte, index.

See Also

System::UnicodeString::Length (see page 3404)

35.1.23.6.22 UnicodeString::IntToHex Method

Converts number into string containing the number's hexadecimal (base 16) representation.

Pascal

```
function IntToHex(value: Integer; digits: Integer): String; static;
```

C++

```
static __fastcall UnicodeString IntToHex(int value, int digits);
```

Description

`Value` is the number to convert. `Digits` indicates the minimum number of hexadecimal digits.

35.1.23.6.23 UnicodeString::IsDelimiter Method

Returns true if the specified character matches any character in `delimiters` string.

Pascal

```
function IsDelimiter(const delimiters: UnicodeString&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const UnicodeString& delimiters, int index) const;
```

Description

System::UnicodeString::IsDelimiter returns true if the character indicated by `index` (counting from 1) is a character contained in `delimiters`. If the character is not in `delimiters`, System::UnicodeString::IsDelimiter returns false.

See Also

System::UnicodeString::LastDelimiter (see page 3404)

35.1.23.6.24 UnicodeString::IsEmpty Method

Indicates if System::UnicodeString (see page 3381) is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::UnicodeString::IsEmpty returns true if a System::UnicodeString (see page 3381) object is empty, false otherwise.

35.1.23.6.25 UnicodeString::IsLeadSurrogate Method

Tests if element is lead surrogate.

Pascal

```
function IsLeadSurrogate(index: Integer): bool;
```

C++

```
__fastcall bool IsLeadSurrogate(int index) const;
```

Description

System::UnicodeString::IsLeadSurrogate returns true if the indexed element is a lead surrogate and false otherwise. Note that **index** is an element index into the string, not a character or byte index.

See Also

System::UnicodeString::ByteType (see page 3391)

System::UnicodeString::IsTrailSurrogate (see page 3403)

35.1.23.6.26 UnicodeString::IsPathDelimiter Method

Indicates if specified character is a path delimiter.

Pascal

```
function IsPathDelimiter(index: Integer): bool;
```

C++

```
__fastcall bool IsPathDelimiter(int index) const;
```

Description

System::UnicodeString::IsPathDelimiter returns true if the character indicated by `index` (counting from 1) is the path delimiter ('\' on Windows, '/' on Linux).

35.1.23.6.27 UnicodeString::IsTrailSurrogate Method

Tests if element is trail surrogate.

Pascal

```
function IsTrailSurrogate(index: Integer): bool;
```

C++

```
__fastcall bool IsTrailSurrogate(int index) const;
```

Description

System::UnicodeString::IsTrailSurrogate returns true if the indexed element is a trail surrogate and false otherwise. Note that **index** is an element index into the string, not a character or byte index.

See Also

System::UnicodeString::ByteType (see page 3391)

System::UnicodeString::IsLeadSurrogate (see page 3403)

35.1.23.6.28 UnicodeString::LastChar Method

Returns pointer to last character in System::UnicodeString (see page 3381).

Pascal

```
function LastChar: wchar_t*;
```

C++

```
__fastcall wchar_t* LastChar() const;
```

Description

Returns a pointer to the last character in a System::UnicodeString (see page 3381) instance. This function supports multibyte characters, returning a multibyte character when appropriate.

35.1.23.6.29 UnicodeString::LastDelimiter Method

Returns index of rightmost character that matches any character in `delimiters` string.

Pascal

```
function LastDelimiter(const delimiters: UnicodeString&): Integer;
```

C++

```
__fastcall int LastDelimiter(const UnicodeString& delimiters) const;
```

Description

System::UnicodeString::LastDelimiter returns the 1-based character index in the System::UnicodeString (see page 3381) object of the rightmost character that matches any character in `delimiters` except NULL The System::UnicodeString (see page 3381) object may contain multibyte characters; `delimiters` must contain non-NULL characters.

See Also

System::UnicodeString::IsDelimiter (see page 3402)

35.1.23.6.30 UnicodeString::Length Method

Returns length in characters of System::UnicodeString (see page 3381).

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::UnicodeString::Length returns the number of characters (not necessarily bytes) in a System::UnicodeString (see page 3381) object.

35.1.23.6.31 LoadStr Method

This is the overview for the LoadStr method overload.

Overload List

	Name	Description
	UnicodeString::LoadStr (HINSTANCE, int) (see page 3405)	Loads string from compiled resources of specified module.
	UnicodeString::LoadStr (int) (see page 3405)	Loads string from compiled resources of specified module.

35.1.23.6.31.1 UnicodeString::LoadStr Method (HINSTANCE, int)

Loads string from compiled resources of specified module.

Pascal

```
function LoadStr(ident: Integer): String; static;
function LoadStr(hInstance: HINSTANCE; ident: Integer): String; static;
```

C++

```
static __fastcall UnicodeString LoadStr(int ident);
static __fastcall UnicodeString LoadStr(HINSTANCE hInstance, int ident);
```

Description

System::UnicodeString::LoadStr loads a specified string resource.

`ident` is the resource identifier of the string.

`hInstance` is the instance handle of the module that contains the string resource. If `hInstance` is not given, System::UnicodeString::LoadStr loads the string from the current module (executable or shared library)

If the string resource does not exist, System::UnicodeString::LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::UnicodeString::LoadString (see page 3405)

System::UnicodeString::FmtLoadStr (see page 3398)

35.1.23.6.32 UnicodeString::LoadString Method

Loads string from compiled resources of specified module.

Pascal

```
function LoadString(hInstance: HINSTANCE; ident: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& LoadString(HINSTANCE hInstance, int ident);
```

Description

System::UnicodeString::LoadString loads a specified string resource.

`ident` is the resource identifier of the string.

`hInstance` is the instance handle of the module that contains the string resource.

If the string resource does not exist, System::UnicodeString::LoadString returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

System::UnicodeString::LoadStr (see page 3405)

System::UnicodeString::FmtLoadStr (see page 3398)

35.1.23.6.33 UnicodeString::LowerCase Method

Returns lower-case version of System::UnicodeString (see page 3381).

Pascal

```
function LowerCase: String;
```

C++

```
__fastcall UnicodeString LowerCase() const;
```

Description

System::UnicodeString::LowerCase returns a new System::UnicodeString (see page 3381) instance that contains all characters in this string, converted to lower case. The conversion uses the current locale when appropriate. The original System::UnicodeString (see page 3381) instance remains unchanged.

See Also

System::UnicodeString::UpperCase (see page 3412)

35.1.23.6.34 UnicodeString::Pos Method

Returns character index at which specified substring begins.

Pascal

```
function Pos(const subStr: UnicodeString&): Integer;
```

C++

```
__fastcall int Pos(const UnicodeString& subStr) const;
```

Description

System::UnicodeString::Pos returns the character index in the System::UnicodeString (see page 3381) instance at which the substring `subStr` begins, where 1 is the index of the first character, 2 is the index of the second character, and so on. If the substring is not contained in the System::UnicodeString (see page 3381), System::UnicodeString::Pos returns 0.

See Also

System::UnicodeString::SubString (see page 3408)

35.1.23.6.35 UnicodeString::printf Method

Sets value of System::UnicodeString (see page 3381) instance, given format string and its arguments.

Pascal

```
function printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int printf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::printf to set the value of a System::UnicodeString (see page 3381) instance, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the length in characters of the final formatted string.

See Also

System::UnicodeString::cat_printf (see page 3392)

System::UnicodeString::Format (see page 3399)

System::UnicodeString::sprintf (see page 3408)

System::UnicodeString::vprintf (see page 3413)

35.1.23.6.36 UnicodeString::RefCount Method

Gets reference count of System::UnicodeString (see page 3381).

Pascal

```
function RefCount: Integer;
```

C++

```
int RefCount() const;
```

Description

System::UnicodeString::RefCount gets the reference count of a System::UnicodeString (see page 3381) instance. If the instance is empty, it returns 0.

See Also

System::UnicodeString::GetRec (see page 3401)

35.1.23.6.37 UnicodeString::SetLength Method

Changes character length of System::UnicodeString (see page 3381).

Pascal

```
function SetLength(newLength: Integer): UnicodeString&;
```

C++

```
__fastcall UnicodeString& SetLength(int newLength);
```

Description

System::UnicodeString::SetLength changes the string's character length to `newLength`, truncating if necessary.

System::UnicodeString::SetLength reallocates the underlying array of characters that stores the value of the System::UnicodeString (see page 3381) instance. Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate the System::UnicodeString (see page 3381) instance, an EOutOfMemory exception is thrown. Following a call to System::UnicodeString::SetLength, the System::UnicodeString (see page 3381) instance is guaranteed not to be a shared reference.

System::UnicodeString::SetLength returns the modified System::UnicodeString (see page 3381) object (*this)

35.1.23.6.38 UnicodeString::sprintf Method

Sets value of System::UnicodeString (see page 3381), given format string and its arguments.

Pascal

```
function sprintf(const format: wchar_t*; : ...): UnicodeString&; cdecl;
```

C++

```
__cdecl UnicodeString& sprintf(const wchar_t* format, ...);
```

Description

Use System::UnicodeString::sprintf to set to the value of the System::UnicodeString (see page 3381) instance, given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

This method returns a reference to the System::UnicodeString (see page 3381) instance (*this) with the resulting value.

See Also

System::UnicodeString::Format (see page 3399)

System::UnicodeString::cat_sprintf (see page 3393)

System::UnicodeString::printf (see page 3406)

System::UnicodeString::vprintf (see page 3413)

35.1.23.6.39 UnicodeString::StringOfChar Method

Returns string with specified number of characters.

Pascal

```
function StringOfChar(ch: wchar_t; count: Integer): String; static;
```

C++

```
static __fastcall UnicodeString StringOfChar(wchar_t ch, int count);
```

Description

System::UnicodeString::StringOfChar returns a string containing `count` characters with the character value given by `ch`. For example,

```
UnicodeString (see page 3381) s = System::UnicodeString::UnicodeString (see page 3381)().StringOfChar('A', 4);
```

sets `s` to the string "AAAA".

35.1.23.6.40 UnicodeString::SubString Method

Returns specified substring of System::UnicodeString (see page 3381).

Pascal

```
function SubString(index: Integer; count: Integer): String;
```

C++

```
__fastcall UnicodeString SubString(int index, int count) const;
```

Description

System::UnicodeString::SubString returns a new System::UnicodeString (see page 3381) instance that is a substring of this System::UnicodeString (see page 3381) instance. The substring contains `count` characters beginning at `index`, which is a character index.

See Also

System::UnicodeString::Pos (see page 3406)

35.1.23.6.41 UnicodeString::swap Method

Swaps System::UnicodeString (see page 3381) with another string.

Pascal

```
function swap(var other: String): UnicodeString&;
```

C++

```
UnicodeString& swap(UnicodeString& other);
```

Description

System::UnicodeString::swap swaps a System::UnicodeString (see page 3381) with a given string. Swapping is not done by copying, but by swapping references. This is a very efficient operation, well suited to sorting strings, for example.

35.1.23.6.42 UnicodeString::t_str Method

Returns System::UnicodeString::UnicodeString (see page 3381) data (see page 3396) as const char*.

Pascal

```
function t_str: char*;
```

C++

```
__fastcall char* t_str();
```

Description

The default VCL string type is now UnicodeString (see page 3381) instead of AnsiString. For backward compatibility, the method System::UnicodeString::t_str returns **const char*** (instead of **const wchar_t***), by narrowing the wide data (see page 3396) of the UnicodeString (see page 3381) instance. This behavior can be unexpected, because code might not expect a call to t_str() to corrupt the underlying data (see page 3396). You can observe this behavior in code that displays the underlying data (see page 3396) in the UI, such as a TListItem.

For example, in the following case the TCHAR mapping option is set by default to char, and the data (see page 3396) displays by the ListView are corrupted after the call to t_str() on the last line of the method:

```
void ProcessSelectedItem(const char* item);
void __fastcall TForm6::ListView1DblClick(TObject *Sender)
{
  int index = ListView1->Selected->Index;
  TListItem *ClassItem = ListView1->Items->Item[index];
  ProcessSelectedItem(ClassItem->Caption.t_str());
}
```

Whenever the TCHAR mapping option is set to **char** (the default setting), you should not use System::UnicodeString::t_str on fields or properties that do not expect the underlying data (see page 3396) to be narrowed. Instead, you can change the behavior of System::UnicodeString::t_str by setting the TCHAR_mapping option to **wchar_t**. You can make this project setting on the **Project ▸ Options ▸ Directories and Conditionals** dialog box, and it will ensure that t_str() returns the UnicodeString (see page 3381)'s wide data (see page 3396) without modifying it.

See Also

System::UnicodeString::c_str (see page 3392)

35.1.23.6.43 UnicodeString::ThrowIfOutOfRange Method

Throws exception if index is out of range in System::UnicodeString (see page 3381).

Pascal

```
procedure ThrowIfOutOfRange(idx: Integer); cdecl;
```

C++

```
__cdecl void ThrowIfOutOfRange(int idx) const;
```

Description

System::UnicodeString::ThrowIfOutOfRange throws an `ERangeError` exception if the given 1-based character index is out of range for a System::UnicodeString (see page 3381) instance. Specifically, it throws an exception if the index is less than 1 or greater than the character length of the string.

35.1.23.6.44 UnicodeString::ToDouble Method

Converts string to floating-point value.

Pascal

```
function ToDouble: double;
```

C++

```
__fastcall double ToDouble() const;
```

Description

System::UnicodeString::ToDouble converts the string to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks in the string are ignored. The `DecimalSeparator` global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, an EConvertError exception is thrown.

35.1.23.6.45 UnicodeString::ToInt Method

Converts string to integer, throwing exception if string is invalid.

Pascal

```
function ToInt: Integer;
```

C++

```
__fastcall int ToInt() const;
```

Description

System::UnicodeString::ToInt converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, an EConvertError exception is thrown.

See Also

System::UnicodeString::ToIntDef (see page 3411)

35.1.23.6.46 UnicodeString::ToIntDef Method

Converts string to integer, returning default value if string is invalid.

Pascal

```
function ToIntDef(defaultValue: Integer): Integer;
```

C++

```
__fastcall int ToIntDef(int defaultValue) const;
```

Description

System::UnicodeString::ToIntDef converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, the value given by `defaultValue` is returned.

See Also

System::UnicodeString::ToInt (see page 3410)

35.1.23.6.47 UnicodeString::Trim Method

Returns new System::UnicodeString (see page 3381) without leading spaces, trailing spaces, or control characters.

Pascal

```
function Trim: String;
```

C++

```
__fastcall UnicodeString Trim() const;
```

Description

System::UnicodeString::Trim returns a new System::UnicodeString (see page 3381) instance, removing leading and trailing spaces and control characters. Use System::UnicodeString::Trim to remove blank space before and after the first printing character. The original System::UnicodeString (see page 3381) instance remains unchanged.

See Also

System::UnicodeString::TrimLeft (see page 3411)

System::UnicodeString::TrimRight (see page 3412)

35.1.23.6.48 UnicodeString::TrimLeft Method

Returns new System::UnicodeString (see page 3381) without leading spaces or control characters.

Pascal

```
function TrimLeft: String;
```

C++

```
__fastcall UnicodeString TrimLeft() const;
```

Description

System::UnicodeString::TrimLeft returns a new System::UnicodeString (see page 3381) instance, removing leading spaces and control characters. Use System::UnicodeString::TrimLeft to remove blank space before the first printing character. The original System::UnicodeString (see page 3381) instance remains unchanged.

See Also

System::UnicodeString::Trim (see page 3411)

System::UnicodeString::TrimRight (see page 3412)

35.1.23.6.49 UnicodeString::TrimRight Method

Returns new System::UnicodeString (see page 3381) without trailing spaces or control characters.

Pascal

```
function TrimRight: String;
```

C++

```
__fastcall UnicodeString TrimRight() const;
```

Description

System::UnicodeString::TrimRight returns a new System::UnicodeString (see page 3381) instance, removing trailing spaces and control characters. Use the System::UnicodeString::TrimRight function to remove blank space after the last printing character. The original System::UnicodeString (see page 3381) instance remains unchanged.

See Also

System::UnicodeString::Trim (see page 3411)

System::UnicodeString::TrimLeft (see page 3411)

35.1.23.6.50 UnicodeString::Unique Method

Makes System::UnicodeString (see page 3381) instance unique.

Pascal

```
function Unique: UnicodeString&;
```

C++

```
__fastcall UnicodeString& Unique();
```

Description

Use this function to make a string unique (refcnt == 1). System::UnicodeString::Unique sets the reference count on the string for the System::UnicodeString (see page 3381) object to 1 so that it is the only object referencing that string.

See Also

System::UnicodeString::RefCount (see page 3407)

35.1.23.6.51 UnicodeString::UpperCase Method

Returns upper-case version of System::UnicodeString (see page 3381).

Pascal

```
function UpperCase: String;
```

C++

```
__fastcall UnicodeString UpperCase() const;
```

Description

System::UnicodeString::UpperCase returns a new System::UnicodeString (see page 3381) instance that contains all characters in this string, converted to upper case. The conversion uses the current locale where appropriate. The original System::UnicodeString (see page 3381) instance remains unchanged.

See Also

System::UnicodeString::LowerCase (see page 3406)

35.1.23.6.52 UnicodeString::vprintf Method

Sets value of System::UnicodeString (see page 3381) given format string and its arguments.

Pascal

```
function vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(const wchar_t* format, va_list);
```

Description

Use System::UnicodeString::vprintf to set to the value of the System::UnicodeString (see page 3381) instance, given a standard C++ format specifier. Pass values to any arguments in the format specifier as a variable argument list that was created using `va_start`.

This method returns the length in characters of the final formatted string.

See Also

System::UnicodeString::Format (see page 3399)

System::UnicodeString::cat_vprintf (see page 3394)

System::UnicodeString::sprintf (see page 3408)

System::UnicodeString::printf (see page 3406)

35.1.23.6.53 UnicodeString::w_str Method

Returns System::UnicodeString::UnicodeString (see page 3381) data (see page 3396).

Pascal

```
function w_str: wchar_t*;
```

C++

```
__fastcall wchar_t* w_str() const;
```

Description

System::UnicodeString::w_str behaves the same as System::UnicodeString::c_str (see page 3392).

See Also

System::UnicodeString::c_str (see page 3392)

35.1.23.7 UnicodeString Operators

The operators of the UnicodeString class are listed here.

Public Operators

	Name	Description
	!= (see page 3414)	Compares two strings.
	[] (see page 3414)	This is the overview for the [] operator overload.

	+ (see page 3415)	Concatenates two System::UnicodeString (see page 3381) values.
	+= (see page 3416)	Append (see page 3520) System::UnicodeString (see page 3381).
	< (see page 3416)	Compares two strings.
	<= (see page 3417)	Compares two strings.
	= (see page 3418)	Assigns string `rhs` to System::UnicodeString (see page 3381).
	== (see page 3418)	Compares System::UnicodeString (see page 3381) with specified string.
	> (see page 3419)	Compares two strings.
	>= (see page 3419)	Compares two strings.

35.1.23.7.1 UnicodeString::!= Operator

Compares two strings.

Pascal

```
operator !=(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator !=(const UnicodeString& rhs) const;
```

Description

Returns true if the strings are not the same, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3381)::!= performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3395) method.

See Also

System::UnicodeString::CompareIC (see page 3395)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

35.1.23.7.2 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	UnicodeString::[] (int) (see page 3414)	Returns character at specified index in string.
	UnicodeString::[] (int) (see page 3414)	Returns character at specified index in string.

35.1.23.7.2.1 UnicodeString::[] Operator (int)

Returns character at specified index in string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall wchar_t operator [](const int idx) const;
__fastcall wchar_t& operator [](const int idx);
```

Description

System::UnicodeString (see page 3381)::[] returns the character in the string at the character index value `idx`. The System::UnicodeString (see page 3381)::[] assumes a base index of 1.

See Also

System::UnicodeString::Substring

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

System::UnicodeString::+= (see page 3416)

35.1.23.7.3 UnicodeString::+ Operator

Concatenates two System::UnicodeString (see page 3381) values.

Pascal

```
operator +(const rhs: UnicodeString&);
```

C++

```
__fastcall UnicodeString operator +(const UnicodeString& rhs) const;
```

Description

The friend function returns a System::UnicodeString (see page 3381) instance that is the concatenation of strings `lhs` and `rhs`.

The member function returns a System::UnicodeString (see page 3381) instance that is the concatenation of this System::UnicodeString (see page 3381) instance and `rhs`.

See Also

System::AnsiStringBase

System::WideString

System::UnicodeString::[] (see page 3414)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+= (see page 3416)

35.1.23.7.4 UnicodeString::+= Operator

Append (see page 3520) System::UnicodeString (see page 3381).

Pascal

```
operator +=(const rhs: UnicodeString&);
```

C++

```
__fastcall UnicodeString& operator +=(const UnicodeString& rhs);
```

Description

System::UnicodeString (see page 3381)::+= concatenates this System::UnicodeString (see page 3381) instance and `rhs`, assigning the result to this System::UnicodeString (see page 3381) instance.

System::UnicodeString (see page 3381)::+= is not supported on a property with getter/setter functions.

See Also

System::AnsiStringBase

System::WideString

System::UnicodeString::[] (see page 3414)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

35.1.23.7.5 UnicodeString::< Operator

Compares two strings.

Pascal

```
operator <(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator <(const UnicodeString& rhs) const;
```

Description

Returns true if a System::UnicodeString (see page 3381) instance is less than `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3381)::< performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3395) method.

See Also

System::UnicodeString::CompareIC (see page 3395)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

35.1.23.7.6 UnicodeString::<= Operator

Compares two strings.

Pascal

```
operator <=(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator <=(const UnicodeString& rhs) const;
```

Description

Returns true if the System::UnicodeString (see page 3381) instance is less than or equal to `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3381)::<= performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3395) method.

See Also

System::UnicodeString::CompareIC (see page 3395)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::+ (see page 3415)

35.1.23.7.7 UnicodeString::= Operator

Assigns string `rhs` to System::UnicodeString (see page 3381).

Pascal

```
operator =(const rhs: UnicodeString&);
```

C++

```
__fastcall UnicodeString& operator =(const UnicodeString& rhs);
```

Description

Use System::UnicodeString (see page 3381)::= to assign the value of the System::UnicodeString (see page 3381) instance from another instance of System::UnicodeString (see page 3381).

See Also

System::AnsiStringBase

System::WideString

System::UnicodeString::[] (see page 3414)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

System::UnicodeString::+= (see page 3416)

35.1.23.7.8 UnicodeString::== Operator

Compares System::UnicodeString (see page 3381) with specified string.

Pascal

```
operator ==(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator ==(const UnicodeString& rhs) const;
```

Description

Returns true if this System::UnicodeString (see page 3381) instance is the same as `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3381)::== performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3395) method.

See Also

System::UnicodeString::CompareIC (see page 3395)

System::UnicodeString::= (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

35.1.23.7.9 UnicodeString::> Operator

Compares two strings.

Pascal

```
operator >(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator >(const UnicodeString& rhs) const;
```

Description

Returns true if this System::UnicodeString (see page 3381) instance is greater than `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3381)::> performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3395) method.

See Also

System::UnicodeString::CompareIC (see page 3395)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::>= (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

35.1.23.7.10 UnicodeString::>= Operator

Compares two strings.

Pascal

```
operator >=(const rhs: UnicodeString&);
```

C++

```
__fastcall bool operator >=(const UnicodeString& rhs) const;
```

Description

Returns true if this System::UnicodeString (see page 3381) instance is greater than or equal to `rhs`, false otherwise. The compare operation is controlled by the current Windows locale, when appropriate.

Note: System::UnicodeString (see page 3381)::>= performs a case sensitive comparison. To ignore case, use the CompareIC (see page 3395) method.

See Also

System::UnicodeString::CompareIC (see page 3395)

System::UnicodeString::= (see page 3418)

System::UnicodeString::== (see page 3418)

System::UnicodeString::!= (see page 3414)

System::UnicodeString::> (see page 3419)

System::UnicodeString::< (see page 3416)

System::UnicodeString::<= (see page 3417)

System::UnicodeString::+ (see page 3415)

35.1.23.8 UnicodeString Friends

The friends of the UnicodeString class are listed here.

Public Friends

Name	Description
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) (see page 3420)	This is friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs).
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) (see page 3420)	This is friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs).

35.1.23.8.1 friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) Friend

Pascal

```
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs)
```

C++

```
friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs);
```

Description

This is friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs).

35.1.23.8.2 friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) Friend

Pascal

```
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs)
```

C++

```
friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs);
```

Description

This is friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs).

35.1.24 Variant Class

System::Variant is a C++ implementation of the Delphi intrinsic type System::Variant.

Class Hierarchy

Pascal

```
Variant = class(TVarData);
```

C++

```
class Variant : public TVarData;
```

File

sysvari.h

Description

The System::Variant type is capable of representing values that change type dynamically. Whereas a variable of any other type is statically bound to that type, a variable of the System::Variant type can assume values of differing types at run-time. The System::Variant type is most commonly used in situations where the actual type to be operated upon varies or is unknown at compile-time. While variants offer great flexibility, they also consume more memory than regular variables, and operations on variants are substantially slower than operations on statically typed values.

System::Variants (see page 4973) can contain various numeric, string, Boolean, and date-time values. If COM is available (Windows only) variants can also contain OLE Automation objects. Finally, variants can contain arrays of any of the above types. The special System::Variant value Unassigned is used to indicate that a variant has not yet been assigned a value, and the special variant value Null is used to indicate unknown or missing data.

A variant can be constructed from the following data types:

short (see page 3462)

int (see page 3461)

float (see page 3460)

double (see page 3460)

Currency (see page 3459)

TDateTime (see page 3463)

bool (see page 3458)

WordBool

Byte

AnsiString (see page 3458)&

char (see page 3459) *

wchar_t * const

Ole2::IDispatch (see page 3494)* const (Windows only)

Ole2::IUnknown (see page 3756)* const (Windows only)

Note: When a System::Variant contains an OLE Automation object, the variant can be used to get and set properties of the

object, and to invoke methods on the object.

System::Variant variables are always initialized to be Unassigned when they are first created. This is true whether a variant variable is global, local, or part of a structure such as an array.

In C++ code, variant arrays are usually created using a System::Variant constructor in conjunction with the OPENARRAY macro:

```
Variant V(OPENARRAY(int,(0,HighVal,0,HighVal)),varInteger);
```

You can also use the VarArrayCreate function, but that function is designed for use in Delphi code, and makes for complex C++ code.

Note: Use only the operators declared within System::Variant. The compiler ignores any operators you overload yourself.

Note: For binary operators, if one operand is of type System::Variant, the other operand is automatically converted to type System::Variant.

See Also

System::AutoCmd

System::TVarRec (see page 3747)

TVarData (see page 3740)

35.1.24.1 Variant Members

The following tables list the members exposed by Variant.

Public Methods

	Name	Description
	~Variant (see page 3424)	Frees a System::Variant::Variant (see page 3421) object.
	ArrayDimCount (see page 3429)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 3429)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 3429)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 3430)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 3430)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 3430)	Unlocks a variant array that was previously locked by ArrayLock (see page 3429).
	AsType (see page 3431)	Casts a System::Variant::Variant (see page 3421) object to the type specified by the argument VarType.
	ChangeType (see page 3431)	Changes the data type of a System::Variant::Variant (see page 3421) object once it has been instantiated.
	Clear (see page 3431)	System::Variant::Clears the value of the System::Variant::Variant (see page 3421).
	Empty (see page 3431)	This is Empty, a member of class Variant.
	GetBaseVariant (see page 3432)	Returns the System::Variant (see page 3421) that is assigned as the value of this System::Variant (see page 3421) when Type (see page 3436) is varSystem::Variant (see page 3421).
	GetElement (see page 3432)	This is the overview for the GetElement method overload.
	GetError (see page 3433)	Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and returns the integer value.
	IsArray (see page 3433)	Queries the VType of the base class TVarData (see page 3740) for the varArray (see page 3861) bit.
	IsEmpty (see page 3433)	Queries the VType of the base class TVarData (see page 3740).
	IsNull (see page 3434)	Queries the VType of the base class TVarData (see page 3740) for the varNull (see page 3868) bit.
	NoParam (see page 3434)	This is NoParam, a member of class Variant.

	PutElement (see page 3434)	This is the overview for the PutElement method overload.
	Sanitized (see page 3435)	This is Sanitized, a member of class Variant.
	SetError (see page 3435)	Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and assigns the integer err as the value of the varError (see page 3866).
	Type (see page 3436)	Returns type code of a System::Variant (see page 3421) instance.
	Variant (see page 3424)	This is the overview for the Variant constructor overload.

Public Operators

	Name	Description
	- (see page 3439)	This is the overview for the - operator overload.
	! (see page 3439)	Returns a new System::Variant::Variant (see page 3421) that is the logical negation of this System::Variant::Variant (see page 3421).
	!= (see page 3440)	This is the overview for the != operator overload.
	% (see page 3440)	This is the overview for the % operator overload.
	%= (see page 3441)	Returns this System::Variant::Variant (see page 3421) as the remainder from dividing its original value and the System::Variant::Variant (see page 3421) specified by rhs.
	& (see page 3442)	This is the overview for the & operator overload.
	&= (see page 3442)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise and operation.
	* (see page 3443)	This is the overview for the * operator overload.
	*= (see page 3443)	Multiplies this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) specified by rhs.
	/ (see page 3444)	This is the overview for the / operator overload.
	/= (see page 3444)	Divides this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) rhs.
	^ (see page 3445)	This is the overview for the ^ operator overload.
	^= (see page 3445)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise xor operation.
	__int64 (see page 3446)	This is __int64, a member of class Variant.
	| (see page 3446)	This is the overview for the | operator overload.
	|= (see page 3446)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise or operation.
	+ (see page 3447)	This is the overview for the + operator overload.
	+= (see page 3447)	Adds the System::Variant::Variant (see page 3421) specified by rhs to this System::Variant::Variant (see page 3421) object.
	< (see page 3448)	This is the overview for the < operator overload.
	<< (see page 3448)	This is the overview for the << operator overload.
	<<= (see page 3449)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3421).
	<= (see page 3449)	This is the overview for the <= operator overload.
	-= (see page 3454)	Subtracts the System::Variant::Variant (see page 3421) specified by rhs from this System::Variant::Variant (see page 3421) object.
	= (see page 3450)	This is the overview for the = operator overload.
	== (see page 3455)	This is the overview for the == operator overload.
	> (see page 3456)	This is the overview for the > operator overload.
	>= (see page 3456)	This is the overview for the >= operator overload.
	>> (see page 3457)	This is the overview for the >> operator overload.
	>>= (see page 3458)	Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 3421).
	AnsiString (see page 3458)	Converts the System::Variant::Variant (see page 3421) to an AnsiString object.
	bool (see page 3458)	Converts the System::Variant::Variant (see page 3421) to a boolean value.
	char (see page 3459)	This is char, a member of class Variant.
	Currency (see page 3459)	Converts the System::Variant::Variant (see page 3421) to a Currency object.
	Currency* (see page 3459)	This is Currency*, a member of class Variant.
	double (see page 3460)	Converts the System::Variant::Variant (see page 3421) to a double value.
	double* (see page 3460)	This is double*, a member of class Variant.
	float (see page 3460)	Converts the System::Variant::Variant (see page 3421) to a float value.

	float* (see page 3460)	This is float*, a member of class Variant.
	IDispatch* (see page 3461)	Converts the System::Variant (see page 3421) to an System::Variant::IDispatch (see page 3494)* interface pointer.
	IInterface* (see page 3461)	This is IInterface*, a member of class Variant.
	int (see page 3461)	Converts the System::Variant::Variant (see page 3421) to an integer value.
	int* (see page 3461)	This is int*, a member of class Variant.
	IUnknown* (see page 3462)	Converts the System::Variant (see page 3421) to an System::Variant::IUnknown (see page 3756)* interface pointer.
	long (see page 3462)	This is long, a member of class Variant.
	long* (see page 3462)	This is long*, a member of class Variant.
	short (see page 3462)	Converts the System::Variant::Variant (see page 3421) to a short value.
	short* (see page 3463)	This is short*, a member of class Variant.
	signed char (see page 3463)	This is signed char, a member of class Variant.
	signed char* (see page 3463)	This is signed char*, a member of class Variant.
	TDateTime (see page 3463)	Converts the System::Variant::Variant (see page 3421) to a TDateTime object.
	TDateTime* (see page 3464)	This is TDateTime*, a member of class Variant.
	UnicodeString (see page 3464)	This is UnicodeString, a member of class Variant.
	unsigned char (see page 3464)	Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value.
	unsigned char* (see page 3465)	Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value. And returns a reference.
	unsigned int (see page 3465)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 3465)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 3465)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 3466)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 3466)	Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value.
	unsigned short* (see page 3466)	Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value and returns a reference to the value.
	WideString (see page 3466)	This is WideString, a member of class Variant.

35.1.24.2 Variant::~Variant Destructor

Frees a System::Variant::Variant (see page 3421) object.

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~Variant();
```

Description

System::Variant (see page 3421)::~Variant (see page 3421) is the destructor for System::Variant::Variant (see page 3421).

35.1.24.3 Variant Constructor

This is the overview for the Variant constructor overload.

Overload List

	Name	Description
	Variant::Variant () (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).

	Variant::Variant (__int64) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (__int64*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (AnsiString&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (bool) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (char) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (char*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (Currency&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (Currency*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (double) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (double*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (float) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (float*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (int) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (int*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (int*, int, Word) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (long) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (long double) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (long*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (short) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (short*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (signed char) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (signed char*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (System::DelphiInterface<IDispatch>&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).

	Variant::Variant (TDateTime&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (TDateTime*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (TVarData&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (UnicodeString&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned char) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned char*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned int) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned int*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned long) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned long*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned short) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (unsigned short*) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (Variant&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (Variant*, int) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (wchar_t*, bool) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).
	Variant::Variant (WideString&) (see page 3426)	Creates an instance of System::Variant::Variant (see page 3421).

35.1.24.3.1 Variant::Variant Constructor ()

Creates an instance of System::Variant::Variant (see page 3421).

Pascal

```
constructor Create;
constructor Create(const src: Variant&);
constructor Create(const src: TVarData&);
constructor Create(const src: bool);
constructor Create(const src: char);
constructor Create(const src: signed char);
constructor Create(const src: unsigned char);
constructor Create(const src: short);
constructor Create(const src: unsigned short);
constructor Create(const src: Integer);
constructor Create(const src: unsigned int);
constructor Create(const src: long);
```

```
constructor Create(const src: unsigned long);
constructor Create(const src: float);
constructor Create(const src: double);
constructor Create(const src: long double);
constructor Create(const src: __int64);
constructor Create(const src: AnsiString&);
constructor Create(const src: Currency&);
constructor Create(const src: TDateTime&);
constructor Create(const src: WideString&);
constructor Create(const src: UnicodeString&);
constructor Create(const src: char*);
constructor Create(var src: wchar_t; makeCopy: bool = true);
constructor Create(const src: System::DelphiInterface<IDispatch>&);
constructor Create(var src: signed char);
constructor Create(var src: unsigned char);
constructor Create(var src: short);
constructor Create(var src: unsigned short);
constructor Create(var src: Integer);
constructor Create(var src: unsigned int);
constructor Create(var src: long);
constructor Create(var src: unsigned long);
constructor Create(var src: float);
constructor Create(var src: double);
constructor Create(var src: __int64);
constructor Create(var src: Currency);
constructor Create(var src: TDateTime);
constructor Create(const bounds: int*; const boundsSize: Integer; varType: Word);
constructor Create(const values: Variant*; const valuesSize: Integer);
```

C++

```
__fastcall Variant();
__fastcall Variant(const Variant& src);
__fastcall Variant(const TVarData& src);
__fastcall Variant(const bool src);
__fastcall Variant(const char src);
__fastcall Variant(const signed char src);
__fastcall Variant(const unsigned char src);
__fastcall Variant(const short src);
__fastcall Variant(const unsigned short src);
__fastcall Variant(const int src);
__fastcall Variant(const unsigned int src);
__fastcall Variant(const long src);
__fastcall Variant(const unsigned long src);
__fastcall Variant(const float src);
__fastcall Variant(const double src);
__fastcall Variant(const long double src);
__fastcall Variant(const __int64 src);
__fastcall Variant(const AnsiString& src);
__fastcall Variant(const Currency& src);
__fastcall Variant(const TDateTime& src);
__fastcall Variant(const WideString& src);
__fastcall Variant(const UnicodeString& src);
__fastcall Variant(const char* src);
__fastcall Variant(wchar_t* src, bool makeCopy = true);
__fastcall Variant(const System::DelphiInterface<IDispatch>& src);
__fastcall Variant(signed char* src);
__fastcall Variant(unsigned char* src);
__fastcall Variant(short* src);
__fastcall Variant(unsigned short* src);
__fastcall Variant(int* src);
__fastcall Variant(unsigned int* src);
__fastcall Variant(long* src);
__fastcall Variant(unsigned long* src);
__fastcall Variant(float* src);
__fastcall Variant(double* src);
__fastcall Variant(__int64* src);
```

```
__fastcall Variant(Currency* src);
__fastcall Variant(TDateTime* src);
__fastcall Variant(const int* bounds, const int boundsSize, Word varType);
__fastcall Variant(const Variant* values, const int valuesSize);
```

Description

The first System::Variant::Variant (see page 3421) constructor above creates a variant instance with the value Unassigned.

The second System::Variant::Variant (see page 3421) constructor creates a System::Variant::Variant (see page 3421) array with the bounds given by bounds and the element type given by varType. The bounds parameter must contain an even number of values, and each pair of values must specify the upper and lower bounds of one dimension of the array. The boundsSize parameter is the index of the last element in bounds. The element type of the array, given by the varType parameter, must be one of the TVarData (see page 3740) type codes, and cannot include the varArray (see page 3861) or varByRef (see page 3862) bits. The element type cannot be varString (see page 3871). To create an array of strings, use the varOleStr (see page 3869) type code. If the element type is varSystem::Variant::Variant (see page 3421), the elements of the array are themselves System::Variant::Variants (see page 4973) and can, in turn, contain System::Variant::Variant (see page 3421) arrays.

The third System::Variant::Variant (see page 3421) constructor creates a one-dimensional System::Variant::Variant (see page 3421) array with the elements given by the values parameter. The low bound of the returned array is zero, the high bound is given by valuesSize (the index of the last value). The element type of the instantiated array is System::Variant::Variant (see page 3421).

The remaining System::Variant::Variant (see page 3421) constructors create a System::Variant::Variant (see page 3421) with the same value as the src argument, converting src as appropriate. Constructors that declare src as a const parameter copy src into the variant's internal representation. Constructors that do not declare src as a const create a reference to src and set the variant's varByRef (see page 3862) bit.

See Also

Unassigned (see page 5058)

TVarData (see page 3740)

35.1.24.4 Variant Methods

The methods of the Variant class are listed here.

Public Methods

	Name	Description
	ArrayDimCount (see page 3429)	Returns the number of dimensions of the given variant.
	ArrayHighBound (see page 3429)	Returns the high bound of the given dimension in the given variant array.
	ArrayLock (see page 3429)	Locks the given variant array and returns a pointer to the data stored in the array.
	ArrayLowBound (see page 3430)	Returns the low bound of the given dimension in the given variant array.
	ArrayRedim (see page 3430)	Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.
	ArrayUnlock (see page 3430)	Unlocks a variant array that was previously locked by ArrayLock (see page 3429).
	AsType (see page 3431)	Casts a System::Variant::Variant (see page 3421) object to the type specified by the argument VarType.
	ChangeType (see page 3431)	Changes the data type of a System::Variant::Variant (see page 3421) object once it has been instantiated.
	Clear (see page 3431)	System::Variant::Clears the value of the System::Variant::Variant (see page 3421).
	Empty (see page 3431)	This is Empty, a member of class Variant.

	GetBaseVariant (see page 3432)	Returns the System::Variant (see page 3421) that is assigned as the value of this System::Variant (see page 3421) when Type (see page 3436) is varSystem::Variant (see page 3421).
	GetElement (see page 3432)	This is the overview for the GetElement method overload.
	GetError (see page 3433)	Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and returns the integer value.
	IsArray (see page 3433)	Queries the VType of the base class TVarData (see page 3740) for the varArray (see page 3861) bit.
	IsEmpty (see page 3433)	Queries the VType of the base class TVarData (see page 3740).
	IsNull (see page 3434)	Queries the VType of the base class TVarData (see page 3740) for the varNull (see page 3868) bit.
	NoParam (see page 3434)	This is NoParam, a member of class Variant.
	PutElement (see page 3434)	This is the overview for the PutElement method overload.
	Sanitized (see page 3435)	This is Sanitized, a member of class Variant.
	SetError (see page 3435)	Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and assigns the integer err as the value of the varError (see page 3866).
	Type (see page 3436)	Returns type code of a System::Variant (see page 3421) instance.

35.1.24.4.1 Variant::ArrayDimCount Method

Returns the number of dimensions of the given variant.

Pascal

```
function ArrayDimCount: Integer;
```

C++

```
__fastcall int ArrayDimCount() const;
```

Description

The resulting value is zero if the given variant is not a System::Variant::Variant (see page 3421) array.

35.1.24.4.2 Variant::ArrayHighBound Method

Returns the high bound of the given dimension in the given variant array.

Pascal

```
function ArrayHighBound(const dim: Integer = 1): Integer;
```

C++

```
__fastcall int ArrayHighBound(const int dim = 1) const;
```

Description

The dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array, or if the dimension specified by dim does not exist.

35.1.24.4.3 Variant::ArrayLock Method

Locks the given variant array and returns a pointer to the data stored in the array.

Pascal

```
function ArrayLock: Pointer;
```

C++

```
__fastcall Pointer ArrayLock();
```

Description

While locked, a variant array can not be resized and any calls to ArrayRedim (see page 3430) will fail. Once locked, a variant array must later be unlocked using ArrayUnlock (see page 3430). An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array.

The pointer returned by System::Variant::ArrayLock points to an array of elements in which the leftmost dimension increases first. In other words, the dimensions of the returned array pointer are reversed from the dimensions of the variant array.

Once it is verified that a variant array has the correct bounds and dimensions, System::Variant::ArrayLock can be used to gain direct access to the array data which results in much improved performance.

35.1.24.4.4 Variant::ArrayLowBound Method

Returns the low bound of the given dimension in the given variant array.

Pascal

```
function ArrayLowBound(const dim: Integer = 1): Integer;
```

C++

```
__fastcall int ArrayLowBound(const int dim = 1) const;
```

Description

The dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array, or if the dimension specified by dim does not exist.

35.1.24.4.5 Variant::ArrayRedim Method

Resizes the given variant array by changing the high bound of the rightmost dimension to the given value.

Pascal

```
procedure ArrayRedim(highBound: Integer);
```

C++

```
__fastcall void ArrayRedim(int highBound);
```

Description

Existing elements of the array are preserved, and new elements are set to zero or empty. An ESystem::Variant::VariantError exception is raised if the variant given by A is not an array, or if there is not enough memory to resize the array.

35.1.24.4.6 Variant::ArrayUnlock Method

Unlocks a variant array that was previously locked by ArrayLock (see page 3429).

Pascal

```
procedure ArrayUnlock;
```

C++

```
__fastcall void ArrayUnlock();
```

Description

System::Variant::ArrayUnlock invalidates the pointer returned by a call to the ArrayLock (see page 3429) method. An ESystem::Variant::VariantError exception is raised if the System::Variant::Variant (see page 3421) is not an array.

35.1.24.4.7 Variant::AsType Method

Casts a System::Variant::Variant (see page 3421) object to the type specified by the argument VarType.

Pascal

```
function AsType(VarType: Integer): Variant;
```

C++

```
__fastcall Variant AsType(int VarType) const;
```

Description

System::Variant::AsType returns a new System::Variant::Variant (see page 3421) with the same value as this System::Variant::Variant (see page 3421) but with the type specified by VarType. After the call to System::Variant::AsType there are two System::Variant::Variant (see page 3421) objects of the same type.

35.1.24.4.8 Variant::ChangeType Method

Changes the data type of a System::Variant::Variant (see page 3421) object once it has been instantiated.

Pascal

```
function ChangeType(VarType: Integer): Variant&;
```

C++

```
__fastcall Variant& ChangeType(int VarType);
```

Description

System::Variant::ChangeType converts the System::Variant::Variant (see page 3421) to the type specified by VarType.

See Also

System::Variant::Type (see page 3436)

35.1.24.4.9 Variant::Clear Method

System::Variant::Clears the value of the System::Variant::Variant (see page 3421).

Pascal

```
procedure Clear;
```

C++

```
__fastcall void Clear();
```

Description

System::Variant::Clear assigns the value Unassigned to this System::Variant::Variant (see page 3421).

See Also

Unassigned (see page 5058)

35.1.24.4.10 Variant::Empty Method

Pascal

```
function Empty: Variant&; static;
```

C++

```
static Variant& Empty();
```

Description

This is Empty, a member of class Variant.

35.1.24.4.11 Variant::GetBaseVariant Method

Returns the System::Variant (see page 3421) that is assigned as the value of this System::Variant (see page 3421) when Type (see page 3436) is varSystem::Variant (see page 3421).

Pascal

```
function GetBaseVariant: Variant*;
```

C++

```
__fastcall Variant* GetBaseVariant();
```

Description

Call System::Variant::GetBaseVariant to remove a level of indirection by accessing a System::Variant (see page 3421) that has been assigned as the value of this System::Variant (see page 3421). Check the Type (see page 3436) method to ensure that the variant is of type var System::Variant (see page 3421) before calling System::Variant::GetBaseVariant.

See Also

System::Variant (see page 3421)

35.1.24.4.12 GetElement Method

This is the overview for the GetElement method overload.

Overload List

	Name	Description
	Variant::GetElement (int) (see page 3432)	Returns the value of a specified element in a System::Variant::Variant (see page 3421) array.
	Variant::GetElement (int, int) (see page 3432)	Returns the value of a specified element in a System::Variant::Variant (see page 3421) array.
	Variant::GetElement (int, int, int) (see page 3432)	Returns the value of a specified element in a System::Variant::Variant (see page 3421) array.
	Variant::GetElement (int, int, int, int) (see page 3432)	Returns the value of a specified element in a System::Variant::Variant (see page 3421) array.
	Variant::GetElement (int, int, int, int, int) (see page 3432)	Returns the value of a specified element in a System::Variant::Variant (see page 3421) array.

35.1.24.4.12.1 Variant::GetElement Method (int)

Returns the value of a specified element in a System::Variant::Variant (see page 3421) array.

Pascal

```
function GetElement(const i1: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer): Variant;
function GetElement(const i1: Integer; const i2: Integer; const i3: Integer; const i4:
```

```
  Integer): Variant;
  function GetElement(const i1: Integer; const i2: Integer; const i3: Integer; const i4:
  Integer; const i5: Integer): Variant;
```

C++

```
  __fastcall Variant GetElement(const int i1) const;
  __fastcall Variant GetElement(const int i1, const int i2) const;
  __fastcall Variant GetElement(const int i1, const int i2, const int i3) const;
  __fastcall Variant GetElement(const int i1, const int i2, const int i3, const int i4) const;
  __fastcall Variant GetElement(const int i1, const int i2, const int i3, const int i4, const
  int i5) const;
```

Description

Use System::Variant::GetElement to access one-dimensional or multi-dimensional System::Variant::Variant (see page 3421) arrays of up to 5 dimensions. The parameters specify the indexes of the desired element.

See Also

System::Variant::PutElement (see page 3435)

35.1.24.4.13 Variant::GetError Method

Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and returns the integer value.

Pascal

```
  function GetError: HResult;
```

C++

```
  __fastcall HResult GetError() const;
```

Description

See the base class TVarData (see page 3740) for more information about varError (see page 3866).

35.1.24.4.14 Variant::IsArray Method

Queries the VType of the base class TVarData (see page 3740) for the varArray (see page 3861) bit.

Pascal

```
  function IsArray: bool;
```

C++

```
  __fastcall bool IsArray() const;
```

Description

System::Variant::IsArray returns true if the given variant is an array, false otherwise.

See Also

TVarData (see page 3740)

35.1.24.4.15 Variant::IsEmpty Method

Queries the VType of the base class TVarData (see page 3740).

Pascal

```
  function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

System::Variant::IsEmpty returns true if the System::Variant::Variant (see page 3421) type code is varEmpty (see page 3865), false otherwise.

See Also

TVarData (see page 3740)

35.1.24.4.16 Variant::IsNull Method

Queries the VType of the base class TVarData (see page 3740) for the varNull (see page 3868) bit.

Pascal

```
function IsNull: bool;
```

C++

```
__fastcall bool IsNull() const;
```

Description

System::Variant::IsNull returns true if the System::Variant::Variant (see page 3421) type code is varNull (see page 3868), false otherwise.

See Also

TVarData (see page 3740)

35.1.24.4.17 Variant::NoParam Method

Pascal

```
function NoParam: Variant&; static;
```

C++

```
static Variant& NoParam();
```

Description

This is NoParam, a member of class Variant.

35.1.24.4.18 PutElement Method

This is the overview for the PutElement method overload.

Overload List

	Name	Description
	Variant::PutElement (Variant&, int) (see page 3435)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3421) array.
	Variant::PutElement (Variant&, int, int) (see page 3435)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3421) array.
	Variant::PutElement (Variant&, int, int, int) (see page 3435)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3421) array.

	Variant::PutElement (Variant&, int, int, int, int) (see page 3435)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3421) array.
	Variant::PutElement (Variant&, int, int, int, int, int) (see page 3435)	Assigns an element as the value of an entry in a System::Variant::Variant (see page 3421) array.

35.1.24.4.18.1 Variant::PutElement Method (Variant&, int)

Assigns an element as the value of an entry in a System::Variant::Variant (see page 3421) array.

Pascal

```
procedure PutElement(const data: Variant&; const i1: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer; const i4: Integer);
procedure PutElement(const data: Variant&; const i1: Integer; const i2: Integer; const i3:
Integer; const i4: Integer; const i5: Integer);
```

C++

```
__fastcall void PutElement(const Variant& data, const int i1);
__fastcall void PutElement(const Variant& data, const int i1, const int i2);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3,
const int i4);
__fastcall void PutElement(const Variant& data, const int i1, const int i2, const int i3,
const int i4, const int i5);
```

Description

Use System::Variant::PutElement to access one-dimensional and multi-dimensional System::Variant::Variant (see page 3421) arrays of up to 5 dimensions. The data parameter is the value to assign. The remaining parameters provide indexes into the dimensions of the array.

See Also

System::Variant::GetElement (see page 3432)

35.1.24.4.19 Variant::Sanitized Method

Pascal

```
function Sanitized: Variant;
```

C++

```
__fastcall Variant Sanitized() const;
```

Description

This is Sanitized, a member of class Variant.

35.1.24.4.20 Variant::SetError Method

Converts the System::Variant::Variant (see page 3421) to a varError (see page 3866) type and assigns the integer err as the value of the varError (see page 3866).

Pascal

```
procedure SetError(const err: HResult);
```

C++

```
__fastcall void SetError(const HResult err);
```

Description

See the base class TVarData (see page 3740) for more information about varError (see page 3866).

See Also

TVarData (see page 3740)

35.1.24.4.21 Variant::Type Method

Returns type code of a System::Variant (see page 3421) instance.

Pascal

```
function Type: Integer;
```

C++

```
__fastcall int Type() const;
```

Description

System::Variant::Type returns the type code of the given variant. The return value is constructed from the constants declared in the system.hpp header file.

The lower twelve bits of a variant type code (the bits defined by the varSystem::Variant::TypeMask bit mask) define the type of the variant. The varArray (see page 3861) bit is set if the variant is an array of the given type. The varByRef (see page 3862) bit is set if the variant is a reference to a value of the given type as opposed to an actual value.

The following table describes each of the variant type codes.

System::Variant (see page 3421) typeReturn valueContents of variant

Variant (see page 3421) type	Return value	Contents of variant
varEmpty (see page 3865)	0x0000	The variant is Unassigned.
varNull (see page 3868)	0x0001	The variant is Null.
varSmallint (see page 3870)	0x0002	16-bit signed integer (type short (see page 3462) int (see page 3461))
varInteger (see page 3867)	0x0003	32-bit signed integer (type int (see page 3461)).
varSingle (see page 3870)	0x0004	Single-precision floating-point value (type float (see page 3460))
varDouble (see page 3865)	0x0005	Double-precision floating-point value (type double (see page 3460)).
varCurrency (see page 3863)	0x0006	Currency (see page 3459) floating-point value (type Currency (see page 3459)).
varDate (see page 3864)	0x0007	Date and time value (type TDateTime (see page 3463)).
varOleStr (see page 3869)	0x0008	Reference to a dynamically allocated Unicode string.

varDispatch (see page 3864)	0x0009	Reference to an OLE automation object (an IDispatch (see page 3494) interface pointer).
varError (see page 3866)	0x000A	Operating system error code.
varBoolean (see page 3862)	0x000B	16-bit boolean (type WordBool).
varVariant (see page 3873)	0x000C	Indicates another Variant (see page 3421).
varUnknown (see page 3873)	0x000D	Reference to an unknown OLE object (an IUnknown (see page 3756) interface pointer).
varByte (see page 3863)	0x0011	Indicates an unsigned char (see page 3459).
varString (see page 3871)	0x0100	Reference to a dynamically allocated string (type AnsiString (see page 3458)).
varTypeMask (see page 3872)	0x0FFF	Indicates the type of each array element.
varArray (see page 3861)	0x2000	Indicates a Variant (see page 3421) array.
varByRef (see page 3862)	0x4000	Indicates that the variant contains a reference as opposed to a value.

The type of a System::Variant (see page 3421) can be changed using the ChangeSystem::Variant::Type member function.

See Also

System::Variant::ChangeType (see page 3431)

35.1.24.5 Variant Operators

The operators of the Variant class are listed here.

Public Operators

	Name	Description
	- (see page 3439)	This is the overview for the - operator overload.
	! (see page 3439)	Returns a new System::Variant::Variant (see page 3421) that is the logical negation of this System::Variant::Variant (see page 3421).
	!= (see page 3440)	This is the overview for the != operator overload.
	% (see page 3440)	This is the overview for the % operator overload.
	%= (see page 3441)	Returns this System::Variant::Variant (see page 3421) as the remainder from dividing its original value and the System::Variant::Variant (see page 3421) specified by rhs.
	& (see page 3442)	This is the overview for the & operator overload.
	&= (see page 3442)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise and operation.
	* (see page 3443)	This is the overview for the * operator overload.
	*= (see page 3443)	Multiplies this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) specified by rhs.
	/ (see page 3444)	This is the overview for the / operator overload.
	/= (see page 3444)	Divides this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) rhs.
	^ (see page 3445)	This is the overview for the ^ operator overload.
	^= (see page 3445)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise xor operation.

	__int64 (see page 3446)	This is __int64, a member of class Variant.
	| (see page 3446)	This is the overview for the | operator overload.
	|= (see page 3446)	Returns this System::Variant::Variant (see page 3421) after performing a bitwise or operation.
	+ (see page 3447)	This is the overview for the + operator overload.
	+= (see page 3447)	Adds the System::Variant::Variant (see page 3421) specified by rhs to this System::Variant::Variant (see page 3421) object.
	< (see page 3448)	This is the overview for the < operator overload.
	<< (see page 3448)	This is the overview for the << operator overload.
	<<= (see page 3449)	Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3421).
	<= (see page 3449)	This is the overview for the <= operator overload.
	= (see page 3450)	This is the overview for the = operator overload.
	-= (see page 3454)	Subtracts the System::Variant::Variant (see page 3421) specified by rhs from this System::Variant::Variant (see page 3421) object.
	== (see page 3455)	This is the overview for the == operator overload.
	> (see page 3456)	This is the overview for the > operator overload.
	>= (see page 3456)	This is the overview for the >= operator overload.
	>> (see page 3457)	This is the overview for the >> operator overload.
	>>= (see page 3458)	Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 3421).
	AnsiString (see page 3458)	Converts the System::Variant::Variant (see page 3421) to an AnsiString object.
	bool (see page 3458)	Converts the System::Variant::Variant (see page 3421) to a boolean value.
	char (see page 3459)	This is char, a member of class Variant.
	Currency (see page 3459)	Converts the System::Variant::Variant (see page 3421) to a Currency object.
	Currency* (see page 3459)	This is Currency*, a member of class Variant.
	double (see page 3460)	Converts the System::Variant::Variant (see page 3421) to a double value.
	double* (see page 3460)	This is double*, a member of class Variant.
	float (see page 3460)	Converts the System::Variant::Variant (see page 3421) to a float value.
	float* (see page 3460)	This is float*, a member of class Variant.
	IDispatch* (see page 3461)	Converts the System::Variant (see page 3421) to an System::Variant::IDispatch (see page 3494)* interface pointer.
	IInterface* (see page 3461)	This is IInterface*, a member of class Variant.
	int (see page 3461)	Converts the System::Variant::Variant (see page 3421) to an integer value.
	int* (see page 3461)	This is int*, a member of class Variant.
	IUnknown* (see page 3462)	Converts the System::Variant (see page 3421) to an System::Variant::IUnknown (see page 3756)* interface pointer.
	long (see page 3462)	This is long, a member of class Variant.
	long* (see page 3462)	This is long*, a member of class Variant.
	short (see page 3462)	Converts the System::Variant::Variant (see page 3421) to a short value.
	short* (see page 3463)	This is short*, a member of class Variant.
	signed char (see page 3463)	This is signed char, a member of class Variant.
	signed char* (see page 3463)	This is signed char*, a member of class Variant.
	TDateTime (see page 3463)	Converts the System::Variant::Variant (see page 3421) to a TDateTime object.
	TDateTime* (see page 3464)	This is TDateTime*, a member of class Variant.
	UnicodeString (see page 3464)	This is UnicodeString, a member of class Variant.
	unsigned char (see page 3464)	Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value.
	unsigned char* (see page 3465)	Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value. And returns a reference.
	unsigned int (see page 3465)	This is unsigned int, a member of class Variant.
	unsigned int* (see page 3465)	This is unsigned int*, a member of class Variant.
	unsigned long (see page 3465)	This is unsigned long, a member of class Variant.
	unsigned long* (see page 3466)	This is unsigned long*, a member of class Variant.
	unsigned short (see page 3466)	Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value.
	unsigned short* (see page 3466)	Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value and returns a reference to the value.

	WideString (see page 3466)	This is WideString, a member of class Variant.

35.1.24.5.1 - Operator

This is the overview for the - operator overload.

Overload List

	Name	Description
	Variant::- (AnsiString&) (see page 3439)	Returns a new System::Variant (see page 3421) that has the negative value of this System::Variant (see page 3421).
	Variant::- () (see page 3439)	Returns a new System::Variant (see page 3421) that has the negative value of this System::Variant (see page 3421).
	Variant::- (double) (see page 3439)	Returns a new System::Variant (see page 3421) that has the negative value of this System::Variant (see page 3421).
	Variant::- (int) (see page 3439)	Returns a new System::Variant (see page 3421) that has the negative value of this System::Variant (see page 3421).
	Variant::- (Variant&) (see page 3439)	Returns a new System::Variant (see page 3421) that has the negative value of this System::Variant (see page 3421).

35.1.24.5.1.1 Variant::- Operator ()

Returns a new System::Variant (see page 3421) that has the negative value of this System::Variant (see page 3421).

Pascal

```
operator -(const rhs: Variant&);
operator -(const rhs: AnsiString&);
operator -(rhs: Integer);
operator -(rhs: double);
operator -;
```

C++

```
__fastcall Variant operator -(const Variant& rhs) const;
__fastcall Variant operator -(const AnsiString& rhs) const;
__fastcall Variant operator -(int rhs) const;
__fastcall Variant operator -(double rhs) const;
__fastcall Variant operator -() const;
```

Description

The unary - operator returns a new System::Variant (see page 3421) whose value has the opposite sign from this System::Variant (see page 3421) (effectively multiplying the System::Variant (see page 3421)'s value by –1).

35.1.24.5.2 Variant::! Operator

Returns a new System::Variant::Variant (see page 3421) that is the logical negation of this System::Variant::Variant (see page 3421).

Pascal

```
operator !;
```

C++

```
__fastcall Variant operator !() const;
```

Description

The System::Variant (see page 3421)::! returns 0 if the System::Variant::Variant (see page 3421) has a nonzero value and 1 if the value of the System::Variant::Variant (see page 3421) is 0 or NULL.

35.1.24.5.3 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	Variant::!= (char*) (see page 3440)	Compares the System::Variant::Variant (see page 3421) with another value, testing for inequality.
	Variant::!= (double) (see page 3440)	Compares the System::Variant::Variant (see page 3421) with another value, testing for inequality.
	Variant::!= (int) (see page 3440)	Compares the System::Variant::Variant (see page 3421) with another value, testing for inequality.
	Variant::!= (TDateTime) (see page 3440)	Compares the System::Variant::Variant (see page 3421) with another value, testing for inequality.
	Variant::!= (Variant&) (see page 3440)	Compares the System::Variant::Variant (see page 3421) with another value, testing for inequality.

35.1.24.5.3.1 Variant::!= Operator (TDateTime)

Compares the System::Variant::Variant (see page 3421) with another value, testing for inequality.

Pascal

```
operator !=(const rhs: Variant&);
operator !=(const rhs: char*);
operator !=(rhs: Integer);
operator !=(rhs: double);
operator !=(rhs: TDateTime);
```

C++

```
__fastcall bool operator !=(const Variant& rhs) const;
__fastcall bool operator !=(const char* rhs) const;
__fastcall bool operator !=(int rhs) const;
__fastcall bool operator !=(double rhs) const;
__fastcall bool operator !=(TDateTime rhs) const;
```

Description

The != System::Variant (see page 3421)::!= returns true if the System::Variant::Variant (see page 3421)'s value is not the same as the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.4 % Operator

This is the overview for the % operator overload.

Overload List

	Name	Description
	Variant::% (AnsiString&) (see page 3441)	Returns the remainder from dividing the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::% (double) (see page 3441)	Returns the remainder from dividing the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::% (int) (see page 3441)	Returns the remainder from dividing the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::% (Variant&) (see page 3441)	Returns the remainder from dividing the System::Variant::Variant (see page 3421)'s value by a specified value.

35.1.24.5.4.1 Variant::% Operator (AnsiString&)

Returns the remainder from dividing the System::Variant::Variant (see page 3421)'s value by a specified value.

Pascal

```
operator %(const rhs: Variant&);
operator %(const rhs: AnsiString&);
operator %(rhs: Integer);
operator %(rhs: double);
```

C++

```
__fastcall Variant operator %(const Variant& rhs) const;
__fastcall Variant operator %(const AnsiString& rhs) const;
__fastcall Variant operator %(int rhs) const;
__fastcall Variant operator %(double rhs) const;
```

Description

The System::Variant (see page 3421)::% returns a new System::Variant::Variant (see page 3421) that is the modulus of this System::Variant::Variant (see page 3421) and rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.5 Variant::%= Operator

Returns this System::Variant::Variant (see page 3421) as the remainder from dividing its original value and the System::Variant::Variant (see page 3421) specified by rhs.

Pascal

```
operator %=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator %=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an

ESystem::Variant::VariantError exception is thrown.

35.1.24.5.6 & Operator

This is the overview for the & operator overload.

Overload List

	Name	Description
	Variant::& (AnsiString&) (see page 3442)	Performs a bitwise and operation with a specified value.
	Variant::& (double) (see page 3442)	Performs a bitwise and operation with a specified value.
	Variant::& (int) (see page 3442)	Performs a bitwise and operation with a specified value.
	Variant::& (Variant&) (see page 3442)	Performs a bitwise and operation with a specified value.

35.1.24.5.6.1 Variant::& Operator (AnsiString&)

Performs a bitwise and operation with a specified value.

Pascal

```
operator &(const rhs: Variant&);
operator &(const rhs: AnsiString&);
operator &(rhs: Integer);
operator &(rhs: double);
```

C++

```
__fastcall Variant operator &(const Variant& rhs) const;
__fastcall Variant operator &(const AnsiString& rhs) const;
__fastcall Variant operator &(int rhs) const;
__fastcall Variant operator &(double rhs) const;
```

Description

The System::Variant (see page 3421)::& returns a new System::Variant::Variant (see page 3421) that is the result of a bitwise and operation on this System::Variant::Variant (see page 3421) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.7 Variant::&= Operator

Returns this System::Variant::Variant (see page 3421) after performing a bitwise and operation.

Pascal

```
operator &=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator &=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.8 * Operator

This is the overview for the * operator overload.

Overload List

	Name	Description
	Variant::* (AnsiString&) (see page 3443)	Multiplies the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::* (double) (see page 3443)	Multiplies the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::* (int) (see page 3443)	Multiplies the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::* (Variant&) (see page 3443)	Multiplies the System::Variant::Variant (see page 3421)'s value by a specified value.

35.1.24.5.8.1 Variant::* Operator (AnsiString&)

Multiplies the System::Variant::Variant (see page 3421)'s value by a specified value.

Pascal

```
operator *(const rhs: Variant&);
operator *(const rhs: AnsiString&);
operator *(rhs: Integer);
operator *(rhs: double);
```

C++

```
__fastcall Variant operator *(const Variant& rhs) const;
__fastcall Variant operator *(const AnsiString& rhs) const;
__fastcall Variant operator *(int rhs) const;
__fastcall Variant operator *(double rhs) const;
```

Description

The System::Variant (see page 3421)::* returns a new System::Variant::Variant (see page 3421) that is the result of multiplying this System::Variant::Variant (see page 3421) with rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.9 Variant::*= Operator

Multiplies this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) specified by rhs.

Pascal

```
operator *=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator *=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.10 / Operator

This is the overview for the / operator overload.

Overload List

	Name	Description
	Variant::/ (AnsiString&) (see page 3444)	Divides the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::/ (double) (see page 3444)	Divides the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::/ (int) (see page 3444)	Divides the System::Variant::Variant (see page 3421)'s value by a specified value.
	Variant::/ (Variant&) (see page 3444)	Divides the System::Variant::Variant (see page 3421)'s value by a specified value.

35.1.24.5.10.1 Variant::/ Operator (AnsiString&)

Divides the System::Variant::Variant (see page 3421)'s value by a specified value.

Pascal

```
operator /(const rhs: Variant&);
operator /(const rhs: AnsiString&);
operator /(rhs: Integer);
operator /(rhs: double);
```

C++

```
__fastcall Variant operator /(const Variant& rhs) const;
__fastcall Variant operator /(const AnsiString& rhs) const;
__fastcall Variant operator /(int rhs) const;
__fastcall Variant operator /(double rhs) const;
```

Description

The System::Variant (see page 3421)::/ returns a new System::Variant::Variant (see page 3421) that is the result of dividing this System::Variant::Variant (see page 3421) by rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.11 Variant::/= Operator

Divides this System::Variant::Variant (see page 3421) by the System::Variant::Variant (see page 3421) rhs.

Pascal

```
operator /=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator /=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.12 ^ Operator

This is the overview for the ^ operator overload.

Overload List

	Name	Description
	Variant::^ (AnsiString&) (see page 3445)	Performs a bitwise xor operation with a specified value.
	Variant::^ (double) (see page 3445)	Performs a bitwise xor operation with a specified value.
	Variant::^ (int) (see page 3445)	Performs a bitwise xor operation with a specified value.
	Variant::^ (Variant&) (see page 3445)	Performs a bitwise xor operation with a specified value.

35.1.24.5.12.1 Variant::^ Operator (AnsiString&)

Performs a bitwise xor operation with a specified value.

Pascal

```
operator ^(const rhs: Variant&);
operator ^(const rhs: AnsiString&);
operator ^(rhs: Integer);
operator ^(rhs: double);
```

C++

```
__fastcall Variant operator ^(const Variant& rhs) const;
__fastcall Variant operator ^(const AnsiString& rhs) const;
__fastcall Variant operator ^(int rhs) const;
__fastcall Variant operator ^(double rhs) const;
```

Description

The ^ System::Variant (see page 3421)::^ returns a new System::Variant::Variant (see page 3421) that is the result of a bitwise xor operation on this System::Variant::Variant (see page 3421) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.13 Variant::^= Operator

Returns this System::Variant::Variant (see page 3421) after performing a bitwise xor operation.

Pascal

```
operator ^=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator ^=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.14 Variant::__int64 Operator

Pascal

```
operator __int64;
```

C++

```
__fastcall operator __int64() const;
```

Description

This is __int64, a member of class Variant.

35.1.24.5.15 | Operator

This is the overview for the | operator overload.

Overload List

	Name	Description
	Variant::| (AnsiString&) (see page 3446)	Performs a bitwise or operation with a specified value.
	Variant::| (double) (see page 3446)	Performs a bitwise or operation with a specified value.
	Variant::| (int) (see page 3446)	Performs a bitwise or operation with a specified value.
	Variant::| (Variant&) (see page 3446)	Performs a bitwise or operation with a specified value.

35.1.24.5.15.1 Variant::| Operator (AnsiString&)

Performs a bitwise or operation with a specified value.

Pascal

```
operator |(const rhs: Variant&);
operator |(const rhs: AnsiString&);
operator |(rhs: Integer);
operator |(rhs: double);
```

C++

```
__fastcall Variant operator |(const Variant& rhs) const;
__fastcall Variant operator |(const AnsiString& rhs) const;
__fastcall Variant operator |(int rhs) const;
__fastcall Variant operator |(double rhs) const;
```

Description

The | System::Variant (see page 3421)::| returns a new System::Variant::Variant (see page 3421) that is the result of a bitwise or operation on this System::Variant::Variant (see page 3421) using rhs as the rvalue.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.16 Variant::|= Operator

Returns this System::Variant::Variant (see page 3421) after performing a bitwise or operation.

Pascal

```
operator |=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator |=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.17 + Operator

This is the overview for the + operator overload.

Overload List

	Name	Description
	Variant::+ (double) (see page 3447)	Adds a specified value to the System::Variant::Variant (see page 3421).
	Variant::+ (int) (see page 3447)	Adds a specified value to the System::Variant::Variant (see page 3421).
	Variant::+ (Variant&) (see page 3447)	Adds a specified value to the System::Variant::Variant (see page 3421).

35.1.24.5.17.1 Variant::+ Operator (Variant&)

Adds a specified value to the System::Variant::Variant (see page 3421).

Pascal

```
operator +(const rhs: Variant&);
operator +(rhs: Integer);
operator +(rhs: double);
```

C++

```
__fastcall Variant operator +(const Variant& rhs) const;
__fastcall Variant operator +(int rhs) const;
__fastcall Variant operator +(double rhs) const;
```

Description

The + System::Variant (see page 3421)::+ returns a new System::Variant::Variant (see page 3421) that is the sum of this System::Variant::Variant (see page 3421) and rhs.

If the operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.18 Variant::+= Operator

Adds the System::Variant::Variant (see page 3421) specified by rhs to this System::Variant::Variant (see page 3421) object.

Pascal

```
operator +=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator +=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.19 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	Variant::< (char*) (see page 3448)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than a specified value.
	Variant::< (double) (see page 3448)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than a specified value.
	Variant::< (int) (see page 3448)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than a specified value.
	Variant::< (TDateTime) (see page 3448)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than a specified value.
	Variant::< (Variant&) (see page 3448)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than a specified value.

35.1.24.5.19.1 Variant::< Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 3421)'s value is less than a specified value.

Pascal

```
operator <(const rhs: Variant&);
operator <(const rhs: char*);
operator <(rhs: Integer);
operator <(rhs: double);
operator <(rhs: TDateTime);
```

C++

```
__fastcall bool operator <(const Variant& rhs) const;
__fastcall bool operator <(const char* rhs) const;
__fastcall bool operator <(int rhs) const;
__fastcall bool operator <(double rhs) const;
__fastcall bool operator <(TDateTime rhs) const;
```

Description

The < System::Variant (see page 3421)::< returns true if the System::Variant::Variant (see page 3421)'s value is less than the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.20 << Operator

This is the overview for the << operator overload.

Overload List

	Name	Description
	Variant::<< (AnsiString&) (see page 3449)	Writes the Variant (see page 3421) to a specified stream.
	Variant::<< (double) (see page 3449)	Writes the Variant (see page 3421) to a specified stream.
	Variant::<< (int) (see page 3449)	Writes the Variant (see page 3421) to a specified stream.
	Variant::<< (Variant&) (see page 3449)	Writes the Variant (see page 3421) to a specified stream.

35.1.24.5.20.1 Variant::<< Operator (AnsiString&)

Writes the Variant (see page 3421) to a specified stream.

Pascal

```
operator <<(const rhs: Variant&);
operator <<(const rhs: AnsiString&);
operator <<(rhs: Integer);
operator <<(rhs: double);
```

C++

```
__fastcall Variant operator <<(const Variant& rhs) const;
__fastcall Variant operator <<(const AnsiString& rhs) const;
__fastcall Variant operator <<(int rhs) const;
__fastcall Variant operator <<(double rhs) const;
```

Description

The << operator is a friend operator that writes out the Variant (see page 3421) specified by arg to the stream specified by os.

Note: This operator is only definedif you #include <iostream> or #define VCL_IOSTREAM before the include statement for sysvari.h (which is typically included indirectly via vcl.h).

35.1.24.5.21 Variant::<<= Operator

Performs a bitwise shl (left shift) operation on this System::Variant::Variant (see page 3421).

Pascal

```
operator <<=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator <<=(const Variant& rhs);
```

Description

If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.22 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	Variant::<= (char*) (see page 3450)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than or equal to a specified value.
	Variant::<= (double) (see page 3450)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than or equal to a specified value.
	Variant::<= (int) (see page 3450)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than or equal to a specified value.
	Variant::<= (TDateTime) (see page 3450)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than or equal to a specified value.
	Variant::<= (Variant&) (see page 3450)	Tests whether the System::Variant::Variant (see page 3421)'s value is less than or equal to a specified value.

35.1.24.5.22.1 Variant::<= Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 3421)'s value is less than or equal to a specified value.

Pascal

```
operator <=(const rhs: Variant&);
operator <=(const rhs: char*);
operator <=(rhs: Integer);
operator <=(rhs: double);
operator <=(rhs: TDateTime);
```

C++

```
__fastcall bool operator <=(const Variant& rhs) const;
__fastcall bool operator <=(const char* rhs) const;
__fastcall bool operator <=(int rhs) const;
__fastcall bool operator <=(double rhs) const;
__fastcall bool operator <=(TDateTime rhs) const;
```

Description

The <= System::Variant (see page 3421)::<= returns true if the System::Variant::Variant (see page 3421)'s value is less than or equal to the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an EVariantErrorError exception is thrown.

35.1.24.5.23 = Operator

This is the overview for the = operator overload.

Overload List

	Name	Description
	Variant::= (__int64) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (__int64*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (AnsiString&) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).

	Variant::= (bool) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (char) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (char*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (Currency&) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (Currency*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (double) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (double*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (float) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (float*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (IDispatch*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (IDispatch**) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (IInterface*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (IInterface**) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (int) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (int*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (IUnknown*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (IUnknown**) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).

	Variant::= (long) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (long*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (short) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (short*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (signed char) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (signed char*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (System::DelphiInterface<IDispatch>&) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (TDateTime&) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (TDateTime*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (UnicodeString&) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (unsigned char) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (unsigned char*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (unsigned int) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (unsigned int*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (unsigned long) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (unsigned long*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (unsigned short) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).

	Variant::= (unsigned short*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (Variant&) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (Variant*) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).
	Variant::= (WideString&) (see page 3453)	Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).

35.1.24.5.23.1 Variant::= Operator (AnsiString&)

Copies the System::Variant::Variant (see page 3421) specified by rhs into this System::Variant::Variant (see page 3421).

Pascal

```
operator =(const rhs: Variant&);
operator =(const src: bool);
operator =(const src: char);
operator =(const src: signed char);
operator =(const src: unsigned char);
operator =(const src: short);
operator =(const src: unsigned short);
operator =(const src: Integer);
operator =(const src: unsigned int);
operator =(const src: long);
operator =(const src: unsigned long);
operator =(const src: float);
operator =(const src: double);
operator =(const src: __int64);
operator =(const src: AnsiString&);
operator =(const src: Currency&);
operator =(const src: TDateTime&);
operator =(const src: WideString&);
operator =(const src: UnicodeString&);
operator =(const src: System::DelphiInterface<IDispatch>&);
operator =(var src: IUnknown);
operator =(var src: IInterface);
operator =(var src: IDispatch);
operator =(const src: char*);
operator =(var src: signed char);
operator =(var src: unsigned char);
operator =(var src: short);
operator =(var src: unsigned short);
operator =(var src: Integer);
operator =(var src: unsigned int);
operator =(var src: long);
operator =(var src: unsigned long);
operator =(var src: float);
operator =(var src: double);
operator =(var src: __int64);
operator =(var src: IUnknown*);
operator =(var src: IInterface*);
operator =(var src: IDispatch*);
operator =(var src: Currency);
operator =(var src: TDateTime);
operator =(var src: Variant);
```

C++

```
__fastcall Variant& operator =(const Variant& rhs);
__fastcall Variant& operator =(const bool src);
__fastcall Variant& operator =(const char src);
__fastcall Variant& operator =(const signed char src);
__fastcall Variant& operator =(const unsigned char src);
__fastcall Variant& operator =(const short src);
__fastcall Variant& operator =(const unsigned short src);
__fastcall Variant& operator =(const int src);
__fastcall Variant& operator =(const unsigned int src);
__fastcall Variant& operator =(const long src);
__fastcall Variant& operator =(const unsigned long src);
__fastcall Variant& operator =(const float src);
__fastcall Variant& operator =(const double src);
__fastcall Variant& operator =(const __int64 src);
__fastcall Variant& operator =(const AnsiString& src);
__fastcall Variant& operator =(const Currency& src);
__fastcall Variant& operator =(const TDateTime& src);
__fastcall Variant& operator =(const WideString& src);
__fastcall Variant& operator =(const UnicodeString& src);
__fastcall Variant& operator =(const System::DelphiInterface<IDispatch>& src);
__fastcall Variant& operator =(IUnknown* src);
__fastcall Variant& operator =(IInterface* src);
__fastcall Variant& operator =(IDispatch* src);
__fastcall Variant& operator =(const char* src);
__fastcall Variant& operator =(signed char* src);
__fastcall Variant& operator =(unsigned char* src);
__fastcall Variant& operator =(short* src);
__fastcall Variant& operator =(unsigned short* src);
__fastcall Variant& operator =(int* src);
__fastcall Variant& operator =(unsigned int* src);
__fastcall Variant& operator =(long* src);
__fastcall Variant& operator =(unsigned long* src);
__fastcall Variant& operator =(float* src);
__fastcall Variant& operator =(double* src);
__fastcall Variant& operator =(__int64* src);
__fastcall Variant& operator =(IUnknown** src);
__fastcall Variant& operator =(IInterface** src);
__fastcall Variant& operator =(IDispatch** src);
__fastcall Variant& operator =(Currency* src);
__fastcall Variant& operator =(TDateTime* src);
__fastcall Variant& operator =(Variant* src);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.24 Variant::-= Operator

Subtracts the System::Variant::Variant (see page 3421) specified by rhs from this System::Variant::Variant (see page 3421) object.

Pascal

```
operator -=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator -=(const Variant& rhs);
```

Description

The actual types of the System::Variant::Variants (see page 4973) determine the meaning of this operation. If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.25 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	Variant::== (AnsiString&) (see page 3455)	Compares the System::Variant::Variant (see page 3421) with another value, testing for equality.
	Variant::== (char*) (see page 3455)	Compares the System::Variant::Variant (see page 3421) with another value, testing for equality.
	Variant::== (double) (see page 3455)	Compares the System::Variant::Variant (see page 3421) with another value, testing for equality.
	Variant::== (int) (see page 3455)	Compares the System::Variant::Variant (see page 3421) with another value, testing for equality.
	Variant::== (TDateTime) (see page 3455)	Compares the System::Variant::Variant (see page 3421) with another value, testing for equality.
	Variant::== (Variant&) (see page 3455)	Compares the System::Variant::Variant (see page 3421) with another value, testing for equality.

35.1.24.5.25.1 Variant::== Operator (AnsiString&)

Compares the System::Variant::Variant (see page 3421) with another value, testing for equality.

Pascal

```
operator ==(const rhs: Variant&);
operator ==(const rhs: char*);
operator ==(rhs: Integer);
operator ==(const rhs: AnsiString&);
operator ==(rhs: double);
operator ==(rhs: TDateTime);
```

C++

```
__fastcall bool operator ==(const Variant& rhs) const;
__fastcall bool operator ==(const char* rhs) const;
__fastcall bool operator ==(int rhs) const;
__fastcall bool operator ==(const AnsiString& rhs) const;
__fastcall bool operator ==(double rhs) const;
__fastcall bool operator ==(TDateTime rhs) const;
```

Description

The == System::Variant (see page 3421)::== returns true if the System::Variant::Variant (see page 3421)'s value is the same as the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.26 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	Variant::> (char*) (see page 3456)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than a specified value.
	Variant::> (double) (see page 3456)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than a specified value.
	Variant::> (int) (see page 3456)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than a specified value.
	Variant::> (TDateTime) (see page 3456)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than a specified value.
	Variant::> (Variant&) (see page 3456)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than a specified value.

35.1.24.5.26.1 Variant::> Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 3421)'s value is greater than a specified value.

Pascal

```
operator >(const rhs: Variant&);
operator >(const rhs: char*);
operator >(rhs: Integer);
operator >(rhs: double);
operator >(rhs: TDateTime);
```

C++

```
__fastcall bool operator >(const Variant& rhs) const;
__fastcall bool operator >(const char* rhs) const;
__fastcall bool operator >(int rhs) const;
__fastcall bool operator >(double rhs) const;
__fastcall bool operator >(TDateTime rhs) const;
```

Description

The > System::Variant (see page 3421)::> returns true if the System::Variant::Variant (see page 3421)'s value is greater than the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.27 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	Variant::>= (char*) (see page 3457)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than or equal to a specified value.
	Variant::>= (double) (see page 3457)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than or equal to a specified value.

	Variant::>= (int) (see page 3457)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than or equal to a specified value.
	Variant::>= (TDateTime) (see page 3457)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than or equal to a specified value.
	Variant::>= (Variant&) (see page 3457)	Tests whether the System::Variant::Variant (see page 3421)'s value is greater than or equal to a specified value.

35.1.24.5.27.1 Variant::>= Operator (TDateTime)

Tests whether the System::Variant::Variant (see page 3421)'s value is greater than or equal to a specified value.

Pascal

```
operator >=(const rhs: Variant&);
operator >=(const rhs: char*);
operator >=(rhs: Integer);
operator >=(rhs: double);
operator >=(rhs: TDateTime);
```

C++

```
__fastcall bool operator >=(const Variant& rhs) const;
__fastcall bool operator >=(const char* rhs) const;
__fastcall bool operator >=(int rhs) const;
__fastcall bool operator >=(double rhs) const;
__fastcall bool operator >=(TDateTime rhs) const;
```

Description

The >= System::Variant (see page 3421)::>= returns true if the System::Variant::Variant (see page 3421)'s value is greater than or equal to the value specified by rhs, false otherwise. If the comparison is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.28 >> Operator

This is the overview for the >> operator overload.

Overload List

	Name	Description
	Variant::>> (AnsiString&) (see page 3457)	Returns a new System::Variant (see page 3421) that is the result of a shr (right shift) operation on this System::Variant (see page 3421).
	Variant::>> (double) (see page 3457)	Returns a new System::Variant (see page 3421) that is the result of a shr (right shift) operation on this System::Variant (see page 3421).
	Variant::>> (int) (see page 3457)	Returns a new System::Variant (see page 3421) that is the result of a shr (right shift) operation on this System::Variant (see page 3421).
	Variant::>> (Variant&) (see page 3457)	Returns a new System::Variant (see page 3421) that is the result of a shr (right shift) operation on this System::Variant (see page 3421).

35.1.24.5.28.1 Variant::>> Operator (AnsiString&)

Returns a new System::Variant (see page 3421) that is the result of a shr (right shift) operation on this System::Variant (see

page 3421).

Pascal

```
operator >>(const rhs: Variant&);
operator >>(const rhs: AnsiString&);
operator >>(rhs: Integer);
operator >>(rhs: double);
```

C++

```
__fastcall Variant operator >>(const Variant& rhs) const;
__fastcall Variant operator >>(const AnsiString& rhs) const;
__fastcall Variant operator >>(int rhs) const;
__fastcall Variant operator >>(double rhs) const;
```

Description

The >> operator returns a new System::Variant (see page 3421) that is the result of a shr (right shift) operation, shifting by the number of bits specified by rhs.

If the operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.29 Variant::>>= Operator

Performs a bitwise shr (right shift) operation on this System::Variant::Variant (see page 3421).

Pascal

```
operator >>=(const rhs: Variant&);
```

C++

```
__fastcall Variant& operator >>=(const Variant& rhs);
```

Description

If the System::Variant::Variants (see page 4973) can be converted to types that make sense for the given operation, the assignment is performed. If any of these operations is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.30 Variant::AnsiString Operator

Converts the System::Variant::Variant (see page 3421) to an AnsiString object.

Pascal

```
operator AnsiString;
```

C++

```
__fastcall operator AnsiString() const;
```

Description

This conversion System::Variant::AnsiString returns an AnsiString object that is a conversion of this System::Variant::Variant (see page 3421).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.31 Variant::bool Operator

Converts the System::Variant::Variant (see page 3421) to a boolean value.

Pascal

```
operator bool;
```

C++

```
__fastcall operator bool() const;
```

Description

This conversion System::Variant::bool returns a bool that is a conversion of this System::Variant::Variant (see page 3421).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.32 Variant::char Operator

Pascal

```
operator char;
```

C++

```
__fastcall operator char() const;
```

Description

This is char, a member of class Variant.

35.1.24.5.33 Variant::Currency Operator

Converts the System::Variant::Variant (see page 3421) to a Currency object.

Pascal

```
operator Currency;
```

C++

```
__fastcall operator Currency() const;
```

Description

The Currency conversion System::Variant::Currency returns a Currency or a reference to a Currency that is a conversion of this System::Variant::Variant (see page 3421). If the Currency* conversion is used, the System::Variant::Currency sets the varByRef (see page 3862) bit, meaning that the System::Variant::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.34 Variant::Currency* Operator

Pascal

```
operator Currency*;
```

C++

```
__fastcall operator Currency*();
```

Description

This is Currency*, a member of class Variant.

35.1.24.5.35 Variant::double Operator

Converts the System::Variant::Variant (see page 3421) to a double value.

Pascal

```
operator double;
```

C++

```
__fastcall operator double() const;
```

Description

The double conversion System::Variant::double returns a double or a reference to a double that is a conversion of this System::Variant::Variant (see page 3421). If the double* conversion is used, the System::Variant::double sets the varByRef (see page 3862) bit, meaning that the System::Variant::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.36 Variant::double* Operator

Pascal

```
operator double*;
```

C++

```
__fastcall operator double*();
```

Description

This is double*, a member of class Variant.

35.1.24.5.37 Variant::float Operator

Converts the System::Variant::Variant (see page 3421) to a float value.

Pascal

```
operator float;
```

C++

```
__fastcall operator float() const;
```

Description

The float conversion System::Variant::float returns a float or a reference to a float that is a conversion of this System::Variant::Variant (see page 3421). If the float* conversion is used, the System::Variant::float sets the varByRef (see page 3862) bit, meaning that the System::Variant::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.38 Variant::float* Operator

Pascal

```
operator float*;
```

C++

```
__fastcall operator float*();
```

Description

This is float*, a member of class Variant.

35.1.24.5.39 Variant::IDispatch* Operator

Converts the System::Variant (see page 3421) to an System::Variant::IDispatch (see page 3494)* interface pointer.

Pascal

```
operator IDispatch*;
```

C++

```
__fastcall operator IDispatch*();
```

Description

This conversion operator returns an OLE Automation object System::Variant::IDispatch (see page 3494)* interface pointer that is a conversion of this System::Variant (see page 3421).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::VariantError exception is thrown.

35.1.24.5.40 Variant::IInterface* Operator

Pascal

```
operator IInterface*;
```

C++

```
__fastcall operator IInterface*();
```

Description

This is IInterface*, a member of class Variant.

35.1.24.5.41 Variant::int Operator

Converts the System::Variant::Variant (see page 3421) to an integer value.

Pascal

```
operator int;
```

C++

```
__fastcall operator int() const;
```

Description

The int conversion System::Variant::int returns an integer or a reference to an integer that is a conversion of this System::Variant::Variant (see page 3421). If the int* conversion is used, the System::Variant::int sets the varByRef (see page 3862) bit, meaning that the System::Variant::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown.

35.1.24.5.42 Variant::int* Operator

Pascal

```
operator int*;
```

C++

```
__fastcall operator int*();
```

Description

This is int*, a member of class Variant.

35.1.24.5.43 Variant::IUnknown* Operator

Converts the System::Variant (see page 3421) to an System::Variant::IUnknown (see page 3756)* interface pointer.

Pascal

```
operator IUnknown*;
```

C++

```
__fastcall operator IUnknown*();
```

Description

This conversion operator returns an OLE object System::Variant::IUnknown (see page 3756)* interface pointer that is a conversion of this System::Variant (see page 3421).

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::VariantError exception is thrown.

35.1.24.5.44 Variant::long Operator

Pascal

```
operator long;
```

C++

```
__fastcall operator long() const;
```

Description

This is long, a member of class Variant.

35.1.24.5.45 Variant::long* Operator

Pascal

```
operator long*;
```

C++

```
__fastcall operator long*();
```

Description

This is long*, a member of class Variant.

35.1.24.5.46 Variant::short Operator

Converts the System::Variant::Variant (see page 3421) to a short value.

Pascal

```
operator short;
```

C++

```
__fastcall operator short() const;
```

Description

The short conversion System::Variant::short returns a short or a reference to a short that is a conversion of this System::Variant::Variant (see page 3421). If the short* conversion is used, the System::Variant::short sets the varByRef (see page 3862) bit, meaning that the System::Variant::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.47 Variant::short* Operator

Pascal

```
operator short*;
```

C++

```
__fastcall operator short*();
```

Description

This is short*, a member of class Variant.

35.1.24.5.48 Variant::signed char Operator

Pascal

```
operator signed char;
```

C++

```
__fastcall operator signed char() const;
```

Description

This is signed char, a member of class Variant.

35.1.24.5.49 Variant::signed char* Operator

Pascal

```
operator signed char*;
```

C++

```
__fastcall operator signed char*();
```

Description

This is signed char*, a member of class Variant.

35.1.24.5.50 Variant::TDateTime Operator

Converts the System::Variant::Variant (see page 3421) to a TDateTime object.

Pascal

```
operator TDateTime;
```

C++

```
__fastcall operator TDateTime() const;
```

Description

The TDateTime conversion System::Variant::TDateTime returns a TDateTime or a reference to a TDateTime that is a conversion of this System::Variant::Variant (see page 3421). If the TDateTime* conversion is used, the System::Variant::TDateTime sets the varByRef (see page 3862) bit, meaning that the System::Variant::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an ESystem::Variant::VariantError exception is thrown.

35.1.24.5.51 Variant::TDateTime* Operator

Pascal

```
operator TDateTime*;
```

C++

```
__fastcall operator TDateTime*();
```

Description

This is TDateTime*, a member of class Variant.

35.1.24.5.52 Variant::UnicodeString Operator

Pascal

```
operator UnicodeString;
```

C++

```
__fastcall operator UnicodeString() const;
```

Description

This is UnicodeString, a member of class Variant.

35.1.24.5.53 Variant::unsigned char Operator

Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value.

Pascal

```
operator unsigned char;
```

C++

```
__fastcall operator unsigned char() const;
```

Description

The System::Variant::unsigned char (see page 3459) conversion operator returns an unsigned char (see page 3459) that is a conversion of this System::Variant::Variant (see page 3421).

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown. The System::Variant::unsigned char (see page 3459) operator replaces the Byte operator.

35.1.24.5.54 Variant::unsigned char* Operator

Converts the System::Variant (see page 3421) to an unsigned char (see page 3459) value. And returns a reference.

Pascal

```
operator unsigned char*;
```

C++

```
__fastcall operator unsigned char*();
```

Description

The System::Variant::unsigned char (see page 3459)* conversion operator returns a reference to an unsigned char (see page 3459) that is a conversion of this System::Variant::Variant (see page 3421). System::Variant::unsigned char (see page 3459)* sets the varByRef (see page 3862) bit, meaning that the System::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown. The System::Variant::unsigned char (see page 3459)* operator replaces the Byte* operator.

35.1.24.5.55 Variant::unsigned int Operator

Pascal

```
operator unsigned int;
```

C++

```
__fastcall operator unsigned int() const;
```

Description

This is unsigned int, a member of class Variant.

35.1.24.5.56 Variant::unsigned int* Operator

Pascal

```
operator unsigned int*;
```

C++

```
__fastcall operator unsigned int*();
```

Description

This is unsigned int*, a member of class Variant.

35.1.24.5.57 Variant::unsigned long Operator

Pascal

```
operator unsigned long;
```

C++

```
__fastcall operator unsigned long() const;
```

Description

This is unsigned long, a member of class Variant.

35.1.24.5.58 Variant::unsigned long* Operator

Pascal

```
operator unsigned long*;
```

C++

```
__fastcall operator unsigned long*();
```

Description

This is unsigned long*, a member of class Variant.

35.1.24.5.59 Variant::unsigned short Operator

Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value.

Pascal

```
operator unsigned short;
```

C++

```
__fastcall operator unsigned short() const;
```

Description

The System::Variant::unsigned short (see page 3462) conversion operator returns an unsigned short (see page 3462) that is a conversion of this System::Variant (see page 3421).

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown. The System::Variant::unsigned short (see page 3462) operator replaces the WordBool operator.

35.1.24.5.60 Variant::unsigned short* Operator

Converts the System::Variant (see page 3421) to an unsigned short (see page 3462) value and returns a reference to the value.

Pascal

```
operator unsigned short*;
```

C++

```
__fastcall operator unsigned short*();
```

Description

The System::Variant::unsigned short (see page 3462)* conversion operator returns a reference to an unsigned short (see page 3462) that is a conversion of this System::Variant (see page 3421). System::Variant::unsigned short (see page 3462)* sets the varByRef (see page 3862) bit, meaning that the System::Variant (see page 3421) contains a reference as opposed to a value.

If the conversion operation is not possible because of an invalid variant type conversion, an EVariantError exception is thrown. The System::Variant::unsigned short (see page 3462)* operator replaces the WordBool* operator.

35.1.24.5.61 Variant::WideString Operator

Pascal

```
operator WideString;
```

C++

```
__fastcall operator WideString() const;
```

Description

This is WideString, a member of class Variant.

35.1.25 WideString Class

System::WideString represents strings of 16-bit characters.

Class Hierarchy

Pascal

```
WideString = class;
```

C++

```
class WideString;
```

File

wstring.h

Description

System::WideString represents strings of 16-bit characters. Each character is stored as a wchar_t value. The C++ class System::WideString is functionally equivalent to a Delphi intrinsic type, also called System::WideString.

Note: Although System::WideString is a platform-independent data (see page 3476) type and often appears in cross-platform code, System::WideString is also useful for working with Microsoft's Component Object Model (COM) System::WideString values are easily converted to or from the BSTR values used in COM.

35.1.25.1 WideString Members

The following tables list the members exposed by WideString.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3471)	This is record System::WideString::TStringFloatFormat.

Public Methods

	Name	Description
	~WideString (see page 3469)	Frees an instance ofSystem::WideString::WideString (see page 3467).
	c_bstr (see page 3473)	Returns the underlying BSTR that implements theSystem::WideString::WideString (see page 3467).
	cat_printf (see page 3473)	Appends to the value of the System::WideString (see page 3467) given a format string and its arguments.
	cat_sprintf (see page 3474)	Appends the result of a given format string and its arguments to the end of the System::WideString (see page 3467).
	cat_vprintf (see page 3474)	Appends the result of a given format string and its arguments to the end of the System::WideString (see page 3467).
	Copy (see page 3475)	Returns a copy of theSystem::WideString::WideString (see page 3467).
	CurrToStr (see page 3475)	Converts a currency value to its string representation.

	CurrToStrF (see page 3475)	Converts a currency value to its string representation using the indicated float format convention.
	data (see page 3476)	Returns a pointer to the underlying string System::WideString::data.
	Delete (see page 3476)	System::WideString::Deletes a substring from the System::WideString::WideString (see page 3467).
	FloatToStrF (see page 3477)	Returns the string representation of a floating-point value.
	Format (see page 3477)	System::WideString::Formats a string assembled from an System::WideString (see page 3467), format, and a series of array arguments, args.
	FormatFloat (see page 3478)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3479)	System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 3467).
	IntToHex (see page 3480)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3480)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 3480)	Returns true if the System::WideString::WideString (see page 3467) is empty.
	IsPathDelimiter (see page 3481)	Returns true if the specified character is '\'.
	LastChar (see page 3481)	Returns a pointer to the last character in a string.
	LastDelimiter (see page 3481)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 3481)	Returns the length of theSystem::WideString::WideString (see page 3467).
	LowerCase (see page 3482)	Returns a lower-case version of System::WideString (see page 3467).
	Pos (see page 3482)	Returns the index at which a specified substring begins.
	printf (see page 3482)	Sets the value of the System::WideString (see page 3467) given a format string and its arguments.
	SetLength (see page 3483)	Changes the length of theSystem::WideString::WideString (see page 3467).
	sprintf (see page 3483)	Sets the value of the System::WideString (see page 3467) given a format string and its arguments.
	StringOfChar (see page 3484)	Returns a string with the specified number of characters.
	SubString (see page 3484)	Returns a specified substring of theSystem::WideString::WideString (see page 3467).
	swap (see page 3484)	Swap string pointers.
	ToDouble (see page 3485)	Converts the string to a floating-point value.
	ToInt (see page 3485)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3485)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3486)	Returns a new System::WideString (see page 3467) without leading spaces, trailing spaces, or control characters.
	TrimLeft (see page 3486)	Returns a new System::WideString (see page 3467) without leading spaces or control characters.
	TrimRight (see page 3486)	Returns a new System::WideString (see page 3467) without trailing spaces or control characters.
	UpperCase (see page 3486)	Returns an upper-case version of this System::WideString (see page 3467).
	vprintf (see page 3487)	Sets the value of the System::WideString (see page 3467) given a format string and its arguments.
	WideString (see page 3469)	This is the overview for the WideString constructor overload.

Public Operators

	Name	Description
	!= (see page 3487)	This is the overview for the != operator overload.
	[] (see page 3488)	This is the overview for the [] operator overload.
	+ (see page 3489)	Concatenates twoSystem::WideString::WideStrings (see page 6807).
	+= (see page 3489)	Concatenates this System::WideString::WideString (see page 3467) and rhs, and assigns the result to this System::WideString::WideString (see page 3467).
	< (see page 3489)	This is the overview for the < operator overload.
	<= (see page 3490)	This is the overview for the <= operator overload.
	= (see page 3490)	Assigns the value of theSystem::WideString::WideString (see page 3467) to equal a source string.
	== (see page 3491)	This is the overview for the == operator overload.
	> (see page 3491)	This is the overview for the > operator overload.

	>= (see page 3492)	This is the overview for the >= operator overload.

35.1.25.2 WideString::~WideString Destructor

Frees an instance ofSystem::WideString::WideString (see page 3467).

Pascal

```
destructor Destroy;
```

C++

```
__fastcall ~WideString();
```

Description

System::WideString (see page 3467)::~WideString (see page 3467) is the destructor forSystem::WideString::WideString (see page 3467).

35.1.25.3 WideString Constructor

This is the overview for the WideString constructor overload.

Overload List

	Name	Description
	WideString::WideString () (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (__int64) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (AnsiStringT<codePage> &) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (char) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (char*) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (char*, int) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (char16_t*, int) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (char32_t*, int) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (double) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (float) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (int) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (long) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (long double) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).

	WideString::WideString (short) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (UnicodeString&) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (unsigned __int64) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (unsigned int) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (unsigned long) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (unsigned short) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (WideChar) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (WideChar*) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (WideChar*, int) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).
	WideString::WideString (WideString&) (see page 3470)	Creates an instance of System::WideString::WideString (see page 3467).

35.1.25.3.1 WideString::WideString Constructor ()

Creates an instance of System::WideString::WideString (see page 3467).

Pascal

```
constructor Create;
constructor Create(const src: char*);
constructor Create(const src: WideString&);
constructor Create(const src: WideChar*; len: Integer);
constructor Create(const src: char*; len: Integer);
constructor Create(const src: WideChar*);
constructor Create(const src: char16_t*; numChar16: Integer = -1);
constructor Create(const src: char32_t*; numChar32: Integer = -1);
constructor Create(const src: WideChar);
constructor Create(src: char);
constructor Create(src: short);
constructor Create(: unsigned short);
constructor Create(src: Integer);
constructor Create(: unsigned int);
constructor Create(: long);
constructor Create(: unsigned long);
constructor Create(: __int64);
constructor Create(: unsigned __int64);
constructor Create(src: float);
constructor Create(src: double);
constructor Create(src: long double);
constructor Create(const src: UnicodeString&);
constructor Create<unsigned short codePage>(const src: AnsiStringT<codePage> &);
```

C++

```
__fastcall WideString();
__fastcall WideString(const char* src);
__fastcall WideString(const WideString& src);
__fastcall WideString(const WideChar* src, int len);
```

```
__fastcall WideString(const char* src, int len);
__fastcall WideString(const WideChar* src);
__fastcall WideString(const char16_t* src, int numChar16 = -1);
__fastcall WideString(const char32_t* src, int numChar32 = -1);
__fastcall explicit WideString(const WideChar src);
__fastcall explicit WideString(char src);
__fastcall explicit WideString(short src);
__fastcall explicit WideString(unsigned short);
__fastcall explicit WideString(int src);
__fastcall explicit WideString(unsigned int);
__fastcall explicit WideString(long);
__fastcall explicit WideString(unsigned long);
__fastcall explicit WideString(__int64);
__fastcall explicit WideString(unsigned __int64);
__fastcall explicit WideString(float src);
__fastcall explicit WideString(double src);
__fastcall explicit WideString(long double src);
__fastcall WideString(const UnicodeString& src);
template <unsigned short codePage> __fastcall WideString(const AnsiStringT<codePage> & src);
```

Description

Call System::WideString::WideString (see page 3467) to create and initialize a wide string. If the constructor is given no arguments, the resulting System::WideString::WideString (see page 3467) is an empty string. If the constructor is passed a single wide character, the resulting System::WideString::WideString (see page 3467) has length one, and contains that character. All other versions of the constructor copy the string passed as an argument, converting characters to wchar_t if necessary. When constructing a System::WideString::WideString (see page 3467) from an array of char or wchar_t, the src string should be null terminated, unless a second argument is given that indicates the number of characters in the array.

35.1.25.4 WideString Enumerations

The enumerations of the WideString class are listed here.

Enumerations

	Name	Description
	TStringFloatFormat (see page 3471)	This is record System::WideString::TStringFloatFormat.

35.1.25.4.1 System::WideString::TStringFloatFormat Enumeration

Pascal

```
TStringFloatFormat = (
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
);
```

C++

```
enum TStringFloatFormat {
  sffGeneral,
  sffExponent,
  sffFixed,
  sffNumber,
  sffCurrency
};
```

File

wstring.h

Description

This is record System::WideString::TStringFloatFormat.

35.1.25.5 WideString Methods

The methods of the WideString class are listed here.

Public Methods

	Name	Description
	c_bstr (see page 3473)	Returns the underlying BSTR that implements theSystem::WideString::WideString (see page 3467).
	cat_printf (see page 3473)	Appends to the value of the System::WideString (see page 3467) given a format string and its arguments.
	cat_sprintf (see page 3474)	Appends the result of a given format string and its arguments to the end of the System::WideString (see page 3467).
	cat_vprintf (see page 3474)	Appends the result of a given format string and its arguments to the end of the System::WideString (see page 3467).
	Copy (see page 3475)	Returns a copy of theSystem::WideString::WideString (see page 3467).
	CurrToStr (see page 3475)	Converts a currency value to its string representation.
	CurrToStrF (see page 3475)	Converts a currency value to its string representation using the indicated float format convention.
	data (see page 3476)	Returns a pointer to the underlying string System::WideString::data.
	Delete (see page 3476)	System::WideString::Deletes a substring from the System::WideString::WideString (see page 3467).
	FloatToStrF (see page 3477)	Returns the string representation of a floating-point value.
	Format (see page 3477)	System::WideString::Formats a string assembled from an System::WideString (see page 3467), format, and a series of array arguments, args.
	FormatFloat (see page 3478)	Formats the floating-point value given by value using the format string given by format.
	Insert (see page 3479)	System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 3467).
	IntToHex (see page 3480)	Converts a number into a string containing the number's hexadecimal (base 16) representation.
	IsDelimiter (see page 3480)	Returns true if the specified character matches any character in the delimiters string.
	IsEmpty (see page 3480)	Returns true if the System::WideString::WideString (see page 3467) is empty.
	IsPathDelimiter (see page 3481)	Returns true if the specified character is '\'.
	LastChar (see page 3481)	Returns a pointer to the last character in a string.
	LastDelimiter (see page 3481)	Returns the index of the rightmost character that matches any character in the delimiters string.
	Length (see page 3481)	Returns the length of theSystem::WideString::WideString (see page 3467).
	LowerCase (see page 3482)	Returns a lower-case version of System::WideString (see page 3467).
	Pos (see page 3482)	Returns the index at which a specified substring begins.
	printf (see page 3482)	Sets the value of the System::WideString (see page 3467) given a format string and its arguments.
	SetLength (see page 3483)	Changes the length of theSystem::WideString::WideString (see page 3467).
	sprintf (see page 3483)	Sets the value of the System::WideString (see page 3467) given a format string and its arguments.
	StringOfChar (see page 3484)	Returns a string with the specified number of characters.
	SubString (see page 3484)	Returns a specified substring of theSystem::WideString::WideString (see page 3467).
	swap (see page 3484)	Swap string pointers.
	ToDouble (see page 3485)	Converts the string to a floating-point value.
	ToInt (see page 3485)	Converts a string to an integer, throwing an exception if the string is invalid.
	ToIntDef (see page 3485)	Converts the string to an integer, returning a default value if the string is invalid.
	Trim (see page 3486)	Returns a new System::WideString (see page 3467) without leading spaces, trailing spaces, or control characters.

	TrimLeft (see page 3486)	Returns a new System::WideString (see page 3467) without leading spaces or control characters.
	TrimRight (see page 3486)	Returns a new System::WideString (see page 3467) without trailing spaces or control characters.
	UpperCase (see page 3486)	Returns an upper-case version of this System::WideString (see page 3467).
	vprintf (see page 3487)	Sets the value of the System::WideString (see page 3467) given a format string and its arguments.

35.1.25.5.1 WideString::c_bstr Method

Returns the underlying BSTR that implements theSystem::WideString::WideString (see page 3467).

Pascal

```
function c_bstr: WideChar*;
```

C++

```
__fastcall WideChar* c_bstr() const;
```

Description

Use c_System::WideString::c_bstr to access the BSTR that implements theSystem::WideString::WideString (see page 3467).

Note: BSTR values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Window.

See Also

System::WideString::c_bstr

System::WideString::Copy (see page 3475)

System::WideString::&

35.1.25.5.2 WideString::cat_printf Method

Appends to the value of the System::WideString (see page 3467) given a format string and its arguments.

Pascal

```
function cat_printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int cat_printf(const wchar_t* format, ...);
```

Description

Use System::WideString::cat_printf to modify the value of the System::WideString (see page 3467) by appending the result of using a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the number of bytes appended to the System::WideString (see page 3467).

See Also

System::WideString::Format (see page 3477)

System::WideString::cat_sprintf (see page 3474)

System::WideString::cat_vprintf (see page 3474)

35.1.25.5.3 WideString::cat_sprintf Method

Appends the result of a given format string and its arguments to the end of the System::WideString (see page 3467).

Pascal

```
function cat_sprintf(const format: wchar_t*; : ...): WideString&; cdecl;
```

C++

```
__cdecl WideString& cat_sprintf(const wchar_t* format, ...);
```

Description

Use cat_System::WideString::cat_sprintf to append to the value of the System::WideString (see page 3467) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

For example, the following code

```
WideString Salutation = "Mr."
WideString s = "";
s.cat_sprintf("Dear %s ", Salutation);
s.cat_sprintf("%s,", "Smith");
```

results in an System::WideString (see page 3467) with the value

```
Dear Mr. Smith,
```

This method returns a reference to the System::WideString (see page 3467) (*this) with the resulting value.

See Also

System::WideString::Format (see page 3477)

System::WideString::cat_vprintf (see page 3474)

35.1.25.5.4 WideString::cat_vprintf Method

Appends the result of a given format string and its arguments to the end of the System::WideString (see page 3467).

Pascal

```
function cat_vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int cat_vprintf(const wchar_t* format, va_list);
```

Description

Use cat_System::WideString::cat_vprintf to append to the value of the System::WideString (see page 3467) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the number of bytes appended to the System::WideString (see page 3467).

See Also

System::WideString::Format (see page 3477)

System::WideString::cat_printf (see page 3473)

System::WideString::cat_sprintf (see page 3474)

35.1.25.5.5 WideString::Copy Method

Returns a copy of theSystem::WideString::WideString (see page 3467).

Pascal

```
function Copy(var : wchar_t; len: Integer): wchar_t*;
```

C++

```
__fastcall wchar_t* Copy(wchar_t *, int len) const;
```

Description

Use System::WideString::Copy to copy theSystem::WideString::WideString (see page 3467) into an array of wchar_t. Use the Length (see page 3481) method to ensure that the destination buffer is large enough to hold the contents of theSystem::WideString::WideString (see page 3467).

WideString (see page 3467) also supports a static copy method that copies an arbitrary source string.

See Also

System::WideString::bstr

System::WideString::c_bstr (see page 3473)

System::WideString::&

System::WideString::Length (see page 3481)

35.1.25.5.6 WideString::CurrToStr Method

Converts a currency value to its string representation.

Pascal

```
function CurrToStr(value: Currency): WideString; static;
```

C++

```
static __fastcall WideString CurrToStr(Currency value);
```

Description

System::WideString::CurrToStr converts the currency value given by value to its string representation. The conversion using the sffGeneral floating-point conversion rules. For further details, see System::WideString::CurrToStrF (see page 3475).

See Also

System::WideString::CurrToStrF (see page 3475)

35.1.25.5.7 WideString::CurrToStrF Method

Converts a currency value to its string representation using the indicated float format convention.

Pascal

```
function CurrToStrF(value: Currency; format: TStringFloatFormat; digits: Integer): WideString;
static;
```

C++

```
static __fastcall WideString CurrToStrF(Currency value, TStringFloatFormat format, int digits);
```

Description

System::WideString::CurrToStrF converts the currency value given by value to its string representation using the indicated float format convention. A call to System::WideString::CurrToStrF corresponds to a call to FloatToStrF (see page 3477) with an implied precision of 19 digits.

35.1.25.5.8 WideString::data Method

Returns a pointer to the underlying string System::WideString::data.

Pascal

```
function data: WideChar*;
```

C++

```
__fastcall WideChar* data();
```

Description

System::WideString::data returns a pointer to a null-terminated character array that the System::WideString (see page 3467) uses to store its value. This is the same value as the one returned by the System::WideString::c_str method, except that System::WideString::data returns a NULL pointer when the value is an empty string.

Note: Because System::WideString::data returns NULL instead of an empty string, it is useful for C++ functions that expect either a valid (non-empty) char* or NULL. The following example assumes that TCHAR mapping is not set to wchar_t:

```
WideString x;
    ShellExecute(0, static_cast<const char*>(x.data()), "File.txt", NULL, NULL, SW_SHOW);
```

If TCHAR mapping is set to wchar_t, ShellExecute becomes the wide version, ShellExecuteW. If you want to use the Ansi version of ShellExecute when TCHAR mapping is set to wchar_t, you should specify ShellExecuteA.

See Also

System::WideString::c_str

TCHAR Mapping

35.1.25.5.9 WideString::Delete Method

System::WideString::Deletes a substring from the System::WideString::WideString (see page 3467).

Pascal

```
procedure Delete(index: Integer; count: Integer);
```

C++

```
__fastcall void Delete(int index, int count);
```

Description

System::WideString::Delete removes count characters from theSystem::WideString::WideString (see page 3467), beginning with the character at position index.

See Also

System::WideString::Insert (see page 3479)

System::WideString::[] (see page 3488)

35.1.25.5.10 WideString::FloatToStrF Method

Returns the string representation of a floating-point value.

Pascal

```
function FloatToStrF(value: long double; format: TStringFloatFormat; precision: Integer;
digits: Integer): WideString; static;
```

C++

```
static __fastcall WideString FloatToStrF(long double value, TStringFloatFormat format, int
precision, int digits);
```

Description

The format parameter controls the format of the resulting string. The precision parameter specifies the precision of the given value. It should be 7 or less for values of type float, 15 or less for values of type double, and 18 or less for values of type Extended. The meaning of the digits parameter depends on the particular format selected.

The possible values of the format parameter, and the meaning of each, are described below.

Value	Meaning
sffGeneral	General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
sffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format.
sffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The sffNumber format corresponds to the sffFixed format, except that the resulting string contains thousand separators.
sffCurrency	Currency (see page 3235) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, and DecimalSeparator global variables, all of which are initialized from the Currency (see page 3235) Format (see page 3477) in the International section of the Windows Control Panels. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18.

For all formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables.

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

35.1.25.5.11 WideString::Format Method

System::WideString::Formats a string assembled from an System::WideString (see page 3467), format, and a series of array arguments, args.

Pascal

```
function Format(const format: WideString&; const args: TVarRec *; size: Integer): WideString;
static;
```

C++

```
static __fastcall WideString Format(const WideString& format, const TVarRec * args, int size);
```

Description

Use System::WideString::Format to obtain an System::WideString (see page 3467) whose value is the result of applying an array of arguments (args) to a given a format string (format). The last parameter (size) is the index of the last element in args (one less than the number of arguments).

Note that System::WideString::Format is a static method. The returned value is a newly allocated System::WideString (see page 3467), even if the method is called from an existing System::WideString (see page 3467) instance.

For information on how System::WideString::Format assembles a final string from its format string and arguments, see System::WideString::Format Strings.

See Also

System::WideString::printf (see page 3482)

System::WideString::sprintf (see page 3483)

35.1.25.5.12 WideString::FormatFloat Method

Formats the floating-point value given by value using the format string given by format.

Pascal

```
function FormatFloat(const format: WideString&; const value: long double&): WideString; static;
```

C++

```
static __fastcall WideString FormatFloat(const WideString& format, const long double& value);
```

Description

The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first dot character ('.') in the format string determines the location of the decimal separator in the formatted value; any additional dot characters are ignored. The actual character used as the decimal separator in the output string is determined by the DecimalSeparator global variable, whose default value is specified in the Number Format (see page 3477) of the International section in the Windows Control Panel.
,	Thousand separator. If the format string contains one or more comma characters (','), the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of comma characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as the thousand separator in the output is determined by the ThousandSeparator global variable, whose default value is specified in the Number Format (see page 3477) of the International section in the Windows Control Panel.

E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain from one to three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If no formatting is specified for positive values, positive values are formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 3477) with the sffGeneral format. General floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The following table shows some sample formats and the results produced when the formats are applied to different values:

Format (see page 3477) string-1234-12340.50

Format (see page 3477) string-	**1234**	**-1234**	**0.5**	**0**
1234	-1234	0.5	0	
0	1234	-1234	1	
0.00	1234.00	-1234.00	0.50	
#	1234	-1234	.5	
#,##0.00	1,234.00	-1,234.00	0.50	
#,##0.00;(#,##0.00)	1,234.00	(1,234.00)	0.50	
#,##0.00;;Zero	1,234.00	-1,234.00	0.50	
0.000E+00	1.234E+03	-1.234E+03	5.000E-01	
#.###E-0	1.234E3	-1.234E3	5E-1	

35.1.25.5.13 WideString::Insert Method

System::WideString::Inserts a specified string into theSystem::WideString::WideString (see page 3467).

Pascal

```
procedure Insert(const str: WideString&; index: Integer);
```

C++

```
__fastcall void Insert(const WideString& str, int index);
```

Description

Call System::WideString::Insert to inserts the string str into thisSystem::WideString::WideString (see page 3467) beginning at the position index.

See Also

System::WideString::Delete (see page 3476)

System::WideString::[] (see page 3488)

35.1.25.5.14 WideString::IntToHex Method

Converts a number into a string containing the number's hexadecimal (base 16) representation.

Pascal

```
function IntToHex(value: Integer; digits: Integer): WideString; static;
```

C++

```
static __fastcall WideString IntToHex(int value, int digits);
```

Description

Value is the number to convert. Digits indicates the minimum number of hexadecimal digits.

35.1.25.5.15 WideString::IsDelimiter Method

Returns true if the specified character matches any character in the delimiters string.

Pascal

```
function IsDelimiter(const delimiters: WideString&; index: Integer): bool;
```

C++

```
__fastcall bool IsDelimiter(const WideString& delimiters, int index) const;
```

Description

System::WideString::IsDelimiter returns true if the byte indicated by index (counting from 1) is a single-byte character contained in delimiters. If the byte is part of a multi-byte character or not in delimiters, System::WideString::IsDelimiter returns false.

35.1.25.5.16 WideString::IsEmpty Method

Returns true if the System::WideString::WideString (see page 3467) is empty.

Pascal

```
function IsEmpty: bool;
```

C++

```
__fastcall bool IsEmpty() const;
```

Description

Call System::WideString::IsEmpty to determine whether the System::WideString::WideString (see page 3467) represents an

empty string.

35.1.25.5.17 WideString::IsPathDelimiter Method

Returns true if the specified character is '\'.

Pascal

```
function IsPathDelimiter(index: Integer): bool;
```

C++

```
__fastcall bool IsPathDelimiter(int index) const;
```

Description

System::WideString::IsPathDelimiter returns true if the byte indicated by index (counting from 1) is a single-byte character containing the path delimiter ('\' on Windows, '/' on Linux).

35.1.25.5.18 WideString::LastChar Method

Returns a pointer to the last character in a string.

Pascal

```
function LastChar: wchar_t*;
```

C++

```
__fastcall wchar_t* LastChar() const;
```

Description

Call System::WideString::LastChar to obtain a pointer to the last character in a string. This function supports multi-byte character sets (MBCS).

35.1.25.5.19 WideString::LastDelimiter Method

Returns the index of the rightmost character that matches any character in the delimiters string.

Pascal

```
function LastDelimiter(const delimiters: WideString&): Integer;
```

C++

```
__fastcall int LastDelimiter(const WideString& delimiters) const;
```

Description

System::WideString::LastDelimiter returns the byte index in this System::WideString (see page 3467) of the rightmost whole character that matches any character in delimiters except NULL The System::WideString (see page 3467) may contain multibyte characters; delimiters must contain only single byte non-NULL characters.

35.1.25.5.20 WideString::Length Method

Returns the length of theSystem::WideString::WideString (see page 3467).

Pascal

```
function Length: Integer;
```

C++

```
__fastcall int Length() const;
```

Description

System::WideString::Length returns the number of characters in thisSystem::WideString::WideString (see page 3467).

35.1.25.5.21 WideString::LowerCase Method

Returns a lower-case version of System::WideString (see page 3467).

Pascal

```
function LowerCase: WideString;
```

C++

```
__fastcall WideString LowerCase() const;
```

Description

System::WideString::LowerCase returns a new System::WideString (see page 3467) that contains all characters in this string, converted to lower case. The conversion uses the current locale . The original System::WideString (see page 3467) remains unchanged.

35.1.25.5.22 WideString::Pos Method

Returns the index at which a specified substring begins.

Pascal

```
function Pos(const subStr: WideString&): Integer;
```

C++

```
__fastcall int Pos(const WideString& subStr) const;
```

Description

Use System::WideString::Pos to locate a substring of theSystem::WideString::WideString (see page 3467). System::WideString::Pos returns the index of the first character in subStr. If subStr is not contained in theSystem::WideString::WideString (see page 3467), System::WideString::Pos returns 0.

See Also

System::WideString::Length (see page 3481)

System::WideString::SubString (see page 3484)

35.1.25.5.23 WideString::printf Method

Sets the value of the System::WideString (see page 3467) given a format string and its arguments.

Pascal

```
function printf(const format: wchar_t*; : ...): Integer; cdecl;
```

C++

```
__cdecl int printf(const wchar_t* format, ...);
```

Description

Use System::WideString::printf to set the value of the System::WideString (see page 3467) given a standard C++ format

specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter. This method returns the length of the final formatted string.

See Also

System::WideString::Format (see page 3477)

System::WideString::sprintf (see page 3483)

System::WideString::cat_printf (see page 3473)

System::WideString::vprintf (see page 3487)

35.1.25.5.24 WideString::SetLength Method

Changes the length of theSystem::WideString::WideString (see page 3467).

Pascal

```
procedure SetLength(newLength: Integer);
```

C++

```
__fastcall void SetLength(int newLength);
```

Description

System::WideString::SetLength changes theSystem::WideString::WideString (see page 3467)'s length to newLength, truncating if necessary.

System::WideString::SetLength reallocates the underlying array of characters that stores the value of theSystem::WideString::WideString (see page 3467). Existing characters in the string are preserved, but the content of newly allocated space is undefined. If there is not enough memory available to reallocate theSystem::WideString::WideString (see page 3467), an EOutOfMemory exception is thrown. Following a call to System::WideString::SetLength, theSystem::WideString::WideString (see page 3467) is guaranteed to reference a unique string — that is a string with a reference count of one.

See Also

System::WideString::Length (see page 3481)

35.1.25.5.25 WideString::sprintf Method

Sets the value of the System::WideString (see page 3467) given a format string and its arguments.

Pascal

```
function sprintf(const format: wchar_t*; : ...): WideString&; cdecl;
```

C++

```
__cdecl WideString& sprintf(const wchar_t* format, ...);
```

Description

Use System::WideString::sprintf to set to the value of the System::WideString (see page 3467) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as additional parameters following the format parameter.

This method returns a reference to the System::WideString (see page 3467) (*this) with the resulting value.

See Also

System::WideString::Format (see page 3477)

System::WideString::cat_sprintf (see page 3474)

System::WideString::printf (see page 3482)

System::WideString::vprintf (see page 3487)

35.1.25.5.26 WideString::StringOfChar Method

Returns a string with the specified number of characters.

Pascal

```
function StringOfChar(ch: WideChar; count: Integer): WideString; static;
```

C++

```
static __fastcall WideString StringOfChar(WideChar ch, int count);
```

Description

System::WideString::StringOfChar returns a string containing count characters with the character value given by ch. For example,

System::WideString (see page 3467) s = System::WideString::System::WideString::StringOfChar('A', 10);

sets s to the string "AAAAAAAAAA".

35.1.25.5.27 WideString::SubString Method

Returns a specified substring of theSystem::WideString::WideString (see page 3467).

Pascal

```
function SubString(index: Integer; count: Integer): WideString;
```

C++

```
__fastcall WideString SubString(int index, int count) const;
```

Description

System::WideString::SubString returns a newSystem::WideString::WideString (see page 3467) that is a substring of thisSystem::WideString::WideString (see page 3467). The substring contains count characters beginning at index.

See Also

System::WideString::Length (see page 3481)

System::WideString::Pos (see page 3482)

35.1.25.5.28 WideString::swap Method

Swap string pointers.

Pascal

```
function swap(var other: WideString): WideString&;
```

C++

```
WideString& swap(WideString& other);
```

Description

System::WideString::swap swaps two System::WideString (see page 3467) strings. Pointers to the data (see page 3476) are swapped, and the data (see page 3476) is not copied. This makes the swap operation efficient for sorting.

35.1.25.5.29 WideString::ToDouble Method

Converts the string to a floating-point value.

Pascal

```
function ToDouble: double;
```

C++

```
__fastcall double ToDouble() const;
```

Description

System::WideString::ToDouble converts the string to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks in the string are ignored. The DecimalSeparator global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, an EConvertError exception is thrown.

35.1.25.5.30 WideString::ToInt Method

Converts a string to an integer, throwing an exception if the string is invalid.

Pascal

```
function ToInt: Integer;
```

C++

```
__fastcall int ToInt() const;
```

Description

System::WideString::ToInt converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, an EConvertError exception is thrown.

See Also

System::WideString::ToIntDef (see page 3485)

35.1.25.5.31 WideString::ToIntDef Method

Converts the string to an integer, returning a default value if the string is invalid.

Pascal

```
function ToIntDef(defaultValue: Integer): Integer;
```

C++

```
__fastcall int ToIntDef(int defaultValue) const;
```

Description

System::WideString::ToIntDef converts the given string to an integer value and returns the integer. If the string doesn't contain a valid value, the value given by defaultValue is returned.

See Also

System::WideString::ToInt (see page 3485)

35.1.25.5.32 WideString::Trim Method

Returns a new System::WideString (see page 3467) without leading spaces, trailing spaces, or control characters.

Pascal

```
function Trim: WideString;
```

C++

```
__fastcall WideString Trim() const;
```

Description

System::WideString::Trim returns a new System::WideString (see page 3467) removing leading and trailing spaces and control characters. Use System::WideString::Trim to remove blank space before and after the first printing character.

35.1.25.5.33 WideString::TrimLeft Method

Returns a new System::WideString (see page 3467) without leading spaces or control characters.

Pascal

```
function TrimLeft: WideString;
```

C++

```
__fastcall WideString TrimLeft() const;
```

Description

System::WideString::TrimLeft returns a new System::WideString (see page 3467) removing leading spaces and control characters. Use System::WideString::TrimLeft to remove blank space before the first printing character.

35.1.25.5.34 WideString::TrimRight Method

Returns a new System::WideString (see page 3467) without trailing spaces or control characters.

Pascal

```
function TrimRight: WideString;
```

C++

```
__fastcall WideString TrimRight() const;
```

Description

System::WideString::TrimRight returns a new System::WideString (see page 3467) removing trailing spaces and control characters. Use the System::WideString::TrimRight function to remove blank space after the last printing character.

35.1.25.5.35 WideString::UpperCase Method

Returns an upper-case version of this System::WideString (see page 3467).

Pascal

```
function UpperCase: WideString;
```

C++

```
__fastcall WideString UpperCase() const;
```

Description

System::WideString::UpperCase returns a new System::WideString (see page 3467) that contains all characters in this string, converted to upper case. The conversion uses the current locale . The original System::WideString (see page 3467) remains unchanged.

35.1.25.5.36 WideString::vprintf Method

Sets the value of the System::WideString (see page 3467) given a format string and its arguments.

Pascal

```
function vprintf(const format: wchar_t*; : va_list): Integer; cdecl;
```

C++

```
__cdecl int vprintf(const wchar_t* format, va_list);
```

Description

Use System::WideString::vprintf to set to the value of the System::WideString (see page 3467) given a standard C++ format specifier. Pass the values to any arguments in the format specifier as a variable argument list that was created using va_start.

This method returns the length of the final formatted string.

See Also

System::WideString::Format (see page 3477)

System::WideString::sprintf (see page 3483)

System::WideString::printf (see page 3482)

System::WideString::cat_vprintf (see page 3474)

35.1.25.6 WideString Operators

The operators of the WideString class are listed here.

Public Operators

	Name	Description
	!= (see page 3487)	This is the overview for the != operator overload.
	[] (see page 3488)	This is the overview for the [] operator overload.
	+ (see page 3489)	Concatenates twoSystem::WideString::WideStrings (see page 6807).
	+= (see page 3489)	Concatenates this System::WideString::WideString (see page 3467) and rhs, and assigns the result to this System::WideString::WideString (see page 3467).
	< (see page 3489)	This is the overview for the < operator overload.
	<= (see page 3490)	This is the overview for the <= operator overload.
	= (see page 3490)	Assigns the value of theSystem::WideString::WideString (see page 3467) to equal a source string.
	== (see page 3491)	This is the overview for the == operator overload.
	> (see page 3491)	This is the overview for the > operator overload.
	>= (see page 3492)	This is the overview for the >= operator overload.

35.1.25.6.1 != Operator

This is the overview for the != operator overload.

Overload List

	Name	Description
	WideString::!= (char*) (see page 3488)	Tests whether theSystem::WideString::WideString (see page 3467) is different from anotherSystem::WideString::WideString (see page 3467).
	WideString::!= (WideString&) (see page 3488)	Tests whether theSystem::WideString::WideString (see page 3467) is different from anotherSystem::WideString::WideString (see page 3467).

35.1.25.6.1.1 WideString::!= Operator (WideString&)

Tests whether theSystem::WideString::WideString (see page 3467) is different from anotherSystem::WideString::WideString (see page 3467).

Pascal

```
operator !=(const rhs: WideString&);
operator !=(const s: char*);
```

C++

```
__fastcall bool operator !=(const WideString& rhs) const;
__fastcall bool operator !=(const char* s) const;
```

Description

The != System::WideString (see page 3467)::!= returns true if the strings are not the same, false otherwise.

See Also

System::WideString::== (see page 3491)

35.1.25.6.2 [] Operator

This is the overview for the [] operator overload.

Overload List

	Name	Description
	WideString::[] (int) (see page 3488)	Returns the character at a specified index in the string.
	WideString::[] (int) (see page 3488)	Returns the character at a specified index in the string.

35.1.25.6.2.1 WideString::[] Operator (int)

Returns the character at a specified index in the string.

Pascal

```
operator [](const idx: Integer);
operator [](const idx: Integer);
```

C++

```
__fastcall WideChar & operator [](const int idx);
__fastcall const WideChar& operator [](const int idx) const;
```

Description

The System::WideString (see page 3467)::[] returns the character in the string at index value idx. The System::WideString (

see page 3467)::[] assumes a base index of 1. For example, given the string “Hello” and an index value of 2, the character returned would be ‘e’.

35.1.25.6.3 WideString::+ Operator

Concatenates twoSystem::WideString::WideStrings (see page 6807).

Pascal

```
operator +(const rhs: WideString&);
```

C++

```
__fastcall WideString operator +(const WideString& rhs) const;
```

Description

The + System::WideString (see page 3467)::+ returns aSystem::WideString::WideString (see page 3467) that has the value passed as rhs concatenated to the end. A related friend System::WideString (see page 3467)::+ takes two arguments and returns the concatenation of lhs with rhs.

35.1.25.6.4 WideString::+= Operator

Concatenates this System::WideString::WideString (see page 3467) and rhs, and assigns the result to this System::WideString::WideString (see page 3467).

Pascal

```
operator +=(const rhs: WideString&);
```

C++

```
__fastcall WideString& operator +=(const WideString& rhs);
```

Description

Use += to concatenate rhs to the end of the System::WideString::WideString (see page 3467). This System::WideString (see page 3467)::+= is not supported when the System::WideString::WideString (see page 3467) is a property value.

35.1.25.6.5 < Operator

This is the overview for the < operator overload.

Overload List

	Name	Description
	WideString::< (char*) (see page 3489)	Compares two strings.
	WideString::< (WideString&) (see page 3489)	Compares two strings.

35.1.25.6.5.1 WideString::< Operator (WideString&)

Compares two strings.

Pascal

```
operator <(const rhs: WideString&);
operator <(const s: char*);
```

C++

```
__fastcall bool operator <(const WideString& rhs) const;
```

```
__fastcall bool operator <(const char* s) const;
```

Description

The < System::WideString (see page 3467)::< returns true if thisSystem::WideString::WideString (see page 3467) is less than rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::<= (see page 3490)

System::WideString::> (see page 3491)

35.1.25.6.6 <= Operator

This is the overview for the <= operator overload.

Overload List

	Name	Description
	WideString::<= (char*) (see page 3490)	Compares two strings.
	WideString::<= (WideString&) (see page 3490)	Compares two strings.

35.1.25.6.6.1 WideString::<= Operator (WideString&)

Compares two strings.

Pascal

```
operator <=(const rhs: WideString&);
operator <=(const s: char*);
```

C++

```
__fastcall bool operator <=(const WideString& rhs) const;
__fastcall bool operator <=(const char* s) const;
```

Description

The <= System::WideString (see page 3467)::<= returns true if thisSystem::WideString::WideString (see page 3467) is less than or equal to rhs, false otherwise. The strings are compared using their numerical value of each character, with case sensitivity.

See Also

System::WideString::>= (see page 3492)

System::WideString::< (see page 3489)

35.1.25.6.7 WideString::= Operator

Assigns the value of theSystem::WideString::WideString (see page 3467) to equal a source string.

Pascal

```
operator =(const rhs: WideString&);
```

C++

```
__fastcall WideString& operator =(const WideString& rhs);
```

Description

Use the assignment System::WideString (see page 3467)::= to set the value of theSystem::WideString::WideString (see

page 3467) to match the string given by rhs.

Note: BSTR values are used with COM objects. Do not use COM objects in platform-independent applications, since COM is only available under Windows.

35.1.25.6.8 == Operator

This is the overview for the == operator overload.

Overload List

	Name	Description
	WideString::== (char*) (see page 3491)	Compares theSystem::WideString::WideString (see page 3467) with a specified string, returning true if they are the same.
	WideString::== (WideString&) (see page 3491)	Compares theSystem::WideString::WideString (see page 3467) with a specified string, returning true if they are the same.

35.1.25.6.8.1 WideString::== Operator (WideString&)

Compares theSystem::WideString::WideString (see page 3467) with a specified string, returning true if they are the same.

Pascal

```
operator ==(const rhs: WideString&);
operator ==(const s: char*);
```

C++

```
__fastcall bool operator ==(const WideString& rhs) const;
__fastcall bool operator ==(const char* s) const;
```

Description

The equality System::WideString (see page 3467)::== returns true if thisSystem::WideString::WideString (see page 3467) is the same as rhs, false otherwise.

See Also

System::WideString::!= (see page 3488)

35.1.25.6.9 > Operator

This is the overview for the > operator overload.

Overload List

	Name	Description
	WideString::> (char*) (see page 3491)	Compares two strings.
	WideString::> (WideString&) (see page 3491)	Compares two strings.

35.1.25.6.9.1 WideString::> Operator (WideString&)

Compares two strings.

Pascal

```
operator >(const rhs: WideString&);
operator >(const s: char*);
```

C++

```
__fastcall bool operator >(const WideString& rhs) const;
__fastcall bool operator >(const char* s) const;
```

Description

The System::WideString (see page 3467)::> returns true if thisSystem::WideString::WideString (see page 3467) is greater than rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::>= (see page 3492)

System::WideString::< (see page 3489)

35.1.25.6.10 >= Operator

This is the overview for the >= operator overload.

Overload List

	Name	Description
	WideString::>= (char*) (see page 3492)	Compares two strings.
	WideString::>= (WideString&) (see page 3492)	Compares two strings.

35.1.25.6.10.1 WideString::>= Operator (WideString&)

Compares two strings.

Pascal

```
operator >=(const rhs: WideString&);
operator >=(const s: char*);
```

C++

```
__fastcall bool operator >=(const WideString& rhs) const;
__fastcall bool operator >=(const char* s) const;
```

Description

The System::WideString (see page 3467)::>= returns true if thisSystem::WideString::WideString (see page 3467) is greater than or equal to rhs, false otherwise. The strings are compared using the numerical value of each character, with case sensitivity.

See Also

System::WideString::<= (see page 3490)

System::WideString::> (see page 3491)

35.2 Interfaces

The following table lists interfaces in this documentation.

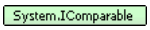
Interfaces

	Name	Description
	IComparable (see page 3493)	IComparable is the generic interface used to compare two objects.
	IDispatch (see page 3494)	IDispatch is the interface used to support Automation.
	IEnumerable (see page 3497)	IEnumerable is the generic interface for enumerable containers.
	IEnumerator (see page 3498)	IEnumerator is the generic interface for container enumerators.
	IEquatable (see page 3500)	IEquatable is the generic interface used to check the equality of two objects.
	IInterface (see page 3502)	IInterface is the base class for all interfaces defined in Delphi code.
	IInvokable (see page 3504)	IInvokable is the base class for invokable interfaces.

35.2.1 IComparable Interface

IComparable is the generic interface used to compare two objects.

Class Hierarchy



Pascal

```
IComparable<T> = interface;
```

C++

```
template <T>
__interface IComparable;
```

File

System

Description

IComparable is implemented by classes that have to support a comparison between instances. IComparable exposes a single method called CompareTo (see page 3494). CompareTo (see page 3494) can be used later to compare two instances of the same class.

See Also

CompareTo (see page 3494)

IEquatable (see page 3500)

35.2.1.1 IComparable Members

The following tables list the members exposed by IComparable.

Public Methods

	Name	Description
	CompareTo (see page 3494)	CompareTo is a generic method used to compare two instances of a class.

35.2.1.2 IComparable Methods

The methods of the IComparable class are listed here.

Public Methods

	Name	Description
	CompareTo (see page 3494)	CompareTo is a generic method used to compare two instances of a class.

35.2.1.2.1 IComparable.CompareTo Method

CompareTo is a generic method used to compare two instances of a class.

Pascal

```
function CompareTo(Value: T): Integer;
```

C++

```
__fastcall int CompareTo(T Value);
```

Description

Use the CompareTo method to compare two instances of the same class. Any class that implements the IComparable (see page 3493) interface should provide an implementation for the CompareTo method.

The return value of Compare must be in the following ranges.

Return value	Description
Result is less than zero (<0)	The object is less than `Value`.
Result is equal to zero(=0)	The object is equal to `Value`.
Result is greater than zero(>0)	The object is greater than `Value`.

See Also

IEquatable (see page 3500)

Equals (see page 3501)

35.2.2 IDispatch Interface

IDispatch is the interface used to support Automation.

Class Hierarchy

Pascal

```
IDispatch = interface(IUnknown);
```

C++

```
__interface IDispatch : public IUnknown;
```

File

System

Description

Implement the IDispatch interface in COM-enabled classes that have to act as Automation controllers. The IDispatch interface exposes a number of methods for Automation clients that require late-binding (such as scripting languages).

See Also

IInterface (see page 3502)

Dispatch Interfaces

IDispatch Interface (MSDN)

35.2.2.1 IDispatch Members

The following tables list the members exposed by IDispatch.

Public Methods

	Name	Description
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.

35.2.2.2 IDispatch Methods

The methods of the IDispatch class are listed here.

Public Methods

	Name	Description
	GetIDsOfNames (see page 3495)	GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.
	GetTypeInfo (see page 3496)	GetTypeInfo returns the object's type information.
	GetTypeInfoCount (see page 3496)	GetTypeInfoCount returns the count of type information interfaces.
	Invoke (see page 3497)	Invoke performs a call to a late-binded property or method.

35.2.2.2.1 IDispatch.GetIDsOfNames Method

GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name.

Pascal

```
function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount: Integer; LocaleID:
Integer; DispIDs: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetIDsOfNames(const TGUID IID, void * Names, int NameCount, int
LocaleID, void * DispIDs);
```

Description

GetIDsOfNames returns a set of dispatch identifiers (**dispid**s) for a given member name. GetIDsOfNames is used primarily to obtain dispatch identifiers for the late-binded methods exposed by the COM object through the IDispatch (see page 3494) interface.

See Also

GetTypeInfoCount (see page 3496)

GetTypeInfo (see page 3496)

Invoke (see page 3497)

35.2.2.2.2 IDispatch.GetTypeInfo Method

GetTypeInfo returns the object's type information.

Pascal

```
function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfo(int Index, int LocaleID,  TypeInfo);
```

Description

GetTypeInfo returns the type information for a specified index. The index is previously obtained by a call to the GetTypeInfoCount (see page 3496) method. If the method returns S_OK (see page 3860), the `TypeInfo` parameter is set to point to the requested type information; otherwise an error code is returned.

See Also

GetTypeInfoCount (see page 3496)

GetIDsOfNames (see page 3495)

Invoke (see page 3497)

35.2.2.2.3 IDispatch.GetTypeInfoCount Method

GetTypeInfoCount returns the count of type information interfaces.

Pascal

```
function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT GetTypeInfoCount(int Count);
```

Description

GetTypeInfoCount returns the count of type information interfaces supported by the implementing COM object. If the method returns S_OK (see page 3860), the `Count` parameter is set to the number of type information interfaces, which is either 1 (if the object supports type information) or 0 (if the object does not support type information). A return value of E_NOTIMPL (see page 3846) means that the method is not implemented in the COM object.

See Also

GetTypeInfo (see page 3496)

GetIDsOfNames (see page 3495)

Invoke (see page 3497)

35.2.2.2.4 IDispatch.Invoke Method

Invoke performs a call to a late-binded property or method.

Pascal

```
function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params;
VarResult: Pointer; ExcepInfo: Pointer; ArgErr: Pointer): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT Invoke(int DispID, const TGUID IID, int LocaleID, Word Flags,
Params, void * VarResult, void * ExcepInfo, void * ArgErr);
```

Description

Invoke actually calls a method or a property exposed by the COM object through IDispatch (see page 3494). Invoke requires a previously obtained dispatch identifier. The identifier uniquely identifies a method, a property, and a list of parameters to be passed to the called method and exception information. The dispatch identifier of a method or property (also known as **dispid**) can be obtained by a calling the GetIDsOfNames (see page 3495) method.

See Also

GetTypeInfoCount (see page 3496)

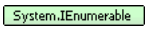
GetTypeInfo (see page 3496)

GetIDsOfNames (see page 3495)

35.2.3 IEnumerable Interface

IEnumerable is the generic interface for enumerable containers.

Class Hierarchy



Pascal

```
IEnumerable<T> = interface(IEnumerable);
```

C++

```
template <T>
__interface IEnumerable : public IEnumerable;
```

File

System

Description

To make your container enumerable, implement the IEnumerable interface. IEnumerable exposes a single method called GetEnumerator (see page 3498), that must return a reference to an IEnumerator (see page 3498) interface.

Classes that implement the IEnumerable interface are considered enumerable in Delphi language and can be used in **for..in** statements.

Note: It is not required that IEnumerable be actually implemented by a class in order to be considered enumerable. It is enough to export a public method called GetEnumerator (see page 3498) that returns an enumerator object. Still, we recommend that

you implement the IEnumerable interface.

See Also

IEnumerator (see page 3498)

GetEnumerator (see page 3498)

35.2.3.1 IEnumerable Members

The following tables list the members exposed by IEnumerable.

Public Methods

	Name	Description
	GetEnumerator (see page 3498)	Returns an enumerator used to iterate over a collection.

35.2.3.2 IEnumerable Methods

The methods of the IEnumerable class are listed here.

Public Methods

	Name	Description
	GetEnumerator (see page 3498)	Returns an enumerator used to iterate over a collection.

35.2.3.2.1 IEnumerable.GetEnumerator Method

Returns an enumerator used to iterate over a collection.

Pascal

```
function GetEnumerator: IEnumerator<T>;
```

C++

```
__fastcall IEnumerator<T> GetEnumerator();
```

Description

Call GetEnumerator to obtain an enumerator to iterate over a collection. Delphi language supports the **for..in** statement, which automatically uses GetEnumerator to obtain a reference to the enumerator object.

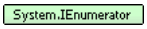
See Also

IEnumerator (see page 3498)

35.2.4 IEnumerator Interface

IEnumerator is the generic interface for container enumerators.

Class Hierarchy



Pascal

```
IEnumerator<T> = interface(IEnumerator);
```

C++

```
template <T>
__interface IEnumerator : public IEnumerator;
```

File

System

Description

Implement the IEnumerator interface in the class that has to act as an enumerator for a container. An IEnumerator is usually obtained by a call to the container's GetEnumerator method.

Classes that implement the IEnumerator interface are considered enumerators in Delphi language and are used automatically by the **for..in** statement.

Note: It is not required that IEnumerator be actually implemented by a class in order to be considered an enumerator. It is enough to simply implement the required methods in your class. Still, we recommend that you implement the IEnumerator interface.

See Also

IEnumerable (see page 3497)

35.2.4.1 IEnumerator Members

The following tables list the members exposed by IEnumerator.

Public Methods

	Name	Description
	GetCurrent (see page 3499)	Returns the currently focused element in the container.

Public Properties

	Name	Description
	Current (see page 3500)	Returns the currently focused element in the container.

35.2.4.2 IEnumerator Methods

The methods of the IEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 3499)	Returns the currently focused element in the container.

35.2.4.2.1 IEnumerator.GetCurrent Method

Returns the currently focused element in the container.

Pascal

```
function GetCurrent: T;
```

C++

```
__fastcall T GetCurrent();
```

Description

GetCurrent is a method that returns the currently focused element in the container.

See Also

Current (see page 3500)

35.2.4.3 IEnumerator Properties

The properties of the IEnumerator class are listed here.

Public Properties

	Name	Description
	Current (see page 3500)	Returns the currently focused element in the container.

35.2.4.3.1 IEnumerator.Current Property

Returns the currently focused element in the container.

Pascal

```
property Current: T;
```

C++

```
__property T Current;
```

Description

Current is a read-only property that gives the currently focused element in the container.

See Also

GetCurrent (see page 3499)

35.2.5 IEquatable Interface

IEquatable is the generic interface used to check the equality of two objects.

Class Hierarchy

System.IEquatable

Pascal

```
IEquatable<T> = interface;
```

C++

```
template <T>
__interface IEquatable;
```

File

System

Description

IEquatable is implemented by classes that have to support equality checking. IEquatable exposes a single method, called Equals (see page 3501). Equals (see page 3501) can be used later to check the equality of two instances of the same class.

See Also

Equals (see page 3501)

IComparable (see page 3493)

35.2.5.1 IEquatable Members

The following tables list the members exposed by IEquatable.

Public Methods

	Name	Description
	Equals (see page 3501)	Equals is a generic method used to check the equality of objects.

35.2.5.2 IEquatable Methods

The methods of the IEquatable class are listed here.

Public Methods

	Name	Description
	Equals (see page 3501)	Equals is a generic method used to check the equality of objects.

35.2.5.2.1 IEquatable.Equals Method

Equals is a generic method used to check the equality of objects.

Pascal

```
function Equals(Value: T): Boolean;
```

C++

```
__fastcall Boolean Equals(T Value);
```

Description

Use the Equals to check the equality of two objects of the same class. Any class that implements the IEquatable (see page 3500) interface is expected to provide the implementation for the Equals method.

The return value of Equals must be one of the following.

Return value	Description
False	The object is different from `Value`.
True	The object is equal to `Value`.

See Also

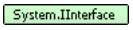
IComparable (see page 3493)

CompareTo (see page 3494)

35.2.6 IInterface Interface

IInterface is the base class for all interfaces defined in Delphi code.

Class Hierarchy



Pascal

```
IInterface = interface;
```

C++

```
__interface IInterface;
```

File

System

Description

IInterface is the base interface for interfaces defined in Delphi code. When used in C++ code, IInterface is considered a child interface of IUnknown (see page 3756) and IUnknown (see page 3756) is considered the base interface for all interfaces.

Note: IInterface introduces the QueryInterface (see page 3504) method, which is useful for discovering and using other interfaces implemented by the same object. IInterface also introduces the reference counting methods, _AddRef (see page 3503) and _Release (see page 3503). The Delphi compiler automatically provides calls to these methods when interfaces are used. The System::TInterfacedObject (see page 3342) class is provided as a base class for objects that implement interfaces. Use of System::TInterfacedObject (see page 3342) is not mandatory, but will often be easier than designing an implementer class from scratch.

Note: C++ method declarations that use IInterface refer to the _di_IInterface type instead. This is a DelphiInterface (see page 3256) wrapper around the IInterface interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IInterface > _di_IInterface;
```

See Also

System::TInterfacedObject (see page 3342)

IInvokable (see page 3504)

IUnknown (see page 3756)

System::DelphiInterface (see page 3256)

35.2.6.1 IInterface Members

The following tables list the members exposed by IInterface.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.

	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

35.2.6.2 IInterface Methods

The methods of the IInterface class are listed here.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

35.2.6.2.1 IInterface._AddRef Method

Increments the reference count for this interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

The Delphi compiler automatically generates a call to __AddRef for every reference to this interface. For most implementations, __AddRef increments the reference count for the interface and returns the new reference count. When the caller is through with the interface, the compiler generates a call to the _Release (see page 3503) method, which decrements the reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __AddRef and _Release (see page 3503) so that they do not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_Release (see page 3503)

35.2.6.2.2 IInterface._Release Method

Decrements the reference count for this interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

The Delphi compiler automatically generates a call to __Release whenever a reference to this interface goes out of scope. For most implementations, __Release decrements the reference count for the interface and returns the new reference count. When the reference count drops to zero, the object is automatically freed.

Note: Implementation objects can implement __Release so that it does not perform any reference counting. For such objects, the reference count never drops to zero and the object is not automatically freed. In these cases, it is the application's responsibility to free the object.

See Also

_AddRef (see page 3503)

35.2.6.2.3 IInterface.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the object that implements this interface supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

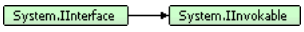
See Also

_AddRef (see page 3503)

35.2.7 IInvokable Interface

IInvokable is the base class for invokable interfaces.

Class Hierarchy



Pascal

```
IInvokable = interface(IInterface);
```

C++

```
__interface IInvokable : public IInterface;
```

File

System

Description

Use IInvokable as a base interface when defining invokable interfaces. IInvokable does not add any properties or methods to the methods it inherits from IInterface (see page 3502). It exists, rather, to ensure that the interface an all its descendants are compiled with runtime type information (RTTI). This RTTI can be used by Web Service applications or the clients of Web Service applications for translating between the code that implements a method and the symbolic encoding of that method call that is sent between client and server applications.

See Also

InvRegistry

TInvokableClassRegistry

TInvokableClass

Understanding Invokable Interfaces

35.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Abs (see page 3511)	Returns an absolute value.
	AcquireExceptionObject (see page 3512)	Retains access to exception object.
	AddModuleUnloadProc (see page 3514)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	AddModuleUnloadProc (see page 3514)	AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.
	Addr (see page 3514)	Returns a pointer to a specified object.
	AllocMem (see page 3515)	AllocMem allocates a memory block and initializes each byte to zero.
	AnsiToUtf8 (see page 3517)	Converts a string encoded in Ansi to UTF-8.
	Append (see page 3520)	Prepares an existing file for adding text to the end.
	ArcTan (see page 3521)	Calculates the arctangent of a given number.
	Assert (see page 3522)	Tests whether a Boolean expression is true.
	Assert (see page 3522)	Tests whether a Boolean expression is true.
	Assigned (see page 3524)	Tests for a nil (unassigned) pointer or procedural variable.
	AssignFile (see page 3525)	Associates the name of an external file with a file variable.
	AttemptToUseSharedMemoryManager (see page 3527)	Attempts to use a shared memory manager.
	BeginThread (see page 3527)	Spawns a separate thread of execution.
	BlockRead (see page 3528)	Reads one or more records from an open file into a variable.
	BlockRead (see page 3528)	Reads one or more records from an open file into a variable.
	BlockWrite (see page 3530)	Writes one or more records from a variable to an open file.
	BlockWrite (see page 3530)	Writes one or more records from a variable to an open file.
	Break (see page 3531)	Causes the flow of control to exit a for, while, or repeat statement.
	ChDir (see page 3534)	Changes the current directory.
	ChDir (see page 3534)	Changes the current directory.
	CheckSafecallResult (see page 3536)	This is function System::CheckSafecallResult.
	Chr (see page 3536)	Returns the character for a specified ASCII value.
	Close (see page 3538)	Terminates the association between a file variable and an external file.
	CloseFile (see page 3539)	Terminates the association between file variable and an external disk file.
	CompToCurrency (see page 3542)	Converts a **Comp** value to a Currency (see page 3235) value.
	CompToDouble (see page 3543)	Converts a Comp value to a double value.
	Concat (see page 3545)	Concatenates two or more strings into one.
	Concat (see page 3545)	Concatenates two or more strings into one.
	Concat (see page 3545)	Concatenates two or more strings into one.
	Continue (see page 3546)	Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.
	Copy (see page 3547)	Returns a substring of a string or a segment of a dynamic array.
	Cos (see page 3549)	Calculates the cosine of an angle.
	CurrencyToComp (see page 3550)	Converts a Currency (see page 3235) value to a Comp value.
	Dec (see page 3550)	Decrements a variable by 1 or N.
	Dec (see page 3550)	Decrements a variable by 1 or N.
	Delete (see page 3552)	Removes a substring from a string.
	Dispose (see page 3553)	Releases memory allocated for a dynamic variable.
	DoubleToComp (see page 3555)	Converts a double value to a Comp.
	DynArrayClear (see page 3555)	This is function System.DynArrayClear.
	DynArrayDim (see page 3555)	This is function System.DynArrayDim.

	DynArraySetLength (see page 3556)	This is function System.DynArraySetLength.
	EndThread (see page 3556)	Terminates the execution of a thread.
	EnumModules (see page 3556)	Executes a callback for all the modules in an application.
	EnumModules (see page 3556)	Executes a callback for all the modules in an application.
	EnumResourceModules (see page 3557)	Executes a callback for all resource modules in an application.
	EnumResourceModules (see page 3557)	Executes a callback for all resource modules in an application.
	Eof (see page 3558)	Tests whether the file position is at the end of a file.
	Eof (see page 3558)	Tests whether the file position is at the end of a file.
	Eof (see page 3558)	Tests whether the file position is at the end of a file.
	Eoln (see page 3559)	Tests whether the file pointer is at the end of a line.
	Eoln (see page 3559)	Tests whether the file pointer is at the end of a line.
	Erase (see page 3560)	Deletes an external file.
	Error (see page 3560)	Error is used to generate a run-time exception.
	ExceptAddr (see page 3562)	Returns the address at which the current exception was raised.
	ExceptObject (see page 3562)	Returns current exception object.
	Exclude (see page 3564)	Removes an element from a Delphi set or removes an event handler in .NET applications.
	Exit (see page 3565)	Exits from the current procedure.
	Exp (see page 3567)	Returns the exponential of X.
	Exp (see page 3567)	Returns the exponential of X.
	FilePos (see page 3567)	Returns the current file position.
	FileSize (see page 3569)	Returns the number of records in a file.
	FillChar (see page 3570)	Fills contiguous bytes with a specified value.
	Finalize (see page 3572)	Uninitializes a dynamically allocated variable.
	Finalize (see page 3572)	Uninitializes a dynamically allocated variable.
	FindClassHInstance (see page 3573)	Returns the instance handle for the module in which a class type is defined.
	FindDynaInst (see page 3573)	This is function System::FindDynaInst.
	FindHInstance (see page 3574)	Returns the instance handle of the module that contains a specified address.
	FindResourceHInstance (see page 3574)	Returns the instance handle of the resource module associated with a specified `HInstance`.
	Flush (see page 3575)	Empties the buffer of a text file opened for output.
	FPower10 (see page 3576)	This is function System.FPower10.
	Frac (see page 3576)	Returns the fractional part of a real number.
	FreeMem (see page 3577)	FreeMem frees a memory block.
	FreeMem (see page 3577)	FreeMem frees a memory block.
	FreeMemory (see page 3579)	FreeMemory frees a memory block.
	Get8087CW (see page 3580)	Returns the value of the 8087 code word.
	GetDir (see page 3580)	Returns the current directory.
	GetHeapStatus (see page 3581)	Returns the current status of the memory manager.
	GetLastError (see page 3582)	Returns the last error reported by an operating system API call.
	GetMem (see page 3584)	GetMem allocates a memory block.
	GetMemory (see page 3585)	GetMemory allocates a memory block.
	GetMemoryManager (see page 3587)	Returns the entry points of the currently installed memory manager.
	GetMemoryManager (see page 3587)	Returns the entry points of the currently installed memory manager.
	GetMemoryManagerState (see page 3591)	Fetches state information for the Memory Manager.
	GetMemoryMap (see page 3592)	Fetches the memory map.
	GetMinimumBlockAlignment (see page 3592)	Gets the minimum memory block alignment.
	GetModuleFileName (see page 3593)	Returns the fully qualified name for a module, given its handle.
	GetVariantManager (see page 3594)	Returns the entry points of the routines that define variant behavior.
	Halt (see page 3594)	Initiates abnormal termination of a program.
	Halt (see page 3594)	Initiates abnormal termination of a program.
	Hi (see page 3595)	Returns the high-order byte of X as an unsigned value.
	High (see page 3596)	Returns the highest value in the range of an argument.
	Inc (see page 3597)	Increments an ordinal value by one or N.
	Inc (see page 3597)	Increments an ordinal value by one or N.

	Include (see page 3598)	Adds an element to a Delphi set or adds an event handler in .NET applications.
	Initialize (see page 3599)	Initializes a dynamically allocated variable.
	Initialize (see page 3599)	Initializes a dynamically allocated variable.
	Insert (see page 3601)	Inserts a substring into a string beginning at a specified point.
	Int (see page 3602)	Returns the integer part of a real number.
	IOResult (see page 3602)	Returns the status of the last I/O operation performed.
	IsMemoryManagerSet (see page 3604)	Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 3660) procedure.
	IsVariantManagerSet (see page 3606)	Indicates whether variant support is implemented in an application.
	Length (see page 3607)	Returns the number of characters in a string or elements in an array.
	Ln (see page 3608)	Returns the natural log of a real expression.
	Lo (see page 3609)	Returns the low order Byte of argument X.
	LoadResourceModule (see page 3609)	This is function System.LoadResourceModule.
	LoadResourceString (see page 3610)	This is function System::LoadResourceString.
	LoadResString (see page 3610)	This is function System.LoadResString.
	Low (see page 3610)	Returns the lowest value in a range.
	Mark (see page 3612)	This is function System.Mark.
	MemoryBarrier (see page 3612)	This is function System.MemoryBarrier.
	MkDir (see page 3612)	Creates a new subdirectory.
	MkDir (see page 3612)	Creates a new subdirectory.
	MonitorEnter (see page 3614)	Prohibits the access of all other threads but the calling one to the specified object.
	MonitorExit (see page 3614)	Releases a lock of the calling thread on the specified object.
	MonitorPulse (see page 3615)	Notifies the next thread in the waiting queue that it will be able to lock the specified object as soon as the calling thread releases it.
	MonitorPulseAll (see page 3615)	Signals all threads in the waiting queue that either one of them will be able to lock the specified object as soon as the calling thread releases it.
	MonitorTryEnter (see page 3616)	Attempts to lock the specified object, so that only the calling thread is able to access it.
	MonitorWait (see page 3616)	Drops the lock on the specified object and puts the calling thread in the waiting queue for a specified amount of time.
	Move (see page 3617)	Copies bytes from a source to a destination.
	MoveChars (see page 3618)	Copies bytes from a source to a destination.
	New (see page 3619)	Creates a new dynamic variable and sets P to point to it.
	Odd (see page 3621)	Returns true if argument is an odd number.
	OleStrToString (see page 3622)	Copies data received from a COM interface to a string.
	OleStrToStrVar (see page 3622)	Copies a string from its COM representation to an existing Delphi string.
	OleStrToStrVar (see page 3622)	Copies a string from its COM representation to an existing Delphi string.
	Ord (see page 3623)	Returns the ordinal value of an ordinal-type expression.
	ParamCount (see page 3623)	Returns the number of parameters passed on the command line.
	ParamStr (see page 3625)	Returns a specified parameter from the command-line.
	Pi (see page 3626)	Returns 3.1415926535897932385.
	Pos (see page 3627)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3627)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pos (see page 3627)	Returns the index value of the first character in a specified substring that occurs in a given string.
	Pred (see page 3628)	Returns the predecessor of the argument.
	Ptr (see page 3630)	Converts a specified address to a pointer.
	PUCS4Chars (see page 3632)	Converts a UCS4 string to a null-terminated array of UCS4 characters.
	RaiseList (see page 3632)	This is function System.RaiseList.
	Random (see page 3632)	Generates random numbers within a specified range.
	Random (see page 3632)	Generates random numbers within a specified range.
	Randomize (see page 3633)	Initializes the random number generator with a random value.
	Read (see page 3634)	Read reads data from a file.
	Read (see page 3634)	Read reads data from a file.
	Read (see page 3634)	Read reads data from a file.

	Read (see page 3634)	Read reads data from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReadLn (see page 3636)	Reads a line of text from a file.
	ReallocMem (see page 3637)	ReallocMem reallocates a memory block.
	ReallocMemory (see page 3638)	ReallocMemory resizes a memory block.
	RegisterExpectedMemoryLeak (see page 3640)	Registers an expected memory leak.
	RegisterModule (see page 3640)	Registers the specified module.
	Release (see page 3641)	This is function System.Release.
	ReleaseExceptionObject (see page 3641)	Releases exception object acquired by AcquireExceptionObject (see page 3512).
	RemoveModuleUnloadProc (see page 3641)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	RemoveModuleUnloadProc (see page 3641)	RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.
	Rename (see page 3642)	Changes the name of an external file.
	Rename (see page 3642)	Changes the name of an external file.
	Reset (see page 3643)	Opens an existing file.
	Reset (see page 3643)	Opens an existing file.
	Reset (see page 3643)	Opens an existing file.
	Reset (see page 3643)	Opens an existing file.
	Rewrite (see page 3645)	Creates a new file and opens it.
	Rewrite (see page 3645)	Creates a new file and opens it.
	RmDir (see page 3647)	Deletes an empty subdirectory.
	RmDir (see page 3647)	Deletes an empty subdirectory.
	Round (see page 3648)	Returns the value of X rounded to the nearest whole number.
	RunError (see page 3649)	Stops execution and generates a runtime error.
	RunError (see page 3649)	Stops execution and generates a runtime error.
	Seek (see page 3650)	Moves the current position of a file to a specified component.
	SeekEof (see page 3651)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEof (see page 3651)	Returns the end-of-file status of a file, ignoring whitespace.
	SeekEoln (see page 3652)	Returns the end-of-line status of a file, ignoring whitespace.
	SeekEoln (see page 3652)	Returns the end-of-line status of a file, ignoring whitespace.
	Set8087CW (see page 3653)	Sets both the control word in the floating point unit and the variable NoErrMsg (see page 3832) declared in the system unit.
	SetCodePage (see page 3655)	This is function System.SetCodePage.
	SetInOutRes (see page 3655)	This is function System.SetInOutRes.
	SetLastError (see page 3656)	SetLastError sets the last API error code.
	SetLength (see page 3657)	Sets the length of a string or dynamic-array variable.
	SetLineBreakStyle (see page 3659)	Determines the end-of-line and end-of-file conventions for text file I/O.
	SetMemoryManager (see page 3660)	Sets entry points of the memory manager.
	SetMemoryManager (see page 3660)	Sets entry points of the memory manager.
	SetMinimumBlockAlignment (see page 3664)	Sets the minimum memory block alignment.
	SetMultiByteConversionCodePage (see page 3665)	Sets the code page used internally whenever multibyte characters are converted to/from WideChar.
	SetRaiseList (see page 3665)	This is function System.SetRaiseList.
	SetString (see page 3665)	Sets the contents and length of the given string.
	SetTextBuf (see page 3667)	Assigns an I/O buffer to a text file.
	SetTextBuf (see page 3667)	Assigns an I/O buffer to a text file.
	SetVariantManager (see page 3668)	Sets the entry points of routines that define variant behavior.
	ShareMemoryManager (see page 3668)	Shares the module's memory manager.
	Sin (see page 3669)	Returns the sine of the angle in radians.
	SizeOf (see page 3670)	Returns the number of bytes occupied by a variable or type.
	Slice (see page 3671)	Returns a sub-section of an array.
	Sqr (see page 3674)	Returns the square of a number.

	Sqr (see page 3674)	Returns the square of a number.
	Sqrt (see page 3675)	Returns the square root of X.
	Str (see page 3676)	Formats a string and returns it to a variable.
	Str (see page 3676)	Formats a string and returns it to a variable.
	StringCodePage (see page 3676)	Returns the code page associated with the string.
	StringCodePage (see page 3676)	Returns the code page associated with the string.
	StringElementSize (see page 3677)	Returns the number of bytes that represent a character in the string.
	StringElementSize (see page 3677)	Returns the number of bytes that represent a character in the string.
	StringOfChar (see page 3677)	Returns a string with a specified number of repeating characters.
	StringOfChar (see page 3677)	Returns a string with a specified number of repeating characters.
	StringRefCount (see page 3678)	Returns the reference count of a string.
	StringRefCount (see page 3678)	Returns the reference count of a string.
	StringToOleStr (see page 3678)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToOleStr (see page 3678)	Allocates memory and copies an AnsiStringa string to the OLE format.
	StringToWideChar (see page 3679)	Returns a UNICODE string from an AnsiString.
	Succ (see page 3680)	Returns the successor of an argument.
	Swap (see page 3680)	Exchanges high order byte with the low order byte of an integer or word.
	SysAllocMem (see page 3681)	This is function System.SysAllocMem.
	SysFreeMem (see page 3681)	Frees the memory pointed to by a specified pointer.
	SysGetMem (see page 3682)	Allocates a specified number of bytes and returns a pointer to them.
	SysReallocMem (see page 3683)	Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.
	SysRegisterExpectedMemoryLeak (see page 3683)	This is function System.SysRegisterExpectedMemoryLeak.
	SysUnregisterExpectedMemoryLeak (see page 3684)	This is function System.SysUnregisterExpectedMemoryLeak.
	TextStart (see page 3684)	This is function System.TextStart.
	Trunc (see page 3684)	Truncates a real number to an integer.
	Truncate (see page 3685)	Deletes all the records after the current file position.
	UCS4StringToUnicodeString (see page 3686)	Converts an UCS-4 encoded string to an Unicode string.
	UCS4StringToWideString (see page 3687)	Converts a string of 4-byte Unicode characters into a WideString (see page 3467) value.
	UnicodeStringToUCS4String (see page 3689)	Converts an Unicode string to its UCS-4 encoded representation.
	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.
	UnicodeToUtf8 (see page 3690)	Converts a string of Unicode characters into a UTF-8 string.
	UniqueString (see page 3693)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3693)	Ensures that a given string has a reference count of one.
	UniqueString (see page 3693)	Ensures that a given string has a reference count of one.
	UnregisterExpectedMemoryLeak (see page 3694)	Unregisters an expected memory leak.
	UnregisterModule (see page 3694)	Unregisters the specified module.
	UpCase (see page 3695)	Converts a character to uppercase.
	UpCase (see page 3695)	Converts a character to uppercase.
	UTF8Decode (see page 3696)	Converts a UTF8 string to a Unicode string (WideString (see page 3467)).
	UTF8Encode (see page 3696)	Converts a WideString (see page 3467) value to a UTF8 string.
	UTF8Encode (see page 3696)	Converts a WideString (see page 3467) value to a UTF8 string.
	UTF8Encode (see page 3696)	Converts a WideString (see page 3467) value to a UTF8 string.
	UTF8EncodeToShortString (see page 3697)	Converts a string to its UTF-8 encoded representation.
	UTF8EncodeToShortString (see page 3697)	Converts a string to its UTF-8 encoded representation.
	UTF8EncodeToShortString (see page 3697)	Converts a string to its UTF-8 encoded representation.
	Utf8ToAnsi (see page 3698)	Converts a UTF8 string to a string of Ansi characters.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	UTF8ToString (see page 3698)	Converts an UTF-8 encoded string to a string.
	Utf8ToUnicode (see page 3699)	Converts a string of UTF-8 characters to a Unicode string.

	Utf8ToUnicode (see page 3699)	Converts a string of UTF-8 characters to a Unicode string.
	UTF8ToUnicodeString (see page 3700)	Converts an UTF-8 encoded string to an Unicode string.
	UTF8ToUnicodeString (see page 3700)	Converts an UTF-8 encoded string to an Unicode string.
	UTF8ToUnicodeString (see page 3700)	Converts an UTF-8 encoded string to an Unicode string.
	UTF8ToWideString (see page 3700)	Converts an UTF-8 encoded string to a WideString (see page 3467).
	Val (see page 3701)	Converts a string to a numeric representation.
	VarArrayRedim (see page 3702)	Resizes a Variant (see page 3421) array.
	VarClear (see page 3704)	Empties a Variant (see page 3421), so that it is unassigned.
	WideCharLenToString (see page 3705)	Converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)**.
	WideCharLenToStrVar (see page 3706)	WideCharLenToStrVar converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)** or **AnsiString**.
	WideCharLenToStrVar (see page 3706)	WideCharLenToStrVar converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)** or **AnsiString**.
	WideCharToString (see page 3706)	Converts null-terminated **WideString (see page 3467)** string to **UnicodeString (see page 3381)**.
	WideCharToStrVar (see page 3707)	Converts **WideString (see page 3467)** string to **UnicodeString (see page 3381)**.
	WideStringToUCS4String (see page 3707)	Converts a WideString (see page 3467) value to a string of 4-byte Unicode characters.
	Write (see page 3707)	Writes to either a typed file or a text file
	Write (see page 3707)	Writes to either a typed file or a text file
	Write (see page 3707)	Writes to either a typed file or a text file
	Write (see page 3707)	Writes to either a typed file or a text file
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	WriteLn (see page 3710)	Writes to a text file and adds an end-of-line marker.
	YieldProcessor (see page 3711)	This is function System.YieldProcessor.

35.3.1 System.Abs Function

Returns an absolute value.

Pascal

```
procedure Abs(X);
```

C++

```
Abs( X);
```

File

System

Description

Abs returns the absolute value of the argument, X.

X is an integer-type or real-type expression.

Delphi Examples:

```
{
This example converts negative floating point numbers and
integers to postive.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  Edit1.Text := '-2.3';
  Edit3.Text := FloatToStr(Abs(StrToFloat(Edit1.Text)));    { 2.3 }
  Edit2.Text := '-157';
  Edit4.Text := IntToStr(Abs(StrToInt(Edit2.Text)));    { 157 }
end;
```

35.3.2 System.AcquireExceptionObject Function

Retains access to exception object.

Pascal

```
function AcquireExceptionObject: Pointer;
```

C++

```
void * AcquireExceptionObject();
```

File

System

Description

AcquireExceptionObject returns a pointer to the current exception object and prevents the exception object from being deallocated when the current exception handler exits.

Note: AcquireExceptionObject increments the exception object's reference count. Make sure that the reference count is decremented when the exception object is no longer needed. This happens automatically if you use the exception object to re-raise the exception. In all other cases, every call to AcquireExceptionObject must have a matching call to ReleaseExceptionObject (see page 3641). AcquireExceptionObject/ReleaseExceptionObject (see page 3641) sequences can be nested.

Delphi Examples:

```
{
This example demonstrates exception handling to catch runtime
errors.
}

procedure TForm2.btIOErrorClick(Sender: TObject);
var
  ExceptionObj : TObject;
begin
  {
  Try to write something onto the console - will raise an
  exception.
  }
  try
  WriteLn('This will generate an error because there is no' +
          ' console attached!');
  except
    ExceptionObj := ExceptObject;
    if ExceptionObj = nil then
      MessageDlg('No exception', mtError, [mbOK], 0)
    else
    begin
      MessageDlg(ExceptionObj.ToString, mtError, [mbOK], 0);
    end;
  end;
end;
```

```
{$OVERFLOWCHECKS ON}
{$OPTIMIZATION OFF}
{$HINTS OFF}
procedure TForm2.btOverflowErrClick(Sender: TObject);
var
  b : Cardinal;
  ExceptionPtr : Pointer;
begin
  {
  Simulate an overflow: Note, enabled the overflow
  checking and disabled optimizations because Delphi's
  compiler will not compile this code otherwise.
  }
  ExceptionPtr := nil;
  try
    b := $FFFFFFFF;
    b := b * b;
  except
    ExceptionPtr := AcquireExceptionObject;
  end;

  // Check exception
  if ExceptionPtr = nil then
    MessageDlg('No exception', mtError, [mbOK], 0)
  else
  begin
    MessageDlg(TObject(ExceptionPtr).ToString, mtError, [mbOK], 0);
    ReleaseExceptionObject;
  end;
end;
{$HINTS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}

procedure TForm2.btRuntimeErrorClick(Sender: TObject);
var

  ExceptionObj : TObject;
  begin
  { Simulate an access violation }
  try
    System.Error(reAccessViolation);
  except
    ExceptionObj := ExceptObject;
    if ExceptionObj = nil then
      MessageDlg('No exception', mtError, [mbOK], 0)
    else
    begin
      MessageDlg(ExceptionObj.ToString, mtError, [mbOK], 0);
    end;
  end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin

end;
```

See Also

ReleaseExceptionObject (see page 3641)

ExceptObject (see page 3562)

35.3.3 System.AddModuleUnloadProc Function

AddModuleUnloadProc adds a call to your defined routine when a package is unloaded.

Pascal

```
procedure AddModuleUnloadProc(Proc: TModuleUnloadProc); overload;
procedure AddModuleUnloadProc(Proc: TModuleUnloadProcLW); overload;
```

C++

```
AddModuleUnloadProc(TModuleUnloadProc Proc);
AddModuleUnloadProc(TModuleUnloadProcLW Proc);
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your **Proc** routine. In essence, AddModuleUnloadProc adds your callback routine to the head of the list of internal routines that Delphi invokes to perform the unload.

Use RemoveModuleUnloadProc (see page 3641) to reverse the effect of AddModuleUnloadProc.

See Also

RemoveModuleUnloadProc (see page 3641)

TModuleUnloadProc (see page 3792)

35.3.4 System.Addr Function

Returns a pointer to a specified object.

Pascal

```
function Addr(var X): Pointer;
```

C++

```
void * Addr( X);
```

File

System

Description

The Addr function returns the address of a specified object. X is any variable, procedure or function identifier. The result is a pointer to X.

The result of Addr is of the predefined type Pointer, which means that it is assignment-compatible with all pointer types but can't be dereferenced directly without a typecast.

Note: Addr is equivalent to the @ operator except that it is unaffected by the $T compiler directive.

Delphi Examples:

```
{
This example uses the Addr function to get the address of a
numeric value that is stored as custom data in a tree node.
Once the assignment has been made, changes to the entries in
the NodeNumbers array can be read by dereferencing the
pointers that are stored in the Data property of the Tree
nodes.  This example requires a populated TreeView.
}

var
  NodeNumbers: array [0 .. 200] of String;

procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
 begin
  with TreeView1 do
  begin
    for I := 0 to Items.Count - 1 do
    begin
      NodeNumbers[I] := Items[I].Text;
      Items[I].Data := Addr(NodeNumbers[I]);
      StringGrid1.Cells[0, I + 1] := IntToStr(I);
      StringGrid1.Cells[1, I + 1] := IntToStr(Integer(Items[I].Data));
      StringGrid1.Cells[2, I + 1] := NodeNumbers[I];
    end;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  I : Integer;
  addr : Pointer;
begin
  I := StrToInt(Edit2.Text);
  addr := TreeView1.Items[I].Data;
  string(addr^) := Edit1.Text;  // NodeNumbers has been altered
  TreeView1.Items[I].Text := NodeNumbers[I]; // Update TreeView
  StringGrid1.Cells[2, I + 1] := NodeNumbers[I];  // Update StringGrid
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[0, 0] := 'Node index';
  StringGrid1.Cells[1, 0] := 'Node addr';
  StringGrid1.Cells[2, 0] := 'Node text';
end;
```

See Also

Ptr (see page 3630)

35.3.5 System.AllocMem Function

AllocMem allocates a memory block and initializes each byte to zero.

Pascal

```
function AllocMem(Size: Cardinal): Pointer;
```

C++

```
void * AllocMem(unsigned Size);
```

File

System

Description

AllocMem allocates a block of the given **Size** on the heap, and returns the address of this memory. Each byte in the allocated buffer is set to zero. To dispose of the buffer, use FreeMem (see page 3577). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory does not need to be zero-initialized, it is more efficient to use GetMem (see page 3584) instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
```

```
  Open dialog box on a form. When the button is clicked, the
  user is prompted for a filename. When the user clicks OK,
  the specified file is opened, read into a buffer, and closed.
  Then the buffer is displayed in two columns of the string
  grid. The first column contains the character values in the
  buffer. The second column contains the numeric values of the
  characters in the buffer.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    iFileHandle: Integer;
    iFileLength: Integer;
    iBytesRead: Integer;
    Buffer: PAnsiChar;
    i: Integer;
  begin
    if OpenDialog1.Execute then
    begin
      try
        iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
        iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
        FileSeek(iFileHandle,0,0);
        Buffer := System.AllocMem(iFileLength + 1);
        iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
        FileClose(iFileHandle);
        for i := 0 to iBytesRead-1 do
        begin
          StringGrid1.RowCount := StringGrid1.RowCount + 1;
          StringGrid1.Cells[1,i+1] := Buffer[i];
          StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
        end;
      finally
        FreeMem(Buffer);
      end;
    end;
  end;
```

See Also

FreeMem (see page 3577)

FreeMemory (see page 3579)

GetMem (see page 3584)

GetMemory (see page 3585)

EOutOfMemory (see page 3942)

ReallocMem (see page 3637)

ReallocMemory (see page 3638)

35.3.6 System.AnsiToUtf8 Function

Converts a string encoded in Ansi to UTF-8.

Pascal

```
function AnsiToUtf8(const S: string): RawByteString;
```

C++

```
RawByteString AnsiToUtf8(const UnicodeString S);
```

File

System

Description

AnsiToUtf8 converts the string specified by **S**, which uses the ANSI encoding system, to UTF-8.

Delphi Examples:

```
{
This example converts various string formats to and from Unicode.
This example requires two text edits and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
  LFileStream: TFileStream;
  LOffset: Integer;
  LEncoding, DestEncoding: TEncoding;
  ComboIndex: Integer;
  ByteString: RawByteString;
  FileText: string;
  ucs4Text: UCS4String;
  wideStringText: WideString;
  utf8Text: UTF8String;
  L: Cardinal;

begin
  LEncoding:= nil;
  ComboIndex := ComboBox1.ItemIndex;
  LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    // Read file into buffer
    SetLength(LBuffer, LFileStream.Size);
    LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));
    // Get data encoding of data read
    LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
  finally
    LFileStream.Free;
  end;

  // Process various types
  case ComboIndex of

    // AnsiToUtf8
    0:
    begin
      // Identify encoding and convert buffer to Ansi
      DestEncoding := TEncoding.ASCII;
      LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
        LOffset, Length(LBuffer) - LOffset);
      LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
      FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
      ByteString := AnsiToUtf8(FileText);
      ShowMessage(UTF8ToString(ByteString));

      // Write to file
      LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
      try
        // New destination encoding
        DestEncoding := TEncoding.UTF8;
        LByteOrderMark := DestEncoding.GetPreamble;
        LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
        LFileStream.Write(ByteString[1], Length(ByteString));
```

```
      finally
        LFileStream.Free;
      end;
    end;

    // UCS4StringToWideString
    1:
    begin
      // Identify encoding and convert buffer to UCS4
      DestEncoding := TEncoding.Unicode;
      LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
        LOffset, Length(LBuffer) - LOffset);
      LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
      FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
      ucs4Text := UnicodeStringToUCS4String(FileText);
      wideStringText := UCS4StringToWideString(ucs4Text);
      ShowMessage(wideStringText);

      // Write to file
      LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
      try
        // New destination encoding
//        DestEncoding := TEncoding.UTF8;
        LByteOrderMark := DestEncoding.GetPreamble;
        LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
        LFileStream.Write(wideStringText[1], Length(wideStringText) * SizeOf(Char));
      finally
        LFileStream.Free;
      end;
    end;

    // UnicodeToUtf8
    2:
    begin
      // Identify encoding and convert buffer to Unicode
      DestEncoding := TEncoding.Unicode;
      LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
        LOffset, Length(LBuffer) - LOffset);
      LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
      FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
      L := Length(FileText);
      SetLength(utf8Text, L * SizeOf(Char) + 1);
      L := UnicodeToUtf8(PAnsiChar(utf8Text), Length(utf8Text), PWideChar(FileText), L);
      ShowMessage(UTF8ToString(utf8Text));

      // Write to file
      LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
      try
        // New destination encoding
        DestEncoding := TEncoding.UTF8;
        LByteOrderMark := DestEncoding.GetPreamble;
        LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
        LFileStream.Write(utf8Text[1], L);
      finally
        LFileStream.Free;
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  StringList: TStrings;
begin
  Edit1.Text := 'Sample.txt';
  Edit2.Text := 'Dest.txt';
```

```
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('AnsiToUtf8');
        Add('UCS4StringToWideString');
        Add('UnicodeToUtf8');
      end;
      with ComboBox1 do begin
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

Utf8ToAnsi (see page 3698)

Utf8ToUnicode (see page 3699)

WideStringToUCS4String (see page 3707)

35.3.7 System.Append Function

Prepares an existing file for adding text to the end.

Pascal

```
procedure Append(var F: Text);
```

C++

```
Append(Text F);
```

File

System

Description

Call Append to ensure that a file is opened with write-only access with the file pointer positioned at the end of the file. F is a text file variable and must be associated with an external file using AssignFile (see page 3525). If no external file of the given name exists, an error occurs. If F is already open, it is closed, then reopened. The current file position is set to the end of the file.

Note: If a Ctrl+Z (ASCII 26) is present in the last 128-byte block of the file, the current file position is set so that the next character added to the file overwrites the first Ctrl+Z in the block. In this way, text can be appended to a file that terminates with a Ctrl+Z.

If F was not assigned a name, then, after the call to Append, F refers to the standard output file.

Delphi Examples:

```
{
Click the button and select a file to append a string to the bottom of the file.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TextFile;
begin
  if OpenDialog1.Execute then
  begin                    { open a text file }
```

```
    AssignFile(f, OpenDialog1.FileName);
    Append(f);
    Writeln(f, 'I am appending some stuff to the end of the file.');
    { insert code here that would require a Flush before closing the file }
    Flush(f);  { ensures that the text was actually written to file }
    CloseFile(f);
    MessageDlg(OpenDialog1.FileName + ' has been altered, please validate.', mtInformation,
[mbOK], 0)
  end;
end;
```

See Also

AssignFile (see page 3525)

FileClose (see page 4121)

Reset (see page 3643)

Rewrite (see page 3645)

35.3.8 System.ArcTan Function

Calculates the arctangent of a given number.

Pascal

```
function ArcTan(const X: Extended): Extended;
```

C++

```
Extended ArcTan(const Extended X);
```

File

System

Description

In Delphi code, ArcTan returns the arctangent of X.

X is a real-type expression that gives an angle in radians.

C++ Examples:

```
/*
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Extended ex = 16 * ArcTan(1.0/5.0) - 4.0 * ArcTan(1.0/239.0); // Machin's formula
  String mystring = FloatToStr(ex);
  Edit2->Text = mystring;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Text = "3.14159265358979";
}
```

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit3.Text := mystring;
  Str(ex:25:23, mystring);
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

See Also

Cos (see page 3549)

Sin (see page 3669)

ArcTan2

Tan

ArcSin

ArcCos

35.3.9 System.Assert Function

Tests whether a Boolean expression is true.

Pascal

```
procedure Assert(expr: Boolean); overload;
procedure Assert(expr: Boolean; const msg: string); overload;
```

C++

```
Assert(Boolean expr);
Assert(Boolean expr, const UnicodeString msg);
```

File

System

Description

In Delphi code, use Assert as a debugging tool to test that conditions assumed to be true are never violated. Assert provides an opportunity to intercept an unexpected condition and halt a program rather than allow execution to continue under unanticipated

conditions.

Assert takes a Boolean expression and an optional message string as parameters. If the Boolean test fails, Assert raises an EAssertionFailed exception. If a message string was passed to Assert, the exception object is created with that string. Otherwise it is created with a default string indicating that the assertion failed. The message is displayed along with the complete path, filename, and the line number on which Assert failed.

The SysUtils (see page 3901) unit causes runtime errors to be turned into exceptions. If SysUtils (see page 3901) is not used anywhere in your application, you will get a runtime error 227 rather than an EAssertionFailed exception. This runtime error will halt the program.

Because assertions are not usually used in shipping versions of a product, compiler directives are provided to disable the generation of assertion code:

$ASSERTIONS ON/OFF(long form)

$C +/-(short form)

These are global switches that affect the entire source file where they occur, regardless of their position in the file. It is not possible to enable and disable assertions for something smaller than a source file. Assertions are on by default.

Delphi Examples:

```
{
This example exercises the System Assert function.  The first
call passes and the second call fails.
}

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
    property Data: string read FData write FData;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, '');
  AStorage.Data := s;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Storage: TStorage;
begin
  Storage := TStorage.Create;
```

```
    try
      ModifyStorage(Storage, 'Hello world');
    finally
      Storage.Free;
    end;

    // The following call is buggy and triggers the Assert
    ModifyStorage(nil, 'Ooops');
  end;
```

See Also

EAssertionFailed (see page 3918)

35.3.10 System.Assigned Function

Tests for a nil (unassigned) pointer or procedural variable.

Pascal

```
function Assigned(const P): Boolean;
```

C++

```
Boolean Assigned(const  P);
```

File

System

Description

Use Assigned to determine whether the pointer or procedure referenced by P is nil. P must be a variable reference of a pointer or procedural type. Assigned(P) corresponds to the test P<> nil for a pointer variable, and @P <> nil for a procedural variable.

Assigned returns false if P is nil, true otherwise.

Note: Assigned can't detect a dangling pointer--that is, one that isn't nil but no longer points to valid data. For example, in the code example for Assigned, Assigned won't detect the fact that P isn't valid.

Delphi Examples:

```
{
This example demonstrates the System Assigned function succeeds.
}
procedure TForm1.Button1Click(Sender: TObject);
var P: Pointer;
begin
  P := nil;
  if Assigned (P) then
    MessageDlg('You won''t see this', mtInformation, [mbOK], 0);
  GetMem(P, 1024);  {P valid}
  FreeMem(P, 1024); {P no longer valid and still not nil}
  if Assigned (P) then
    MessageDlg('You''ll see this', mtInformation, [mbOK], 0);
end;
```

35.3.11 System.AssignFile Function

Associates the name of an external file with a file variable.

Pascal

```
procedure AssignFile(var F; FileName: string);
```

C++

```
AssignFile( F, UnicodeString FileName);
```

File

System

Description

Call AssignFile to initialize a file variable in Delphi code. F is a file variable of any file type. FileName is a string-type expression or an expression of type PChar if extended syntax is enabled.

After calling AssignFile, F is associated with the external file until F is closed. All further operations on the file variable F operate on the external file named by FileName.

When the FileName parameter is empty, AssignFile associates F with the standard input or standard output file. If assigned an empty name, after a call to Reset (see page 3643) (F), F refers to the standard input file, and after a call to Rewrite (see page 3645) (F), F refers to the standard output file.

Do not use AssignFile on a file variable that is already open.

Note: To avoid scope conflicts, AssignFile replaces the Assign procedure that was available in early versions of the Delphi product. However, for backward compatibility Assign is still available.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
```

```
        if (filehandle = -1) then
        begin
          MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                     ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
          Continue;
        end
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
     finally
     { do something here }
     end;
    end;
  end;
  {
  Use a TOpenDialog to select a file.  Place the first line of
  the file into a TEdit.  Click on the text edit to open the
  open dialog box.
  }
  procedure TForm1.Edit1Click(Sender: TObject);
  var
    F: TextFile;
    S: string;
  begin
    if OpenDialog1.Execute then            { Display Open dialog box }
    begin
      AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
      Reset(F);
      Readln(F, S);                        { Read first line of file }
      Edit1.Text := S;                     { Put string in a TEdit control }
      CloseFile(F);
    end;
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

Append (see page 3520)

CloseFile (see page 3539)

Reset (see page 3643)

Rewrite (see page 3645)

35.3.12 System.AttemptToUseSharedMemoryManager Function

Attempts to use a shared memory manager.

Pascal

```
function AttemptToUseSharedMemoryManager: Boolean;
```

C++

```
Boolean AttemptToUseSharedMemoryManager();
```

File

System

Description

AttemptToUseSharedMemoryManager searches the current process for a shared memory manager. If no memory has been allocated using this module's memory manager, it will switch to using the shared memory manager. It will return **True** if it found a shared memory manager to which it could switch. It will return **False** otherwise.

Note: The memory manager cannot be changed once memory has been allocated, since the new memory manager cannot free the pointers allocated through a different memory manager. This function is not thread safe.

See Also

Memory Management

Configuring the Memory Manager

Sharing Memory

35.3.13 System.BeginThread Function

Spawns a separate thread of execution.

Pascal

```
function BeginThread(SecurityAttributes: Pointer; StackSize: LongWord; ThreadFunc:
TThreadFunc; Parameter: Pointer; CreationFlags: LongWord; var ThreadId: LongWord): Integer;
```

C++

```
int BeginThread(void * SecurityAttributes, LongWord StackSize, TThreadFunc ThreadFunc, void *
Parameter, LongWord CreationFlags, LongWord ThreadId);
```

File

System

Description

Use this routine or a TThread object to spawn separate threads of execution. BeginThread spawns a new thread of execution and sets the global IsMultiThread (see page 3829) variable, thereby making the heap thread-safe.

ThreadFunc is the thread function. This is the code that executes in the new thread of execution. ThreadFunc should handle all of its own exceptions. However, because BeginThread sets up an execution frame, the system's default exception handler will catch any exceptions that escape ThreadFunc.

Parameter points to a single 32-bit parameter that is passed, uninterpreted, to ThreadFunc.

ThreadId returns the unique identifier for the new thread. This ID appears in the Thread Status box of the debugger.

SecurityAttributes is a pointer to a Windows SecurityAttributes record (structure). For details on security attributes, see the Microsoft documentation.

StackSize is the size of the stack reserved for the new thread when CreationFlags includes STACK_SIZE_IS_A_RESERVATION. Otherwise, this parameter is ignored.

CreationFlags is a set of bits or'ed together that are drawn from the following:

Value	Meaning
CREATE_SUSPENDED	The thread is created in a suspended state and does not run until the Windows API ResumeThread function is called. If this flag is not included, the new thread starts running as soon as it is created.
STACK_SIZE_PARAM_IS_A_RESERVATION	The StackSize parameter specifies the stack size for the thread. If this flag is not included, the new thread is assigned a default stack size by the linker.

On Windows, BeginThread returns the Windows thread handle. A return value of 0 indicates failure.

Attribute is a structure containing information about the thread such as its priority, its schedule policy, and its stack size.

On Linux, Begin Thread returns 0 if the thread is successfully spawned, an error code otherwise.

See Also

EndThread (see page 3556)

IsMultiThread (see page 3829)

Defining Thread Objects

35.3.14 System.BlockRead Function

Reads one or more records from an open file into a variable.

Pascal

```
procedure BlockRead(var F: File; var Buf; Count: Integer); overload;
procedure BlockRead(var F: File; var Buf; Count: Integer; var AmtTransferred: Integer);
overload;
```

C++

```
BlockRead(File F,  Buf, int Count);
BlockRead(File F,  Buf, int Count, int AmtTransferred);
```

File

System

Description

F is an untyped file variable, Buf is any variable, Count is an expression of type Integer, and AmtTransferred is an optional variable of type Integer.

BlockRead reads Count or fewer records from the file F into memory, starting at the first byte occupied by Buf. The actual number of complete records read (less than or equal to Count) is returned in AmtTransferred.

The entire transferred block occupies at most Count * RecSize bytes. RecSize is the record size specified when the file was opened (or 128 if the record size was not specified).

If the entire block was transferred, AmtTransferred is equal to Count.

If AmtTransferred is less than Count, ReadBlock reached the end of the file before the transfer was complete. If the file's record size is greater than 1, AmtTransferred returns the number of complete records read.

If AmtTransferred isn't specified, an I/O error occurs if the number of records read isn't equal to Count. If the $I+ compiler directive is in effect, errors raise an EInOutError exception.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      // Use CloseFile rather than Close; Close provided for backward compatibility
      CloseFile(FromF);
      CloseFile(ToF);
      Canvas.TextOut(120, 10, ' done.');
    end;
  end;
end;
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
  F: file;
  Size: Integer;
  Buffer: PAnsiChar;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    Reset(F, 1);
    try
      Size := FileSize(F);
      GetMem(Buffer, Size);
      try
        BlockRead(F, Buffer^, Size);
        Memo1.Lines.Add(AnsiString(Buffer));
      finally
        FreeMem(Buffer);
      end;
    finally
      CloseFile(F);
    end;
  end;
end;
```

See Also

TFileStream

EInOutError (see page 3929)

35.3.15 System.BlockWrite Function

Writes one or more records from a variable to an open file.

Pascal

```
procedure BlockWrite(var f: File; var Buf; Count: Integer); overload;
procedure BlockWrite(var f: File; var Buf; Count: Integer; var AmtTransferred: Integer);
overload;
```

C++

```
BlockWrite(File f,  Buf, int Count);
BlockWrite(File f,  Buf, int Count, int AmtTransferred);
```

File

System

Description

F is an untyped file variable, Buf is any variable, Count is an expression of type Integer, and AmtTransferred is an optional variable of type Integer.

BlockWrite writes Count or fewer records to the file F from memory, starting at the first byte occupied by Buf. The actual number of complete records written (less than or equal to Count) is returned in AmtTransferred.

The entire block transferred occupies at most Count * RecSize bytes. RecSize is the record size specified when the file was opened (or 128 if the record size was unspecified).

If the entire block is transferred, AmtTransferred is equal to Count on return.

If AmtTransferred is less than Count, the disk became full before the transfer was complete. In this case, if the file's record size is greater than 1, AmtTransferred returns the number of complete records written.

BlockWrite advances the current file position by AmtTransferred records.

If AmtTransferred isn't specified, an I/O error occurs if the number written isn't equal to Count. If the $I+ compiler directive is in effect, errors raise an EInOutError exception.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      // Use CloseFile rather than Close; Close provided for backward compatibility
      CloseFile(FromF);
      CloseFile(ToF);
      Canvas.TextOut(120, 10, ' done.');
    end;
  end;
end;
```

See Also

TFileStream

EInOutError (see page 3929)

35.3.16 System.Break Function

Causes the flow of control to exit a for, while, or repeat statement.

Pascal

```
procedure Break;
```

C++

```
Break();
```

File

System

Description

The Break procedure causes the flow of control in Delphi code to exit a for, while, or repeat statement and continue at the next

statement following the loop statement.

A call to Break must be contained in a for, while, or repeat statement, or the compiler reports an error.

Note: Break does not violate the flow of control dictated by a try..finally construct. If a break occurs inside the try clause, the finally clause is entered.

Delphi Examples:

```
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'country.list';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
```

```
    TmpStr: string;
  begin
    { first, process the keystroke to obtain the current string }
    { This code requires all items in list to be uppercase}
    if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
    ct := (Sender as TComboBox).Items.Count;
    with (Sender as TComboBox) do
    begin
      SelSt := SelStart;
      if (Key = Chr(vk_Back)) and (SelLength <> 0) then
       TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
      else if Key = Chr(vk_Back) then {SelLength = 0}
       TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
      else {Key in ['A'..'Z', etc]}
       TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
      if TmpStr = '' then Exit;
      { update SelSt to the current insertion point }
      if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
      else if Key <> Chr(vk_Back) then Inc(SelSt);
      Key := #0; { indicate that key was handled }
      if SelSt = 0 then
      begin
        Text:= '';
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

Halt (see page 3594)

35.3.17 System.ChDir Function

Changes the current directory.

Pascal

```
procedure ChDir(const S: string); overload;
procedure ChDir(P: PChar); overload;
```

C++

```
ChDir(const UnicodeString S);
ChDir(const char * P);
```

File

System

Description

Note: ChDir changes the current directory to the path specified by S or P. If this operation fails, EInOutError is raised.

Note: On Windows, the path can include a drive specifier, which causes the current drive to be changed as well.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for an I/O error.

C++ Examples:

```
/*
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
*/
void __fastcall TMyChDir::Button1Click(TObject *Sender)
{
  try
  {
    System::ChDir(Edit1->Text);
  }
  catch (...)
  {
    MessageDlg("Cannot find directory", mtWarning, TMsgDlgButtons() << mbOK, 0);
  }
  Edit2->Text = GetCurrentDir();
  FileListBox1->Directory = GetCurrentDir();
  DirectoryListBox1->Directory = GetCurrentDir();
}

void __fastcall TMyChDir::DirectoryListBox1Change(TObject *Sender)
{
```

```
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}

void __fastcall TMyChDir::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}
```

Delphi Examples:

```
{
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  System.ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot find directory', mtWarning, [mbOk], 0);
  Edit2.Text := GetCurrentDir;
  FileListBox1.Directory := GetCurrentDir;
  DirectoryListBox1.Directory := GetCurrentDir;
end;

procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;
```

See Also

GetDir (see page 3580)

MkDir (see page 3612)

RmDir (see page 3647)

SetCurrentDir (see page 4240)

EInOutError (see page 3929)

35.3.18 System::CheckSafecallResult Function

Pascal

```
procedure CheckSafecallResult(hr: HRESULT);
```

C++

```
__fastcall void CheckSafecallResult(HRESULT hr);
```

File

sysclass.h

Description

This is function System::CheckSafecallResult.

35.3.19 System.Chr Function

Returns the character for a specified ASCII value.

Pascal

```
function Chr(X: Byte): Char;
```

C++

```
Char Chr(Byte X);
```

File

System

Description

Chr returns the character with the ordinal value (ASCII value) of the byte-type expression, X.

Delphi Examples:

```
{
This example requires: TRichEdit, TRadioGroup, TCheckBox,
TButton. Clicking the Radio Group control sets the ScrollBar
style of the RichEdit. Toggling the Check Box turns
HideScrollBars on and off.  Clicking the button loads the
Rich Edit with a large amount of contiguous text.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  RichEdit1.Lines.Clear;
  for I := 0 to 500 do
  begin
    RichEdit1.Text :=
      RichEdit1.Text + 'test line ' + IntToStr(I) + ' ';
    if (i mod 4 = 0) then
      RichEdit1.Text := RichEdit1.Text + Chr(13);
  end;
  // move text cursor to top of editend;
  RichEdit1.SelStart := 0;
end;
```

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  RichEdit1.HideScrollbars := CheckBox1.Checked;
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
const
  ScrollBarA: array[0..3] of TScrollStyle = (
    ssBoth,
    ssHorizontal,
    ssNone,
    ssVertical);
begin
  RichEdit1.ScrollBars :=
    ScrollBarA[RadioGroup1.ItemIndex];
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with RadioGroup1, Items do
  begin
    Caption := 'Scroll Bars';
    Clear;
    Add('ssBoth');
    Add('ssHorizontal');
    Add('ssNone');
    Add('ssVertical');
  end;
  with CheckBox1 do
  begin
    Caption := 'HideScrollBars';
    Checked := True;
  end;
  Button1.Caption := 'Loaded';
  RichEdit1.Text := 'Just a little text.';
  RichEdit1.WordWrap := False;
end;
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
```

```
       TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
      else {Key in ['A'..'Z', etc]}
       TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
      if TmpStr = '' then Exit;
      { update SelSt to the current insertion point }
      if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
      else if Key <> Chr(vk_Back) then Inc(SelSt);
      Key := #0; { indicate that key was handled }
      if SelSt = 0 then
      begin
        Text:= '';
        Exit;
      end;
     {Now that TmpStr is the currently typed string, see if we can locate a match }
      Found := False;
      for i := 1 to ct do
      begin
        if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
        begin
          Text := Items[i-1]; { update to the match that was found }
          ItemIndex := i-1;
          Found := True;
          Break;
        end;
      end;
      if Found then { select the untyped end of the string }
      begin
        SelStart := SelSt;
        SelLength := Length(Text) - SelSt;
      end
      else Beep;
    end;
  end;

  var
    CharSetList : TStringList;

  procedure TForm1.AddCharacterSet(const S: string);
  begin
    CharSetList.Add(S);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    CharSetList := TStringList.Create;
    Graphics.GetCharsetValues(AddCharacterSet);
    ComboBox1.Items := CharSetList;
  end;
```

See Also

Ord (see page 3623)

35.3.20 System.Close Function

Terminates the association between a file variable and an external file.

Pascal

```
procedure Close(var F);
```

C++

```
Close( F);
```

File

System

Description

Close is provided for backward compatibility with existing code. When writing new applications in Delphi, use CloseFile (see page 3539) instead.

F is a file variable of any file type opened using Reset (see page 3643), Rewrite (see page 3645), or Append (see page 3520). The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      // Use CloseFile rather than Close; Close provided for backward compatibility
      CloseFile(FromF);
      CloseFile(ToF);
      Canvas.TextOut(120, 10, ' done.');
    end;
  end;
end;
```

See Also

Append (see page 3520)

AssignFile (see page 3525)

Reset (see page 3643)

Rewrite (see page 3645)

CloseFile (see page 3539)

35.3.21 System.CloseFile Function

Terminates the association between file variable and an external disk file.

Pascal

```
procedure CloseFile(var F);
```

C++

```
CloseFile( F);
```

File

System

Description

Due to naming conflicts, CloseFile replaces the Close (see page 3538) procedure. Use the CloseFile procedure instead of Close (see page 3538) to terminate the association between a file variable and an external disk file.

F is a file variable of any file type opened using Reset (see page 3643), Rewrite (see page 3645), or Append (see page 3520). The external file associated with F is completely updated and then closed, freeing the file handle for reuse.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      // Use CloseFile rather than Close; Close provided for backward compatibility
      CloseFile(FromF);
      CloseFile(ToF);
      Canvas.TextOut(120, 10, ' done.');
    end;
  end;
end;
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
```

```
    if OpenDialog1.Execute then begin
      AssignFile(F1, OpenDialog1.Filename);
      Reset(F1);
      if SaveDialog1.Execute then begin
        AssignFile(F2, SaveDialog1.Filename);
        Rewrite(F2);
        while not Eof(F1) do
        begin
          Read(F1, Ch);
          Write(F2, Ch);
        end;
        CloseFile(F2);
      end;
      CloseFile(F1);
    end;
  end;
  {
  Use a TOpenDialog to select a file.  Place the first line of
  the file into a TEdit.  Click on the text edit to open the
  open dialog box.
  }
  procedure TForm1.Edit1Click(Sender: TObject);
  var
    F: TextFile;
    S: string;
  begin
    if OpenDialog1.Execute then            { Display Open dialog box }
    begin
      AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
      Reset(F);
      Readln(F, S);                        { Read first line of file }
      Edit1.Text := S;                     { Put string in a TEdit control }
      CloseFile(F);
    end;
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

AssignFile (see page 3525)

35.3.22 System.CompToCurrency Function

Converts a **Comp** value to a Currency (see page 3235) value.

Pascal

```
function CompToCurrency(Value: Comp): Currency; cdecl;
```

C++

```
__cdecl Currency CompToCurrency(Comp Value);
```

File

System

Description

CompToCurrency converts a Comp value into a Currency (see page 3235) value containing the decimal representation of that number.

C++ Examples:

```
/*
This example demonstrates Comp conversion functions.
*/

void __fastcall TForm2::Button1Click(TObject *Sender)
{
    Comp CompValue;
    Currency CurrencyValue;

    try {
        CompValue = StrToInt64(Edit1->Text);
        CurrencyValue = CompToCurrency(CompValue);
        Edit2->Text = CurrToStr(CurrencyValue);
        }
    catch (...) {
        MessageDlg("Enter a signed integer with abs value <= 9 * 10**18", mtError,
            TMsgDlgButtons() << mbOK, 0);
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm2::Button2Click(TObject *Sender)
{
    Comp CompValue;
    Double DoubleValue;

    try {
        CompValue = StrToInt64(Edit3->Text);
        DoubleValue = CompToDouble(CompValue);
        Edit4->Text = FloatToStr(DoubleValue);
        }
    catch (...) {
        MessageDlg("Enter a signed integer with abs value <= 9 * 10**18", mtError,
            TMsgDlgButtons() << mbOK, 0);
    }
}
```

Delphi Examples:

```
{
This example demonstrates Comp conversion functions.
```

```
}

procedure TForm2.Button1Click(Sender: TObject);
var
  CompValue: Comp;
  CurrencyValue: Currency;
begin
  try
    CompValue := StrToInt64(Edit1.Text);
    CurrencyValue := CompToCurrency(CompValue);
    Edit2.Text := CurrToStr(CurrencyValue);
  except
    MessageDlg('Enter a signed integer with abs value <= 9 * 10**18', mtError, [mbOK], 0);
  end;
end;

procedure TForm2.Button2Click(Sender: TObject);
var
  CompValue: Comp;
  DoubleValue: Double;
begin
  try
    CompValue := StrToInt64(Edit3.Text);
    DoubleValue := CompToDouble(CompValue);
    Edit4.Text := FloatToStr(DoubleValue);
  except
    MessageDlg('Enter a signed integer with abs value <= 9 * 10**18', mtError, [mbOK], 0);
  end;
end;
```

See Also

CompToDouble (see page 3543)

CurrencyToComp (see page 3550)

DoubleToComp (see page 3555)

35.3.23 System.CompToDouble Function

Converts a Comp value to a double value.

Pascal

```
function CompToDouble(Value: Comp): Double; cdecl;
```

C++

```
__cdecl Double CompToDouble(Comp Value);
```

File

System

Description

CompToDouble converts a Comp value into a double value containing the decimal representation of that number.

C++ Examples:

```
/*
This example demonstrates Comp conversion functions.
*/

void __fastcall TForm2::Button1Click(TObject *Sender)
```

```
{
    Comp CompValue;
    Currency CurrencyValue;

    try {
        CompValue = StrToInt64(Edit1->Text);
        CurrencyValue = CompToCurrency(CompValue);
        Edit2->Text = CurrToStr(CurrencyValue);
        }
    catch (...) {
        MessageDlg("Enter a signed integer with abs value <= 9 * 10**18", mtError,
            TMsgDlgButtons() << mbOK, 0);
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm2::Button2Click(TObject *Sender)
{
    Comp CompValue;
    Double DoubleValue;

    try {
        CompValue = StrToInt64(Edit3->Text);
        DoubleValue = CompToDouble(CompValue);
        Edit4->Text = FloatToStr(DoubleValue);
        }
    catch (...) {
        MessageDlg("Enter a signed integer with abs value <= 9 * 10**18", mtError,
            TMsgDlgButtons() << mbOK, 0);
    }
}
```

Delphi Examples:

```
{
This example demonstrates Comp conversion functions.
}

procedure TForm2.Button1Click(Sender: TObject);
var
  CompValue: Comp;
  CurrencyValue: Currency;
begin
  try
    CompValue := StrToInt64(Edit1.Text);
    CurrencyValue := CompToCurrency(CompValue);
    Edit2.Text := CurrToStr(CurrencyValue);
  except
    MessageDlg('Enter a signed integer with abs value <= 9 * 10**18', mtError, [mbOK], 0);
  end;
end;

procedure TForm2.Button2Click(Sender: TObject);
var
  CompValue: Comp;
  DoubleValue: Double;
begin
  try
    CompValue := StrToInt64(Edit3.Text);
    DoubleValue := CompToDouble(CompValue);
    Edit4.Text := FloatToStr(DoubleValue);
  except
    MessageDlg('Enter a signed integer with abs value <= 9 * 10**18', mtError, [mbOK], 0);
  end;
end;
```

See Also

CompToCurrency (see page 3542)

CurrencyToComp (see page 3550)

DoubleToComp (see page 3555)

35.3.24 System.Concat Function

Concatenates two or more strings into one.

Pascal

```
function Concat(s1: string): string; overload;
function Concat(s1: string; s2: string): string; overload;
function Concat(s1: string; s2: string; s3: string): string; overload;
```

C++

```
UnicodeString Concat(UnicodeString s1);
UnicodeString Concat(UnicodeString s1, UnicodeString s2);
UnicodeString Concat(UnicodeString s1, UnicodeString s2, UnicodeString s3);
```

File

System

Description

In Delphi code, use Concat to concatenate an arbitrary number of strings. Each parameter is a string-type expression. The result is the concatenation of all the string parameters.

The syntax shown here for Concat demonstrates that the procedure can take a variable number of arguments.

Using the plus (+) operator has the same effect on two strings as using the Concat function:

S := 'ABC' + 'DEF';

Tip: The plus operator is faster than Concat.

Delphi Examples:

```
{
This example requires a button and three text fields.
Select the button to concatenate the two texts into
the third.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  Edit3.Text := Concat(Edit1.Text, Edit2.Text);     { 'ABCDEF' }
end;
```

See Also

Copy (see page 3547)

Delete (see page 3552)

Insert (see page 3601)

Length (see page 3607)

Pos (see page 3627)

35.3.25 System.Continue Function

Allows the flow of control to proceed to the next iteration of for, while, or repeat statements.

Pascal

```
procedure Continue;
```

C++

```
Continue();
```

File

System

Description

In Delphi code, the Continue procedure causes the flow of control to proceed to the next iteration of the enclosing for, while, or repeat statement.

The compiler reports an error if a call to Continue isn't enclosed by a for, while, or repeat statement.

Note: Continue does not violate the flow of control dictated by a try..finally construct. If a call to Continue causes control to leave the try clause, the finally clause is entered.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
```

```
        else
          FileClose(filehandle);

        AssignFile(F, FileListBox1.Items.Strings[i]);
        Reset(F, 1);
        ListBox1.Items.Add(
          FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
        CloseFile(F);
      end;
    finally
    { do something here }
    end;
  end;
end;
```

See Also

Break (see page 3531)

Exit (see page 3565)

Halt (see page 3594)

Abort (see page 4047)

35.3.26 System.Copy Function

Returns a substring of a string or a segment of a dynamic array.

Pascal

```
function Copy(S; Index: Integer; Count: Integer): array; overload;
```

C++

```
array Copy( S, int Index, int Count);
```

File

System

Description

S is an expression of a string or dynamic-array type. Index and Count are integer-type expressions. Copy returns a substring or subarray containing Count characters or elements starting at S[Index]. The substring or subarray is a unique copy (that is, it does not share memory with S, although if the elements of the array are pointers or objects, these are not copied as well.)

If Index is larger than the length of S, Copy returns an empty string or array.

If Count specifies more characters or array elements than are available, only the characters or elements from S[Index] to the end of S are returned.

Note: When S is a dynamic array, you can omit the Index and Count parameters and Copy copies the entire array.

Delphi Examples:

```
{
The following example shows how to complete partial strings
typed into a combo box. The code represents the OnKeyPress
event handler of the combo box, which performs most of the
default keystroke handling before finding a matching list
item and updating the text.
```

```
Note:   This OnKeyPress event handler does not deal with the
case when the user types the Delete key. That case must be
caught in the OnKeyDown event handler instead.
Note: Set the TComboBox AutoComplete to false to turn off
default auto completion.
}
procedure TForm1.ComboBox1KeyPress(Sender: TObject; var Key: Char);
var
  Found: boolean;
  i, ct, SelSt: Integer;
  TmpStr: string;
begin
  { first, process the keystroke to obtain the current string }
  { This code requires all items in list to be uppercase}
  if Key in ['a'..'z'] then Dec(Key,32); {Force Uppercase only!}
  ct := (Sender as TComboBox).Items.Count;
  with (Sender as TComboBox) do
  begin
    SelSt := SelStart;
    if (Key = Chr(vk_Back)) and (SelLength <> 0) then
     TmpStr := Copy(Text, 1, SelStart) + Copy(Text, SelLength + SelStart + 1, 255)
    else if Key = Chr(vk_Back) then {SelLength = 0}
     TmpStr := Copy(Text, 1, SelStart-1) + Copy(Text, SelStart + 1,255)
    else {Key in ['A'..'Z', etc]}
     TmpStr := Copy(Text, 1, SelStart) + Key + Copy(Text, SelLength + SelStart + 1, 255);
    if TmpStr = '' then Exit;
    { update SelSt to the current insertion point }
    if (Key = Chr(vk_Back)) and (SelSt > 0) then Dec(SelSt)
    else if Key <> Chr(vk_Back) then Inc(SelSt);
    Key := #0; { indicate that key was handled }
    if SelSt = 0 then
    begin
      Text:= '';
      Exit;
    end;
   {Now that TmpStr is the currently typed string, see if we can locate a match }
    Found := False;
    for i := 1 to ct do
    begin
      if Copy(Items[i - 1], 1, Length(TmpStr)) = TmpStr then
      begin
        Text := Items[i-1]; { update to the match that was found }
        ItemIndex := i-1;
        Found := True;
        Break;
      end;
    end;
    if Found then { select the untyped end of the string }
    begin
      SelStart := SelSt;
      SelLength := Length(Text) - SelSt;
    end
    else Beep;
  end;
end;

var
  CharSetList : TStringList;

procedure TForm1.AddCharacterSet(const S: string);
begin
  CharSetList.Add(S);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  CharSetList := TStringList.Create;
  Graphics.GetCharsetValues(AddCharacterSet);
  ComboBox1.Items := CharSetList;
end;
```

See Also

Concat (see page 3545)

Delete (see page 3552)

Insert (see page 3601)

Length (see page 3607)

Pos (see page 3627)

Slice (see page 3671)

LeftStr (see page 3040)

RightStr (see page 3047)

Parameters

35.3.27 System.Cos Function

Calculates the cosine of an angle.

Pascal

```
function Cos(const X: Extended): Extended;
```

C++

```
Extended Cos(const Extended X);
```

File

System

Description

In Delphi code, Cos returns the cosine of the angle X.

X is a real-type expression that represents an angle in radians.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
```

```
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
```

See Also

ArcCos

Sin (see page 3669)

Tan

Cosh

35.3.28 System.CurrencyToComp Function

Converts a Currency (see page 3235) value to a Comp value.

Pascal

```
procedure CurrencyToComp(Value: Currency; var Result: Comp); cdecl;
```

C++

```
__cdecl CurrencyToComp(Currency Value, Comp Result);
```

File

System

Description

CurrencyToComp converts a Currency (see page 3235) value into a Comp.

See Also

CompToDouble (see page 3543)

CompToCurrency (see page 3542)

DoubleToComp (see page 3555)

35.3.29 System.Dec Function

Decrements a variable by 1 or N.

Pascal

```
procedure Dec(var X); overload;
procedure Dec(var X; N: Longint); overload;
```

C++

```
Dec( X);
Dec( X, Longint N);
```

File

System

Description

In Delphi code, Dec subtracts one or N from a variable.

X is a variable of an ordinal type (including Int64), or a pointer type if the extended syntax is enabled.

N is an integer-type expression.

X decrements by 1, or by N if N is specified; that is, Dec(X) corresponds to the statement X := X - 1, and Dec(X, N) corresponds to the statement X := X - N. On some platforms, Dec may generate optimized code, especially useful in tight loops.

If X is a pointer type, it decrements X by N times the size of the type pointed to. Thus, given

```
type
  PMyType = ^TMyType;
```

and

```
var
  P: PMyType;
```

the statement `Dec(P)` decrements `P` by `SizeOf (` see page 3670`)(TMyType)`.

Warning: Do not use Dec on ordinal-type properties if the property uses a write procedure.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
```

See Also

Inc (see page 3597)

Pred (see page 3628)

Succ (see page 3680)

35.3.30 System.Delete Function

Removes a substring from a string.

Pascal

```
procedure Delete(var S: string; Index: Integer; Count: Integer);
```

C++

```
Delete(UnicodeString S, int Index, int Count);
```

File

System

Description

In Delphi code, Delete removes a substring of Count characters from string S starting with S[Index]. S is a string-type variable. Index and Count are integer-type expressions.

If index is larger than the length of the string or less than 1, no characters are deleted.

If count specifies more characters than remain starting at the index, Delete removes the rest of the string. If count is less than or equal to 0, no characters are deleted.

Delphi Examples:

```
{
This example uses the Delete function to edit a string.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  s: string;
begin
  s := 'Honest Abe Lincoln';
  Canvas.TextOut(10, 10, s);
  Delete(s,8,4);
  Canvas.TextOut(10, 40, s);    { 'Honest Lincoln' }
end;
```

See Also

Concat (see page 3545)

Copy (see page 3547)

Insert (see page 3601)

Length (see page 3607)

Pos (see page 3627)

35.3.31 System.Dispose Function

Releases memory allocated for a dynamic variable.

Pascal

```
procedure Dispose(var P: Pointer);
```

C++

```
Dispose(void * P);
```

File

System

Description

Use Dispose in Delphi code to free the memory which a pointer addresses. After a call to Dispose, the value of P is undefined and it is an error to reference P.

Note: If compiling with {$I+}, use exceptions to handle this error. When an invalid pointer is passed to Dispose it raises an EInvalidPointer exception.

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
{
The following code adds an object to MyList if it isn't
already in the list.
}
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;
var
  Form1: TForm1;
  MyList: TList;
  ARecord1, ARecord2: PMyList;

implementation

{$R *.dfm}

procedure DisplayTList(TheList: TList);
var
```

```
    ARecord: PMyList;
    B: Byte;
    Y: Word;
  begin
      { Now paint the items onto the paintbox}
      Y := 10;             {Variable used in TextOut function}
      for B := 0 to (TheList.Count - 1) do
      begin
        ARecord := TheList.Items[B];
        Form1.PaintBox1.Canvas.TextOut(10, Y, IntToStr(ARecord^.I)); {Display I}
        Y := Y + 30;  {Increment Y Value again}
        Form1.PaintBox1.Canvas.TextOut(10, Y, ARecord^.C);  {Display C}
        Y := Y + 30;  {Increment Y Value}
      end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    if (MyList.IndexOf(ARecord1) = -1) then MyList.Add(ARecord1);
    DisplayTList(MyList);
  end;

  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (MyList.IndexOf(ARecord2) = -1) then MyList.Add(ARecord2);
    DisplayTList(MyList);
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  var
    ARecord: PMyList;
    B: Integer;
  begin
    if (MyList.Count <> 0) then
    begin
      for B := (MyList.Count - 1) downto 0 do
      begin
        ARecord := MyList.Items[B];
        MyList.Remove(ARecord);
      end;
    end;
    Form1.PaintBox1.Repaint;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    MyList := TList.Create;
    New(ARecord1);
    ARecord1^.I := 100;
    ARecord1^.C := 'Z';
    New(ARecord2);
    ARecord2^.I := 200;
    ARecord2^.C := 'X';
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    Dispose(ARecord1);
    Dispose(ARecord2);
    MyList.Free;
  end;
```

See Also

EInvalidPointer (see page 3937)

FreeMem (see page 3577)

GetMem (see page 3584)

New (see page 3619)

35.3.32 System.DoubleToComp Function

Converts a double value to a Comp.

Pascal

```
procedure DoubleToComp(Value: Double; var Result: Comp); cdecl;
```

C++

```
__cdecl DoubleToComp(Double Value, Comp Result);
```

File

System

Description

DoubleToComp converts a Double value into a Comp.

See Also

CompToDouble (see page 3543)

CompToCurrency (see page 3542)

CurrencyToComp (see page 3550)

35.3.33 System.DynArrayClear Function

Pascal

```
procedure DynArrayClear(var a: Pointer; typeInfo: Pointer);
```

C++

```
DynArrayClear(void * a, void * typeInfo);
```

File

System

Description

This is function System.DynArrayClear.

35.3.34 System.DynArrayDim Function

Pascal

```
function DynArrayDim(typeInfo: PDynArrayTypeInfo): Integer;
```

C++

```
int DynArrayDim(PDynArrayTypeInfo typeInfo);
```

File

System

Description

This is function System.DynArrayDim.

35.3.35 System.DynArraySetLength Function

Pascal

```
procedure DynArraySetLength(var a: Pointer; typeInfo: Pointer; dimCnt: Longint; lengthVec:
PLongint);
```

C++

```
DynArraySetLength(void * a, void * typeInfo, Longint dimCnt, PLongint lengthVec);
```

File

System

Description

This is function System.DynArraySetLength.

35.3.36 System.EndThread Function

Terminates the execution of a thread.

Pascal

```
procedure EndThread(ExitCode: Integer);
```

C++

```
EndThread(int ExitCode);
```

File

System

Description

EndThread terminates execution of a thread with the specified ExitCode (see page 3823).

Note: On Windows, EndThread calls the ExitThread WinAPI with the specified ExitCode (see page 3823).

See Also

BeginThread (see page 3527)

System::TThreadFunc (see page 3797)

35.3.37 System.EnumModules Function

Executes a callback for all the modules in an application.

Pascal

```
procedure EnumModules(Func: TEnumModuleFunc; Data: Pointer); overload;
procedure EnumModules(Func: TEnumModuleFuncLW; Data: Pointer); overload;
```

C++

```
EnumModules(TEnumModuleFunc Func, void * Data);
EnumModules(TEnumModuleFuncLW Func, void * Data);
```

File

System

Description

EnumModules enumerates the executable and all packages in an application by passing the handle of the executable and then of each package, in turn, to a user-defined callback function. EnumModules continues until the last module in the application is enumerated, or until the callback function returns false. The parameters to EnumModules areas follows:

Parameter	Meaning
Func	The procedure-instance address of the callback function.
Data	A 32-bit user-defined value that is passed to the callback function.

See Also

EnumResourceModules (see page 3557)

LibModuleList (see page 3830)

35.3.38 System.EnumResourceModules Function

Executes a callback for all resource modules in an application.

Pascal

```
procedure EnumResourceModules(Func: TEnumModuleFunc; Data: Pointer); overload;
procedure EnumResourceModules(Func: TEnumModuleFuncLW; Data: Pointer); overload;
```

C++

```
EnumResourceModules(TEnumModuleFunc Func, void * Data);
EnumResourceModules(TEnumModuleFuncLW Func, void * Data);
```

File

System

Description

EnumResourceModules executes the callback specified by the Func parameter for all of the instance handles associated with resources of the current program. The instance handle (HInstance) of each resource is passed to Func. EnumResourceModules continues until the last resource module in the application is enumerated, or until the callback returns false. The parameters to EnumResourceModules are as follows:

Parameter	Meaning
Func	The procedure-instance address of the callback function.
Data	A 32-bit user-defined value that is passed to the callback function.

See Also

EnumModules (see page 3556)

LibModuleList (see page 3830)

35.3.39 System.Eof Function

Tests whether the file position is at the end of a file.

Pascal

```
function Eof: Boolean; overload;
function Eof(var F): Boolean; overload;
function Eof(var F: Text): Boolean; overload;
```

C++

```
Boolean Eof();
Boolean Eof( F);
Boolean Eof(Text F);
```

File

System

Description

In Delphi code, Eof tests whether the current file position is the end-of-file. F is a file variable that has been opened for reading. If F is omitted, the standard file variable Input (see page 3827) is assumed.

Eof(F) returns true if the current file position is beyond the last character of the file or if the file is empty; otherwise, Eof(F) returns false.

Note: Eof fails if the file F has been opened in write-only mode. For example, you can't use Eof with files opened using the Append (see page 3520) or Rewrite (see page 3645), which open a file in write-only mode.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
    end;
```

```
    CloseFile(F1);
  end;
end;
```

See Also

DeleteFile (see page 4098)

Eoln (see page 3559)

SeekEof (see page 3651)

SetLineBreakStyle (see page 3659)

35.3.40 System.Eoln Function

Tests whether the file pointer is at the end of a line.

Pascal

```
function Eoln: Boolean; overload;
function Eoln(var F: Text): Boolean; overload;
```

C++

```
Boolean Eoln();
Boolean Eoln(Text F);
```

File

System

Description

In Delphi code, Eoln tests whether the current file position is the end-of-line of a text file. F, if specified, is a text file variable. If F is omitted, the standard file variable Input (see page 3827) is assumed.

Eoln(F) returns true if the current file position is at an end-of-line or if Eof (see page 3558)(F) is true; otherwise, Eoln(F) returns false.

Delphi Examples:

```
{
This example waits for keyboard input before continuing.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   WriteLn('Type something on the console and hit return');
   Eoln;
   WriteLn('Write this after you type something');
   Readln;
end;
```

See Also

Eof (see page 3558)

SeekEoln (see page 3652)

SetLineBreakStyle (see page 3659)

35.3.41 System.Erase Function

Deletes an external file.

Pascal

```
procedure Erase(var F);
```

C++

```
Erase( F);
```

File

System

Description

In Delphi code, Erase deletes the external file associated with F. F is a file variable of any file type.

Always close a file before erasing it.

Delphi Examples:

```
{
Click the button to open a TOpenDialog, then select a file
to delete.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    try
      Reset(F);
      if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
        mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        CloseFile(F);
        Erase(F);
      end;
    except
      on EInOutError do
        MessageDlg('File I/O error.', mtError, [mbOk], 0);
    end;
  end;
end;
```

See Also

Rename (see page 3642)

35.3.42 System.Error Function

Error is used to generate a run-time exception.

Pascal

```
procedure Error(errorCode: TRuntimeError);
```

C++

```
Error(TRuntimeError errorCode);
```

File

System

Description

Error should be invoked when you want to generate an exception. **errorCode** defines the type of error to be reported.

Delphi Examples:

```
{
This example demostrates the use of ErrorProc and how to catch runtime
errors.
}
procedure MyErrorProc(ErrorCode : Byte; ErrorPtr : Pointer);
var
  ErrorName : String;
begin
  { Get the "name" of the enum value using RTTI }
  ErrorName := GetEnumName(TypeInfo(TRuntimeError), ErrorCode);

  { Show an error box }
  MessageDlg(Format('Error with code %d (%s) risen at $%X.',
      [ErrorCode, ErrorName, Integer(ErrorPtr)]), mtError, [mbOK], 0);
end;

procedure TForm2.btIOErrorClick(Sender: TObject);
begin
  {
  Try to write something onto the console - will raise an
  exception.
  }
  WriteLn('This will generate an error because there is no' +
          ' console attached!');
end;

{$OVERFLOWCHECKS ON}
{$OPTIMIZATION OFF}
{$HINTS OFF}
procedure TForm2.btOverflowErrClick(Sender: TObject);
var
  b : Cardinal;
begin
  {
  Simulate an overflow: Note, enabled the overflow
  checking and disabled optimizations because Delphi's
  compiler will not compile this code otherwise.
  }
  b := $FFFFFFFF;
  b := b * b;
end;
{$HINTS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}

procedure TForm2.btRuntimeErrorClick(Sender: TObject);
begin
  { Simulate an access violation }
  System.Error(reAccessViolation);
end;
```

```
procedure TForm2.FormCreate(Sender: TObject);
begin
  { Register my System error proc }
  ErrorProc := MyErrorProc;
end;
```

See Also

TRuntimeError (see page 3734)

35.3.43 System.ExceptAddr Function

Returns the address at which the current exception was raised.

Pascal

```
function ExceptAddr: Pointer;
```

C++

```
void * ExceptAddr();
```

File

System

Description

Use ExceptAddr to obtain the address at which an exception was raised. If there is no current exception, ExceptAddr returns nil (Delphi) or NULL (C++).

See Also

ExceptionErrorMessage (see page 4107)

ExceptObject (see page 3562)

35.3.44 System.ExceptObject Function

Returns current exception object.

Pascal

```
function ExceptObject: TObject;
```

C++

```
TObject * ExceptObject();
```

File

System

Description

ExceptObject returns the current exception object. This is the Exception object that represents the exception currently being handled. If no exception is currently being handled, ExceptObject returns nil (Delphi) or NULL (C++).

ExceptObject is useful when the exception variable (declared in the Delphi on block or C++ catch statement) is not accessible. This can occur when the exception handler calls a procedure, putting the exception variable is out of scope. This is also true in the Delphi else exception handler, which cannot declare an exception variable.

Note: The object returned by ExceptObject may not exist after the exception handler finishes. Thus you cannot use ExceptObject to retain a reference to the exception object, or to re-raise the exception. If you need to do either of these things, call AcquireExceptionObject (see page 3512).

Delphi Examples:

```
{
This example demonstrates exception handling to catch runtime
errors.
}

procedure TForm2.btIOErrorClick(Sender: TObject);
var
  ExceptionObj : TObject;
begin
  {
  Try to write something onto the console - will raise an
  exception.
  }
  try
  WriteLn('This will generate an error because there is no' +
          ' console attached!');
  except
    ExceptionObj := ExceptObject;
    if ExceptionObj = nil then
      MessageDlg('No exception', mtError, [mbOK], 0)
    else
    begin
      MessageDlg(ExceptionObj.ToString, mtError, [mbOK], 0);
    end;
  end;
end;

{$OVERFLOWCHECKS ON}
{$OPTIMIZATION OFF}
{$HINTS OFF}
procedure TForm2.btOverflowErrClick(Sender: TObject);
var
  b : Cardinal;
  ExceptionPtr : Pointer;
begin
  {
  Simulate an overflow: Note, enabled the overflow
  checking and disabled optimizations because Delphi's
  compiler will not compile this code otherwise.
  }
  ExceptionPtr := nil;
  try
    b := $FFFFFFFF;
    b := b * b;
  except
    ExceptionPtr := AcquireExceptionObject;
  end;

  // Check exception
  if ExceptionPtr = nil then
    MessageDlg('No exception', mtError, [mbOK], 0)
  else
  begin
    MessageDlg(TObject(ExceptionPtr).ToString, mtError, [mbOK], 0);
    ReleaseExceptionObject;
  end;
end;
{$HINTS ON}
```

```
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}

procedure TForm2.btRuntimeErrorClick(Sender: TObject);
var

  ExceptionObj : TObject;
  begin
  { Simulate an access violation }
  try
    System.Error(reAccessViolation);
  except
    ExceptionObj := ExceptObject;
    if ExceptionObj = nil then
      MessageDlg('No exception', mtError, [mbOK], 0)
    else
    begin
      MessageDlg(ExceptionObj.ToString, mtError, [mbOK], 0);
    end;
  end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin

end;
```

See Also

ExceptionErrorMessage (see page 4107)

Exception (see page 3949)

35.3.45 System.Exclude Function

Removes an element from a Delphi set or removes an event handler in .NET applications.

Pascal

```
procedure Exclude(var S: set of T; I: T);
```

C++

```
Exclude(set of T S, T I);
```

File

System

Description

When the first argument of Include (see page 3598) is a set type variable, the Exclude procedure removes an element from the set. If S is a set type variable, and I is an expression of a type compatible with the base type of S. then `Exclude(S,I)` corresponds to `S := S - [I]` but the Exclude procedure generates more efficient code.

On the .NET platform, Exclude is overloaded. When the first argument of Exclude is a multicast event and the second argument is an event handler, Include (see page 3598) removes the event handler from the event's listeners.

Delphi Examples:

```
{
This example demonstrates the use of Include and Exclude routines.
The example assumes a form with three checkboxes and a button. Also
a field called FMsgButtons of type TMsgDlgButtons is declared in the form.
```

```
}
procedure TOptionsForm.btShowMessageClick(Sender: TObject);
begin
  //Creates a Message Dialog with the Selected Buttons
  MessageDlg('Hello World!', mtInformation, FMsgButtons, 0);
end;

procedure TOptionsForm.CheckBoxClick(Sender: TObject);
begin
  { Check wether an option is selected and add an element into a set,
    or exclude it from the set}

  if cbYes.Checked then
    Include(FMsgButtons, mbYes)
  else
    Exclude(FMsgButtons, mbYes);

  if cbNo.Checked then
    Include(FMsgButtons, mbNo)
  else
    Exclude(FMsgButtons, mbNo);

  if cbCancel.Checked then
    Include(FMsgButtons, mbCancel)
  else
    Exclude(FMsgButtons, mbCancel);
end;
```

See Also

Include (see page 3598)

35.3.46 System.Exit Function

Exits from the current procedure.

Pascal

```
procedure Exit;
```

C++

```
Exit();
```

File

System

Description

In Delphi, the Exit procedure immediately passes control away from the current procedure. If the current procedure is the main program, Exit causes the program to terminate.

Exit will cause the calling procedure to continue with the statement after the point at which the procedure was called.

Note: Exit passes control away from the current procedure, not merely the current block. But Exit does not violate the flow of control dictated by a try..finally construct; if Exit is called inside the try clause, the finally clause is still executed.

Beginning in Delphi 2009, Exit can take a parameter specifying a result. The parameter must be of the same type as the result of the function. For example:

```
function DoSomething(aInteger: integer): string
begin
if aInteger < 0 then
```

```
                begin
                   Exit('Negative');
                end;
                Result := 'Positive';
             end;
```

Delphi Examples:

```
{
This example uses a memo and six buttons on a form. This
example illustrates the different ways to exit a function.
}
function GetAString(input: string): string;
var
  mystring : string;
begin
  mystring := DateToStr(Date);
  if (input = 'date') then
    Exit(mystring);
  if (input = 'Tuesday') then
  begin
    Result := 'Tuesday';
    Exit;
  end;
  Result := 'nothing';
end;

function GetANumber(input: string): Integer;
var
  myInt : Integer;
begin
  myInt := Round(Date);
  if (input = 'date') then
    Exit(myInt);
  if (input = 'Tuesday') then
  begin
    Result := 2;
    Exit;
  end;
  Result := 0;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.Add(GetAString('date'));
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Memo1.Lines.Add(GetAString('babble'));
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Memo1.Lines.Add(GetAString('Tuesday'));
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  Memo1.Lines.Add(IntToStr(GetANumber('date')));
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  Memo1.Lines.Add(IntToStr(GetANumber('nothing')));
end;
```

```
procedure TForm1.Button6Click(Sender: TObject);
begin
  Memo1.Lines.Add(IntToStr(GetANumber('Tuesday')));
end;
```

See Also

Break (see page 3531)

Halt (see page 3594)

Continue (see page 3546)

35.3.47 System.Exp Function

Returns the exponential of X.

Pascal

```
function Exp(X: Real): Real;
function Exp(const X: Extended): Extended;
```

C++

```
Real Exp(Real X);
Extended Exp(const Extended X);
```

File

System

Description

In Delphi code, Exp returns the value of e raised to the power of X, where e is the base of the natural logarithms.

Delphi Examples:

```
{
This example calculated the exponential of 1.0 and then calculates
the natural log of that quantity to show that the natural log
function is the inverse of the exponential function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   e : real;
   S : string;
begin
   e := Exp(1.0);
   Str(ln(e):3:2, S);
   S := 'e = ' + FloatToStr(e) + '; ln(e) = ' + S;
   Canvas.TextOut(10, 10, S);
end;
```

See Also

Ln (see page 3608)

35.3.48 System.FilePos Function

Returns the current file position.

Pascal

```
function FilePos(var F): Longint;
```

C++

```
Longint FilePos( F);
```

File

System

Description

In Delphi code, use FilePos on the file variable of an open file to determine the current position in the file. If the current position in the file is at the very beginning, FilePos returns 0. Otherwise, FilePos returns the byte offset from the beginning of the current position.

FilePos will not work on a file that is not open.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example displays a open dialog when the user clicks a
button. When the user selects the file, it reports the size
of the file, seeks to halfway through the file, and reports
the file position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON
($I+). That is, it expects errors to generate exceptions
rather than checking IOResult to be sure that I/O routines
succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      S := 'Seeking halfway into file...';
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      Seek(f, size div 2);
      S := 'Position is now ' + IntToStr(FilePos(f));
      Canvas.TextOut(5, y, S);
    finally
      CloseFile(f);
    end;
  end;
end;
```

See Also

FileSize (see page 3569)

Seek (see page 3650)

35.3.49 System.FileSize Function

Returns the number of records in a file.

Pascal

```
function FileSize(var F): Integer;
```

C++

```
int FileSize( F);
```

File

System

Description

In Delphi code, call FileSize to determine the size of the file specified by the file variable F. The size is expressed as the number of records in a record file. Thus:

If the file is declared as a file of byte, then the record size defaults to one byte, and FileSize returns the number of bytes in the file.

The Reset (see page 3643) procedure can set the record size (in bytes) when it opens the file. In this case, FileSize returns the number of records in the file.

Note: If the file is declared as an untyped file and you don't specify a record size when you call Reset (see page 3643), then FileSize assumes a record size of 128. That is, FileSize gives the number of bytes divided by 128.

To use FileSize, the file must be open. If the file is empty, FileSize(F) returns 0.

Delphi Examples:

```
{
This example displays a open dialog when the user clicks a
button. When the user selects the file, it reports the size
of the file, seeks to halfway through the file, and reports
the file position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON
($I+). That is, it expects errors to generate exceptions
rather than checking IOResult to be sure that I/O routines
succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
```

```
        S := 'Seeking halfway into file...';
        Canvas.TextOut(5, y, S);
        y := y + Canvas.TextHeight(S) + 5;
        Seek(f, size div 2);
        S := 'Position is now ' + IntToStr(FilePos(f));
        Canvas.TextOut(5, y, S);
      finally
        CloseFile(f);
      end;
    end;
  end;
  {
  The following example opens a file of your choice and reads
  the entire file into a dynamically allocated buffer. The
  buffer and the size of the file are then passed to a routine
  that processes the text, and finally the dynamically
  allocated buffer is freed and the file is closed.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: file;
    Size: Integer;
    Buffer: PAnsiChar;
  begin
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      Reset(F, 1);
      try
        Size := FileSize(F);
        GetMem(Buffer, Size);
        try
          BlockRead(F, Buffer^, Size);
          Memo1.Lines.Add(AnsiString(Buffer));
        finally
          FreeMem(Buffer);
        end;
      finally
        CloseFile(F);
      end;
    end;
  end;
```

See Also

FilePos (see page 3567)

Reset (see page 3643)

35.3.50 System.FillChar Function

Fills contiguous bytes with a specified value.

Pascal

```
procedure FillChar(var X; Count: Integer; Value: Byte);
```

C++

```
FillChar( X, int Count, Byte Value);
```

File

System

Description

In Delphi, FillChar fills **Count** contiguous bytes (referenced by **X**) with the value specified by **Value** (**Value** can be type **Byte** or **Char**).

Note that if **X** is a **UnicodeString (see page 3381)**, this may not work as expected because FillChar expects a byte count, which not be the same as the character count.

In addition, the fill character is a single byte character. Therefore, when `Buf` is a **UnicodeString (see page 3381)**, the code `FillChar(Buf, Length (see page 3607)(Buf), #9);` fills `Buf` with the code point $0909, not $09. For such cases, you probably want to use the function StringOfChar (see page 3677).

Warning: This function does not perform any range checking.

Delphi Examples:

```
{
This example creates a rotated font and displays it on the
form by assigning the handle of the rotated to the Form's
Canvas Font via the Font's Handle property.  Rotating fonts
is a straightforward process, so long as the Windows Font
Mapper can supply a rotated font based on the font you
request. If you are using a TrueType font there is virtually
no reason for failure.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  lf: LOGFONT; // Windows native font structure
begin
  Canvas.Brush.Style := bsClear; // set the brush style to transparent
  FillChar(lf, SizeOf(lf), Byte(0));
  lf.lfHeight := 20;
  lf.lfEscapement := 10 * 45; // degrees to rotate
  lf.lfOrientation := 10 * 45;
  lf.lfCharSet := DEFAULT_CHARSET;
  StrCopy(lf.lfFaceName, 'Tahoma');

  Canvas.Font.Handle := CreateFontIndirect(lf);
  Canvas.TextOut(10, 100, 'Rotated text'); // output the rotated font
end;
{
This example fills a char array with characters and then
assigns the array to a text edit.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: array[0..79] of AnsiChar;
begin
  { Set to all Xs }
  FillChar(S, SizeOf(S), Ord('X'));
  Edit1.Text := S;
end;
```

See Also

Move (see page 3617)

StringOfChar (see page 3677)

35.3.51 System.Finalize Function

Uninitializes a dynamically allocated variable.

Pascal

```
procedure Finalize(var V); overload;
procedure Finalize(var V; Count: Integer); overload;
```

C++

```
Finalize( V);
Finalize( V, int Count);
```

File

System

Description

Finalize should be used only in Delphi code where a dynamically allocated variable is deallocated by other means than the Dispose (see page 3553) procedure. Dynamic arrays can never be deallocated using the Dispose (see page 3553) procedure, but can be freed by passing them to Finalize.

For global variables, local variables, objects, and dynamic variables deallocated using Dispose (see page 3553), the compiler generates code that finalizes all long strings, variants, and interfaces contained by the variable when the instance is destroyed.

If a dynamic variable meets the following two conditions, a call to Finalize is required to finalize the variable before it can be deallocated.

The variable is deallocated by other means than the Dispose (see page 3553) standard procedure (for example using FreeMem (see page 3577)).

The variable contains long strings, variants, or interfaces, not all of which are empty or Unassigned.

Finalize simply sets all long strings to empty and all variants and interfaces to Unassigned, thus properly releasing any memory that was referenced by the long strings and variants.

In cases where several variables are deallocated in a contiguous memory block such as a dynamically allocated array of strings, the additional Count parameter can be specified to finalize all variables in one operation.

If the variable specified in a call to Finalize contains no long strings, variants, or interfaces, the compiler eliminates the call and generates no code for it.

Delphi Examples:

```
{
This example demonstrates the use of Initialize and Finalize functions
used to initialize and finalize a RTTI-enabled structure.
}
type
  { Declaring a data type that needs initilization }
  PPerson = ^TPerson;
  TPerson = record
    FFirstName: String;
    FLastName: String;
    FAge: Integer;
  end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Person: PPerson;
```

```
begin
  {
    Allocate a block of memory for the structure
    We're using an "unsafe" function: GetMem
    which doesn't use RTTI to initialize the
    strings inside the structure.
  }
  GetMem(Person, SizeOf(TPerson));

  { Initialize the structure in the Heap - RTTI will be used }
  Initialize(Person^);

  { Assign some data to the structure }
  Person.FFirstName := 'John';
  Person.FLastName := 'Smith';
  Person.FAge := 31;

  { Use RTTI to cleanup the internals of the structure }
  Finalize(Person^);
  FreeMem(Person);
end;
```

See Also

Initialize (see page 3599)

35.3.52 System.FindClassHInstance Function

Returns the instance handle for the module in which a class type is defined.

Pascal

```
function FindClassHInstance(ClassType: TClass): LongWord;
```

C++

```
LongWord FindClassHInstance(TClass ClassType);
```

File

System

Description

FindClassHInstance finds the `HInstance` of the executable or library that "owns", or contains, the runtime type information (RTTI) and the Virtual Method Table (VMT) for the specified class passed in as the Class Type parameter. When reading a component (which is a type), this information is needed in order to be able to load the form file (resources) associated with that component. The component streaming system uses FindClassHInstance for this purpose.

FindClassHInstance returns 0 if the instance is not found. Otherwise it returns the class owner's `HInstance`.

See Also

FindResourceHInstance (see page 3574)

35.3.53 System::FindDynaInst Function

Pascal

```
function FindDynaInst(var vmt; ID: Integer): Pointer;
```

C++

```
__fastcall void * FindDynaInst(void * vmt, int ID);
```

File

sysclass.h

Description

This is function System::FindDynaInst.

35.3.54 System.FindHInstance Function

Returns the instance handle of the module that contains a specified address.

Pascal

```
function FindHInstance(Address: Pointer): LongWord;
```

C++

```
LongWord FindHInstance(void * Address);
```

File

System

Description

Use FindHInstance to obtain the `HInstance` of the module that "owns", or contains, the address specified by the Address parameter. If no `HInstance` is found for that address, FindHInstance returns 0.

See Also

FindClassHInstance (see page 3573)

FindResourceHInstance (see page 3574)

MainInstance (see page 3830)

35.3.55 System.FindResourceHInstance Function

Returns the instance handle of the resource module associated with a specified `HInstance`.

Pascal

```
function FindResourceHInstance(Instance: LongWord): LongWord;
```

C++

```
LongWord FindResourceHInstance(LongWord Instance);
```

File

System

Description

FindResourceHInstance finds and returns the instance handle of the resource module associated with the executable, library, or package identified by the Instance parameter.

FindResourceHInstance returns 0 upon failure.

Note: The Instance parameter must be a valid instance handle.

See Also

FindClassHInstance (see page 3573)

FindHInstance (see page 3574)

35.3.56 System.Flush Function

Empties the buffer of a text file opened for output.

Pascal

```
function Flush(var t: Text): Integer;
```

C++

```
int Flush(Text t);
```

File

System

Description

F is a text file variable.

When a text file is opened for output using Rewrite (see page 3645) or Append (see page 3520), Flush empties the file's buffer. This guarantees that all characters written to the file at that time have actually been written to the external file. Flush has no effect on files opened for input.

Flush returns 0 if the operation succeeded. Otherwise, it returns the error code. When compiled using the {$I-} flag, the IOResult (see page 3602) method returns this value.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
Click the button and select a file to append a string to the bottom of the file.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  f: TextFile;
begin
  if OpenDialog1.Execute then
  begin                    { open a text file }
    AssignFile(f, OpenDialog1.FileName);
    Append(f);
    Writeln(f, 'I am appending some stuff to the end of the file.');
    { insert code here that would require a Flush before closing the file }
    Flush(f);  { ensures that the text was actually written to file }
    CloseFile(f);
    MessageDlg(OpenDialog1.FileName + ' has been altered, please validate.', mtInformation,
[mbOK], 0)
  end;
end;
```

See Also

Append (see page 3520)

Rewrite (see page 3645)

35.3.57 System.FPower10 Function

Pascal

```
procedure FPower10;
```

C++

```
FPower10();
```

File

System

Description

This is function System.FPower10.

35.3.58 System.Frac Function

Returns the fractional part of a real number.

Pascal

```
function Frac(const X: Extended): Extended;
```

C++

```
Extended Frac(const Extended X);
```

File

System

Description

In Delphi code, the Frac function returns the fractional part of the argument X.

X is a real-type expression. The result is the fractional part of X; that is, Frac(X) = X - Int (see page 3602)(X).

Delphi Examples:

```
{
This example illustrates the use of the System Frac function.
The number in the text edit can be a negative or positive
real number.
Notice how the type passed (R) matters.
}
procedure TForm1.Button1Click(Sender: TObject);
var
 R: Extended;
begin
  R := StrToFloat(Edit1.Text);
  ListBox1.Items.Add(FloatToStr(Frac(R)));
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
var
 R: Real;
begin
  R := StrToFloat(Edit1.Text);
  ListBox1.Items.Add(FloatToStr(Frac(R)));
end;
```

See Also

Int (see page 3602)

35.3.59 System.FreeMem Function

FreeMem frees a memory block.

Pascal

```
procedure FreeMem(var P: Pointer); overload;
procedure FreeMem(var P: Pointer; Size: Integer); overload;
```

C++

```
FreeMem(void * P);
FreeMem(void * P, int Size);
```

File

System

Description

GetMem (see page 3584) allocates a block on the heap. To dispose of the buffer, use FreeMem.

Note: FreeMemory (see page 3579) is the C++ compatible version of FreeMem.

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
```

```
        iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
        FileClose(iFileHandle);
        for i := 0 to iBytesRead-1 do
        begin
          StringGrid1.RowCount := StringGrid1.RowCount + 1;
          StringGrid1.Cells[1,i+1] := Buffer[i];
          StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
        end;
      finally
        FreeMem(Buffer);
      end;
    end;
  end;
  {
  The following example opens a file of your choice and reads
  the entire file into a dynamically allocated buffer. The
  buffer and the size of the file are then passed to a routine
  that processes the text, and finally the dynamically
  allocated buffer is freed and the file is closed.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: file;
    Size: Integer;
    Buffer: PAnsiChar;
  begin
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      Reset(F, 1);
      try
        Size := FileSize(F);
        GetMem(Buffer, Size);
        try
          BlockRead(F, Buffer^, Size);
          Memo1.Lines.Add(AnsiString(Buffer));
        finally
          FreeMem(Buffer);
        end;
      finally
        CloseFile(F);
      end;
    end;
  end;
```

See Also

AllocMem (see page 3515)

FreeMemory (see page 3579)

GetMem (see page 3584)

GetMemory (see page 3585)

EOutOfMemory (see page 3942)

ReallocMem (see page 3637)

ReallocMemory (see page 3638)

35.3.60 System.FreeMemory Function

FreeMemory frees a memory block.

Pascal

```
function FreeMemory(P: Pointer): Integer; cdecl;
```

C++

```
__cdecl int FreeMemory(void * P);
```

File

System

Description

GetMemory (see page 3585) allocates a block on the heap. To dispose of the buffer, use FreeMemory.

Note: FreeMemory is the C++ compatible version of FreeMem (see page 3577).

C++ Examples:

```
/*
The following code demostrates the use of memory management functions in C++. Three
edit boxes and a button are expected onthe form.
*/
String __fastcall FastStrCat(String s1, String s2)
{
  /*
   Allocated enough space in FinalStr to copy the contents
   of the initial string + $00 character
  */
  wchar_t* finalStr = (wchar_t *)GetMemory((s1.Length() + 1) * sizeof(wchar_t));

  // Copy the contents of the first string
  MoveChars(s1.data(), finalStr, s1.Length() + 1);

  // And now expand the final string + $00 character
  finalStr = (wchar_t *)ReallocMemory(finalStr, (s1.Length() + s2.Length() + 1) *
sizeof(wchar_t));

  // Copy the contents of the second string
  MoveChars(s2.data(), finalStr + s1.Length(), s2.Length() + 1);

  // Get the result in String
  String result = String(finalStr);
  FreeMemory(finalStr);

  return result;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Concatenate 2 string
  Edit3->Text = FastStrCat(Edit1->Text, Edit2->Text);
}
```

See Also

AllocMem (see page 3515)

FreeMem (see page 3577)

GetMem (see page 3584)

GetMemory (see page 3585)

EOutOfMemory (see page 3942)

ReallocMem (see page 3637)

ReallocMemory (see page 3638)

35.3.61 System.Get8087CW Function

Returns the value of the 8087 code word.

Pascal

```
function Get8087CW: Word;
```

C++

```
Word Get8087CW();
```

File

System

Description

Use Get8087CW to determine the current value of the floating-point unit control word. This control word controls the precision of floating point calculations, the rounding mode, and whether certain floating-point operations trigger exceptions. See Intel's processor documentation for details.

See Also

NoErrMsg (see page 3832)

SetExceptionMask

SetPrecisionMode

SetRoundMode

Set8087CW (see page 3653)

35.3.62 System.GetDir Function

Returns the current directory.

Pascal

```
procedure GetDir(D: Byte; var S: string);
```

C++

```
GetDir(Byte D, UnicodeString S);
```

File

System

Description

Note: On Windows, GetDir sets the S parameter to the current directory on the drive specified by the D parameter. D can be set to any of the following values:

Value	Drive
0	Default
1	A
2	B
3	C

Note: GetDir performs no error checking. For example, if the invalid value 5 (J) were specified, S would return J:\ as if it were the root directory of the invalid drive.

Note: On Linux, GetDir returns the current working directory using the S parameter. The first parameter is ignored.

Delphi Examples:

```
{
This example reports the current directory and drive.  Set
Help on GetDir for reporting information on other drives.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  GetDir(0,s); { 0 = Current drive }
  MessageDlg('Current drive and directory: ' + s, mtInformation, [mbOk] , 0);
end;
```

See Also

ChDir (see page 3534)

MkDir (see page 3612)

RmDir (see page 3647)

DirectoryExists (see page 4100)

CreateDir (see page 4083)

GetCurrentDir (see page 4202)

35.3.63 System.GetHeapStatus Function

Returns the current status of the memory manager.

Pascal

```
function GetHeapStatus: THeapStatus; deprecated; platform;
```

C++

```
THeapStatus GetHeapStatus();
```

File

System

Description

GetHeapStatus returns the current status of the memory manager in a THeapStatus (see page 3720) type. If the ShareMem unit is not included in the current project, GetHeapStatus returns information about the global heap. If the ShareMem unit is included, GetHeapStatus returns information about memory that can be shared between between the current project and other binary modules (such as DLLs) that also make use of the ShareMem unit. GetHeapStatus can only return useful information if the built-in memory manager is used. For example, when using CodeGuard, the memory manager is replaced and GetHeapStatus always returns nil (Delphi) or NULL (C++).

If you want information about the memory manager that is shared between modules, make sure to call the function GetHeapStatus provided in the ShareMem unit. The function GetHeapStatus in the System (see page 3159) unit returns no values for a project that uses shared memory.

Note: Since GetHeapStatus is deprecated, use GetMemoryManagerState (see page 3591) routine instead.

Warning: ShareMem must be the first unit included in the project file to prevent mixing of memory management models.

See Also

GetMemoryManagerState (see page 3591)

35.3.64 System.GetLastError Function

Returns the last error reported by an operating system API call.

Pascal

```
function GetLastError: Integer; stdcall;
```

C++

```
__stdcall int GetLastError();
```

File

System

Description

GetLastError returns the last error reported by an API call into the operating system. Calling this function usually resets the operating system error state.

C++ Examples:

```
//
//This code demonstrates the use of system error codes
//and how to use transform them into C++ exceptions.
//
void __fastcall TForm2::btRaiseLastClick(TObject *Sender)
{
    /* Set the last OS error to a bogus value */
    System::SetLastError(ERROR_ACCESS_DENIED);
```

```
    try
    {
        RaiseLastOSError();
    }
    catch(EOSError* ex)
    {
        MessageDlg(String("Caught an OS error with code: ") +
            IntToStr((int)ex->ErrorCode), mtError,
            TMsgDlgButtons() << mbOK, 0);
    }


    /* Let the Delphi's Exception dialog pop-up */
    RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

    /* Finally set the last error to none */
    System::SetLastError(ERROR_SUCCESS);

    if (GetLastError() != ERROR_SUCCESS)
        MessageDlg(String("Whoops, something went wrong ") +
          " in the mean time!", mtError, TMsgDlgButtons() << mbOK, 0);

    /* No exception should be thrown here */
    RaiseLastOSError();
}
```

Delphi Examples:

```
{
This code demonstrates the use of system error codes
and how to transform them into Delphi exceptions.
}
procedure TForm2.btRaiseLastClick(Sender: TObject);
begin
  { Set the last OS error to a bogus value }
  System.SetLastError(ERROR_ACCESS_DENIED);

  try
    RaiseLastOSError();
  except
    on Ex : EOSError do
       MessageDlg('Caught an OS error with code: ' +
          IntToStr(Ex.ErrorCode), mtError, [mbOK], 0);
  end;

  { Let the Delphi's Exception dialog pop-up }
  RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

  { Finally set the last error to none }
  System.SetLastError(ERROR_SUCCESS);

  if GetLastError() <> ERROR_SUCCESS then
     MessageDlg('Whoops, something went wrong in the ' +
                'mean time!', mtError, [mbOK], 0);

  { No exception should be thrown here }
  RaiseLastOSError();
end;
```

See Also

Win32Check (see page 4318)

35.3.65 System.GetMem Function

GetMem allocates a memory block.

Pascal

```
procedure GetMem(var P: Pointer; Size: Integer);
```

C++

```
GetMem(void * P, int Size);
```

File

System

Description

GetMem allocates a block of the given **Size** on the heap, and returns the address of this memory in parameter **P**. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use FreeMem (see page 3577). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory needs to be zero-initialized, use AllocMem (see page 3515) instead.

Delphi Examples:

```
{
The following example opens a file of your choice and reads
the entire file into a dynamically allocated buffer. The
buffer and the size of the file are then passed to a routine
that processes the text, and finally the dynamically
allocated buffer is freed and the file is closed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: file;
  Size: Integer;
  Buffer: PAnsiChar;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    Reset(F, 1);
    try
      Size := FileSize(F);
      GetMem(Buffer, Size);
      try
        BlockRead(F, Buffer^, Size);
        Memo1.Lines.Add(AnsiString(Buffer));
      finally
        FreeMem(Buffer);
      end;
    finally
      CloseFile(F);
    end;
  end;
end;
```

See Also

AllocMem (see page 3515)

FreeMem (see page 3577)

FreeMemory (see page 3579)

GetMemory (see page 3585)

EOutOfMemory (see page 3942)

ReallocMem (see page 3637)

ReallocMemory (see page 3638)

35.3.66 System.GetMemory Function

GetMemory allocates a memory block.

Pascal

```
function GetMemory(Size: Integer): Pointer; cdecl;
```

C++

```
__cdecl void * GetMemory(int Size);
```

File

System

Description

GetMemory allocates a block of the given **Size** on the heap, and returns the address of this memory. The bytes of the allocated buffer are not set to zero. To dispose of the buffer, use FreeMem (see page 3577). If there isn't enough memory available to allocate the block, an EOutOfMemory exception is raised.

Note: If the memory needs to be zero-initialized, use AllocMem (see page 3515) instead.

Note: GetMemory is the C++ compatible version of GetMem (see page 3584).

C++ Examples:

```
/*
The following code demostrates the use of memory management functions in C++. Three
edit boxes and a button are expected onthe form.
*/
String __fastcall FastStrCat(String s1, String s2)
{
  /*
   Allocated enough space in FinalStr to copy the contents
   of the initial string + $00 character
  */
  wchar_t* finalStr = (wchar_t *)GetMemory((s1.Length() + 1) * sizeof(wchar_t));

  // Copy the contents of the first string
  MoveChars(s1.data(), finalStr, s1.Length() + 1);

  // And now expand the final string + $00 character
  finalStr = (wchar_t *)ReallocMemory(finalStr, (s1.Length() + s2.Length() + 1) *
sizeof(wchar_t));

  // Copy the contents of the second string
```

```
  MoveChars(s2.data(), finalStr + s1.Length(), s2.Length() + 1);

  // Get the result in String
  String result = String(finalStr);
  FreeMemory(finalStr);

  return result;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Concatenate 2 string
  Edit3->Text = FastStrCat(Edit1->Text, Edit2->Text);
}
```

Delphi Examples:

```
{
This example requires a button, a test edit, and a populated
ClientDataSet.  Pipe the ClientDataSet through a DataSource
to a DGGrid or DBNavigator to control the current field.
Cast the data correctly according to the field type when
assigning to the test edit.
}

{$IFNDEF UNICODE}
uses SwSystem;
{$ENDIF}

procedure TForm1.Button1Click(Sender: TObject);
var MyBuffer: Pointer;
begin
{ Retrieve the "raw" data from Field1 }
with CDS.Fields[0] do
  begin
    if not IsBlob then { this does not work for BLOB fields }
    begin
      { Allocate space }
      MyBuffer:= GetMemory(DataSize);
      try
        if not GetData(MyBuffer) then
          MessageDlg(DisplayName + ' is NULL', mtInformation, [mbOK], 0)
        else
          { Do something with the data };
          Edit1.Text:= string(PAnsiChar(MyBuffer)); // for a stringfield
      finally
        { Free the space }
        FreeMem(MyBuffer, DataSize);
      end;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
{$IFDEF UNICODE}
  CDS.LoadFromFile(GetCurrentDir + '\CDS.XML');
{$ELSE}
  CDS.LoadFromFile(gsAppPath + 'CDS.XML');
{$ENDIF}
end;
```

See Also

AllocMem (see page 3515)

FreeMem (see page 3577)

FreeMemory (see page 3579)

GetMem (see page 3584)

EOutOfMemory (see page 3942)

ReallocMem (see page 3637)

ReallocMemory (see page 3638)

35.3.67 System.GetMemoryManager Function

Returns the entry points of the currently installed memory manager.

Pascal

```
procedure GetMemoryManager(var MemMgr: TMemoryManager); overload; deprecated;
procedure GetMemoryManager(var MemMgrEx: TMemoryManagerEx); overload;
```

C++

```
GetMemoryManager(TMemoryManager MemMgr);
GetMemoryManager(TMemoryManagerEx MemMgrEx);
```

File

System

Description

GetMemoryManager returns the entry points of the currently installed memory manager in a TMemoryManager (see page 3724) record.

C++ Examples:

```
/*
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
*/

bool isMyMemMgr;
TMemoryManagerEx oldMemMgr;
volatile long getMemCalls;
volatile long freeMemCalls;
volatile long reallocMemCalls;
volatile long allocMemCalls;

void* __fastcall myGetMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&getMemCalls);

    //route the call
    return oldMemMgr.GetMem(size);
}

int __fastcall myFreeMem(void* p)
{
    //safely increment the counter
    InterlockedIncrement(&freeMemCalls);

    //route the call
    return oldMemMgr.FreeMem(p);
```

```
}

void* __fastcall myReallocMem(void* p, int size)
{
    //safely increment the counter
    InterlockedIncrement(&reallocMemCalls);

    //route the call
    return oldMemMgr.ReallocMem(p, size);
}

void* __fastcall myAllocMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&allocMemCalls);

    //route the call
    return oldMemMgr.AllocMem(size);
}

void __fastcall TForm3::btUseSysMemMgrClick(TObject *Sender)
{
    //switch button states
    btUseSysMemMgr->Enabled = false;
    btUseMyMemMgr->Enabled = true;

    //set the old memory manager back
    SetMemoryManager(oldMemMgr);

    isMyMemMgr = false;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;
}

void __fastcall TForm3::FormDestroy(TObject *Sender)
{
    //set the old memory manager back
    if (isMyMemMgr)
    {
        SetMemoryManager(oldMemMgr);
    }
}

void __fastcall TForm3::Timer1Timer(TObject *Sender)
{
    //Note that IntToStr calls will also Allocate/Free
    //memory so at each timer tick the counts will be +4;

    Form3->edGetMem->Text = IntToStr((int)getMemCalls);
    Form3->edFreeMem->Text = IntToStr((int)freeMemCalls);
    Form3->edReallocMem->Text = IntToStr((int)reallocMemCalls);
    Form3->edAllocMem->Text = IntToStr((int)allocMemCalls);
}

void __fastcall TForm3::btUseMyMemMgrClick(TObject *Sender)
{
    TMemoryManagerEx myMemMgr;

    //switch button states
    btUseSysMemMgr->Enabled = true;
    btUseMyMemMgr->Enabled = false;
```

```
    //get the old memory manager
    GetMemoryManager(oldMemMgr);

    //create out instance
    myMemMgr.GetMem = myGetMem;
    myMemMgr.FreeMem = myFreeMem;
    myMemMgr.AllocMem = myAllocMem;
    myMemMgr.ReallocMem = myReallocMem;

    //use defaults for this - not important
    myMemMgr.RegisterExpectedMemoryLeak = oldMemMgr.RegisterExpectedMemoryLeak;
    myMemMgr.UnregisterExpectedMemoryLeak = oldMemMgr.UnregisterExpectedMemoryLeak;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;

    //install the new memory manager
    SetMemoryManager(myMemMgr);

    isMyMemMgr = true;
}

void __fastcall TForm3::btDoSomethingClick(TObject *Sender)
{
    void *ptr;
    //perform some new and delete
    for (int i=0; i < 1000; ++i)
    {
        ptr = GetMemory(4);
        FreeMemory(ptr);
    }
}
```

Delphi Examples:

```
{
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
}
var
  OldMemMgr       : TMemoryManagerEx;
  GetMemCalls     : Integer;
  FreeMemCalls    : Integer;
  ReallocMemCalls : Integer;
  AllocMemCalls   : Integer;

function MyGetMem(Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.GetMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(GetMemCalls);
end;

function MyFreeMem(P: Pointer): Integer;
begin
  { Route the call }
  Result := OldMemMgr.FreeMem(P);

  { Safely increment the counter }
  InterlockedIncrement(FreeMemCalls);
end;
```

```
function MyReallocMem(P: Pointer; Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.ReallocMem(P, Size);

  { Safely increment the counter }
  InterlockedIncrement(ReallocMemCalls);
end;

function MyAllocMem(Size: Cardinal): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.AllocMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(AllocMemCalls);
end;

procedure TForm2.btPerformSomethingClick(Sender: TObject);
var
  X : Integer;
  I : ^Integer;
begin
  { Perform some allocations and frees }
  for X := 0 to 999 do
  begin
    New(I);
    Dispose(I);
  end;
end;

procedure TForm2.btUseMyMemMgrClick(Sender: TObject);
var
  MyMemMgr : TMemoryManagerEx;
begin
  { Switch button states }
  btUseSystemMemMgr.Enabled := true;
  btUseMyMemMgr.Enabled := false;

  { Get the old memory manager }
  GetMemoryManager(OldMemMgr);

  { Create our instance }
  MyMemMgr.GetMem := MyGetMem;
  MyMemMgr.FreeMem := MyFreeMem;
  MyMemMgr.ReallocMem := MyReallocMem;
  MyMemMgr.AllocMem := MyAllocMem;

  { Use the defaults for this - not important }
  MyMemMgr.RegisterExpectedMemoryLeak := OldMemMgr.RegisterExpectedMemoryLeak;
  MyMemMgr.UnregisterExpectedMemoryLeak := OldMemMgr.UnregisterExpectedMemoryLeak;

  { Clear out the count variables }
  GetMemCalls := 0;
  FreeMemCalls := 0;
  ReallocMemCalls := 0;
  AllocMemCalls := 0;

  { Install the new memory manager }
  SetMemoryManager(MyMemMgr);
end;

procedure TForm2.btUseSystemMemMgrClick(Sender: TObject);
begin
  { Switch button states }
```

```
    btUseSystemMemMgr.Enabled := false;
    btUseMyMemMgr.Enabled := true;

    { Set the old memory manager back! }
    SetMemoryManager(OldMemMgr);

    GetMemCalls := 0;
    FreeMemCalls := 0;
    ReallocMemCalls := 0;
    AllocMemCalls := 0;
  end;

  procedure TForm2.FormDestroy(Sender: TObject);
  begin
    { Set the old memory manager back }
    if IsMemoryManagerSet then
       SetMemoryManager(OldMemMgr);
  end;

  procedure TForm2.Timer1Timer(Sender: TObject);
  begin
    {
    Note that IntToStr calls will also Allocate/Free
    memory so at each timer tick the counts will be increased;
    }
    Form2.edtGetMemCalls.Text := IntToStr(GetMemCalls);
    Form2.edtFreeMemCalls.Text := IntToStr(FreeMemCalls);
    Form2.edtReallocMemCalls.Text := IntToStr(ReallocMemCalls);
    Form2.edtAllocMemCalls.Text := IntToStr(AllocMemCalls);
  end;
```

See Also

SetMemoryManager (see page 3660)

IsMemoryManagerSet (see page 3604)

35.3.68 System.GetMemoryManagerState Function

Fetches state information for the Memory Manager.

Pascal

```
procedure GetMemoryManagerState(var AMemoryManagerState: TMemoryManagerState);
```

C++

```
GetMemoryManagerState(TMemoryManagerState AMemoryManagerState);
```

File

System

Description

GetMemoryManagerState returns a structure containing memory usage statistics for each block type - small, medium and large blocks. In the current implementation, small blocks are blocks that are smaller than roughly 2.5K, large blocks are blocks that are larger than roughly 256K, and medium blocks are the sizes in between. It provides additional detail for the small block type (which is usually the most frequently used block type) down to each individual block size.

The structure returned by this procedure makes a distinction between the reserved address space and the allocated address space for each block type. The allocated address space is that part of the address space that is currently in use by the application, i.e. memory that was allocated through GetMem (see page 3584) and not yet freed. The reserved address space

is the amount of address space that was obtained from the operating system for current and future use by the application, i.e. the allocated address space plus all overhead.

See Also

Memory Management

Configuring the Memory Manager

Monitoring the Memory Manager

35.3.69 System.GetMemoryMap Function

Fetches the memory map.

Pascal

```
procedure GetMemoryMap(var AMemoryMap: TMemoryMap);
```

C++

```
GetMemoryMap(TMemoryMap AMemoryMap);
```

File

System

Description

GetMemoryMap returns a map of the address space of the process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. GetMemoryMap obtains a status value for each of them in the TMemoryMap (see page 3791) array. The sum of the number of blocks with the status of **csUnallocated** multiplied by the chunk size (64K) gives an indication of how much more memory the application can allocate before it will run out of address space.

Status Value	Description
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

Memory Management

Configuring the Memory Manager

Monitoring the Memory Manager

35.3.70 System.GetMinimumBlockAlignment Function

Gets the minimum memory block alignment.

Pascal

```
function GetMinimumBlockAlignment: TMinimumBlockAlignment;
```

C++

```
TMinimumBlockAlignment GetMinimumBlockAlignment();
```

File

System

Description

GetMinimumBlockAlignment fetches the Memory Manager's minimum block alignment.

See Also

Memory Management

Configuring the Memory Manager

35.3.71 System.GetModuleFileName Function

Returns the fully qualified name for a module, given its handle.

Pascal

```
function GetModuleFileName(Module: HMODULE; Buffer: PChar; BufLen: Integer): Integer;
```

C++

```
int GetModuleFileName(HMODULE Module, const char * Buffer, int BufLen);
```

File

System

Description

Call GetModuleFileName to obtain the fully qualified name of a module, given its handle. The version of GetModuleFileName defined in the System (see page 3159) unit is available only on Linux. On Windows, use the Windows API with the same name instead.

Module is the handle of the module whose file name is required.

Buffer is a buffer that receives the file name.

BufLen is the number of bytes in Buffer.

GetModuleFileName returns the number of bytes written to Buffer.

Warning: GetModuleFileName is not always successful. Depending on the way the application is called, the presence of the /proc file system, and other factors, there are times when GetModuleFileName may return a name that is not fully qualified or in some cases, no file name at all. Be sure to check the results of this function before using the returned value.

See Also

GetModuleName (see page 4207)

35.3.72 System.GetVariantManager Function

Returns the entry points of the routines that define variant behavior.

Pascal

```
procedure GetVariantManager(var VarMgr: TVariantManager); deprecated;
```

C++

```
GetVariantManager(TVariantManager VarMgr);
```

File

System

Description

GetVariantManager returns the entry points of the routines that perform variant type conversion and define other variant behavior.

Note: Although GetVariantManager is defined in System.hpp, do not use this procedure in C++ code. Working directly with the Variant (see page 3421) manger is not guaranteed to work in C++.

See Also

SetVariantManager (see page 3668)

IsVariantManagerSet (see page 3606)

35.3.73 System.Halt Function

Initiates abnormal termination of a program.

Pascal

```
procedure Halt; overload;
procedure Halt(Exitcode: Integer); overload;
```

C++

```
Halt();
Halt(int Exitcode);
```

File

System

Description

Halt performs an abnormal termination of a program and returns to the operating system.

To perform a normal termination of a Delphi application, call the Terminate method on the global Application object. If the application does not use a unit that provides an Application object, call the Exit (see page 3565) procedure from the main Program block.

Exitcode is an optional expression that specifies an exit code for the program.

Delphi Examples:

```
{
```

```
This example halts before the TextOut and terminates the
application.  Do not expect to one step this procedure.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   if 1 = 1 then
     begin
       if 2 = 2 then
         begin
           if 3 = 3 then
             begin
               Halt(1); { Halt right here! }
             end;
         end;
     end;
   Canvas.TextOut(10, 10, 'This will not be executed');
 end;
```

35.3.74 System.Hi Function

Returns the high-order byte of X as an unsigned value.

Pascal

```
function Hi(X): Byte;
```

C++

```
Byte Hi( X);
```

File

System

Description

In Delphi code, use Hi to obtain the high-order byte of an expression of type Word or Integer.

Note: Hi treats Integer as a 16-bit value, even if it is 32-bit.

Delphi Examples:

```
{
This example demonstrate use of the System Hi function to
store the high byte of a word or integer into a byte.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  B : Byte;
begin
  B := Hi($1234);
  ShowMessage(Format('High byte of $1234 = $%x', [B]));
end;
```

See Also

Lo (see page 3609)

35.3.75 System.High Function

Returns the highest value in the range of an argument.

Pascal

```
procedure High(X);
```

C++

```
High( X);
```

File

System

Description

In Delphi code, call High to obtain the upper limit of an Ordinal, Array, or ShortString value. The result type is X, or the index type of X.

X is either a type identifier or a variable reference. The type denoted by X, or the type of the variable denoted by X, must be one of the following:

For this type	High returns
Ordinal type (includes Int64)	The highest value in the range of the type
Array type	The highest value within the range of the index type of the array. For empty arrays, High returns –1.
short string type	The declared size of the string
Open array	The value, of type Integer, giving the number of elements in the actual parameter minus one
short string parameter	The value, of type Integer, giving the number of elements in the actual parameter minus one

Delphi Examples:

```
{
This example requires a button and two list boxes.  Two lists
of floating point numbers are generated initially and
displayed in the list boxes. Clicking the button sums the lists.
Notice that the range of the list arrays can be any integers,
as long as Low is less than High.
}
var
  List1: array[0..3] of Double;
  List2: array[5..17] of Double;

function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Real;
begin
  S := 0; { Note that open array index range is always zero-based. }
  for I := 0 to High(X) do S := S + X[I];
  Sum := S;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  S, TempStr: string;
begin
```

```
    Str(Sum(List1):4:2, S);
    S := 'Sum of List1: ' + S + #13#10;
    S := S + 'Sum of List2: ';
    Str(Sum(List2):4:2, TempStr);
    S := S + TempStr;
    MessageDlg(S, mtInformation, [mbOk], 0);

  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    X: Word;
  begin
    for X := Low(List1) to High(List1) do
    begin
        List1[X] := X * 3.4;
        ListBox1.Items.Add(FloatToStr(List1[X]));
    end;
    for X := Low(List2) to High(List2) do
    begin
        List2[X] := X * 0.0123;
        ListBox2.Items.Add(FloatToStr(List2[X]));
    end;
  end;
```

See Also

Low (see page 3610)

VarArrayHighBound (see page 5065)

35.3.76 System.Inc Function

Increments an ordinal value by one or N.

Pascal

```
procedure Inc(var X); overload;
procedure Inc(var X; N: Longint); overload;
```

C++

```
Inc( X);
Inc( X, Longint N);
```

File

System

Description

In Delphi code, Inc adds one or N to the variable X.

X is a variable of an ordinal type (including Int64), or a pointer type if the extended syntax is enabled.

N is an integer-type expression.

X increments by 1, or by N if N is specified; that is, Inc(X) corresponds to the statement X := X + 1, and Inc(X, N) corresponds to the statement X := X + N. On some platforms, Inc may generate optimized code, especially useful in tight loops.

If X is a pointer type, it increments X by N times the size of the type pointed to. Thus, given

```
type
  PMyType = ^TMyType;
```

and

```
var
  P: PMyType;
```

the statement `Inc(P)` increments `P` by `SizeOf (` see page 3670`)(TMyType)`.

Warning: You can't use Inc on properties because it modifies the parameter.

Note: Inc(S, I) where S is a ShortInt and I is a number greater than 127 will cause an EIntOverFlow exception to be raised if range and overflow checking are on. In Delphi 1.0, this did not raise an exception.

Delphi Examples:

```
{
This example requires two buttons and two text fields.  The
text fields can be set to any integer and long integer value.
Click the buttons to execute Inc.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  IntVar: Integer;
begin
  IntVar := StrToInt(Edit1.Text);
  Inc(IntVar);
  Edit1.Text := IntToStr(IntVar);       { IntVar := IntVar + 1 }
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  LongintVar: Longint;
begin
  LongintVar := StrToInt(Edit2.Text);
  Inc(LongintVar, 5);
  Edit2.Text := IntToStr(LongintVar);   { LongintVar := LongintVar + 5 }
end;
```

See Also

Dec (see page 3550)

Pred (see page 3628)

Succ (see page 3680)

35.3.77 System.Include Function

Adds an element to a Delphi set or adds an event handler in .NET applications.

Pascal

```
procedure Include(var S: set of T; I: T);
```

C++

```
Include(set of T S, T I);
```

File

System

Description

When the first argument of Include is a set type variable, Include adds an element to the set. The expression `Include(S,I)` corresponds to `S := S + [I]`, where S is a set type variable, and I is an expression of a type compatible with the base type of

S. The Include procedure generates more efficient code than using the + operator.

On the .NET platform, Include is overloaded. When the first argument of Include is a multicast event, and the second argument is an event handler, Include adds the event handler as a listener to the event.

Delphi Examples:

```
{
This example demonstrates the use of Include and Exclude routines.
The example assumes a form with three checkboxes and a button. Also
a field called FMsgButtons of type TMsgDlgButtons is declared in the form.
}
procedure TOptionsForm.btShowMessageClick(Sender: TObject);
begin
  //Creates a Message Dialog with the Selected Buttons
  MessageDlg('Hello World!', mtInformation, FMsgButtons, 0);
end;

procedure TOptionsForm.CheckBoxClick(Sender: TObject);
begin
  { Check wether an option is selected and add an element into a set,
    or exclude it from the set}

  if cbYes.Checked then
    Include(FMsgButtons, mbYes)
  else
    Exclude(FMsgButtons, mbYes);

  if cbNo.Checked then
    Include(FMsgButtons, mbNo)
  else
    Exclude(FMsgButtons, mbNo);

  if cbCancel.Checked then
    Include(FMsgButtons, mbCancel)
  else
    Exclude(FMsgButtons, mbCancel);
end;
```

See Also

Exclude (see page 3564)

35.3.78 System.Initialize Function

Initializes a dynamically allocated variable.

Pascal

```
procedure Initialize(var V); overload;
procedure Initialize(var V; Count: Integer); overload;
```

C++

```
Initialize( V);
Initialize( V, int Count);
```

File

System

Description

Initialize should be used only in Delphi code where a variable is dynamically allocated by other means than the New (see page

3619) standard procedure.

For global variables, local variables, objects, and dynamic variables allocated using New (see page 3619), the compiler generates code that initializes all long strings and variants contained by a variable upon creation of the variable. A call to Initialize is required to initialize a variable before it can be used if:

- the dynamic variable is created by other means than the New (see page 3619) standard function (for example using GetMem (see page 3584) or ReallocMem (see page 3637)).
- the variable contains long strings, variants, or interfaces.
- the dynamic variable is created by other means than the New (see page 3619) standard function (for example using GetMem (see page 3584) or ReallocMem (see page 3637)).
- the variable contains long strings, variants, or interfaces.
- the memory allocated for the variable is not initialized to zeros.

Initialize simply zeros out the memory occupied by long strings, variants, and interfaces, causing long strings to be empty and variants and interfaces to be Unassigned.

In cases where several variables are allocated in a contiguous memory block, the additional Count parameter can be specified to initialize all variables in one operation.

If the variable specified in a call to Initialize contains no long strings, variants, or interfaces, the compiler eliminates the call and generates no code for it.

Delphi Examples:

```
{
This example demonstrates the use of Initialize and Finalize functions
used to initialize and finalize a RTTI-enabled structure.
}
type
  { Declaring a data type that needs initilization }
  PPerson = ^TPerson;
  TPerson = record
    FFirstName: String;
    FLastName: String;
    FAge: Integer;
  end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Person: PPerson;
begin
  {
    Allocate a block of memory for the structure
    We're using an "unsafe" function: GetMem
    which doesn't use RTTI to initialize the
    strings inside the structure.
  }
  GetMem(Person, SizeOf(TPerson));

  { Initialize the structure in the Heap - RTTI will be used }
  Initialize(Person^);

  { Assign some data to the structure }
  Person.FFirstName := 'John';
  Person.FLastName := 'Smith';
  Person.FAge := 31;

  { Use RTTI to cleanup the internals of the structure }
  Finalize(Person^);
  FreeMem(Person);
end;
```

See Also

Finalize (see page 3572)

New (see page 3619)

35.3.79 System.Insert Function

Inserts a substring into a string beginning at a specified point.

Pascal

```
procedure Insert(Source: string; var S: string; Index: Integer);
```

C++

```
Insert(UnicodeString Source, UnicodeString S, int Index);
```

File

System

Description

In Delphi code, Insert merges Source into S at the position S[index].

Source is a string-type expression. S is a string-type variable of any length. Index is an integer-type expression. It is a character index and not a byte index.

If Index is less than 1, it is mapped to a 1. If it is past the end of the string, it is set to the length of the string, turning the operation into an append.

If the Source parameter is an empty string, Insert does nothing.

Insert throws an EOutOfMemory exception if it is unable to allocate enough memory to accomodate the new returned string.

Delphi Examples:

```
{
This example uses the Insert function to edit a string.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
begin
  S := 'Honest Lincoln';
  Canvas.TextOut(10, 10, S);
  Insert('Abe ', S, 8);                    { 'Honest Abe Lincoln' }
  Canvas.TextOut(10, 40, S);
end;
```

See Also

Concat (see page 3545)

Copy (see page 3547)

Delete (see page 3552)

Length (see page 3607)

Pos (see page 3627)

35.3.80 System.Int Function

Returns the integer part of a real number.

Pascal

```
function Int(const X: Extended): Extended;
```

C++

```
Extended Int(const Extended X);
```

File

System

Description

In Delphi code, Int returns the integer part of X; that is, X rounded toward zero. X is a real-type expression.

Delphi Examples:

```
{
Click the button to display the integer part of any positive or
negative real number.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  R: Real;
begin
  R := StrToFloat(Edit1.Text);     { 123.456 }
  R := Int(R);                     { 123.0 }
  Edit2.Text := FloatToStr(R);
end;
```

See Also

Frac (see page 3576)

Round (see page 3648)

Trunc (see page 3684)

35.3.81 System.IOResult Function

Returns the status of the last I/O operation performed.

Pascal

```
function IOResult: Integer;
```

C++

```
int IOResult();
```

File

System

Description

IOResult returns the result of the last Delphi I/O routine when I/O-checking is off (that is, when using the {$I-} compiler option). If

IOResult returns 0, the last I/O operation succeeded. Otherwise, IOResult returns an error code.

If an I/O error occurs and I/O-checking is off, all subsequent I/O operations are ignored until the internal error flag is cleared. Calling IOResult clears the internal error flag.

An alternative way to handle I/O errors is to use exception handling with the {$I+} compiler option.

The following table lists all I/O errors, numbers, and descriptions.

Number	Name	Description
100	Disk read error	Reported by Read (see page 3634) on a typed file if you attempt to read past the end of the file.
101	Disk write error	Reported by CloseFile (see page 3539), Write (see page 3707), WriteIn, or Flush (see page 3575) if the disk becomes full.
102	File not assigned	Reported by Reset (see page 3643), Rewrite (see page 3645), Append (see page 3520), Rename (see page 3642), or Erase (see page 3560) if the file variable has not been assigned a name through a call to Assign or AssignFile (see page 3525).
103	File not open	Reported by CloseFile (see page 3539), Read (see page 3634) Write (see page 3707), Seek (see page 3650), Eof (see page 3558), FilePos (see page 3567), FileSize (see page 3569), Flush (see page 3575), BlockRead (see page 3528), or BlockWrite (see page 3530) if the file is not open.
104	File not open for input	Reported by Read (see page 3634), Readln, Eof (see page 3558), Eoln (see page 3559), SeekEof (see page 3651), or SeekEoln (see page 3652) on a text file if the file is not open for input.
105	File not open for output	Reported by Write (see page 3707) or Writeln on a text file if you do not generate a Console application.
106	Invalid numeric format	Reported by Read (see page 3634) or Readln if a numeric value read from a text file does not conform to the proper numeric format.

Delphi Examples:

```
{
This example illustrates how to use IOResult to check the success of the Reset
method. If the call to Reset is successful, a message dialog displays the size
of the file that the user selects in an Open dialog. If the call to Reset fails,
a message warns the user that there was a file access error. To test the failure
part of this example, just misspell a file name in the open dialog.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File of Byte;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    {$I-}
    Reset(F);
    {$I+}
    if IOResult = 0 then
    begin
      MessageDlg('File size in bytes: ' + IntToStr(FileSize(F)),
        mtInformation, [mbOk], 0);
      CloseFile(F);
    end
    else
      MessageDlg('File access error', mtWarning, [mbOk], 0);
  end;
end;
```

See Also

I/O errors

35.3.82 System.IsMemoryManagerSet Function

Indicates whether the memory manager has been overridden using the SetMemoryManager (see page 3660) procedure.

Pascal

```
function IsMemoryManagerSet: Boolean;
```

C++

```
Boolean IsMemoryManagerSet();
```

File

System

Description

Call IsMemoryManagerSet to determine whether the default system memory management has been changed by a call to SetMemoryManager (see page 3660). IsMemoryManagerSet returns false if the default system memory management is in place, true otherwise.

Note: In C++, IsMemoryManagerSet always returns true, since C++ Builder automatically loads a custom memory manager.

Delphi Examples:

```
{
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
}
var
  OldMemMgr       : TMemoryManagerEx;
  GetMemCalls     : Integer;
  FreeMemCalls    : Integer;
  ReallocMemCalls : Integer;
  AllocMemCalls   : Integer;

function MyGetMem(Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.GetMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(GetMemCalls);
end;

function MyFreeMem(P: Pointer): Integer;
begin
  { Route the call }
  Result := OldMemMgr.FreeMem(P);

  { Safely increment the counter }
  InterlockedIncrement(FreeMemCalls);
end;

function MyReallocMem(P: Pointer; Size: Integer): Pointer;
```

```
begin
  { Route the call }
  Result := OldMemMgr.ReallocMem(P, Size);

  { Safely increment the counter }
  InterlockedIncrement(ReallocMemCalls);
end;

function MyAllocMem(Size: Cardinal): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.AllocMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(AllocMemCalls);
end;

procedure TForm2.btPerformSomethingClick(Sender: TObject);
var
  X : Integer;
  I : ^Integer;
begin
  { Perform some allocations and frees }
  for X := 0 to 999 do
  begin
    New(I);
    Dispose(I);
  end;
end;

procedure TForm2.btUseMyMemMgrClick(Sender: TObject);
var
  MyMemMgr : TMemoryManagerEx;
begin
  { Switch button states }
  btUseSystemMemMgr.Enabled := true;
  btUseMyMemMgr.Enabled := false;

  { Get the old memory manager }
  GetMemoryManager(OldMemMgr);

  { Create our instance }
  MyMemMgr.GetMem := MyGetMem;
  MyMemMgr.FreeMem := MyFreeMem;
  MyMemMgr.ReallocMem := MyReallocMem;
  MyMemMgr.AllocMem := MyAllocMem;

  { Use the defaults for this - not important }
  MyMemMgr.RegisterExpectedMemoryLeak := OldMemMgr.RegisterExpectedMemoryLeak;
  MyMemMgr.UnregisterExpectedMemoryLeak := OldMemMgr.UnregisterExpectedMemoryLeak;

  { Clear out the count variables }
  GetMemCalls := 0;
  FreeMemCalls := 0;
  ReallocMemCalls := 0;
  AllocMemCalls := 0;

  { Install the new memory manager }
  SetMemoryManager(MyMemMgr);
end;

procedure TForm2.btUseSystemMemMgrClick(Sender: TObject);
begin
  { Switch button states }
  btUseSystemMemMgr.Enabled := false;
  btUseMyMemMgr.Enabled := true;
```

```
    { Set the old memory manager back! }
    SetMemoryManager(OldMemMgr);

    GetMemCalls := 0;
    FreeMemCalls := 0;
    ReallocMemCalls := 0;
    AllocMemCalls := 0;
  end;

  procedure TForm2.FormDestroy(Sender: TObject);
  begin
    { Set the old memory manager back }
    if IsMemoryManagerSet then
       SetMemoryManager(OldMemMgr);
  end;

  procedure TForm2.Timer1Timer(Sender: TObject);
  begin
    {
    Note that IntToStr calls will also Allocate/Free
    memory so at each timer tick the counts will be increased;
    }
    Form2.edtGetMemCalls.Text := IntToStr(GetMemCalls);
    Form2.edtFreeMemCalls.Text := IntToStr(FreeMemCalls);
    Form2.edtReallocMemCalls.Text := IntToStr(ReallocMemCalls);
    Form2.edtAllocMemCalls.Text := IntToStr(AllocMemCalls);
  end;
```

See Also

GetMemoryManager (see page 3587)

SetMemoryManager (see page 3660)

35.3.83 System.IsVariantManagerSet Function

Indicates whether variant support is implemented in an application.

Pascal

```
function IsVariantManagerSet: Boolean; deprecated;
```

C++

```
Boolean IsVariantManagerSet();
```

File

System

Description

Use IsVariantManagerSet in Delphi code to determine whether the compiler knows the locations of functions that define variant behavior. In any program or unit that uses the Variants (see page 4973) unit (directly or indirectly), IsVariantManagerSet should return true.

Note: Although IsVariantManagerSet is defined in System.hpp, do not use this function in C++ code. Working directly with the Variant (see page 3421) manger is not guaranteed to work in C++.

See Also

GetVariantManager (see page 3594)

SetVariantManager (see page 3668)

35.3.84 System.Length Function

Returns the number of characters in a string or elements in an array.

Pascal

```
function Length(S): Integer;
```

C++

```
int Length( S);
```

File

System

Description

In Delphi code, Length returns the number of characters actually used in the string or the number of elements in the array. In C++ code, use the method of the same name on the AnsiString or DynamicArray (see page 3261) class.

For single-byte and multibyte strings, Length returns the number of bytes used by the string. For Unicode (WideString (see page 3467)) strings, Length returns the number of bytes divided by two.

S is a string- or array-valued expression.

Delphi Examples:

```
{
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  Label1.Caption := '';
  for i := 1 to Length(Edit1.Text) do
  begin
    try
      Label1.Caption :=
        Label1.Caption +
        SysUtils.IntToHex(Byte(Edit1.Text[i]),2) + ' ';
    except
      Beep;
    end;
  end;
end;
{
The following OnFind event handler searches a memo component
for the text specified in the FindText property of a find
dialog component. If found, the first occurrence of the text
in Memo1 is selected. The code uses the Pos function to
compare strings, and stores the number of characters to skip
when determining the selection position in the SkipChars
variable. Because there is no handling of case, whole word,
or search direction in this algorithm, it is assumed that
the Options property of FindDialog1 is set to
[frHideMatchCase, frHideWholeWord, frHideUpDown].
}
```

```
procedure TForm1.FindDialog1Find(Sender: TObject);
var
  I, J, PosReturn, SkipChars: Integer;
begin
  for I := 0 to Memo1.Lines.Count do
  begin
    PosReturn := Pos(FindDialog1.FindText,Memo1.Lines[I]);
    if PosReturn <> 0 then {found!}
    begin
      SkipChars := 0;
      for J := 0 to I - 1 do
        SkipChars := SkipChars + Length(Memo1.Lines[J]);
      SkipChars := SkipChars + (I*2);
      SkipChars := SkipChars + PosReturn - 1;
      Memo1.SetFocus;
      Memo1.SelStart := SkipChars;
      Memo1.SelLength := Length(FindDialog1.FindText);
      Break;
    end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
const Path = 'country.list';
begin
  Memo1.Lines.LoadFromFile(Path);
end;
```

See Also

Concat (see page 3545)

Copy (see page 3547)

Delete (see page 3552)

Insert (see page 3601)

Pos (see page 3627)

35.3.85 System.Ln Function

Returns the natural log of a real expression.

Pascal

```
function Ln(const X: Extended): Extended;
```

C++

```
Extended Ln(const Extended X);
```

File

System

Description

In Delphi code, Ln returns the natural logarithm (Ln(e) = 1) of the real-type expression X.

Delphi Examples:

```
{
This example calculated the exponential of 1.0 and then calculates
the natural log of that quantity to show that the natural log
```

```
function is the inverse of the exponential function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   e : real;
   S : string;
begin
   e := Exp(1.0);
   Str(ln(e):3:2, S);
   S := 'e = ' + FloatToStr(e) + '; ln(e) = ' + S;
   Canvas.TextOut(10, 10, S);
end;
```

See Also

Exp (see page 3567)

LnXP1

35.3.86 System.Lo Function

Returns the low order Byte of argument X.

Pascal

```
function Lo(X): Byte;
```

C++

```
Byte Lo( X);
```

File

System

Description

In Delphi code, Lo returns the low-order Byte of the argument X as an unsigned value. X is an expression of type Integer.

Delphi Examples:

```
{
This example demonstrate use of the System Lo function to
store the low byte of a word or integer into a byte.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  B : Byte;
begin
  B := Lo($1234);
  ShowMessage(Format('Low byte of $1234 = $%x', [B]));
end;
```

See Also

Hi (see page 3595)

35.3.87 System.LoadResourceModule Function

Pascal

```
function LoadResourceModule(ModuleName: PChar; CheckOwner: Boolean = True): LongWord;
```

C++

```
LongWord LoadResourceModule(const char * ModuleName, Boolean CheckOwner = True);
```

File

System

Description

This is function System.LoadResourceModule.

35.3.88 System::LoadResourceString Function

Pascal

```
function LoadResourceString(rsrc: ResourceString const * const): String;
```

C++

```
__fastcall String LoadResourceString(ResourceString const * const rsrc);
```

File

sysclass.h

Description

This is function System::LoadResourceString.

35.3.89 System.LoadResString Function

Pascal

```
function LoadResString(ResStringRec: PResStringRec): string;
```

C++

```
UnicodeString LoadResString(PResStringRec ResStringRec);
```

File

System

Description

This is function System.LoadResString.

35.3.90 System.Low Function

Returns the lowest value in a range.

Pascal

```
procedure Low(X);
```

C++

```
Low( X);
```

File

System

Description

In Delphi code, call Low to obtain the lowest value or first element of an Ordinal, Array or ShortString. Result type is X, or the index type of X where X is either a type identifier or a variable reference.

Type	Low returns
Ordinal type (includes Int64)	The lowest value in the range of the type
Array type	The lowest value within the range of the index type of the array
short string type	Returns 0
Open array	Returns 0
short string parameter	Returns 0

Delphi Examples:

```
{
This example requires a button and two list boxes.  Two lists
of floating point numbers are generated initially and
displayed in the list boxes. Clicking the button sums the lists.
Notice that the range of the list arrays can be any integers,
as long as Low is less than High.
}
var
  List1: array[0..3] of Double;
  List2: array[5..17] of Double;

function Sum( var X: array of Double): Double;
var
  I: Word;
  S: Real;
begin
  S := 0; { Note that open array index range is always zero-based. }
  for I := 0 to High(X) do S := S + X[I];
  Sum := S;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  S, TempStr: string;
begin
  Str(Sum(List1):4:2, S);
  S := 'Sum of List1: ' + S + #13#10;
  S := S + 'Sum of List2: ';
  Str(Sum(List2):4:2, TempStr);
  S := S + TempStr;
  MessageDlg(S, mtInformation, [mbOk], 0);

end;

procedure TForm1.FormCreate(Sender: TObject);
var
  X: Word;
begin
  for X := Low(List1) to High(List1) do
  begin
      List1[X] := X * 3.4;
      ListBox1.Items.Add(FloatToStr(List1[X]));
  end;
  for X := Low(List2) to High(List2) do
```

```
  begin
      List2[X] := X * 0.0123;
      ListBox2.Items.Add(FloatToStr(List2[X]));
  end;
end;
```

See Also

High (see page 3596)

VarArrayLowBound (see page 5067)

35.3.91 System.Mark Function

Pascal

```
procedure Mark; deprecated;
```

C++

```
Mark();
```

File

System

Description

This is function System.Mark.

35.3.92 System.MemoryBarrier Function

Pascal

```
procedure MemoryBarrier;
```

C++

```
MemoryBarrier();
```

File

System

Description

This is function System.MemoryBarrier.

35.3.93 System.MkDir Function

Creates a new subdirectory.

Pascal

```
procedure MkDir(const S: string); overload;
procedure MkDir(P: PChar); overload;
```

C++

```
MkDir(const UnicodeString S);
MkDir(const char * P);
```

File

System

Description

MkDir creates a new subdirectory with the path specified by string S or P. The last item in the path cannot be an existing file name.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for an I/O error.

Delphi Examples:

```
{
The following sample code sets the directory of
DirectoryListBox1 to C:\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
mytemp is added to C:\WINDOWS, but note that it is not
updated in DirectoryListBox1 until Button2 is pressed
and Update is called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current path to see the magic happen.
}

procedure TForm1.Button1Click(Sender: TObject);
begin
  MkDir(Edit1.Text);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  DirectoryListBox1.Update;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  DirectoryListBox1.Directory := 'c:\WINDOWS';
end;
{
This example creates a directory using the path and directory name
entered in the text edit.  The path is relative to the project
directory.  Type in the name of an existing directory to induce the
error.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  {$I-}
  { Get directory name from TEdit control }
  MkDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot create directory', mtWarning, [mbOk], 0, mbOk)
  else
    MessageDlg('New directory created', mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

ChDir (see page 3534)

GetDir (see page 3580)

RmDir (see page 3647)

ForceDirectories (see page 4186)

35.3.94 System.MonitorEnter Function

Prohibits the access of all other threads but the calling one to the specified object.

Pascal

```
function MonitorEnter(AObject: TObject; Timeout: Cardinal = INFINITE): Boolean;
```

C++

```
Boolean MonitorEnter(TObject * AObject, unsigned Timeout = INFINITE);
```

File

System

Description

Use the MonitorEnter function to prohibit `AObject` from being accessed by threads, other than the calling one.

Before attempting to lock the specified object, MonitorEnter tests whether the thread that is currently locking the object, if any, has made an equal number of calls to MonitorEnter and to MonitorExit (see page 3614). If not, the calling thread is blocked from accessing the object until the previous thread releases it.

MonitorEnter attempts to lock the specified object for the amount of time designated by the `Timeout` parameter. If the lock attempt is successful, MonitorEnter returns True, otherwise it returns False.

See Also

MonitorExit (see page 3614)

TMonitor (see page 3349)

Enter (see page 3350)

35.3.95 System.MonitorExit Function

Releases a lock of the calling thread on the specified object.

Pascal

```
procedure MonitorExit(AObject: TObject);
```

C++

```
MonitorExit(TObject * AObject);
```

File

System

Description

Use MonitorExit to release a lock of the calling thread on `AObject`, which is also owned by the thread. The object is released from all the locks of the calling thread as soon as the number of the thread's calls to MonitorExit is equal to the number of calls to

MonitorEnter (see page 3614).

See Also

MonitorEnter (see page 3614)

TMonitor (see page 3349)

Exit (see page 3351)

35.3.96 System.MonitorPulse Function

Notifies the next thread in the waiting queue that it will be able to lock the specified object as soon as the calling thread releases it.

Pascal

```
procedure MonitorPulse(AObject: TObject);
```

C++

```
MonitorPulse(TObject * AObject);
```

File

System

Description

MonitorPulse notifies the next thread in the waiting queue that it can move to the ready queue. Thus, as soon as the calling thread releases the lock on `AObject`, the next thread in the ready queue will lock it.

See Also

MonitorPulseAll (see page 3615)

TMonitor (see page 3349)

Pulse (see page 3352)

35.3.97 System.MonitorPulseAll Function

Signals all threads in the waiting queue that either one of them will be able to lock the specified object as soon as the calling thread releases it.

Pascal

```
procedure MonitorPulseAll(AObject: TObject);
```

C++

```
MonitorPulseAll(TObject * AObject);
```

File

System

Description

MonitorPulseAll notifies all threads in the waiting queue that they can move to the ready queue. As soon as the calling thread releases the lock on `AObject`, the first thread in the ready queue will lock it.

See Also

MonitorPulse (see page 3615)

TMonitor (see page 3349)

PulseAll (see page 3352)

35.3.98 System.MonitorTryEnter Function

Attempts to lock the specified object, so that only the calling thread is able to access it.

Pascal

```
function MonitorTryEnter(AObject: TObject): Boolean;
```

C++

```
Boolean MonitorTryEnter(TObject * AObject);
```

File

System

Description

The MonitorTryEnter function attempts to lock the object given as parameter, so that only the calling thread can access it.

MonitorTryEnter returns True if it manages to lock the object, or False otherwise.

As opposed to the MonitorEnter (see page 3614) method, MonitorTryEnter does not block the calling thread if the given object is currently owned by other threads.

See Also

MonitorEnter (see page 3614)

TMonitor (see page 3349)

Enter (see page 3350)

35.3.99 System.MonitorWait Function

Drops the lock on the specified object and puts the calling thread in the waiting queue for a specified amount of time.

Pascal

```
function MonitorWait(AObject: TObject; Timeout: Cardinal): Boolean;
```

C++

```
Boolean MonitorWait(TObject * AObject, unsigned Timeout);
```

File

System

Description

Use MonitorWait to drop the lock on the specified object and also to put the calling thread in the waiting queue for the amount of time specified in the `Timeout` parameter. Hence, the calling thread is blocked until it is again able to lock `AObject`.

See Also

TMonitor (see page 3349)

Wait (see page 3353)

35.3.100 System.Move Function

Copies bytes from a source to a destination.

Pascal

```
procedure Move(const Source; var Dest; Count: Integer);
```

C++

```
Move(const  Source,  Dest, int Count);
```

File

System

Description

Move copies Count bytes from Source to Dest. No range checking is performed. Move compensates for overlaps between the source and destination blocks.

Whenever possible, use the global SizeOf (see page 3670) function (Delphi) or the sizeof operator (C++) to determine the count.

Delphi Examples:

```
{
This example moves characters from a Char array into an integer.
Displaying the integer as a hex, you can see that 'W' (0x57) is
stored in the least significant byte of the integer.
}
var
  A: array[1..4] of Char;
  B: Integer;

procedure DisplayAB;
begin
  Form1.ListBox1.Items[0] := A[1];
  Form1.ListBox1.Items[1] := A[2];
  Form1.ListBox1.Items[2] := A[3];
  Form1.ListBox1.Items[3] := A[4];
  Form1.Edit1.Text := IntToHex(B, 1);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Move(A, B, SizeOf(B));  { SizeOf = safety! }
  DisplayAB;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I : Integer;
begin
  A[1] := 'W';
  A[2] := 'H';
  A[3] := 'A';
```

```
  A[4] := 'T';
  B := 5;
  DisplayAB;
end;
```

35.3.101 System.MoveChars Function

Copies bytes from a source to a destination.

Pascal

```
procedure MoveChars(const Source; var Dest; Length: Integer);
```

C++

```
MoveChars(const  Source,  Dest, int Length);
```

File

System

Description

MoveChars copies Length (see page 3607) bytes from Source to Dest. MoveChars compensates for overlaps between the source and destination blocks.

Note: MoveChars assumes that the size of Char is 2 Bytes (when Unicode is enabled). Use the Move (see page 3617) function to copy characters from a string that has character size different from SizeOf (see page 3670)(Char).

C++ Examples:

```
/*
The following code demostrates the use of memory management functions in C++. Three
edit boxes and a button are expected onthe form.
*/
String __fastcall FastStrCat(String s1, String s2)
{
  /*
   Allocated enough space in FinalStr to copy the contents
   of the initial string + $00 character
  */
  wchar_t* finalStr = (wchar_t *)GetMemory((s1.Length() + 1) * sizeof(wchar_t));

  // Copy the contents of the first string
  MoveChars(s1.data(), finalStr, s1.Length() + 1);

  // And now expand the final string + $00 character
  finalStr = (wchar_t *)ReallocMemory(finalStr, (s1.Length() + s2.Length() + 1) *
sizeof(wchar_t));

  // Copy the contents of the second string
  MoveChars(s2.data(), finalStr + s1.Length(), s2.Length() + 1);

  // Get the result in String
  String result = String(finalStr);
  FreeMemory(finalStr);

  return result;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Concatenate 2 string
```

```
  Edit3->Text = FastStrCat(Edit1->Text, Edit2->Text);
}
```

Delphi Examples:

```
{
The following code demostrates the use of ReallocMem function. Three
edit boxes and a button are expected onthe form.
}
function FastStrCat(const S1, S2: String): String;
var
  FinalStr: PChar;
begin
  {
   Allocated enough space in FinalStr to copy the contents
   of the initial string + $00 character
  }
  GetMem(FinalStr, (Length(S1) + 1) * SizeOf(Char));

  { Copy the contents of the first string }
  MoveChars(S1[1], FinalStr^, Length(S1) + 1);

  { And now expand the final string + $00 character }
  ReallocMem(FinalStr, (Length(S1) + Length(S2) + 1) * SizeOf(Char));

  { Copy the contents of the second string }
  MoveChars(S2[1], FinalStr[Length(S1)], Length(S2) + 1);

  { Get the result in String }
  SetString(Result, FinalStr, Length(S1) + Length(S2));
  FreeMem(FinalStr);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  { Concatenate 2 string }
  Edit3.Text := FastStrCat(Edit1.Text, Edit2.Text);
end;
```

See Also

Move (see page 3617)

35.3.102 System.New Function

Creates a new dynamic variable and sets P to point to it.

Pascal

```
procedure New(var P: Pointer);
```

C++

```
New(void * P);
```

File

System

Description

In Delphi code, the New procedure creates a new dynamic variable and sets a pointer variable to point to it. P is a variable of any pointer type. The size of the allocated memory block corresponds to the size of the type that P points to. The newly created variable can be referenced as P^. If there isn't enough memory available to allocate the dynamic variable, an EOutOfMemory

exception is raised.

When an application is finished using a dynamic variable created with New, it should dispose of the memory allocated for the variable using the Dispose (see page 3553) standard procedure.

Delphi Examples:

```
{
The following code explicitly allocates memory for a pointer
in the OnCreate event of Form1, then releases the memory in
the OnDestroy event. Assume that MyPtr is a Pointer type
field of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  New(MyPtr);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Dispose(MyPtr);
end;
{
This example requires three buttons, four TEdits, two labels
and a populated TreeView.  The following code defines a
record type of TMyRec and a record pointer type of PMyRec.
}
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    TreeView1: TTreeView;
    Label2: TLabel;
    Label3: TLabel;
    Button3: TButton;
    Edit4: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  type
  PMyRec = ^TMyRec;
  TMyRec = record
    FName: string;
    LName: string;
  end;


var
  Form1: TForm1;

implementation

{$R *.dfm}

{
Assuming these types are used, the following code adds a
node to TreeView1 as the last sibling of the selected node.
A TMyRec record is associated with the added item. The FName
```

```
  and LName fields are obtained from edit boxes Edit1 and
  Edit2.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    MyRecPtr: PMyRec;
    TreeViewIndex: LongInt;
  begin
    New(MyRecPtr);
    MyRecPtr^.FName := Edit1.Text;
    MyRecPtr^.LName := Edit2.Text;
    with TreeView1 do
    begin
      TreeViewIndex := Selected.AbsoluteIndex;
      if Items.Count = 0 then
        Items.AddObject(nil, 'Item' + IntToStr(TreeViewIndex), MyRecPtr)
      else if (TreeViewIndex < Items.Count) and (TreeViewIndex >= 0) then
        Items.AddObject(Items[TreeViewIndex], 'Item' + IntToStr(TreeViewIndex), MyRecPtr);
    end;
  end;
  {
  After an item containing a TMyRec record has been added, the
  following code retrieves the FName and LName values
  associated with the item and displays the values in a label.
  }
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    if (TreeView1.Selected.Data <> nil) then { query only works on new nodes }
      Edit3.Text := PMyRec(TreeView1.Selected.Data)^.FName + ' ' +
                    PMyRec(TreeView1.Selected.Data)^.LName;
  end;
  procedure TForm1.Button3Click(Sender: TObject);
  begin
    Edit4.Text := IntToStr(TreeView1.Selected.AbsoluteIndex);
  end;
```

See Also

Dispose (see page 3553)

FreeMem (see page 3577)

GetMem (see page 3584)

35.3.103 System.Odd Function

Returns true if argument is an odd number.

Pascal

```
function Odd(X: Integer): Boolean;
```

C++

```
Boolean Odd(int X);
```

File

System

Description

In Delphi code, Odd tests if the argument is an odd number. It returns true if X is an odd number, false if X is even.

Delphi Examples:

```
{
This example requires a text edit and a button with some clear
canvas in the top left ocrner of the form.  Enter the number to be
tested in the text edit and click to button to test if the entry
is odd.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   Canvas.TextOut(10, 10, '                                          ');
   if Odd(StrToInt(Edit1.Text)) then
     Canvas.TextOut(10, 10, Edit1.Text + ' is odd.')
   else
     Canvas.TextOut(10, 10, Edit1.Text + ' is even.');
end;
```

35.3.104 System.OleStrToString Function

Copies data received from a COM interface to a string.

Pascal

```
function OleStrToString(Source: PWideChar): UnicodeString;
```

C++

```
UnicodeString OleStrToString(PWideChar Source);
```

File

System

Description

OleStrToString copies data from the format used by COM into a Delphi string.

See Also

OleStrToStrVar (see page 3622)

StringToOleStr (see page 3678)

35.3.105 System.OleStrToStrVar Function

Copies a string from its COM representation to an existing Delphi string.

Pascal

```
procedure OleStrToStrVar(Source: PWideChar; var Dest: AnsiString); overload;
procedure OleStrToStrVar(Source: PWideChar; var Dest: UnicodeString); overload;
```

C++

```
OleStrToStrVar(PWideChar Source, AnsiString Dest);
OleStrToStrVar(PWideChar Source, UnicodeString Dest);
```

File

System

Description

OleStrToStrVar converts a wide string (the format used by COM interfaces) into an existing string variable.

See Also

OleStrToString (see page 3622)

StringToOleStr (see page 3678)

35.3.106 System.Ord Function

Returns the ordinal value of an ordinal-type expression.

Pascal

```
procedure Ord(X);
```

C++

```
Ord( X);
```

File

System

Description

X is a Delphi ordinal-type expression. The result is the ordinal position of X; its type is the smallest standard integer type that can hold all values of X's type.

Ord cannot operate on Int64 values.

Delphi Examples:

```
{
This example displays the ordinal value for different enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
   Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'BLUE has an ordinal value of ' + IntToStr(Ord(BLUE)) + #13#10;
  S := S + 'The ASCII code for "c" is ' + IntToStr(Ord('c')) +  ' decimal';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOK);
end;
```

See Also

Chr (see page 3536)

35.3.107 System.ParamCount Function

Returns the number of parameters passed on the command line.

Pascal

```
function ParamCount: Integer;
```

C++

```
int ParamCount();
```

File

System

Description

ParamCount returns the number of parameters passed to the program on the command line. Separate parameters with spaces or tabs. Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

C++ Examples:

```
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

ParamStr (see page 3625)

CmdLine (see page 3812)

FindCmdLineSwitch (see page 4174)

35.3.108 System.ParamStr Function

Returns a specified parameter from the command-line.

Pascal

```
function ParamStr(Index: Integer): string;
```

C++

```
UnicodeString ParamStr(int Index);
```

File

System

Description

ParamStr returns the parameter from the command line that corresponds to Index, or an empty string if Index is greater than ParamCount (see page 3623). For example, an Index value of 2 returns the second command-line parameter.

Note: On Windows, ParamStr(0) returns the path and file name of the executing program (for example, C:\TEST\MYPROG.EXE).

Note: On Linux, ParamStr(0) returns the command used to execute the program, without parameters (for example, ./myprogram). This behavior is dependent on information returned by the shell program and may not be consistent among all shells.

Note: Use double quotes to wrap multiple words as one parameter (such as long file names containing spaces).

C++ Examples:

```
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
The following example beeps once for each "beep" passed in
```

```
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

ParamCount (see page 3623)

CmdLine (see page 3812)

FindCmdLineSwitch (see page 4174)

35.3.109 System.Pi Function

Returns 3.1415926535897932385.

Pascal

```
function Pi: Extended;
```

C++

```
Extended Pi();
```

File

System

Description

Use Pi in Delphi code to represent the mathematical value pi, the ratio of a circle's circumference to its diameter. Pi is approximated as 3.1415926535897932385.

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
```

```
  Edit3.Text := mystring;
  Str(ex:25:23, mystring);
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

35.3.110 System.Pos Function

Returns the index value of the first character in a specified substring that occurs in a given string.

Pascal

```
function Pos(const substr: UnicodeString; const str: UnicodeString): Integer; overload;
function Pos(const substr: RawByteString; const str: RawByteString): Integer; overload;
function Pos(const substr: WideString; const str: WideString): Integer; overload;
```

C++

```
int Pos(const UnicodeString substr, const UnicodeString str);
int Pos(const RawByteString substr, const RawByteString str);
int Pos(const BSTR substr, const BSTR str);
```

File

System

Description

In Delphi, Pos searches for a substring, Substr, in a string, S. Substr and S are string-type expressions.

Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos is case-sensitive. If Substr is not found, Pos returns zero.

The PosEx function is similar to Pos, but provides additional features and can be used in C++ code.

Delphi Examples:

```
{
The following example updates the strings in a list box
given the strings contained in another list box. If a string
in the source list box has the form Name=Value and the
destination list box contains a string with the same Name
part, the Value part in the destination list box will be
replaced by the source's value.  TListBox objects that are
not name-value pairs are not assigned a Name and IndexOfName
will return -1.
}
procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
```

```
    begin
    for I := 0 to Source.Count - 1 do
    begin
      if Pos ('=', Source[I]) > 1 then
      begin
        DI := Dest.IndexOfName(Source.Names[I]);
        if DI > -1 then Dest[DI] := Source[I];
      end;
    end;
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    MergeStrings(ListBox1.Items, ListBox2.Items);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Plants = 10');
    ListBox1.Items.Add('Animals = 20');
    ListBox1.Items.Add('Minerals = 15');
    ListBox2.Items.Add('Animals = 4');
    ListBox2.Items.Add('Plants = 3');
    ListBox2.Items.Add('Minerals = 78');
  end;
  {
  This example demonstrates how to edit particular elements of a string.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var S: string;
  begin
    S := Edit1.Text;
    { Convert spaces to zeros }
    while Pos(' ', S) > 0 do
      S[Pos(' ', S)] := '0';
    Edit1.Text := S;
  end;
```

See Also

Concat (see page 3545)

Copy (see page 3547)

Delete (see page 3552)

Insert (see page 3601)

Length (see page 3607)

35.3.111 System.Pred Function

Returns the predecessor of the argument.

Pascal

```
procedure Pred(X);
```

C++

```
Pred( X);
```

File

System

Description

X is an expression of a Delphi ordinal type (including Int64). The result, of the same type as X, is the predecessor of X.

Delphi Examples:

```
{
The following code is a self-contained unit that you can add to a project
that logs failed assertions to the event log rather than raising an
exception. If /Debug is passed to the application as a command-line
parameter, all failed Assert calls cause a call to OutputDebugString,
which sends a messages to the Event Log window if run from Delphi, or
does nothing if run outside the IDE (OutputDebugString is a no-op if not
run from within a debugger). A nice side benefit is that all the line
numbers and unit names can be stripped from the EXE by turning off
assertions and rebuilding the project. Build the executable in the IDE
by hitting "Run".  Then go to the debug directory for the project and
execute the command line "AssertErrorProc_proj /Debug".
Note: AssertErrorHandler should do as little as possible.  This routine
does no cleanup, so no variables, no compiler generated variables.
Procedure calls are OK.
Note:   This example was originally provided by Brian Long.
}
type
  TForm1 = class(TForm)
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
    property Data: string read FData write FData;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  var
  Form1: TForm1;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);

implementation

{$R *.dfm}

{$ASSERTIONS ON}

type
  EMyAssert = class(Exception);


procedure DoMessage(Message, Filename: String; LineNumber: Integer; ErrorAddr: Pointer);
var
  S: String;
begin
  S := Format('%s (%s, line %d, address $%x)',
    [Message, Filename, LineNumber, Pred(Integer(ErrorAddr))]);
  OutputDebugString(PChar(S));
end;
```

```
procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);
{ No local variables.  Not compiler generated temporary variables. }
{ Using the call stack here will cause Access Violation errors. }
begin
  DoMessage(Message, Filename, LineNumber, ErrorAddr);
  raise EMyAssert.Create('Boom!');
end;

procedure AssertErrorNoHandler(const Message, Filename: string; LineNumber: Integer;
ErrorAddr: Pointer);
begin
end;

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, 'Value');
  AStorage.Data := s;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ModifyStorage(nil, 'Oops');
end;

initialization
  if FindCmdLineSwitch('Debug', ['/', '-'], True) then
  begin
    MessageDlg('Command line switch found!', mtInformation, [mbOK], 0);
//    OldProc := System.AssertErrorProc;
    System.AssertErrorProc := @AssertErrorHandler
  end
  else
    System.AssertErrorProc := @AssertErrorNoHandler;

  ModifyStorage(nil, 'Ooops');
{
This example demonstrates how Pred and Succ can be used to
manage enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'The predecessor of 5 is ' + IntToStr(Pred(5)) + #13#10;
  S := S + 'The successor of 10 is ' + IntToStr(Succ(10)) + #13#10;
  if Succ(RED) = BLUE then
    S := S + 'In the type Colors, RED is the predecessor of BLUE.';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Succ (see page 3680)

35.3.112 System.Ptr Function

Converts a specified address to a pointer.

Pascal

```
function Ptr(Address: Integer): Pointer;
```

C++

```
void * Ptr(int Address);
```

File

System

Description

In Delphi code, Ptr converts the given **Address** to a pointer. A call to this function generates no code, but simply treats the 32-bit value given by Address as a pointer. Like nil, the result of Ptr is assignment compatible with all pointer types.

Delphi Examples:

```
{
The following code demostrates the use of Ptr function. The example
makes use of Ptr to simulate pointer arithmetics.
}
function FastUpperCase(const S: String): String;
var
  C: ^Char;
  I: Integer;
begin
  { Get the text in the edit box }
  Result := S;

  { Ensure the string reference stored in Result is unique,
    otherwise we may change the original string also.
    No copy-on-write is done if we're accesing the character
    directly.
  }
  UniqueString(Result);

  { Find the address of the first char in the string }
  C := Addr(Result[1]);

  for I := 0 to Length(Result) - 1 do
  begin
    { Up-case the character }
    C^ := UpCase(C^);

    { Move to the next character }
    C := Ptr(Integer(C) + SizeOf(Char));
  end;
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  { Uppercase the string in the edit box }
  Edit1.Text := FastUpperCase(Edit1.Text);
end;
```

See Also

Addr (see page 3514)

35.3.113 System.PUCS4Chars Function

Converts a UCS4 string to a null-terminated array of UCS4 characters.

Pascal

```
function PUCS4Chars(const S: UCS4String): PUCS4Char;
```

C++

```
PUCS4Char PUCS4Chars(const UCS4String S);
```

File

System

Description

PUCS4Chars converts the UCS-4 string specified by S to a null-terminated sequence of UCS4Char (see page 3802) values.

See Also

UCS4StringToWideString (see page 3687)

Utf8ToUnicode (see page 3699)

WideStringToUCS4String (see page 3707)

35.3.114 System.RaiseList Function

Pascal

```
function RaiseList: Pointer; deprecated;
```

C++

```
void * RaiseList();
```

File

System

Description

This is function System.RaiseList.

35.3.115 System.Random Function

Generates random numbers within a specified range.

Pascal

```
function Random(const ARange: Integer): Integer; overload;
function Random: Extended; overload;
```

C++

```
int Random(const int ARange);
Extended Random();
```

File

System

Description

In Delphi code, Random returns a random number within the range 0 <= X < Range. If Range is not specified, the result is a real-type random number within the range

`0 <= X < 1.`

To initialize the random number generator, add a single call Randomize (see page 3633) or assign a value to the RandSeed (see page 3834) variable before making any calls to Random.

Note: Because the implementation of the Random function may change between compiler versions, we do not recommend using Random for encryption or other purposes that require reproducible sequences of pseudo-random numbers.

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

RandSeed (see page 3834)

35.3.116 System.Randomize Function

Initializes the random number generator with a random value.

Pascal

```
procedure Randomize;
```

C++

```
Randomize();
```

File

System

Description

Randomize initializes the built-in random number generator with a random value (obtained from the system clock). The random number generator should be initialized by making a call to Randomize, or by assigning a value to RandSeed (see page 3834).

Do not combine the call to Randomize in a loop with calls to the Random (see page 3632) function. Typically, Randomize is called only once, before all calls to Random (see page 3632).

Delphi Examples:

```
{
This example draws many rectangles of various sizes and
colors on a form maximized to fill the entire screen. To run
the code, drop a TTimer component on the form and use the
object inspector to create the OnTimer and OnActivate event
handlers.
}
var
  X, Y: Integer;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WindowState := wsMaximized;
  Timer1.Interval := 50;
  Randomize;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X := Random(Screen.Width - 10);
  Y := Random(Screen.Height - 10);
  Canvas.Pen.Color := Random(65535);
  case Random(5) of
    0: Canvas.Pen.Style := psSolid;
    1: Canvas.Pen.Style := psDash;
    2: Canvas.Pen.Style := psDot;
    3: Canvas.Pen.Style := psDashDot;
    4: Canvas.Pen.Style := psDashDotDot;
  end;
  Canvas.Rectangle(X, Y, X + Random(400), Y + Random(400));
end;
```

See Also

RandG

RandSeed (see page 3834)

Random (see page 3632)

35.3.117 System.Read Function

Read reads data from a file.

Pascal

```
procedure Read(F; V1); overload;
procedure Read(F; V1; Vn); overload;
procedure Read(F: Text; V1); overload;
procedure Read(F: Text; V1; Vn); overload;
```

C++

```
Read( F,  V1);
Read( F,  V1,  Vn);
Read(Text F,  V1);
Read(Text F,  V1,  Vn);
```

File

System

Description

The Read procedure can be used in Delphi code in the following ways.

For typed files, it reads a file component into a variable.

For text files, it reads one or more values into one or more variables.

The syntax shown here for Read demonstrates that the procedure can take a variable number of arguments.

Read reads all characters up to, but not including, the next end-of-line marker or until Eof (see page 3558)(F) becomes true; it does not skip to the next line after reading. If the resulting string is longer than the maximum length of the string variable, it is truncated.

After the first Read, each subsequent Read sees the end-of-line marker and returns a zero-length string.

Use multiple Readln calls to read successive string values.

When the extended syntax is enabled, Read can read null-terminated strings into zero-based character arrays.

Read reads one character from the file and assigns it to the variable. If CRLF mode is enabled and Eof (see page 3558)(F) was true before Read was executed, the value Chr (see page 3536)(26) (a Ctrl-Z character) is assigned to the variable. (To enable CRLF mode, use SetLineBreakStyle (see page 3659).)

Read skips any blanks, tabs, or end-of-line markers preceding the numeric string.

If the numeric string does not conform to the expected format, an I/O error occurs; otherwise, the value is assigned to the variable.

The next Read starts with the blank, tab, or end-of-line marker that terminated the numeric string.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
```

```
        begin
          Read(F1, Ch);
          Write(F2, Ch);
        end;
        CloseFile(F2);
      end;
      CloseFile(F1);
    end;
  end;
```

See Also

Eof (see page 3558)

ReadLn (see page 3636)

Write (see page 3707)

WriteLn (see page 3710)

SetLineBreakStyle (see page 3659)

35.3.118 System.ReadLn Function

Reads a line of text from a file.

Pascal

```
procedure ReadLn(V1); overload;
procedure ReadLn(V1; Vn); overload;
procedure ReadLn(var F: Text; V1); overload;
procedure ReadLn(var F: Text; V1; Vn); overload;
```

C++

```
ReadLn( V1);
ReadLn( V1,  Vn);
ReadLn(Text F,  V1);
ReadLn(Text F,  V1,  Vn);
```

File

System

Description

In Delphi code, the ReadLn procedure reads a line of text and then skips to the next line of the file.

The syntax shown here for ReadLn demonstrates that the procedure can take a variable number of arguments.

ReadLn(F) with no parameters causes the current file position to advance to the beginning of the next line if there is one; otherwise, it goes to the end of the file.

If F is omitted, the global variable Input (see page 3827) is used, which accesses the processes standard input file. Use of Input (see page 3827) in GUI applications raises special issues.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example assumes you are compiling as a console
application.  Notice how the form deactivates when the button
click procedure exits.
}
```

```
{$APPTYPE CONSOLE}

procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  Write('Enter a line of text: ');
  Readln(s);
  Writeln('You typed: ',s);
  Writeln('Hit <Enter> to exit');
  Readln;
end;
{
Use a TOpenDialog to select a file.  Place the first line of
the file into a TEdit.  Click on the text edit to open the
open dialog box.
}
procedure TForm1.Edit1Click(Sender: TObject);
var
  F: TextFile;
  S: string;
begin
  if OpenDialog1.Execute then            { Display Open dialog box }
  begin
    AssignFile(F, OpenDialog1.FileName); { File selected in dialog }
    Reset(F);
    Readln(F, S);                        { Read first line of file }
    Edit1.Text := S;                     { Put string in a TEdit control }
    CloseFile(F);
  end;
end;
```

See Also

Read (see page 3634)

WriteLn (see page 3710)

SetLineBreakStyle (see page 3659)

35.3.119 System.ReallocMem Function

ReallocMem reallocates a memory block.

Pascal

```
procedure ReallocMem(var P: Pointer; Size: Integer);
```

C++

```
ReallocMem(void * P, int Size);
```

File

System

Description

GetMem (see page 3584) allocates a block of the given **Size** on the heap. If you need to change the size of this memory block, call ReallocMem, passing the existing memory block pointer in **P**, and the revised block size in **Size**. If ReallocMem can't expand the memory block pointed to by **P**, it frees the referenced memory and copies the values to the newly allocated memory that is returned.

If there isn't enough memory available to extend the memory block to the desired size, an EOutOfMemory exception is raised.

Note: ReallocMemory (see page 3638) is the C++ compatible version of ReallocMem.

Delphi Examples:

```
{
The following code demostrates the use of ReallocMem function. Three
edit boxes and a button are expected onthe form.
}
function FastStrCat(const S1, S2: String): String;
var
  FinalStr: PChar;
begin
  {
   Allocated enough space in FinalStr to copy the contents
   of the initial string + $00 character
  }
  GetMem(FinalStr, (Length(S1) + 1) * SizeOf(Char));

  { Copy the contents of the first string }
  MoveChars(S1[1], FinalStr^, Length(S1) + 1);

  { And now expand the final string + $00 character }
  ReallocMem(FinalStr, (Length(S1) + Length(S2) + 1) * SizeOf(Char));

  { Copy the contents of the second string }
  MoveChars(S2[1], FinalStr[Length(S1)], Length(S2) + 1);

  { Get the result in String }
  SetString(Result, FinalStr, Length(S1) + Length(S2));
  FreeMem(FinalStr);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  { Concatenate 2 string }
  Edit3.Text := FastStrCat(Edit1.Text, Edit2.Text);
end;
```

See Also

AllocMem (see page 3515)

FreeMem (see page 3577)

FreeMemory (see page 3579)

GetMem (see page 3584)

GetMemory (see page 3585)

EOutOfMemory (see page 3942)

ReallocMemory (see page 3638)

35.3.120 System.ReallocMemory Function

ReallocMemory resizes a memory block.

Pascal

```
function ReallocMemory(P: Pointer; Size: Integer): Pointer; cdecl;
```

C++

```
__cdecl void * ReallocMemory(void * P, int Size);
```

File

System

Description

GetMem (see page 3584) allocates a block of the given **Size** on the heap. If you need to change the size of this memory block, call ReallocMemory, passing the existing memory block pointer in **P**, and the revised block size in **Size**. A pointer to the resized memory block is returned. If ReallocMemory can't expand the memory block pointed to by **P**, it frees the referenced memory and copies the values to the newly allocated memory that is returned.

If there isn't enough memory available to extend the memory block to the desired size, an EOutOfMemory exception is raised.

Note: ReallocMemory is the C++ compatible version of ReallocMem (see page 3637).

C++ Examples:

```
/*
The following code demostrates the use of memory management functions in C++. Three
edit boxes and a button are expected onthe form.
*/
String __fastcall FastStrCat(String s1, String s2)
{
  /*
   Allocated enough space in FinalStr to copy the contents
   of the initial string + $00 character
  */
  wchar_t* finalStr = (wchar_t *)GetMemory((s1.Length() + 1) * sizeof(wchar_t));

  // Copy the contents of the first string
  MoveChars(s1.data(), finalStr, s1.Length() + 1);

  // And now expand the final string + $00 character
  finalStr = (wchar_t *)ReallocMemory(finalStr, (s1.Length() + s2.Length() + 1) *
sizeof(wchar_t));

  // Copy the contents of the second string
  MoveChars(s2.data(), finalStr + s1.Length(), s2.Length() + 1);

  // Get the result in String
  String result = String(finalStr);
  FreeMemory(finalStr);

  return result;
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Concatenate 2 string
  Edit3->Text = FastStrCat(Edit1->Text, Edit2->Text);
}
```

See Also

AllocMem (see page 3515)

FreeMem (see page 3577)

FreeMemory (see page 3579)

GetMem (see page 3584)

GetMemory (see page 3585)

EOutOfMemory (see page 3942)

ReallocMem (see page 3637)

35.3.121 System.RegisterExpectedMemoryLeak Function

Registers an expected memory leak.

Pascal

```
function RegisterExpectedMemoryLeak(P: Pointer): boolean;
```

C++

```
boolean RegisterExpectedMemoryLeak(void * P);
```

File

System

Description

RegisterExpectedMemoryLeak registers a memory location that an application has allocated and does not expect to free.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

UnregisterExpectedMemoryLeak (see page 3694)

ReportMemoryLeaksOnShutdown (see page 3835)

35.3.122 System.RegisterModule Function

Registers the specified module.

Pascal

```
procedure RegisterModule(LibModule: PLibModule);
```

C++

```
RegisterModule(PLibModule LibModule);
```

File

System

Description

This is an internal function and should not be used directly.

See Also

UnregisterModule (see page 3694)

35.3.123 System.Release Function

Pascal

```
procedure Release; deprecated;
```

C++

```
Release();
```

File

System

Description

This is function System.Release.

35.3.124 System.ReleaseExceptionObject Function

Releases exception object acquired by AcquireExceptionObject (see page 3512).

Pascal

```
procedure ReleaseExceptionObject;
```

C++

```
ReleaseExceptionObject();
```

File

System

Description

ReleaseExceptionObject releases an exception object previously acquired by AcquireExceptionObject (see page 3512).

See Also

AcquireExceptionObject (see page 3512)

ExceptObject (see page 3562)

35.3.125 System.RemoveModuleUnloadProc Function

RemoveModuleUnloadProc removes a call to your defined routine when a package is unloaded.

Pascal

```
procedure RemoveModuleUnloadProc(Proc: TModuleUnloadProc); overload;
procedure RemoveModuleUnloadProc(Proc: TModuleUnloadProcLW); overload;
```

C++

```
RemoveModuleUnloadProc(TModuleUnloadProc Proc);
RemoveModuleUnloadProc(TModuleUnloadProcLW Proc);
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your **Proc** routine. In essence, AddModuleUnloadProc (see page 3514) adds your routine to the head of the list of internal routines that Delphi invokes to perform the unload. RemoveModuleUnloadProc reverses the effect of AddModuleUnloadProc (see page 3514).

See Also

AddModuleUnloadProc (see page 3514)

TModuleUnloadProc (see page 3792)

35.3.126 System.Rename Function

Changes the name of an external file.

Pascal

```
procedure Rename(var F; Newname: string); overload;
procedure Rename(var F; Newname: PChar); overload;
```

C++

```
Rename( F, UnicodeString Newname);
Rename( F, const char * Newname);
```

File

System

Description

The external file associated with F is renamed Newname. Further operations on F operate on the external file with the new name.

F is a variable of any file type. Newname is a string-type expression or an expression of type PChar if the extended syntax is enabled.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example requires a button, an open dialog and a save
dialog.  Click the button to rename a file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  f : file;
begin
  OpenDialog1.Title := 'Choose a file... ';
  if OpenDialog1.Execute then
  begin
    SaveDialog1.Title := 'Rename to...';
    if SaveDialog1.Execute then
    begin
```

```
        AssignFile(f, OpenDialog1.FileName);
        Canvas.TextOut(5, 10, 'Renaming ' + OpenDialog1.FileName +
                       ' to ' + SaveDialog1.FileName);
        Rename(f, SaveDialog1.FileName);
      end;
    end;
  end;
```

See Also

Rename File

35.3.127 System.Reset Function

Opens an existing file.

Pascal

```
procedure Reset(var F); overload;
procedure Reset(var F; RecSize: Word); overload;
procedure Reset(var F: File); overload;
procedure Reset(var F: File; RecSize: Word); overload;
```

C++

```
Reset( F);
Reset( F, Word RecSize);
Reset(File F);
Reset(File F, Word RecSize);
```

File

System

Description

In Delphi code, Reset opens the existing external file with the name assigned to F using the mode specified by the global FileMode (see page 3824) variable. An error results if no existing external file of the given name exists or if the file can't be opened with the current file mode. If F is already open, it is first closed and then reopened. The current file position is set to the beginning of the file.

Warning: The default value of FileMode (see page 3824) is 2 (Read (see page 3634)/Write (see page 3707) access). If this is not changed to a read-only file mode before calling Reset, attempts to open read-only files will fail.

F is a variable of any file type associated with an external file using AssignFile (see page 3525). RecSize is an optional expression, which can be specified only if F is an untyped file. If F is an untyped file, RecSize specifies the record size to be used in data transfers. If RecSize is omitted, a default record size of 128 bytes is assumed.

If F is assigned an empty name, such as AssignFile (see page 3525)(F, ''), then after the call to Reset, F refers to the standard input file.

If F is a text file, F becomes read-only.

After a call to Reset, Eof (see page 3558)(F) is true if the file is empty; otherwise, Eof (see page 3558)(F) is false.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
```

```
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example should be executed from a command line
formatted like this: "SystemReset_proj foo.txt".  The file
path is relative to the project directory.
}
function FileIsThere(FileName: string): Boolean;
{ Boolean function that returns True if the file exists; otherwise,
  it returns False. Closes the file if it exists. }
 var
  F: file;
begin
  {$I-}
  AssignFile(F, FileName);
  FileMode := 0;  {Set file access to read only }
  Reset(F);
  CloseFile(F);
  {$I+}
  FileIsThere := (IOResult = 0) and (FileName <> '');
end;  { FileIsThere }

procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
    if FileIsThere(ParamStr(1)) then {Get file name from command line}
      Canvas.TextOut(10, 10, 'File exists')
    else
      Canvas.TextOut(10, 10, 'File not found');
  end;
  {
  Click the button to open a TOpenDialog, then select a file
  to delete.
  }

  procedure TForm1.Button1Click(Sender: TObject);
  var
    F: Textfile;
  begin
    OpenDialog1.Title := 'Delete File';
    if OpenDialog1.Execute then
    begin
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

Append (see page 3520)

AssignFile (see page 3525)

CloseFile (see page 3539)

Rewrite (see page 3645)

Truncate (see page 3685)

FileMode (see page 3824)

35.3.128 System.Rewrite Function

Creates a new file and opens it.

Pascal

```
procedure Rewrite(var F: File); overload;
procedure Rewrite(var F: File; Recsize: Word); overload;
```

C++

```
Rewrite(File F);
Rewrite(File F, Word Recsize);
```

File

System

Description

In Delphi code, Rewrite creates a new external file with the name assigned to F.

F is a variable of any file type associated with an external file using AssignFile (see page 3525). RecSize is an optional expression, which can be specified only if F is an untyped file. If F is an untyped file, RecSize specifies the record size to be used in data transfers. If RecSize is omitted, a default record size of 128 bytes is assumed.

If an external file with the same name already exists, it is deleted and a new empty file is created in its place.

If F is already open, it is first closed and then re-created. The current file position is set to the beginning of the empty file.

If F was assigned an empty name, such as AssignFile (see page 3525)(F,''), then after the call to Rewrite, F refers to the standard output file.

If F is a text file, F becomes write-only.

After calling Rewrite, Eof (see page 3558)(F) is always true.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example reads an entire file into a buffer with one
command and then writes it into a saved file.  Validate
the saved file contents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FromF, ToF: file;
  NumRead, NumWritten: Integer;
  Buf: array[1..2048] of Char;
begin
  if OpenDialog1.Execute then     { Display Open dialog box }
  begin
    AssignFile(FromF, OpenDialog1.FileName);
    Reset(FromF, 1);    { Record size = 1 }
    if SaveDialog1.Execute then      { Display Save dialog box}
    begin
      AssignFile(ToF, SaveDialog1.FileName);    { Open output file }
      Rewrite(ToF, 1);  { Record size = 1 }
      Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))
        + ' bytes...');
      repeat
        System.BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
        BlockWrite(ToF, Buf, NumRead, NumWritten);
      until (NumRead = 0) or (NumWritten <> NumRead);
      // Use CloseFile rather than Close; Close provided for backward compatibility
      CloseFile(FromF);
      CloseFile(ToF);
      Canvas.TextOut(120, 10, ' done.');
    end;
  end;
end;
{
Rewrite example
}

procedure TForm1.Button1Click(Sender: TObject);
var F: TextFile;
begin
  AssignFile(F, 'NEWFILE.$$$');
  Rewrite(F);  // default record size is 128 bytes
```

```
  Writeln(F, 'Just created file with this text in it...');
  CloseFile(F);
  MessageDlg('NEWFILE.$$$ has been created in the ' + GetCurrentDir + ' directory.',
    mtInformation, [mbOk], 0, mbOK);
end;
```

See Also

Append (see page 3520)

AssignFile (see page 3525)

Reset (see page 3643)

Truncate (see page 3685)

35.3.129 System.RmDir Function

Deletes an empty subdirectory.

Pascal

```
procedure RmDir(const S: string); overload;
procedure RmDir(P: PChar); overload;
```

C++

```
RmDir(const UnicodeString S);
RmDir(const char * P);
```

File

System

Description

RmDir removes the subdirectory with the path specified by S or P. If the path does not exist, is non-empty, or is the currently logged directory, an I/O error occurs.

Note: In Delphi, {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example requires a text edit and a button.  Enter the
folder to remove in the text edit and click the button.  The
path is relative to the project folder.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   {$I-}
   { Get directory name from TEdit control }
   RmDir(Edit1.Text);
   if IOResult <> 0 then
     MessageDlg('Cannot remove directory', mtWarning, [mbOk], 0, mbOk)
   else
     MessageDlg('Directory removed', mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

ChDir (see page 3534)

GetDir (see page 3580)

MkDir (see page 3612)

RemoveDir (see page 4232)

35.3.130 System.Round Function

Returns the value of X rounded to the nearest whole number.

Pascal

```
function Round(X: Extended): Int64;
```

C++

```
Int64 Round(Extended X);
```

File

System

Description

In Delphi, the Round function rounds a real-type value to an integer-type value.

X is a real-type expression. Round returns an Int64 value that is the value of X rounded to the nearest whole number. If X is exactly halfway between two whole numbers, the result is always the even number. This method of rounding is often called "Banker's Rounding".

If the rounded value of X is not within the Int64 range, a runtime error is generated, which can be handled using the EInvalidOp exception.

Note: The behavior of Round can be affected by the Set8087CW (see page 3653) procedure or SetRoundMode function.

Delphi Examples:

```
{
This example displays the Cosine of any value between 0 and
12.  Since the scrollbar position is an integer between the
scrollbar  min and max, multiply the range and divide the
position to fine tune it.  With the scrollbar selected use
the left and right buttons to change the position by 1E-6.
Or use the Dec function to decrement (or increment) by a
specific amount.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  new : Integer;
begin
  new := ScrollBar1.Position;
  System.Dec(new, Round(StrToFloat(Edit3.Text)*1000000));
  ScrollBar1.Position := new;
  Edit1.Text := FloatToStr(ScrollBar1.Position/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollBar1.Position/1000000));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Max := Round(Pi * 4000000);   // 4 * Pi
end;
```

```
procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
  var ScrollPos: Integer);
begin
  Edit1.Text := FloatToStr(ScrollPos/1000000);
  Edit2.Text := FloatToStr(Cos(ScrollPos/1000000));
end;
{
This example illustrates the rules for rounding positive and
negative real numbers to integers.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   S, T: string;
begin
   Str(1.4:2:1, T);
   S := T + ' rounds to ' + IntToStr(Round(1.4)) + #13#10;
   Str(1.5:2:1, T);
   S := S + T + ' rounds to ' + IntToStr(Round(1.5)) + #13#10;
   Str(-1.4:2:1, T);
   S := S + T + ' rounds to ' + IntToStr(Round(-1.4)) + #13#10;
   Str(-1.5:2:1, T);
   S := S + T + ' rounds to ' + IntToStr(Round(-1.5));
   MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Int (see page 3602)

Trunc (see page 3684)

Set8087CW (see page 3653)

SetRoundMode

RoundTo

SimpleRoundTo

35.3.131 System.RunError Function

Stops execution and generates a runtime error.

Pascal

```
procedure RunError; overload;
procedure RunError(Errorcode: Byte); overload;
```

C++

```
RunError();
RunError(Byte Errorcode);
```

File

System

Description

In Delphi, the RunError procedure stops program execution by generating a runtime error with the given number at the current statement. Errorcode is the runtime error number (0 if omitted).

Delphi Examples:

```
{
```

```
This example demonstrates how to set a run error, the behavior of
execution and the message that follows.
}
var P: Pointer;

procedure TForm1.Button1Click(Sender: TObject);
begin
{$IFDEF Debug}
  if P = nil then
    RunError(204);
{$ENDIF}
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  P:= nil;
end;
```

See Also

Halt (see page 3594)

35.3.132 System.Seek Function

Moves the current position of a file to a specified component.

Pascal

```
procedure Seek(var F; N: Longint);
```

C++

```
Seek( F, Longint N);
```

File

System

Description

Use Seek in Delphi code to move to a specified position in open typed or untyped files. The current file position of F moves to component number N, where the number of the first component of a file is 0.

In the above syntax, F is a typed or untyped file variable. The file specified by F must be open. N is an expression of type Longint

To expand a file, seek one component beyond the last component; that is, the statement Seek(F, FileSize (see page 3569)(F)) moves the current file position to the end of the file.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example displays a open dialog when the user clicks a
button. When the user selects the file, it reports the size
of the file, seeks to halfway through the file, and reports
the file position at that point.
Note:   This example assumes the compiler flag $IOCHECKS ON
($I+). That is, it expects errors to generate exceptions
rather than checking IOResult to be sure that I/O routines
succeed.
}
procedure TForm1.Button1Click(Sender: TObject);
```

```
var
   f: file of Byte;
   size: Longint;
   S: string;
   y: Integer;
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(f, OpenDialog1.FileName);
    Reset(f);
    try
      size := FileSize(f);
      S := 'File size in bytes: ' + IntToStr(size);
      y := 10;
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      S := 'Seeking halfway into file...';
      Canvas.TextOut(5, y, S);
      y := y + Canvas.TextHeight(S) + 5;
      Seek(f, size div 2);
      S := 'Position is now ' + IntToStr(FilePos(f));
      Canvas.TextOut(5, y, S);
    finally
      CloseFile(f);
    end;
  end;
end;
```

See Also

FilePos (see page 3567)

35.3.133 System.SeekEof Function

Returns the end-of-file status of a file, ignoring whitespace.

Pascal

```
function SeekEof: Boolean; overload;
function SeekEof(var F: Text): Boolean; overload;
```

C++

```
Boolean SeekEof();
Boolean SeekEof(Text F);
```

File

System

Description

In Delphi code, call SeekEof to determine whether there is only whitespace between the file pointer and the end of a file. SeekEof moves the file pointer past any whitespace, leaving it positioned at the end of a file or on the next non-whitespace character, whichever comes first. If it leaves the file pointer at the end of the file, SeekEof returns true. Otherwise, it returns false.

SeekEof can only be used on open text files.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
This example creates a file TEST.TXT, puts some text in it
and then reads the file to EOF.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f : System.TextFile;
   j, Y : Integer;
 begin
   AssignFile(f,'TEST.TXT');
   Rewrite(f);
   { Create a file with 8 numbers and some
     whitespace at the ends of the lines }
   Writeln(f,'1 2 3 4 ');
   Writeln(f,'5 6 7 8 ');
   Reset(f);
   { Read the numbers back. SeekEof returns TRUE if there is no
     more text (other than whitespace) in the file. }
   Y := 5;
   while not SeekEof(f) do
   begin
     Read(f,j);
     Canvas.TextOut(5, Y, IntToStr(j));
     Y := Y + Canvas.TextHeight(IntToStr(j)) + 5;
   end;
 end;
```

See Also

Eof (see page 3558)

SeekEoln (see page 3652)

SetLineBreakStyle (see page 3659)

35.3.134 System.SeekEoln Function

Returns the end-of-line status of a file, ignoring whitespace.

Pascal

```
function SeekEoln: Boolean; overload;
function SeekEoln(var F: Text): Boolean; overload;
```

C++

```
Boolean SeekEoln();
Boolean SeekEoln(Text F);
```

File

System

Description

In Delphi code, call SeekEoln to determine whether there is only whitespace between the file pointer and the end of the current line. SeekEoln moves the file pointer past any whitespace, leaving it positioned at the end of the current line, the end of the file, or on the next non-whitespace character (whichever comes first). If SeekEoln leaves the file pointer at the end of the current line or at the end of the file, it returns true. Otherwise, it returns false.

SeekEoln can be used only on open text files.

Note: {$I+} handles runtime errors using exceptions. When using {$I-}, use IOResult (see page 3602) to check for I/O errors.

Delphi Examples:

```
{
Create data with white space at beginning and end of line to
demonstrate that SeekEoln is immune to this.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: System.TextFile;
   j, Y: Integer;
   s: string;
begin
   AssignFile(f,'TEST.TXT');
   Rewrite(f);

   Writeln(f, '    1 2 3 4      ');
   Writeln(f, 'Some text');

   Reset(f);
   Y := 5;

   while not SeekEoln(f) do // Will skip white space
   begin
     Read(f, j);
     Canvas.TextOut(5, Y, IntToStr(j));
     Y := Y + Canvas.TextHeight(IntToStr(j)) + 5;
   end;
   Readln(f); // Read line-end marker

   Readln(f, s); // Read string with line-end marker
   Canvas.TextOut(5, Y, s);
end;
```

See Also

Eoln (see page 3559)

SeekEof (see page 3651)

SetLineBreakStyle (see page 3659)

35.3.135 System.Set8087CW Function

Sets both the control word in the floating point unit and the variable NoErrMsg (see page 3832) declared in the system unit.

Pascal

```
procedure Set8087CW(NewCW: Word);
```

C++

```
Set8087CW(Word NewCW);
```

File

System

Description

The floating-point unit control word controls the precision of floating point calculations, the rounding mode, and whether certain floating-point operations trigger exceptions. See Intel's processor documentation for details.

This routine allows the programmer to have direct access to the CW. Be aware that using this routine to change the value of the

8087CW will change the behavior of the program's FP calculations. It is the programmer's responsibility to reset it.

It is recommended that you disable all floating-point exceptions when using OpenGL to render 3D graphics. To do this, call Set8087CW(0x133f) in your main form's OnCreate event before calling any OpenGL functions.

C++ Examples:

```
/*
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
*/
Word Saved8087CW;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit3->Text = FloatToStr(StrToFloat(Edit1->Text) / StrToFloat(Edit2->Text));
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "FPU Exceptions")
    System::Set8087CW(Saved8087CW);
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "No FPU Exceptions")
    System::Set8087CW(0x133f); // Disable all fpu exceptions
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("No FPU Exceptions");
  RadioGroup1->Items->Add("FPU Exceptions");
  RadioGroup1->ItemIndex = 2;
  Saved8087CW = Default8087CW;  // Save this because Set8087CW changes it!
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  System::Set8087CW(Saved8087CW); // Default value (with exceptions) is 0x1372
}
```

Delphi Examples:

```
{
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
}
var
  Form1: TForm1;
  Saved8087CW: Word;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit3.Text := FloatToStr(StrToFloat(Edit1.Text) / StrToFloat(Edit2.Text));
end;

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'FPU Exceptions' then
    System.Set8087CW(Saved8087CW);
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'No FPU Exceptions' then
    System.Set8087CW($133f); { Disable all fpu exceptions }
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('No FPU Exceptions');
  RadioGroup1.Items.Add('FPU Exceptions');
  RadioGroup1.ItemIndex := 2;
  Saved8087CW := Default8087CW;  // Save this because Set8087CW changes it!
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  System.Set8087CW(Saved8087CW); // Default value (with exceptions) is $1372
end;
```

See Also

NoErrMsg (see page 3832)

SetExceptionMask

SetPrecisionMode

SetRoundMode

35.3.136 System.SetCodePage Function

Pascal

```
procedure SetCodePage(var S: RawByteString; CodePage: Word; Convert: Boolean = True);
```

C++

```
SetCodePage(RawByteString S, Word CodePage, Boolean Convert = True);
```

File

System

Description

This is function System.SetCodePage.

35.3.137 System.SetInOutRes Function

Pascal

```
procedure SetInOutRes(NewValue: Integer);
```

C++

```
SetInOutRes(int NewValue);
```

File

System

Description

This is function System.SetInOutRes.

35.3.138 System.SetLastError Function

SetLastError sets the last API error code.

Pascal

```
procedure SetLastError(ErrorCode: Integer); stdcall;
```

C++

```
__stdcall SetLastError(int ErrorCode);
```

File

System

Description

GetLastError (see page 3582) returns the last API error code encountered. It then resets it to 0. SetLastError forces the API error code to the specified **ErrorCode** value. A call to GetLastError (see page 3582) will then return this value before again resetting the error code to 0.

C++ Examples:

```
//
//This code demonstrates the use of system error codes
//and how to use transform them into C++ exceptions.
//
void __fastcall TForm2::btRaiseLastClick(TObject *Sender)
{
    /* Set the last OS error to a bogus value */
    System::SetLastError(ERROR_ACCESS_DENIED);

    try
    {
        RaiseLastOSError();
    }
    catch(EOSError* ex)
    {
        MessageDlg(String("Caught an OS error with code: ") +
            IntToStr((int)ex->ErrorCode), mtError,
            TMsgDlgButtons() << mbOK, 0);
    }


    /* Let the Delphi's Exception dialog pop-up */
    RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

    /* Finally set the last error to none */
    System::SetLastError(ERROR_SUCCESS);

    if (GetLastError() != ERROR_SUCCESS)
        MessageDlg(String("Whoops, something went wrong ") +
          " in the mean time!", mtError, TMsgDlgButtons() << mbOK, 0);

    /* No exception should be thrown here */
    RaiseLastOSError();
}
```

Delphi Examples:

```
{
This code demonstrates the use of system error codes
and how to transform them into Delphi exceptions.
```

```
}
procedure TForm2.btRaiseLastClick(Sender: TObject);
begin
  { Set the last OS error to a bogus value }
  System.SetLastError(ERROR_ACCESS_DENIED);

  try
    RaiseLastOSError();
  except
    on Ex : EOSError do
       MessageDlg('Caught an OS error with code: ' +
          IntToStr(Ex.ErrorCode), mtError, [mbOK], 0);
  end;

  { Let the Delphi's Exception dialog pop-up }
  RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

  { Finally set the last error to none }
  System.SetLastError(ERROR_SUCCESS);

  if GetLastError() <> ERROR_SUCCESS then
     MessageDlg('Whoops, something went wrong in the ' +
                'mean time!', mtError, [mbOK], 0);

  { No exception should be thrown here }
  RaiseLastOSError();
end;
```

See Also

GetLastError (see page 3582)

35.3.139 System.SetLength Function

Sets the length of a string or dynamic-array variable.

Pascal

```
procedure SetLength(var S; NewLength: Integer);
```

C++

```
SetLength( S, int NewLength);
```

File

System

Description

S is a Delphi string or dynamic-array variable.

NewLength is the new number of characters or elements in S.

For a short-string variable, SetLength simply sets the length-indicator character (the character at S[0]) to the given value. In this case, NewLength must be a value between 0 and 255.

For a long-string or dynamic-array variable, SetLength reallocates the string or array referenced by S to the given length. Existing characters in the string or elements in the array are preserved, but the content of newly allocated space is undefined. The one exception is when increasing the length of a dynamic array in which the elements are types that must be initialized (strings, Variants (see page 4973), Variant (see page 3421) arrays, or records that contain such types). When S is a dynamic array of types that must be initialized, newly allocated space is set to 0 or nil.

For dynamic arrays, SetLength may take more than one length parameter (up to the number of array dimensions). Each parameter specifies the number of elements along a particular dimension.

Following a call to SetLength, S is guaranteed to reference a unique string or array—that is, a string or array with a reference count of one. If there is not enough memory available to reallocate the variable, SetLength raises an EOutOfMemory exception.

Delphi Examples:

```
{
This example determines the encoding of a source file and then
writes the file in any encoding chosen.  Use this project to
create files of any encoding.  This example requires two text edits
and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
  LOffset: Integer;
  LEncoding, DestEncoding: TEncoding;
  LFileStream: TFileStream;
  EncodingArray: array[0..5] of TEncoding;
begin
  LEncoding:= nil;
  EncodingArray[0]:= TEncoding.UTF8;
  EncodingArray[1]:= TEncoding.UTF7;
  EncodingArray[2]:= TEncoding.Unicode;
  EncodingArray[3]:= TEncoding.Default;
  EncodingArray[4]:= TEncoding.BigEndianUnicode;
  EncodingArray[5]:= TEncoding.ASCII;
  DestEncoding := EncodingArray[ComboBox1.ItemIndex];
  LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    // Read file into buffer
    SetLength(LBuffer, LFileStream.Size);
//    LFileStream.Read(LBuffer[0], Length(LBuffer));
    LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));

    // Identify encoding and convert buffer to UTF8
    LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
//    for I := 0 to Length(LBuffer) - 1 do
//      Memo1.Lines.Add((LBuffer[I]));
    LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
      LOffset, Length(LBuffer) - LOffset);
  finally
    LFileStream.Free;
  end;

  LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
  try
    // Write an encoding byte order mark and buffer to output file
    LByteOrderMark := DestEncoding.GetPreamble;
    LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
    LFileStream.Write(LBuffer[0], Length(LBuffer));
  finally
    LFileStream.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  StringList: TStrings;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
```

```
        Add('TEncoding.UTF8');
        Add('TEncoding.UTF7');
        Add('TEncoding.Unicode');
        Add('TEncoding.Default');
        Add('TEncoding.BigEndianUnicode');
        Add('TEncoding.ASCII');
      end;
      with ComboBox1 do begin
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

EOutOfMemory (see page 3942)

35.3.140 System.SetLineBreakStyle Function

Determines the end-of-line and end-of-file conventions for text file I/O.

Pascal

```
procedure SetLineBreakStyle(var T: Text; Style: TTextLineBreakStyle);
```

C++

```
SetLineBreakStyle(Text T, TTextLineBreakStyle Style);
```

File

System

Description

In Delphi, call SetLineBreakStyle to determine how text file routines find or mark the end of a line and the end of a file. Affected routines include Read (see page 3634), Write (see page 3707), ReadLn (see page 3636), WriteLn (see page 3710), Eof (see page 3558), Eoln (see page 3559), SeekEof (see page 3651), and SeekEoln (see page 3652).

T is the text file whose line break style is altered. It must be closed, or SetLineBreakStyle generates a runtime error.

Style indicates the style to use. When Style is tlbsLF, end-of-line is marked using only a linefeed character. When Style is tlbsCRLF, end-of-line is marked using a carriage return and linefeed.

See Also

ReadLn (see page 3636)

WriteLn (see page 3710)

Read (see page 3634)

Write (see page 3707)

Eof (see page 3558)

Eoln (see page 3559)

SeekEof (see page 3651)

SeekEoln (see page 3652)

DefaultTextLineBreakStyle (see page 3815)

35.3.141 System.SetMemoryManager Function

Sets entry points of the memory manager.

Pascal

```
procedure SetMemoryManager(const MemMgr: TMemoryManager); overload; deprecated;
procedure SetMemoryManager(const MemMgrEx: TMemoryManagerEx); overload;
```

C++

```
SetMemoryManager(const TMemoryManager MemMgr);
SetMemoryManager(const TMemoryManagerEx MemMgrEx);
```

File

System

Description

SetMemoryManager sets the entry points of the memory manager to the three functions encapsulated by the MemMgr parameter. Use SetMemoryManager to change the memory allocation routines that are used for:

Allocation and deallocation of dynamic strings and dynamic arrays.

The GetMem (see page 3584), ReallocMem (see page 3637), and FreeMem (see page 3577) standard procedures (Delphi only)

The New (see page 3619) and Dispose (see page 3553) standard procedures (Delphi only).

C++ Examples:

```
/*
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
*/

bool isMyMemMgr;
TMemoryManagerEx oldMemMgr;
volatile long getMemCalls;
volatile long freeMemCalls;
volatile long reallocMemCalls;
volatile long allocMemCalls;

void* __fastcall myGetMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&getMemCalls);

    //route the call
    return oldMemMgr.GetMem(size);
}

int __fastcall myFreeMem(void* p)
{
    //safely increment the counter
    InterlockedIncrement(&freeMemCalls);

    //route the call
    return oldMemMgr.FreeMem(p);
}
```

```
void* __fastcall myReallocMem(void* p, int size)
{
    //safely increment the counter
    InterlockedIncrement(&reallocMemCalls);

    //route the call
    return oldMemMgr.ReallocMem(p, size);
}

void* __fastcall myAllocMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&allocMemCalls);

    //route the call
    return oldMemMgr.AllocMem(size);
}

void __fastcall TForm3::btUseSysMemMgrClick(TObject *Sender)
{
    //switch button states
    btUseSysMemMgr->Enabled = false;
    btUseMyMemMgr->Enabled = true;

    //set the old memory manager back
    SetMemoryManager(oldMemMgr);

    isMyMemMgr = false;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;
}

void __fastcall TForm3::FormDestroy(TObject *Sender)
{
    //set the old memory manager back
    if (isMyMemMgr)
    {
        SetMemoryManager(oldMemMgr);
    }
}

void __fastcall TForm3::Timer1Timer(TObject *Sender)
{
    //Note that IntToStr calls will also Allocate/Free
    //memory so at each timer tick the counts will be +4;

    Form3->edGetMem->Text = IntToStr((int)getMemCalls);
    Form3->edFreeMem->Text = IntToStr((int)freeMemCalls);
    Form3->edReallocMem->Text = IntToStr((int)reallocMemCalls);
    Form3->edAllocMem->Text = IntToStr((int)allocMemCalls);
}

void __fastcall TForm3::btUseMyMemMgrClick(TObject *Sender)
{
    TMemoryManagerEx myMemMgr;

    //switch button states
    btUseSysMemMgr->Enabled = true;
    btUseMyMemMgr->Enabled = false;

    //get the old memory manager
```

```
    GetMemoryManager(oldMemMgr);

    //create out instance
    myMemMgr.GetMem = myGetMem;
    myMemMgr.FreeMem = myFreeMem;
    myMemMgr.AllocMem = myAllocMem;
    myMemMgr.ReallocMem = myReallocMem;

    //use defaults for this - not important
    myMemMgr.RegisterExpectedMemoryLeak = oldMemMgr.RegisterExpectedMemoryLeak;
    myMemMgr.UnregisterExpectedMemoryLeak = oldMemMgr.UnregisterExpectedMemoryLeak;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;

    //install the new memory manager
    SetMemoryManager(myMemMgr);

    isMyMemMgr = true;
}

void __fastcall TForm3::btDoSomethingClick(TObject *Sender)
{
    void *ptr;
    //perform some new and delete
    for (int i=0; i < 1000; ++i)
    {
        ptr = GetMemory(4);
        FreeMemory(ptr);
    }
}
```

Delphi Examples:

```
{
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
}
var
  OldMemMgr       : TMemoryManagerEx;
  GetMemCalls     : Integer;
  FreeMemCalls    : Integer;
  ReallocMemCalls : Integer;
  AllocMemCalls   : Integer;

function MyGetMem(Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.GetMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(GetMemCalls);
end;

function MyFreeMem(P: Pointer): Integer;
begin
  { Route the call }
  Result := OldMemMgr.FreeMem(P);

  { Safely increment the counter }
  InterlockedIncrement(FreeMemCalls);
end;
```

```
function MyReallocMem(P: Pointer; Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.ReallocMem(P, Size);

  { Safely increment the counter }
  InterlockedIncrement(ReallocMemCalls);
end;

function MyAllocMem(Size: Cardinal): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.AllocMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(AllocMemCalls);
end;

procedure TForm2.btPerformSomethingClick(Sender: TObject);
var
  X : Integer;
  I : ^Integer;
begin
  { Perform some allocations and frees }
  for X := 0 to 999 do
  begin
    New(I);
    Dispose(I);
  end;
end;

procedure TForm2.btUseMyMemMgrClick(Sender: TObject);
var
  MyMemMgr : TMemoryManagerEx;
begin
  { Switch button states }
  btUseSystemMemMgr.Enabled := true;
  btUseMyMemMgr.Enabled := false;

  { Get the old memory manager }
  GetMemoryManager(OldMemMgr);

  { Create our instance }
  MyMemMgr.GetMem := MyGetMem;
  MyMemMgr.FreeMem := MyFreeMem;
  MyMemMgr.ReallocMem := MyReallocMem;
  MyMemMgr.AllocMem := MyAllocMem;

  { Use the defaults for this - not important }
  MyMemMgr.RegisterExpectedMemoryLeak := OldMemMgr.RegisterExpectedMemoryLeak;
  MyMemMgr.UnregisterExpectedMemoryLeak := OldMemMgr.UnregisterExpectedMemoryLeak;

  { Clear out the count variables }
  GetMemCalls := 0;
  FreeMemCalls := 0;
  ReallocMemCalls := 0;
  AllocMemCalls := 0;

  { Install the new memory manager }
  SetMemoryManager(MyMemMgr);
end;

procedure TForm2.btUseSystemMemMgrClick(Sender: TObject);
begin
  { Switch button states }
  btUseSystemMemMgr.Enabled := false;
```

```
    btUseMyMemMgr.Enabled := true;

    { Set the old memory manager back! }
    SetMemoryManager(OldMemMgr);

    GetMemCalls := 0;
    FreeMemCalls := 0;
    ReallocMemCalls := 0;
    AllocMemCalls := 0;
  end;

  procedure TForm2.FormDestroy(Sender: TObject);
  begin
    { Set the old memory manager back }
    if IsMemoryManagerSet then
       SetMemoryManager(OldMemMgr);
  end;

  procedure TForm2.Timer1Timer(Sender: TObject);
  begin
    {
    Note that IntToStr calls will also Allocate/Free
    memory so at each timer tick the counts will be increased;
    }
    Form2.edtGetMemCalls.Text := IntToStr(GetMemCalls);
    Form2.edtFreeMemCalls.Text := IntToStr(FreeMemCalls);
    Form2.edtReallocMemCalls.Text := IntToStr(ReallocMemCalls);
    Form2.edtAllocMemCalls.Text := IntToStr(AllocMemCalls);
  end;
```

See Also

GetMemoryManager (see page 3587)

GetMem (see page 3584)

ReallocMem (see page 3637)

FreeMem (see page 3577)

New (see page 3619)

Dispose (see page 3553)

IsMemoryManagerSet (see page 3604)

35.3.142 System.SetMinimumBlockAlignment Function

Sets the minimum memory block alignment.

Pascal

```
procedure SetMinimumBlockAlignment(AMinimumBlockAlignment: TMinimumBlockAlignment);
```

C++

```
SetMinimumBlockAlignment(TMinimumBlockAlignment AMinimumBlockAlignment);
```

File

System

Description

SetMinimumBlockAlignment sets the Memory Manager's minimum block alignment. Existing allocations are not affected if this

setting is changed. The memory manager will enforce the new setting for all allocations made after the change.

See Also

Memory Management

Configuring the Memory Manager

35.3.143 System.SetMultiByteConversionCodePage Function

Sets the code page used internally whenever multibyte characters are converted to/from WideChar.

Pascal

```
procedure SetMultiByteConversionCodePage(CodePage: Integer);
```

C++

```
SetMultiByteConversionCodePage(int CodePage);
```

File

System

Description

SetMultiByteConversionCodePage sets the code page used internally whenever multibyte characters are converted to or from WideChar. The default is the active system code page. This function allows you to override the default.

See Also

Multibyte Character Sets (MBCS)

strlen

35.3.144 System.SetRaiseList Function

Pascal

```
function SetRaiseList(NewPtr: Pointer): Pointer; deprecated;
```

C++

```
void * SetRaiseList(void * NewPtr);
```

File

System

Description

This is function System.SetRaiseList.

35.3.145 System.SetString Function

Sets the contents and length of the given string.

Pascal

```
procedure SetString(var s: string; buffer: PChar; len: Integer);
```

C++

```
SetString(UnicodeString s, const char * buffer, int len);
```

File

System

Description

In Delphi code, SetString sets the contents and length of the given string variable to the block of characters given by the Buffer and Len parameters.

For a short string variable, SetString sets the length indicator character (the character at **S**[0]) to the value given by **Len** and then, if the **Buffer** parameter is not nil, copies **Len** characters from **Buffer** into the string starting at **S**[1]. For a short string variable, the **Len** parameter must be a value between 0 and 255.

For a long string variable, SetString sets **S** to reference a newly allocated string of the given length. If the **Buffer** parameter is not nil, SetString then copies **Len** characters from **Buffer** into the string; otherwise, the contents of the new string is left uninitialized. If there is not enough memory available to create the string, an EOutOfMemory exception is raised. Following a call to SetString, **S** is guaranteed to reference a unique string, that is a string with a reference count of one.

Delphi Examples:

```
{
The following code demostrates the use of ReallocMem function. Three
edit boxes and a button are expected onthe form.
}
function FastStrCat(const S1, S2: String): String;
var
  FinalStr: PChar;
begin
  {
   Allocated enough space in FinalStr to copy the contents
   of the initial string + $00 character
  }
  GetMem(FinalStr, (Length(S1) + 1) * SizeOf(Char));

  { Copy the contents of the first string }
  MoveChars(S1[1], FinalStr^, Length(S1) + 1);

  { And now expand the final string + $00 character }
  ReallocMem(FinalStr, (Length(S1) + Length(S2) + 1) * SizeOf(Char));

  { Copy the contents of the second string }
  MoveChars(S2[1], FinalStr[Length(S1)], Length(S2) + 1);

  { Get the result in String }
  SetString(Result, FinalStr, Length(S1) + Length(S2));
  FreeMem(FinalStr);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  { Concatenate 2 string }
  Edit3.Text := FastStrCat(Edit1.Text, Edit2.Text);
end;
```

See Also

EOutOfMemory (see page 3942)

35.3.146 System.SetTextBuf Function

Assigns an I/O buffer to a text file.

Pascal

```
procedure SetTextBuf(var F: Text; var Buf); overload;
procedure SetTextBuf(var F: Text; var Buf; Size: Integer); overload;
```

C++

```
SetTextBuf(Text F,  Buf);
SetTextBuf(Text F,  Buf, int Size);
```

File

System

Description

In Delphi code, SetTextBuf changes the text file **F** to use the buffer specified by **Buf** instead of F's internal buffer. F is a text file variable, Buf is any variable, and Size is an optional expression.

Each Text file variable has an internal 128-byte buffer that buffers Read (see page 3634) and Write (see page 3707) operations. This buffer is adequate for most operations. However, heavily I/O-bound programs benefit from a larger buffer to reduce disk head movement and file system overhead.

Size specifies the size of the buffer in bytes. If Size is omitted, SizeOf (see page 3670)(Buf) is assumed. The new buffer remains in effect until F is next passed to AssignFile (see page 3525).

SetTextBuf can be called immediately after Reset (see page 3643), Rewrite (see page 3645), and Append (see page 3520), but never apply it to an open file.

When SetTextBuf is called on an open file once I/O operations have taken place, data could be lost because of the change of buffer.

The Delphi runtime library does not ensure that the buffer exists for the entire duration of I/O operations on the file. A common error is to install a local variable as a buffer, then use the file outside the procedure that declared the buffer.

Delphi Examples:

```
{
This example reads a file into a buffer and then dumps
the buffer into a new file WOOF.DOG
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F, FTwo: System.TextFile;
  Ch: Char;
  Buf: array[1..4096] of Char;  { 4K buffer }
begin
  if OpenDialog1.Execute then
  begin
    AssignFile(F, OpenDialog1.FileName);
    { Larger buffer for faster reads }
    System.SetTextBuf(F, Buf);
    Reset(F);
    { Dump text file into another file }
    AssignFile(FTwo, 'WOOF.DOG');
    Rewrite(FTwo);
    while not Eof(f) do
    begin
```

```
      Read(F, Ch);
      Write(FTwo, Ch);
    end;
    System.CloseFile(F);
    System.CloseFile(FTwo);
  end;
```

See Also

Append (see page 3520)

AssignFile (see page 3525)

Read (see page 3634)

Reset (see page 3643)

Rewrite (see page 3645)

SizeOf (see page 3670)

Write (see page 3707)

35.3.147 System.SetVariantManager Function

Sets the entry points of routines that define variant behavior.

Pascal

```
procedure SetVariantManager(const VarMgr: TVariantManager); deprecated;
```

C++

```
SetVariantManager(const TVariantManager VarMgr);
```

File

System

Description

SetVariantManager sets the entry points of the routines that perform variant type conversion and define other variant behavior.

Note: Although System.hpp defines the SetVariantManager procedure, do not use this routine in C++ code. Customized Variant (see page 3421) support is not fully supported in C++.

See Also

GetVariantManager (see page 3594)

IsVariantManagerSet (see page 3606)

35.3.148 System.ShareMemoryManager Function

Shares the module's memory manager.

Pascal

```
function ShareMemoryManager: Boolean;
```

C++

```
Boolean ShareMemoryManager();
```

File

System

Description

ShareMemoryManager makes the module's memory manager available for sharing to other modules in the same process. Only one memory manager may be shared per process. It returns **False** if another module in the process is already sharing its memory manager.

Note: This function is not thread safe.

See Also

Memory Management

Configuring the Memory Manager

Sharing Memory

35.3.149 System.Sin Function

Returns the sine of the angle in radians.

Pascal

```
function Sin(const X: Extended): Extended;
```

C++

```
Extended Sin(const Extended X);
```

File

System

Description

In Delphi code, the Sin function returns the sine of the argument.

X is a real-type expression. Sin returns the sine of the angle X in radians.

Delphi Examples:

```
{
This example uses the Sine trig function to compute
the sine of Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  R: Extended;
  S: string;
begin
  R := Sin(Pi);
  Str(R:5:3, S);
  Canvas.TextOut(10, 10, 'The Sin of Pi is ' + S);
end;
```

See Also

ArcTan (see page 3521)

Cos (see page 3549)

35.3.150 System.SizeOf Function

Returns the number of bytes occupied by a variable or type.

Pascal

```
function SizeOf(X): Integer;
```

C++

```
int SizeOf( X);
```

File

System

Description

Pass a Delphi variable reference to SizeOf to determine the number of bytes used to represent the variable. Pass a type identifier to SizeOf to determine the number of bytes used to represent instances of that type. SizeOf is useful for determining the amount of memory to specify for the FillChar (see page 3570), Move (see page 3617), or GetMem (see page 3584) procedures.

SizeOf returns 0 when the argument is an untyped variable.

Delphi Examples:

```
{
This example fills a char array with characters and then
assigns the array to a text edit.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: array[0..79] of AnsiChar;
begin
  { Set to all Xs }
  FillChar(S, SizeOf(S), Ord('X'));
  Edit1.Text := S;
end;
{
This example makes a record.  Reports the size of the reocrd,
fills it and reports the contents.
}
type
  CustRec = record
    Name: string[30];
    Phone: string[14];
  end;

procedure TForm1.Button1Click(Sender: TObject);
 var
  P: ^CustRec;
  size: Integer;
begin
  GetMem(P, SizeOf(CustRec));
  Canvas.TextOut(10, 10, 'The size of the record is ' + IntToStr(SizeOf(CustRec)));
  P.Name := 'a12345678901234567890123456789';
  P.Phone := '9,1,4085551212';
  Canvas.TextOut(10, 40, 'Name was assigned this string: ' + P.Name);
  Canvas.TextOut(10, 70, 'Phone was assigned this string: ' + P.Phone);
  size := SizeOf(CustRec);
```

```
  FreeMem (P, size);
end;
```

See Also

FillChar (see page 3570)

GetMem (see page 3584)

Move (see page 3617)

35.3.151 System.Slice Function

Returns a sub-section of an array.

Pascal

```
function Slice(var A: array; Count: Integer): array;
```

C++

```
array Slice(array A, int Count);
```

File

System

Description

In Delphi code, use Slice to select a sub-section (slice) of an array to be passed as an open array parameter. Slice is only allowed as a parameter in a call to a procedure or function that expects an open array parameter. Slice selects the first **Count** elements of the array given by **A** to be passed as an open array parameter. A can be an open array parameter itself, allowing dynamically allocated arrays to be passed to open array parameters.

Delphi Examples:

```
{
This example demonstrates the use and the need for the TThread
Synchronize method by executing three sorting routines in three
separate threads with and without synchronizaiton.
This example is a snippet from the Threads demo in the the
Demos directory of your RAD Studio installation.  Please refer
to that demo for the remaining support code and all design
requirements. The UseSynchronize feature in not implemented
in the demo, but all the changes you need are here.
}
type

{ TSortThread }

  PThreadSortArray = ^TThreadSortArray;
  TThreadSortArray = array[0..MaxInt div SizeOf(Integer) - 1] of Integer;

  TSortThread = class(TThread)
  private
    FBox: TPaintBox;
    FSortArray: PThreadSortArray;
    FSize: Integer;
    FA, FB, FI, FJ: Integer;
    procedure DoVisualSwap;
  protected
    procedure Execute; override;
    procedure VisualSwap(A, B, I, J: Integer);
    procedure Sort(var A: array of Integer); virtual; abstract;
```

```
    public
      constructor Create(Box: TPaintBox; var SortArray: array of Integer);
    end;

  { TBubbleSort }

    TBubbleSort = class(TSortThread)
    protected
      procedure Sort(var A: array of Integer); override;
    end;

  { TSelectionSort }

    TSelectionSort = class(TSortThread)
    protected
      procedure Sort(var A: array of Integer); override;
    end;

  { TQuickSort }

    TQuickSort = class(TSortThread)
    protected
      procedure Sort(var A: array of Integer); override;
    end;

  procedure PaintLine(Canvas: TCanvas; I, Len: Integer);

  var UseSynchronize: Boolean;

  implementation

  procedure PaintLine(Canvas: TCanvas; I, Len: Integer);
  begin
    Canvas.PolyLine([Point(0, I * 2 + 1), Point(Len, I * 2 + 1)]);
  end;

  { TSortThread }

  constructor TSortThread.Create(Box: TPaintBox; var SortArray: array of Integer);
  begin
    FBox := Box;
    FSortArray := @SortArray;
    FSize := High(SortArray) - Low(SortArray) + 1;
    FreeOnTerminate := True;
    inherited Create(False);
  end;

  { Since DoVisualSwap uses a VCL component (i.e., the TPaintBox) it should never
    be called directly by this thread.  DoVisualSwap should be called by passing
    it to the Synchronize method which causes DoVisualSwap to be executed by the
    main VCL thread, avoiding multi-thread conflicts. See VisualSwap for an
    example of calling Synchronize. }

  procedure TSortThread.DoVisualSwap;
  begin
    with FBox do
    begin
      Canvas.Pen.Color := clBtnFace;
      PaintLine(Canvas, FI, FA);
      PaintLine(Canvas, FJ, FB);
      Canvas.Pen.Color := clRed;
      PaintLine(Canvas, FI, FB);
      PaintLine(Canvas, FJ, FA);
    end;
  end;
```

```
{ VisusalSwap is a wrapper on DoVisualSwap making it easier to use.  The
  parameters are copied to instance variables so they are accessable
  by the main VCL thread when it executes DoVisualSwap }

procedure TSortThread.VisualSwap(A, B, I, J: Integer);
begin
  FA := A;
  FB := B;
  FI := I;
  FJ := J;
  if (UseSynchronize) then Synchronize(DoVisualSwap)
  else DoVisualSwap;
end;

{ The Execute method is called when the thread starts }

procedure TSortThread.Execute;
begin
  Sort(Slice(FSortArray^, FSize));
end;

{ TBubbleSort }

procedure TBubbleSort.Sort(var A: array of Integer);
var
  I, J, T: Integer;
begin
  for I := High(A) downto Low(A) do
    for J := Low(A) to High(A) - 1 do
      if A[J] > A[J + 1] then
      begin
        VisualSwap(A[J], A[J + 1], J, J + 1);
        T := A[J];
        A[J] := A[J + 1];
        A[J + 1] := T;
        if Terminated then Exit;
      end;
end;

{ TSelectionSort }

procedure TSelectionSort.Sort(var A: array of Integer);
var
  I, J, T: Integer;
begin
  for I := Low(A) to High(A) - 1 do
    for J := High(A) downto I + 1 do
      if A[I] > A[J] then
      begin
        VisualSwap(A[I], A[J], I, J);
        T := A[I];
        A[I] := A[J];
        A[J] := T;
        if Terminated then Exit;
      end;
end;

{ TQuickSort }

procedure TQuickSort.Sort(var A: array of Integer);

  procedure QuickSort(var A: array of Integer; iLo, iHi: Integer);
  var
    Lo, Hi, Mid, T: Integer;
  begin
    Lo := iLo;
```

```
      Hi := iHi;
      Mid := A[(Lo + Hi) div 2];
      repeat
        while A[Lo] < Mid do Inc(Lo);
        while A[Hi] > Mid do Dec(Hi);
        if Lo <= Hi then
        begin
          VisualSwap(A[Lo], A[Hi], Lo, Hi);
          T := A[Lo];
          A[Lo] := A[Hi];
          A[Hi] := T;
          Inc(Lo);
          Dec(Hi);
        end;
      until Lo > Hi;
      if Hi > iLo then QuickSort(A, iLo, Hi);
      if Lo < iHi then QuickSort(A, Lo, iHi);
      if Terminated then Exit;
    end;

  begin
    QuickSort(A, Low(A), High(A));
  end;
```

See Also

SetLength (see page 3657)

35.3.152 System.Sqr Function

Returns the square of a number.

Pascal

```
function Sqr(X: Extended): Extended; overload;
function Sqr(X: Integer): Integer; overload;
```

C++

```
Extended Sqr(Extended X);
int Sqr(int X);
```

File

System

Description

In Delphi code, the Sqr function returns the square of the argument.

X is a floating-point expression. The result, of the same type as X, is the square of X, or X*X.

Delphi Examples:

```
{
This example uses the System Sqr and Sqrt functions to calculate
the some mathematical values.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  S, Temp: string;
begin
   Str(Sqr(5.5):3:1, Temp);
   S := '5.5 squared is ' + Temp + #13#10;
```

```
   MessageDlg(S, mtInformation, [mbOk], 0);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  S, Temp: string;
begin
   Str(Sqrt(2.0):5:4, Temp);
   S := S + 'The square root of 2 is ' + Temp;
   MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

See Also

Sqrt (see page 3675)

Power

35.3.153 System.Sqrt Function

Returns the square root of X.

Pascal

```
function Sqrt(const X: Extended): Extended;
```

C++

```
Extended Sqrt(const Extended X);
```

File

System

Description

X is a Delphi floating-point expression. The result is the square root of X.

Delphi Examples:

```
{
This example uses the System Sqr and Sqrt functions to calculate
the some mathematical values.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  S, Temp: string;
begin
   Str(Sqr(5.5):3:1, Temp);
   S := '5.5 squared is ' + Temp + #13#10;
   MessageDlg(S, mtInformation, [mbOk], 0);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  S, Temp: string;
begin
   Str(Sqrt(2.0):5:4, Temp);
   S := S + 'The square root of 2 is ' + Temp;
   MessageDlg(S, mtInformation, [mbOk], 0);
end;
```

See Also

Sqr (see page 3674)

35.3.154 System.Str Function

Formats a string and returns it to a variable.

Pascal

```
procedure Str(X: Width; var S); overload;
procedure Str(X: Decimals; var S); overload;
```

C++

```
Str(Width X,  S);
Str(Decimals X,  S);
```

File

System

Description

In Delphi code, Str converts X to a string representation according to the Width and Decimals formatting parameters. The effect is like a call to Write (see page 3707) except the resulting string is stored in S instead of being written to a text file.

X is an integer-type or real-type expression. Width and Decimals are integer-type expressions. S is a string-type variable or a zero-based character array variable if extended syntax is enabled.

Delphi Examples:

```
{
This example makes a simple string conversion routine
and then prints an integer to the screen.
}

function MakeItAString(I: Longint): string;
{ Convert any integer type to a string }
var
  S: string[11];
begin
  Str(I, S);
  Result:= S;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Canvas.TextOut(10, 10, MakeItAString(-5322));
end;
```

See Also

Write (see page 3707)

35.3.155 System.StringCodePage Function

Returns the code page associated with the string.

Pascal

```
function StringCodePage(const S: UnicodeString): Word; overload;
function StringCodePage(const S: RawByteString): Word; overload;
```

C++

```
Word StringCodePage(const UnicodeString S);
Word StringCodePage(const RawByteString S);
```

File

System

Description

Call StringCodePage to obtain the code page of the string. Each string in RAD Studio is associated to a code page. The runtime library uses this information to safely convert and manipulate international character sets.

See Also

TEncoding (see page 3969)

StringElementSize (see page 3677)

35.3.156 System.StringElementSize Function

Returns the number of bytes that represent a character in the string.

Pascal

```
function StringElementSize(const S: UnicodeString): Word; overload;
function StringElementSize(const S: RawByteString): Word; overload;
```

C++

```
Word StringElementSize(const UnicodeString S);
Word StringElementSize(const RawByteString S);
```

File

System

Description

Call StringElementSize to obtain the number of bytes necessary to represent one character in the current string. Calling StringElementSize for Unicode strings always returns a result of 2.

See Also

TEncoding (see page 3969)

StringCodePage (see page 3676)

35.3.157 System.StringOfChar Function

Returns a string with a specified number of repeating characters.

Pascal

```
function StringOfChar(ch: AnsiChar; Count: Integer): AnsiString; overload;
function StringOfChar(ch: WideChar; Count: Integer): UnicodeString; overload;
```

C++

```
AnsiString StringOfChar(AnsiChar ch, int Count);
UnicodeString StringOfChar(WideChar ch, int Count);
```

File

System

Description

In Delphi code, StringOfChar returns a string containing **Count** characters with the character value given by **Ch**. For example,

S := StringOfChar('A', 10);

sets **S** to the string 'AAAAAAAAAA'.

Note that StringOfChar returns the same type as **ch**: either an **AnsiString** or **UnicodeString (see page 3381)**, so it works properly for both types.

See Also

FillChar (see page 3570)

DupeString (see page 3037)

35.3.158 System.StringRefCount Function

Returns the reference count of a string.

Pascal

```
function StringRefCount(const S: UnicodeString): Longint; overload;
function StringRefCount(const S: RawByteString): Longint; overload;
```

C++

```
Longint StringRefCount(const UnicodeString S);
Longint StringRefCount(const RawByteString S);
```

File

System

Description

Call StringRefCount to obtain the reference count of a string. Each string has an associated reference count. This value is used to detect how many references a string has.

To ensure that a string has a reference count of one, call UniqueString (see page 3693).

See Also

UniqueString (see page 3693)

35.3.159 System.StringToOleStr Function

Allocates memory and copies an AnsiStringa string to the OLE format.

Pascal

```
function StringToOleStr(const Source: AnsiString): PWideChar; overload;
function StringToOleStr(const Source: UnicodeString): PWideChar; overload;
```

C++

```
PWideChar StringToOleStr(const AnsiString Source);
PWideChar StringToOleStr(const UnicodeString Source);
```

File

System

Description

StringToOleStr allocates memory and copies string data from a native AnsiString into a format that can be passed to OLE functions.

See Also

OleStrToString (see page 3622)

OleStrToStrVar (see page 3622)

StringToWideChar (see page 3679)

35.3.160 System.StringToWideChar Function

Returns a UNICODE string from an AnsiString.

Pascal

```
function StringToWideChar(const Source: UnicodeString; Dest: PWideChar; DestSize: Integer):
PWideChar;
```

C++

```
PWideChar StringToWideChar(const UnicodeString Source, PWideChar Dest, int DestSize);
```

File

System

Description

StringToWideChar converts a string from ANSI to UNICODE and stores the result in a specified buffer.

Source is the string to convert. It must include at most DestSize – 1 characters.

Dest is the buffer into which StringToWideChar writes the corresponding UNICODE string. The caller allocates this memory to include at least DestSize wide characters. Following the call, Dest contains at most DestSize - 1 characters, terminated by a NULL wide character.

DestSize is the size of the buffer Dest.

StringToWideChar returns a pointer to Dest.

See Also

WideCharToString (see page 3706)

WideCharLenToString (see page 3705)

WideCharToStrVar (see page 3707)

WideCharLenToStrVar (see page 3706)

35.3.161 System.Succ Function

Returns the successor of an argument.

Pascal

```
procedure Succ(X);
```

C++

```
Succ( X);
```

File

System

Description

In Delphi code, Succ returns the successor of the argument, X.

X is an expression of an ordinal type (including Int64). The result, of the same type as X, is the successor of X.

Delphi Examples:

```
{
This example demonstrates how Pred and Succ can be used to
manage enumerations.
}
procedure TForm1.Button1Click(Sender: TObject);
type
  Colors = (RED,BLUE,GREEN);
var
  S: string;
begin
  S := 'The predecessor of 5 is ' + IntToStr(Pred(5)) + #13#10;
  S := S + 'The successor of 10 is ' + IntToStr(Succ(10)) + #13#10;
  if Succ(RED) = BLUE then
    S := S + 'In the type Colors, RED is the predecessor of BLUE.';
  MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Pred (see page 3628)

35.3.162 System.Swap Function

Exchanges high order byte with the low order byte of an integer or word.

Pascal

```
procedure Swap(X);
```

C++

```
Swap( X);
```

File

System

Description

In Delphi code, Swap exchanges the high-order bytes with the low-order bytes of the argument. **X** is an expression of type SmallInt, as a 16-bit value, or Word. This is provided for backward compatibility only.

Delphi Examples:

```
{
This example demonstrate use of the System Swap function to
exchange the high and low portions of a word.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  W: Word;
begin
  W := Swap($1234);
  ShowMessage(Format('The swapped value of $1234 = $%x', [W]));
end;
```

See Also

Hi (see page 3595)

Lo (see page 3609)

35.3.163 System.SysAllocMem Function

Pascal

```
function SysAllocMem(Size: Cardinal): Pointer;
```

C++

```
void * SysAllocMem(unsigned Size);
```

File

System

Description

This is function System.SysAllocMem.

35.3.164 System.SysFreeMem Function

Frees the memory pointed to by a specified pointer.

Pascal

```
function SysFreeMem(P: Pointer): Integer;
```

C++

```
int SysFreeMem(void * P);
```

File

System

Description

When implementing a custom memory manager, use SysFreeMem to free the memory pointed to by the P parameter.

Note: On Windows, the System (see page 3159) unit provides a limited implementation of SysFreeMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysFreeMem provided by the ShareMem unit. When used, ShareMem must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 3159) unit provides an implementation of SysFreeMem that supports reference counting and shared objects. The ShareMem unit is not provided for Linux.

See Also

GetHeapStatus (see page 3581)

SysGetMem (see page 3682)

SetMemoryManager (see page 3660)

35.3.165 System.SysGetMem Function

Allocates a specified number of bytes and returns a pointer to them.

Pascal

```
function SysGetMem(Size: Integer): Pointer;
```

C++

```
void * SysGetMem(int Size);
```

File

System

Description

When implementing a custom memory manager, use SysGetMem to dynamically allocate Size bytes of memory.

Note: On Windows, the System (see page 3159) unit provides a limited implementation of SysGetMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysGetMem provided by the ShareMem unit. When used, ShareMem must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 3159) unit provides an implementation of SysGetMem that supports reference counting and shared objects. The ShareMem unit is not provided for Linux.

See Also

SysFreeMem (see page 3681)

SysReallocMem (see page 3683)

SetMemoryManager (see page 3660)

35.3.166 System.SysReallocMem Function

Returns a pointer to a specified number of bytes, preserving the values pointed to by the Pointer parameter.

Pascal

```
function SysReallocMem(P: Pointer; Size: Integer): Pointer;
```

C++

```
void * SysReallocMem(void * P, int Size);
```

File

System

Description

When implementing a custom memory manager, use SysReallocMem to change the size of a dynamically allocated chunk of memory. If SysReallocMem can't expand the memory block point to by P, it frees the referenced memory and copies the values to the newly allocated memory that is returned. If Size is less than the number of bytes pointed to by P, only the first Size bytes are preserved. Otherwise, the entire value of the memory pointed to by P is preserved in the newly allocated memory.

Note: On Windows, the System (see page 3159) unit provides a limited implementation of SysReallocMem. If the custom memory manager needs to support reference counting and objects that can be shared between processes, use the implementation of SysReallocMem provided by the ShareMem unit. When used, ShareMem must be the first unit in the project file, to enforce a consistent memory management model.

Note: On Linux, the System (see page 3159) unit provides an implementation of SysReallocMem that supports reference counting and shared objects. The ShareMem unit is not provided for Linux.

See Also

SysFreeMem (see page 3681)

SysGetMem (see page 3682)

SetMemoryManager (see page 3660)

35.3.167 System.SysRegisterExpectedMemoryLeak Function

Pascal

```
function SysRegisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysRegisterExpectedMemoryLeak(void * P);
```

File

System

Description

This is function System.SysRegisterExpectedMemoryLeak.

35.3.168 System.SysUnregisterExpectedMemoryLeak Function

Pascal

```
function SysUnregisterExpectedMemoryLeak(P: Pointer): Boolean;
```

C++

```
Boolean SysUnregisterExpectedMemoryLeak(void * P);
```

File

System

Description

This is function System.SysUnregisterExpectedMemoryLeak.

35.3.169 System.TextStart Function

Pascal

```
procedure TextStart; deprecated;
```

C++

```
TextStart();
```

File

System

Description

This is function System.TextStart.

35.3.170 System.Trunc Function

Truncates a real number to an integer.

Pascal

```
function Trunc(X: Extended): Int64;
```

C++

```
Int64 Trunc(Extended X);
```

File

System

Description

In Delphi code, the Trunc function truncates a real-type value to an integer-type value. X is a real-type expression. Trunc returns an Int64 value that is the value of X rounded toward zero.

If the truncated value of X is not within the Int64 range, an EInvalidOp exception is raised.

Delphi Examples:

```
{
This example demonstrates how the Trunc function operates
on various real numbers.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   S, T: string;
begin
   Str(1.4:2:1, T);
   S := T + ' Truncs to ' + IntToStr(Trunc(1.4)) + #13#10;
   Str(1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(1.5)) + #13#10;
   Str(1.6:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(1.6)) + #13#10;
   Str(-1.4:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.4)) + #13#10;
   Str(-1.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.5)) + #13#10;
   Str(-1.6:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-1.6)) + #13#10;
   Str(2.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(2.5)) + #13#10;
   Str(-2.5:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-2.5)) + #13#10;
   Str(-2.6:2:1, T);
   S := S + T + ' Truncs to ' + IntToStr(Trunc(-2.6));
   MessageDlg(S, mtInformation, [mbOk], 0, mbOk);
end;
```

See Also

Int (see page 3602)

Round (see page 3648)

Floor

Ceil

35.3.171 System.Truncate Function

Deletes all the records after the current file position.

Pascal

```
procedure Truncate(var F);
```

C++

```
Truncate( F);
```

File

System

Description

In Delphi code, Call Truncate to cause the current file position to become end-of-file (Eof (see page 3558)(F) is true).

F is a file variable of any type except a text file. Truncate does not work on text files. F must be open.

Delphi Examples:

```
{
This example shows how to truncate a file after the position
set by the last read statement.
}
procedure TForm1.Button1Click(Sender: TObject);
var
   f: file of Integer;
   i,j: Integer;
 begin
   AssignFile(f,'TEST.INT');
   Rewrite(f);
   for i := 1 to 6 do
     Write(f,i);
   Writeln('File before truncation:');
   Reset(f);
   while not Eof(f) do
   begin
     Read(f,i);
     Writeln(i);
   end;
   Reset(f);
   for i := 1 to 3 do
     Read(f,j); { Read ahead 3 records }
   Truncate(f); { Cut file off here }
   Writeln;
   Writeln('File after truncation:');
   Reset(f);
   while not Eof(f) do
   begin
     Read(f,i);
     Writeln(i);
   end;
   CloseFile(f);
   Erase(f);
 end;
```

See Also

Reset (see page 3643)

Rewrite (see page 3645)

Seek (see page 3650)

35.3.172 System.UCS4StringToUnicodeString Function

Converts an UCS-4 encoded string to an Unicode string.

Pascal

```
function UCS4StringToUnicodeString(const S: UCS4String): UnicodeString;
```

C++

```
UnicodeString UCS4StringToUnicodeString(const UCS4String S);
```

File

System

Description

Call UCS4StringToUnicodeString to convert an UCS-4 encoded string to Unicode. S is a string that contains UCS-4 encoded

characters. The result of the function is the corresponding Unicode (UTF-16) string value.

See Also

UnicodeStringToUCS4String (see page 3689)

35.3.173 System.UCS4StringToWideString Function

Converts a string of 4-byte Unicode characters into a WideString (see page 3467) value.

Pascal

```
function UCS4StringToWideString(const S: UCS4String): WideString;
```

C++

```
BSTR UCS4StringToWideString(const UCS4String S);
```

File

System

Description

Call UCS4StringToWideString to convert a string of UCS4 (4-byte Unicode) characters into a WideString (see page 3467) value. UCS4 strings are returned by some Linux operating system calls.

Delphi Examples:

```
{
This example converts various string formats to and from Unicode.
This example requires two text edits and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
  LFileStream: TFileStream;
  LOffset: Integer;
  LEncoding, DestEncoding: TEncoding;
  ComboIndex: Integer;
  ByteString: RawByteString;
  FileText: string;
  ucs4Text: UCS4String;
  wideStringText: WideString;
  utf8Text: UTF8String;
  L: Cardinal;

begin
  LEncoding:= nil;
  ComboIndex := ComboBox1.ItemIndex;
  LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    // Read file into buffer
    SetLength(LBuffer, LFileStream.Size);
    LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));
    // Get data encoding of data read
    LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
  finally
    LFileStream.Free;
  end;

  // Process various types
  case ComboIndex of
```

```
    // AnsiToUtf8
    0:
    begin
      // Identify encoding and convert buffer to Ansi
      DestEncoding := TEncoding.ASCII;
      LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
        LOffset, Length(LBuffer) - LOffset);
      LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
      FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
      ByteString := AnsiToUtf8(FileText);
      ShowMessage(UTF8ToString(ByteString));

      // Write to file
      LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
      try
        // New destination encoding
        DestEncoding := TEncoding.UTF8;
        LByteOrderMark := DestEncoding.GetPreamble;
        LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
        LFileStream.Write(ByteString[1], Length(ByteString));
      finally
        LFileStream.Free;
      end;
    end;

    // UCS4StringToWideString
    1:
    begin
      // Identify encoding and convert buffer to UCS4
      DestEncoding := TEncoding.Unicode;
      LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
        LOffset, Length(LBuffer) - LOffset);
      LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
      FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
      ucs4Text := UnicodeStringToUCS4String(FileText);
      wideStringText := UCS4StringToWideString(ucs4Text);
      ShowMessage(wideStringText);

      // Write to file
      LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
      try
        // New destination encoding
//        DestEncoding := TEncoding.UTF8;
        LByteOrderMark := DestEncoding.GetPreamble;
        LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
        LFileStream.Write(wideStringText[1], Length(wideStringText) * SizeOf(Char));
      finally
        LFileStream.Free;
      end;
    end;

    // UnicodeToUtf8
    2:
    begin
      // Identify encoding and convert buffer to Unicode
      DestEncoding := TEncoding.Unicode;
      LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
        LOffset, Length(LBuffer) - LOffset);
      LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
      FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
      L := Length(FileText);
      SetLength(utf8Text, L * SizeOf(Char) + 1);
      L := UnicodeToUtf8(PAnsiChar(utf8Text), Length(utf8Text), PWideChar(FileText), L);
      ShowMessage(UTF8ToString(utf8Text));
```

```
        // Write to file
        LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
        try
          // New destination encoding
          DestEncoding := TEncoding.UTF8;
          LByteOrderMark := DestEncoding.GetPreamble;
          LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
          LFileStream.Write(utf8Text[1], L);
        finally
          LFileStream.Free;
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    StringList: TStrings;
  begin
    Edit1.Text := 'Sample.txt';
    Edit2.Text := 'Dest.txt';
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('AnsiToUtf8');
        Add('UCS4StringToWideString');
        Add('UnicodeToUtf8');
      end;
      with ComboBox1 do begin
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

WideStringToUCS4String (see page 3707)

PUCS4Chars (see page 3632)

UnicodeToUtf8 (see page 3690)

Utf8ToUnicode (see page 3699)

35.3.174 System.UnicodeStringToUCS4String Function

Converts an Unicode string to its UCS-4 encoded representation.

Pascal

```
function UnicodeStringToUCS4String(const S: UnicodeString): UCS4String;
```

C++

```
UCS4String UnicodeStringToUCS4String(const UnicodeString S);
```

File

System

Description

Call UnicodeStringToUCS4String to convert an Unicode string to its UCS-4 encoded representation. S is an Unicode string that contains UTF-16 characters. The result of the function is the corresponding UCS-4 string value.

See Also

UCS4StringToUnicodeString (see page 3686)

35.3.175 System.UnicodeToUtf8 Function

Converts a string of Unicode characters into a UTF-8 string.

Pascal

```
function UnicodeToUtf8(Dest: PChar; MaxDestBytes: Cardinal; Source: PWideChar; SourceChars:
Cardinal): Cardinal; overload;
function UnicodeToUtf8(Dest: PChar; Source: PWideChar; MaxBytes: Integer): Integer; overload;
deprecated;
function UnicodeToUtf8(Dest: PAnsiChar; Source: PWideChar; MaxBytes: Integer): Integer;
overload; deprecated;
function UnicodeToUtf8(Dest: PAnsiChar; MaxDestBytes: Cardinal; Source: PWideChar;
SourceChars: Cardinal): Cardinal; overload;
```

C++

```
unsigned UnicodeToUtf8(const char * Dest, unsigned MaxDestBytes, PWideChar Source, unsigned
SourceChars);
int UnicodeToUtf8(const char * Dest, PWideChar Source, int MaxBytes);
int UnicodeToUtf8(PAnsiChar Dest, PWideChar Source, int MaxBytes);
unsigned UnicodeToUtf8(PAnsiChar Dest, unsigned MaxDestBytes, PWideChar Source, unsigned
SourceChars);
```

File

System

Description

Call UnicodeToUtf8 to convert a Unicode string to a null-terminated sequence of UTF-8 characters. Use the second syntax. The deprecated syntax is included only for backward compatibility.

Dest receives the resulting null-terminated array of UTF-8 characters.

MaxDestBytes or **MaxBytes** indicates the number of bytes available in the buffer specified by **Dest**, including the byte for the null terminator.

Source is an array of Unicode characters.

SourceChars is the number of characters in **Source**. If **SourceChars** is not specified, **Source** must be null-terminated.

UnicodeToUtf8 returns the number of bytes written to **Dest**.

Delphi Examples:

```
{
This example converts various string formats to and from Unicode.
This example requires two text edits and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
```

```
    LFileStream: TFileStream;
    LOffset: Integer;
    LEncoding, DestEncoding: TEncoding;
    ComboIndex: Integer;
    ByteString: RawByteString;
    FileText: string;
    ucs4Text: UCS4String;
    wideStringText: WideString;
    utf8Text: UTF8String;
    L: Cardinal;

  begin
    LEncoding:= nil;
    ComboIndex := ComboBox1.ItemIndex;
    LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
    try
      // Read file into buffer
      SetLength(LBuffer, LFileStream.Size);
      LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));
      // Get data encoding of data read
      LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
    finally
      LFileStream.Free;
    end;

    // Process various types
    case ComboIndex of

      // AnsiToUtf8
      0:
      begin
        // Identify encoding and convert buffer to Ansi
        DestEncoding := TEncoding.ASCII;
        LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
          LOffset, Length(LBuffer) - LOffset);
        LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
        FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
        ByteString := AnsiToUtf8(FileText);
        ShowMessage(UTF8ToString(ByteString));

        // Write to file
        LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
        try
          // New destination encoding
          DestEncoding := TEncoding.UTF8;
          LByteOrderMark := DestEncoding.GetPreamble;
          LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
          LFileStream.Write(ByteString[1], Length(ByteString));
        finally
          LFileStream.Free;
        end;
      end;

      // UCS4StringToWideString
      1:
      begin
        // Identify encoding and convert buffer to UCS4
        DestEncoding := TEncoding.Unicode;
        LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
          LOffset, Length(LBuffer) - LOffset);
        LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
        FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
        ucs4Text := UnicodeStringToUCS4String(FileText);
        wideStringText := UCS4StringToWideString(ucs4Text);
        ShowMessage(wideStringText);
```

```
        // Write to file
        LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
        try
          // New destination encoding
//          DestEncoding := TEncoding.UTF8;
          LByteOrderMark := DestEncoding.GetPreamble;
          LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
          LFileStream.Write(wideStringText[1], Length(wideStringText) * SizeOf(Char));
        finally
          LFileStream.Free;
        end;
      end;

      // UnicodeToUtf8
      2:
      begin
        // Identify encoding and convert buffer to Unicode
        DestEncoding := TEncoding.Unicode;
        LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
          LOffset, Length(LBuffer) - LOffset);
        LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
        FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
        L := Length(FileText);
        SetLength(utf8Text, L * SizeOf(Char) + 1);
        L := UnicodeToUtf8(PAnsiChar(utf8Text), Length(utf8Text), PWideChar(FileText), L);
        ShowMessage(UTF8ToString(utf8Text));

        // Write to file
        LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
        try
          // New destination encoding
          DestEncoding := TEncoding.UTF8;
          LByteOrderMark := DestEncoding.GetPreamble;
          LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
          LFileStream.Write(utf8Text[1], L);
        finally
          LFileStream.Free;
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    StringList: TStrings;
  begin
    Edit1.Text := 'Sample.txt';
    Edit2.Text := 'Dest.txt';
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('AnsiToUtf8');
        Add('UCS4StringToWideString');
        Add('UnicodeToUtf8');
      end;
      with ComboBox1 do begin
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

UCS4StringToWideString (see page 3687)

Utf8ToUnicode (see page 3699)

UTF8Encode (see page 3696)

UTF8Decode (see page 3696)

UTF8String (see page 3804)

35.3.176 System.UniqueString Function

Ensures that a given string has a reference count of one.

Pascal

```
procedure UniqueString(var str: AnsiString); overload;
procedure UniqueString(var str: WideString); overload;
procedure UniqueString(var str: UnicodeString); overload;
```

C++

```
UniqueString(AnsiString str);
UniqueString(BSTR str);
UniqueString(UnicodeString str);
```

File

System

Description

UniqueString forces the reference count on a string to one, copying the string in memory if necessary. For normal string handling, there is no need to call UniqueString. UniqueString is used only in cases where an application modifies the contents of a string after:

- casting the string to a **PAnsiChar** or **PWideChar** in Delphi.
- using the `c_str()`, `t_str()` or `data()` methods of **UnicodeString (see page 3381)** or **AnsiString** or the `c_bstr()` method of **WideString (see page 3467)** in C++. (It's better to use `data()` than `c_str()` because `c_str()` returns an empty string for a NULL **UnicodeString (see page 3381)** or **AnsiString**.)

Delphi Examples:

```
{
The following code demostrates the use of Ptr function. The example
makes use of Ptr to simulate pointer arithmetics.
}
function FastUpperCase(const S: String): String;
var
  C: ^Char;
  I: Integer;
begin
  { Get the text in the edit box }
  Result := S;

  { Ensure the string reference stored in Result is unique,
    otherwise we may change the original string also.
    No copy-on-write is done if we're accesing the character
    directly.
  }
  UniqueString(Result);

  { Find the address of the first char in the string }
  C := Addr(Result[1]);
```

```
    for I := 0 to Length(Result) - 1 do
    begin
      { Up-case the character }
      C^ := UpCase(C^);

      { Move to the next character }
      C := Ptr(Integer(C) + SizeOf(Char));
    end;
  end;


  procedure TForm1.Button1Click(Sender: TObject);
  begin
    { Uppercase the string in the edit box }
    Edit1.Text := FastUpperCase(Edit1.Text);
  end;
```

See Also

SetString (see page 3665)

SetLength (see page 3657)

35.3.177 System.UnregisterExpectedMemoryLeak Function

Unregisters an expected memory leak.

Pascal

```
function UnregisterExpectedMemoryLeak(P: Pointer): boolean;
```

C++

```
boolean UnregisterExpectedMemoryLeak(void * P);
```

File

System

Description

UnregisterExpectedMemoryLeak removes a memory location from the Memory Manager's list of expected memory leaks.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

RegisterExpectedMemoryLeak (see page 3640)

ReportMemoryLeaksOnShutdown (see page 3835)

35.3.178 System.UnregisterModule Function

Unregisters the specified module.

Pascal

```
procedure UnregisterModule(LibModule: PLibModule);
```

C++

```
UnregisterModule(PLibModule LibModule);
```

File

System

Description

This is an internal function and should not be used directly.

See Also

RegisterModule (see page 3640)

35.3.179 System.UpCase Function

Converts a character to uppercase.

Pascal

```
function UpCase(Ch: AnsiChar): AnsiChar; overload;
function UpCase(Ch: WideChar): WideChar; overload;
```

C++

```
AnsiChar UpCase(AnsiChar Ch);
WideChar UpCase(WideChar Ch);
```

File

System

Description

UpCase converts a character to uppercase. Ch is an expression of type Char. Character values not in the range a..z are unaffected.

C++ Examples:

```
/*
The following code uppercases every other character in the
text of an edit control when the user clicks the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Get string from TEdit control
  UnicodeString s = Edit1->Text;
  for (int i = 1; i <= s.Length(); i += 2)
    s[i] = System::UpCase(s[i]);
  Edit1->Text = s;
}
```

Delphi Examples:

```
{
The following code uppercases every other character in the
text of an edit control when the user clicks the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
  s : string;
  i : Integer;
begin
  { Get string from TEdit control }
  s := Edit1.Text;
  for i := 1 to Length(s) do
    if i mod 2 = 0 then s[i] := System.UpCase(s[i]);
  Edit1.Text := s;
end;
```

See Also

AnsiUpperCase (see page 4062)

StrUpper (see page 4292)

UpperCase (see page 4311)

35.3.180 System.UTF8Decode Function

Converts a UTF8 string to a Unicode string (WideString (see page 3467)).

Pascal

```
function UTF8Decode(const S: RawByteString): WideString; deprecated;
```

C++

```
BSTR UTF8Decode(const RawByteString S);
```

File

System

Description

Call UTF8Decode to convert a UTF-8 string to Unicode. S is a string encoded in UTF-8. UTF8Decode returns the corresponding WideString (see page 3467) value that represents the string in Unicode.

See Also

UnicodeToUtf8 (see page 3690)

UTF8Encode (see page 3696)

Utf8ToUnicode (see page 3699)

Utf8ToAnsi (see page 3698)

AnsiToUtf8 (see page 3517)

35.3.181 System.UTF8Encode Function

Converts a WideString (see page 3467) value to a UTF8 string.

Pascal

```
function UTF8Encode(const WS: WideString): RawByteString; overload;
function UTF8Encode(const US: UnicodeString): RawByteString; overload;
function UTF8Encode(const A: RawByteString): RawByteString; overload;
```

C++

```
RawByteString UTF8Encode(const BSTR WS);
RawByteString UTF8Encode(const UnicodeString US);
RawByteString UTF8Encode(const RawByteString A);
```

File

System

Description

Call UTF8Encode to convert a Unicode string to UTF-8. WS is the Unicode string to convert. Utf8Encode returns the corresponding UTF-8 string.

See Also

UnicodeToUtf8 (see page 3690)

UTF8Decode (see page 3696)

Utf8ToUnicode (see page 3699)

35.3.182 System.UTF8EncodeToShortString Function

Converts a string to its UTF-8 encoded representation.

Pascal

```
function UTF8EncodeToShortString(const WS: WideString): ShortString; overload;
function UTF8EncodeToShortString(const US: UnicodeString): ShortString; overload;
function UTF8EncodeToShortString(const A: RawByteString): ShortString; overload;
```

C++

```
ShortString UTF8EncodeToShortString(const BSTR WS);
ShortString UTF8EncodeToShortString(const UnicodeString US);
ShortString UTF8EncodeToShortString(const RawByteString A);
```

File

System

Description

Call UTF8EncodeToShortString to convert a string to its UTF-8 representation. S is the string to be converted. The result of the function is the corresponding UTF-8 encoded ShortString value.

See Also

UTF8ToWideString (see page 3700)

UTF8ToUnicodeString (see page 3700)

UnicodeToUtf8 (see page 3690)

UTF8Encode (see page 3696)

Utf8ToUnicode (see page 3699)

Utf8ToAnsi (see page 3698)

AnsiToUtf8 (see page 3517)

35.3.183 System.Utf8ToAnsi Function

Converts a UTF8 string to a string of Ansi characters.

Pascal

```
function Utf8ToAnsi(const S: RawByteString): string;
```

C++

```
UnicodeString Utf8ToAnsi(const RawByteString S);
```

File

System

Description

Call Utf8ToAnsi to convert a UTF-8 string to Ansi. S is a string encoded in UTF-8. Utf8ToAnsi returns the corresponding string that uses the Ansi character set.

See Also

UnicodeToUtf8 (see page 3690)

UTF8Encode (see page 3696)

Utf8ToUnicode (see page 3699)

UTF8Decode (see page 3696)

AnsiToUtf8 (see page 3517)

35.3.184 System.UTF8ToString Function

Converts an UTF-8 encoded string to a string.

Pascal

```
function UTF8ToString(const S: RawByteString): string;
function UTF8ToString(const S: ShortString): string;
function UTF8ToString(const S: PAnsiChar): string;
function UTF8ToString(const S: array of AnsiChar): string; overload;
```

C++

```
UnicodeString UTF8ToString(const RawByteString S);
UnicodeString UTF8ToString(const ShortString S);
UnicodeString UTF8ToString(const PAnsiChar S);
UnicodeString UTF8ToString(const array of AnsiChar S);
```

File

System

Description

Call UTF8ToString to convert an UTF-8 encoded string to a string. S is a string or an array of UTF-8 encoded characters. The result of the function is the corresponding decoded string value.

See Also

UTF8ToWideString (see page 3700)

UTF8ToUnicodeString (see page 3700)

UnicodeToUtf8 (see page 3690)

UTF8Encode (see page 3696)

Utf8ToUnicode (see page 3699)

Utf8ToAnsi (see page 3698)

AnsiToUtf8 (see page 3517)

35.3.185 System.Utf8ToUnicode Function

Converts a string of UTF-8 characters to a Unicode string.

Pascal

```
function Utf8ToUnicode(Dest: PWideChar; Source: PAnsiChar; MaxChars: Integer): Integer;
overload; deprecated;
function Utf8ToUnicode(Dest: PWideChar; MaxDestChars: Cardinal; Source: PAnsiChar;
SourceBytes: Cardinal): Cardinal; overload;
```

C++

```
int Utf8ToUnicode(PWideChar Dest, PAnsiChar Source, int MaxChars);
unsigned Utf8ToUnicode(PWideChar Dest, unsigned MaxDestChars, PAnsiChar Source, unsigned
SourceBytes);
```

File

System

Description

Call Utf8ToUnicode to convert a UTF-8 string to Unicode. Use the second syntax. The deprecated syntax is provided for backward compatibility only.

Dest receives the resulting null-terminated array of Unicode characters.

MaxDestChars or MaxChars indicates the maximum number of Unicode characters that the buffer specified by Dest can hold, not counting the null terminator.

Source is an array of UTF-8 characters.

SourceBytes is the number of bytes in Source. If SourceBytes is not specified, Source must be null-terminated.

Utf8ToUnicode returns the number of characters written to Dest.

In case of errors, the Utf8ToUnicode ... : Cardinal variant returns `Cardinal(-1) == $FFFFFFFF` and the Utf8ToUnicode ... : Integer returns `-1`.

See Also

UCS4StringToWideString (see page 3687)

UnicodeToUtf8 (see page 3690)

UTF8Encode (see page 3696)

UTF8Decode (see page 3696)

UTF8String (see page 3804)

35.3.186 System.UTF8ToUnicodeString Function

Converts an UTF-8 encoded string to an Unicode string.

Pascal

```
function UTF8ToUnicodeString(const S: RawByteString): UnicodeString; overload;
function UTF8ToUnicodeString(const S: ShortString): UnicodeString; overload;
function UTF8ToUnicodeString(const S: PAnsiChar): UnicodeString; overload;
```

C++

```
UnicodeString UTF8ToUnicodeString(const RawByteString S);
UnicodeString UTF8ToUnicodeString(const ShortString S);
UnicodeString UTF8ToUnicodeString(const PAnsiChar S);
```

File

System

Description

Call UTF8ToUnicodeString to convert an UTF-8 encoded string to Unicode. S is a string that contains UTF-8 encoded characters. The result of the function is the corresponding Unicode string value.

See Also

UTF8ToWideString (see page 3700)

UnicodeToUtf8 (see page 3690)

UTF8Encode (see page 3696)

Utf8ToUnicode (see page 3699)

Utf8ToAnsi (see page 3698)

AnsiToUtf8 (see page 3517)

35.3.187 System.UTF8ToWideString Function

Converts an UTF-8 encoded string to a WideString (see page 3467).

Pascal

```
function UTF8ToWideString(const S: RawByteString): WideString;
```

C++

```
BSTR UTF8ToWideString(const RawByteString S);
```

File

System

Description

Call UTF8ToWideString to convert an UTF-8 encoded string to Unicode. S is a RawByteString (see page 3783) that contains

UTF-8 encoded characters. The result of the function is the corresponding WideString (see page 3467) value that represents the string in Unicode.

See Also

UTF8ToUnicodeString (see page 3700)

UnicodeToUtf8 (see page 3690)

UTF8Encode (see page 3696)

Utf8ToUnicode (see page 3699)

Utf8ToAnsi (see page 3698)

AnsiToUtf8 (see page 3517)

35.3.188 System.Val Function

Converts a string to a numeric representation.

Pascal

```
procedure Val(S; var V; var Code: Integer);
```

C++

```
Val( S,  V, int Code);
```

File

System

Description

In Delphi code, Val converts the string value S to its numeric representation, as if it were read from a text file with Read (see page 3634).

S is a string-type expression; it must be a sequence of characters that form a signed real number.

V is an integer-type or real-type variable. If V is an integer-type variable, S must form a whole number.

Code is a variable of type Integer.

If the string is invalid, the index of the offending character is stored in Code; otherwise, Code is set to zero. For a null-terminated string, the error position returned in Code is one larger than the actual zero-based index of the character in error.

Delphi Examples:

```
{
This example uses the System Val function to convert a string to
an integer.  Enter an integer in the text edit and then click the
button.  An error will occur if the value entered is not an integer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I, Code: Integer;
  mystr: string;
begin
  { Get text from TEdit control }
  mystr := Edit1.Text;
  Val(mystr, I, Code);
  { Error during conversion to integer? }
  if Code <> 0 then
```

```
    MessageDlg('Error at position: ' + IntToStr(Code), mtWarning, [mbOk], 0, mbOk)
  else
    Canvas.TextOut(10, 10, 'Value = ' + IntToStr(I));
end;
```

See Also

Str (see page 3676)

35.3.189 System.VarArrayRedim Function

Resizes a Variant (see page 3421) array.

Pascal

```
procedure VarArrayRedim(A: Variant; HighBound: Integer);
```

C++

```
VarArrayRedim(Variant A, int HighBound);
```

File

System

Description

VarArrayRedim resizes the given Variant (see page 3421) array by changing the high bound of the rightmost dimension to the given value. The A parameter can be a Variant (see page 3421) or an OleVariant (see page 3274), and it must be possible to assign a value to A (A must be an lvalue).

Existing elements of the array are preserved, and new elements are set to zero or empty. An EVariantArrayCreateError exception is thrown if the Variant (see page 3421) given by A is not an array, or if there is not enough memory to resize the array.

Note: A Variant (see page 3421) array can't be resized while it is locked.

C++ Examples:

```
/*
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Variant array;
    int bounds[2] = {0, 9};

    /*
    Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer
    */
    array = VarArrayCreate(bounds, 1, varInteger);

    // Increase the length of the variant array
    VarArrayRedim(array, 49);

    MessageDlg(String("Variant array has ") + IntToStr(VarArrayDimCount(array)) +
        " dimensions", mtInformation, TMsgDlgButtons() << mbOK, 0);

    // Traverse the array from lower to higher bounds
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
```

```
    {
        // Put the element I at index I
        VarArrayPut(array, i, &i, 0);
    }

    // Now read the elements of the array
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        if (VarArrayGet(array, &i, 0) != i)
            MessageDlg("Error! Invalid element found at current position!",
                mtError, TMsgDlgButtons() << mbOK, 0);
    }

    // Clear the variant array
    VarClear(array);
}
```

Delphi Examples:

```
{
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Arr: Variant;
  I: Integer;
begin
  { Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer }
  Arr := VarArrayCreate([0, 9], varInteger);

  { Increase the length of the variant array }
  VarArrayRedim(Arr, 49);

  MessageDlg('Variant array has ' + IntToStr(VarArrayDimCount(Arr)) + ' dimensions',
      mtInformation, [mbOK], 0);

  { Traverse the array from lower to higher bounds }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    VarArrayPut(Arr, I, [I]);
  end;

  { Now read the elements of the array }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    if VarArrayGet(Arr, [I]) <> I then
      MessageDlg('Error! Invalid element found at current position!', mtError, [mbOK], 0);
  end;

  { Clear the variant array }
  VarClear(Arr);
end;
```

See Also

VarArrayCreate (see page 5059)

VarArrayLock (see page 5067)

EVariantArrayCreateError (see page 4978)

35.3.190 System.VarClear Function

Empties a Variant (see page 3421), so that it is unassigned.

Pascal

```
procedure VarClear(V: Variant);
```

C++

```
VarClear(Variant V);
```

File

System

Description

Calling VarClear is equivalent to assigning the Unassigned constant to the Variant (see page 3421). V can be either a Variant (see page 3421) or an OleVariant (see page 3274), but it must be possible to assign a value to it (it must be an lvalue).

After calling VarClear, the VarIsEmpty function returns true, and the VarType function returns varEmpty (see page 3865). Using an unassigned variant in an expression causes an exception to be thrown. Likewise, if you attempt to convert an unassigned Variant (see page 3421) to another type (using VarAsType), an exception is thrown.

Note: Do not confuse clearing a Variant (see page 3421), which leaves it unassigned, with assigning a Null value. A Null Variant (see page 3421) is still assigned, but has the value Null. Unlike unassigned Variants (see page 4973), Null Variants (see page 4973) can be used in expressions and can be converted to other types of Variants (see page 4973).

C++ Examples:

```
/*
This example shows how to use the TVarData type in typecasts
of Variant variables to access the internals of a variable.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Variant V;
  AnsiString mystr;
  if (TVarData(V).VType == varEmpty)
    ListBox1->Items->Add("Variants start empty.");
  VarClear(V);
  TVarData(V).VType = varString;
  mystr = "Here is my string";
  std::auto_ptr<char> MyBuffer(new char[mystr.Length() + 1]);
  StrCopy(MyBuffer.get(), mystr.c_str());
  TVarData(V).VString  = MyBuffer.get();
  mystr = PAnsiChar(TVarData(V).VString);
  ListBox1->Items->Add("This variant is now a string: " + mystr);
  VarClear(V);
  TVarData(V).VType = varInteger;
  TVarData(V).VInteger = 1234567;
  ListBox1->Items->Add("This variant is now an integer: " + IntToStr(TVarData(V).VInteger));
}
```

Delphi Examples:

```
{{
This example shows how to use the TVarData type in typecasts
```

```
  of Variant variables to access the internals of a variable.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    V: Variant;
    mystr: string;
    MyBuffer: PChar;
  begin
    if TVarData(V).VType = varEmpty then
      ListBox1.Items.Add('Variants start empty.');
    VarClear(V);
    TVarData(V).VType := varString;
    mystr := 'Here is my string';
    MyBuffer:= GetMemory(Length(mystr));
    StrCopy(MyBuffer, PChar(mystr));
    TVarData(V).VString := MyBuffer;
    ListBox1.Items.Add('This variant is now a string: ' + PChar(TVarData(V).VString));
    VarClear(V);
    TVarData(V).VType := varInteger;
    TVarData(V).VInteger := 1234567;
    ListBox1.Items.Add('This variant is now an integer: ' + IntToStr(TVarData(V).VInteger));
  end;
```

See Also

Unassigned (see page 5058)

VarIsEmpty (see page 5080)

VarAsType (see page 5073)

VarType (see page 5090)

35.3.191 System.WideCharLenToString Function

Converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)**.

Pascal

```
function WideCharLenToString(Source: PWideChar; SourceLen: Integer): UnicodeString;
```

C++

```
UnicodeString WideCharLenToString(PWideChar Source, int SourceLen);
```

File

System

Description

WideCharLenToString converts `SourceLen` **WideString (see page 3467)** characters from the buffer given by `Source` to a **UnicodeString (see page 3381)**.

See Also

StringToWideChar (see page 3679)

WideCharToString (see page 3706)

35.3.192 System.WideCharLenToStrVar Function

WideCharLenToStrVar converts **WideString (see page 3467)** characters to **UnicodeString (see page 3381)** or **AnsiString**.

Pascal

```
procedure WideCharLenToStrVar(Source: PWideChar; SourceLen: Integer; var Dest: UnicodeString);
overload;
procedure WideCharLenToStrVar(Source: PWideChar; SourceLen: Integer; var Dest: AnsiString);
overload;
```

C++

```
WideCharLenToStrVar(PWideChar Source, int SourceLen, UnicodeString Dest);
WideCharLenToStrVar(PWideChar Source, int SourceLen, AnsiString Dest);
```

File

System

Description

The WideCharLenToStrVar procedure converts `SourceLen` **WideString (see page 3467)** characters from `Source` into a **UnicodeString (see page 3381)** or **AnsiString**, and copies the result into `Dest`.

See Also

WideCharToString (see page 3706)

35.3.193 System.WideCharToString Function

Converts null-terminated **WideString (see page 3467)** string to **UnicodeString (see page 3381)**.

Pascal

```
function WideCharToString(Source: PWideChar): UnicodeString;
```

C++

```
UnicodeString WideCharToString(PWideChar Source);
```

File

System

Description

WideCharToString converts the given null-terminated **WideString (see page 3467)** string to a **UnicodeString (see page 3381)**.

See Also

StringToWideChar (see page 3679)

WideCharLenToString (see page 3705)

35.3.194 System.WideCharToStrVar Function

Converts **WideString (see page 3467)** string to **UnicodeString (see page 3381)**.

Pascal

```
procedure WideCharToStrVar(Source: PWideChar; var Dest: UnicodeString);
```

C++

```
WideCharToStrVar(PWideChar Source, UnicodeString Dest);
```

File

System

Description

WideCharToStrVar converts the **WideString (see page 3467)** string `Source` into a **UnicodeString (see page 3381)**, and copies the result into `Dest`. The source must be null-terminated.

See Also

WideCharToString (see page 3706)

35.3.195 System.WideStringToUCS4String Function

Converts a WideString (see page 3467) value to a string of 4-byte Unicode characters.

Pascal

```
function WideStringToUCS4String(const S: WideString): UCS4String;
```

C++

```
UCS4String WideStringToUCS4String(const BSTR S);
```

File

System

Description

Call WideStringToUCS4String to convert a WideString (see page 3467) value to a string of UCS4 (4-byte Unicode) characters. UCS4 strings are used by some Linux operating system calls.

35.3.196 System.Write Function

Writes to either a typed file or a text file

Pascal

```
procedure Write(P1); overload;
procedure Write(P1; Pn); overload;
procedure Write(var F: Text; P1); overload;
procedure Write(var F: Text; P1; Pn); overload;
```

C++

```
Write( P1);
Write( P1,  Pn);
Write(Text F,  P1);
Write(Text F,  P1,  Pn);
```

File

System

Description

The syntax shown here for the Write procedure is an example that illustrates that Write can take a variable number of arguments.

Write procedure for typed files:

In Delphi code, Write writes a file to a file component. F is a file variable, and each V is a variable of the same type as the component type of F. For each variable written, the current file position is advanced to the next component. If the current file position is at the end of the file (that is, if Eof (see page 3558)(F) is true), the file is expanded.

Write procedure for text files:

In Delphi code, Write writes one or more values to a text file. F, if specified, is a text file variable. If F is omitted, the standard file variable Output (see page 3833) is assumed. Each P is a write parameter. Each write parameter includes an output expression whose value is to be written to the file. A write parameter can also contain the specifications of a field width and a number of decimal places. Each output expression must be of a type Char, one of the Integer types (Byte, Shortint, Word, Longint, Cardinal), one of the floating-point types (Single, Real, Double, Extended, Currency (see page 3235)), one of the string types (PChar, AnsiString, ShortString), a packed string, or one of the Boolean types (Boolean, Bool).

A write parameter has the form:

```
OutExpr [: MinWidth [: DecPlaces ] ]
```

where OutExpr is an output expression. MinWidth and DecPlaces are type integer expressions.

MinWidth specifies the minimum field width, which must be greater than 0. Exactly MinWidth characters are written (using leading blanks if necessary) except when OutExpr has a value that must be represented in more than MinWidth characters. In that case, enough characters are written to represent the value of OutExpr. Likewise, if MinWidth is omitted, then the necessary number of characters is written to represent the value of OutExpr.

DecPlaces specifies the number of decimal places in a fixed-point representation of one of the Real types. It can be specified only if OutExpr is one of the Real types, and if MinWidth is also specified. When MinWidth is specified, it must be greater than or equal to 0.

Write with a character-type value: If MinWidth is omitted, the character value of OutExpr is written to the file. Otherwise, MinWidth - 1 blanks followed by the character value of OutExpr is written.

Write with one of the integer type values: If MinWidth is omitted, the decimal representation of OutExpr is written to the file with no preceding blanks. If MinWidth is specified and its value is larger than the length of the decimal string, enough blanks are written before the decimal string to make the field width MinWidth.

Write with one of the real type values: If OutExpr has one of the real type values, its decimal representation is written to the file. The format of the representation depends on the presence or absence of DecPlaces. If DecPlaces is omitted (or if it's present but has a negative value), a floating-point decimal string is written. If MinWidth is also omitted, a default MinWidth of 17 is assumed; otherwise, if MinWidth is less than 8, it's assumed to be 8. The format of the floating-point string is:

```
[   | - ] <digit> . <decimals> E [ + | - ] <exponent>
```

The following table lists the components of the output string.

Component	Meaning
[| —]	" “ or “-”I, according to the sign of OutExpr
<digit>	Single digit, “0” only if OutExpr is 0
<decimals>	Digit string of MinWidth-7 (but at most 10) digits
E	Uppercase [E] character
[+ | —]	According to the sign of the exponent
<exponent>	Two-digit decimal exponent

If DecPlaces is present, a fixed-point decimal string is written. If DecPlaces is larger than 11, it's assumed to be 11. The format of the fixed-point string follows:

```
[<blanks>] [ - ] <digits> [.<decimals>]
```

The following table lists the components of the fixed-point string:

Component	Meaning
[<blanks>	Blanks to satisfy MinWidth
[-]	If OutExpr is negative
<digits>	At least one digit, but no leading zeros
[.<decimals>]	Decimals if DecPlaces > 0

Write with one of the string-type values: If MinWidth is omitted, the string value of OutExpr is written to the file with no leading blanks. If MinWidth is specified, and its value is larger than the length of OutExpr, enough blanks are written before the decimal string to make the field width MinWidth.

Write with a packed string-type value: If OutExpr is of packed string type, the effect is the same as writing a string whose length is the number of elements in the packed string type.

Write with one of the Boolean type values: If OutExpr is of type Boolean, the effect is the same as writing the strings true or false, depending on the value of OutExpr.

Note: When using Write the file must be open for output.

Delphi Examples:

```
{
This example reads a file one character at a time and writes
it into a file of your chosing.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F1, F2: TextFile;
  Ch: AnsiChar;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F1, OpenDialog1.Filename);
    Reset(F1);
    if SaveDialog1.Execute then begin
      AssignFile(F2, SaveDialog1.Filename);
      Rewrite(F2);
      while not Eof(F1) do
      begin
        Read(F1, Ch);
        Write(F2, Ch);
      end;
      CloseFile(F2);
```

```
    end;
    CloseFile(F1);
  end;
end;
```

See Also

WriteLn (see page 3710)

35.3.197 System.WriteLn Function

Writes to a text file and adds an end-of-line marker.

Pascal

```
procedure WriteLn(P1); overload;
procedure WriteLn(P1; Pn); overload;
procedure WriteLn(var F: Text; P1); overload;
procedure WriteLn(var F: Text; P1; Pn); overload;
```

C++

```
WriteLn( P1);
WriteLn( P1,  Pn);
WriteLn(Text F,  P1);
WriteLn(Text F,  P1,  Pn);
```

File

System

Description

WriteLn is an extension to the Write (see page 3707) procedure, as it is defined for text files.

The syntax shown here for the WriteLn procedure is an example that illustrates that WriteLn can take a variable number of arguments.

After executing Write (see page 3707), WriteLn writes an end-of-line marker (line feed or carriage return/line feed) to the file.

If F is omitted, the global variable Output (see page 3833) is used, which accesses the processes standard input file. Use of Output (see page 3833) in GUI applications raises special issues.

Delphi Examples:

```
{
This example assumes you are compiling as a console
application.  Notice how the form deactivates when the button
click procedure exits.
}

{$APPTYPE CONSOLE}

procedure TForm1.Button1Click(Sender: TObject);
var
  s : string;
begin
  Write('Enter a line of text: ');
  Readln(s);
  Writeln('You typed: ',s);
  Writeln('Hit <Enter> to exit');
  Readln;
end;
```

See Also

Write (see page 3707)

SetLineBreakStyle (see page 3659)

Output (see page 3833)

35.3.198 System.YieldProcessor Function

Pascal

```
procedure YieldProcessor;
```

C++

```
YieldProcessor();
```

File

System

Description

This is function System.YieldProcessor.

35.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TChunkStatus (see page 3714)	TChunkStatus enumerates the possible states of a chunk of memory.
	TMinimumBlockAlignment (see page 3732)	TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.
	TRuntimeError (see page 3734)	TRuntimeError is an enumeration of internal runtime error codes.
	TTextLineBreakStyle (see page 3737)	TTextLineBreakStyle indicates the default set of characters used to divide lines in text.

Records

	Name	Description
	PackageInfoTable (see page 3713)	A variable of type PackageInfoTable provides information on initializing or finalizing a series of package units.
	PackageUnitEntry (see page 3713)	A variable of type PackageUnitEntry provides information on initializing or finalizing a package unit.
	TCallDesc (see page 3714)	TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 3758) is a pointer to this descriptior.
	TCVModInfo (see page 3715)	This is record System.TCVModInfo.
	TDispatchMessage (see page 3716)	This is record System.TDispatchMessage.
	TDispDesc (see page 3716)	This is record System.TDispDesc.
	TDynArrayTypeInfo (see page 3716)	This is record System.TDynArrayTypeInfo.
	TFileRec (see page 3717)	TFileRec is the internal format for typed and untyped files in the Delphi language.
	TGUID (see page 3718)	TGUID is a structured form of the value that uniquely identifies an interface.
	THeapStatus (see page 3720)	System::THeapStatus represents information about the global heap.
	TInitContext (see page 3721)	This is record System.TInitContext.
	TInterfaceEntry (see page 3722)	TInterfaceEntry encapsulates the information needed to dispatch interface calls.
	TInterfaceTable (see page 3723)	TInterfaceTable lists the interfaces that a class supports, and PInterfaceTable (see page 3766) points to this list.
	TLibModule (see page 3723)	System::TLibModule describes a module that is compiled with the application.
	TMemoryManager (see page 3724)	System::TMemoryManager defines the memory block entry points.
	TMemoryManagerEx (see page 3725)	System::TMemoryManagerEx defines extended memory block entry points.
	TMemoryManagerState (see page 3730)	System::TMemoryManagerState holds statistics about memory allocations.
	TMethod (see page 3731)	TMethod stores Code and Data fields to represent a method.
	TModuleUnloadRec (see page 3732)	This is record System::TModuleUnloadRec.
	TMonitorSupport (see page 3733)	This is record System.TMonitorSupport.
	TResStringRec (see page 3733)	This is record System::TResStringRec.
	TSmallBlockTypeState (see page 3736)	System::TSmallBlockTypeState holds statistics about a small memory block.
	TTextRec (see page 3738)	TTextRec is the internal format for text files.
	TVarArray (see page 3739)	This is the descriptor type for safe arrays.
	TVarArrayBound (see page 3740)	Gives bound information for one dimension of a safe array.
	TVarData (see page 3740)	TVarData represents Variant (see page 3421) information.
	TVariantManager (see page 3745)	System::TVariantManager defines entry points for variant conversion routines and other procedures that define variant behavior.
	TVarRec (see page 3747)	System::TVarRec is used inside a function with an untyped array parameter.

Structures

	Name	Description
	THeapBlock (see page 3720)	This is record System::THeapBlock.
	TCallDesc (see page 3749)	This is type System::TCallDesc.

	TDispDesc (see page 3749)	This is type System::TDispDesc.

35.4.1 System.PackageInfoTable Record

A variable of type PackageInfoTable provides information on initializing or finalizing a series of package units.

Pascal

```
PackageInfoTable = packed record
  UnitCount: Integer;
  UnitInfo: PUnitEntryTable;
end;
```

C++

```
struct PackageInfoTable {
  int UnitCount;
  PUnitEntryTable UnitInfo;
};
```

File

System

Description

A variable of type PackageInfoTable provides information on initializing or finalizing a series of package units.

PackageInfoTable is a record containing a pointer to an array of type UnitEntryTable (see page 3804), as the field `UnitInfo`, and also the number of entries in this array, as the field `UnitCount`.

The value of `UnitCount` must always be greater than or equal to 1.

See Also

UnitEntryTable (see page 3804)

35.4.2 System.PackageUnitEntry Record

A variable of type PackageUnitEntry provides information on initializing or finalizing a package unit.

Pascal

```
PackageUnitEntry = packed record
  FInit: Pointer;
  Init: Pointer;
end;
```

C++

```
struct PackageUnitEntry {
  void * FInit;
  void * Init;
};
```

File

System

Description

PackageUnitEntry is a record containing the pointer fields `Init` and `FInit`. These fields provide information on initializing or

finalizing a package unit.

See Also

UnitEntryTable (see page 3804)

35.4.3 System.TCallDesc Record

TCallDesc is the dispatch call descriptor for variants, and PCallDesc (see page 3758) is a pointer to this descriptior.

Pascal

```
TCallDesc = packed record
  CallType: Byte;
  ArgCount: Byte;
  NamedArgCount: Byte;
  ArgTypes: array[0..255] of Byte;
end;
```

C++

```
struct TCallDesc {
  Byte CallType;
  Byte ArgCount;
  Byte NamedArgCount;
  array[0..255] of Byte ArgTypes;
};
```

File

System

Description

TCallDesc identifies a method call that was made using a Variant (see page 3421).

CallType identifies the method that was called.

ArgCount indicates the total number of arguments passed to the method.

NamedArgCount

ArgTypes is an array giving the type codes for each argument.

See Also

System::PCallDesc (see page 3758)

35.4.4 System.TChunkStatus Enumeration

TChunkStatus enumerates the possible states of a chunk of memory.

Pascal

```
TChunkStatus = (
  csUnallocated,
  csAllocated,
  csReserved,
  csSysAllocated,
  csSysReserved
);
```

C++

```
enum TChunkStatus {
  csUnallocated,
  csAllocated,
  csReserved,
  csSysAllocated,
  csSysReserved
};
```

File

System

Description

GetMemoryMap (see page 3592) returns a map of the address space of the current process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. TMemoryMap (see page 3791) obtains a TChunkStatus status value for each of them in the TMemoryMap (see page 3791) array. TChunkStatus defines the following values :

Status value	Meaning
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

GetMemoryMap (see page 3592)

TMemoryMap (see page 3791)

35.4.5 System.TCVModInfo Record

Pascal

```
TCVModInfo = packed record
  ModName: PAnsiChar;
  LibName: PAnsiChar;
  UserData: Pointer;
end;
```

C++

```
struct TCVModInfo {
  PAnsiChar ModName;
  PAnsiChar LibName;
  void * UserData;
};
```

File

System

Description

This is record System.TCVModInfo.

35.4.6 System.TDispatchMessage Record

Pascal

```
TDispatchMessage = record
  MsgID: Word;
end;
```

C++

```
struct TDispatchMessage {
  Word MsgID;
};
```

File

System

Description

This is record System.TDispatchMessage.

35.4.7 System.TDispDesc Record

Pascal

```
TDispDesc = packed record
  DispID: Integer;
  ResType: Byte;
  CallDesc: TCallDesc;
end;
```

C++

```
struct TDispDesc {
  int DispID;
  Byte ResType;
  TCallDesc CallDesc;
};
```

File

System

Description

This is record System.TDispDesc.

35.4.8 System.TDynArrayTypeInfo Record

Pascal

```
TDynArrayTypeInfo = packed record
  kind: Byte;
  name: string[0];
  elSize: Longint;
  elType: ^PDynArrayTypeInfo;
```

```
    varType: Integer;
  end;
```

C++

```
  struct TDynArrayTypeInfo {
    Byte kind;
    string[0] name;
    Longint elSize;
    ^PDynArrayTypeInfo elType;
    int varType;
  };
```

File

System

Description

This is record System.TDynArrayTypeInfo.

35.4.9 System.TFileRec Record

TFileRec is the internal format for typed and untyped files in the Delphi language.

Pascal

```
  TFileRec = packed record
    Handle: Integer;
    Mode: Word;
    Flags: Word;
    case Byte of
      0: (RecSize: Cardinal;);
      1: (BufSize: Cardinal;
          BufPos: Cardinal;
          BufEnd: Cardinal;
          BufPtr: PAnsiChar;
          OpenFunc: Pointer;
          InOutFunc: Pointer;
          FlushFunc: Pointer;
          CloseFunc: Pointer;
          UserData: array[1..32] of Byte;
          Name: array[0..259] of WideChar;);
  end;
```

C++

```
  struct TFileRec {
    int Handle;
    Word Mode;
    Word Flags;
    unsigned RecSize;
    unsigned BufSize;
    unsigned BufPos;
    unsigned BufEnd;
    PAnsiChar BufPtr;
    void * OpenFunc;
    void * InOutFunc;
    void * FlushFunc;
    void * CloseFunc;
    array[1..32] of Byte UserData;
    array[0..259] of WideChar Name;
  };
```

File

System

Description

TFileRec is the internal format for typed and untyped files. TFileRec enables you to typecast a Delphi file variable to access its internal fields.

Note: You would normally never declare a variable of this type.

35.4.10 System.TGUID Record

TGUID is a structured form of the value that uniquely identifies an interface.

Pascal

```
TGUID = packed record
  D1: LongWord;
  D2: Word;
  D3: Word;
  D4: array[0..7] of Byte;
end;
```

C++

```
struct TGUID {
  LongWord D1;
  Word D2;
  Word D3;
  array[0..7] of Byte D4;
};
```

File

System

Description

An interface declaration can specify a globally unique identifier (GUID), represented by a string literal enclosed in brackets immediately preceding the member list. The GUID part of the declaration must have the form :

['{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}']

where each x is a hexadecimal digit (0 through 9 or A through F). On Windows, the Type Library editor automatically generates GUIDs for new interfaces. You can also generate GUIDs by pressing Ctrl+Shift+G in the Code editor.

TGUID provides a structured access to this GUID :

D1 maps to the first 8 hexadecimal digits.

D2 maps to the next 4 hexadecimal digits.

D3 maps to the next 4 hexadecimal digits.

D4 maps to the final 8 hexadecimal digits.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
```

```
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
```

```
    if IsEqualGUID(MyGuid0, MyGuid1) then
       Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                       '2 randomly generated GUIDs cannot be equal!');
  end;
```

35.4.11 System::THeapBlock Structure

Pascal

```
  THeapBlock = record
    Start: Pointer;
    Size: unsigned int;
  end;
```

C++

```
  struct THeapBlock {
    void * Start;
    unsigned int Size;
  };
```

File

sysclass.h

Description

This is record System::THeapBlock.

35.4.12 System::THeapStatus Record

System::THeapStatus represents information about the global heap.

Pascal

```
  THeapStatus = record
    TotalAddrSpace: Cardinal;
    TotalUncommitted: Cardinal;
    TotalCommitted: Cardinal;
    TotalAllocated: Cardinal;
    TotalFree: Cardinal;
    FreeSmall: Cardinal;
    FreeBig: Cardinal;
    Unused: Cardinal;
    Overhead: Cardinal;
    HeapErrorCode: Cardinal;
  end;
```

C++

```
  struct THeapStatus {
    Cardinal TotalAddrSpace;
    Cardinal TotalUncommitted;
    Cardinal TotalCommitted;
    Cardinal TotalAllocated;
    Cardinal TotalFree;
    Cardinal FreeSmall;
    Cardinal FreeBig;
    Cardinal Unused;
    Cardinal Overhead;
    Cardinal HeapErrorCode;
  };
```

File

System.pas

Description

System::THeapStatus represents information about the current memory manager. These are the meanings of the values in each field:

Field	Meaning
TotalAddrSpace	The (current) total address space available to your program, in bytes. This will grow as your program's dynamic memory usage grows.
TotalUncommitted	The total number of bytes (of TotalAddrSpace) for which space has not been allocated in the swap file.
TotalCommitted	The total number of bytes (of TotalAddrSpace) for which space has been allocated in the swap file. Note : TotalUncommitted + TotalCommitted = TotalAddrSpace
TotalAllocated	The total number of bytes dynamically allocated by your program.
TotalFree	The total number of free bytes available in the (current) address space for allocation by your program. If this number is exceeded, and enough virtual memory is available, more address space will be allocated from the OS; TotalAddrSpace will be incremented accordingly.
FreeSmall	Total bytes of small memory blocks which are not currently allocated by your program.
FreeBig	Total bytes of big memory blocks which are not currently allocated by your program. Large free blocks can be created by coalescing smaller, contiguous, free blocks or by freeing a large dynamic allocation. (The exact size of the blocks is immaterial).
Unused	Total bytes which have never been allocated by your program. Note : Unused + FreeBig + FreeSmall = TotalFree These three fields (Unused, FreeBig, and FreeSmall) refer to dynamic allocation by the user program.
Overhead	The total number of bytes required by the heap manager to manage all the blocks dynamically allocated by your program.
HeapErrorCode	Indicates the current status of the heap, as internally determined.

Note: TotalAddrSpace, TotalUncommitted and TotalCommitted refer to OS memory used by the program, whereas TotalAllocated and TotalFree refer to the heap memory used within the program by dynamic allocations. Therefore, to monitor dynamic memory used in your program use TotalAllocated and TotalFree.

35.4.13 System.TInitContext Record

Pascal

```
TInitContext = record
  OuterContext: PInitContext;
  ExcFrame: Pointer;
  InitTable: PackageInfo;
  InitCount: Integer;
  Module: PLibModule;
  DLLSaveEBP: Pointer;
  DLLSaveEBX: Pointer;
  DLLSaveESI: Pointer;
  DLLSaveEDI: Pointer;
  ExitProcessTLS: procedure;
  DLLInitState: Byte;
  PkgLoaderLock: Pointer;
end;
```

C++

```
struct TInitContext {
  PInitContext OuterContext;
  void * ExcFrame;
  PackageInfo InitTable;
  int InitCount;
  PLibModule Module;
  void * DLLSaveEBP;
  void * DLLSaveEBX;
  void * DLLSaveESI;
  void * DLLSaveEDI;
  procedure ExitProcessTLS;
  Byte DLLInitState;
  void * PkgLoaderLock;
};
```

File

System

Description

This is record System.TInitContext.

35.4.14 System.TInterfaceEntry Record

TInterfaceEntry encapsulates the information needed to dispatch interface calls.

Pascal

```
TInterfaceEntry = packed record
  IID: TGUID;
  VTable: Pointer;
  IOffset: Integer;
  ImplGetter: Integer;
end;
```

C++

```
struct TInterfaceEntry {
  TGUID IID;
  void * VTable;
  int IOffset;
  int ImplGetter;
};
```

File

System

Description

PInterfaceEntry (see page 3766) is a pointer to a TInterfaceEntry value. TInterfaceEntry contains the following fields:

Field	Meaning
IID	The GUID that uniquely identifies the interface.
VTable	The VTable to use for dispatching interface calls.
IOffset	The offset of the interface in the implementing object.
ImplGetter	A method pointer for accessing the interface if IOffset is not available.

See Also

PInterfaceEntry (see page 3766)

35.4.15 System.TInterfaceTable Record

TInterfaceTable lists the interfaces that a class supports, and PInterfaceTable (see page 3766) points to this list.

Pascal

```
TInterfaceTable = packed record
  EntryCount: Integer;
  Entries: array[0..9999] of TInterfaceEntry;
end;
```

C++

```
struct TInterfaceTable {
  int EntryCount;
  array[0..9999] of TInterfaceEntry Entries;
};
```

File

System

Description

PInterfaceTable (see page 3766) is a pointer to a TInterfaceTable value. TInterfaceTable contains the following fields:

Field	Meaning
EntryCount	The number of interface descriptors in the table.
Entries	An array of interface descriptors.

See Also

PInterfaceTable (see page 3766)

35.4.16 System::TLibModule Record

System::TLibModule describes a module that is compiled with the application.

Pascal

```
TLibModule = record
  Next: PLibModule;
  Instance: LongWord;
  CodeInstance: LongWord;
  DataInstance: LongWord;
  ResInstance: LongWord;
  Reserved: Integer;
end;
```

C++

```
struct TLibModule {
  PLibModule Next;
  LongWord Instance;
```

```
    LongWord CodeInstance;
    LongWord DataInstance;
    LongWord ResInstance;
    Integer Reserved;
  };
```

File

System.pas

Description

System::TLibModule is an entry in a linked list of data structures that describe the modules in an application.

Next is a pointer to the next entry in the list.

The **Instance** field represents the instance handle of the executable or package and the instance handles of the text, data, and resource associated with that executable instance.

The additional fields on Linux are used internally to access information in the module header.

See Also

PLibModule (see page 3767)

35.4.17 System::TMemoryManager Record

System::TMemoryManager defines the memory block entry points.

Pascal

```
TMemoryManager = record
  GetMem: function(Size: Integer): Pointer;
  FreeMem: function(P: Pointer): Integer;
  ReallocMem: function(P: Pointer; Size: Integer): Pointer;
end;
```

C++

```
struct TMemoryManager {
  function(Size: Integer): Pointer GetMem;
  function(P: Pointer): Integer FreeMem;
  function(P: Pointer; Size: Integer): Pointer ReallocMem;
};
```

File

System.pas

Description

The System::TMemoryManager type is used by the GetMemoryManager (see page 3587) and SetMemoryManager (see page 3660) procedures. It defines the routines that allocate and free memory.

The GetMem field (Delphi) or method (C++) must specify a function that allocates the given number of bytes and returns a pointer to the newly allocated block, as invoked by the GetMemory (see page 3585) routine.

The Size parameter passed to the GetMem function will never be zero. If the GetMem function cannot allocate a block of the given size, it should return nil (Delphi) or NULL (C++).

The FreeMem field (Delphi) or method (C++) must specify a function that deallocates the given block. The pointer parameter passed to the FreeMem function will never be nil (Delphi) or NULL (C++). If the FreeMem function successfully deallocates the given block, it should return zero. Otherwise, it should return a non-zero value. As invoked by the FreeMemory (see page 3579) routine.

The ReallocMem field (Delphi) or method (C++) must specify a function that reallocates the given block to the given new size. The pointer parameter passed to the ReallocMem function will never be nil (Delphi) or NULL (C++), and the Size parameter will never be zero. The ReallocMem function must reallocate the given block to the given new size, possibly moving the block if it cannot be resized in place. Any existing contents of the block must be preserved, but newly allocated space can be uninitialized. The ReallocMem function must return a pointer to the reallocated block, or nil (Delphi) or NULL (C++) if the block cannot be reallocated. As invoked by the ReallocMem routine.

See Also

AllocMem (see page 3515)

FreeMemory (see page 3579)

GetMemoryManager (see page 3587)

ReallocMem (see page 3637)

SetMemoryManager (see page 3660)

35.4.18 System::TMemoryManagerEx Record

System::TMemoryManagerEx defines extended memory block entry points.

Pascal

```
TMemoryManagerEx = record
  GetMem: function(Size: Integer): Pointer;
  FreeMem: function(P: Pointer): Integer;
  ReallocMem: function(P: Pointer; Size: Integer): Pointer;
  AllocMem: function(Size: Cardinal): Pointer;
  RegisterExpectedMemoryLeak: function(P: Pointer): Boolean;
  UnregisterExpectedMemoryLeak: function(P: Pointer): Boolean;
end;
```

C++

```
struct TMemoryManagerEx {
  function(Size: Integer): Pointer GetMem;
  function(P: Pointer): Integer FreeMem;
  function(P: Pointer; Size: Integer): Pointer ReallocMem;
  function(Size: Cardinal): Pointer AllocMem;
  function(P: Pointer): Boolean RegisterExpectedMemoryLeak;
  function(P: Pointer): Boolean UnregisterExpectedMemoryLeak;
};
```

File

System.pas

Description

The RegisterExpectedMemoryLeak type is used by the GetMemoryManager (see page 3587) and SetMemoryManager (see page 3660) procedures. It defines the routines that allocate and free memory.

The GetMem field (Delphi) or method (C++) must specify a function that allocates the given number of bytes and returns a pointer to the newly allocated block, as invoked by the GetMemory (see page 3585) routine.

The Size parameter passed to the GetMem function will never be zero. If the GetMem function cannot allocate a block of the given size, it should return nil (Delphi) or NULL (C++).

The FreeMem field (Delphi) or method (C++) must specify a function that deallocates the given block. The pointer parameter passed to the FreeMem function will never be nil (Delphi) or NULL (C++). If the FreeMem function successfully deallocates the

given block, it should return zero. Otherwise, it should return a non-zero value. As invoked by the FreeMemory (see page 3579) routine.

The ReallocMem field (Delphi) or method (C++) must specify a function that reallocates the given block to the given new size. The pointer parameter passed to the ReallocMem function will never be nil (Delphi) or NULL (C++), and the Size parameter will never be zero. The ReallocMem function must reallocate the given block to the given new size, possibly moving the block if it cannot be resized in place. Any existing contents of the block must be preserved, but newly allocated space can be uninitialized. The ReallocMem function must return a pointer to the reallocated block, or nil (Delphi) or NULL (C++) if the block cannot be reallocated. As invoked by the ReallocMem routine.

The AllocMem field (Delphi) or method (C++) must specify a function that allocates the given number of bytes and returns a pointer to the newly allocated block, as invoked by the AllocMem routine.

The RegisterExpectedMemoryLeak field (Delphi) or method (C++) must specify a function that registers a memory location that an application has allocated and does not expect to free, as invoked by the RegisterExpectedMemoryLeak routine.

The UnregisterExpectedMemoryLeak field (Delphi) or method (C++) must specify a function that removes a memory location from the Memory Manager's list of expected memory leaks, as invoked by the UnregisterExpectedMemoryLeak routine.

C++ Examples:

```
/*
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
*/

bool isMyMemMgr;
TMemoryManagerEx oldMemMgr;
volatile long getMemCalls;
volatile long freeMemCalls;
volatile long reallocMemCalls;
volatile long allocMemCalls;

void* __fastcall myGetMem(int size)
{
    //safely increment the counter
    InterlockedIncrement(&getMemCalls);

    //route the call
    return oldMemMgr.GetMem(size);
}

int __fastcall myFreeMem(void* p)
{
    //safely increment the counter
    InterlockedIncrement(&freeMemCalls);

    //route the call
    return oldMemMgr.FreeMem(p);
}

void* __fastcall myReallocMem(void* p, int size)
{
    //safely increment the counter
    InterlockedIncrement(&reallocMemCalls);

    //route the call
    return oldMemMgr.ReallocMem(p, size);
}

void* __fastcall myAllocMem(int size)
{
    //safely increment the counter
```

```
    InterlockedIncrement(&allocMemCalls);

    //route the call
    return oldMemMgr.AllocMem(size);
}

void __fastcall TForm3::btUseSysMemMgrClick(TObject *Sender)
{
    //switch button states
    btUseSysMemMgr->Enabled = false;
    btUseMyMemMgr->Enabled = true;

    //set the old memory manager back
    SetMemoryManager(oldMemMgr);

    isMyMemMgr = false;

    //clear out the variables
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;
}

void __fastcall TForm3::FormDestroy(TObject *Sender)
{
    //set the old memory manager back
    if (isMyMemMgr)
    {
        SetMemoryManager(oldMemMgr);
    }
}

void __fastcall TForm3::Timer1Timer(TObject *Sender)
{
    //Note that IntToStr calls will also Allocate/Free
    //memory so at each timer tick the counts will be +4;

    Form3->edGetMem->Text = IntToStr((int)getMemCalls);
    Form3->edFreeMem->Text = IntToStr((int)freeMemCalls);
    Form3->edReallocMem->Text = IntToStr((int)reallocMemCalls);
    Form3->edAllocMem->Text = IntToStr((int)allocMemCalls);
}

void __fastcall TForm3::btUseMyMemMgrClick(TObject *Sender)
{
    TMemoryManagerEx myMemMgr;

    //switch button states
    btUseSysMemMgr->Enabled = true;
    btUseMyMemMgr->Enabled = false;

    //get the old memory manager
    GetMemoryManager(oldMemMgr);

    //create out instance
    myMemMgr.GetMem = myGetMem;
    myMemMgr.FreeMem = myFreeMem;
    myMemMgr.AllocMem = myAllocMem;
    myMemMgr.ReallocMem = myReallocMem;

    //use defaults for this - not important
    myMemMgr.RegisterExpectedMemoryLeak = oldMemMgr.RegisterExpectedMemoryLeak;
    myMemMgr.UnregisterExpectedMemoryLeak = oldMemMgr.UnregisterExpectedMemoryLeak;

    //clear out the variables
```

```
    getMemCalls     = 0;
    freeMemCalls    = 0;
    reallocMemCalls = 0;
    allocMemCalls   = 0;

    //install the new memory manager
    SetMemoryManager(myMemMgr);

    isMyMemMgr = true;
}

void __fastcall TForm3::btDoSomethingClick(TObject *Sender)
{
    void *ptr;
    //perform some new and delete
    for (int i=0; i < 1000; ++i)
    {
        ptr = GetMemory(4);
        FreeMemory(ptr);
    }
}
```

Delphi Examples:

```
{
This example demonstrates the use of SetMemoryManager and
GetMemoryManager routines. Note that this example is
thread safe.
}
var
  OldMemMgr       : TMemoryManagerEx;
  GetMemCalls     : Integer;
  FreeMemCalls    : Integer;
  ReallocMemCalls : Integer;
  AllocMemCalls   : Integer;

function MyGetMem(Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.GetMem(Size);

  { Safely increment the counter }
  InterlockedIncrement(GetMemCalls);
end;

function MyFreeMem(P: Pointer): Integer;
begin
  { Route the call }
  Result := OldMemMgr.FreeMem(P);

  { Safely increment the counter }
  InterlockedIncrement(FreeMemCalls);
end;

function MyReallocMem(P: Pointer; Size: Integer): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.ReallocMem(P, Size);

  { Safely increment the counter }
  InterlockedIncrement(ReallocMemCalls);
end;

function MyAllocMem(Size: Cardinal): Pointer;
begin
  { Route the call }
  Result := OldMemMgr.AllocMem(Size);
```

```
  { Safely increment the counter }
  InterlockedIncrement(AllocMemCalls);
end;

procedure TForm2.btPerformSomethingClick(Sender: TObject);
var
  X : Integer;
  I : ^Integer;
begin
  { Perform some allocations and frees }
  for X := 0 to 999 do
  begin
    New(I);
    Dispose(I);
  end;
end;

procedure TForm2.btUseMyMemMgrClick(Sender: TObject);
var
  MyMemMgr : TMemoryManagerEx;
begin
  { Switch button states }
  btUseSystemMemMgr.Enabled := true;
  btUseMyMemMgr.Enabled := false;

  { Get the old memory manager }
  GetMemoryManager(OldMemMgr);

  { Create our instance }
  MyMemMgr.GetMem := MyGetMem;
  MyMemMgr.FreeMem := MyFreeMem;
  MyMemMgr.ReallocMem := MyReallocMem;
  MyMemMgr.AllocMem := MyAllocMem;

  { Use the defaults for this - not important }
  MyMemMgr.RegisterExpectedMemoryLeak := OldMemMgr.RegisterExpectedMemoryLeak;
  MyMemMgr.UnregisterExpectedMemoryLeak := OldMemMgr.UnregisterExpectedMemoryLeak;

  { Clear out the count variables }
  GetMemCalls := 0;
  FreeMemCalls := 0;
  ReallocMemCalls := 0;
  AllocMemCalls := 0;

  { Install the new memory manager }
  SetMemoryManager(MyMemMgr);
end;

procedure TForm2.btUseSystemMemMgrClick(Sender: TObject);
begin
  { Switch button states }
  btUseSystemMemMgr.Enabled := false;
  btUseMyMemMgr.Enabled := true;

  { Set the old memory manager back! }
  SetMemoryManager(OldMemMgr);

  GetMemCalls := 0;
  FreeMemCalls := 0;
  ReallocMemCalls := 0;
  AllocMemCalls := 0;
end;

procedure TForm2.FormDestroy(Sender: TObject);
begin
```

```
    { Set the old memory manager back }
    if IsMemoryManagerSet then
       SetMemoryManager(OldMemMgr);
  end;

  procedure TForm2.Timer1Timer(Sender: TObject);
  begin
    {
    Note that IntToStr calls will also Allocate/Free
    memory so at each timer tick the counts will be increased;
    }
    Form2.edtGetMemCalls.Text := IntToStr(GetMemCalls);
    Form2.edtFreeMemCalls.Text := IntToStr(FreeMemCalls);
    Form2.edtReallocMemCalls.Text := IntToStr(ReallocMemCalls);
    Form2.edtAllocMemCalls.Text := IntToStr(AllocMemCalls);
  end;
```

See Also

AllocMem (see page 3515)

FreeMemory (see page 3579)

GetMemory (see page 3585)

GetMemoryManager (see page 3587)

ReallocMem (see page 3637)

RegisterExpectedMemoryLeak (see page 3640)

SetMemoryManager (see page 3660)

UnregisterExpectedMemoryLeak (see page 3694)

35.4.19 System::TMemoryManagerState Record

System::TMemoryManagerState holds statistics about memory allocations.

Pascal

```
  TMemoryManagerState = packed record
    SmallBlockTypeStates: TSmallBlockTypeStates;
    AllocatedMediumBlockCount: Cardinal;
    TotalAllocatedMediumBlockSize: Cardinal;
    ReservedMediumBlockAddressSpace: Cardinal;
    AllocatedLargeBlockCount: Cardinal;
    TotalAllocatedLargeBlockSize: Cardinal;
    ReservedLargeBlockAddressSpace: Cardinal;
  end;
```

C++

```
  struct TMemoryManagerState {
    TSmallBlockTypeStates SmallBlockTypeStates;
    Cardinal AllocatedMediumBlockCount;
    Cardinal TotalAllocatedMediumBlockSize;
    Cardinal ReservedMediumBlockAddressSpace;
    Cardinal AllocatedLargeBlockCount;
    Cardinal TotalAllocatedLargeBlockSize;
    Cardinal ReservedLargeBlockAddressSpace;
  };
```

File

System.pas

Description

GetMemoryManagerState (see page 3591) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a System::TMemoryManagerState record. AllocMem (see page 3515) and GetMemory (see page 3585) can be used to allocate memory blocks. The record holds the following information :

Field	Meaning
SmallBlockTypeStates	An array of small memory block information records.
AllocatedMediumBlockCount	The number of medium sized memory blocks (blocks less than 264,749 bytes in size).
TotalAllocatedMediumBlockSize	The total number of bytes oiccupied by the medium memory blocks.
ReservedMediumBlockAddressSpace	The amount of memory reserved for medium memory blocks.
AllocatedLargeBlockCount	The number of large sized memory blocks (blocks greater than 264,748 bytes in size).
TotalAllocatedLargeBlockSize	The total number of bytes oiccupied by the large memory blocks.
ReservedLargeBlockAddressSpace	The amount of memory reserved for large memory blocks.

See Also

AllocMem (see page 3515)

GetMemory (see page 3585)

GetMemoryManagerState (see page 3591)

System::TSmallBlockTypeStates (see page 3795)

35.4.20 System.TMethod Record

TMethod stores Code and Data fields to represent a method.

Pascal

```
TMethod = record
  Code: Pointer;
  Data: Pointer;
end;
```

C++

```
struct TMethod {
  void * Code;
  void * Data;
};
```

File

System

Description

The TMethod stores the Code and Data pointers for a method. This type can be used in a type cast of a method pointer to access the code and data parts of the method pointer.

You can also furnish a TMethod variable by assigning Data a pointer to an object, and assigning Code using System::TObject::MethodAddress (see page 3378), specifying the method name as a string parameter to that method.

See Also

System::TObject::MethodAddress (see page 3378)

35.4.21 System.TMinimumBlockAlignment Enumeration

TMinimumBlockAlignment defines the mimimum byte alignment for memory block allocation.

Pascal

```
TMinimumBlockAlignment = (
  mba8Byte,
  mba16Byte
);
```

C++

```
enum TMinimumBlockAlignment {
  mba8Byte,
  mba16Byte
};
```

File

System

Description

The TMinimumBlockAlignment type defines either 8 byte (**mba8Byte**) or 16 byte (**mba16Byte**) block alignments. Namely, that the memory address is divisible exactly by 8 or 16 respectively.

See Also

AllocMem (see page 3515)

GetMemory (see page 3585)

35.4.22 System::TModuleUnloadRec Record

Pascal

```
TModuleUnloadRec = record
  Next: PModuleUnloadRec;
  Proc: TModuleUnloadProcLW;
end;
```

C++

```
struct TModuleUnloadRec {
  PModuleUnloadRec Next;
  TModuleUnloadProcLW Proc;
};
```

File

System.pas

Description

This is record System::TModuleUnloadRec.

35.4.23 System.TMonitorSupport Record

Pascal

```
TMonitorSupport = record
  NewSyncObject: function: Pointer;
  FreeSyncObject: procedure (SyncObject: Pointer);
  NewWaitObject: function: Pointer;
  FreeWaitObject: procedure (WaitObject: Pointer);
  WaitAndOrSignalObject: function (SignalObject, WaitObject: Pointer; Timeout: Cardinal):
Cardinal;
end;
```

C++

```
struct TMonitorSupport {
  function: Pointer NewSyncObject;
  procedure (SyncObject: Pointer) FreeSyncObject;
  function: Pointer NewWaitObject;
  procedure (WaitObject: Pointer) FreeWaitObject;
  function (SignalObject, WaitObject: Pointer; Timeout: Cardinal): Cardinal
WaitAndOrSignalObject;
};
```

File

System

Description

This is record System.TMonitorSupport.

35.4.24 System::TResStringRec Record

Pascal

```
TResStringRec = packed record
  Module: ^Cardinal;
  Identifier: Integer;
end;
```

C++

```
struct TResStringRec {
  ^Cardinal Module;
  Integer Identifier;
};
```

File

System.pas

Description

This is record System::TResStringRec.

35.4.25 System.TRuntimeError Enumeration

TRuntimeError is an enumeration of internal runtime error codes.

Pascal

```
TRuntimeError = (
  reNone,
  reOutOfMemory,
  reInvalidPtr,
  reDivByZero,
  reRangeError,
  reIntOverflow,
  reInvalidOp,
  reZeroDivide,
  reOverflow,
  reUnderflow,
  reInvalidCast,
  reAccessViolation,
  rePrivInstruction,
  reControlBreak,
  reStackOverflow,
  reVarTypeCast,
  reVarInvalidOp,
  reVarDispatch,
  reVarArrayCreate,
  reVarNotArray,
  reVarArrayBounds,
  reAssertionFailed,
  reExternalException,
  reIntfCastError,
  reSafeCallError,
  reMonitorNotLocked,
  reNoMonitorSupport
);
```

C++

```
enum TRuntimeError {
  reNone,
  reOutOfMemory,
  reInvalidPtr,
  reDivByZero,
  reRangeError,
  reIntOverflow,
  reInvalidOp,
  reZeroDivide,
  reOverflow,
  reUnderflow,
  reInvalidCast,
  reAccessViolation,
  rePrivInstruction,
  reControlBreak,
  reStackOverflow,
  reVarTypeCast,
  reVarInvalidOp,
  reVarDispatch,
  reVarArrayCreate,
  reVarNotArray,
  reVarArrayBounds,
  reAssertionFailed,
  reExternalException,
  reIntfCastError,
```

```
    reSafeCallError,
    reMonitorNotLocked,
    reNoMonitorSupport
  };
```

File

System

Description

The TRuntimeError type is used internally within Delphi when Error (see page 3560) is used to generate a run-time exception. It defines the type of run time error that has occured. The values are :

Constant	Meaning
reNone	Undefined error.
reOutOfMemory	Out of memory
reInvalidPtr	An attempt was made to use a nil or invalid pointer.
reDivByZero	An attempt was made to divide by zero.
reRangeError	A value out of range was encountered.
reIntOverflow	An integer variable exceeded its capacity.
reInvalidOp	An invalid operation was attempted.
reZeroDivide	An attempt was made to divide by zero.
reOverflow	A floating point number exceeded its positive capacity.
reUnderflow	A floating point number exceeded its negative capacity
reInvalidCast	An invalid type cast operation was attempted.
reAccessViolation	An attempt was made to access an invalid storage address.
rePrivInstruction	An attempt was made to execute a priveleged instruction.
reControlBreak	The user pressed Ctrl-Break (see page 3531) to interrupt an operation.
reStackOverflow	The internal stack exceeded its capacity.
reVarTypeCast	An invalid variable type cast operation was attempted.
reVarInvalidOp	An attempt was made to process an invalid variable type.
reVarDispatch	
reVarArrayCreate	
reVarNotArray	
reVarArrayBounds	
reAssertionFailed	
reExternalException	An external exception was encountered.
reIntfCastError	
reSafeCallError	

Delphi Examples:

```
{
This example demostrates the use of ErrorProc and how to catch runtime
errors.
}
procedure MyErrorProc(ErrorCode : Byte; ErrorPtr : Pointer);
var
```

```
    ErrorName : String;
  begin
    { Get the "name" of the enum value using RTTI }
    ErrorName := GetEnumName(TypeInfo(TRuntimeError), ErrorCode);

    { Show an error box }
    MessageDlg(Format('Error with code %d (%s) risen at $%X.',
        [ErrorCode, ErrorName, Integer(ErrorPtr)]), mtError, [mbOK], 0);
  end;

  procedure TForm2.btIOErrorClick(Sender: TObject);
  begin
    {
    Try to write something onto the console - will raise an
    exception.
    }
    WriteLn('This will generate an error because there is no' +
            ' console attached!');
  end;

  {$OVERFLOWCHECKS ON}
  {$OPTIMIZATION OFF}
  {$HINTS OFF}
  procedure TForm2.btOverflowErrClick(Sender: TObject);
  var
    b : Cardinal;
  begin
    {
    Simulate an overflow: Note, enabled the overflow
    checking and disabled optimizations because Delphi's
    compiler will not compile this code otherwise.
    }
    b := $FFFFFFFF;
    b := b * b;
  end;
  {$HINTS ON}
  {$OPTIMIZATION ON}
  {$OVERFLOWCHECKS OFF}

  procedure TForm2.btRuntimeErrorClick(Sender: TObject);
  begin
    { Simulate an access violation }
    System.Error(reAccessViolation);
  end;

  procedure TForm2.FormCreate(Sender: TObject);
  begin
    { Register my System error proc }
    ErrorProc := MyErrorProc;
  end;
```

See Also

Error (see page 3560)

35.4.26 System::TSmallBlockTypeState Record

System::TSmallBlockTypeState holds statistics about a small memory block.

Pascal

```
  TSmallBlockTypeState = packed record
    InternalBlockSize: Cardinal;
```

```
    UseableBlockSize: Cardinal;
    AllocatedBlockCount: Cardinal;
    ReservedAddressSpace: Cardinal;
  end;
```

C++

```
  struct TSmallBlockTypeState {
    Cardinal InternalBlockSize;
    Cardinal UseableBlockSize;
    Cardinal AllocatedBlockCount;
    Cardinal ReservedAddressSpace;
  };
```

File

System.pas

Description

GetMemoryManagerState (see page 3591) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a System::TMemoryManagerState (see page 3730) record. Within that record is held a System::TSmallBlockTypeStates (see page 3795) field - an array of System::TSmallBlockTypeState records. Each System::TSmallBlockTypeState record holds the following information :

Field	Meaning
InternalBlockSize	Actual memory allocated to each small memory block - at least 4 bytes of which are not available in the UseableBlockSize.
UseableBlockSize	The useable size of each allocated small memory block.
AllocatedBlockCount	The number of small memory blocks allocated of the given InternalBlockSize.
ReservedAddressSpace	The amount memory reserved for this group of small memory blocks.

See Also

System::TSmallBlockTypeStates (see page 3795)

GetMemoryManagerState (see page 3591)

35.4.27 System.TTextLineBreakStyle Enumeration

TTextLineBreakStyle indicates the default set of characters used to divide lines in text.

Pascal

```
  TTextLineBreakStyle = (
    tlbsLF,
    tlbsCRLF
  );
```

C++

```
  enum TTextLineBreakStyle {
    tlbsLF,
    tlbsCRLF
  };
```

File

System

Description

TTextLineBreakStyle can have one of the following values:

Value	Meaning
tlbsLF	Linefeed only
tlbsCRLF	Carriage return followed by Linefeed

35.4.28 System.TTextRec Record

TTextRec is the internal format for text files.

Pascal

```
TTextRec = packed record
  Handle: Integer;
  Mode: Word;
  Flags: Word;
  BufSize: Cardinal;
  BufPos: Cardinal;
  BufEnd: Cardinal;
  BufPtr: PAnsiChar;
  OpenFunc: Pointer;
  InOutFunc: Pointer;
  FlushFunc: Pointer;
  CloseFunc: Pointer;
  UserData: array[1..32] of Byte;
  Name: array[0..259] of WideChar;
  Buffer: TTextBuf;
end;
```

C++

```
struct TTextRec {
  int Handle;
  Word Mode;
  Word Flags;
  unsigned BufSize;
  unsigned BufPos;
  unsigned BufEnd;
  PAnsiChar BufPtr;
  void * OpenFunc;
  void * InOutFunc;
  void * FlushFunc;
  void * CloseFunc;
  array[1..32] of Byte UserData;
  array[0..259] of WideChar Name;
  TTextBuf Buffer;
};
```

File

System

Description

TTextRec is the internal format of a Delphi variable of type text. You would never declare a variable of this type. However, you can use TTextRec to typecast a text file variable to access the internal data fields, such as the file name.

TextBuf is the type of the Buffer entry in the TTextRec record, and indicates the size of the buffer used.

Note: Do not use this unless you are familiar with writing text file device drivers for the Delphi language.

35.4.29 System::TVarArray Record

This is the descriptor type for safe arrays.

Pascal

```
TVarArray = packed record
  DimCount: Word;
  Flags: Word;
  ElementSize: Integer;
  LockCount: Integer;
  Data: Pointer;
  Bounds: TVarArrayBoundArray;
end;
```

C++

```
struct TVarArray {
  Word DimCount;
  Word Flags;
  Integer ElementSize;
  Integer LockCount;
  Pointer Data;
  TVarArrayBoundArray Bounds;
};
```

File

System.pas

Description

A variable of type System::TVarArray stores all the necessary information about a safe array. The meaning of each field in the System::TVarArray record is given in the following table.

Field	Meaning
DimCount	The number of dimensions of the safe array.
Flags	Flags providing different kind of information about the safe array.
ElementSize	The size in bytes of a safe array element.
LockCount	The number of times the safe array has been locked.
Data	The actual data in the safe array.
Bounds	The bounds of the safe array corresponding to all its dimensions. This is an array of System::TVarArrayBoundArray (see page 3798) type.

See Also

System::TVarArrayBoundArray (see page 3798)

SafeArrayCreate (see page 5118)

SafeArrayDestroy (see page 5119)

SafeArrayGetElemSize (see page 5122)

SafeArrayGetLBound (see page 5122)

SafeArrayGetUBound (see page 5123)

35.4.30 System::TVarArrayBound Record

Gives bound information for one dimension of a safe array.

Pascal

```
TVarArrayBound = packed record
  ElementCount: Integer;
  LowBound: Integer;
end;
```

C++

```
struct TVarArrayBound {
  Integer ElementCount;
  Integer LowBound;
};
```

File

System.pas

Description

Declare a variable of System::TVarArrayBound type to store information about the array bounds, corresponding to one dimension of a safe array.

The following list describes each field in the System::TVarArrayBound type.

Field	Meaning
ElementCount	The number of elements in the corresponding dimension.
LowBound	The low bound of the index range, in the corresponding dimension.

See Also

TVarArrayBoundArray (see page 3798)

35.4.31 System.TVarData Record

TVarData represents Variant (see page 3421) information.

Pascal

```
TVarData = packed record
  case Integer of
    0: (VType: TVarType;
        Reserved1: Word;
        Reserved2: Word;
        Reserved3: Word;);
    1: (VLongs: array[0..2] of LongInt);
        RawData: array [0..3] of LongInt););
    2: (VWords: array [0..6] of Word);
        VBytes: array [0..13] of Byte););
    varAny: (VAny: Pointer;);
    varArray: (VArray: PVarArray;);
    varBoolean: (VBoolean: WordBool;);
    varByRef: (VPointer: Pointer;);
    varByte: (VByte: Byte;);
```

```
      varCurrency: (VCurrency: Currency;);
      varDate: (VDate: TDateTime;);
      varDispatch: (VDispatch: Pointer;);
      varDouble: (VDouble: Double;);
      varError: (VError: HRESULT;);
      varInt64: (VInt64: Int64;);
      varInteger: (VInteger: Integer;);
      varLongWord: (VLongWord: LongWord;);
      varOleStr: (VOleStr: PWideChar;);
      varShortInt: (VShortInt: ShortInt;);
      varSingle: (VSingle: Single;);
      varSmallInt: (VSmallInt: SmallInt;);
      varString: (VString: Pointer;);
      varUInt64: (VUInt64: UInt64;);
      varUnknown: (VUnknown: Pointer;);
      varUString: (VUString: Pointer;);
      varWord: (VWord: Word;);
  end;
```

C++

```
  struct TVarData {
    TVarType VType;
    Word Reserved1;
    Word Reserved2;
    Word Reserved3;
    SmallInt VSmallInt;
    int VInteger;
    Single VSingle;
    Double VDouble;
    Currency VCurrency;
    TDateTime VDate;
    PWideChar VOleStr;
    void * VDispatch;
    HRESULT VError;
    WordBool VBoolean;
    void * VUnknown;
    ShortInt VShortInt;
    Byte VByte;
    Word VWord;
    LongWord VLongWord;
    Int64 VInt64;
    UInt64 VUInt64;
    void * VString;
    void * VAny;
    PVarArray VArray;
    void * VPointer;
    void * VUString;
    array[0..2] of LongInt) VLongs;
    array [0..6] of Word) VWords;
    array [0..13] of Byte) VBytes;
    array [0..3] of LongInt) RawData;
  };
```

File

System

Description

TVarData is the base class for Variant (see page 3421).

Note: The internal structure of a Variant (see page 3421) as designated by TVarData is identical to the Variant (see page 3421) type used by OLE and the Win32 APIs.

struct TVarData

  {

```
  union
      {
        Word     VType;    // Delphi-compatible - Variant Type member
        VARTYPE  vt;       // tagVARIANT compatible member
      };
      Word Reserved1;
      Word Reserved2;
      Word Reserved3;
  union
      {
        // Delphi-compatible Variant members
        Smallint      VSmallint;    //  iVal
        Integer       VInteger;     //  lVal
        Single        VSingle;      //  fltVal
        Double        VDouble;      //  dblVal
        CurrencyBase  VCurrency;    //  cyVal
        TDateTimeBase VDate;        //  date
        PWideChar     VOleStr;      //  bstrVal
        IDispatch*    VDispatch;    //  pdispVal
        HResult       VError;       //  scode
        WordBool      VBoolean;     //  boolVal
        IUnknown*     VUnknown;     //  punkVal
        Byte          VByte;        //  bVal
        Pointer       VString;      //  ??????
        PVarArray     VArray;       //  parray
        Pointer       VPointer;     //  byref
        // tagVARIANT compatible members (from OAIDL.H)
        // Allowing all types marked as [V] (may appear in a VARIANT to be initialized
        LONG          lVal;
        BYTE          bVal;
        SHORT         iVal;
        FLOAT         fltVal;
        DOUBLE        dblVal;
        VARIANT_BOOL  boolVal;
        SCODE         scode;
        CY            cyVal;
        DATE          date;
        BSTR          bstrVal;
        IUnknown     *punkVal;
        IDispatch    *pdispVal;
        SAFEARRAY    *parray;
        BYTE         *pbVal;SHORT        *piVal;
        LONG         *plVal;
        FLOAT        *pfltVal;
        DOUBLE       *pdblVal;
        VARIANT_BOOL *pboolVal;
        SCODE        *pscode;
        CY           *pcyVal;
        DATE         *pdate;
        BSTR         *pbstrVal;
        IUnknown    **ppunkVal;
        IDispatch   **ppdispVal;
        SAFEARRAY   **pparray;
        VARIANT      *pvarVal;
        PVOID         byref;
        CHAR          cVal;
        USHORT        uiVal;
        ULONG         ulVal;
        INT           intVal;
        UINT          uintVal;
        DECIMAL      *pdecVal;
        CHAR         *pcVal;
        USHORT       *puiVal;
        ULONG        *pulVal;
        INT          *pintVal;
        UINT         *puintVal;
```

```
    };
  };
```

The VType of TVarData contains the type code of the variant in the lower twelve bits (the bits defined by varTypeMask (see page 3872)).

The following values are used for VType:

#define varEmpty (see page 3865) (Byte)(0)

```
#define varNull (ByteByte_Type)(1)
#define varSmallint (ByteByte_Type)(2)
#define varInteger (ByteByte_Type)(3)
#define varSingle (ByteByte_Type)(4)
#define varDouble (ByteByte_Type)(5)
#define varCurrency (ByteByte_Type)(6)
#define varDate (ByteByte_Type)(7)
#define varOleStr (ByteByte_Type)(8)
#define varDispatch (ByteByte_Type)(9)
#define varError (ByteByte_Type)(10)
#define varBoolean (ByteByte_Type)(11)
#define varVariant (ByteByte_Type)(12)
#define varUnknown (ByteByte_Type)(13)
#define varByte (ByteByte_Type)(17)
#define varString (WordWord_Type)(256)
#define varTypeMask (WordWord_Type)(4095)
#define varArray (WordWord_Type)(8192)
#define varByRef (WordWord_Type)(16384)
#define varStrArg (ByteByte_Type)(72)
#define vtInteger (ByteByte_Type)(0)
#define vtBoolean (ByteByte_Type)(1)
#define vtChar (ByteByte_Type)(2)
#define vtExtended (ByteByte_Type)(3)
#define vtString (ByteByte_Type)(4)
#define vtPointer (ByteByte_Type)(5)
#define vtPChar (ByteByte_Type)(6)
#define vtObject (ByteByte_Type)(7)
#define vtClass (ByteByte_Type)(8)
#define vtWideChar (ByteByte_Type)(9)
#define vtPWideChar (ByteByte_Type)(10)
#define vtAnsiString (ByteByte_Type)(11)
#define vtCurrency (ByteByte_Type)(12)
#define vtVariant (ByteByte_Type)(13)
```

In addition, the varArray (see page 3861) bit may be set to indicate that the variant is an array, and the varByRef (see page 3862) bit may be set to indicate that the variant contains a reference as opposed to a value.

Word Reserved1;

```
    Word Reserved2;
    Word Reserved3;
```

The Reserved1, Reserved2, and Reserved3 members of TVarData are unused.

union { Smallint VSmallint;

```
      Integer VInteger;
      Single VSingle;
      Double VDouble;
      CurrencyBase VCurrency;
      TDateTimeBase VDate;
      PWideChar VOleStr;
      Ole2::IDispatch* VDispatch;
      Integer VError;
      WordBool VBoolean;
      Ole2::IUnknown* VUnknown;
      Byte VByte;
```

```
      Pointer VString;
      PVarArray VArray;
      Pointer VPointer;
    };
};
```

The contents of the remaining eight bytes of TVarData depend on the VType field. If neither the varArray (see page 3861) nor the varByRef (see page 3862) bits are set, the variant contains a value of the given type.

If the varArray (see page 3861) bit is set, the variant contains a pointer to a TVarArray (see page 3739) structure that defines the specifics of the array. The type of each array element is given by the varTypeMask (see page 3872) bits in VType.

If the varByRef (see page 3862) bit is set, the variant contains a reference to a value of the type given by the varTypeMask (see page 3872) and varArray (see page 3861) bits in VType.

C++ Examples:

```
/*
This example shows how to use the TVarData type in typecasts
of Variant variables to access the internals of a variable.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Variant V;
  AnsiString mystr;
  if (TVarData(V).VType == varEmpty)
    ListBox1->Items->Add("Variants start empty.");
  VarClear(V);
  TVarData(V).VType = varString;
  mystr = "Here is my string";
  std::auto_ptr<char> MyBuffer(new char[mystr.Length() + 1]);
  StrCopy(MyBuffer.get(), mystr.c_str());
  TVarData(V).VString  = MyBuffer.get();
  mystr = PAnsiChar(TVarData(V).VString);
  ListBox1->Items->Add("This variant is now a string: " + mystr);
  VarClear(V);
  TVarData(V).VType = varInteger;
  TVarData(V).VInteger = 1234567;
  ListBox1->Items->Add("This variant is now an integer: " + IntToStr(TVarData(V).VInteger));
}
```

Delphi Examples:

```
{{
This example shows how to use the TVarData type in typecasts
of Variant variables to access the internals of a variable.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  V: Variant;
  mystr: string;
  MyBuffer: PChar;
begin
  if TVarData(V).VType = varEmpty then
    ListBox1.Items.Add('Variants start empty.');
  VarClear(V);
  TVarData(V).VType := varString;
  mystr := 'Here is my string';
  MyBuffer:= GetMemory(Length(mystr));
  StrCopy(MyBuffer, PChar(mystr));
  TVarData(V).VString := MyBuffer;
  ListBox1.Items.Add('This variant is now a string: ' + PChar(TVarData(V).VString));
  VarClear(V);
  TVarData(V).VType := varInteger;
  TVarData(V).VInteger := 1234567;
```

```
  ListBox1.Items.Add('This variant is now an integer: ' + IntToStr(TVarData(V).VInteger));
end;
```

35.4.32 System::TVariantManager Record

System::TVariantManager defines entry points for variant conversion routines and other procedures that define variant behavior.

Pascal

```
TVariantManager = record
  VarClear: procedure(var V : Variant);
  VarCopy: procedure(var Dest: Variant; const Source: Variant);
  VarCopyNoInd: procedure;
  VarCast: procedure(var Dest: Variant; const Source: Variant; VarType: Integer);
  VarCastOle: procedure(var Dest: Variant; const Source: Variant; VarType: Integer);
  VarToInt: function(const V: Variant): Integer;
  VarToInt64: function(const V: Variant): Int64;
  VarToBool: function(const V: Variant): Boolean;
  VarToReal: function(const V: Variant): Extended;
  VarToCurr: function(const V: Variant): Currency;
  VarToPStr: procedure(var S; const V: Variant);
  VarToLStr: procedure(var S: string; const V: Variant);
  VarToWStr: procedure(var S: WideString; const V: Variant);
  VarToIntf: procedure(var Unknown: IInterface; const V: Variant);
  VarToDisp: procedure(var Dispatch: IDispatch; const V: Variant);
  VarToDynArray: procedure(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer);
  VarFromInt: procedure(var V: Variant; const Value: Integer; const Range: ShortInt);
  VarFromInt64: procedure(var V: Variant; const Value: Int64);
  VarFromBool: procedure(var V: Variant; const Value: Boolean);
  VarFromReal: procedure;
  VarFromTDateTime: procedure;
  VarFromCurr: procedure;
  VarFromPStr: procedure(var V: Variant; const Value: ShortString);
  VarFromLStr: procedure(var V: Variant; const Value: string);
  VarFromWStr: procedure(var V: Variant; const Value: WideString);
  VarFromIntf: procedure(var V: Variant; const Value: IInterface);
  VarFromDisp: procedure(var V: Variant; const Value: IDispatch);
  VarFromDynArray: procedure(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer);
  OleVarFromPStr: procedure(var V: OleVariant; const Value: ShortString);
  OleVarFromLStr: procedure(var V: OleVariant; const Value: string);
  OleVarFromVar: procedure(var V: OleVariant; const Value: Variant);
  OleVarFromInt: procedure(var V: OleVariant; const Value: Integer; const Range: ShortInt);
  OleVarFromInt64: procedure(var V: OleVariant; const Value: Int64);
  VarOp: procedure(var Left: Variant; const Right: Variant; OpCode: TVarOp);
  VarCmp: procedure(const Left, Right: TVarData; const OpCode: TVarOp);
  VarNeg: procedure(var V: Variant);
  VarNot: procedure(var V: Variant);
  DispInvoke: procedure(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer);
  VarAddRef: procedure(var V: Variant);
  VarArrayRedim: procedure(var A : Variant; HighBound: Integer);
  VarArrayGet: function(var A: Variant; IndexCount: Integer; Indices: Integer): Variant;
  VarArrayPut: procedure(var A: Variant; const Value: Variant; IndexCount: Integer; Indices:
Integer);
  WriteVariant: function(var T: Text; const V: Variant; Width: Integer): Pointer;
  Write0Variant: function(var T: Text; const V: Variant): Pointer;
end;
```

C++

```
struct TVariantManager {
  procedure(var V : Variant) VarClear;
  procedure(var Dest: Variant; const Source: Variant) VarCopy;
  procedure VarCopyNoInd;
```

```
    procedure(var Dest: Variant; const Source: Variant; VarType: Integer) VarCast;
    procedure(var Dest: Variant; const Source: Variant; VarType: Integer) VarCastOle;
    function(const V: Variant): Integer VarToInt;
    function(const V: Variant): Int64 VarToInt64;
    function(const V: Variant): Boolean VarToBool;
    function(const V: Variant): Extended VarToReal;
    function(const V: Variant): Currency VarToCurr;
    procedure(var S; const V: Variant) VarToPStr;
    procedure(var S: string; const V: Variant) VarToLStr;
    procedure(var S: WideString; const V: Variant) VarToWStr;
    procedure(var Unknown: IInterface; const V: Variant) VarToIntf;
    procedure(var Dispatch: IDispatch; const V: Variant) VarToDisp;
    procedure(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer) VarToDynArray;
    procedure(var V: Variant; const Value: Integer; const Range: ShortInt) VarFromInt;
    procedure(var V: Variant; const Value: Int64) VarFromInt64;
    procedure(var V: Variant; const Value: Boolean) VarFromBool;
    procedure VarFromReal;
    procedure VarFromTDateTime;
    procedure VarFromCurr;
    procedure(var V: Variant; const Value: ShortString) VarFromPStr;
    procedure(var V: Variant; const Value: string) VarFromLStr;
    procedure(var V: Variant; const Value: WideString) VarFromWStr;
    procedure(var V: Variant; const Value: IInterface) VarFromIntf;
    procedure(var V: Variant; const Value: IDispatch) VarFromDisp;
    procedure(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer) VarFromDynArray;
    procedure(var V: OleVariant; const Value: ShortString) OleVarFromPStr;
    procedure(var V: OleVariant; const Value: string) OleVarFromLStr;
    procedure(var V: OleVariant; const Value: Variant) OleVarFromVar;
    procedure(var V: OleVariant; const Value: Integer; const Range: ShortInt) OleVarFromInt;
    procedure(var V: OleVariant; const Value: Int64) OleVarFromInt64;
    procedure(var Left: Variant; const Right: Variant; OpCode: TVarOp) VarOp;
    procedure(const Left, Right: TVarData; const OpCode: TVarOp) VarCmp;
    procedure(var V: Variant) VarNeg;
    procedure(var V: Variant) VarNot;
    procedure(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params: Pointer)
DispInvoke;
    procedure(var V: Variant) VarAddRef;
    procedure(var A : Variant; HighBound: Integer) VarArrayRedim;
    function(var A: Variant; IndexCount: Integer; Indices: Integer): Variant VarArrayGet;
    procedure(var A: Variant; const Value: Variant; IndexCount: Integer; Indices: Integer)
VarArrayPut;
    function(var T: Text; const V: Variant; Width: Integer): Pointer WriteVariant;
    function(var T: Text; const V: Variant): Pointer Write0Variant;
  };
```

File

System.pas

Description

The System::TVariantManager type is used by the GetVariantManager (see page 3594) and SetVariantManager (see page 3668) procedures. It defines routines that perform Variant (see page 3421) type conversions and other tasks that implement Variant (see page 3421) behavior.

Note: Although a version of System::TVariantManager is declared in sysclass.h, it should not be used in C++ code. The use of a Variant (see page 3421) manager to customize the behavior of Variants (see page 4973) is not fully supported in C++.

See Also

GetVariantManager (see page 3594)

SetVariantManager (see page 3668)

35.4.33 System::TVarRec Record

System::TVarRec is used inside a function with an untyped array parameter.

Pascal

```
TVarRec = record
  case Byte of
    vtAnsiString: (VAnsiString: Pointer;);
    vtBoolean: (VBoolean: Boolean;);
    vtChar: (VChar: AnsiChar;);
    vtClass: (VClass: TClass;);
    vtCurrency: (VCurrency: PCurrency;);
    vtExtended: (VExtended: PExtended;);
    vtInt64: (VInt64: PInt64;);
    vtInteger: (VInteger: Integer;
                VType: Byte;);
    vtInterface: (VInterface: Pointer;);
    vtObject: (VObject: TObject;);
    vtPChar: (VPChar: PAnsiChar;);
    vtPointer: (VPointer: Pointer;);
    vtPWideChar: (VPWideChar: PWideChar;);
    vtString: (VString: PShortString;);
    vtUnicodeString: (VUnicodeString: Pointer;);
    vtVariant: (VVariant: PVariant;);
    vtWideChar: (VWideChar: WideChar;);
    vtWideString: (VWideString: Pointer;);
end;
```

C++

```
struct TVarRec {
  Integer VInteger;
  Byte VType;
  Boolean VBoolean;
  AnsiChar VChar;
  PExtended VExtended;
  PShortString VString;
  Pointer VPointer;
  PAnsiChar VPChar;
  TObject VObject;
  TClass VClass;
  WideChar VWideChar;
  PWideChar VPWideChar;
  Pointer VAnsiString;
  PCurrency VCurrency;
  PVariant VVariant;
  Pointer VInterface;
  Pointer VWideString;
  PInt64 VInt64;
  Pointer VUnicodeString;
};
```

File

System.pas

Description

The System::TVarRec type is used inside a function with a parameter type of array of const in Delphi. In Delphi, the compiler automatically translates each element in the array to a System::TVarRec value. In C++, the function parameter appears as an array of System::TVarRec values.

Use System::TVarRec to work with individual elements of the array when implementing the function that has an array of const parameter.

The VType field indicates the simple type of each System::TVarRec value. The variable type constants represent the values passed in the tag of the System::TVarRec structure. These constants, listed below, are defined in the System (see page 3159) unit.

Delphi constants :

vtInteger (see page 3886)	= 0;
vtBoolean (see page 3883)	= 1;
vtCharacter	= 2;
vtExtended (see page 3885)	= 3;
vtString (see page 3889)	= 4;
vtPointer (see page 3888)	= 5;
vtPChar (see page 3888)	= 6;
vtObject (see page 3887)	= 7;
vtClass (see page 3884)	= 8;
vtWideChar (see page 3890)	= 9;
vtPWideChar (see page 3889)	= 10;
vtAnsiString (see page 3883)	= 11;
vtCurrency (see page 3885)	= 12;
vtVariant (see page 3890)	= 13;
vtInterface (see page 3887)	= 14;
vtWideString (see page 3891)	= 15;
vtInt64 (see page 3886)	= 16;

C++ constants :

static const Shortint vtInteger (see page 3886)	= 0x0;
static const Shortint vtBoolean (see page 3883)	= 0x1;
static const Shortint vtCharacter	= 0x2;
static const Shortint vtExtended (see page 3885)	= 0x3;
static const Shortint vtString (see page 3889)	= 0x4;
static const Shortint vtPointer (see page 3888)	= 0x5;
static const Shortint vtPChar (see page 3888)	= 0x6;
static const Shortint vtObject (see page 3887)	= 0x7;
static const Shortint vtClass (see page 3884)	= 0x8;
static const Shortint vtWideChar (see page 3890)	= 0x9;
static const Shortint vtPWideChar (see page 3889)	= 0xa;
static const Shortint vtAnsiString (see page 3883)	= 0xb;
static const Shortint vtCurrency (see page 3885)	= 0xc;
static const Shortint vtVariant (see page 3890)	= 0xd;

static const Shortint vtInterface (see page 3887)	= 0xe;
static const Shortint vtWideString (see page 3891)	= 0xf;
static const Shortint vtInt64 (see page 3886)	= 0x10;

Note: In C++, you can create an array of System::TVarRec values using the ARRAYOFCONST macro, which is defined in sysopen.h.

35.4.34 System::TCallDesc Structure

Pascal

```
TCallDesc = struct TCallDesc@1;
```

C++

```
typedef struct {
  unsigned char CallType;
  unsigned char ArgCount;
  unsigned char NamedArgCount;
  unsigned char ArgTypes[256];
} TCallDesc, * PCallDesc;
```

File

systvar.h

Description

This is type System::TCallDesc.

35.4.35 System::TDispDesc Structure

Pascal

```
TDispDesc = struct TDispDesc@1;
```

C++

```
typedef struct {
  int DispID;
  unsigned char ResType;
  TCallDesc CallDesc;
} TDispDesc, * PDispDesc;
```

File

systvar.h

Description

This is type System::TDispDesc.

35.5 Types

The following table lists types in this documentation.

Types

Name	Description
CppLongInt (see page 3752)	This is type System.CppLongInt.
CppULongInt (see page 3753)	This is type System.CppULongInt.
GetPackageInfoTable (see page 3753)	This is type System.GetPackageInfoTable.
HGLOBAL (see page 3753)	This is type System.HGLOBAL.
HINST (see page 3754)	This is type System.HINST.
HMODULE (see page 3754)	This is type System.HMODULE.
HRESULT (see page 3754)	This is type System.HRESULT.
HRSRC (see page 3755)	This is type System.HRSRC.
Int16 (see page 3755)	Alias for `SmallInt`.
Int32 (see page 3755)	Alias for `Integer`.
Int8 (see page 3756)	Alias for `ShortInt`.
IntegerArray (see page 3756)	This is type System.IntegerArray.
IUnknown (see page 3756)	IUnknown is a special base interface.
PackageInfo (see page 3757)	PackageInfo is a pointer to a PackageInfoTable (see page 3713) variable.
PAnsiString (see page 3757)	PAnsiString is a pointer to an `AnsiString` variable.
PBoolean (see page 3758)	PBoolean is a pointer to a `Boolean` variable.
PByte (see page 3758)	PByte is a pointer to a `Byte` variable.
PCallDesc (see page 3758)	This is type System::PCallDesc.
PCardinal (see page 3759)	PCardinal is a pointer to a `Cardinal` variable.
PComp (see page 3759)	PComp is a pointer to a `Comp` variable.
PCurrency (see page 3759)	PCurrency is a pointer to a `Currency (see page 3235)` variable.
PCVModInfo (see page 3760)	This is type System.PCVModInfo.
PDate (see page 3760)	PDate is a pointer to a Date variable.
PDateTime (see page 3761)	PDateTime is a pointer to a TDateTime (see page 3323) variable.
PDispatch (see page 3761)	This is type System.PDispatch.
PDispDesc (see page 3761)	This is type System::PDispDesc.
PDouble (see page 3762)	PDouble is a pointer to a `Double` variable.
PDynArrayTypeInfo (see page 3762)	This is type System.PDynArrayTypeInfo.
PError (see page 3762)	This is type System.PError.
PExtended (see page 3763)	PExtended is a pointer to an `Extended` variable.
PGUID (see page 3763)	PGUID is a pointer to a TGUID (see page 3718) variable.
PInitContext (see page 3765)	This is type System.PInitContext.
PInt64 (see page 3765)	PInt64 is a pointer to an `Int64` variable.
PInteger (see page 3765)	PInteger is a pointer to an `Integer` variable.
PIntegerArray (see page 3766)	PIntegerArray is a pointer to an `IntegerArray (see page 3756)` variable.
PInterfaceEntry (see page 3766)	PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3722) variable.
PInterfaceTable (see page 3766)	PInterfaceTable is a pointer to a TInterfaceTable (see page 3723) variable.
PLibModule (see page 3767)	PLibModule is a pointer to a System::TLibModule (see page 3723) variable.
PLongint (see page 3767)	PLongint is a pointer to a `LongInt` variable.
PLongWord (see page 3768)	PLongWord is a pointer to a `LongWord` variable.
PMemoryManager (see page 3768)	System::PMemoryManager is a pointer to a System::TMemoryManager (see page 3724) variable.
PMemoryManagerEx (see page 3768)	System::PMemoryManagerEx is a pointer to a System::TMemoryManagerEx (see page 3725) variable.

PModuleUnloadRec (see page 3769)	This is type System::PModuleUnloadRec.
PMonitor (see page 3769)	This is type System.PMonitor.
PMonitorSupport (see page 3769)	This is type System.PMonitorSupport.
PointerArray (see page 3770)	This is type System.PointerArray.
POleVariant (see page 3770)	POleVariant is a pointer to a System::OleVariant (see page 3274) variable.
PPAnsiChar (see page 3770)	This is type System.PPAnsiChar.
PPChar (see page 3771)	This is type System.PPChar.
PPCharArray (see page 3771)	This is type System.PPCharArray.
PPDispatch (see page 3771)	This is type System.PPDispatch.
PPMonitor (see page 3772)	This is type System.PPMonitor.
PPointer (see page 3772)	PPointer is a pointer to a `Pointer` variable.
PPointerArray (see page 3772)	PPointerArray is a pointer to a PointerArray (see page 3770) variable.
PPUnknown (see page 3773)	This is type System.PPUnknown.
PPWideChar (see page 3773)	This is type System.PPWideChar.
PRawByteString (see page 3773)	This is type System.PRawByteString.
PResStringRec (see page 3774)	This is type System::PResStringRec.
PShortInt (see page 3774)	PShortInt is a pointer to a `ShortInt` variable.
PShortString (see page 3774)	PShortString is a pointer to a `ShortString` variable.
PSingle (see page 3775)	PSingle is a pointer to a `Single` variable.
PSmallBlockTypeState (see page 3775)	This is type System::PSmallBlockTypeState.
PSmallInt (see page 3775)	PSmallInt is a pointer to a `SmallInt` variable.
PString (see page 3776)	PString is a pointer to a `String` variable.
PTextBuf (see page 3776)	This is type System.PTextBuf.
PUCS2Char (see page 3776)	This is type System.PUCS2Char.
PUCS4Char (see page 3777)	This is type System.PUCS4Char.
PUCS4CharArray (see page 3777)	This is type System.PUCS4CharArray.
PUInt64 (see page 3777)	This is type System.PUInt64.
PUnicodeString (see page 3778)	This is type System.PUnicodeString.
PUnitEntryTable (see page 3778)	PUnitEntryTable is a pointer to a UnitEntryTable (see page 3804) variable.
PUnknown (see page 3778)	PUnknown is a pointer to an IUnknown (see page 3756) variable.
PUTF8String (see page 3779)	This is type System.PUTF8String.
PVarArray (see page 3779)	Pointer type to a TVarArray (see page 3739).
PVarArrayBoundArray (see page 3779)	This is type System.PVarArrayBoundArray.
PVarArrayCoorArray (see page 3780)	Pointer type to a TVarArrayCoorArray (see page 3799).
PVarData (see page 3780)	This is type System.PVarData.
PVariant (see page 3780)	PVariant is a pointer to a System::Variant (see page 3421) variable.
PVariantManager (see page 3781)	This is type System::PVariantManager.
PVarRec (see page 3781)	System::PVarRec is a pointer to a System::TVarRec (see page 3747) variable.
PWideString (see page 3782)	PWideString is a pointer to a `WideString` (see page 3467) variable.
PWord (see page 3782)	PWord is a pointer to a `Word` variable.
PWordBool (see page 3782)	PWordBool is a pointer to a `WordBool` variable.
RawByteString (see page 3783)	RawByteString enables the passing of string data of any code page without doing any codepage conversions.
ResourceString (see page 3783)	This is type System::ResourceString.
TAssertErrorProc (see page 3783)	This is type System.TAssertErrorProc.
TBooleanDynArray (see page 3784)	This is type System::TBooleanDynArray.
TBoundArray (see page 3784)	This is type System.TBoundArray.
TByteDynArray (see page 3784)	This is type System::TByteDynArray.
TCardinalDynArray (see page 3785)	This is type System::TCardinalDynArray.
TClass (see page 3785)	TClass defines the metaclass for System::TObject (see page 3354).
TClassHelperBaseClass (see page 3786)	This is type System.TClassHelperBaseClass.
TDate (see page 3786)	TDate is used to represent a date.
TDateTime (see page 3787)	TDateTime represents a date-and-time value in the Delphi language.
TDLLProc (see page 3788)	This is type System.TDLLProc.
TDLLProcEx (see page 3788)	This is type System.TDLLProcEx.

TDoubleDynArray (see page 3788)	This is type System::TDoubleDynArray.
TEnumModuleFunc (see page 3789)	This is type System.TEnumModuleFunc.
TEnumModuleFuncLW (see page 3789)	This is type System.TEnumModuleFuncLW.
TFileIOFunc (see page 3789)	This is type System.TFileIOFunc.
THandle (see page 3790)	THandle is a general purpose data reference (handle).
TInt64DynArray (see page 3790)	This is type System::TInt64DynArray.
TIntegerDynArray (see page 3790)	This is type System::TIntegerDynArray.
TInterfacedClass (see page 3791)	TInterfacedClass defines the metaclass for System::TInterfacedObject (see page 3342).
TLongWordDynArray (see page 3791)	This is type System::TLongWordDynArray.
TMemoryMap (see page 3791)	TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 3714) values.
TModuleUnloadProc (see page 3792)	TModuleUnloadProc defines a procedure that can be called during package unload.
TModuleUnloadProcLW (see page 3793)	TModuleUnloadProcLW dehines a procedure that can be called during package unload.
TPCharArray (see page 3793)	This is type System.TPCharArray.
TResourceHandle (see page 3793)	This is type System.TResourceHandle.
TSafeCallErrorProc (see page 3794)	This is type System.TSafeCallErrorProc.
TShortIntDynArray (see page 3794)	This is type System::TShortIntDynArray.
TSingleDynArray (see page 3794)	This is type System::TSingleDynArray.
TSmallBlockTypeStates (see page 3795)	System::TSmallBlockTypeStates is an array of small memory block information records.
TSmallIntDynArray (see page 3795)	This is type System::TSmallIntDynArray.
TStringDynArray (see page 3796)	This is type System::TStringDynArray.
TSystemThreadEndProc (see page 3796)	This is type System.TSystemThreadEndProc.
TSystemThreadFuncProc (see page 3796)	This is type System.TSystemThreadFuncProc.
TTextBuf (see page 3797)	This is type System.TTextBuf.
TTextIOFunc (see page 3797)	This is type System.TTextIOFunc.
TThreadFunc (see page 3797)	System::TThreadFunc defines a new thread of execution.
TTime (see page 3798)	TTime is used to represent a time.
TUCS4CharArray (see page 3798)	This is type System.TUCS4CharArray.
TVarArrayBoundArray (see page 3798)	Gives bound information for all the dimensions of a safe array.
TVarArrayCoorArray (see page 3799)	Gives the position of an element in a safe array.
TVarOp (see page 3799)	TVarOp defines a Variant (see page 3421) operation.
TVarType (see page 3801)	This is type System.TVarType.
TWideStringDynArray (see page 3801)	This is type System::TWideStringDynArray.
TWordDynArray (see page 3801)	This is type System::TWordDynArray.
UCS2Char (see page 3802)	This is type System.UCS2Char.
UCS4Char (see page 3802)	UCS4Char is used for working with 4–byte Unicode characters.
UCS4String (see page 3802)	UCS4String is used for working with 4–byte Unicode characters.
UInt16 (see page 3803)	Alias for `Word`.
UInt32 (see page 3803)	Alias for `Cardinal`.
UInt8 (see page 3804)	Alias for `Byte`.
UnitEntryTable (see page 3804)	UnitEntryTable is an array type with entries of type PackageUnitEntry (see page 3713).
UTF8String (see page 3804)	UTF8String represents UTF-8 encoded Unicode strings.

35.5.1 System.CppLongInt Type

Pascal

```
CppLongInt = type LongInt;
```

C++

```
type LongInt CppLongInt;
```

File

System

Description

This is type System.CppLongInt.

35.5.2 System.CppULongInt Type

Pascal

```
CppULongInt = type LongWord;
```

C++

```
type LongWord CppULongInt;
```

File

System

Description

This is type System.CppULongInt.

35.5.3 System.GetPackageInfoTable Type

Pascal

```
GetPackageInfoTable = function : PackageInfo;
```

C++

```
: PackageInfo ( GetPackageInfoTable)();
```

File

System

Description

This is type System.GetPackageInfoTable.

35.5.4 System.HGLOBAL Type

Pascal

```
HGLOBAL = THandle;
```

C++

```
THandle HGLOBAL;
```

File

System

Description

This is type System.HGLOBAL.

35.5.5 System.HINST Type

Pascal

```
HINST = THandle;
```

C++

```
THandle HINST;
```

File

System

Description

This is type System.HINST.

35.5.6 System.HMODULE Type

Pascal

```
HMODULE = HINST;
```

C++

```
HINST HMODULE;
```

File

System

Description

This is type System.HMODULE.

35.5.7 System.HRESULT Type

Pascal

```
HRESULT = type Longint;
```

C++

```
type Longint HRESULT;
```

File

System

Description

This is type System.HRESULT.

35.5.8 System.HRSRC Type

Pascal

```
HRSRC = THandle;
```

C++

```
THandle HRSRC;
```

File

System

Description

This is type System.HRSRC.

35.5.9 System.Int16 Type

Alias for `SmallInt`.

Pascal

```
Int16 = SmallInt;
```

C++

```
SmallInt Int16;
```

File

System

Description

Int16 is an alias for a `SmallInt` variable.

See Also

Simple Types

35.5.10 System.Int32 Type

Alias for `Integer`.

Pascal

```
Int32 = Integer;
```

C++

```
int Int32;
```

File

System

Description

Int32 is an alias for a `Integer` variable.

See Also

Simple Types

35.5.11 System.Int8 Type

Alias for `ShortInt`.

Pascal

```
Int8 = ShortInt;
```

C++

```
ShortInt Int8;
```

File

System

Description

Int8 is an alias for a `ShortInt` variable.

See Also

Simple Types

35.5.12 System.IntegerArray Type

Pascal

```
IntegerArray = array[0..$efffffff] of Integer;
```

C++

```
array[0..$efffffff] of Integer IntegerArray;
```

File

System

Description

This is type System.IntegerArray.

35.5.13 System.IUnknown Type

IUnknown is a special base interface.

Pascal

```
IUnknown = IInterface;
```

C++

```
IInterface IUnknown;
```

File

System

Description

IUnknown is a base interface with features that are important to C++ programmers and also programmers who utilize COM objects.

Note: In Delphi code, IUnknown is simply another name for IInterface (see page 3502). Descending from IUnknown instead of IInterface (see page 3502) informs the Delphi compiler that the interface must be compatible with COM objects. Cross-platform applications can use interfaces descended from IUnknown, but must not use them to access COM objects, which are only available under Windows.

See Also

IInterface (see page 3502)

System::DelphiInterface (see page 3256)

35.5.14 System.PackageInfo Type

PackageInfo is a pointer to a PackageInfoTable (see page 3713) variable.

Pascal

```
PackageInfo = ^PackageInfoTable;
```

C++

```
^PackageInfoTable PackageInfo;
```

File

System

Description

PackageInfo is a pointer to a PackageInfoTable (see page 3713) variable.

See Also

PackageInfoTable (see page 3713)

35.5.15 System.PAnsiString Type

PAnsiString is a pointer to an `AnsiString` variable.

Pascal

```
PAnsiString = ^AnsiString;
```

C++

```
^AnsiString PAnsiString;
```

File

System

Description

PAnsiString is a pointer to an `AnsiString` variable.

35.5.16 System.PBoolean Type

PBoolean is a pointer to a `Boolean` variable.

Pascal

```
PBoolean = ^Boolean;
```

C++

```
^Boolean PBoolean;
```

File

System

Description

PBoolean is a pointer to a `Boolean` variable.

35.5.17 System.PByte Type

PByte is a pointer to a `Byte` variable.

Pascal

```
PByte = ^Byte;
```

C++

```
^Byte PByte;
```

File

System

Description

PByte is a pointer to a `Byte` variable.

35.5.18 System::PCallDesc Type

Pascal

```
PCallDesc = ^TCallDesc;
```

C++

```
^TCallDesc PCallDesc;
```

File

System.pas

Description

This is type System::PCallDesc.

35.5.19 System.PCardinal Type

PCardinal is a pointer to a `Cardinal` variable.

Pascal

```
PCardinal = ^Cardinal;
```

C++

```
^Cardinal PCardinal;
```

File

System

Description

PCardinal is a pointer to a `Cardinal` variable.

35.5.20 System.PComp Type

PComp is a pointer to a `Comp` variable.

Pascal

```
PComp = ^Comp;
```

C++

```
^Comp PComp;
```

File

System

Description

PComp is a pointer to a `Comp` variable.

Note: The `Comp` data type is deprecated.

35.5.21 System.PCurrency Type

PCurrency is a pointer to a `Currency` (see page 3235) variable.

Pascal

```
PCurrency = ^Currency;
```

C++

```
^Currency PCurrency;
```

File

System

Description

PCurrency is a pointer to a `Currency (see page 3235)` variable.

35.5.22 System.PCVModInfo Type

Pascal

```
PCVModInfo = ^TCVModInfo;
```

C++

```
^TCVModInfo PCVModInfo;
```

File

System

Description

This is type System.PCVModInfo.

35.5.23 System.PDate Type

PDate is a pointer to a Date variable.

Pascal

```
PDate = ^Double;
```

C++

```
^Double PDate;
```

File

System

Description

PDate is a pointer to a Date variable.

See Also

Date (see page 4087)

35.5.24 System.PDateTime Type

PDateTime is a pointer to a TDateTime (see page 3323) variable.

Pascal

```
PDateTime = ^TDateTime;
```

C++

```
^TDateTime PDateTime;
```

File

System

Description

PDateTime is a pointer to a TDateTime (see page 3323) variable.

See Also

TDateTime (see page 3787)

35.5.25 System.PDispatch Type

Pascal

```
PDispatch = ^IDispatch;
```

C++

```
^IDispatch PDispatch;
```

File

System

Description

This is type System.PDispatch.

35.5.26 System::PDispDesc Type

Pascal

```
PDispDesc = ^TDispDesc;
```

C++

```
^TDispDesc PDispDesc;
```

File

System.pas

Description

This is type System::PDispDesc.

35.5.27 System.PDouble Type

PDouble is a pointer to a `Double` variable.

Pascal

```
PDouble = ^Double;
```

C++

```
^Double PDouble;
```

File

System

Description

PDouble is a pointer to a `Double` variable.

35.5.28 System.PDynArrayTypeInfo Type

Pascal

```
PDynArrayTypeInfo = ^TDynArrayTypeInfo;
```

C++

```
^TDynArrayTypeInfo PDynArrayTypeInfo;
```

File

System

Description

This is type System.PDynArrayTypeInfo.

35.5.29 System.PError Type

Pascal

```
PError = ^LongWord;
```

C++

```
^LongWord PError;
```

File

System

Description

This is type System.PError.

35.5.30 System.PExtended Type

PExtended is a pointer to an Extended variable.

Pascal

```
PExtended = ^Extended;
```

C++

```
^Extended PExtended;
```

File

System

Description

PExtended is a pointer to an Extended variable.

35.5.31 System.PGUID Type

PGUID is a pointer to a TGUID (see page 3718) variable.

Pascal

```
PGUID = ^TGUID;
```

C++

```
^TGUID PGUID;
```

File

System

Description

PGUID is a pointer to a TGUID (see page 3718) variable.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
```

```
          myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                "String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
  if IsEqualGUID(MyGuid0, MyGuid1) then
     Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                     '2 randomly generated GUIDs cannot be equal!');
end;
```

See Also

TGUID (see page 3718)

35.5.32 System.PInitContext Type

Pascal

```
PInitContext = ^TInitContext;
```

C++

```
^TInitContext PInitContext;
```

File

System

Description

This is type System.PInitContext.

35.5.33 System.PInt64 Type

PInt64 is a pointer to an `Int64` variable.

Pascal

```
PInt64 = ^Int64;
```

C++

```
^Int64 PInt64;
```

File

System

Description

PInt64 is a pointer to an `Int64` variable.

35.5.34 System.PInteger Type

PInteger is a pointer to an `Integer` variable.

Pascal

```
PInteger = ^Integer;
```

C++

```
^Integer PInteger;
```

File

System

Description

PInteger is a pointer to an `Integer` variable.

35.5.35 System.PIntegerArray Type

PIntegerArray is a pointer to an `IntegerArray (see page 3756)` variable.

Pascal

```
PIntegerArray = ^IntegerArray;
```

C++

```
^IntegerArray PIntegerArray;
```

File

System

Description

PIntegerArray is a pointer to an `IntegerArray (see page 3756)` variable.

See Also

IntegerArray (see page 3756)

35.5.36 System.PInterfaceEntry Type

PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3722) variable.

Pascal

```
PInterfaceEntry = ^TInterfaceEntry;
```

C++

```
^TInterfaceEntry PInterfaceEntry;
```

File

System

Description

PInterfaceEntry is a pointer to a TInterfaceEntry (see page 3722) variable.

See Also

TInterfaceEntry (see page 3722)

35.5.37 System.PInterfaceTable Type

PInterfaceTable is a pointer to a TInterfaceTable (see page 3723) variable.

Pascal

```
PInterfaceTable = ^TInterfaceTable;
```

C++

```
^TInterfaceTable PInterfaceTable;
```

File

System

Description

PInterfaceTable is a pointer to a TInterfaceTable (see page 3723) variable.

See Also

TInterfaceTable (see page 3723)

35.5.38 System.PLibModule Type

PLibModule is a pointer to a System::TLibModule (see page 3723) variable.

Pascal

```
PLibModule = ^TLibModule;
```

C++

```
^TLibModule PLibModule;
```

File

System

Description

PLibModule is a pointer to a System::TLibModule (see page 3723) variable.

See Also

System::TLibModule (see page 3723)

35.5.39 System.PLongint Type

PLongint is a pointer to a `LongInt` variable.

Pascal

```
PLongint = ^Longint;
```

C++

```
^Longint PLongint;
```

File

System

Description

PLongint is a pointer to a PLongint variable.

See Also

PLongint (see page 4665)

35.5.40 System.PLongWord Type

PLongWord is a pointer to a `LongWord` variable.

Pascal

```
PLongWord = ^LongWord;
```

C++

```
^LongWord PLongWord;
```

File

System

Description

PLongWord is a pointer to a `LongWord` variable.

35.5.41 System::PMemoryManager Type

System::PMemoryManager is a pointer to a System::TMemoryManager (see page 3724) variable.

Pascal

```
PMemoryManager = ^TMemoryManager;
```

C++

```
^TMemoryManager PMemoryManager;
```

File

System.pas

Description

System::PMemoryManager is a pointer to a System::TMemoryManager (see page 3724) variable.

See Also

System::TMemoryManager (see page 3724)

35.5.42 System::PMemoryManagerEx Type

System::PMemoryManagerEx is a pointer to a System::TMemoryManagerEx (see page 3725) variable.

Pascal

```
PMemoryManagerEx = ^TMemoryManagerEx;
```

C++

```
^TMemoryManagerEx PMemoryManagerEx;
```

File

System.pas

Description

System::PMemoryManagerEx is a pointer to a System::TMemoryManagerEx (see page 3725) variable.

See Also

System::TMemoryManagerEx (see page 3725)

35.5.43 System::PModuleUnloadRec Type

Pascal

```
PModuleUnloadRec = ^TModuleUnloadRec;
```

C++

```
^TModuleUnloadRec PModuleUnloadRec;
```

File

System.pas

Description

This is type System::PModuleUnloadRec.

35.5.44 System.PMonitor Type

Pascal

```
PMonitor = ^TMonitor;
```

C++

```
^TMonitor PMonitor;
```

File

System

Description

This is type System.PMonitor.

35.5.45 System.PMonitorSupport Type

Pascal

```
PMonitorSupport = ^TMonitorSupport;
```

C++

```
^TMonitorSupport PMonitorSupport;
```

File

System

Description

This is type System.PMonitorSupport.

35.5.46 System.PointerArray Type

Pascal

```
PointerArray = array [0..512*1024*1024 - 2] of Pointer;
```

C++

```
array [0..512*1024*1024 - 2] of Pointer PointerArray;
```

File

System

Description

This is type System.PointerArray.

35.5.47 System.POleVariant Type

POleVariant is a pointer to a System::OleVariant (see page 3274) variable.

Pascal

```
POleVariant = ^OleVariant;
```

C++

```
^OleVariant POleVariant;
```

File

System

Description

POleVariant is a pointer to a System::OleVariant (see page 3274) variable.

See Also

System::OleVariant (see page 3274)

POleVariant1 (see page 807)

35.5.48 System.PPAnsiChar Type

Pascal

```
PPAnsiChar = ^PAnsiChar;
```

C++

```
^PAnsiChar PPAnsiChar;
```

File

System

Description

This is type System.PPAnsiChar.

35.5.49 System.PPChar Type

Pascal

```
PPChar = PPWideChar;
```

C++

```
PPWideChar PPChar;
```

File

System

Description

This is type System.PPChar.

35.5.50 System.PPCharArray Type

Pascal

```
PPCharArray = ^TPCharArray;
```

C++

```
^TPCharArray PPCharArray;
```

File

System

Description

This is type System.PPCharArray.

35.5.51 System.PPDispatch Type

Pascal

```
PPDispatch = ^PDispatch;
```

C++

```
^PDispatch PPDispatch;
```

File

System

Description

This is type System.PPDispatch.

35.5.52 System.PPMonitor Type

Pascal

```
PPMonitor = ^PMonitor;
```

C++

```
^PMonitor PPMonitor;
```

File

System

Description

This is type System.PPMonitor.

35.5.53 System.PPointer Type

PPointer is a pointer to a `Pointer` variable.

Pascal

```
PPointer = ^Pointer;
```

C++

```
^Pointer PPointer;
```

File

System

Description

PPointer is a pointer to a `Pointer` variable.

35.5.54 System.PPointerArray Type

PPointerArray is a pointer to a PointerArray (see page 3770) variable.

Pascal

```
PPointerArray = ^PointerArray;
```

C++

```
^PointerArray PPointerArray;
```

File

System

Description

PPointerArray is a pointer to a PointerArray (see page 3770) variable.

See Also

PointerArray (see page 3770)

35.5.55 System.PPUnknown Type

Pascal

```
PPUnknown = ^PUnknown;
```

C++

```
^PUnknown PPUnknown;
```

File

System

Description

This is type System.PPUnknown.

35.5.56 System.PPWideChar Type

Pascal

```
PPWideChar = ^PWideChar;
```

C++

```
^PWideChar PPWideChar;
```

File

System

Description

This is type System.PPWideChar.

35.5.57 System.PRawByteString Type

Pascal

```
PRawByteString = ^RawByteString;
```

C++

```
^RawByteString PRawByteString;
```

File

System

Description

This is type System.PRawByteString.

35.5.58 System::PResStringRec Type

Pascal

```
PResStringRec = ^TResStringRec;
```

C++

```
^TResStringRec PResStringRec;
```

File

System.pas

Description

This is type System::PResStringRec.

35.5.59 System.PShortInt Type

PShortInt is a pointer to a `ShortInt` variable.

Pascal

```
PShortInt = ^ShortInt;
```

C++

```
^ShortInt PShortInt;
```

File

System

Description

PShortInt is a pointer to a `ShortInt` variable.

35.5.60 System.PShortString Type

PShortString is a pointer to a `ShortString` variable.

Pascal

```
PShortString = ^ShortString;
```

C++

```
^ShortString PShortString;
```

File

System

Description

PShortString is a pointer to a `ShortString` variable.

35.5.61 System.PSingle Type

PSingle is a pointer to a `Single` variable.

Pascal

```
PSingle = ^Single;
```

C++

```
^Single PSingle;
```

File

System

Description

PSingle is a pointer to a `Single` variable.

35.5.62 System::PSmallBlockTypeState Type

Pascal

```
PSmallBlockTypeState = TSmallBlockTypeState *;
```

C++

```
typedef TSmallBlockTypeState * PSmallBlockTypeState;
```

File

sysclass.h

Description

This is type System::PSmallBlockTypeState.

35.5.63 System.PSmallInt Type

PSmallInt is a pointer to a `SmallInt` variable.

Pascal

```
PSmallInt = ^SmallInt;
```

C++

```
^SmallInt PSmallInt;
```

File

System

Description

PSmallInt is a pointer to a `SmallInt` variable.

35.5.64 System.PString Type

PString is a pointer to a `String` variable.

Pascal

```
PString = PUnicodeString;
```

C++

```
PUnicodeString PString;
```

File

System

Description

PString is a pointer to a `String` variable.

35.5.65 System.PTextBuf Type

Pascal

```
PTextBuf = ^TTextBuf;
```

C++

```
^TTextBuf PTextBuf;
```

File

System

Description

This is type System.PTextBuf.

35.5.66 System.PUCS2Char Type

Pascal

```
PUCS2Char = PWideChar;
```

C++

```
PWideChar PUCS2Char;
```

File

System

Description

This is type System.PUCS2Char.

35.5.67 System.PUCS4Char Type

Pascal

```
PUCS4Char = ^UCS4Char;
```

C++

```
^UCS4Char PUCS4Char;
```

File

System

Description

This is type System.PUCS4Char.

35.5.68 System.PUCS4CharArray Type

Pascal

```
PUCS4CharArray = ^TUCS4CharArray;
```

C++

```
^TUCS4CharArray PUCS4CharArray;
```

File

System

Description

This is type System.PUCS4CharArray.

35.5.69 System.PUInt64 Type

Pascal

```
PUInt64 = ^UInt64;
```

C++

```
^UInt64 PUInt64;
```

File

System

Description

This is type System.PUInt64.

35.5.70 System.PUnicodeString Type

Pascal

```
PUnicodeString = ^UnicodeString;
```

C++

```
^UnicodeString PUnicodeString;
```

File

System

Description

This is type System.PUnicodeString.

35.5.71 System.PUnitEntryTable Type

PUnitEntryTable is a pointer to a UnitEntryTable (see page 3804) variable.

Pascal

```
PUnitEntryTable = ^UnitEntryTable;
```

C++

```
^UnitEntryTable PUnitEntryTable;
```

File

System

Description

PUnitEntryTable is a pointer to a UnitEntryTable (see page 3804) variable.

See Also

UnitEntryTable (see page 3804)

35.5.72 System.PUnknown Type

PUnknown is a pointer to an IUnknown (see page 3756) variable.

Pascal

```
PUnknown = ^IUnknown;
```

C++

```
^IUnknown PUnknown;
```

File

System

Description

PUnknown is a pointer to an IUnknown (see page 3756) variable.

See Also

IUnknown (see page 3756)

35.5.73 System.PUTF8String Type

Pascal

```
PUTF8String = ^UTF8String;
```

C++

```
^UTF8String PUTF8String;
```

File

System

Description

This is type System.PUTF8String.

35.5.74 System.PVarArray Type

Pointer type to a TVarArray (see page 3739).

Pascal

```
PVarArray = ^TVarArray;
```

C++

```
^TVarArray PVarArray;
```

File

System

Description

This is a pointer to a TVarArray (see page 3739) data type, giving information about a safe array.

See Also

TVarArray (see page 3739)

35.5.75 System.PVarArrayBoundArray Type

Pascal

```
PVarArrayBoundArray = ^TVarArrayBoundArray;
```

C++

```
^TVarArrayBoundArray PVarArrayBoundArray;
```

File

System

Description

This is type System.PVarArrayBoundArray.

35.5.76 System.PVarArrayCoorArray Type

Pointer type to a TVarArrayCoorArray (see page 3799).

Pascal

```
PVarArrayCoorArray = ^TVarArrayCoorArray;
```

C++

```
^TVarArrayCoorArray PVarArrayCoorArray;
```

File

System

Description

This is a pointer to a TVarArrayCoorArray (see page 3799) data type, giving information about the position of an element in a safe array.

See Also

TVarArrayCoorArray (see page 3799)

35.5.77 System.PVarData Type

Pascal

```
PVarData = ^TVarData;
```

C++

```
^TVarData PVarData;
```

File

System

Description

This is type System.PVarData.

35.5.78 System.PVariant Type

PVariant is a pointer to a System::Variant (see page 3421) variable.

Pascal

```
PVariant = ^Variant;
```

C++

```
^Variant PVariant;
```

File

System

Description

PVariant is a pointer to a System::Variant (see page 3421) variable.

See Also

System::Variant (see page 3421)

35.5.79 System::PVariantManager Type

Pascal

```
PVariantManager = ^TVariantManager;
```

C++

```
^TVariantManager PVariantManager;
```

File

System.pas

Description

This is type System::PVariantManager.

35.5.80 System::PVarRec Type

System::PVarRec is a pointer to a System::TVarRec (see page 3747) variable.

Pascal

```
PVarRec = ^TVarRec;
```

C++

```
^TVarRec PVarRec;
```

File

System.pas

Description

System::PVarRec is a pointer to a System::TVarRec (see page 3747) variable.

See Also

System::TVarRec (see page 3747)

35.5.81 System.PWideString Type

PWideString is a pointer to a `WideString (see page 3467)` variable.

Pascal

```
PWideString = ^WideString;
```

C++

```
^WideString PWideString;
```

File

System

Description

PWideString is a pointer to a `WideString (see page 3467)` variable.

35.5.82 System.PWord Type

PWord is a pointer to a `Word` variable.

Pascal

```
PWord = ^Word;
```

C++

```
^Word PWord;
```

File

System

Description

PWord is a pointer to a `Word` variable.

35.5.83 System.PWordBool Type

PWordBool is a pointer to a `WordBool` variable.

Pascal

```
PWordBool = ^WordBool;
```

C++

```
^WordBool PWordBool;
```

File

System

Description

PWordBool is a pointer to a `WordBool` variable.

35.5.84 System.RawByteString Type

RawByteString enables the passing of string data of any code page without doing any codepage conversions.

Pascal

```
RawByteString = type AnsiString($ffff);
```

C++

```
type AnsiString($ffff) RawByteString;
```

File

System

Description

RawByteString enables the passing of string data of any code page without doing any codepage conversions. The purpose of RawByteString is to reduce the need for multiple overloads of procedures that read string data. This means that parameters of routines that process strings without regard for the string's code page should typically be of type RawByteString.

RawByteString should only be used as a parameter type, and only in routines which otherwise would need multiple overloads for AnsiStrings with different codepages. Such routines need to be written with care for the actual codepage of the string at runtime.

In general it is recommended that string processing routines should simply use "string" as the string type. Declaring variables or fields of type RawByteString should rarely, if ever, be done, because this practice can lead to undefined behavior and potential data loss.

35.5.85 System::ResourceString Type

Pascal

```
ResourceString = TResStringRec;
```

C++

```
typedef TResStringRec ResourceString;
```

File

sysclass.h

Description

This is type System::ResourceString.

35.5.86 System.TAssertErrorProc Type

Pascal

```
TAssertErrorProc = procedure (const Message, Filename: string; LineNumber: Integer; ErrorAddr:
Pointer);
```

C++

```
(const Message, Filename: string; LineNumber: Integer; ErrorAddr: Pointer) (
```

```
  TAssertErrorProc)();
```

File

System

Description

This is type System.TAssertErrorProc.

35.5.87 System::TBooleanDynArray Type

Pascal

```
  TBooleanDynArray = DynamicArray<Boolean>;
```

C++

```
  typedef DynamicArray<Boolean> TBooleanDynArray;
```

File

sysdyn.h

Description

This is type System::TBooleanDynArray.

35.5.88 System.TBoundArray Type

Pascal

```
  TBoundArray = array of Integer;
```

C++

```
  array of Integer TBoundArray;
```

File

System

Description

This is type System.TBoundArray.

35.5.89 System::TByteDynArray Type

Pascal

```
  TByteDynArray = DynamicArray<Byte>;
```

C++

```
  typedef DynamicArray<Byte> TByteDynArray;
```

File

sysdyn.h

Description

This is type System::TByteDynArray.

35.5.90 System::TCardinalDynArray Type

Pascal

```
TCardinalDynArray = DynamicArray<Cardinal>;
```

C++

```
typedef DynamicArray<Cardinal> TCardinalDynArray;
```

File

sysdyn.h

Description

This is type System::TCardinalDynArray.

35.5.91 System.TClass Type

TClass defines the metaclass for System::TObject (see page 3354).

Pascal

```
TClass = class of TObject;
```

C++

```
TObject * TClass;
```

File

System

Description

TClass is the metaclass for System::TObject (see page 3354). Its value is the class reference for System::TObject (see page 3354) or for one of its descendants.

Delphi Examples:

```
{
This example shows how to obtain the ancestry of a component
using the ClassType and ClassParent properties. It uses a
button and a list box on a form. When the user clicks the
button, the name of the button's class and the names of its
parent classes are added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ClassRef: TClass;
begin
  ListBox1.Clear;
  ClassRef := Sender.ClassType;
  while ClassRef <> nil do
  begin
    ListBox1.Items.Add(ClassRef.ClassName);
```

```
      ClassRef := ClassRef.ClassParent;
    end;
  end;
  {
  The list box contains the following strings after the user clicks the button:
      TButton
      TButtonControl
      TWinControl or TWinControl
      TControl
      TComponent
      TPersistent
      TObject
  }
```

See Also

System::TObject (see page 3354)

Class References

35.5.92 System.TClassHelperBaseClass Type

Pascal

```
TClassHelperBaseClass = class of TClassHelperBase;
```

C++

```
TClassHelperBase TClassHelperBaseClass;
```

File

System

Description

This is type System.TClassHelperBaseClass.

35.5.93 System.TDate Type

TDate is used to represent a date.

Pascal

```
TDate = type TDateTime;
```

C++

```
type TDateTime TDate;
```

File

System

Description

TDate represents a special type of TDateTime (see page 3323) value that has no decimal part. A TDate value represents the number of days that have elapsed since 12/30/1899.

See Also

TTimeStamp (see page 4334)

35.5.94 System.TDateTime Type

TDateTime represents a date-and-time value in the Delphi language.

Pascal

```
TDateTime = type Double;
```

C++

```
type Double TDateTime;
```

File

System

Description

In Delphi, TDateTime is a type that maps to a Double. In C++, the TDateTime class corresponds to the Delphi TDateTime type.

The integral part of a Delphi TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of the TDateTime value is fraction of a 24 hour day that has elapsed.

Following are some examples of TDateTime values and their corresponding dates and times:

0	12/30/1899 12:00 am
2.75	1/1/1900 6:00 pm
-1.25	12/29/1899 6:00 am
35065	1/1/1996 12:00 am

To find the fractional number of days between two dates, simply subtract the two values, unless one of the TDateTime values is negative. Similarly, to increment a date and time value by a certain fractional number of days, add the fractional number to the date and time value if the TDateTime value is positive.

When working with negative TDateTime values, computations must handle time portion separately. The fractional part reflects the fraction of a 24-hour day without regard to the sign of the TDateTime value. For example, 6:00 am on 12/29/1899 is –1.25, not –1 + 0.25, which would be –0.75. There are no TDateTime values between –1 and 0.

Note: Delphi 1.0 calculated the date from year 1 instead of from 1899. To convert a Delphi 1.0 date to a TDateTime value in later versions of the Delphi language, subtract 693594.0 from the Delphi 1.0 date.

See Also

Date (see page 4087)

Time (see page 4297)

TDate (see page 3786)

TTimeStamp (see page 4334)

TSQLTimeStamp (see page 1737)

DateTimeToFileDate (see page 4088)

DecodeDateTime

CompareDateTime

FormatDateTime (see page 4195)

35.5.95 System.TDLLProc Type

Pascal

```
TDLLProc = procedure (Reason: Integer);
```

C++

```
(Reason: Integer) ( TDLLProc)();
```

File

System

Description

This is type System.TDLLProc.

35.5.96 System.TDLLProcEx Type

Pascal

```
TDLLProcEx = procedure (Reason: Integer; Reserved: Integer);
```

C++

```
(Reason: Integer; Reserved: Integer) ( TDLLProcEx)();
```

File

System

Description

This is type System.TDLLProcEx.

35.5.97 System::TDoubleDynArray Type

Pascal

```
TDoubleDynArray = DynamicArray<Double>;
```

C++

```
typedef DynamicArray<Double> TDoubleDynArray;
```

File

sysdyn.h

Description

This is type System::TDoubleDynArray.

35.5.98 System.TEnumModuleFunc Type

Pascal

```
TEnumModuleFunc = function (HInstance: Integer; Data: Pointer): Boolean;
```

C++

```
(HInstance: Integer; Data: Pointer): Boolean ( TEnumModuleFunc)();
```

File

System

Description

This is type System.TEnumModuleFunc.

35.5.99 System.TEnumModuleFuncLW Type

Pascal

```
TEnumModuleFuncLW = function (HInstance: LongWord; Data: Pointer): Boolean;
```

C++

```
(HInstance: LongWord; Data: Pointer): Boolean ( TEnumModuleFuncLW)();
```

File

System

Description

This is type System.TEnumModuleFuncLW.

35.5.100 System.TFileIOFunc Type

Pascal

```
TFileIOFunc = function (var F: TFileRec): Integer;
```

C++

```
(var F: TFileRec): Integer ( TFileIOFunc)();
```

File

System

Description

This is type System.TFileIOFunc.

35.5.101 System.THandle Type

THandle is a general purpose data reference (handle).

Pascal

```
THandle = LongWord;
```

C++

```
LongWord THandle;
```

File

System

Description

For example, the SaveToClipboardFormat uses a THandle variable to hold a handle to the data of a bit map. SetAsHandle can then be used to store the bit map data, via its handle, to the clipboard.

Handles are used extensively in communication with Windows APIs.

See Also

SaveToClipboardFormat

SetAsHandle

35.5.102 System::TInt64DynArray Type

Pascal

```
TInt64DynArray = DynamicArray<__int64>;
```

C++

```
typedef DynamicArray<__int64> TInt64DynArray;
```

File

sysdyn.h

Description

This is type System::TInt64DynArray.

35.5.103 System::TIntegerDynArray Type

Pascal

```
TIntegerDynArray = DynamicArray<Integer>;
```

C++

```
typedef DynamicArray<Integer> TIntegerDynArray;
```

File

sysdyn.h

Description

This is type System::TIntegerDynArray.

35.5.104 System.TInterfacedClass Type

TInterfacedClass defines the metaclass for System::TInterfacedObject (see page 3342).

Pascal

```
TInterfacedClass = class of TInterfacedObject;
```

C++

```
TInterfacedObject TInterfacedClass;
```

File

System

Description

TInterfacedClass is the metaclass for System::TInterfacedObject (see page 3342). Its value is the class reference for System::TInterfacedObject (see page 3342) or for one of its descendants.

See Also

System::TInterfacedObject (see page 3342)

Class References

35.5.105 System::TLongWordDynArray Type

Pascal

```
TLongWordDynArray = DynamicArray<unsigned>;
```

C++

```
typedef DynamicArray<unsigned> TLongWordDynArray;
```

File

sysdyn.h

Description

This is type System::TLongWordDynArray.

35.5.106 System.TMemoryMap Type

TMemoryMap is a map of an application's address space as an array of TChunkStatus (see page 3714) values.

Pascal

```
TMemoryMap = array[0..65535] of TChunkStatus;
```

C++

```
array[0..65535] of TChunkStatus TMemoryMap;
```

File

System

Description

GetMemoryMap (see page 3592) returns a map of the address space of the current process. For the purposes of the map, the entire 4GB address space is broken down into 64K chunks. There are 65536 such chunks. TMemoryMap obtains a status value for each of them in the TMemoryMap array. Each TMemoryMap array element may have the following value :

Status value	Meaning
csUnallocated	Free
csAllocated	In use by the process
csReserved	Reserved for future use by the process
csSysAllocated	In use by the operating system
csSysReserved	Reserved for future use by the operating system

See Also

GetMemoryMap (see page 3592)

35.5.107 System.TModuleUnloadProc Type

TModuleUnloadProc defines a procedure that can be called during package unload.

Pascal

```
TModuleUnloadProc = procedure (HInstance: Integer);
```

C++

```
(HInstance: Integer) ( TModuleUnloadProc)();
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your TModuleUnloadProc routine. In essence, AddModuleUnloadProc (see page 3514) adds your callback routine to the head of the list of internal routines that Delphi invokes to perform the unload.

Proc is the Handle to the DLL Package instance being unloaded.

See Also

AddModuleUnloadProc (see page 3514)

TModuleUnloadProcLW (see page 3793)

35.5.108 System.TModuleUnloadProcLW Type

TModuleUnloadProcLW dehines a procedure that can be called during package unload.

Pascal

```
TModuleUnloadProcLW = procedure (HInstance: LongWord);
```

C++

```
(HInstance: LongWord) ( TModuleUnloadProcLW)();
```

File

System

Description

Before Delphi unloads a package that was used by your application, it can optionally call your TModuleUnloadProcLW routine. In essence, AddModuleUnloadProc (see page 3514) adds your callback routine to the head of the list of internal routines that Delphi invokes to perform the unload.

Proc is the Handle to the DLL Package instance being unloaded.

The LW part of the name refers to the Long Word type of package handle. This is in line with Windows handles, and the THandle (see page 3790) type.

See Also

AddModuleUnloadProc (see page 3514)

TModuleUnloadProc (see page 3792)

35.5.109 System.TPCharArray Type

Pascal

```
TPCharArray = array[0..(MaxLongint div SizeOf(PChar))-1] of PChar;
```

C++

```
array[0..(MaxLongint div SizeOf(PChar))-1] of PChar TPCharArray;
```

File

System

Description

This is type System.TPCharArray.

35.5.110 System.TResourceHandle Type

Pascal

```
TResourceHandle = HRSRC;
```

C++

```
HRSRC TResourceHandle;
```

File

System

Description

This is type System.TResourceHandle.

35.5.111 System.TSafeCallErrorProc Type

Pascal

```
TSafeCallErrorProc = procedure (ErrorCode: HResult; ErrorAddr: Pointer);
```

C++

```
(ErrorCode: HResult; ErrorAddr: Pointer) ( TSafeCallErrorProc)();
```

File

System

Description

This is type System.TSafeCallErrorProc.

35.5.112 System::TShortIntDynArray Type

Pascal

```
TShortIntDynArray = DynamicArray<Shortint>;
```

C++

```
typedef DynamicArray<Shortint> TShortIntDynArray;
```

File

sysdyn.h

Description

This is type System::TShortIntDynArray.

35.5.113 System::TSingleDynArray Type

Pascal

```
TSingleDynArray = DynamicArray<Single>;
```

C++

```
typedef DynamicArray<Single> TSingleDynArray;
```

File

sysdyn.h

Description

This is type System::TSingleDynArray.

35.5.114 System::TSmallBlockTypeStates Type

System::TSmallBlockTypeStates is an array of small memory block information records.

Pascal

```
TSmallBlockTypeStates = array[0..NumSmallBlockTypes - 1] of TSmallBlockTypeState;
```

C++

```
array[0..NumSmallBlockTypes - 1] of TSmallBlockTypeState
TSmallBlockTypeStates[NumSmallBlockTypes];
```

File

System.pas

Description

GetMemoryManagerState (see page 3591) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a System::TMemoryManagerState (see page 3730) record. Within that record is held a System::TSmallBlockTypeStates field - an array of System::TSmallBlockTypeState (see page 3736) records. There are NumSmallBlockTypes (see page 3849) (55) different small memory block sizes, and therefore 55 System::TSmallBlockTypeState (see page 3736) records in this array.

See Also

GetMemoryManagerState (see page 3591)

NumSmallBlockTypes (see page 3849)

System::TSmallBlockTypeState (see page 3736)

35.5.115 System::TSmallIntDynArray Type

Pascal

```
TSmallIntDynArray = DynamicArray<Smallint>;
```

C++

```
typedef DynamicArray<Smallint> TSmallIntDynArray;
```

File

sysdyn.h

Description

This is type System::TSmallIntDynArray.

35.5.116 System::TStringDynArray Type

Pascal

```
TStringDynArray = DynamicArray<String>;
```

C++

```
typedef DynamicArray<String> TStringDynArray;
```

File

sysdyn.h

Description

This is type System::TStringDynArray.

35.5.117 System.TSystemThreadEndProc Type

Pascal

```
TSystemThreadEndProc = procedure (ExitCode: Integer);
```

C++

```
(ExitCode: Integer) ( TSystemThreadEndProc)();
```

File

System

Description

This is type System.TSystemThreadEndProc.

35.5.118 System.TSystemThreadFuncProc Type

Pascal

```
TSystemThreadFuncProc = function (ThreadFunc: TThreadFunc; Parameter: Pointer): Pointer;
```

C++

```
(ThreadFunc: TThreadFunc; Parameter: Pointer): Pointer ( TSystemThreadFuncProc)();
```

File

System

Description

This is type System.TSystemThreadFuncProc.

35.5.119 System.TTextBuf Type

Pascal

```
TTextBuf = array[0..127] of AnsiChar;
```

C++

```
array[0..127] of AnsiChar TTextBuf;
```

File

System

Description

This is type System.TTextBuf.

35.5.120 System.TTextIOFunc Type

Pascal

```
TTextIOFunc = function (var F: TTextRec): Integer;
```

C++

```
(var F: TTextRec): Integer ( TTextIOFunc)();
```

File

System

Description

This is type System.TTextIOFunc.

35.5.121 System::TThreadFunc Type

System::TThreadFunc defines a new thread of execution.

Pascal

```
TThreadFunc = function (Parameter: Pointer): Integer;
```

C++

```
(Parameter: Pointer): Integer (* TThreadFunc)(Pointer Parameter);
```

File

System.pas

Description

Use BeginThread (see page 3527) or a TThread object to spawn separate threads of execution. BeginThread (see page 3527) spawns a new thread of execution via the System::TThreadFunc defined function.

Parameter is a pointer to any data you want to pass to the new thread.

When the thread completes, it returns a status code.

See Also

BeginThread (see page 3527)

TThread

35.5.122 System.TTime Type

TTime is used to represent a time.

Pascal

```
TTime = type TDateTime;
```

C++

```
type TDateTime TTime;
```

File

System

Description

TTime is simply a TDateTime (see page 3323) type.

See Also

TDateTime (see page 3787)

35.5.123 System.TUCS4CharArray Type

Pascal

```
TUCS4CharArray = array [0..$ffffff] of UCS4Char;
```

C++

```
array [0..$ffffff] of UCS4Char TUCS4CharArray;
```

File

System

Description

This is type System.TUCS4CharArray.

35.5.124 System.TVarArrayBoundArray Type

Gives bound information for all the dimensions of a safe array.

Pascal

```
TVarArrayBoundArray = array [0..0] of TVarArrayBound;
```

C++

```
array [0..0] of TVarArrayBound TVarArrayBoundArray;
```

File

System

Description

Declare a variable of TVarArrayBoundArray type to store information about the array bounds, corresponding to all the dimensions of a safe array.

Each element in a TVarArrayBoundArray array is of TVarArrayBound (see page 3740) type. The following list describes each field in the TVarArrayBound (see page 3740) type.

Field	Meaning
ElementCount	The number of elements, in the corresponding dimension.
LowBound	The low bound of the index range, in the corresponding dimension.

See Also

TVarArrayBound (see page 3740)

35.5.125 System.TVarArrayCoorArray Type

Gives the position of an element in a safe array.

Pascal

```
TVarArrayCoorArray = array [0..0] of Integer;
```

C++

```
array [0..0] of Integer TVarArrayCoorArray;
```

File

System

Description

Declare a variable of TVarArrayCoorArray type to store information about an element's position in a safe array. Such a variable can then be used in function calls to SafeArrayGetElement or to SafeArrayPutElement.

Each element in an array of TVarArrayCoorArray type stores the coordinate of the element in the safe array, in the increasing order of dimensions.

See Also

SafeArrayGetElement (see page 5121)

SafeArrayPutElement (see page 5125)

35.5.126 System.TVarOp Type

TVarOp defines a Variant (see page 3421) operation.

Pascal

```
TVarOp = Integer;
```

C++

```
int TVarOp;
```

File

System

Description

TVarOp is used to define an Variant (see page 3421) operation. There are three operation categories :

Type	Meaning
Two operand processing	A logical operation of left and right operands. Such as BinaryOp.
Two operand comparison	A comparison of left and right operands. Such as CompareOp.
One operand processing	Processing of a single operand. Such as UnaryOp.

TVarOp may hold one of the following values :

Value	Meaning
opAdd (see page 3850)	Addition.
opAnd (see page 3850)	Bitwise and.
opCmpEQ (see page 3851)	Compare for equality.
opCmpGE (see page 3851)	Compare left greater than or equal to right.
opCmpGT (see page 3852)	Compare left greater than right.
opCmpLE (see page 3852)	Compare left less than or equal to right.
opCmpLT (see page 3853)	Compare left less than right.
opCmpNE (see page 3853)	Compare for inequality.
opCompare (see page 3854)	Generalised comparison.
opDivide (see page 3854)	Division of left by right.
opIntDivide (see page 3855)	Integer division of left by right.
opModulus (see page 3855)	Remainder of left divide by right.
opMultiply (see page 3855)	Left multiplied by right.
opNegate (see page 3856)	Negation of a single operand value.
opNot (see page 3856)	Bitwise negation of a single operand.
opOr (see page 3857)	Bitwise or.
opShiftLeft (see page 3857)	Bitwise shift left.
opShiftRight (see page 3858)	Bitwise shift right.
opSubtract (see page 3858)	Right subtracted from left.
opXor (see page 3859)	Bitwise exclusive or.

See Also

BinaryOp (see page 4994)

CompareOp (see page 4998)

UnaryOp (see page 5006)

35.5.127 System.TVarType Type

Pascal

```
TVarType = Word;
```

C++

```
Word TVarType;
```

File

System

Description

This is type System.TVarType.

35.5.128 System::TWideStringDynArray Type

Pascal

```
TWideStringDynArray = DynamicArray<WideString>;
```

C++

```
typedef DynamicArray<WideString> TWideStringDynArray;
```

File

sysdyn.h

Description

This is type System::TWideStringDynArray.

35.5.129 System::TWordDynArray Type

Pascal

```
TWordDynArray = DynamicArray<Word>;
```

C++

```
typedef DynamicArray<Word> TWordDynArray;
```

File

sysdyn.h

Description

This is type System::TWordDynArray.

35.5.130 System.UCS2Char Type

Pascal

```
UCS2Char = WideChar;
```

C++

```
WideChar UCS2Char;
```

File

System

Description

This is type System.UCS2Char.

35.5.131 System.UCS4Char Type

UCS4Char is used for working with 4–byte Unicode characters.

Pascal

```
UCS4Char = type LongWord;
```

C++

```
type LongWord UCS4Char;
```

File

System

Description

UCS4Char represents a 4–byte Unicode character. UCS4String (see page 3802) is a dynamic array of UCS4Char.

See Also

UCS4String (see page 3802)

UCS4StringToWideString (see page 3687)

35.5.132 System.UCS4String Type

UCS4String is used for working with 4–byte Unicode characters.

Pascal

```
UCS4String = array of UCS4Char;
```

C++

```
array of UCS4Char UCS4String;
```

File

System

Description

UCS4Char (see page 3802) represents a 4–byte Unicode character. UCS4String, which is a dynamic array of UCS4Char (see page 3802), represents a string encoded using UCS4 (4–byte Unicode).

See Also

UCS4Char (see page 3802)

UCS4StringToWideString (see page 3687)

35.5.133 System.UInt16 Type

Alias for `Word`.

Pascal

```
UInt16 = Word;
```

C++

```
Word UInt16;
```

File

System

Description

UInt16 is an alias for a `Word` variable.

See Also

Simple Types

35.5.134 System.UInt32 Type

Alias for `Cardinal`.

Pascal

```
UInt32 = Cardinal;
```

C++

```
unsigned UInt32;
```

File

System

Description

UInt32 is an alias for a `Cardinal` variable.

See Also

Simple Types

35.5.135 System.UInt8 Type

Alias for `Byte`.

Pascal

```
UInt8 = Byte;
```

C++

```
Byte UInt8;
```

File

System

Description

UInt8 is an alias for a `Byte` variable.

See Also

Simple Types

35.5.136 System.UnitEntryTable Type

UnitEntryTable is an array type with entries of type PackageUnitEntry (see page 3713).

Pascal

```
UnitEntryTable = array [0..9999999] of PackageUnitEntry;
```

C++

```
array [0..9999999] of PackageUnitEntry UnitEntryTable;
```

File

System

Description

UnitEntryTable is an array type with entries of type PackageUnitEntry (see page 3713).

A variable of type UnitEntryTable contains a compiler generated table to be processed sequentially, in order to initialize and finalize all package units.

See Also

PackageUnitEntry (see page 3713)

35.5.137 System.UTF8String Type

UTF8String represents UTF-8 encoded Unicode strings.

Pascal

```
UTF8String = type AnsiString(65001);
```

C++

```
type AnsiString(65001) UTF8String;
```

File

System

Description

UTF8String represents a string encoded using UTF-8 (variable number of bytes Unicode). It is a System::AnsiStringBase (see page 3174) type with a UTF-8 code page.

In Delphi, UTF8String is a true compiler type. The compiler does conversions between System::UnicodeString (see page 3381) and UTF8String as necessary.

See Also

System::AnsiStringBase (see page 3174)

System::UnicodeString (see page 3381)

35.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
AbstractErrorProc (see page 3807)	AbstractErrorProc defines a callback routine for abstract method call errors.
AllocMemCount (see page 3807)	Represents the total number of allocated memory blocks in a Delphi application.
AllocMemSize (see page 3809)	Represents the total size of allocated memory blocks.
AssertErrorProc (see page 3810)	Points to the assertion error-handler.
CmdLine (see page 3812)	Points to the command-line arguments specified when the application is invoked.
CmdShow (see page 3812)	Specifies the flags for displaying the application's main window.
CPUCount (see page 3813)	This is variable System.CPUCount.
DebugHook (see page 3813)	DebugHook is set if an application is running under the IDE debugger.
Default8087CW (see page 3813)	Specifies the Default 8087 control word.
DefaultSystemCodePage (see page 3815)	This is variable System.DefaultSystemCodePage.
DefaultTextLineBreakStyle (see page 3815)	Specifies the characters that are used by default to separate lines in text.
DefaultUnicodeCodePage (see page 3816)	This is variable System.DefaultUnicodeCodePage.
DispCallByIDProc (see page 3816)	This is variable System.DispCallByIDProc.
DLLProc (see page 3816)	Points to a procedure invoked by a DLL entry point.
envp (see page 3817)	This is variable System.envp.
ErrorAddr (see page 3817)	Contains the address of a statement causing a runtime error.
ErrorProc (see page 3818)	Points to the RTL runtime error handler.
ErrOutput (see page 3820)	ErrOutput specifies a write-only Text file associated with the process's error output file.
ExceptClsProc (see page 3820)	ExceptClsProc defines a pointer to a function called during exception handling.
ExceptionAcquired (see page 3821)	This is variable System.ExceptionAcquired.
ExceptionClass (see page 3821)	This is variable System.ExceptionClass.
ExceptObjProc (see page 3821)	ExceptObjProc defines a pointer to a function called during exception handling.
ExceptProc (see page 3822)	Points to the lowest-level exception handler.
ExitCode (see page 3823)	Contains the application's exit code.
ExitProc (see page 3823)	Points to a program's exit procedure (provided for backward compatibility).
ExitProcessProc (see page 3824)	Specifies the last procedure to execute before the application shuts down.
FileMode (see page 3824)	Indicates the access mode on typed and untyped files opened by the Reset (see page 3643) procedure.
HeapAllocFlags (see page 3825)	Indicates how the memory manager obtains memory from the operating system.
HPrevInst (see page 3826)	This is variable System.HPrevInst.
InitProc (see page 3826)	InitProc is the last installed initialization procedure.
Input (see page 3827)	Specifies a read-only Text file associated with the processes standard Input file.
IsConsole (see page 3828)	Indicates whether the module was compiled as a console application.
IsLibrary (see page 3828)	Indicates whether the module is a shared library.
IsMultiThread (see page 3829)	Indicates whether the application spawned additional threads using BeginThread (see page 3527) or TThread objects.
JITEnable (see page 3829)	Controls when the just-in-time debugger is called.
LibModuleList (see page 3830)	Lists module records for enumeration procedures.
MainInstance (see page 3830)	Indicates the instance handle for the main executable.
MainThreadID (see page 3831)	Indicates the instance handle for the current module's main thread of execution.
ModuleUnloadList (see page 3831)	This is variable System.ModuleUnloadList.
MonitorSupport (see page 3831)	This is variable System.MonitorSupport.
NeverSleepOnMMThreadContention (see page 3832)	Dictates memory manager behavior during a thread contention.
NoErrMsg (see page 3832)	Controls whether the application displays an error message when a runtime error occurs.

Output (see page 3833)	Specifies a write-only Text file associated with the process's standard output file.
RaiseExceptionProc (see page 3833)	Points to RaiseException Windows function.
RaiseExceptObjProc (see page 3834)	This is variable System.RaiseExceptObjProc.
RandSeed (see page 3834)	RandSeed stores the built-in random number generator's seed.
ReportMemoryLeaksOnShutdown (see page 3835)	Report memory leaks on shutdown.
RTLUnwindProc (see page 3835)	This is variable System.RTLUnwindProc.
SafeCallErrorProc (see page 3836)	SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.
SystemThreadEndProc (see page 3836)	This is variable System.SystemThreadEndProc.
SystemThreadFuncProc (see page 3837)	This is variable System.SystemThreadFuncProc.
Test8086 (see page 3837)	This is variable System.Test8086.
Test8087 (see page 3837)	This is variable System.Test8087.
TestFDIV (see page 3838)	This is variable System.TestFDIV.
UTF8CompareLocale (see page 3838)	This is variable System.UTF8CompareLocale.
VarAddRefProc (see page 3838)	The VarAddRefProc variable is used internally.
VarClearProc (see page 3839)	The VarClearProc variable is used internally.
VarCopyProc (see page 3839)	The VarCopyProc variable is used internally.
VarToLStrProc (see page 3840)	The VarToLStrProc variable is used internally.
VarToWStrProc (see page 3840)	The VarToWStrProc variable is used internally.

35.6.1 System.AbstractErrorProc Variable

AbstractErrorProc defines a callback routine for abstract method call errors.

Pascal

```
AbstractErrorProc: procedure;
```

C++

```
procedure AbstractErrorProc;
```

File

System

Description

An abstract method is a virtual or dynamic method that has no implementation in the class where it is declared. Its implementation is deferred to a descendant class. An EAbstractError exception will be thrown if a call is attempted to an abstact method.

You can define an AbstractErrorProc routine that will be called prior to the exception throw.

See Also

EAbstractError (see page 3916)

35.6.2 System.AllocMemCount Variable

Represents the total number of allocated memory blocks in a Delphi application.

Pascal

```
AllocMemCount: Integer deprecated;
```

C++

```
Integer deprecated AllocMemCount;
```

File

System

Description

Note: On Windows, AllocMemCount is the number of currently allocated blocks of memory that were requested by the user. AllocMemCount is incremented each time a block of memory is allocated and is decremented each time a block of memory is freed. Use AllocMemCount to find out how many unfreed blocks of memory remain.

Note: On Linux, the AllocMemCount variable is not maintained by default. Unless it is updated by a third-party memory manager, this variable always has a value of zero.

Warning: Caution:Because AllocMemCount is a global variable declared in the System (see page 3159) unit, it is not always safe to access it directly. In situations where modules are statically linked to the RTL, each module will have a different instance of AllocMemCount. The following table is a summary. Note that applications that use runtime packages are dynamically linked to the RTL, whereas applications that do not are statically linked.

Application type	Accessibility of AllocMemCount
EXE	Applications that do not use packages or Delphi DLLs can safely access this global variable directly because there is only one instance of it.
EXE with packages and no Delphi DLLs	Applications that use runtime packages and do not use any statically linked libraries can safely access AllocMemCount directly. In this situation all modules are dynamically linked to the RTL and there is only one instance of the variable. Unlike standard DLLs, packages allow access to global variables.
EXE with statically linked DLLs	If an application is statically linked to the RTL and also uses libraries that are statically linked to the RTL, then the AllocMemCount variable should never be accessed directly, since the EXE and each library will have its own instance of it. Instead, use the GetAllocMemCount function, implemented in BorlandMM, which returns the instance of the AllocMemCount global variable in the BorlandMM DLL. This DLL manages the memory allocation for all modules that include the ShareMem unit as the first unit in the application or library's uses clause. It is, therefore, this instance of the variable that contains the correct value. A function is used to access AllocMemCount in this situation because global variables in a DLL are private to the DLL.
EXE with packages and statically linked DLLs	It is not advisable to create applications that mix runtime packages with statically linked libraries. In this circumstance, use caution when allocating, freeing, and accessing memory. Be aware that each version of the RTL contains an instance of AllocMemCount that references memory that is allocated and freed only by that particular module.

See Also

AllocMemSize (see page 3809)

AllocMem (see page 3515)

35.6.3 System.AllocMemSize Variable

Represents the total size of allocated memory blocks.

Pascal

```
AllocMemSize: Integer deprecated;
```

C++

```
Integer deprecated AllocMemSize;
```

File

System

Description

Note: On Windows, AllocMemSize is the total size, in bytes, of all currently allocated blocks of memory in use by an application. Use AllocMemSize to find out how many bytes of memory an application is currently using.

Note: On Linux, the AllocMemSize variable is not maintained by default. Unless it is updated by a third-party memory manager, this variable always has a value of zero.

Warning: Caution:Because AllocMemSize is a global variable declared in the System (see page 3159) unit, it is not always safe to access it directly. In situations where modules are statically linked to the RTL, each module will have a different instance of AllocMemSize. The following table is a summary. Note that applications that use runtime packages are dynamically linked to the RTL, whereas applications that do not are statically linked.

Application type	Accessibility of AllocMemSize
EXE	Applications that do not use packages or Delphi DLLs can safely access this global variable directly because there is only one instance of it.
EXE with packages and no Delphi DLLs	Applications that use runtime packages and do not use any statically linked DLLs can safely access AllocMemSize directly. In this situation all modules are dynamically linked to the RTL and there is only one instance of the variable. Unlike standard DLLs, packages allow access to global variables.
EXE with statically linked DLLs	If an application is statically linked to the RTL and if that application is also using DLLs that are statically linked to the RTL then the AllocMemSize variable should never be accessed directly, since the EXE and each DLL will have its own instance of it. Instead use the GetAllocMemSize function, implemented in BorlandMM, which returns the instance of the AllocMemSize global variable in the BorlandMM DLL. This DLL manages the memory allocation for all modules that include the ShareMem unit as the first unit in the application or library's uses clause. It is, therefore, this instance of the variable that contains the correct value. A function is used to access AllocMemSize in this situation because global variables in a DLL are private to the DLL.
EXE with packages and statically linked DLLS	It is not advisable to create applications that mix runtime packages with statically linked DLLs. In this circumstance use caution when allocating, freeing, and accessing memory. Be aware that each version of the RTL contains an instance of AllocMemSize that references memory that is allocated and freed only by that particular module.

See Also

AllocMem (see page 3515)

AllocMemCount (see page 3807)

35.6.4 System.AssertErrorProc Variable

Points to the assertion error-handler.

Pascal

```
AssertErrorProc: TAssertErrorProc;
```

C++

```
TAssertErrorProc AssertErrorProc;
```

File

System

Description

AssertErrorProc points to a routine that produces runtime error 227 in Delphi code. The SysUtils (see page 3901) unit replaces it to cause an EAssertionFailed exception. If you do not use assertions, you can replace this with a method that that does something else, such as call OutputDebugString to add messages to the event log (see the example).

The value of AssertErrorProc must be a procedure with the following signature

procedure AssertErrorHandler(const Message, Filename: string; LineNumber: Integer; ErrorAddr (see page 3817): Pointer);

Delphi Examples:

```
{
The following code is a self-contained unit that you can add to a project
that logs failed assertions to the event log rather than raising an
exception. If /Debug is passed to the application as a command-line
parameter, all failed Assert calls cause a call to OutputDebugString,
which sends a messages to the Event Log window if run from Delphi, or
does nothing if run outside the IDE (OutputDebugString is a no-op if not
run from within a debugger). A nice side benefit is that all the line
numbers and unit names can be stripped from the EXE by turning off
assertions and rebuilding the project. Build the executable in the IDE
by hitting "Run".  Then go to the debug directory for the project and
execute the command line "AssertErrorProc_proj /Debug".
Note: AssertErrorHandler should do as little as possible.  This routine
does no cleanup, so no variables, no compiler generated variables.
Procedure calls are OK.
Note:   This example was originally provided by Brian Long.
}
type
  TForm1 = class(TForm)
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStorage = class(TObject)
    FData: string;
    property Data: string read FData write FData;
```

```
    private
      { Private declarations }
    public
      { Public declarations }
    end;

    var
    Form1: TForm1;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);

implementation

{$R *.dfm}

{$ASSERTIONS ON}

type
  EMyAssert = class(Exception);


procedure DoMessage(Message, Filename: String; LineNumber: Integer; ErrorAddr: Pointer);
var
  S: String;
begin
  S := Format('%s (%s, line %d, address $%x)',
    [Message, Filename, LineNumber, Pred(Integer(ErrorAddr))]);
  OutputDebugString(PChar(S));
end;

procedure AssertErrorHandler(const Message, Filename: string;  LineNumber: Integer; ErrorAddr:
Pointer);
{ No local variables.  Not compiler generated temporary variables. }
{ Using the call stack here will cause Access Violation errors. }
begin
  DoMessage(Message, Filename, LineNumber, ErrorAddr);
  raise EMyAssert.Create('Boom!');
end;

procedure AssertErrorNoHandler(const Message, Filename: string; LineNumber: Integer;
ErrorAddr: Pointer);
begin
end;

procedure ModifyStorage(AStorage: TStorage; const s: string);
begin
  Assert(AStorage <> nil, 'Value');
  AStorage.Data := s;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  ModifyStorage(nil, 'Oops');
end;

initialization
  if FindCmdLineSwitch('Debug', ['/', '-'], True) then
  begin
    MessageDlg('Command line switch found!', mtInformation, [mbOK], 0);
//    OldProc := System.AssertErrorProc;
    System.AssertErrorProc := @AssertErrorHandler
  end
  else
    System.AssertErrorProc := @AssertErrorNoHandler;
```

```
ModifyStorage(nil, 'Ooops');
```

See Also

EAssertionFailed (see page 3918)

Assert (see page 3522)

ErrorProc (see page 3818)

35.6.5 System.CmdLine Variable

Points to the command-line arguments specified when the application is invoked.

Pascal

```
CmdLine: PChar platform;
```

C++

```
PChar platform CmdLine;
```

File

System

Description

Note: On Windows, the CmdLine variable points to a null-terminated string that contains the command-line arguments specified when the application was started. Use ParamStr (see page 3625) to access individual arguments. In a library (DLL), CmdLine is nil (Delphi) or NULL (C++).

Note: On Linux, CmdLine is always nil (Delphi) or NULL (C++).

See Also

ParamStr (see page 3625)

FindCmdLineSwitch (see page 4174)

35.6.6 System.CmdShow Variable

Specifies the flags for displaying the application's main window.

Pascal

```
CmdShow: Integer platform;
```

C++

```
Integer platform CmdShow;
```

File

System

Description

Note: On Windows, CmdShow specifies the parameter that should be passed to the Windows API ShowWindow when displaying the main window of the application. In a library (DLL), CmdShow is always 0.

Note: On Linux, CmdShow is always 0. It exists only for compatibility with Windows code in cross-platform applications.

35.6.7 System.CPUCount Variable

Pascal

```
CPUCount: Integer;
```

C++

```
int CPUCount;
```

File

System

Description

This is variable System.CPUCount.

35.6.8 System.DebugHook Variable

DebugHook is set if an application is running under the IDE debugger.

Pascal

```
DebugHook: Byte platform = 0;
```

C++

```
Byte platform DebugHook = 0;
```

File

System

Description

DebugHook is 0 when an application operates outside the IDE debugger. DebugHook is 1 when an application operates inside the IDE debugger.

35.6.9 System.Default8087CW Variable

Specifies the Default 8087 control word.

Pascal

```
Default8087CW: Word = $1332;
```

C++

```
Word Default8087CW = 0x1332;
```

File

System

Description

Default8087CW is the value to which the Floating Point Unit (FPU) control register is set. The FPU control word includes bits that control the FPU's precision, rounding mode, and whether exceptions generate signals if they occur. See Intel's processor documentation for details.

The Set8087CW (see page 3653) procedure allows the programmer to have direct access to the Default8087CW. Setting the floating-point precision and rounding mechanism can be useful if you are reusing old code that is sensitive to the floating-point precision standard used. However, be aware that using this procedure to change the value of the 8087CW will change the behavior of the program's FP calculations. It is the programmer's responsibility to reset it.

C++ Examples:

```
/*
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
*/
Word Saved8087CW;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Edit3->Text = FloatToStr(StrToFloat(Edit1->Text) / StrToFloat(Edit2->Text));
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "FPU Exceptions")
    System::Set8087CW(Saved8087CW);
  if (RadioGroup1->Items->Strings[RadioGroup1->ItemIndex] == "No FPU Exceptions")
    System::Set8087CW(0x133f); // Disable all fpu exceptions
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RadioGroup1->Items->Add("No FPU Exceptions");
  RadioGroup1->Items->Add("FPU Exceptions");
  RadioGroup1->ItemIndex = 2;
  Saved8087CW = Default8087CW;  // Save this because Set8087CW changes it!
}

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  System::Set8087CW(Saved8087CW); // Default value (with exceptions) is 0x1372
}
```

Delphi Examples:

```
{
This example accesses the Floating Point Unit (FPU) control
register.  Try turning floating point exceptions off and on
and dividing a number by zero to test it.
}
var
  Form1: TForm1;
  Saved8087CW: Word;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit3.Text := FloatToStr(StrToFloat(Edit1.Text) / StrToFloat(Edit2.Text));
end;
```

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'FPU Exceptions' then
    System.Set8087CW(Saved8087CW);
  if RadioGroup1.Items[RadioGroup1.ItemIndex] = 'No FPU Exceptions' then
    System.Set8087CW($133f); { Disable all fpu exceptions }
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioGroup1.Items.Add('No FPU Exceptions');
  RadioGroup1.Items.Add('FPU Exceptions');
  RadioGroup1.ItemIndex := 2;
  Saved8087CW := Default8087CW;  // Save this because Set8087CW changes it!
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  System.Set8087CW(Saved8087CW); // Default value (with exceptions) is $1372
end;
```

See Also

Set8087CW (see page 3653)

35.6.10 System.DefaultSystemCodePage Variable

Pascal

```
DefaultSystemCodePage: Integer;
```

C++

```
int DefaultSystemCodePage;
```

File

System

Description

This is variable System.DefaultSystemCodePage.

35.6.11 System.DefaultTextLineBreakStyle Variable

Specifies the characters that are used by default to separate lines in text.

Pascal

```
DefaultTextLineBreakStyle: TTextLineBreakStyle = tlbsCRLF;
```

C++

```
TTextLineBreakStyle DefaultTextLineBreakStyle = tlbsCRLF;
```

File

System

Description

DefaultTextLineBreakStyle controls how the code editor represents the separation between lines of text in new files. For existing files, it maintains the line break style that the file currently uses.

See Also

SetLineBreakStyle (see page 3659)

35.6.12 System.DefaultUnicodeCodePage Variable

Pascal

```
DefaultUnicodeCodePage: Integer;
```

C++

```
int DefaultUnicodeCodePage;
```

File

System

Description

This is variable System.DefaultUnicodeCodePage.

35.6.13 System.DispCallByIDProc Variable

Pascal

```
DispCallByIDProc: Pointer;
```

C++

```
void * DispCallByIDProc;
```

File

System

Description

This is variable System.DispCallByIDProc.

35.6.14 System.DLLProc Variable

Points to a procedure invoked by a DLL entry point.

Pascal

```
DLLProc: Pointer;
```

C++

```
void * DLLProc;
```

File

System

Description

DLLProc is used to specify a procedure that is invoked every time a DLL's entry point is called. A procedure assigned to DLLProc must take one parameter of type Integer or int. For example,

- procedure LibraryProc(Reason: Integer); { Delphi }
- void LibraryProc(int Reason); // C++
- When the procedure is invoked, this single parameter contains a value between 0 and 3 as defined by the following group of constants (from the Windows unit):
- const
- DLL_PROCESS_DETACH = 0;
- DLL_PROCESS_ATTACH = 1;
- DLL_THREAD_ATTACH = 2;
- DLL_THREAD_DETACH = 3;
- For further details on the meaning of these constants, refer to the description of the DllEntryPoint function in the Win32 API online help.

Note: DLL_PROCESS_ATTACH is passed to the procedure only if the DLL's initialization code calls the procedure and specifies DLL_PROCESS_ATTACH as a parameter.

Note: When the Reason parameter is DLL_PROCESS_ATTACH, setting ExitCode (see page 3823) to a nonzero value causes the entry point to return false.

35.6.15 System.envp Variable

Pascal

```
envp: PPChar platform;
```

C++

```
PPChar platform envp;
```

File

System

Description

This is variable System.envp.

35.6.16 System.ErrorAddr Variable

Contains the address of a statement causing a runtime error.

Pascal

```
ErrorAddr: Pointer = nil;
```

C++

```
void * ErrorAddr = nil;
```

File

System

Description

If a program terminates normally or stops due to a call to Halt (see page 3594), ErrorAddr is nil (Delphi) or NULL (C++). If a program ends because of a runtime error, ErrorAddr contains the address of the statement in error.

See Also

ErrorProc (see page 3818)

ExceptProc (see page 3822)

AcquireExceptionObject (see page 3512)

ExceptAddr (see page 3562)

ExceptionErrorMessage (see page 4107)

ExceptObject (see page 3562)

GetLastError (see page 3582)

35.6.17 System.ErrorProc Variable

Points to the RTL runtime error handler.

Pascal

```
ErrorProc: procedure (ErrorCode: Byte; ErrorAddr: Pointer);
```

C++

```
procedure (ErrorCode: Byte; ErrorAddr: Pointer) ErrorProc;
```

File

System

Description

In Delphi, ErrorProc is a procedure variable pointing to the runtime error handler. The standard ErrorProc procedure reports the runtime error and terminates the program.

The ErrorProc variable is retained primarily for compatibility with older code. Newer applications should not specify an alternate runtime error procedure using ErrorProc as SysUtils (see page 3901) uses this variable to convert runtime errors into exceptions.

If you use SysUtils (see page 3901) in your program, it will force ErrorProc to its own routine and convert the runtime error into an exception. When a runtime library error occurs, the ErrorProc is called. The ErrorCode parameter contains the runtime error number and the ErrorAddr (see page 3817) parameter contains the machine code address that caused the runtime error.

Delphi Examples:

```
{
This example demostrates the use of ErrorProc and how to catch runtime
errors.
}
```

```
procedure MyErrorProc(ErrorCode : Byte; ErrorPtr : Pointer);
var
  ErrorName : String;
begin
  { Get the "name" of the enum value using RTTI }
  ErrorName := GetEnumName(TypeInfo(TRuntimeError), ErrorCode);

  { Show an error box }
  MessageDlg(Format('Error with code %d (%s) risen at $%X.',
      [ErrorCode, ErrorName, Integer(ErrorPtr)]), mtError, [mbOK], 0);
end;

procedure TForm2.btIOErrorClick(Sender: TObject);
begin
  {
  Try to write something onto the console - will raise an
  exception.
  }
  WriteLn('This will generate an error because there is no' +
          ' console attached!');
end;

{$OVERFLOWCHECKS ON}
{$OPTIMIZATION OFF}
{$HINTS OFF}
procedure TForm2.btOverflowErrClick(Sender: TObject);
var
  b : Cardinal;
begin
  {
  Simulate an overflow: Note, enabled the overflow
  checking and disabled optimizations because Delphi's
  compiler will not compile this code otherwise.
  }
  b := $FFFFFFFF;
  b := b * b;
end;
{$HINTS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}

procedure TForm2.btRuntimeErrorClick(Sender: TObject);
begin
  { Simulate an access violation }
  System.Error(reAccessViolation);
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  { Register my System error proc }
  ErrorProc := MyErrorProc;
end;
```

See Also

ErrorAddr (see page 3817)

ExceptAddr (see page 3562)

ExceptionErrorMessage (see page 4107)

ExceptObject (see page 3562)

GetLastError (see page 3582)

ExceptProc (see page 3822)

AcquireExceptionObject (see page 3512)

35.6.18 System.ErrOutput Variable

ErrOutput specifies a write-only Text file associated with the process's error output file.

Pascal

```
ErrOutput: Text;
```

C++

```
Text ErrOutput;
```

File

System

Description

In Delphi, the ErrOutput variable is a write-only Text file associated with the process's error output file.

Note: On Windows, most processes don't have an error output file, and writing to ErrOutput raises an error. Delphi programs have a standard output file if they are linked as console applications.

Note: On Linux, all processes have an error output file. There are various ways to view the error output. One simple method is to run the program from a shell window.

See Also

Input (see page 3827)

Output (see page 3833)

35.6.19 System.ExceptClsProc Variable

ExceptClsProc defines a pointer to a function called during exception handling.

Pascal

```
ExceptClsProc: Pointer;
```

C++

```
void * ExceptClsProc;
```

File

System

Description

By default, ExceptClsProc is defined internally to point to a function that handles exceptions on your behalf. It returns the exception class. If you override ExceptClsProc to point to your function, that function will be called when an exception occurs in a try-except block. It will be passed an updateable TExceptionRecord value that describes the exception.

Internally, the class returned is used to determine which exception handler to use before ExceptObjProc (see page 3821) is called to create the exception instance.

See Also

ExceptObjProc (see page 3821)

TExceptionRecord (see page 4322)

35.6.20 System.ExceptionAcquired Variable

Pascal

```
ExceptionAcquired: Pointer;
```

C++

```
void * ExceptionAcquired;
```

File

System

Description

This is variable System.ExceptionAcquired.

35.6.21 System.ExceptionClass Variable

Pascal

```
ExceptionClass: TClass;
```

C++

```
TClass ExceptionClass;
```

File

System

Description

This is variable System.ExceptionClass.

35.6.22 System.ExceptObjProc Variable

ExceptObjProc defines a pointer to a function called during exception handling.

Pascal

```
ExceptObjProc: Pointer;
```

C++

```
void * ExceptObjProc;
```

File

System

Description

ExceptClsProc (see page 3820) defines the function that returns a class that is used to determine which exception handler to use before an exception object is created.

You can override the default exception object creation by defining your own function that returns a TObject (see page 3354). You point ExceptObjProc to that function. It is passed the an updateable version of the TExceptionRecord.

See Also

ExceptClsProc (see page 3820)

TExceptionRecord (see page 4322)

35.6.23 System.ExceptProc Variable

Points to the lowest-level exception handler.

Pascal

```
ExceptProc: Pointer;
```

C++

```
void * ExceptProc;
```

File

System

Description

In Delphi code, ExceptProc handles unhandled exceptions (exceptions that have not been caught in the except portion of a try..except statement.) If your application has a TApplication object, it automatically catches exceptions and generates an OnException event (to which you can respond using either the application's or a TApplicationEvents object's OnException event handler). Thus, because the TApplication object catches the exceptions, they never reach the ExceptProc handler. However, even if you have a TApplication object, it does not catch exceptions generated during initialization and finalization. These are handled by ExceptProc.

You can hook into ExceptProc to change how unhandled exceptions are reported. To do so, save the contexts of ExceptProc before changing it to the address of your own exception handler. The first statement of your new exception handler must then reinstall the saved value of ExceptProc. In this way, the ExceptProc procedure becomes part of a chain of exception procedures, each of which is executed in reverse order of installation.

The procedure assigned to ExceptProc must have a signature such as the following:

procedure ExceptHandler(ExceptObject (see page 3562): System::TObject (see page 3354); ExceptAddr (see page 3562): Pointer);

Note: :On Linux, ExceptProc may be called when most of the runtime has already shut down. This happens if an application terminates with an unhandled exception.

See Also

ErrorAddr (see page 3817)

ErrorProc (see page 3818)

TApplication

ExceptObject (see page 3562)

35.6.24 System.ExitCode Variable

Contains the application's exit code.

Pascal

```
ExitCode: Integer = 0;
```

C++

```
int ExitCode = 0;
```

File

System

Description

In Delphi code, the ExitCode variable contains the application's exit code. An exit procedure can learn the cause of termination by examining ExitCode. If the program terminates normally, ExitCode is zero. If the program terminates due to a call to Halt (see page 3594), ExitCode contains the value passed to Halt (see page 3594). If the program terminates due to a runtime error, ExitCode contains the error code.

Note: On Windows, setting the ExitCode variable to a value other than 0 in the initialization section of a unit will make loading of a DLL fail (DllMain will return false). You can also set ExitCode in code called from DLLProc (see page 3816) or DllProcEx to signal failure of loading.

Note: On Linux, although the ExitCode variable is accessible from shared object code, its value does not affect the loader of the shared object. Thus, using ExitCode in a shared object has no effect.

See Also

AddExitProc (see page 4048)

AddTerminateProc (see page 4048)

CallTerminateProcs (see page 4071)

ExitProc (see page 3823)

35.6.25 System.ExitProc Variable

Points to a program's exit procedure (provided for backward compatibility).

Pascal

```
ExitProc: Pointer;
```

C++

```
void * ExitProc;
```

File

System

Description

In Delphi code, ExitProc points to an exit procedure in an application. The exit procedure always gets called as part of a program's termination. ExitProc is only used when generating executable files.

An exit procedure takes no parameters and must be compiled with a far procedure directive to force it to use the far call model.

Do not use ExitProc in a dynamically loaded package. This procedure is called when the process halts, not when a package or DLL is unloaded. If you set ExitProc from a package or DLL, it may interfere with the ExitProc of the host executable. Because ExitProc is not compatible with packages, it is recommended that you add code to the finalization section of a unit instead. Inside a DLL, you can use the DLLProc (see page 3816) variable instead.

See Also

AddExitProc (see page 4048)

ExitCode (see page 3823)

AddTerminateProc (see page 4048)

CallTerminateProcs (see page 4071)

DLLProc (see page 3816)

35.6.26 System.ExitProcessProc Variable

Specifies the last procedure to execute before the application shuts down.

Pascal

```
ExitProcessProc: procedure;
```

C++

```
procedure ExitProcessProc;
```

File

System

Description

Assign a value to ExitProcessProc to specify a procedure that you want to execute when the application shuts down. The value of ExitProcessProc is a procedure with no parameters.

See Also

InitProc (see page 3826)

35.6.27 System.FileMode Variable

Indicates the access mode on typed and untyped files opened by the Reset (see page 3643) procedure.

Pascal

```
FileMode: Byte = 2;
```

C++

```
Byte FileMode = 2;
```

File

System

Description

In Delphi code, FileMode determines the access mode to use when typed and untyped files (not text files) are opened using the Reset (see page 3643) procedure.

The default FileMode is 2 (Read (see page 3634)/Write (see page 3707) access). Assigning another value to FileMode causes all subsequent Resets to use that mode. The SysUtils (see page 3901) unit declares a number of file open mode constants that can be assigned to the FileMode variable.

Note: Be sure to reset FileMode before calling Reset (see page 3643) with a read-only file. Even if you are only reading from the file, the default FileMode value (Read (see page 3634)/Write (see page 3707) access) will cause an attempt to open a read-only file to fail.

Warning: FileMode is declared as a variable, not as a threadvar. Therefore this variable is not thread-safe.

See Also

Reset (see page 3643)

Rewrite (see page 3645)

TFileStream

35.6.28 System.HeapAllocFlags Variable

Indicates how the memory manager obtains memory from the operating system.

Pascal

```
HeapAllocFlags: Word platform = 2;
```

C++

```
Word platform HeapAllocFlags = 2;
```

File

System

Description

On Windows, use HeapAllocFlags to customize the flags that the system memory manager uses when requesting blocks of memory from the operating system. By default, the value is GMEM_MOVEABLE.

HeapAllocFlags can be a combination of the following values:

GMEM_FIXED	Allocates fixed memory. The operating system can't move blocks, so there is no need to lock memory when obtaining a pointer. (can't be combined with GMEM_MOVEABLE).
GMEM_MOVEABLE	Allocates movable memory. In Win32, memory blocks are never moved in physical memory, but they can be moved within the default heap.
GMEM_ZEROINIT	Initializes memory contents to zero.
GMEM_MODIFY	Used to modify the attributes of an already-allocated block of memory.

GMEM_DDESHARE GMEM_SHARE	This flag is primarily for compatibility with 16-bit Window, but. can be used to enhance the performance of DDE operations. It should only be specified if the memory is to be used for DDE.
GPTR	Combines the GMEM_FIXED and GMEM_ZEROINIT flags.
GHND	Combines the GMEM_MOVEABLE and GMEM_ZEROINIT flags.

See Also

SetMemoryManager (see page 3660)

35.6.29 System.HPrevInst Variable

Pascal

```
HPrevInst: LongWord deprecated;
```

C++

```
LongWord deprecated HPrevInst;
```

File

System

Description

This is variable System.HPrevInst.

35.6.30 System.InitProc Variable

InitProc is the last installed initialization procedure.

Pascal

```
InitProc: Pointer;
```

C++

```
void * InitProc;
```

File

System

Description

Assign a value to InitProc to specify a procedure that you want to execute when the application starts up. The value of InitProc is a procedure with no parameters. For example:

In Delphi, the assignment can be made in the initialization section of a unit:

```
procedure MyInitProcedure;
...
initialization
  InitProc = @MyInitProcedure;
begin
end;
```

In C++, the assignment is most easily made in the project source file. This example is for Windows. On Linux applications, the code is similar.

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    // Note that call must precede call to Application->Initialize();
    System.InitProc = MyInitProcedure;
    Application->Initialize();
    Application->CreateForm(__classid(TForm1), &Form1);
    Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  catch (...)
  {
    try
    {
      throw Exception("");
    }
    catch (Exception &exception)
    {
      Application->ShowException(&exception);
    }
  }
  return 0;
}
```

Only one initialization procedure can be assigned to the InitProc variable. If your application defines more than one initialization procedure, only the last assignment to InitProc executes. To allow other initialization procedures to execute, you must "chain" the procedures together, calling the previously assigned value of InitProc from the new value.

See Also

TApplication

ExitProcessProc (see page 3824)

35.6.31 System.Input Variable

Specifies a read-only Text file associated with the processes standard Input file.

Pascal

```
Input: Text;
```

C++

```
Text Input;
```

File

System

Description

In Delphi, the Input variable is a read-only Text file associated with the process's standard Input file.

Note: On Windows, most processes don't have a standard Input file, and reading from Input raises an error. Delphi programs have a standard Input file if they are linked as console applications.

Note: On Linux, all processes have a standard Input file. There are various ways to use the standard Input. One simple method

is to run the program from a shell window.

See Also

Output (see page 3833)

35.6.32 System.IsConsole Variable

Indicates whether the module was compiled as a console application.

Pascal

```
IsConsole: Boolean;
```

C++

```
Boolean IsConsole;
```

File

System

Description

The IsConsole variable is true if the module was linked as a console application. This applies both to an executable and a shared library.

Note: On Linux, IsConsole is always true.

See Also

IsLibrary (see page 3828)

IsMultiThread (see page 3829)

Input (see page 3827)

Output (see page 3833)

ErrOutput (see page 3820)

35.6.33 System.IsLibrary Variable

Indicates whether the module is a shared library.

Pascal

```
IsLibrary: Boolean;
```

C++

```
Boolean IsLibrary;
```

File

System

Description

The IsLibrary variable is true if the module is a dynamic link library (Windows) or shared object (Linux).

See Also

IsConsole (see page 3828)

IsMultiThread (see page 3829)

35.6.34 System.IsMultiThread Variable

Indicates whether the application spawned additional threads using BeginThread (see page 3527) or TThread objects.

Pascal

```
IsMultiThread: Boolean;
```

C++

```
Boolean IsMultiThread;
```

File

System

Description

IsMultiThread is set to true to indicate that the memory manager should support multiple threads. IsMultiThread is set to true by BeginThread (see page 3527) and class factories.

See Also

BeginThread (see page 3527)

IsConsole (see page 3828)

IsLibrary (see page 3828)

CoInitFlags

TComObjectFactory

35.6.35 System.JITEnable Variable

Controls when the just-in-time debugger is called.

Pascal

```
JITEnable: Byte platform = 0;
```

C++

```
Byte platform JITEnable = 0;
```

File

System

Description

Use JITEnable to indicate what types of exceptions trigger the just-in-time debugger. JITEnable only has an effect on applications that run under Windows.

When JITEnable is 0 (the default), all exceptions are handled using the application's exception handling mechanism. The

just-in-time debugger is only called if the application provides no mechanism to catch exceptions. Note that any application that includes the Forms unit provides exception handling (via the Application object) for all but the initialization and finalization sections in Delphi units and does not trigger the just-in-time debugger.

When JITEnable is 1, any non-native exceptions (Exceptions raised by code that was compiled using other products) trigger the just-in-time debugger. Native exceptions are handled by the application's exception handling mechanism.

When JITEnable is greater than 1, all exceptions trigger the just-in-time debugger.

Note: When you install the IDE, it is registered as the just-in-time debugger. If you install other tools with debuggers, they may register themselves, overriding the registration of the IDE.

35.6.36 System.LibModuleList Variable

Lists module records for enumeration procedures.

Pascal

```
LibModuleList: PLibModule = nil;
```

C++

```
PLibModule LibModuleList = nil;
```

File

System

Description

LibModuleList implements a linked list of records for executables and packages in an application. LibModuleList is used by the EnumModules (see page 3556) and the EnumResourceModules (see page 3557) procedures.

See Also

EnumModules (see page 3556)

EnumResourceModules (see page 3557)

35.6.37 System.MainInstance Variable

Indicates the instance handle for the main executable.

Pascal

```
MainInstance: LongWord;
```

C++

```
LongWord MainInstance;
```

File

System

Description

Use MainInstance to obtain the instance handle for the main executable of an application. This is useful in applications that use runtime libraries or packages, when you need the handle for the executable rather than for the library.

Note:

See Also

FindHInstance (see page 3574)

35.6.38 System.MainThreadID Variable

Indicates the instance handle for the current module's main thread of execution.

Pascal

```
MainThreadID: LongWord;
```

C++

```
LongWord MainThreadID;
```

File

System

Description

Use MainThreadID to obtain the thread ID for the main execution thread (the current thread when the module was initialized).

During debugging, this ID appears in the thread status box for the main thread.

Note: On Windows, the value of MainThreadID is compatible with the thread ID required by some Win32 API calls.

See Also

TThread

35.6.39 System.ModuleUnloadList Variable

Pascal

```
ModuleUnloadList: PModuleUnloadRec = nil;
```

C++

```
PModuleUnloadRec ModuleUnloadList = nil;
```

File

System

Description

This is variable System.ModuleUnloadList.

35.6.40 System.MonitorSupport Variable

Pascal

```
MonitorSupport: PMonitorSupport;
```

C++

```
PMonitorSupport MonitorSupport;
```

File

System

Description

This is variable System.MonitorSupport.

35.6.41 System.NeverSleepOnMMThreadContention Variable

Dictates memory manager behavior during a thread contention.

Pascal

```
NeverSleepOnMMThreadContention: Boolean;
```

C++

```
Boolean NeverSleepOnMMThreadContention;
```

File

System

Description

The Memory Manager is a shared resource, and when many threads in the application attempt to perform a memory operation at the same time, one or more threads may have to wait for a pending operation in another thread to complete before it can continue. This situation is called a thread contention. When a thread contention occurs inside the Memory Manager the default behavior is to first relinquish the remaining time in the thread's time slice. If the resource is still not available when the thread enters its next time slice, the Memory Manager calls the OS procedure Sleep before trying again.

This behavior works well on machines with single or dual core CPUs, and also when the ratio of the number of running threads to number of CPU cores is relatively high (greater than 2:1). In other situations, better performance can be obtained by entering a busy waiting loop until the resource becomes available. If NeverSleepOnMMThreadContention is **True**, the Memory Manager will use this alternate strategy when a thread contention occurs. The default value for NeverSleepOnMMThreadContention is **False**.

See Also

Memory Management

Configuring the Memory Manager

35.6.42 System.NoErrMsg Variable

Controls whether the application displays an error message when a runtime error occurs.

Pascal

```
NoErrMsg: Boolean platform = False;
```

C++

```
Boolean platform NoErrMsg = False;
```

File

System

Description

Note: On Windows, set NoErrMsg to control whether a message appears to indicate that a runtime error has occurred. When NoErrMsg is false (the default), runtime errors cause the application to display a message box indicating the type of error that occurred. When NoErrMsg is true, these messages are suppressed.

On Linux, NoErrMsg only no effect.

Note: The SysUtils (see page 3901) unit turns most runtime errors into exceptions. If your application includes SysUtils (see page 3901), the runtime error message box is unlikely to appear, even if NoErrMsg is false.

35.6.43 System.Output Variable

Specifies a write-only Text file associated with the process's standard output file.

Pascal

```
Output: Text;
```

C++

```
Text Output;
```

File

System

Description

In Delphi, the Output variable is a write-only Text file associated with the process's standard output file.

Note: On Windows, most processes don't have a standard output file, and writing to Output raises an error. Delphi programs have a standard output file if they are linked as console applications.

Note: On Linux, all processes have a standard output file. There are various ways to view the standard error output. One simple method is to run the program from a shell window.

See Also

Input (see page 3827)

35.6.44 System.RaiseExceptionProc Variable

Points to RaiseException Windows function.

Pascal

```
RaiseExceptionProc: Pointer;
```

C++

```
void * RaiseExceptionProc;
```

File

System

Description

RaiseExceptionProc points to the RaiseException Windows API function. This can be used to invoke the function from assembly language.

35.6.45 System.RaiseExceptObjProc Variable

Pascal

```
RaiseExceptObjProc: Pointer;
```

C++

```
void * RaiseExceptObjProc;
```

File

System

Description

This is variable System.RaiseExceptObjProc.

35.6.46 System.RandSeed Variable

RandSeed stores the built-in random number generator's seed.

Pascal

```
RandSeed: Longint = 0;
```

C++

```
Longint RandSeed = 0;
```

File

System

Description

By assigning a specific value to RandSeed, you can use the Random (see page 3632) function to repetitively generate a specific sequence of random numbers. This is useful for applications that deal with data statistics and simulations. For applications that deal with encryption see the note under the Random (see page 3632) function.

See Also

Random (see page 3632)

Randomize (see page 3633)

35.6.47 System.ReportMemoryLeaksOnShutdown Variable

Report memory leaks on shutdown.

Pascal

```
ReportMemoryLeaksOnShutdown: Boolean;
```

C++

```
Boolean ReportMemoryLeaksOnShutdown;
```

File

System

Description

Set ReportMemoryLeaksOnShutdown to report memory leaks on shutdown.

The memory manager can report memory that was allocated but not freed by the time the memory manager shuts down. Such memory blocks are called memory leaks and are often the result of programming errors. When this global variable is set to **True**, the Memory Manager will scan the memory pool when it shuts down and report all unregistered memory leaks in a message dialog. The default value for ReportMemoryLeaksOnShutdown is **False**.

Expected memory leaks can be registered so the Memory Manager will ignore them during its checks for memory leaks. Use the procedures RegisterExpectedMemoryLeak (see page 3640) and UnregisterExpectedMemoryLeak (see page 3694) to control the memory areas that the Memory Manager checks for leaks.

See Also

Memory Management

Configuring the Memory Manager

Registering Memory Leaks

RegisterExpectedMemoryLeak (see page 3640)

UnregisterExpectedMemoryLeak (see page 3694)

35.6.48 System.RTLUnwindProc Variable

Pascal

```
RTLUnwindProc: Pointer;
```

C++

```
void * RTLUnwindProc;
```

File

System

Description

This is variable System.RTLUnwindProc.

35.6.49 System.SafeCallErrorProc Variable

SafeCallErrorProc defines a procedure that is called when a safecall routine incurs an error.

Pascal

```
SafeCallErrorProc: TSafeCallErrorProc;
```

C++

```
TSafeCallErrorProc SafeCallErrorProc;
```

File

System

Description

A function or procedure defined with the safecall directive is invoked using a safe calling convention. When such a routine incurs an error, the System::TObject::SafeCallException (see page 3380) is invoked. However, you can override this call by defining your own handling routine, and setting SafeCallErrorProc to point to it.

SafeCallErrorProc should point to a procedure that takes two parameters :

ErrorCode is an integer holding the error code.

ErrorAddr (see page 3817) is the code address at the time of error.

If SafeCallErrorProc returns normally, or is set to nil, then the ESafecallException is raised.

See Also

ESafecallException (see page 3946)

System::TObject::SafeCallException (see page 3380)

35.6.50 System.SystemThreadEndProc Variable

Pascal

```
SystemThreadEndProc: TSystemThreadEndProc = nil;
```

C++

```
TSystemThreadEndProc SystemThreadEndProc = nil;
```

File

System

Description

This is variable System.SystemThreadEndProc.

35.6.51 System.SystemThreadFuncProc Variable

Pascal

```
SystemThreadFuncProc: TSystemThreadFuncProc = nil;
```

C++

```
TSystemThreadFuncProc SystemThreadFuncProc = nil;
```

File

System

Description

This is variable System.SystemThreadFuncProc.

35.6.52 System.Test8086 Variable

Pascal

```
Test8086: Byte;
```

C++

```
Byte Test8086;
```

File

System

Description

This is variable System.Test8086.

35.6.53 System.Test8087 Variable

Pascal

```
Test8087: Byte = 3;
```

C++

```
Byte Test8087 = 3;
```

File

System

Description

This is variable System.Test8087.

35.6.54 System.TestFDIV Variable

Pascal

```
TestFDIV: Shortint;
```

C++

```
Shortint TestFDIV;
```

File

System

Description

This is variable System.TestFDIV.

35.6.55 System.UTF8CompareLocale Variable

Pascal

```
UTF8CompareLocale: Cardinal;
```

C++

```
unsigned UTF8CompareLocale;
```

File

System

Description

This is variable System.UTF8CompareLocale.

35.6.56 System.VarAddRefProc Variable

The VarAddRefProc variable is used internally.

Pascal

```
VarAddRefProc: procedure (var v: TVarData) = nil;
```

C++

```
procedure (var v: TVarData) VarAddRefProc = nil;
```

File

System

Description

Do not modify or use the VarAddRefProc variable manually. VarAddRefProc is used internally by the Variant (see page 3421) manager.

Note: Applications should not modify or rely on the value of the VarAddRefProc variable. Changes made to VarAddRefProc may lead to unexpected errors.

See Also

VarClearProc (see page 3839)

VarCopyProc (see page 3839)

VarToLStrProc (see page 3840)

VarToWStrProc (see page 3840)

35.6.57 System.VarClearProc Variable

The VarClearProc variable is used internally.

Pascal

```
VarClearProc: procedure (var v: TVarData) = nil;
```

C++

```
procedure (var v: TVarData) VarClearProc = nil;
```

File

System

Description

Do not modify or use the VarClearProc variable manually. VarClearProc is used internally by the Variant (see page 3421) manager.

Note: Applications should not modify or rely on the value of the VarClearProc variable. Changes made to VarClearProc may lead to unexpected errors.

See Also

VarAddRefProc (see page 3838)

VarCopyProc (see page 3839)

VarToLStrProc (see page 3840)

VarToWStrProc (see page 3840)

35.6.58 System.VarCopyProc Variable

The VarCopyProc variable is used internally.

Pascal

```
VarCopyProc: procedure (var Dest: TVarData; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: TVarData; const Source: TVarData) VarCopyProc = nil;
```

File

System

Description

Do not modify or use the VarCopyProc variable manually. VarCopyProc is used internally by the Variant (see page 3421) manager.

Note: Applications should not modify or rely on the value of the VarCopyProc variable. Changes made to VarCopyProc may lead to unexpected errors.

See Also

VarClearProc (see page 3839)

VarAddRefProc (see page 3838)

VarToLStrProc (see page 3840)

VarToWStrProc (see page 3840)

35.6.59 System.VarToLStrProc Variable

The VarToLStrProc variable is used internally.

Pascal

```
VarToLStrProc: procedure (var Dest: AnsiString; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: AnsiString; const Source: TVarData) VarToLStrProc = nil;
```

File

System

Description

Do not modify or use the VarToLStrProc variable manually. VarToLStrProc is used internally by the Variant (see page 3421) manager.

Note: Applications should not modify or rely on the value of the VarToLStrProc variable. Changes made to VarToLStrProc may lead to unexpected errors.

See Also

VarClearProc (see page 3839)

VarAddRefProc (see page 3838)

VarCopyProc (see page 3839)

VarToWStrProc (see page 3840)

35.6.60 System.VarToWStrProc Variable

The VarToWStrProc variable is used internally.

Pascal

```
VarToWStrProc: procedure (var Dest: WideString; const Source: TVarData) = nil;
```

C++

```
procedure (var Dest: WideString; const Source: TVarData) VarToWStrProc = nil;
```

File

System

Description

Do not modify or use the VarToWStrProc variable manually. VarToWStrProc is used internally by the Variant (see page 3421) manager.

Note: Applications should not modify or rely on the value of VarToWStrProc variable. Changes made to VarToWStrProc may lead to unexpected errors.

See Also

VarClearProc (see page 3839)

VarAddRefProc (see page 3838)

VarCopyProc (see page 3839)

VarToLStrProc (see page 3840)

35.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CompilerVersion (see page 3844)	Designates the version number of the Delphi compiler.
CPUi386 (see page 3844)	This is constant System.CPUi386.
CPUi486 (see page 3845)	This is constant System.CPUi486.
CPUPentium (see page 3845)	This is constant System.CPUPentium.
E_NOINTERFACE (see page 3845)	This is constant System.E_NOINTERFACE.
E_NOTIMPL (see page 3846)	This is constant System.E_NOTIMPL.
E_UNEXPECTED (see page 3846)	This is constant System.E_UNEXPECTED.
fmClosed (see page 3846)	This is constant System.fmClosed.
fmInOut (see page 3847)	This is constant System.fmInOut.
fmInput (see page 3847)	This is constant System.fmInput.
fmOutput (see page 3847)	This is constant System.fmOutput.
hfFieldSize (see page 3848)	This is constant System.hfFieldSize.
hfMonitorOffset (see page 3848)	This is constant System.hfMonitorOffset.
INFINITE (see page 3848)	This is constant System.INFINITE.
MaxInt (see page 3849)	The maximum value of the Integer data type.
MaxLongint (see page 3849)	The maximum value of the Longint data type.
NumSmallBlockTypes (see page 3849)	NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.
opAdd (see page 3850)	opAdd defines an addition operation.
opAnd (see page 3850)	opAnd defines a logical and operation.
opCmpEQ (see page 3851)	opCmpEQ defines an equality comparison operator.
opCmpGE (see page 3851)	opCmpGE defines a greater than or equal comparison operator.
opCmpGT (see page 3852)	opCmpGT defines a greater than comparison operator.
opCmpLE (see page 3852)	opCmpLE defines a less than or equal comparison operator.
opCmpLT (see page 3853)	opCmpLT defines a less than comparison operator.
opCmpNE (see page 3853)	opCmpNE defines an inequality comparison operator.
opCompare (see page 3854)	opCompare defines a comparison operation.
opDivide (see page 3854)	opDivide defines a divide operation.
opIntDivide (see page 3855)	opIntDivide defines an integer divide operation.
opModulus (see page 3855)	opModulus defines a modulus operation.
opMultiply (see page 3855)	opMultiply defines a multiply operation.
opNegate (see page 3856)	opNegate defines a negate operation.
opNot (see page 3856)	opNot defines a binary negation operation.
opOr (see page 3857)	opOr defines a logical or operation.
opShiftLeft (see page 3857)	opShiftLeft defines a bitwise shift left operation.
opShiftRight (see page 3858)	opShiftRight defines a bitwise shift right operation.
opSubtract (see page 3858)	opSubtract defines a subtraction operation.
opXor (see page 3859)	opXor defines a logical exclusive or operation.
RTLVersion (see page 3859)	Designates the version number of the Delphi runtime library.
S_FALSE (see page 3859)	This is constant System.S_FALSE.
S_OK (see page 3860)	This is constant System.S_OK.
sLineBreak (see page 3860)	Represents the character(s) used to generate a line feed.
tfCRLF (see page 3861)	This is constant System.tfCRLF.
varAny (see page 3861)	Represents a Variant (see page 3421) that can hold any value.
varArray (see page 3861)	Represents a Variant (see page 3421) that holds dynamic array data.

varBoolean (see page 3862)	Represents a Variant (see page 3421) that holds WordBool data.
varByRef (see page 3862)	Represents a Variant (see page 3421) that points to another variable.
varByte (see page 3863)	Represents a Variant (see page 3421) that holds byte data.
varCurrency (see page 3863)	Represents a Variant (see page 3421) that holds a monetary floating point value.
varDate (see page 3864)	Represents a Variant (see page 3421) that holds a TDateTime (see page 3323) value.
varDispatch (see page 3864)	Represents a Variant (see page 3421) that has been assigned an IDispatch (see page 3494) value.
varDouble (see page 3865)	Represents a Variant (see page 3421) that holds a floating point value.
varEmpty (see page 3865)	Represents a Variant (see page 3421) that has not yet been assigned a value.
varError (see page 3866)	Represents a Variant (see page 3421) that holds a 32 bit error code.
varInt64 (see page 3867)	Represents a Variant (see page 3421) that holds Int64 data.
varInteger (see page 3867)	Represents a Variant (see page 3421) that holds Integer data.
varLongWord (see page 3868)	Represents a Variant (see page 3421) that holds LongWord data.
varNull (see page 3868)	Represents a Variant (see page 3421) that holds a null value.
varOleStr (see page 3869)	Represents a Variant (see page 3421) that has been assigned a COM interface type string value.
varShortInt (see page 3869)	Represents a Variant (see page 3421) that holds ShortInt data.
varSingle (see page 3870)	Represents a Variant (see page 3421) that holds byte data.
varSmallint (see page 3870)	Represents a Variant (see page 3421) that holds Smallint data.
varStrArg (see page 3871)	Represents a Variant (see page 3421) that has holds a COM compatible string value.
varString (see page 3871)	Represents a Variant (see page 3421) that holds string data.
varTypeMask (see page 3872)	Defines a mask for filtering out the meta data part of a Variant (see page 3421) data type.
varUInt64 (see page 3872)	This is constant System.varUInt64.
varUnknown (see page 3873)	Represents a Variant (see page 3421) that holds an unknown value.
varUString (see page 3873)	This is constant System.varUString.
varVariant (see page 3873)	Represents a Variant (see page 3421) data type.
varWord (see page 3874)	Represents a Variant (see page 3421) that holds Word data.
vmtAddRef (see page 3874)	This is constant System.vmtAddRef.
vmtAfterConstruction (see page 3875)	This is constant System.vmtAfterConstruction.
vmtAutoTable (see page 3875)	This is constant System.vmtAutoTable.
vmtBeforeDestruction (see page 3875)	This is constant System.vmtBeforeDestruction.
vmtClassName (see page 3876)	This is constant System.vmtClassName.
vmtCreateObject (see page 3876)	This is constant System.vmtCreateObject.
vmtDefaultHandler (see page 3876)	This is constant System.vmtDefaultHandler.
vmtDestroy (see page 3877)	This is constant System.vmtDestroy.
vmtDispatch (see page 3877)	This is constant System.vmtDispatch.
vmtDynamicTable (see page 3877)	This is constant System.vmtDynamicTable.
vmtEquals (see page 3878)	This is constant System.vmtEquals.
vmtFieldTable (see page 3878)	This is constant System.vmtFieldTable.
vmtFreeInstance (see page 3878)	This is constant System.vmtFreeInstance.
vmtGetHashCode (see page 3879)	This is constant System.vmtGetHashCode.
vmtInitTable (see page 3879)	This is constant System.vmtInitTable.
vmtInstanceSize (see page 3879)	This is constant System.vmtInstanceSize.
vmtIntfTable (see page 3880)	This is constant System.vmtIntfTable.
vmtMethodTable (see page 3880)	This is constant System.vmtMethodTable.
vmtNewInstance (see page 3880)	This is constant System.vmtNewInstance.
vmtParent (see page 3881)	This is constant System.vmtParent.
vmtQueryInterface (see page 3881)	This is constant System.vmtQueryInterface.
vmtRelease (see page 3881)	This is constant System.vmtRelease.
vmtSafeCallException (see page 3882)	This is constant System.vmtSafeCallException.
vmtSelfPtr (see page 3882)	This is constant System.vmtSelfPtr.
vmtToString (see page 3882)	This is constant System.vmtToString.

vmtTypeInfo (see page 3883)	This is constant System.vmtTypeInfo.
vtAnsiString (see page 3883)	vtAnsiString defines an AnsiString TVarData (see page 3740) VType value.
vtBoolean (see page 3883)	vtBoolean defines a Boolean TVarData (see page 3740) VType value.
vtChar (see page 3884)	vtChar defines a Char TVarData (see page 3740) VType value.
vtClass (see page 3884)	vtClass defines a TClass (see page 3785) TVarData (see page 3740) VType value.
vtCurrency (see page 3885)	vtCurrency defines a Currency (see page 3235) TVarData (see page 3740) VType value.
vtExtended (see page 3885)	vtExtended defines an Extended TVarData (see page 3740) VType value.
vtInt64 (see page 3886)	vtInt64 defines an Int64 TVarData (see page 3740) VType value.
vtInteger (see page 3886)	vtInteger defines an Integer TVarData (see page 3740) VType value.
vtInterface (see page 3887)	vtInterface defines an Interface pointer TVarData (see page 3740) VType value.
vtObject (see page 3887)	vtObject defines a TObject (see page 3354) TVarData (see page 3740) VType value.
vtPChar (see page 3888)	vtPChar defines a PChar TVarData (see page 3740) VType value.
vtPointer (see page 3888)	vtPointer defines a Pointer TVarData (see page 3740) VType value.
vtPWideChar (see page 3889)	vtPWideChar defines a PWideChar TVarData (see page 3740) VType value.
vtString (see page 3889)	vtString defines a String TVarData (see page 3740) VType value.
vtUnicodeString (see page 3890)	This is constant System.vtUnicodeString.
vtVariant (see page 3890)	vtString (see page 3889) defines a Variant (see page 3421) TVarData (see page 3740) VType value.
vtWideChar (see page 3890)	vtWideChar defines a WideChar TVarData (see page 3740) VType value.
vtWideString (see page 3891)	vtWideString defines a WideString (see page 3467) TVarData (see page 3740) VType value.

35.7.1 System.CompilerVersion Constant

Designates the version number of the Delphi compiler.

Pascal

```
CompilerVersion = 18.0;
```

C++

```
CompilerVersion = 18.0;
```

File

System

Description

Use CompilerVersion to test the version level of the Delphi compiler. This is useful if you want to conditionalize code that takes advantage of newer features in the Delphi language.

See Also

RTLVersion (see page 3859)

35.7.2 System.CPUi386 Constant

Pascal

```
CPUi386 = 2;
```

C++

```
CPUi386 = 2;
```

File

System

Description

This is constant System.CPUi386.

35.7.3 System.CPUi486 Constant

Pascal

```
CPUi486 = 3;
```

C++

```
CPUi486 = 3;
```

File

System

Description

This is constant System.CPUi486.

35.7.4 System.CPUPentium Constant

Pascal

```
CPUPentium = 4;
```

C++

```
CPUPentium = 4;
```

File

System

Description

This is constant System.CPUPentium.

35.7.5 System.E_NOINTERFACE Constant

Pascal

```
E_NOINTERFACE = HRESULT($80004002);
```

C++

```
E_NOINTERFACE = HRESULT($80004002);
```

File

System

Description

This is constant System.E_NOINTERFACE.

35.7.6 System.E_NOTIMPL Constant

Pascal

```
E_NOTIMPL = HRESULT($80004001);
```

C++

```
E_NOTIMPL = HRESULT($80004001);
```

File

System

Description

This is constant System.E_NOTIMPL.

35.7.7 System.E_UNEXPECTED Constant

Pascal

```
E_UNEXPECTED = HRESULT($8000FFFF);
```

C++

```
E_UNEXPECTED = HRESULT($8000FFFF);
```

File

System

Description

This is constant System.E_UNEXPECTED.

35.7.8 System.fmClosed Constant

Pascal

```
fmClosed = $D7B0;
```

C++

```
fmClosed = 0xD7B0;
```

File

System

Description

This is constant System.fmClosed.

35.7.9 System.fmInOut Constant

Pascal

```
fmInOut = $D7B3;
```

C++

```
fmInOut = 0xD7B3;
```

File

System

Description

This is constant System.fmInOut.

35.7.10 System.fmInput Constant

Pascal

```
fmInput = $D7B1;
```

C++

```
fmInput = 0xD7B1;
```

File

System

Description

This is constant System.fmInput.

35.7.11 System.fmOutput Constant

Pascal

```
fmOutput = $D7B2;
```

C++

```
fmOutput = 0xD7B2;
```

File

System

Description

This is constant System.fmOutput.

35.7.12 System.hfFieldSize Constant

Pascal

```
hfFieldSize = 4;
```

C++

```
hfFieldSize = 4;
```

File

System

Description

This is constant System.hfFieldSize.

35.7.13 System.hfMonitorOffset Constant

Pascal

```
hfMonitorOffset = 0;
```

C++

```
hfMonitorOffset = 0;
```

File

System

Description

This is constant System.hfMonitorOffset.

35.7.14 System.INFINITE Constant

Pascal

```
INFINITE = Cardinal($FFFFFFFF);
```

C++

```
INFINITE = Cardinal($FFFFFFFF);
```

File

System

Description

This is constant System.INFINITE.

35.7.15 System.MaxInt Constant

The maximum value of the Integer data type.

Pascal

```
MaxInt = 2147483647;
```

C++

```
MaxInt = 2147483647;
```

File

System

Description

MaxInt represents the highest value in the range of the integer data type (2147483647).

See Also

High (see page 3596)

35.7.16 System.MaxLongint Constant

The maximum value of the Longint data type.

Pascal

```
MaxLongint = 2147483647;
```

C++

```
MaxLongint = 2147483647;
```

File

System

Description

MaxLongint represents the highest value in the range of the Longint (Delphi) or int (C++) data type (2147483647).

See Also

High (see page 3596)

35.7.17 System.NumSmallBlockTypes Constant

NumSmallBlockTypes is the number of small memory block types employed by the default memory manager.

Pascal

```
NumSmallBlockTypes = 55;
```

C++

```
NumSmallBlockTypes = 55;
```

File

System

Description

GetMemoryManagerState (see page 3591) returns a structure containing memory usage statistics for each block type - small, medium and large blocks. It returns this data in a System::TMemoryManagerState (see page 3730) record. Within that record is held a System::TSmallBlockTypeStates (see page 3795) field - an array of System::TSmallBlockTypeState (see page 3736) records. There are NumSmallBlockTypes (55) different small memory block sizes, and therefore 55 System::TSmallBlockTypeState (see page 3736) records in this array.

See Also

GetMemoryManagerState (see page 3591)

System::TMemoryManagerState (see page 3730)

System::TSmallBlockTypeState (see page 3736)

System::TSmallBlockTypeStates (see page 3795)

35.7.18 System.opAdd Constant

opAdd defines an addition operation.

Pascal

```
opAdd = 0;
```

C++

```
opAdd = 0;
```

File

System

Description

opAdd is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opAdd represents an addition of two operands.

See Also

TVarOp (see page 3799)

35.7.19 System.opAnd Constant

opAnd defines a logical and operation.

Pascal

```
opAnd = 8;
```

C++

```
opAnd = 8;
```

File

System

Description

opAnd is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opAnd represents a logical and of two operands.

See Also

TVarOp (see page 3799)

35.7.20 System.opCmpEQ Constant

opCmpEQ defines an equality comparison operator.

Pascal

```
opCmpEQ = 14;
```

C++

```
opCmpEQ = 14;
```

File

System

Description

opCmpEQ is one of the opCmp* comparison values that TVarOp (see page 3799) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpEQ represents a comparison that two operands are equal in value.

See Also

CompareOp (see page 4998)

TVarOp (see page 3799)

35.7.21 System.opCmpGE Constant

opCmpGE defines a greater than or equal comparison operator.

Pascal

```
opCmpGE = 19;
```

C++

```
opCmpGE = 19;
```

File

System

Description

opCmpGE is one of the opCmp* comparison values that TVarOp (see page 3799) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpGE represents the comparison : left operand greater than or equal to the right operand.

See Also

CompareOp (see page 4998)

TVarOp (see page 3799)

35.7.22 System.opCmpGT Constant

opCmpGT defines a greater than comparison operator.

Pascal

```
opCmpGT = 18;
```

C++

```
opCmpGT = 18;
```

File

System

Description

opCmpGT is one of the opCmp* comparison values that TVarOp (see page 3799) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpGT represents the comparison : left operand greater than the right operand.

See Also

CompareOp (see page 4998)

TVarOp (see page 3799)

35.7.23 System.opCmpLE Constant

opCmpLE defines a less than or equal comparison operator.

Pascal

```
opCmpLE = 17;
```

C++

```
opCmpLE = 17;
```

File

System

Description

opCmpLE is one of the opCmp* comparison values that TVarOp (see page 3799) may be set to when performing left/right

operand comparisons, such as CompareOp.

opCmpLE represents the comparison : left operand less than or equal to the right operand.

See Also

CompareOp (see page 4998)

TVarOp (see page 3799)

35.7.24 System.opCmpLT Constant

opCmpLT defines a less than comparison operator.

Pascal

```
opCmpLT = 16;
```

C++

```
opCmpLT = 16;
```

File

System

Description

opCmpLT is one of the opCmp* comparison values that TVarOp (see page 3799) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpLT represents the comparison : left operand less than the right operand.

See Also

CompareOp (see page 4998)

TVarOp (see page 3799)

35.7.25 System.opCmpNE Constant

opCmpNE defines an inequality comparison operator.

Pascal

```
opCmpNE = 15;
```

C++

```
opCmpNE = 15;
```

File

System

Description

opCmpNE is one of the opCmp* comparison values that TVarOp (see page 3799) may be set to when performing left/right operand comparisons, such as CompareOp.

opCmpNE represents a comparison that two operands are not equal in value.

See Also

CompareOp (see page 4998)

TVarOp (see page 3799)

35.7.26 System.opCompare Constant

opCompare defines a comparison operation.

Pascal

```
opCompare = 11;
```

C++

```
opCompare = 11;
```

File

System

Description

opCompare defines a comparison operation.

See Also

TVarOp (see page 3799)

35.7.27 System.opDivide Constant

opDivide defines a divide operation.

Pascal

```
opDivide = 3;
```

C++

```
opDivide = 3;
```

File

System

Description

opDivide is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opDivide represents the division of the left operand by the right operation.

See Also

TVarOp (see page 3799)

35.7.28 System.opIntDivide Constant

opIntDivide defines an integer divide operation.

Pascal

```
opIntDivide = 4;
```

C++

```
opIntDivide = 4;
```

File

System

Description

opIntDivide is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opIntDivide represents the integer division of the left operand by the right operation.

See Also

TVarOp (see page 3799)

35.7.29 System.opModulus Constant

opModulus defines a modulus operation.

Pascal

```
opModulus = 5;
```

C++

```
opModulus = 5;
```

File

System

Description

opModulus is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opModulus represents the remainder of the division of the left operand by the right operand.

See Also

TVarOp (see page 3799)

35.7.30 System.opMultiply Constant

opMultiply defines a multiply operation.

Pascal

```
opMultiply = 2;
```

C++

```
opMultiply = 2;
```

File

System

Description

opMultiply is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opMultiply represents the multiplication of the left operand by the right operation.

See Also

TVarOp (see page 3799)

35.7.31 System.opNegate Constant

opNegate defines a negate operation.

Pascal

```
opNegate = 12;
```

C++

```
opNegate = 12;
```

File

System

Description

opNegate is one of the operation types that TVarOp (see page 3799) may be set to when performing unary operations such as UnaryOp.

opNegate represents the inversion of an operand value.

See Also

TVarOp (see page 3799)

35.7.32 System.opNot Constant

opNot defines a binary negation operation.

Pascal

```
opNot = 13;
```

C++

```
opNot = 13;
```

File

System

Description

opNegate (see page 3856) represents the binary (bitwise) inversion of an operand value.

See Also

TVarOp (see page 3799)

35.7.33 System.opOr Constant

opOr defines a logical or operation.

Pascal

```
opOr = 9;
```

C++

```
opOr = 9;
```

File

System

Description

opOr is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opOr represents a logical or of two operands.

See Also

TVarOp (see page 3799)

35.7.34 System.opShiftLeft Constant

opShiftLeft defines a bitwise shift left operation.

Pascal

```
opShiftLeft = 6;
```

C++

```
opShiftLeft = 6;
```

File

System

Description

opShiftLeft is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opShiftLeft represents the left operand shifted left N times, where N is the numeric value of the right operand.

See Also

TVarOp (see page 3799)

35.7.35 System.opShiftRight Constant

opShiftRight defines a bitwise shift right operation.

Pascal

```
opShiftRight = 7;
```

C++

```
opShiftRight = 7;
```

File

System

Description

opShiftRight is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opShiftRight represents the left operand shifted right N times, where N is the numeric value of the right operand.

See Also

TVarOp (see page 3799)

35.7.36 System.opSubtract Constant

opSubtract defines a subtraction operation.

Pascal

```
opSubtract = 1;
```

C++

```
opSubtract = 1;
```

File

System

Description

opSubtract is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opSubtract represents the right operand subtracted from the left operand.

See Also

TVarOp (see page 3799)

35.7.37 System.opXor Constant

opXor defines a logical exclusive or operation.

Pascal

```
opXor = 10;
```

C++

```
opXor = 10;
```

File

System

Description

opXor is one of the operation types that TVarOp (see page 3799) may be set to when performing binary operations such as BinaryOp.

opXor represents a logical exclusive or of two operands.

See Also

TVarOp (see page 3799)

35.7.38 System.RTLVersion Constant

Designates the version number of the Delphi runtime library.

Pascal

```
RTLVersion = 20.00;
```

C++

```
RTLVersion = 20.00;
```

File

System

Description

Use RTLVersion to test the Delphi runtime library version level independently of the compiler version level.

See Also

CompilerVersion (see page 3844)

35.7.39 System.S_FALSE Constant

Pascal

```
S_FALSE = $00000001;
```

C++

```
S_FALSE = 0x00000001;
```

File

System

Description

This is constant System.S_FALSE.

35.7.40 System.S_OK Constant

Pascal

```
S_OK = 0;
```

C++

```
S_OK = 0;
```

File

System

Description

This is constant System.S_OK.

35.7.41 System.sLineBreak Constant

Represents the character(s) used to generate a line feed.

Pascal

```
sLineBreak = AnsiString(#13#10);
```

C++

```
sLineBreak = AnsiString(#13#10);
```

File

System

Description

Windows systems define sLineBreak as a combination of carriage return (#13) and line feed (#10).

Linux systems define sLineBreak as simply a line feed (#10).

It is preferable to use this constabnt rather than hard code #13#10 or #10 (as appropriate) to move to a new line in a text output.

35.7.42 System.tfCRLF Constant

Pascal

```
tfCRLF = $1;
```

C++

```
tfCRLF = 0x1;
```

File

System

Description

This is constant System.tfCRLF.

35.7.43 System.varAny Constant

Represents a Variant (see page 3421) that can hold any value.

Pascal

```
varAny = $0101;
```

C++

```
varAny = 0x0101;
```

File

System

Description

A variant with a varAny type is used as a placeholder for any variant value, such as when passed as a paremeter to a routine. A varAny variant cannot be directly processed - use ChangeAnyProc to convert it to a variant type that can be processed.

See Also

TVarType (see page 3801)

ChangeAnyProc (see page 5101)

VarType (see page 5090)

35.7.44 System.varArray Constant

Represents a Variant (see page 3421) that holds dynamic array data.

Pascal

```
varArray = $2000;
```

C++

```
varArray = 0x2000;
```

File

System

Description

The VarType routine returns a TVarType (see page 3801) value that describes the current data type used to hold the contents of a Variant (see page 3421) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The low order bits determine the basic data type, such as varDouble (see page 3865), varSmallInt.

The high order bits represent meta data types varArray and varByRef (see page 3862).

Logically And varArray with a Variant (see page 3421) to determine if a Variant (see page 3421) holds array data.

Use VarArrayCreate to generate an array Variant (see page 3421);

See Also

TVarType (see page 3801)

VarArrayCreate (see page 5059)

VarType (see page 5090)

35.7.45 System.varBoolean Constant

Represents a Variant (see page 3421) that holds WordBool data.

Pascal

```
varBoolean = $000B;
```

C++

```
varBoolean = 0x000B;
```

File

System

Description

varBoolean represents a Variant (see page 3421) variable that currently holds a 16-bit boolean (type WordBool) value.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.46 System.varByRef Constant

Represents a Variant (see page 3421) that points to another variable.

Pascal

```
varByRef = $4000;
```

C++

```
varByRef = 0x4000;
```

File

System

Description

The VarType routine returns a TVarType (see page 3801) value that describes the current data type used to hold the contents of a Variant (see page 3421) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The low order bits determine the basic data type, such as varDouble (see page 3865), varSmallInt.

The high order bits represent meta data types varArray (see page 3861) and varByRef.

In essence, a varByRef variant is a pointer. The data type of the variable it points to is determined in the low order bits.

Logically And varByRef with a Variant (see page 3421) to determine if a Variant (see page 3421) is a pointer.

See Also

TVarType (see page 3801)

VarType (see page 5090)

35.7.47 System.varByte Constant

Represents a Variant (see page 3421) that holds byte data.

Pascal

```
varByte = $0011;
```

C++

```
varByte = 0x0011;
```

File

System

Description

When a Variant (see page 3421) is assigned an integer value less than 256, it will be set to a varByte Variant (see page 3421) type. varByte represents a Variant (see page 3421) variable that currently holds a byte data value.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.48 System.varCurrency Constant

Represents a Variant (see page 3421) that holds a monetary floating point value.

Pascal

```
varCurrency = $0006;
```

C++

```
varCurrency = 0x0006;
```

File

System

Description

When a Variant (see page 3421) is assigned a floating point value with less than 5 decimal places, it is set to the varCurrency type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.49 System.varDate Constant

Represents a Variant (see page 3421) that holds a TDateTime (see page 3323) value.

Pascal

```
varDate = $0007;
```

C++

```
varDate = 0x0007;
```

File

System

Description

When a Variant (see page 3421) is assignned a TDateTime (see page 3323) value, it is set to the varDate type.

Use VarType to determine the variant type.

See Also

TDateTime (see page 3787)

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.50 System.varDispatch Constant

Represents a Variant (see page 3421) that has been assigned an IDispatch (see page 3494) value.

Pascal

```
varDispatch = $0009;
```

C++

```
varDispatch = 0x0009;
```

File

System

Description

When a Variant (see page 3421) is assigned an IDispatch (see page 3494) value, it is set to the varDispatch type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.51 System.varDouble Constant

Represents a Variant (see page 3421) that holds a floating point value.

Pascal

```
varDouble = $0005;
```

C++

```
varDouble = 0x0005;
```

File

System

Description

When a Variant (see page 3421) is assigned a floating point value with more than 4 decimal places, it is set to the varDouble type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.52 System.varEmpty Constant

Represents a Variant (see page 3421) that has not yet been assigned a value.

Pascal

```
varEmpty = $0000;
```

C++

```
varEmpty = 0x0000;
```

File

System

Description

When a Variant (see page 3421) is first created, it is set to the varEmpty type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarIsEmpty (see page 5080)

VarType (see page 5090)

35.7.53 System.varError Constant

Represents a Variant (see page 3421) that holds a 32 bit error code.

Pascal

```
varError = $000A;
```

C++

```
varError = 0x000A;
```

File

System

Description

When a Variant (see page 3421) is assigned a 32 bit error code, it is set to the varError type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarIsEmpty (see page 5080)

VarType (see page 5090)

35.7.54 System.varInt64 Constant

Represents a Variant (see page 3421) that holds Int64 data.

Pascal

```
varInt64 = $0014;
```

C++

```
varInt64 = 0x0014;
```

File

System

Description

When a Variant (see page 3421) is assigned an integer value greater in value than an LongWord type can hold, it will be set to a varInt64 Variant (see page 3421) type. varInt64 represents a Variant (see page 3421) variable that currently holds an Int64 data value.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.55 System.varInteger Constant

Represents a Variant (see page 3421) that holds Integer data.

Pascal

```
varInteger = $0003;
```

C++

```
varInteger = 0x0003;
```

File

System

Description

To define a Variant (see page 3421) to be a varInteger type, you should use VarAsType to create one. By default, a Variant (see page 3421) with an integer value greater than the Word type capacity, but less than the Int64 type capacity will normally be assigned a varLongWord (see page 3868) type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.56 System.varLongWord Constant

Represents a Variant (see page 3421) that holds LongWord data.

Pascal

```
varLongWord = $0013;
```

C++

```
varLongWord = 0x0013;
```

File

System

Description

When a Variant (see page 3421) is assigned an integer value greater in value than a Word type can hold, but less than an Int64 type can hold, it will be set to a varLongWord Variant (see page 3421) type. varLongWord represents a Variant (see page 3421) variable that currently holds a LongWord data value.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.57 System.varNull Constant

Represents a Variant (see page 3421) that holds a null value.

Pascal

```
varNull = $0001;
```

C++

```
varNull = 0x0001;
```

File

System

Description

When a Variant (see page 3421) is assigned a null value, it is set to the varNull type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarIsNull (see page 5083)

VarType (see page 5090)

35.7.58 System.varOleStr Constant

Represents a Variant (see page 3421) that has been assigned a COM interface type string value.

Pascal

```
varOleStr = $0008;
```

C++

```
varOleStr = 0x0008;
```

File

System

Description

When a Variant (see page 3421) is assigned a string value via a COM interface, it is set to the varOleStr type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.59 System.varShortInt Constant

Represents a Variant (see page 3421) that holds ShortInt data.

Pascal

```
varShortInt = $0010;
```

C++

```
varShortInt = 0x0010;
```

File

System

Description

To define a Variant (see page 3421) to be a varShortInt type, you should use VarAsType to create one. By default, a Variant (see page 3421) with an integer value less than the Word type capacity will normally be assigned a varByte (see page 3863) type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.60 System.varSingle Constant

Represents a Variant (see page 3421) that holds byte data.

Pascal

```
varSingle = $0004;
```

C++

```
varSingle = 0x0004;
```

File

System

Description

To define a Variant (see page 3421) to be a varSingle type, you should use VarAsType to create one. By default, a Variant (see page 3421) with a floating point value with less than 4 decimal places will be assigned the varCurrency (see page 3863) type. With greater than 4 decimal places, it will be assigned by default the varDouble (see page 3865) type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.61 System.varSmallint Constant

Represents a Variant (see page 3421) that holds Smallint data.

Pascal

```
varSmallint = $0002;
```

C++

```
varSmallint = 0x0002;
```

File

System

Description

To define a Variant (see page 3421) to be a varSmallint type, you should use VarAsType to create one. By default, a Variant (see page 3421) with an integer value less than the Byte type capacity, but less than the LongWord type capacity will normally be assigned a varWord (see page 3874) type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.62 System.varStrArg Constant

Represents a Variant (see page 3421) that has holds a COM compatible string value.

Pascal

```
varStrArg = $0048;
```

C++

```
varStrArg = 0x0048;
```

File

System

Description

In order to use a Variant (see page 3421) with a COM interface, you should create a varStrArg type using VarAsType.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.63 System.varString Constant

Represents a Variant (see page 3421) that holds string data.

Pascal

```
varString = $0100;
```

C++

```
varString = 0x0100;
```

File

System

Description

When a Variant (see page 3421) is assigned a string value, it will be set to a varString Variant (see page 3421) type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.64 System.varTypeMask Constant

Defines a mask for filtering out the meta data part of a Variant (see page 3421) data type.

Pascal

```
varTypeMask = $0FFF;
```

C++

```
varTypeMask = 0x0FFF;
```

File

System

Description

The VarType routine returns a TVarType (see page 3801) value that describes the current data type used to hold the contents of a Variant (see page 3421) data type. The data type is a 16 bit value, split into a high order 4 bits and a low order 12 bits. The high order bits give a meta data type (varArray (see page 3861) and varByRef (see page 3862)). The low order bits determine the basic data type, such as varDouble (see page 3865), varSmallInt. Use varTypeMask to mask out the meta data so as to be able to determine the basic data type of a Variant (see page 3421).

See Also

TVarType (see page 3801)

varArray (see page 3861)

VarAsType (see page 5073)

varByRef (see page 3862)

VarType (see page 5090)

35.7.65 System.varUInt64 Constant

Pascal

```
varUInt64 = $0015;
```

C++

```
varUInt64 = 0x0015;
```

File

System

Description

This is constant System.varUInt64.

35.7.66 System.varUnknown Constant

Represents a Variant (see page 3421) that holds an unknown value.

Pascal

```
varUnknown = $000D;
```

C++

```
varUnknown = 0x000D;
```

File

System

Description

When a Variant (see page 3421) is assignned a value that is not recognised, it is set to the varUnknown type.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.67 System.varUString Constant

Pascal

```
varUString = $0102;
```

C++

```
varUString = 0x0102;
```

File

System

Description

This is constant System.varUString.

35.7.68 System.varVariant Constant

Represents a Variant (see page 3421) data type.

Pascal

```
varVariant = $000C;
```

C++

```
varVariant = 0x000C;
```

File

System

Description

varVariant is principally used when using VarArrayCreate to create an array of Variants (see page 4973). It determines that such an arrray has unspecified Variant (see page 3421) types. Use such an array when element contents may differ.

See Also

TVarType (see page 3801)

VarArrayCreate (see page 5059)

VarType (see page 5090)

35.7.69 System.varWord Constant

Represents a Variant (see page 3421) that holds Word data.

Pascal

```
varWord = $0012;
```

C++

```
varWord = 0x0012;
```

File

System

Description

When a Variant (see page 3421) is assigned an integer value greater in value than a Byte type can hold, but less than an LongWord type can hold, it will be set to a varWord Variant (see page 3421) type. varWord represents a Variant (see page 3421) variable that currently holds a Word data value.

Use VarType to determine the variant type.

See Also

TVarType (see page 3801)

VarAsType (see page 5073)

VarType (see page 5090)

35.7.70 System.vmtAddRef Constant

Pascal

```
vmtAddRef = 4 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtAddRef = 4 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtAddRef.

35.7.71 System.vmtAfterConstruction Constant

Pascal

```
vmtAfterConstruction = -28 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtAfterConstruction = -28 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtAfterConstruction.

35.7.72 System.vmtAutoTable Constant

Pascal

```
vmtAutoTable = -80;
```

C++

```
vmtAutoTable = -80;
```

File

System

Description

This is constant System.vmtAutoTable.

35.7.73 System.vmtBeforeDestruction Constant

Pascal

```
vmtBeforeDestruction = -24 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtBeforeDestruction = -24 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtBeforeDestruction.

35.7.74 System.vmtClassName Constant

Pascal

```
vmtClassName = -56;
```

C++

```
vmtClassName = -56;
```

File

System

Description

This is constant System.vmtClassName.

35.7.75 System.vmtCreateObject Constant

Pascal

```
vmtCreateObject = 12 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtCreateObject = 12 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtCreateObject.

35.7.76 System.vmtDefaultHandler Constant

Pascal

```
vmtDefaultHandler = -16 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtDefaultHandler = -16 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtDefaultHandler.

35.7.77 System.vmtDestroy Constant

Pascal

```
vmtDestroy = -4 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtDestroy = -4 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtDestroy.

35.7.78 System.vmtDispatch Constant

Pascal

```
vmtDispatch = -20 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtDispatch = -20 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtDispatch.

35.7.79 System.vmtDynamicTable Constant

Pascal

```
vmtDynamicTable = -60;
```

C++

```
vmtDynamicTable = -60;
```

File

System

Description

This is constant System.vmtDynamicTable.

35.7.80 System.vmtEquals Constant

Pascal

```
vmtEquals = -44 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtEquals = -44 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtEquals.

35.7.81 System.vmtFieldTable Constant

Pascal

```
vmtFieldTable = -68;
```

C++

```
vmtFieldTable = -68;
```

File

System

Description

This is constant System.vmtFieldTable.

35.7.82 System.vmtFreeInstance Constant

Pascal

```
vmtFreeInstance = -8 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtFreeInstance = -8 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtFreeInstance.

35.7.83 System.vmtGetHashCode Constant

Pascal

```
vmtGetHashCode = -40 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtGetHashCode = -40 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtGetHashCode.

35.7.84 System.vmtInitTable Constant

Pascal

```
vmtInitTable = -76;
```

C++

```
vmtInitTable = -76;
```

File

System

Description

This is constant System.vmtInitTable.

35.7.85 System.vmtInstanceSize Constant

Pascal

```
vmtInstanceSize = -52;
```

C++

```
vmtInstanceSize = -52;
```

File

System

Description

This is constant System.vmtInstanceSize.

35.7.86 System.vmtIntfTable Constant

Pascal

```
vmtIntfTable = -84;
```

C++

```
vmtIntfTable = -84;
```

File

System

Description

This is constant System.vmtIntfTable.

35.7.87 System.vmtMethodTable Constant

Pascal

```
vmtMethodTable = -64;
```

C++

```
vmtMethodTable = -64;
```

File

System

Description

This is constant System.vmtMethodTable.

35.7.88 System.vmtNewInstance Constant

Pascal

```
vmtNewInstance = -12 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtNewInstance = -12 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtNewInstance.

35.7.89 System.vmtParent Constant

Pascal

```
vmtParent = -48;
```

C++

```
vmtParent = -48;
```

File

System

Description

This is constant System.vmtParent.

35.7.90 System.vmtQueryInterface Constant

Pascal

```
vmtQueryInterface = 0 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtQueryInterface = 0 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtQueryInterface.

35.7.91 System.vmtRelease Constant

Pascal

```
vmtRelease = 8 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtRelease = 8 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtRelease.

35.7.92 System.vmtSafeCallException Constant

Pascal

```
vmtSafeCallException = -32 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtSafeCallException = -32 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtSafeCallException.

35.7.93 System.vmtSelfPtr Constant

Pascal

```
vmtSelfPtr = -88;
```

C++

```
vmtSelfPtr = -88;
```

File

System

Description

This is constant System.vmtSelfPtr.

35.7.94 System.vmtToString Constant

Pascal

```
vmtToString = -36 deprecated 'Use VMTOFFSET in asm code';
```

C++

```
vmtToString = -36 deprecated 'Use VMTOFFSET in asm code';
```

File

System

Description

This is constant System.vmtToString.

35.7.95 System.vmtTypeInfo Constant

Pascal

```
vmtTypeInfo = -72;
```

C++

```
vmtTypeInfo = -72;
```

File

System

Description

This is constant System.vmtTypeInfo.

35.7.96 System.vtAnsiString Constant

vtAnsiString defines an AnsiString TVarData (see page 3740) VType value.

Pascal

```
vtAnsiString = 11;
```

C++

```
vtAnsiString = 11;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.97 System.vtBoolean Constant

vtBoolean defines a Boolean TVarData (see page 3740) VType value.

Pascal

```
vtBoolean = 1;
```

C++

```
vtBoolean = 1;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.98 System.vtChar Constant

vtChar defines a Char TVarData (see page 3740) VType value.

Pascal

```
vtChar = 2;
```

C++

```
vtChar = 2;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.99 System.vtClass Constant

vtClass defines a TClass (see page 3785) TVarData (see page 3740) VType value.

Pascal

```
vtClass = 8;
```

C++

```
vtClass = 8;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.100 System.vtCurrency Constant

vtCurrency defines a Currency (see page 3235) TVarData (see page 3740) VType value.

Pascal

```
vtCurrency = 12;
```

C++

```
vtCurrency = 12;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.101 System.vtExtended Constant

vtExtended defines an Extended TVarData (see page 3740) VType value.

Pascal

```
vtExtended = 3;
```

C++

```
vtExtended = 3;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows,

this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.102 System.vtInt64 Constant

vtInt64 defines an Int64 TVarData (see page 3740) VType value.

Pascal

```
vtInt64 = 16;
```

C++

```
vtInt64 = 16;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.103 System.vtInteger Constant

vtInteger defines an Integer TVarData (see page 3740) VType value.

Pascal

```
vtInteger = 0;
```

C++

```
vtInteger = 0;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.104 System.vtInterface Constant

vtInterface defines an Interface pointer TVarData (see page 3740) VType value.

Pascal

```
vtInterface = 14;
```

C++

```
vtInterface = 14;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.105 System.vtObject Constant

vtObject defines a TObject (see page 3354) TVarData (see page 3740) VType value.

Pascal

```
vtObject = 7;
```

C++

```
vtObject = 7;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.106 System.vtPChar Constant

vtPChar defines a PChar TVarData (see page 3740) VType value.

Pascal

```
vtPChar = 6;
```

C++

```
vtPChar = 6;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.107 System.vtPointer Constant

vtPointer defines a Pointer TVarData (see page 3740) VType value.

Pascal

```
vtPointer = 5;
```

C++

```
vtPointer = 5;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.108 System.vtPWideChar Constant

vtPWideChar defines a PWideChar TVarData (see page 3740) VType value.

Pascal

```
vtPWideChar = 10;
```

C++

```
vtPWideChar = 10;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.109 System.vtString Constant

vtString defines a String TVarData (see page 3740) VType value.

Pascal

```
vtString = 4;
```

C++

```
vtString = 4;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.110 System.vtUnicodeString Constant

Pascal

```
vtUnicodeString = 17;
```

C++

```
vtUnicodeString = 17;
```

File

System

Description

This is constant System.vtUnicodeString.

35.7.111 System.vtVariant Constant

vtString (see page 3889) defines a Variant (see page 3421) TVarData (see page 3740) VType value.

Pascal

```
vtVariant = 13;
```

C++

```
vtVariant = 13;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.112 System.vtWideChar Constant

vtWideChar defines a WideChar TVarData (see page 3740) VType value.

Pascal

```
vtWideChar = 9;
```

C++

```
vtWideChar = 9;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.7.113 System.vtWideString Constant

vtWideString defines a WideString (see page 3467) TVarData (see page 3740) VType value.

Pascal

```
vtWideString = 15;
```

C++

```
vtWideString = 15;
```

File

System

Description

The TVarData (see page 3740) type represents the internal structure of a Variant (see page 3421) variable (on Windows, this is identical to the Variant (see page 3421) type used by COM and the Win32 API).

TVarData.VType defines the type of Variant (see page 3421) it holds.

See Also

TVarData (see page 3740)

System::TVarRec (see page 3747)

35.8 Macros

The following table lists macros in this documentation.

Macros

Name	Description
USE (see page 3892)	This is macro System::USE.
USEADDITIONALFILES (see page 3892)	This is macro System::USEADDITIONALFILES.
USEASM (see page 3893)	This is macro System::USEASM.
USEDATAMODULE (see page 3893)	This is macro System::USEDATAMODULE.
USEDATAMODULENS (see page 3893)	This is macro System::USEDATAMODULENS.
USEDEF (see page 3894)	This is macro System::USEDEF.
USEFILE (see page 3894)	This is macro System::USEFILE.
USEFORM (see page 3895)	This is macro System::USEFORM.
USEFORMNS (see page 3895)	This is macro System::USEFORMNS.
USEFORMRES (see page 3895)	This is macro System::USEFORMRES.
USEIDL (see page 3896)	This is macro System::USEIDL.
USELIB (see page 3896)	This is macro System::USELIB.
USEOBJ (see page 3896)	This is macro System::USEOBJ.
USEPACKAGE (see page 3897)	This is macro System::USEPACKAGE.
USERC (see page 3897)	This is macro System::USERC.
USERES (see page 3897)	This is macro System::USERES.
USETLB (see page 3898)	This is macro System::USETLB.
USEUNIT (see page 3898)	This is macro System::USEUNIT.

35.8.1 System::USE Macro

Pascal

USE

C++

```
#define USE(FileName, ContainerId) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USE.

35.8.2 System::USEADDITIONALFILES Macro

Pascal

USEADDITIONALFILES

C++

```
#define USEADDITIONALFILES(Exts) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEADDITIONALFILES.

35.8.3 System::USEASM Macro

Pascal

USEASM

C++

```
#define USEASM(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEASM.

35.8.4 System::USEDATAMODULE Macro

Pascal

USEDATAMODULE

C++

```
#define USEDATAMODULE(FileName, DataModuleName) \
    class DELPHICLASS T##DataModuleName;       \
    extern PACKAGE T##DataModuleName *DataModuleName;
```

File

sysclass.h

Description

This is macro System::USEDATAMODULE.

35.8.5 System::USEDATAMODULENS Macro

Pascal

USEDATAMODULENS

C++

```
#define USEDATAMODULENS(FileName, UnitName, DataModuleName) \
    namespace UnitName {          \
    class DELPHICLASS T##DataModuleName;       \
    extern PACKAGE T##DataModuleName *DataModuleName;              \
    };                \
    using namespace UnitName
```

File

sysclass.h

Description

This is macro System::USEDATAMODULENS.

35.8.6 System::USEDEF Macro

Pascal

```
USEDEF
```

C++

```
#define USEDEF(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEDEF.

35.8.7 System::USEFILE Macro

Pascal

```
USEFILE
```

C++

```
#define USEFILE(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEFILE.

35.8.8 System::USEFORM Macro

Pascal

```
USEFORM
```

C++

```
#define USEFORM(FileName, FormName) \
    class DELPHICLASS T##FormName;       \
    extern PACKAGE T##FormName *FormName;
```

File

sysclass.h

Description

This is macro System::USEFORM.

35.8.9 System::USEFORMNS Macro

Pascal

```
USEFORMNS
```

C++

```
#define USEFORMNS(FileName, UnitName, FormName) \
    namespace UnitName {         \
    class DELPHICLASS T##FormName;       \
    extern PACKAGE T##FormName *FormName;          \
    };               \
    using namespace UnitName
```

File

sysclass.h

Description

This is macro System::USEFORMNS.

35.8.10 System::USEFORMRES Macro

Pascal

```
USEFORMRES
```

C++

```
#define USEFORMRES(FileName, FormName, AncestorName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEFORMRES.

35.8.11 System::USEIDL Macro

Pascal

```
USEIDL
```

C++

```
#define USEIDL(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEIDL.

35.8.12 System::USELIB Macro

Pascal

```
USELIB
```

C++

```
#define USELIB(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USELIB.

35.8.13 System::USEOBJ Macro

Pascal

```
USEOBJ
```

C++

```
#define USEOBJ(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEOBJ.

35.8.14 System::USEPACKAGE Macro

Pascal

USEPACKAGE

C++

```
#define USEPACKAGE(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEPACKAGE.

35.8.15 System::USERC Macro

Pascal

USERC

C++

```
#define USERC(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USERC.

35.8.16 System::USERES Macro

Pascal

USERES

C++

```
#define USERES(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USERES.

35.8.17 System::USETLB Macro

Pascal

```
USETLB
```

C++

```
#define USETLB(FileName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USETLB.

35.8.18 System::USEUNIT Macro

Pascal

```
USEUNIT
```

C++

```
#define USEUNIT(ModName) \
   extern PACKAGE DummyThatIsNeverReferenced
```

File

sysclass.h

Description

This is macro System::USEUNIT.

36 Sysutils Namespace

This is namespace Sysutils.

Topics

Name	Description
Functions (see page 3900)	The following table lists functions in this documentation.

Functions

	Name	Description
	IntToStr (see page 3900)	This is function Sysutils::IntToStr.
	IntToStr (see page 3900)	This is function Sysutils::IntToStr.

36.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	IntToStr (see page 3900)	This is function Sysutils::IntToStr.
	IntToStr (see page 3900)	This is function Sysutils::IntToStr.

36.1.1 Sysutils::IntToStr Function

Pascal

```
function IntToStr(Value: __int64): PACKAGE String;
```

C++

```
__fastcall PACKAGE String IntToStr(__int64 Value);
```

File

dstring.h

Description

This is function Sysutils::IntToStr.

36.1.2 Sysutils::IntToStr Function

Pascal

```
function IntToStr(Value: Integer): PACKAGE String;
```

C++

```
__fastcall PACKAGE String IntToStr(int Value);
```

File

dstring.h

Description

This is function Sysutils::IntToStr.

37 SysUtils Namespace

This is namespace SysUtils.

Topics

Name	Description
Classes (see page 3915)	The following table lists classes in this documentation.
Interfaces (see page 4037)	The following table lists interfaces in this documentation.
Functions (see page 4039)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4320)	The following table lists structs, records, enums in this documentation.
Types (see page 4336)	The following table lists types in this documentation.
Variables (see page 4347)	The following table lists variables in this documentation.
Constants (see page 4368)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAbort (see page 3916)	EAbort is the exception class for errors that should not display an error message dialog box.
	EAbstractError (see page 3916)	EAbstractError is the exception class for attempts to call an abstract method.
	EAccessViolation (see page 3917)	EAccessViolation is the exception class for invalid memory access errors.
	EArgumentException (see page 3918)	This is class SysUtils.EArgumentException.
	EArgumentOutOfRangeException (see page 3918)	This is class SysUtils.EArgumentOutOfRangeException.
	EAssertionFailed (see page 3918)	EAssertionFailed is the exception class for Delphi assertion failure.
	EControlC (see page 3919)	EControlC is the exception class for user interrupts.
	EConvertError (see page 3919)	EConvertError is the exception class for string and object conversion errors.
	EDivByZero (see page 3921)	EDivByZero is the exception class for integer divide-by-zero errors.
	EEncodingError (see page 3922)	This is class SysUtils.EEncodingError.
	EExternal (see page 3922)	EExternal is the exception class that captures external error information.
	EExternalException (see page 3925)	EExternalException is the exception class for invalid exception codes.
	EHeapException (see page 3926)	EHeapException is the exception class for errors related to heap-allocated memory.
	EInOutError (see page 3929)	EInOutError is the exception class for file input/output errors.
	EIntError (see page 3933)	EIntError is the base class for integer math error exceptions.
	EIntfCastError (see page 3934)	EIntfCastError is the exception class for interface casting errors.
	EIntOverflow (see page 3934)	EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.
	EInvalidCast (see page 3935)	EInvalidCast is an exception class for typecasting errors.
	EInvalidContainer (see page 3935)	This is class SysUtils.EInvalidContainer.

	EInvalidInsert (see page 3936)	This is class SysUtils.EInvalidInsert.
	EInvalidOp (see page 3936)	EInvalidOp is the exception class for undefined floating-point operations.
	EInvalidPointer (see page 3937)	EInvalidPointer is the exception class for invalid pointer operations.
	EMathError (see page 3937)	EMathError is the base exception class for floating-point math errors.
	EMonitor (see page 3938)	This is class SysUtils.EMonitor.
	EMonitorLockException (see page 3938)	This is class SysUtils.EMonitorLockException.
	ENoConstructException (see page 3939)	This is class SysUtils.ENoConstructException.
	ENoMonitorSupportException (see page 3939)	This is class SysUtils.ENoMonitorSupportException.
	EOSError (see page 3939)	EOSError is the exception class for operating system errors.
	EOutOfMemory (see page 3942)	EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.
	EOverflow (see page 3942)	EOverflow is the exception class for floating-point register overflows.
	EPackageError (see page 3943)	EPackageError is the exception class for package-related errors.
	EPrivilege (see page 3943)	EPrivilege is the exception class for processor privilege violations.
	EProgrammerNotFound (see page 3944)	This is class SysUtils.EProgrammerNotFound.
	EPropReadOnly (see page 3944)	EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.
	EPropWriteOnly (see page 3945)	EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.
	ERangeError (see page 3945)	ERangeError is the exception for Delphi range errors.
	ESafecallException (see page 3946)	ESafecallException is the exception for problems with the safecall calling convention.
	EStackOverflow (see page 3947)	EStackOverflow is the exception class for stack overflows.
	EUnderflow (see page 3947)	EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.
	EVariantError (see page 3948)	EVariantError is the exception class for errors involving Variant (see page 3421) data types.
	EWin32Error (see page 3949)	EWin32Error is the exception class for Windows errors.
	Exception (see page 3949)	Exception is the base class for all runtime exceptions.
	EZeroDivide (see page 3966)	EZeroDivide is the exception class for floating-point divide-by-zero errors.
	TBigEndianUnicodeEncoding (see page 3967)	This is class SysUtils.TBigEndianUnicodeEncoding.
	TEncoding (see page 3969)	TEncoding is a class that defines a specific encoding for a character or string.
	TLanguages (see page 3987)	TLanguages lists all available Windows locales.
	TMBCSEncoding (see page 3992)	This is class SysUtils.TMBCSEncoding.
	TMultiReadExclusiveWriteSynchronizer (see page 3997)	TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.
	TSimpleRWSync (see page 4002)	This is class SysUtils.TSimpleRWSync.
	TStringBuilder (see page 4006)	An editable array of characters whose contents has all the functionality of a string object.
	TThreadLocalCounter (see page 4023)	This is class SysUtils.TThreadLocalCounter.
	TUnicodeEncoding (see page 4024)	This is class SysUtils.TUnicodeEncoding.
	TUTF7Encoding (see page 4029)	This is class SysUtils.TUTF7Encoding.
	TUTF8Encoding (see page 4034)	This is class SysUtils.TUTF8Encoding.

Constants

Name	Description
DateDelta (see page 4369)	Specifies the difference between TDateTime (see page 3323) and TTimeStamp (see page 4334) starting values.
DefaultFalseBoolStr (see page 4370)	Specifies the default string representation of the boolean false value.
DefaultTrueBoolStr (see page 4370)	Specifies the default string representation of the boolean true value.
DriveDelim (see page 4371)	Specifies the character used to separate the Windows drive from the path in a file name.
faAnyFile (see page 4371)	Specifies any file type.
faArchive (see page 4372)	Specifies only Windows archived file types.
faDirectory (see page 4372)	Specifies only directory 'file' types.
faHidden (see page 4373)	Specifies only hidden file types.
faNormal (see page 4374)	This is constant SysUtils.faNormal.

faReadOnly (see page 4374)	Specifies only read only file types.
faSymLink (see page 4375)	Specifies only symbolic link file types.
faSysFile (see page 4375)	Specifies only system file types.
faTemporary (see page 4376)	This is constant SysUtils.faTemporary.
faVolumeID (see page 4376)	Specifies Volume ID file types.
fmOpenRead (see page 4377)	Specifies a file should be opened for reading only.
fmOpenReadWrite (see page 4378)	Specifies a file should be opened for reading and writing.
fmOpenWrite (see page 4378)	Specifies a file should be opened for writing only.
fmShareCompat (see page 4379)	Specifies file sharing that is FCB compatible.
fmShareDenyNone (see page 4380)	Specifies file sharing that does not prevent file access by other applications.
fmShareDenyRead (see page 4380)	Specifies file sharing that prevents read file access by other applications.
fmShareDenyWrite (see page 4381)	Specifies file sharing that prevents write file access by other applications.
fmShareExclusive (see page 4382)	Specifies file sharing that prevents any file access by other applications.
HoursPerDay (see page 4382)	Specifies the number of hours in a day.
MaxCurrency (see page 4383)	Defines the largest value supported for Currency (see page 3235) data types.
MaxDateTime (see page 4383)	Specifies the highest supported TDateTime (see page 3323) value.
MaxEraCount (see page 4384)	This is constant SysUtils.MaxEraCount.
MinCurrency (see page 4384)	Defines the largest negative value supported for Currency (see page 3235) data types.
MinDateTime (see page 4384)	Specifies the earliest supported TDateTime (see page 3323) value.
MinsPerDay (see page 4385)	Specifies the number of minutes in a day.
MinsPerHour (see page 4386)	Specifies the number of minutes in an hour.
MonthDays (see page 4386)	Defines the number of days in each month for normal and leap years.
MSecsPerDay (see page 4387)	Specifies the number of milliseconds in a day.
MSecsPerSec (see page 4387)	Specifies the number of milliseconds in a second.
PathDelim (see page 4388)	Specifies the character used to separate each directory name in a file path.
PathSep (see page 4388)	Specifies the character used to separate a sequence of file or directory names.
pfBCB4Produced (see page 4389)	Indicates that a package was produced by via C++.
pfConsumerBCB (see page 4389)	Indicates that C++ can be a consumer of a package.
pfConsumerCompat (see page 4390)	This is constant SysUtils.pfConsumerCompat.
pfConsumerDelphi (see page 4390)	Indicates that Delphi can be a consumer of a package.
pfConsumerMask (see page 4391)	Indicates the comsumer of a package.
pfDelphi4Produced (see page 4391)	Indicates that a package was produced by Delphi 4 or later.
pfDesignOnly (see page 4392)	Indicates that a package was compiled with {$DESIGNONLY ON}.
pfExeModule (see page 4392)	Indicates that a module is an executable.
pfIgnoreDupUnits (see page 4392)	Indicates that a package was compiled with {$DUPLICATEENABLE ON}.
pfLibraryModule (see page 4393)	Indicates that a module is a library.
pfModuleTypeMask (see page 4393)	Indicates that a package is a module.
pfNeverBuild (see page 4394)	Indicates that a package was compiled with the -z flag.
pfPackageModule (see page 4394)	Indicates that a module is a package.
pfProducerMask (see page 4395)	Indicates the producer of a package.
pfProducerUndefined (see page 4395)	Indicates that a package was produced by an unknown developmemt environment.
pfRunOnly (see page 4396)	Indicates that a package was compiled with {$RUNONLY ON}.
pfV3Produced (see page 4396)	Indicates that a package was produced by Delphi 3 or later.
SecsPerDay (see page 4397)	Specifies the number of seconds in a day.
SecsPerHour (see page 4397)	This is constant SysUtils.SecsPerHour.
SecsPerMin (see page 4398)	Specifies the number of seconds in a minute.
SwitchChars (see page 4398)	Specifies the prefix character(s) for program command line arguments.
ufImplicitUnit (see page 4399)	Indicates that a unit was implicitly imported into a package.
ufMainUnit (see page 4399)	Indicates that a unit is the main one in a project.
ufOrgWeakUnit (see page 4400)	This is constant SysUtils.ufOrgWeakUnit.
ufPackageUnit (see page 4400)	Indicates that a unit is a package.
ufWeakPackageUnit (see page 4400)	Indicates that a unit is a either a weak unit or a package unit.
ufWeakUnit (see page 4401)	Indicates that a unit is a weak unit.

UnixDateDelta (see page 4401)	Specifies the difference between TDateTime (see page 3323) and Lunix date/time origins.

Functions

	Name	Description
	Abort (see page 4047)	Raises a silent exception.
	AddExitProc (see page 4048)	Adds a procedure to the exit list.
	AddTerminateProc (see page 4048)	Adds a terminate procedure to the system list of termination procedures.
	AdjustLineBreaks (see page 4049)	Adjusts line break characters to reflect Windows or Linux style.
	AnsiCompareFileName (see page 4049)	Compares file names based on the current locale.
	AnsiCompareStr (see page 4050)	Compares strings based on the current locale with case sensitivity.
	AnsiCompareText (see page 4051)	Compares strings based on the current locale without case sensitivity.
	AnsiDequotedStr (see page 4051)	Converts a quoted string into an unquoted string
	AnsiExtractQuotedStr (see page 4052)	Converts a quoted string into an unquoted string.
	AnsiExtractQuotedStr (see page 4052)	Converts a quoted string into an unquoted string.
	AnsiLastChar (see page 4052)	Returns a pointer to the last character in a string.
	AnsiLastChar (see page 4052)	Returns a pointer to the last character in a string.
	AnsiLowerCase (see page 4053)	Returns a string that is a copy of the given string converted to lower case.
	AnsiLowerCaseFileName (see page 4053)	Converts a file name to lower case.
	AnsiPos (see page 4054)	Locates the position of a substring.
	AnsiQuotedStr (see page 4054)	Returns the quoted version of a string.
	AnsiSameStr (see page 4055)	Compares strings based on the current locale with case sensitivity.
	AnsiSameText (see page 4055)	Compares strings based on the current locale without case sensitivity.
	AnsiStrAlloc (see page 4056)	This is function SysUtils.AnsiStrAlloc.
	AnsiStrComp (see page 4056)	Compares null-terminated character strings.
	AnsiStrComp (see page 4056)	Compares null-terminated character strings.
	AnsiStrIComp (see page 4057)	Compares null terminated character strings case insensitively.
	AnsiStrIComp (see page 4057)	Compares null terminated character strings case insensitively.
	AnsiStrLastChar (see page 4057)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLastChar (see page 4057)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLComp (see page 4058)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLComp (see page 4058)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLIComp (see page 4058)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLIComp (see page 4058)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLower (see page 4059)	Converts all characters in a null-terminated string to lower case.
	AnsiStrLower (see page 4059)	Converts all characters in a null-terminated string to lower case.
	AnsiStrPos (see page 4060)	Returns a pointer to the first occurrence of a substring.
	AnsiStrPos (see page 4060)	Returns a pointer to the first occurrence of a substring.
	AnsiStrRScan (see page 4060)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrRScan (see page 4060)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrScan (see page 4061)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrScan (see page 4061)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrUpper (see page 4061)	Converts all characters in a null-terminated string to upper case.
	AnsiStrUpper (see page 4061)	Converts all characters in a null-terminated string to upper case.
	AnsiUpperCase (see page 4062)	Converts a string to upper case.
	AnsiUpperCaseFileName (see page 4062)	Converts a file name to upper case.
	AppendStr (see page 4063)	Appends a dynamically allocated string to an existing string.
	AssignStr (see page 4063)	Assigns a new dynamically allocated string to a specified pointer.
	Beep (see page 4063)	Emits a message beep.
	BoolToStr (see page 4066)	Converts a Boolean value to a string.
	ByteLength (see page 4068)	This is function SysUtils.ByteLength.
	BytesOf (see page 4068)	Converts a string into a byte array.
	BytesOf (see page 4068)	Converts a string into a byte array.

	BytesOf (see page 4068)	Converts a string into a byte array.
	BytesOf (see page 4068)	Converts a string into a byte array.
	ByteToCharIndex (see page 4069)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharIndex (see page 4069)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharLen (see page 4070)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteToCharLen (see page 4070)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteType (see page 4070)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	ByteType (see page 4070)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	CallTerminateProcs (see page 4071)	Calls all of the functions in the termination procedure list.
	ChangeFileExt (see page 4071)	Changes the extension of a file name.
	ChangeFilePath (see page 4073)	Changes the path of a file name.
	CharInSet (see page 4073)	CharInSet checks whether a given character is contained within a set.
	CharInSet (see page 4073)	CharInSet checks whether a given character is contained within a set.
	CharLength (see page 4074)	Returns number of bytes used by character.
	CharLength (see page 4074)	Returns number of bytes used by character.
	CharToByteIndex (see page 4075)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteIndex (see page 4075)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteLen (see page 4076)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToByteLen (see page 4076)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToElementIndex (see page 4076)	This is function SysUtils.CharToElementIndex.
	CharToElementIndex (see page 4076)	This is function SysUtils.CharToElementIndex.
	CharToElementLen (see page 4077)	This is function SysUtils.CharToElementLen.
	CharToElementLen (see page 4077)	This is function SysUtils.CharToElementLen.
	CheckWin32Version (see page 4077)	Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 3159).
	CompareMem (see page 4078)	Performs a binary comparison of two memory images.
	CompareStr (see page 4078)	Compares two strings case sensitively.
	CompareStr (see page 4078)	Compares two strings case sensitively.
	CompareText (see page 4079)	Compares two strings by ordinal value without case sensitivity.
	CompareText (see page 4079)	Compares two strings by ordinal value without case sensitivity.
	CreateDir (see page 4083)	Creates a new directory.
	CreateGUID (see page 4084)	Creates a globally unique identifier.
	CurrentYear (see page 4086)	Returns the current year.
	CurrToStr (see page 4086)	Formats a Currency (see page 3235) value as a string.
	CurrToStr (see page 4086)	Formats a Currency (see page 3235) value as a string.
	CurrToStrF (see page 4086)	Converts a Currency (see page 3235) value to a string, using a specified format.
	CurrToStrF (see page 4086)	Converts a Currency (see page 3235) value to a string, using a specified format.
	Date (see page 4087)	Returns the current date.
	DateTimeToFileDate (see page 4088)	Converts a TDateTime (see page 3323) value to an OS timestamp value.
	DateTimeToStr (see page 4089)	Converts a TDateTime (see page 3323) value to a string.
	DateTimeToStr (see page 4089)	Converts a TDateTime (see page 3323) value to a string.
	DateTimeToString (see page 4089)	Converts a TDateTime (see page 3323) value to a string using a specified Format (see page 4188).
	DateTimeToString (see page 4089)	Converts a TDateTime (see page 3323) value to a string using a specified Format (see page 4188).
	DateTimeToSystemTime (see page 4091)	Converts a TDateTime (see page 3323) value into the Win32 API's system time type.
	DateTimeToTimeStamp (see page 4092)	Converts a TDateTime (see page 3323) value into the corresponding TTimeStamp (see page 4334) value.
	DateToStr (see page 4092)	Converts a TDateTime (see page 3323) value to a string.
	DateToStr (see page 4092)	Converts a TDateTime (see page 3323) value to a string.
	DayOfWeek (see page 4094)	Returns the day of the week for a specified date.

	DecodeDate (see page 4095)	Returns Year, Month, and Day values for a TDateTime (see page 3323) value.
	DecodeDateFully (see page 4096)	Returns Year, Month, and Day, and Day-of-Week values for a TDateTime (see page 3323) value.
	DecodeTime (see page 4097)	Breaks a TDateTime (see page 3323) value into hours, minutes, seconds, and milliseconds.
	DelegatesEqual (see page 4098)	This is function SysUtils.DelegatesEqual.
	DeleteFile (see page 4098)	Deletes a file from disk.
	DirectoryExists (see page 4100)	Determines whether a specified directory exists.
	DiskFree (see page 4101)	Returns the number of free bytes on a specified drive.
	DiskSize (see page 4102)	Returns the size, in bytes, of a specified drive.
	DisposeStr (see page 4103)	Disposes of a string pointer that was allocated using NewStr (see page 4227).
	ElementToCharIndex (see page 4104)	This is function SysUtils.ElementToCharIndex.
	ElementToCharIndex (see page 4104)	This is function SysUtils.ElementToCharIndex.
	ElementToCharLen (see page 4104)	This is function SysUtils.ElementToCharLen.
	ElementToCharLen (see page 4105)	This is function SysUtils.ElementToCharLen.
	EncodeDate (see page 4105)	Returns a TDateTime (see page 3323) value that represents a specified Year, Month, and Day.
	EncodeTime (see page 4106)	Returns a TDateTime (see page 3323) value for a specified Hour, Min, Sec, and MSec.
	ExceptionErrorMessage (see page 4107)	Formats a standard error message.
	ExcludeTrailingBackslash (see page 4108)	Returns a path name without a trailing delimiter.
	ExcludeTrailingPathDelimiter (see page 4109)	Returns a path name without a trailing delimiter.
	ExpandFileName (see page 4109)	Returns the full path name for a relative file name.
	ExpandFileNameCase (see page 4110)	Returns the full path name for a relative file name on a case-sensitive file system.
	ExpandUNCFileName (see page 4111)	Returns the full path of a file name in UNC format, if appropriate.
	ExtractFileDir (see page 4112)	Extracts the drive and directory parts from FileName.
	ExtractFileDrive (see page 4112)	Returns the drive portion of a file name.
	ExtractFileExt (see page 4113)	Returns the extension portions of a file name.
	ExtractFileName (see page 4113)	Extracts the name and extension parts of a file name.
	ExtractFilePath (see page 4115)	Returns the drive and directory portions of a file name.
	ExtractRelativePath (see page 4117)	Returns a relative path name, relative to a specific base directory.
	ExtractShortPathName (see page 4118)	Converts a file name to the short 8.3 form.
	FileAge (see page 4119)	Returns the OS timestamp of a file.
	FileAge (see page 4119)	Returns the OS timestamp of a file.
	FileClose (see page 4121)	Closes a specified file.
	FileCreate (see page 4128)	Creates a new file.
	FileCreate (see page 4128)	Creates a new file.
	FileDateToDateTime (see page 4130)	Converts an OS timestamp value to TDateTime (see page 3323) value.
	FileExists (see page 4133)	Tests if a specified file exists.
	FileGetAttr (see page 4139)	Returns the file attributes of FileName.
	FileGetDate (see page 4142)	Returns an OS timestamp for a specified file.
	FileIsReadOnly (see page 4142)	Reports whether a file is read-only.
	FileOpen (see page 4143)	Opens a specified file using a specified access mode.
	FileRead (see page 4150)	Reads a specified number of bytes from a file.
	FileSearch (see page 4156)	Searches a specified directory path for a file.
	FileSeek (see page 4157)	Repositions read/write point.
	FileSeek (see page 4157)	Repositions read/write point.
	FileSetAttr (see page 4162)	Sets the file attributes of a specified file.
	FileSetDate (see page 4164)	Sets the OS time stamp for a specified file.
	FileSetDate (see page 4164)	Sets the OS time stamp for a specified file.
	FileSetReadOnly (see page 4165)	Sets the file permissions to read-only.
	FileWrite (see page 4166)	Writes the contents of a buffer to the current position in a file.
	FinalizePackage (see page 4172)	Finalizes the specified package (.bpl file).
	FindClose (see page 4172)	Releases memory allocated by FindFirst (see page 4175).
	FindCmdLineSwitch (see page 4174)	Determines whether a string was passed as a command line argument to the application.

	FindCmdLineSwitch (see page 4174)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 4174)	Determines whether a string was passed as a command line argument to the application.
	FindFirst (see page 4175)	Searches for the first instance of a file name with a given set of attributes in a specified directory.
	FindNext (see page 4177)	Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 4175).
	FloatToCurr (see page 4179)	Converts a floating-point value to a System::Currency (see page 3235) value.
	FloatToDateTime (see page 4179)	Converts a floating-point value to a TDateTime (see page 3323) value.
	FloatToDecimal (see page 4180)	Converts a floating-point value to a decimal representation.
	FloatToStr (see page 4180)	Converts a floating point value to a string.
	FloatToStr (see page 4180)	Converts a floating point value to a string.
	FloatToStrF (see page 4182)	Converts a floating point value to a string, using a specified Format (see page 4188), Precision, and Digits.
	FloatToStrF (see page 4182)	Converts a floating point value to a string, using a specified Format (see page 4188), Precision, and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FmtLoadStr (see page 4185)	Returns formatted output using a resourced format string.
	FmtStr (see page 4185)	Assembles a formatted string using a format string and an array of arguments.
	FmtStr (see page 4185)	Assembles a formatted string using a format string and an array of arguments.
	ForceDirectories (see page 4186)	Creates a new directory, including the creation of parent directories as needed.
	Format (see page 4188)	Returns a formatted string assembled from a format string and an array of arguments.
	Format (see page 4188)	Returns a formatted string assembled from a format string and an array of arguments.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatCurr (see page 4194)	Formats a System::Currency (see page 3235) object.
	FormatCurr (see page 4194)	Formats a System::Currency (see page 3235) object.
	FormatDateTime (see page 4195)	Formats a TDateTime (see page 3323) value.
	FormatDateTime (see page 4195)	Formats a TDateTime (see page 3323) value.
	FormatFloat (see page 4197)	Formats a floating point value.
	FormatFloat (see page 4197)	Formats a floating point value.
	FreeAndNil (see page 4201)	Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).
	GDAL (see page 4202)	Get Delphi Access Licence.
	GetCurrentDir (see page 4202)	Returns the name of the current directory.
	GetEnvironmentVariable (see page 4204)	Retrieves an environment variable value.
	GetFileVersion (see page 4205)	Returns file version.

	GetFormatSettings (see page 4206)	Resets the date and number format parameters to initial values.
	GetLocaleChar (see page 4206)	Retrieves the first character of a localisation value.
	GetLocaleFormatSettings (see page 4206)	Populates a TFormatSettings (see page 4326) data structure.
	GetLocaleStr (see page 4207)	Retrieves a localisation value.
	GetModuleName (see page 4207)	Returns the fully qualified name for a module, given its handle.
	GetPackageDescription (see page 4208)	Returns the description resource from a package.
	GetPackageInfo (see page 4209)	Enumerates all the units and required packages for a package.
	GetTime (see page 4209)	Returns the current time.
	GUIDToString (see page 4210)	Converts a class identifier GUID to a string.
	HashName (see page 4212)	This is function SysUtils.HashName.
	IncAMonth (see page 4212)	Increments date data by one month.
	IncludeTrailingBackslash (see page 4213)	Ensures path name ends with delimiter
	IncludeTrailingPathDelimiter (see page 4213)	Ensures path name ends with delimiter.
	IncMonth (see page 4214)	Returns a date shifted by a specified number of months.
	InitializePackage (see page 4215)	Validates and initializes a given package.
	InitializePackage (see page 4215)	Validates and initializes a given package.
	IntToHex (see page 4215)	Returns the hex representation of an integer.
	IntToHex (see page 4215)	Returns the hex representation of an integer.
	IntToStr (see page 4216)	Converts an integer to a string.
	IntToStr (see page 4216)	Converts an integer to a string.
	IsAssembly (see page 4217)	Determines whether a dll is an Assembly.
	IsDelimiter (see page 4218)	Indicates whether a specified character in a string matches one of a set of delimiters.
	IsEqualGUID (see page 4218)	Indicates whether two TGUID (see page 3718) values are the same.
	IsLeadChar (see page 4220)	This is function SysUtils.IsLeadChar.
	IsLeadChar (see page 4220)	This is function SysUtils.IsLeadChar.
	IsLeapYear (see page 4220)	Indicates whether a specified year is a leap year.
	IsPathDelimiter (see page 4222)	Indicates whether the byte at a specified position in a string is the path delimiter.
	IsValidIdent (see page 4222)	Determines whether a string conforms to Delphi identifier rules.
	Languages (see page 4223)	Lists the locales for which support is available.
	LastDelimiter (see page 4223)	Returns the byte index of the last character that matches any character in a specified set of delimiters.
	LoadPackage (see page 4224)	Loads a specified package.
	LoadPackage (see page 4224)	Loads a specified package.
	LoadStr (see page 4224)	Loads a string resource from the application's executable file.
	LowerCase (see page 4225)	Converts an ASCII string to lowercase.
	LowerCase (see page 4225)	Converts an ASCII string to lowercase.
	MSecsToTimeStamp (see page 4226)	Converts a specified number of milliseconds into a TTimeStamp (see page 4334) value.
	NewStr (see page 4227)	Allocates a string on the heap.
	NextCharIndex (see page 4227)	Returns the byte index of the next character.
	NextCharIndex (see page 4227)	Returns the byte index of the next character.
	Now (see page 4227)	Returns the current date and time.
	OutOfMemoryError (see page 4228)	Raises an EOutOfMemory (see page 3942) exception.
	PlatformBytesOf (see page 4229)	This is function SysUtils.PlatformBytesOf.
	PlatformStringOf (see page 4229)	This is function SysUtils.PlatformStringOf.
	QuotedStr (see page 4229)	Returns the quoted version of a string.
	RaiseLastOSError (see page 4230)	Raises an exception for the last occurring OS or system library error.
	RaiseLastOSError (see page 4230)	Raises an exception for the last occurring OS or system library error.
	RaiseLastWin32Error (see page 4231)	Raises an exception for the last occurring Win32 error.
	RCS (see page 4232)	Check for Delphi Client/Server Licence.
	RemoveDir (see page 4232)	Deletes an existing empty directory.
	RenameFile (see page 4233)	Changes a file name.
	ReplaceDate (see page 4236)	Replaces the date portion of a TDateTime (see page 3323) value with a specified date.

	ReplaceTime (see page 4236)	Replaces the time portion of a TDateTime (see page 3323) value with a specified time.
	RPR (see page 4237)	Check for Delphi Professional Licence.
	SafeLoadLibrary (see page 4238)	Safe and simple library loading.
	SameFileName (see page 4238)	Compares file names based on the current locale.
	SameStr (see page 4239)	Compares two strings by ordinal value with case sensitivity.
	SameStr (see page 4239)	Compares two strings by ordinal value with case sensitivity.
	SameText (see page 4239)	Compares two strings by ordinal value without case sensitivity.
	SameText (see page 4239)	Compares two strings by ordinal value without case sensitivity.
	SetCurrentDir (see page 4240)	Sets the current directory.
	ShowException (see page 4240)	Displays an exception message with its physical address
	Sleep (see page 4241)	Delay program execution for a specified number of milliseconds.
	StrAlloc (see page 4241)	Allocates a buffer for a null-terminated string and returns a pointer to its first character.
	StrBufSize (see page 4242)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4241).
	StrBufSize (see page 4242)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4241).
	StrByteType (see page 4242)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrByteType (see page 4242)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrCat (see page 4243)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCat (see page 4243)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCharLength (see page 4244)	Returns the length, in bytes, of a character.
	StrCharLength (see page 4244)	Returns the length, in bytes, of a character.
	StrComp (see page 4244)	Compares two strings with case sensitivity.
	StrComp (see page 4244)	Compares two strings with case sensitivity.
	StrCopy (see page 4246)	Copies a null-terminated string.
	StrCopy (see page 4246)	Copies a null-terminated string.
	StrDispose (see page 4247)	Disposes of a string.
	StrDispose (see page 4247)	Disposes of a string.
	StrECopy (see page 4248)	Copies null-terminated string.
	StrECopy (see page 4248)	Copies null-terminated string.
	StrEnd (see page 4249)	Returns a pointer to the end of a null terminated string.
	StrEnd (see page 4249)	Returns a pointer to the end of a null terminated string.
	StrFmt (see page 4250)	Formats entries in an array.
	StrFmt (see page 4250)	Formats entries in an array.
	StrFmt (see page 4250)	Formats entries in an array.
	StrFmt (see page 4250)	Formats entries in an array.
	StrIComp (see page 4251)	Compares two strings without case sensitivity.
	StrIComp (see page 4251)	Compares two strings without case sensitivity.
	StringOf (see page 4253)	This is function SysUtils.StringOf.
	StringReplace (see page 4253)	Replaces occurrences of a substring within a string.
	StringToGUID (see page 4254)	Converts a string to a GUID.
	StrLCat (see page 4255)	Appends up to a specified maximum number of characters to string.
	StrLCat (see page 4255)	Appends up to a specified maximum number of characters to string.
	StrLComp (see page 4257)	Compares up to a specified maximum number of characters in two strings.
	StrLComp (see page 4257)	Compares up to a specified maximum number of characters in two strings.
	StrLCopy (see page 4258)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLCopy (see page 4258)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLen (see page 4260)	Returns number of characters in a string excluding the null terminator.
	StrLen (see page 4260)	Returns number of characters in a string excluding the null terminator.
	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.

	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.
	StrLIComp (see page 4262)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLIComp (see page 4262)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLower (see page 4264)	Converts a string to lowercase.
	StrLower (see page 4264)	Converts a string to lowercase.
	StrMove (see page 4265)	Copies specified number of characters to string.
	StrMove (see page 4265)	Copies specified number of characters to string.
	StrNew (see page 4266)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNew (see page 4266)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNextChar (see page 4268)	Returns pointer to following character.
	StrNextChar (see page 4268)	Returns pointer to following character.
	StrPas (see page 4268)	Converts null-terminated string to an AnsiString (long string).
	StrPas (see page 4269)	Converts null-terminated string to an AnsiString (long string).
	StrPCopy (see page 4271)	Copies an AnsiString (long string) to a null-terminated string.
	StrPCopy (see page 4271)	Copies an AnsiString (long string) to a null-terminated string.
	StrPLCopy (see page 4272)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPLCopy (see page 4272)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPos (see page 4272)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrPos (see page 4273)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrRScan (see page 4274)	Returns a pointer to the last occurrence of a specified character in a string.
	StrRScan (see page 4274)	Returns a pointer to the last occurrence of a specified character in a string.
	StrScan (see page 4275)	Returns a pointer to first occurrence of a specified character in a string.
	StrScan (see page 4275)	Returns a pointer to first occurrence of a specified character in a string.
	StrToBool (see page 4276)	Converts a string to a Boolean value.
	StrToBoolDef (see page 4277)	Converts a string to a Boolean value, with error default.
	StrToCurr (see page 4278)	Converts a string to a System::Currency (see page 3235) value.
	StrToCurr (see page 4278)	Converts a string to a System::Currency (see page 3235) value.
	StrToCurrDef (see page 4278)	Converts a string to a System::Currency (see page 3235) value, with error default.
	StrToCurrDef (see page 4278)	Converts a string to a System::Currency (see page 3235) value, with error default.
	StrToDate (see page 4279)	Converts a string to a TDateTime (see page 3323) value.
	StrToDate (see page 4279)	Converts a string to a TDateTime (see page 3323) value.
	StrToDateDef (see page 4280)	Converts a string to a TDateTime (see page 3323) value, with error default.
	StrToDateDef (see page 4280)	Converts a string to a TDateTime (see page 3323) value, with error default.
	StrToDateTime (see page 4281)	Converts a string to a TDateTime (see page 3323) value.
	StrToDateTime (see page 4281)	Converts a string to a TDateTime (see page 3323) value.
	StrToDateTimeDef (see page 4282)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrToDateTimeDef (see page 4282)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrToFloat (see page 4282)	Converts a given string to a floating-point value.
	StrToFloat (see page 4282)	Converts a given string to a floating-point value.
	StrToFloatDef (see page 4286)	Converts a given string to a floating-point value with error default.
	StrToFloatDef (see page 4286)	Converts a given string to a floating-point value with error default.
	StrToInt (see page 4286)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64 (see page 4287)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64Def (see page 4288)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToIntDef (see page 4288)	Converts a string that represents an integer (decimal or hex notation) to a number with error default.
	StrToTime (see page 4291)	Converts a string to a TDateTime (see page 3323) value.
	StrToTime (see page 4291)	Converts a string to a TDateTime (see page 3323) value.

	StrToTimeDef (see page 4292)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrToTimeDef (see page 4292)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrUpper (see page 4292)	Returns a string in upper case.
	StrUpper (see page 4292)	Returns a string in upper case.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	SysErrorMessage (see page 4294)	Converts OS error codes into strings.
	SystemTimeToDateTime (see page 4295)	Converts a system time value into a TDateTime (see page 3323) value.
	TextPos (see page 4296)	This is function SysUtils.TextPos.
	TextPos (see page 4296)	This is function SysUtils.TextPos.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	Time (see page 4297)	Returns the current time.
	TimeStampToDateTime (see page 4298)	Converts a TTimeStamp (see page 4334) value into the corresponding TDateTime (see page 3323) value.
	TimeStampToMSecs (see page 4299)	Converts a TTimeStamp (see page 4334) value into an absolute number of milliseconds.
	TimeToStr (see page 4299)	Returns a string that represents a TDateTime (see page 3323) value.
	TimeToStr (see page 4299)	Returns a string that represents a TDateTime (see page 3323) value.
	Trim (see page 4301)	Trims leading and trailing spaces and control characters from a string.
	TrimLeft (see page 4301)	Trims leading spaces and control characters from a string.
	TrimRight (see page 4302)	Trims trailing spaces and control characters from a string.
	TryEncodeDate (see page 4302)	Creates a TDateTime (see page 3323) value that represents a specified Year, Month, and Day.
	TryEncodeTime (see page 4303)	Creates a TDateTime (see page 3323) value for a specified Hour, Min, Sec, and MSec.
	TryFloatToCurr (see page 4304)	Converts a floating-point value to a System::Currency (see page 3235) value.
	TryFloatToDateTime (see page 4304)	Converts a floating-point value to a TDateTime (see page 3323) value.
	TryStrToBool (see page 4305)	Converts a string to a Boolean value, with a Boolean success code.
	TryStrToCurr (see page 4305)	Converts a string to a System::Currency (see page 3235) value, with a Boolean success code.
	TryStrToCurr (see page 4305)	Converts a string to a System::Currency (see page 3235) value, with a Boolean success code.
	TryStrToDate (see page 4306)	Converts a string to a TDateTime (see page 3323) value, with a Boolean success code.
	TryStrToDate (see page 4306)	Converts a string to a TDateTime (see page 3323) value, with a Boolean success code.
	TryStrToDateTime (see page 4307)	Converts a string to a TDateTime (see page 3323) value with a Boolean success code.
	TryStrToDateTime (see page 4307)	Converts a string to a TDateTime (see page 3323) value with a Boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToInt (see page 4308)	Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.
	TryStrToInt64 (see page 4309)	Converts a string that represents an integer (decimal or hex notation) to a number.
	TryStrToTime (see page 4309)	Converts a string to a TDateTime (see page 3323) value with an error default.
	TryStrToTime (see page 4309)	Converts a string to a TDateTime (see page 3323) value with an error default.

	UIntToStr (see page 4310)	This is function SysUtils.UIntToStr.
	UIntToStr (see page 4310)	This is function SysUtils.UIntToStr.
	UnloadPackage (see page 4310)	Unloads a package.
	UpperCase (see page 4311)	Returns a copy of a string in uppercase.
	UpperCase (see page 4311)	Returns a copy of a string in uppercase.
	WideBytesOf (see page 4312)	Converts an Unicode string to an array of bytes.
	WideCompareStr (see page 4313)	Compares Unicode strings based on the current locale with case sensitivity.
	WideCompareText (see page 4313)	Compares Unicode strings based on the current locale without case sensitivity.
	WideFmtStr (see page 4314)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFmtStr (see page 4314)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFormat (see page 4314)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormat (see page 4314)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormatBuf (see page 4315)	Formats the arguments from an array, placing the result in a buffer.
	WideFormatBuf (see page 4315)	Formats the arguments from an array, placing the result in a buffer.
	WideLowerCase (see page 4316)	Returns Unicode string converted to lower case.
	WideSameStr (see page 4316)	Compares Unicode strings based on the current locale with case sensitivity.
	WideSameText (see page 4316)	Compares Unicode strings based on the current locale without case sensitivity.
	WideStrAlloc (see page 4317)	This is function SysUtils.WideStrAlloc.
	WideStringOf (see page 4317)	This is function SysUtils.WideStringOf.
	WideUpperCase (see page 4318)	Returns Unicode string converted to upper case.
	Win32Check (see page 4318)	Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.
	WrapText (see page 4318)	Splits a string into multiple lines as its length approaches a specified size.
	WrapText (see page 4318)	Splits a string into multiple lines as its length approaches a specified size.

Interfaces

	Name	Description
	IReadWriteSync (see page 4037)	This is class SysUtils.IReadWriteSync.

Structs, Records, Enums

	Name	Description
	Int64Rec (see page 4320)	Int64Rec declares a utility record to provide access to the bytes of an Int64 value.
	LongRec (see page 4321)	LongRec declares a utility record to provide access to the bytes of a LongWord value.
	TCharSearch (see page 4321)	This is record SysUtils.TCharSearch.
	TExceptionRecord (see page 4322)	TExceptionRecord holds information about an exception in an application.
	TFilenameCaseMatch (see page 4323)	TFilenameCaseMatch indicates how close a match was found to a specified file name.
	TFloatFormat (see page 4323)	TFloatFormat is an enumerated list of formatting codes for float functions.
	TFloatRec (see page 4325)	TFloatRec declares a utility record that provides access to the attributes of a floating point number.
	TFloatValue (see page 4325)	TFloatValue is an enumeration of two basic floating point types.
	TFormatSettings (see page 4326)	TFormatSettings defines a thread-safe string formatting context.
	TLangRec (see page 4327)	TLangRec declares a record used to describe a Windows locale.
	TLocaleOptions (see page 4328)	TLocaleOptions defines a choice of dependent and independent locale options.
	TMbcsByteType (see page 4329)	TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).
	TNameType (see page 4329)	This is record SysUtils.TNameType.
	TSearchRec (see page 4330)	TSearchRec defines file information searched for by FindFirst (see page 4175) or FindNext (see page 4177).
	TSysLocale (see page 4333)	TSysLocale represents locale information.
	TThreadInfo (see page 4334)	This is record SysUtils.TThreadInfo.
	TTimeStamp (see page 4334)	TTimeStamp represents time and date values.

	WordRec (see page 4335)	WordRec declares a utility record to store high and low order bytes of a variable.

Types

Name	Description
ExceptClass (see page 4336)	ExceptClass declares an Exception (see page 3949) class.
PByteArray (see page 4337)	PByteArray declares a pointer to an array of Bytes.
PDayTable (see page 4337)	PDayTable is a pointer to a TDayTable (see page 4340) type.
PExceptionRecord (see page 4338)	PExceptionRecord is a pointer to a TExceptionRecord (see page 4322) type.
PThreadInfo (see page 4338)	This is type SysUtils.PThreadInfo.
PWordArray (see page 4338)	PWordArray is a pointer to a TWordArray (see page 4345) type.
TByteArray (see page 4339)	TByteArray declares an array of Bytes.
TBytes (see page 4339)	TBytes declares an array of Bytes.
TCharArray (see page 4340)	This is type SysUtils.TCharArray.
TDayTable (see page 4340)	TDayTable is used to hold the number of days of each month of a year.
TFileName (see page 4340)	TFileName is a string that represents a file name.
TFunc (see page 4341)	This is type SysUtils.TFunc.
TIntegerSet (see page 4341)	TIntegerSet is used to treat an integer as a set of bits.
TMREWSync (see page 4341)	TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 3997).
TPackageInfoProc (see page 4342)	TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 4209).
TPredicate (see page 4342)	This is type SysUtils.TPredicate.
TProc (see page 4343)	This is type SysUtils.TProc.
TProcedure (see page 4343)	TProcedure declares a procedure.
TReplaceFlags (see page 4343)	TReplaceFlags indicates how to perform a search-and-replace operation
TSysCharSet (see page 4344)	TSysCharSet is a set or characters.
TTerminateProc (see page 4344)	TTerminateProc is a call-back function invoked by CallTerminateProcs (see page 4071).
TValidatePackageProc (see page 4345)	TValidatePackageProc is a call-back function invoked by LoadPackage (see page 4224).
TWordArray (see page 4345)	TWordArray declares an array of Words.

Variables

Name	Description
CurrencyDecimals (see page 4348)	CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.
CurrencyFormat (see page 4348)	CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 \| 1 = 1$ | 2 = $ 1 | 3 = 1 $
CurrencyString (see page 4349)	CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.
DateSeparator (see page 4349)	DateSeparator is the character used to separate the year, month, and day parts of a date value.
DecimalSeparator (see page 4350)	DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.
EmptyAnsiStr (see page 4351)	This is variable SysUtils.EmptyAnsiStr.
EmptyStr (see page 4351)	An empty string constant.
EmptyWideStr (see page 4351)	An empty string version of the WideString (see page 3467) constant.
EraNames (see page 4352)	Defines the names of calendar eras for East Asian locales.
EraYearOffsets (see page 4352)	Controls the DateTimeToStr (see page 4089) function (for East Asian locales).
FalseBoolStrs (see page 4352)	Lists strings that can represent the boolean value false.
GetDiskFreeSpaceEx (see page 4353)	This is variable SysUtils.GetDiskFreeSpaceEx.
HexDisplayPrefix (see page 4353)	HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.
LeadBytes (see page 4354)	Lists all values that can be the first byte of a double byte character on the local system.
ListSeparator (see page 4354)	The character used to separate items in a list.

LongDateFormat (see page 4354)	LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.
LongDayNames (see page 4355)	LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 4195) causes a long day name to be substituted.
LongMonthNames (see page 4355)	LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 4195) causes a long month name to be substituted.
LongTimeFormat (see page 4356)	LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.
NegCurrFormat (see page 4356)	NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.
NullAnsiStr (see page 4357)	This is variable SysUtils.NullAnsiStr.
NullStr (see page 4357)	A pointer to an empty string.
NullWideStr (see page 4358)	A pointer to an empty wide string.
ShortDateFormat (see page 4358)	ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.
ShortDayNames (see page 4359)	ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 4195) causes a short day name to be substituted.
ShortMonthNames (see page 4359)	ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 4195) causes a short month name to be substituted.
ShortTimeFormat (see page 4360)	ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.
SysLocale (see page 4360)	SysLocale holds locale information.
ThousandSeparator (see page 4362)	ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.
TimeAMString (see page 4362)	TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.
TimePMString (see page 4362)	TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.
TimeSeparator (see page 4363)	TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.
TrueBoolStrs (see page 4363)	Lists strings that can represent the boolean value false.
TwoDigitYearCenturyWindow (see page 4364)	Determines what century is added to two-digit years when string dates are converted to numeric dates.
Win32BuildNumber (see page 4364)	Specifies which the build level of the Windows Operating System (see page 3159) operating.
Win32CSDVersion (see page 4365)	Specifies the Win32 latest installed service pack.
Win32MajorVersion (see page 4365)	Specifies which major Windows Operating System (see page 3159) version is operating.
Win32MinorVersion (see page 4366)	Specifies which minor Windows Operating System (see page 3159) version is operating.
Win32Platform (see page 4366)	Specifies the Windows Operating System (see page 3159) platform identifier.

37.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAbort (see page 3916)	EAbort is the exception class for errors that should not display an error message dialog box.
	EAbstractError (see page 3916)	EAbstractError is the exception class for attempts to call an abstract method.
	EAccessViolation (see page 3917)	EAccessViolation is the exception class for invalid memory access errors.
	EArgumentException (see page 3918)	This is class SysUtils.EArgumentException.
	EArgumentOutOfRangeException (see page 3918)	This is class SysUtils.EArgumentOutOfRangeException.
	EAssertionFailed (see page 3918)	EAssertionFailed is the exception class for Delphi assertion failure.
	EControlC (see page 3919)	EControlC is the exception class for user interrupts.
	EConvertError (see page 3919)	EConvertError is the exception class for string and object conversion errors.
	EDivByZero (see page 3921)	EDivByZero is the exception class for integer divide-by-zero errors.
	EEncodingError (see page 3922)	This is class SysUtils.EEncodingError.
	EExternal (see page 3922)	EExternal is the exception class that captures external error information.
	EExternalException (see page 3925)	EExternalException is the exception class for invalid exception codes.
	EHeapException (see page 3926)	EHeapException is the exception class for errors related to heap-allocated memory.
	EInOutError (see page 3929)	EInOutError is the exception class for file input/output errors.
	EIntError (see page 3933)	EIntError is the base class for integer math error exceptions.
	EIntfCastError (see page 3934)	EIntfCastError is the exception class for interface casting errors.
	EIntOverflow (see page 3934)	EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.
	EInvalidCast (see page 3935)	EInvalidCast is an exception class for typecasting errors.
	EInvalidContainer (see page 3935)	This is class SysUtils.EInvalidContainer.
	EInvalidInsert (see page 3936)	This is class SysUtils.EInvalidInsert.
	EInvalidOp (see page 3936)	EInvalidOp is the exception class for undefined floating-point operations.
	EInvalidPointer (see page 3937)	EInvalidPointer is the exception class for invalid pointer operations.
	EMathError (see page 3937)	EMathError is the base exception class for floating-point math errors.
	EMonitor (see page 3938)	This is class SysUtils.EMonitor.
	EMonitorLockException (see page 3938)	This is class SysUtils.EMonitorLockException.
	ENoConstructException (see page 3939)	This is class SysUtils.ENoConstructException.
	ENoMonitorSupportException (see page 3939)	This is class SysUtils.ENoMonitorSupportException.
	EOSError (see page 3939)	EOSError is the exception class for operating system errors.
	EOutOfMemory (see page 3942)	EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.
	EOverflow (see page 3942)	EOverflow is the exception class for floating-point register overflows.
	EPackageError (see page 3943)	EPackageError is the exception class for package-related errors.
	EPrivilege (see page 3943)	EPrivilege is the exception class for processor privilege violations.
	EProgrammerNotFound (see page 3944)	This is class SysUtils.EProgrammerNotFound.
	EPropReadOnly (see page 3944)	EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.
	EPropWriteOnly (see page 3945)	EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.
	ERangeError (see page 3945)	ERangeError is the exception for Delphi range errors.
	ESafecallException (see page 3946)	ESafecallException is the exception for problems with the safecall calling convention.
	EStackOverflow (see page 3947)	EStackOverflow is the exception class for stack overflows.
	EUnderflow (see page 3947)	EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.

	EVariantError (see page 3948)	EVariantError is the exception class for errors involving Variant (see page 3421) data types.
	EWin32Error (see page 3949)	EWin32Error is the exception class for Windows errors.
	Exception (see page 3949)	Exception is the base class for all runtime exceptions.
	EZeroDivide (see page 3966)	EZeroDivide is the exception class for floating-point divide-by-zero errors.
	TBigEndianUnicodeEncoding (see page 3967)	This is class SysUtils.TBigEndianUnicodeEncoding.
	TEncoding (see page 3969)	TEncoding is a class that defines a specific encoding for a character or string.
	TLanguages (see page 3987)	TLanguages lists all available Windows locales.
	TMBCSEncoding (see page 3992)	This is class SysUtils.TMBCSEncoding.
	TMultiReadExclusiveWriteSynchronizer (see page 3997)	TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.
	TSimpleRWSync (see page 4002)	This is class SysUtils.TSimpleRWSync.
	TStringBuilder (see page 4006)	An editable array of characters whose contents has all the functionality of a string object.
	TThreadLocalCounter (see page 4023)	This is class SysUtils.TThreadLocalCounter.
	TUnicodeEncoding (see page 4024)	This is class SysUtils.TUnicodeEncoding.
	TUTF7Encoding (see page 4029)	This is class SysUtils.TUTF7Encoding.
	TUTF8Encoding (see page 4034)	This is class SysUtils.TUTF8Encoding.

37.1.1 EAbort Class

EAbort is the exception class for errors that should not display an error message dialog box.

Class Hierarchy

Pascal

```
EAbort = class(Exception);
```

C++

```
class EAbort : public Exception;
```

File

SysUtils

Description

Use EAbort to raise an exception without displaying an error message in a dialog box. If applications do not trap such "silent" exceptions, the EAbort exception is passed to the HandleException method of the Application object.

The Abort (see page 4047) procedure provides a simple, standard way to raise EAbort.

See Also

Abort (see page 4047)

TApplication

Application

37.1.2 EAbstractError Class

EAbstractError is the exception class for attempts to call an abstract method.

Class Hierarchy

Pascal

```
EAbstractError = class(Exception);
```

C++

```
class EAbstractError : public Exception;
```

File

SysUtils

Description

EAbstractError is raised when an application tries to call a Delphi abstract method. It is also raised at design time when a component with Delphi abstract method is placed on a form.

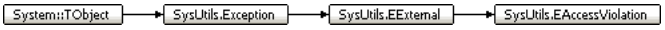
Delphi abstract methods are declared with the abstract directive and must be overridden in descendant classes before an instance object can be created.

C++ code can catch this error, but never throws it.

37.1.3 EAccessViolation Class

EAccessViolation is the exception class for invalid memory access errors.

Class Hierarchy



Pascal

```
EAccessViolation = class(EExternal);
```

C++

```
class EAccessViolation : public EExternal;
```

File

SysUtils

Description

EAccessViolation is raised when an application

Dereferences a nil (Delphi) or NULL (C++) pointer.

Writes to memory reserved for executable code.

Attempts to access a memory address for which there is no virtual memory allocated to the application.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

Note: Applications should not raise EAccessViolation directly, but should instead rely on the runtime to raise this exception.

See Also

EExternal (see page 3922)

EOutOfMemory (see page 3942)

EOutOfResources

37.1.4 EArgumentException Class

Class Hierarchy

Pascal

```
EArgumentException = class(Exception);
```

C++

```
class EArgumentException : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EArgumentException.

37.1.5 EArgumentOutOfRangeException Class

Class Hierarchy

Pascal

```
EArgumentOutOfRangeException = class(EArgumentException);
```

C++

```
class EArgumentOutOfRangeException : public EArgumentException;
```

File

SysUtils

Description

This is class SysUtils.EArgumentOutOfRangeException.

37.1.6 EAssertionFailed Class

EAssertionFailed is the exception class for Delphi assertion failure.

Class Hierarchy

Pascal

```
EAssertionFailed = class(Exception);
```

C++

```
class EAssertionFailed : public Exception;
```

File

SysUtils

Description

EAssertionFailed is raised when an assertion fails in Delphi code.

Note: To raise this exception, use the Assert (see page 3522) procedure. The exception is raised only in code compiled with the $ASSERTIONS ON directive.

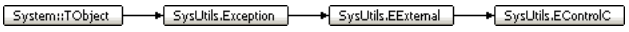
See Also

Assert (see page 3522)

37.1.7 EControlC Class

EControlC is the exception class for user interrupts.

Class Hierarchy



Pascal

```
EControlC = class(EExternal);
```

C++

```
class EControlC : public EExternal;
```

File

SysUtils

Description

EControlC is raised when the application is interrupted by the user.

Note: On Windows, a user interrupt can only occur in console applications, when the user presses Ctrl+C.

Note: On Linux, EControlC is raised when the program receives a TERM (15) signal. The TERM signal can be generated by the user typing the interrupt character (normally Ctrl+C) or by another program using the signal system call.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

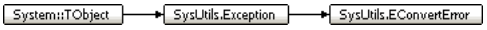
See Also

EExternal (see page 3922)

37.1.8 EConvertError Class

EConvertError is the exception class for string and object conversion errors.

Class Hierarchy



Pascal

```
EConvertError = class(Exception);
```

C++

```
class EConvertError : public Exception;
```

File

SysUtils

Description

EConvertError is raised when

An application makes an unsuccessful attempt to convert an integer, float, date, or time to a string, or to convert a string to one of these other types.

An application passes an invalid argument to a formatting routine.

An application attempts to assign one type of component derived from TPersistent to another component derived from TPersistent when such an assignment is not possible. For example, EConvertError is raised by the attempted assignment of a TButton control to a TEdit control.

In the examples below, an EConvertError exception is raised on the attempt to convert a String to a TDateTime (see page 3323) and the date in the String is invalid. The type of exception and the error message are displayed.

//Delphi example

```
const
  LF = #10;
var
  TempDate: TDateTime;
begin
try
    TempDate := StrToDateTime('99/99/1998');
except
on E: EConvertError do
      ShowMessage(E.ClassName + LF + E.Message);
```

end;

//C++ example

```
  TDateTime TempDate;
try
  {
    TempDate = StrToDateTime("99/99/1998");
  }
catch (EConvertError &E)
  {
    ShowMessage(AnsiString(E.ClassName()) + "\n" + AnsiString(E.Message));
  }
```

See Also

DateTimeToStr (see page 4089)

DateTimeToString (see page 4089)

DateToStr (see page 4092)

EncodeDate (see page 4105)

EncodeTime (see page 4106)

FmtLoadStr (see page 4185)

FmtStr (see page 4185)

Format (see page 4188)

FormatBuf (see page 4193)

FormatCurr (see page 4194)

FormatDateTime (see page 4195)

FormatFloat (see page 4197)

IntToStr (see page 4216)

StrFmt (see page 4250)

StrLFmt (see page 4261)

StrToCurr (see page 4278)

StrToDate (see page 4279)

StrToDateTime (see page 4281)

StrToFloat (see page 4282)

StrToInt (see page 4286)

StrToTime (see page 4291)

TimeToStr (see page 4299)

TPersistent

37.1.9 EDivByZero Class

EDivByZero is the exception class for integer divide-by-zero errors.

Class Hierarchy

Pascal

```
EDivByZero = class(EIntError);
```

C++

```
class EDivByZero : public EIntError;
```

File

SysUtils

Description

EDivByZero is raised when an application tries to divide an integer by zero.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

Note: Floating-point divide-by-zero errors raise the EZeroDivide (see page 3966) exception.

See Also

EExternal (see page 3922)

EIntOverflow (see page 3934)

ERangeError (see page 3945)

EZeroDivide (see page 3966)

37.1.10 EEncodingError Class

Class Hierarchy

Pascal

```
EEncodingError = class(Exception);
```

C++

```
class EEncodingError : public Exception;
```

File

SysUtils

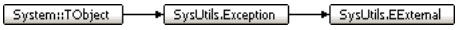
Description

This is class SysUtils.EEncodingError.

37.1.11 EExternal Class

EExternal is the exception class that captures external error information.

Class Hierarchy



Pascal

```
EExternal = class(Exception);
```

C++

```
class EExternal : public Exception;
```

File

SysUtils

Description

EExternal is the ancestor of all exception classes that represent external run time errors.

Note: Under Windows, EExternal classes represent Win32 exception codes. The ExceptionRecord (see page 3925) member points to a Win32 exception record:

//Delphi syntax

```
type
PExceptionRecord = ^TExceptionRecord;
TExceptionRecord = record
  ExceptionCode: Cardinal;
  ExceptionFlags: Cardinal;
  ExceptionRecord: PExceptionRecord;
  ExceptionAddress: Pointer;
  NumberParameters: Cardinal;
  ExceptionInformation: array[0..14] of Cardinal;
end;
```

ExceptionRecord (see page 3925): PExceptionRecord (see page 4338);

//C++ syntax

```
struct TExceptionRecord;
typedef TExceptionRecord *PExceptionRecord;
struct TExceptionRecord
{
unsigned ExceptionCode;
unsigned ExceptionFlags;
  TExceptionRecord *ExceptionRecord;
void *ExceptionAddress;
unsigned NumberParameters;
unsigned ExceptionInformation[15];
} ;
```

Note: Under Linux, EExternal classes represent various runtime errors. EExternal has three fields with exception information:

ExceptionAddress, which contains the address of the instruction that caused the error;

AccessAddress, the address that was being accessed when the error occurred;

SignalNumber, for exceptions raised by Linux signals.

See Also

EAccessViolation (see page 3917)

EIntError (see page 3933)

EMathError (see page 3937)

EPrivilege (see page 3943)

EStackOverflow (see page 3947)

37.1.11.1 EExternal Members

The following tables list the members exposed by EExternal.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	ExceptionRecord (see page 3925)	ExceptionRecord stores the external error information.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.

	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 3952)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 3960)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3380)	Returns a string containing the class name.
	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

37.1.11.2 EExternal Fields

The fields of the EExternal class are listed here.

Public Fields

	Name	Description
	ExceptionRecord (see page 3925)	ExceptionRecord stores the external error information.

37.1.11.2.1 EExternal.ExceptionRecord Field

ExceptionRecord stores the external error information.

Pascal

```
ExceptionRecord: PExceptionRecord platform;
```

C++

```
PExceptionRecord platform ExceptionRecord;
```

Description

ExceptionRecord is a member of Exception (see page 3949), containing information associated with the error that is being thrown.

External exceptions are thrown when the application uses external tools (such as databases, 3rd party libraries, GDI API functions). When an exception needs to be thrown, the 3rd party populates the record with the error information.

See Also

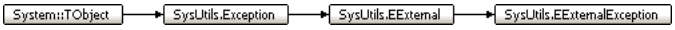
Exception (see page 3949)

TExceptionRecord (see page 4322)

37.1.12 EExternalException Class

EExternalException is the exception class for invalid exception codes.

Class Hierarchy



Pascal

```
EExternalException = class(EExternal);
```

C++

```
class EExternalException : public EExternal;
```

File

SysUtils

Description

EExternalException is raised when the run time library detects an unrecognized exception code. The run time library recognizes the following exception codes:

STATUS_INTEGER_DIVIDE_BY_ZERO

STATUS_ARRAY_BOUNDS_EXCEEDED

STATUS_FLOAT_OVERFLOW

STATUS_FLOAT_INEXACT_RESULT

STATUS_FLOAT_INVALID_OPERATION

STATUS_FLOAT_STACK_CHECK

STATUS_FLOAT_DIVIDE_BY_ZERO

STATUS_INTEGER_OVERFLOW

STATUS_FLOAT_UNDERFLOW

STATUS_FLOAT_DENORMAL_OPERAND

STATUS_ACCESS_VIOLATION

STATUS_PRIVILEGED_INSTRUCTION

STATUS_CONTROL_C_EXIT

STATUS_STACK_OVERFLOW

Any exception code that is not on the list above raises an EExternalException.

Note: As a descendant of EExternal (see page 3922), EExternalException saves the exception record in its ExceptionRecord (see page 3925) data structure. An exception record is only generated under Windows.

See Also

EExternal (see page 3922)

37.1.13 EHeapException Class

EHeapException is the exception class for errors related to heap-allocated memory.

Class Hierarchy

Pascal

```
EHeapException = class(Exception);
```

C++

```
class EHeapException : public Exception;
```

File

SysUtils

Description

EHeapException's descendants—EOutOfMemory (see page 3942) and EInvalidPointer (see page 3937)—are used to handle failed allocations of dynamic memory and invalid pointer operations.

Note: Memory for these exceptions is pre-allocated whenever an application starts and remains allocated as long as the application is running. Never raise EHeapException or its descendants directly.

See Also

EOutOfMemory (see page 3942)

EInvalidPointer (see page 3937)

37.1.13.1 EHeapException Members

The following tables list the members exposed by EHeapException.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 3929)	This is RaisingException, a member of class EHeapException.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 3952)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.

	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 3960)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3929)	Deallocates the memory occupied by an EHeapException (see page 3926) instance.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3380)	Returns a string containing the class name.
	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

37.1.13.2 EHeapException Methods

The methods of the EHeapException class are listed here.

Protected Methods

	Name	Description
	RaisingException (see page 3929)	This is RaisingException, a member of class EHeapException.

Public Methods

	Name	Description
	FreeInstance (see page 3929)	Deallocates the memory occupied by an EHeapException (see page 3926) instance.

37.1.13.2.1 EHeapException.FreeInstance Method

Deallocates the memory occupied by an EHeapException (see page 3926) instance.

Pascal

```
procedure FreeInstance; override;
```

C++

```
virtual __fastcall FreeInstance();
```

Description

FreeInstance is called internally when an application shuts down. Do not call this method directly or attempt to create or destroy instances of EHeapException (see page 3926).

37.1.13.2.2 EHeapException.RaisingException Method

Pascal

```
procedure RaisingException(P: PExceptionRecord); override;
```

C++

```
virtual __fastcall RaisingException(PExceptionRecord P);
```

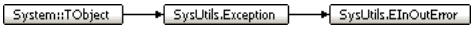
Description

This is RaisingException, a member of class EHeapException.

37.1.14 EInOutError Class

EInOutError is the exception class for file input/output errors.

Class Hierarchy



Pascal

```
EInOutError = class(Exception);
```

C++

```
class EInOutError : public Exception;
```

File

SysUtils

Description

EInOutError is raised when an file input/output error occurs, provided I/O checking is enabled.

Note: In Delphi code, use the $I+ directive to enable I/O checking. If an I/O error occurs while this directive is disabled, the application must call IOResult (see page 3602) to clear the error.

Note: In C++ programs, I/O checking is a project option.

The error code is available in the ErrorCode (see page 3933) class member. Error (see page 3560) codes are come in

ranges: 0-99 (native OS errors)

Error (see page 3560) codes in the range 0-99 represent OS error conditions, which are different for Windows and Linux. Refer to the OS documentation for complete error summaries. The SysErrorMessage (see page 4294) function returns descriptive text for OS errors.

Here are some common OS I/O errors, arranged by rough equivalents in Linux and Windows:

LinuxWindows

Linux		**Windows**	
Error (see page 3560) Code	Description	Error (see page 3560) Code	Description
2	No such file or directory	2	File not found
3	Path not found		
5	I/O Error (see page 3560)		
13	Permission denied	5	Access denied
20	Not a directory		
21	Is a directory		
32	Sharing violation		
Error (see page 3560) codes in the range 100 to 149 represent error conditions raised by CLX. Here are the CLX I/O error codes:			

Error (see page 3560) code	**Meaning**
100	End of `File`
101	Disk Full
102	File variable not assigned
103	File not open
104	File not open for input
105	File not open for output
106	Error (see page 3560) in formatted input
107	File already open

Delphi Examples:

```
{
Click the button to open a TOpenDialog, then select a file
to delete.
}

procedure TForm1.Button1Click(Sender: TObject);
var
  F: Textfile;
begin
  OpenDialog1.Title := 'Delete File';
  if OpenDialog1.Execute then
  begin
```

```
      AssignFile(F, OpenDialog1.FileName);
      try
        Reset(F);
        if MessageDlg('Erase ' + OpenDialog1.FileName + '?',
          mtConfirmation, [mbYes, mbNo], 0) = mrYes then
        begin
          CloseFile(F);
          Erase(F);
        end;
      except
        on EInOutError do
          MessageDlg('File I/O error.', mtError, [mbOk], 0);
      end;
    end;
  end;
```

See Also

IOResult (see page 3602)

EStreamError

SysErrorMessage (see page 4294)

37.1.14.1 EInOutError Members

The following tables list the members exposed by EInOutError.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	ErrorCode (see page 3933)	This is ErrorCode, a member of class EInOutError.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 3952)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.

	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 3960)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 3380)	Returns a string containing the class name.
V	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

37.1.14.2 EInOutError Fields

The fields of the EInOutError class are listed here.

Public Fields

	Name	Description
	ErrorCode (see page 3933)	This is ErrorCode, a member of class EInOutError.

37.1.14.2.1 EInOutError.ErrorCode Field

Pascal

```
ErrorCode: Integer;
```

C++

```
int ErrorCode;
```

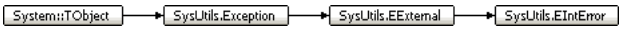
Description

This is ErrorCode, a member of class EInOutError.

37.1.15 EIntError Class

EIntError is the base class for integer math error exceptions.

Class Hierarchy



Pascal

```
EIntError = class(EExternal);
```

C++

```
class EIntError : public EExternal;
```

File

SysUtils

Description

EIntError is a generic integer math exception. Although it is never raised in the runtime library, it is the base from which other integer math exceptions descend. These are the integer math exceptions:

Exception (see page 3949)	Meaning
EDivByZero (see page 3921)	An attempt was made to divide by zero
ERangeError (see page 3945)	Number or expression out of range
EIntOverflow (see page 3934)	Integer operation overflowed

Run-time exception information is saved in fields provided by EExternal (see page 3922).

See Also

EDivByZero (see page 3921)

EExternal (see page 3922)

ERangeError (see page 3945)

EIntOverflow (see page 3934)

37.1.16 EIntfCastError Class

EIntfCastError is the exception class for interface casting errors.

Class Hierarchy

Pascal

```
EIntfCastError = class(Exception);
```

C++

```
class EIntfCastError : public Exception;
```

File

SysUtils

Description

EIntfCastError is raised when an illegal cast is attempted on an interface.

See Also

EInvalidCast (see page 3935)

37.1.17 EIntOverflow Class

EIntOverflow is the exception class for integer calculations whose results are too large to fit in the allocated register.

Class Hierarchy

Pascal

```
EIntOverflow = class(EIntError);
```

C++

```
class EIntOverflow : public EIntError;
```

File

SysUtils

Description

EIntOverflow is a occurs when data is lost because an integer result is too large to retain.

Note: In Delphi Code, EIntOverflow is raised only if overflow checking is turned on. To turn on overflow checking, include the $Q+ directive in project source code, or select Project|Options, choose the Compiler tab, and check the Overflow-checking option in the dialog box.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

Note: Floating-point overflows raise the EOverflow (see page 3942) exception.

See Also

EExternal (see page 3922)

ERangeError (see page 3945)

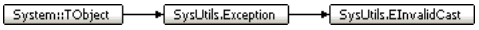
EDivByZero (see page 3921)

EOverflow (see page 3942)

37.1.18 EInvalidCast Class

EInvalidCast is an exception class for typecasting errors.

Class Hierarchy



Pascal

```
EInvalidCast = class(Exception);
```

C++

```
class EInvalidCast : public Exception;
```

File

SysUtils

Description

EInvalidCast is raised when an illegal typecast is attempted.

See Also

EIntfCastError (see page 3934)

37.1.19 EInvalidContainer Class

Class Hierarchy

Pascal

```
EInvalidContainer = class(Exception);
```

C++

```
class EInvalidContainer : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EInvalidContainer.

37.1.20 EInvalidInsert Class

Class Hierarchy

Pascal

```
EInvalidInsert = class(Exception);
```

C++

```
class EInvalidInsert : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EInvalidInsert.

37.1.21 EInvalidOp Class

EInvalidOp is the exception class for undefined floating-point operations.

Class Hierarchy

Pascal

```
EInvalidOp = class(EMathError);
```

C++

```
class EInvalidOp : public EMathError;
```

File

SysUtils

Description

EInvalidOp is raised when the CPU encounters an undefined instruction, invalid operation, or floating-point stack overflow.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

Note: Do not confuse EInvalidOp with EInvalidOperation, the exception class for invalid component operations.

See Also

EExternal (see page 3922)

EInvalidOperation

EInvalidArgument

EOverflow (see page 3942)

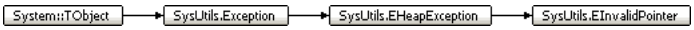
EUnderflow (see page 3947)

EZeroDivide (see page 3966)

37.1.22 EInvalidPointer Class

EInvalidPointer is the exception class for invalid pointer operations.

Class Hierarchy



Pascal

```
EInvalidPointer = class(EHeapException);
```

C++

```
class EInvalidPointer : public EHeapException;
```

File

SysUtils

Description

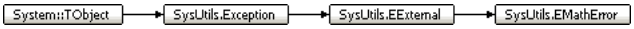
EInvalidPointer is raised when an application attempts an invalid pointer operation. For example, it can occur if an application tries to dispose of the same pointer twice, or refers to a pointer which has already been disposed of.

Note: Never raise an EInvalidPointer exception directly. EInvalidPointer is raised internally by the memory manager.

37.1.23 EMathError Class

EMathError is the base exception class for floating-point math errors.

Class Hierarchy



Pascal

```
EMathError = class(EExternal);
```

C++

```
class EMathError : public EExternal;
```

File

SysUtils

Description

EMathError is the base class for floating-point math exception classes. EMathError itself is never raised.

The following exceptions descend from EMathError:

Exception (see page 3949)	Meaning
EInvalidArgument	Parameter out of range
EInvalidOp (see page 3936)	Processor encountered an undefined instruction
EOverflow (see page 3942)	Floating-point operation produced result too large to store

EUnderflow (see page 3947)	Floating-point operation produced result with no precision
EZeroDivide (see page 3966)	Attempt to divide by zero

Run-time exception information is saved in fields provided by EExternal (see page 3922).

See Also

EExternal (see page 3922)

EInvalidArgument

EInvalidOp (see page 3936)

EOverflow (see page 3942)

EUnderflow (see page 3947)

EZeroDivide (see page 3966)

37.1.24 EMonitor Class

Class Hierarchy

Pascal

```
EMonitor = class(Exception);
```

C++

```
class EMonitor : public Exception;
```

File

SysUtils

Description

This is class SysUtils.EMonitor.

37.1.25 EMonitorLockException Class

Class Hierarchy

Pascal

```
EMonitorLockException = class(EMonitor);
```

C++

```
class EMonitorLockException : public EMonitor;
```

File

SysUtils

Description

This is class SysUtils.EMonitorLockException.

37.1.26 ENoConstructException Class

Class Hierarchy

Pascal

```
ENoConstructException = class(Exception);
```

C++

```
class ENoConstructException : public Exception;
```

File

SysUtils

Description

This is class SysUtils.ENoConstructException.

37.1.27 ENoMonitorSupportException Class

Class Hierarchy

Pascal

```
ENoMonitorSupportException = class(EMonitor);
```

C++

```
class ENoMonitorSupportException : public EMonitor;
```

File

SysUtils

Description

This is class SysUtils.ENoMonitorSupportException.

37.1.28 EOSError Class

EOSError is the exception class for operating system errors.

Class Hierarchy

Pascal

```
EOSError = class(Exception);
```

C++

```
class EOSError : public Exception;
```

File

SysUtils

Description

EOSError is raised by the RaiseLastOSError (see page 4230) procedure if the operating system returns an error code. The exception dialog contains the error number and the message associated with the error.

Note: EOSError replaces EWin32Error (see page 3949), which is now deprecated.

See Also

RaiseLastOSError (see page 4230)

37.1.28.1 EOSError Members

The following tables list the members exposed by EOSError.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	ErrorCode (see page 3942)	This is ErrorCode, a member of class EOSError.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 3952)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.

	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 3960)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 3380)	Returns a string containing the class name.
V	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

37.1.28.2 EOSError Fields

The fields of the EOSError class are listed here.

Public Fields

	Name	Description
	ErrorCode (see page 3942)	This is ErrorCode, a member of class EOSError.

37.1.28.2.1 EOSError.ErrorCode Field

Pascal

```
ErrorCode: DWORD;
```

C++

```
DWORD ErrorCode;
```

Description

This is ErrorCode, a member of class EOSError.

37.1.29 EOutOfMemory Class

EOutOfMemory is the exception class for unsuccessful attempts to allocate memory.

Class Hierarchy

Pascal

```
EOutOfMemory = class(EHeapException);
```

C++

```
class EOutOfMemory : public EHeapException;
```

File

SysUtils

Description

EOutOfMemory occurs when an application attempts to allocate dynamic memory, but there is not enough free memory in the system to meet the request.

Memory for the EOutOfMemory exception is pre-allocated whenever an application starts and remains allocated as long as the application is running.

Note: Never raise EOutOfMemory directly. Instead, call the global OutOfMemoryError (see page 4228) procedure.

See Also

EOutOfResources

OutOfMemoryError (see page 4228)

EHeapException (see page 3926)

37.1.30 EOverflow Class

EOverflow is the exception class for floating-point register overflows.

Class Hierarchy

Pascal

```
EOverflow = class(EMathError);
```

C++

```
class EOverflow : public EMathError;
```

File

SysUtils

Description

EOverflow is raised when a calculated result is too large to fit in the floating-point register allocated for it and data is therefore lost.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

Note: Integer register overflows raise the EIntOverflow (see page 3934) exception.

See Also

EExternal (see page 3922)

EInvalidArgument

EInvalidOp (see page 3936)

EUnderflow (see page 3947)

EZeroDivide (see page 3966)

EIntOverflow (see page 3934)

37.1.31 EPackageError Class

EPackageError is the exception class for package-related errors.

Class Hierarchy

Pascal

```
EPackageError = class(Exception);
```

C++

```
class EPackageError : public Exception;
```

File

SysUtils

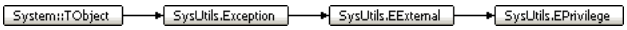
Description

EPackageError is raised at design time only, when an error occurs loading or utilizing a package.

37.1.32 EPrivilege Class

EPrivilege is the exception class for processor privilege violations.

Class Hierarchy



Pascal

```
EPrivilege = class(EExternal);
```

C++

```
class EPrivilege : public EExternal;
```

File

SysUtils

Description

EPrivilege is raised when an application tries to execute a processor instruction that is invalid for the current processor privilege level.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

See Also

EExternal (see page 3922)

37.1.33 EProgrammerNotFound Class

Class Hierarchy

Pascal

```
EProgrammerNotFound = class(Exception);
```

C++

```
class EProgrammerNotFound : public Exception;
```

File

SysUtils

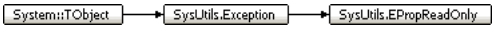
Description

This is class SysUtils.EProgrammerNotFound.

37.1.34 EPropReadOnly Class

EPropReadOnly is the exception class for invalid attempts to write to a property using OLE automation.

Class Hierarchy



Pascal

```
EPropReadOnly = class(Exception);
```

C++

```
class EPropReadOnly : public Exception;
```

File

SysUtils

Description

EPropReadOnly is raised when an application tries to write to a read-only property using OLE automation.

37.1.35 EPropWriteOnly Class

EPropWriteOnly is the exception class for invalid attempts to read a property using OLE automation.

Class Hierarchy

Pascal

```
EPropWriteOnly = class(Exception);
```

C++

```
class EPropWriteOnly : public Exception;
```

File

SysUtils

Description

EPropWriteOnly is raised when an application tries to read a write-only property using OLE automation.

37.1.36 ERangeError Class

ERangeError is the exception for Delphi range errors.

Class Hierarchy

Pascal

```
ERangeError = class(EIntError);
```

C++

```
class ERangeError : public EIntError;
```

File

SysUtils

Description

ERangeError occurs in Delphi code when range checking is enabled and an ordinal values goes outside its declared range.

Note: Range checking is not a C++ feature. However, ERangeError can occur when C++ code calls Delphi code.

Note: By default, the Delphi compiler disables range checking. The compiler can always detect obvious range errors, which prevents the following code from compiling.

//Delphi subrange example

```
var
  SmallValue: 1..3;
  SevenBits: byte;
begin;
  SmallValue := 4;  //Won't compile: constant expression out of bounds
  SevenBits := 256; //Won't compile: constant expression out of bounds
end;
```

Note: However, the compiler can't anticipate out-of-range values provided at run time:

//Delphi subrange example

```
var
  SmallValue: 1..3;
  SevenBits: byte;
  ThirtyTwoBits: Integer
begin;
  ThirtyTwoBits := 4;
  SmallValue := ThirtyTwoBits; //Assigns 4 to SmallValue (SmallValue is stored as a Byte)
  ThirtyTwoBits := 256;
  SevenBits := ThirtyTwoBits;  //Assigns 0 to ThirtyTwoBits (high-order bits discarded)
end;
```

Note: If range checking is enabled, both out-of-bounds assignments throw ERangeError exceptions. To enable range checking, use the Project option in the IDE or the $R+ directive.

Note: Range checking is considered a debugging feature. It produces bigger and slower executables.

Note: Do not confuse ERangeError with ERangeException, which relates to errors using graduated controls in VisualCLX.

See Also

EExternal (see page 3922)

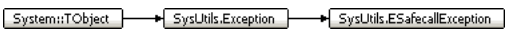
EIntOverflow (see page 3934)

EDivByZero (see page 3921)

37.1.37 ESafecallException Class

ESafecallException is the exception for problems with the safecall calling convention.

Class Hierarchy



Pascal

```
ESafecallException = class(Exception);
```

C++

```
class ESafecallException : public Exception;
```

File

SysUtils

Description

ESafecallException is raised when the safecall error handler has not been set up and a safecall routine returns a non-0 HResult, or if the safecall error handler does not raise an exception.

If this exception occurs, the Comobj unit is probably missing from the application's uses list (Delphi) or not included in the project source file (C++). You may want to consider removing the safecall calling convention from the routine that gave rise to the exception.

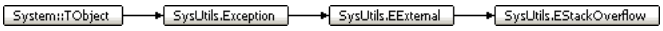
See Also

TComponent

37.1.38 EStackOverflow Class

EStackOverflow is the exception class for stack overflows.

Class Hierarchy



Pascal

```
EStackOverflow = class(EExternal);
```

C++

```
class EStackOverflow : public EExternal;
```

File

SysUtils

Description

EStackOverflow is raised when the system is unable to allocate necessary stack space. This can happen because of extremely large local variables, deeply recursive routines, or invalid machine-language code.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

See Also

EExternal (see page 3922)

37.1.39 EUnderflow Class

EUnderflow is the exception class for values that are too small to be represented with a floating-point variable.

Class Hierarchy

Pascal

```
EUnderflow = class(EMathError);
```

C++

```
class EUnderflow : public EMathError;
```

File

SysUtils

Description

When a floating-point math operation produces a value that is too small to be represented in a floating-point variable, the result ordinarily becomes zero. However, an application can change the numeric coprocessor's control word to unmask underflow hardware exceptions. In this case, EUnderflow is raised when the processor reports an underflow hardware exception. The control word is reset after the exception.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

See Also

EExternal (see page 3922)

EInvalidArgument

EInvalidOp (see page 3936)

EOverflow (see page 3942)

EZeroDivide (see page 3966)

37.1.40 EVariantError Class

EVariantError is the exception class for errors involving Variant (see page 3421) data types.

Class Hierarchy

Pascal

```
EVariantError = class(Exception);
```

C++

```
class EVariantError : public Exception;
```

File

SysUtils

Description

EVariantError is raised when a variant operation fails. EVariantError has a number of descendant classes which provide more specific information about the error.

See Also

EVariantArrayCreateError (see page 4978)

EVariantArrayLockedError (see page 4979)

EVariantBadIndexError (see page 4980)

EVariantBadVarTypeError (see page 4980)

EVariantDispatchError (see page 4981)

EVariantInvalidArgError (see page 4981)

EVariantInvalidOpError (see page 4983)

EVariantNotImplError (see page 4984)

EVariantOutOfMemoryError (see page 4984)

EVariantOverflowError (see page 4985)

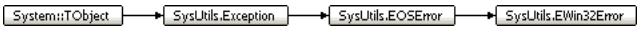
EVariantTypeCastError (see page 4986)

EVariantUnexpectedError (see page 4987)

37.1.41 EWin32Error Class

EWin32Error is the exception class for Windows errors.

Class Hierarchy



Pascal

```
EWin32Error = class(EOSError);
```

C++

```
class EWin32Error : public EOSError;
```

File

SysUtils

Description

EWin32Error is now deprecated in favor of EOSError (see page 3939).

See Also

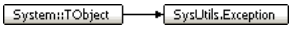
RaiseLastWin32Error (see page 4231)

EOSError (see page 3939)

37.1.42 Exception Class

Exception is the base class for all runtime exceptions.

Class Hierarchy



Pascal

```
Exception = class(TObject);
```

C++

```
class Exception : public TObject;
```

File

SysUtils

Description

Exception encapsulates the fundamental properties and methods for all exceptions. Use Exception as the base class for creating custom exceptions.

All methods introduced for the Exception object are constructors that provide alternative ways to create exception messages. Some constructors also create help context IDs. Typically, an application calls these constructors dynamically when an exception occurs.

Exception messages can be hard-coded strings, formatted strings, or strings (including formatted strings) loaded from an application resource.

Exceptions are raised when a runtime error occurs in an application, such as attempting to divide by zero. When an exception is raised, typically an exception instance displays a dialog box describing the error condition. If an application does not handle the exception condition, then the default exception handler is called. This handler also displays a dialog box with an OK button that usually permits an application to continue processing when the user clicks OK.

Note: Standard practice is to derive all exception classes from Exception, and specify a name beginning with "E", followed by an abbreviated, descriptive name.

C++ Examples:

```
/*
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
*/

#include <System.hpp>
#include <SysUtils.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Application->OnException = AppException;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::AppException(TObject *Sender, Exception *E)
{
  Application->ShowException(E);
  Application->Terminate();
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  throw(Exception("Hardware error: Divide by 0"));
}
```

37.1.42.1 Exception Members

The following tables list the members exposed by Exception.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3952)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3960)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.

	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

37.1.42.2 Exception.Create Constructor

Instantiates an instance of an exception with a simple message string.

Pascal

```
constructor Create(const Msg: string);
```

C++

```
__fastcall Exception(const UnicodeString Msg);
```

Description

Call Create to construct an exception object with a simple message string.

Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
```

```
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
{
In addition to displaying the exception message, which
happens by default, the following code shuts down the
application when an exception is not caught and handled.
AppException should be declared a method of TForm1.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnException := AppException;
end;

procedure TForm1.AppException(Sender: TObject; E: Exception);
begin
  Application.ShowException(E);
  Application.Terminate;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  raise EPasswordInvalid.Create('Incorrect password entered');
end;
```

See Also

CreateFmt (see page 3954)

CreateRes (see page 3956)

CreateResFmt (see page 3957)

Message (see page 3965)

37.1.42.3 Exception.CreateFmt Constructor

Instantiates an instance of an exception with a formatted message string.

Pascal

```
constructor CreateFmt(const Msg: string; const Args: array of const);
```

C++

```
__fastcall Exception(const UnicodeString Msg, const array of const Args);
```

Description

Call CreateFmt to construct an exception with a formatted message string. Msg is a string constant containing format specifiers

to be replaced with values from Args. Args is an array of constants containing values to format according to format specifiers in Msg, and insert into Msg. CreateFmt calls the Format (see page 4188) function to transform Msg with the values in Arg.

See Also

CreateResFmt (see page 3957)

CreateResFmtHelp (see page 3958)

Format (see page 4188)

Message (see page 3965)

37.1.42.4 Exception.CreateFmtHelp Constructor

Instantiates an instance of an exception with a formatted message string and a help-context ID.

Pascal

```
constructor CreateFmtHelp(const Msg: string; const Args: array of const; AHelpContext:
Integer);
```

C++

```
__fastcall Exception(const UnicodeString Msg, const array of const Args, int AHelpContext);
```

Description

Call CreateFmtHelp to construct an exception with a formatted message string and a help context ID. Msg is a string constant containing format specifiers to be replaced with values from Args. Args is an array of constants containing values to format according to format specifiers in Msg, and insert into Msg. CreateFmt (see page 3954) calls the Format (see page 4188) function to transform Msg with the values in Arg.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

CreateHelp (see page 3955)

CreateResFmtHelp (see page 3958)

Format (see page 4188)

HelpContext (see page 3964)

Message (see page 3965)

37.1.42.5 Exception.CreateHelp Constructor

Creates an instance of an exception with a simple message string and a help- context ID.

Pascal

```
constructor CreateHelp(const Msg: string; AHelpContext: Integer);
```

C++

```
__fastcall Exception(const UnicodeString Msg, int AHelpContext);
```

Description

Call CreateHelp to construct an exception with a simple message string and a help-context ID. Msg is the string containing the runtime error message to display in the exception dialog box. Msg can be a hard-coded string, or can be a function call that returns a string.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

HelpContext (see page 3964)

Message (see page 3965)

CreateFmtHelp (see page 3955)

CreateResFmtHelp (see page 3958)

Format (see page 4188)

37.1.42.6 CreateRes Constructor

This is the overview for the CreateRes constructor overload.

Overload List

	Name	Description
	Exception.CreateRes (Integer) (see page 3956)	Creates an instance of an exception with a simple message string that is loaded from the application's resources.
	Exception.CreateRes (PResStringRec) (see page 3956)	Creates an instance of an exception with a simple message string that is loaded from the application's resources.

37.1.42.6.1 Exception.CreateRes Constructor (Integer)

Creates an instance of an exception with a simple message string that is loaded from the application's resources.

Pascal

```
constructor CreateRes(Ident: Integer); overload;
constructor CreateRes(ResStringRec: PResStringRec); overload;
```

C++

```
__fastcall Exception(int Ident);
__fastcall Exception(PResStringRec ResStringRec);
```

Description

Call CreateRes to construct an exception with a simple message string loaded from an application's resources. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateRes creates an empty message string for the exception.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'You shouldn''t have done that';
...
Exception.Create(@sMyNewErrorMessage);
```

See Also

Message (see page 3965)

CreateResFmt (see page 3957)

CreateResFmtHelp (see page 3958)

37.1.42.7 CreateResFmt Constructor

This is the overview for the CreateResFmt constructor overload.

Overload List

	Name	Description
	Exception.CreateResFmt (Integer, array of const) (see page 3957)	Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.
	Exception.CreateResFmt (PResStringRec, array of const) (see page 3957)	Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.

37.1.42.7.1 Exception.CreateResFmt Constructor (Integer, array of const)

Creates an instance of an exception with a message string that is loaded from the application's resources and then formatted.

Pascal

```
constructor CreateResFmt(Ident: Integer; const Args: array of const); overload;
constructor CreateResFmt(ResStringRec: PResStringRec; const Args: array of const); overload;
```

C++

```
__fastcall Exception(int Ident, const array of const Args);
__fastcall Exception(PResStringRec ResStringRec, const array of const Args);
```

Description

Call CreateResFmt to construct an exception with a message string loaded from an application's resources and then formatted with additional information. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResFmt creates an empty message string for the exception.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'Illegal value: %s';
...
Exception.CreateResFmt(@sMyNewErrorMessage, [-1]);
```

Args is an array of constants containing values to:

1Format (see page 4188) according to format specifiers embedded in the string, and

2Insert (see page 3601) into the message string.

CreateResFmt calls the Format (see page 4188) function to transform the message string with the values in Arg.

See Also

Message (see page 3965)

CreateRes (see page 3956)

CreateResFmtHelp (see page 3958)

Format (see page 4188)

37.1.42.8 CreateResFmtHelp Constructor

This is the overview for the CreateResFmtHelp constructor overload.

Overload List

	Name	Description
	Exception.CreateResFmtHelp (Integer, array of const, Integer) (see page 3958)	Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources
	Exception.CreateResFmtHelp (PResStringRec, array of const, Integer) (see page 3958)	Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources

37.1.42.8.1 Exception.CreateResFmtHelp Constructor (Integer, array of const, Integer)

Creates an instance of an exception with a help-context ID and a formatted message string loaded from the application resources

Pascal

```
constructor CreateResFmtHelp(ResStringRec: PResStringRec; const Args: array of const;
AHelpContext: Integer); overload;
constructor CreateResFmtHelp(Ident: Integer; const Args: array of const; AHelpContext:
Integer); overload;
```

C++

```
__fastcall Exception(PResStringRec ResStringRec, const array of const Args, int AHelpContext);
__fastcall Exception(int Ident, const array of const Args, int AHelpContext);
```

Description

Call CreateResFmtHelp to construct an exception with a:

1Message (see page 3965) string loaded from an application's resources and formatted with additional information.

2Help-context ID.

Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file. Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResFmt (see page 3957) creates an empty message string.

ResStringRec is a pointer to a resource string. This syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'Illegal value: %s';
const IDH_MYNEWERROR: Integer = 30000;
...
Exception.CreateResFmt(@sMyNewErrorMessage, [-1], IDH_MYNEWERROR);
```

Args is an array of constants containing values to:

1Format (see page 4188) according to format specifiers embedded in the string, and

2Insert (see page 3601) into the message string.

CreateResFmtHelp calls the Format (see page 4188) function to transform the message string with the values in Arg.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

See Also

HelpContext (see page 3964)

Message (see page 3965)

CreateHelp (see page 3955)

CreateRes (see page 3956)

CreateResHelp (see page 3959)

Format (see page 4188)

37.1.42.9 CreateResHelp Constructor

This is the overview for the CreateResHelp constructor overload.

Overload List

	Name	Description
	Exception.CreateResHelp (Integer, Integer) (see page 3959)	Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.
	Exception.CreateResHelp (PResStringRec, Integer) (see page 3959)	Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.

37.1.42.9.1 Exception.CreateResHelp Constructor (Integer, Integer)

Creates an instance of an exception with a help-context ID and a simple message string that is loaded from the application's resources.

Pascal

```
constructor CreateResHelp(Ident: Integer; AHelpContext: Integer); overload;
constructor CreateResHelp(ResStringRec: PResStringRec; AHelpContext: Integer); overload;
```

C++

```
__fastcall Exception(int Ident, int AHelpContext);
__fastcall Exception(PResStringRec ResStringRec, int AHelpContext);
```

Description

Call CreateResHelp to construct an exception with a help-context ID and a simple message string loaded from an application's resources. Resources are bound into the application executable at compile time, but at design time they exist in a separate resource file.

Ident is the unique ID of the resource as specified in the resource file. If Ident is not a valid resource ID, CreateResHelp creates an empty message string for the exception.

AHelpContext is an integer that specifies the context-sensitive help ID for the exception.

ResStringRec is a pointer to a resource string. Its syntax appears as follows:

```
resourcestring sMyNewErrorMessage = 'You shouldn''t have done that';
const IDH_MYNEWERROR: Integer = 30000;
...
Exception.Create(@sMyNewErrorMessage, IDH_MYNEWERROR);
```

See Also

HelpContext (see page 3964)

Message (see page 3965)

CreateHelp (see page 3955)

CreateRes (see page 3956)

CreateResFmtHelp (see page 3958)

Format (see page 4188)

37.1.42.10 Exception.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~Exception();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

37.1.42.11 Exception Fields

The fields of the Exception class are listed here.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

37.1.42.11.1 Exception.CleanUpStackInfoProc Field

Pascal

```
class var CleanUpStackInfoProc: procedure (Info: Pointer);
```

C++

```
procedure (Info: Pointer) CleanUpStackInfoProc;
```

Description

This is CleanUpStackInfoProc, a member of class Exception.

37.1.42.11.2 Exception.GetExceptionStackInfoProc Field

Pascal

```
class var GetExceptionStackInfoProc: function (P: PExceptionRecord): Pointer;
```

C++

```
function (P: PExceptionRecord): Pointer GetExceptionStackInfoProc;
```

Description

This is GetExceptionStackInfoProc, a member of class Exception.

37.1.42.11.3 Exception.GetStackInfoStringProc Field

Pascal

```
class var GetStackInfoStringProc: function (Info: Pointer): string;
```

C++

```
function (Info: Pointer): string GetStackInfoStringProc;
```

Description

This is GetStackInfoStringProc, a member of class Exception.

37.1.42.12 Exception Methods

The methods of the Exception class are listed here.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Methods

	Name	Description
V	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 3963)	Returns a string containing the class name.

37.1.42.12.1 Exception.GetBaseException Method

Pascal

```
function GetBaseException: Exception; virtual;
```

C++

```
virtual __fastcall Exception GetBaseException();
```

Description

This is GetBaseException, a member of class Exception.

37.1.42.12.2 Exception.GetStackTrace Method

Pascal

```
function GetStackTrace: string;
```

C++

```
__fastcall UnicodeString GetStackTrace();
```

Description

This is GetStackTrace, a member of class Exception.

37.1.42.12.3 Exception.RaiseOuterException Method

Pascal

```
class procedure RaiseOuterException(E: Exception); static;
```

C++

```
static __fastcall RaiseOuterException(Exception E);
```

Description

This is RaiseOuterException, a member of class Exception.

37.1.42.12.4 Exception.RaisingException Method

Pascal

```
procedure RaisingException(P: PExceptionRecord); virtual;
```

C++

```
virtual __fastcall RaisingException(PExceptionRecord P);
```

Description

This is RaisingException, a member of class Exception.

37.1.42.12.5 Exception.SetInnerException Method

Pascal

```
procedure SetInnerException;
```

C++

```
__fastcall SetInnerException();
```

Description

This is SetInnerException, a member of class Exception.

37.1.42.12.6 Exception.SetStackInfo Method

Pascal

```
procedure SetStackInfo(AStackInfo: Pointer);
```

C++

```
__fastcall SetStackInfo(void * AStackInfo);
```

Description

This is SetStackInfo, a member of class Exception.

37.1.42.12.7 Exception.ThrowOuterException Method

Pascal

```
class procedure ThrowOuterException(E: Exception); static;
```

C++

```
static __fastcall ThrowOuterException(Exception E);
```

Description

This is ThrowOuterException, a member of class Exception.

37.1.42.12.8 Exception.ToString Method

Returns a string containing the class name.

Pascal

```
function ToString: string; override;
```

C++

```
virtual __fastcall UnicodeString ToString();
```

Description

By default, the System::TObject::ToString returns a string containing the class name of the instance that is being called.

Example

Calling System::TObject::ToString on a TButton instance returns a string containing "TButton".

Note: System::TObject::ToString is intended to be overridden in user derived classes, in order to provide consumer objects with a string representation.

See Also

System::TObject::ClassType (see page 3364)

System::TObject::ClassName (see page 3359)

System::TObject::ClassParent (see page 3361)

System::TObject::FieldAddress (see page 3372)

System::TObject::MethodAddress (see page 3378)

System::TObject::InstanceSize (see page 3377)

System::TObject::MethodName (see page 3379)

System::TObject::InheritsFrom (see page 3376)

37.1.42.13 Exception Properties

The properties of the Exception class are listed here.

Public Properties

	Name	Description
	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

37.1.42.13.1 Exception.BaseException Property

Pascal

```
property BaseException: Exception;
```

C++

```
__property Exception BaseException;
```

Description

This is BaseException, a member of class Exception.

37.1.42.13.2 Exception.HelpContext Property

Indicates the context ID number for context-sensitive online help associated with the exception.

Pascal

```
property HelpContext: Integer;
```

C++

```
__property int HelpContext;
```

Description

HelpContext provides storage for the help context ID number associated with the exception object. The context ID number determines the contents of the help screen that appears when a user presses F1 in response to an exception error message.

Note: To initialize HelpContext when the Exception (see page 3949) object is instantiated, use the appropriate constructor.

Note: In Delphi code, instantiate the Exception (see page 3949) object with CreateFmtHelp (see page 3955), CreateHelp (see page 3955), CreateResFmtHelp (see page 3958), or CreateResHelp (see page 3959).

Note: In C++ code, use a form of the constructor that takes a help context parameter.

See Also

Create (see page 3952)

CreateFmtHelp (see page 3955)

CreateHelp (see page 3955)

CreateResFmtHelp (see page 3958)

CreateResHelp (see page 3959)

37.1.42.13.3 Exception.InnerException Property

Pascal

```
property InnerException: Exception;
```

C++

```
__property Exception InnerException;
```

Description

This is InnerException, a member of class Exception.

37.1.42.13.4 Exception.Message Property

Message contains the text string to display in the exception dialog box when the exception is raised.

Pascal

```
property Message: string;
```

C++

```
__property UnicodeString Message;
```

Description

Message stores the error-message string to display when the exception is raised. All Exception (see page 3949) constructors expect a string parameter to store in Message. Message text can be hard-coded as a parameter to an Exception (see page 3949) constructor, created as dynamic parameter, or loaded from a resource file as a static or dynamically formatted parameter.

See Also

Create (see page 3952)

CreateFmt (see page 3954)

CreateFmtHelp (see page 3955)

CreateHelp (see page 3955)

CreateRes (see page 3956)

CreateResFmt (see page 3957)

CreateResFmtHelp (see page 3958)

CreateResHelp (see page 3959)

37.1.42.13.5 Exception.StackInfo Property

Pascal

```
property StackInfo: Pointer;
```

C++

```
__property void * StackInfo;
```

Description

This is StackInfo, a member of class Exception.

37.1.42.13.6 Exception.StackTrace Property

Pascal

```
property StackTrace: string;
```

C++

```
__property UnicodeString StackTrace;
```

Description

This is StackTrace, a member of class Exception.

37.1.43 EZeroDivide Class

EZeroDivide is the exception class for floating-point divide-by-zero errors.

Class Hierarchy

Pascal

```
EZeroDivide = class(EMathError);
```

C++

```
class EZeroDivide : public EMathError;
```

File

SysUtils

Description

EZeroDivide exception is raised when an application attempts to divide a floating-point value by zero.

Run-time exception information is saved in fields provided by EExternal (see page 3922).

Note: Integer divide-by-zero errors raise the EDivByZero (see page 3921) exception.

See Also

EExternal (see page 3922)

EInvalidArgument

EInvalidOp (see page 3936)

EOverflow (see page 3942)

EUnderflow (see page 3947)

EDivByZero (see page 3921)

37.1.44 TBigEndianUnicodeEncoding Class

Class Hierarchy

Pascal

```
TBigEndianUnicodeEncoding = class(TUnicodeEncoding);
```

C++

```
class TBigEndianUnicodeEncoding : public TUnicodeEncoding;
```

File

SysUtils

Description

This is class SysUtils.TBigEndianUnicodeEncoding.

37.1.44.1 TBigEndianUnicodeEncoding Members

The following tables list the members exposed by TBigEndianUnicodeEncoding.

Public Methods

	Name	Description
	Convert (see page 3974)	This is the overview for the Convert method overload.
V	Create (see page 4026)	This is Create, a member of class TUnicodeEncoding.
	FreeEncodings (see page 3975)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 3976)	Determines the encoding for a given buffer.
	GetEncoding (see page 3980)	Returns the encoding used for the code page supplied.
A	GetMaxByteCount (see page 3980)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxByteCount (see page 4028)	Calculates the maximum bytes generated to encode the specified number of characters.
A	GetMaxCharCount (see page 3980)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetMaxCharCount (see page 4028)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 3969)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 3981)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 3984)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 3985)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 3985)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 3985)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 3986)	This boolean specifies if the current encoding is of single-byte format.

	Unicode (see page 3986)	Returns an encoding for the UTF-16 format with little-endian byte order.
	UTF7 (see page 3986)	Returns an encoding for the UTF-7 format.
	UTF8 (see page 3987)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 3973)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 3974)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4026)	Returns the number of bytes generated by encoding Chars.
	GetByteCount (see page 3976)	This is the overview for the GetByteCount method overload.
	GetBytes (see page 3968)	Encodes a set of characters into a sequence of bytes.
	GetBytes (see page 3977)	This is the overview for the GetBytes method overload.
	GetCharCount (see page 4027)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetCharCount (see page 3978)	This is the overview for the GetCharCount method overload.
	GetChars (see page 3969)	Decodes a set of bytes into a set of characters.
	GetChars (see page 3979)	This is the overview for the GetChars method overload.

37.1.44.2 TBigEndianUnicodeEncoding Methods

The methods of the TBigEndianUnicodeEncoding class are listed here.

Public Methods

	Name	Description
	GetPreamble (see page 3969)	Returns a preamble in a byte array that specifies the encoding used.

Strict protected Methods

	Name	Description
	GetBytes (see page 3968)	Encodes a set of characters into a sequence of bytes.
	GetChars (see page 3969)	Decodes a set of bytes into a set of characters.

37.1.44.2.1 TBigEndianUnicodeEncoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

37.1.44.2.2 TBigEndianUnicodeEncoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

37.1.44.2.3 TBigEndianUnicodeEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

37.1.45 TEncoding Class

TEncoding is a class that defines a specific encoding for a character or string.

Class Hierarchy

Pascal

```
TEncoding = class;
```

C++

```
class TEncoding;
```

File

SysUtils

Description

Use the TEncoding class to define a specific encoding desired for a given string. For instance, the TStringStream class now includes an overloaded constructor that enables the developer to define encoding for strings that will be added to an instance of TStringStream.

To simplify cross platform developement, there are similarities between this interface and System.Text.Encoding. For that reason, links are provided to MSDN counterparts in Encoding (MSDN) class.

C++ Examples:

```
/*
This example reads any file and converts the encoding to UTF8.
This example uses the TBytesStream to read the files into a TBytes'
and also uses the auto_ptr construction to clean up created class
instances.

*/
#include <tchar.h>
#include <memory>       //for STL auto_ptr class
//---------------------------------------------------------------------------

#pragma argsused
int _tmain(int argc, _TCHAR* argv[])
{
  // Sample to convert a file of any encoding to UTF8.
  TEncoding *LEncoding = NULL;
  std::auto_ptr<TFileStream> LFileStream(new TFileStream("..\\Sample.txt", fmOpenRead));

  // Read file into buffer
  TBytes myBytes;
  std::auto_ptr<TBytesStream> myBytesStream(new TBytesStream(myBytes));
  myBytesStream->CopyFrom(LFileStream.get(), LFileStream->Size);

  // Identify encoding and convert buffer to UTF8
  int LOffset = TEncoding::GetBufferEncoding(myBytesStream->Bytes, LEncoding);
  if (LOffset == 0)
    return 1; // Unknown encoding, don't convert.
  myBytes = TEncoding::Convert(LEncoding, TEncoding::UTF8,
                               myBytesStream->Bytes,
                               LOffset, myBytesStream->Size-LOffset);
  // Create output file
  std::auto_ptr<TFileStream> DestFileStream(new TFileStream("..\\SampleUTF8.txt", fmCreate));

  // Write UTF8 byte order mark and buffer to output file
  TBytes LByteOrderMark;
  LByteOrderMark = TEncoding::UTF8->GetPreamble();

  // Grab preamble and write to destination
  DestFileStream->Write(&LByteOrderMark[0], LByteOrderMark.Length);
```

```
  // Write converted buffer
  DestFileStream->Write(&myBytes[0], myBytes.Length);
  return 0;
}
```

Delphi Examples:

```
{
This example determines the encoding of a source file and then
writes the file in any encoding chosen.  Use this project to
create files of any encoding.  This example requires two text edits
and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
  LOffset: Integer;
  LEncoding, DestEncoding: TEncoding;
  LFileStream: TFileStream;
  EncodingArray: array[0..5] of TEncoding;
begin
  LEncoding:= nil;
  EncodingArray[0]:= TEncoding.UTF8;
  EncodingArray[1]:= TEncoding.UTF7;
  EncodingArray[2]:= TEncoding.Unicode;
  EncodingArray[3]:= TEncoding.Default;
  EncodingArray[4]:= TEncoding.BigEndianUnicode;
  EncodingArray[5]:= TEncoding.ASCII;
  DestEncoding := EncodingArray[ComboBox1.ItemIndex];
  LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    // Read file into buffer
    SetLength(LBuffer, LFileStream.Size);
//    LFileStream.Read(LBuffer[0], Length(LBuffer));
    LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));

    // Identify encoding and convert buffer to UTF8
    LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
//    for I := 0 to Length(LBuffer) - 1 do
//      Memo1.Lines.Add((LBuffer[I]));
    LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
      LOffset, Length(LBuffer) - LOffset);
  finally
    LFileStream.Free;
  end;

  LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
  try
    // Write an encoding byte order mark and buffer to output file
    LByteOrderMark := DestEncoding.GetPreamble;
    LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
    LFileStream.Write(LBuffer[0], Length(LBuffer));
  finally
    LFileStream.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  StringList: TStrings;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('TEncoding.UTF8');
      Add('TEncoding.UTF7');
```

```
        Add('TEncoding.Unicode');
        Add('TEncoding.Default');
        Add('TEncoding.BigEndianUnicode');
        Add('TEncoding.ASCII');
      end;
      with ComboBox1 do begin
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

Convert (see page 3975)

FreeEncodings (see page 3975)

GetBufferEncoding (see page 3976)

GetByteCount (see page 3976)

GetBytes (see page 3977)

GetCharCount (see page 3978)

GetChars (see page 3979)

GetEncoding (see page 3980)

GetMaxByteCount (see page 3980)

GetMaxCharCount (see page 3980)

GetPreamble (see page 3981)

GetString (see page 3981)

IsStandardEncoding (see page 3984)

ASCII (see page 3985)

BigEndianUnicode (see page 3985)

Default (see page 3985)

IsSingleByte (see page 3986)

Unicode (see page 3986)

UTF7 (see page 3986)

UTF8 (see page 3987)

Encoding (MSDN)

Using TEncoding for Unicode

37.1.45.1 TEncoding Members

The following tables list the members exposed by TEncoding.

Public Methods

	Name	Description
	Convert (see page 3974)	This is the overview for the Convert method overload.
	FreeEncodings (see page 3975)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 3976)	Determines the encoding for a given buffer.
	GetEncoding (see page 3980)	Returns the encoding used for the code page supplied.
A	GetMaxByteCount (see page 3980)	Calculates the maximum bytes generated to encode the specified number of characters.
A	GetMaxCharCount (see page 3980)	Calculates the maximum number of characters generated to decode a specified number of bytes.
A	GetPreamble (see page 3981)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 3981)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 3984)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 3985)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 3985)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 3985)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 3986)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 3986)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 3986)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 3987)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 3973)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 3974)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
A	GetByteCount (see page 3976)	This is the overview for the GetByteCount method overload.
A	GetBytes (see page 3977)	This is the overview for the GetBytes method overload.
A	GetCharCount (see page 3978)	This is the overview for the GetCharCount method overload.
A	GetChars (see page 3979)	This is the overview for the GetChars method overload.

37.1.45.2 TEncoding Fields

The fields of the TEncoding class are listed here.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 3973)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 3974)	This is FMaxCharSize, a member of class TEncoding.

37.1.45.2.1 TEncoding.FIsSingleByte Field

Pascal

```
FIsSingleByte: Boolean;
```

C++

```
Boolean FIsSingleByte;
```

Description

This is FIsSingleByte, a member of class TEncoding.

37.1.45.2.2 TEncoding.FMaxCharSize Field

Pascal

```
FMaxCharSize: Integer;
```

C++

```
int FMaxCharSize;
```

Description

This is FMaxCharSize, a member of class TEncoding.

37.1.45.3 TEncoding Methods

The methods of the TEncoding class are listed here.

Public Methods

	Name	Description
	Convert (see page 3974)	This is the overview for the Convert method overload.
	FreeEncodings (see page 3975)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 3976)	Determines the encoding for a given buffer.
	GetEncoding (see page 3980)	Returns the encoding used for the code page supplied.
A	GetMaxByteCount (see page 3980)	Calculates the maximum bytes generated to encode the specified number of characters.
A	GetMaxCharCount (see page 3980)	Calculates the maximum number of characters generated to decode a specified number of bytes.
A	GetPreamble (see page 3981)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 3981)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 3984)	Reports if this instance is a standard encoding instance.

Strict protected Methods

	Name	Description
A	GetByteCount (see page 3976)	This is the overview for the GetByteCount method overload.
A	GetBytes (see page 3977)	This is the overview for the GetBytes method overload.
A	GetCharCount (see page 3978)	This is the overview for the GetCharCount method overload.
A	GetChars (see page 3979)	This is the overview for the GetChars method overload.

37.1.45.3.1 Convert Method

This is the overview for the Convert method overload.

Overload List

	Name	Description
	TEncoding.Convert (TEncoding, TEncoding, TBytes) (see page 3975)	Convert an array of bytes from one encoding to another.
	TEncoding.Convert (TEncoding, TEncoding, TBytes, Integer, Integer) (see page 3975)	Convert an array of bytes from one encoding to another.

37.1.45.3.1.1 TEncoding.Convert Method (TEncoding, TEncoding, TBytes)

Convert an array of bytes from one encoding to another.

Pascal

```
class function Convert(Source: TEncoding; Destination: TEncoding; Bytes: TBytes): TBytes;
overload;
class function Convert(Source: TEncoding; Destination: TEncoding; Bytes: TBytes; StartIndex:
Integer; Count: Integer): TBytes; overload;
```

C++

```
__fastcall TBytes Convert(TEncoding Source, TEncoding Destination, TBytes Bytes);
__fastcall TBytes Convert(TEncoding Source, TEncoding Destination, TBytes Bytes, int
StartIndex, int Count);
```

Description

TEncoding.Convert converts an array of bytes from one encoding to another.

Source refers to the encoding type of the encoding of the original bytes.

Destination refers to the encoding type that the bytes are to be converted to.

StartIndex specifies the index of the array to begin converting.

Count specifies the number of bytes to convert.

The **Return Value** is the converted array of bytes.

See Also

Encoding.Convert (MSDN)

37.1.45.3.2 TEncoding.FreeEncodings Method

Frees the internal standard encoding instances.

Pascal

```
class procedure FreeEncodings;
```

C++

```
__fastcall FreeEncodings();
```

Description

TEncoding.FreeEncodings clears and frees the internal standard encodings returned by the properties ASCII (see page 3985), Default (see page 3985), UTF8 (see page 3987), Unicode (see page 3986) and BigEndianUnicode (see page 3985). This method is for internal use only, do not call directly.

See Also

ASCII (see page 3985)

Default (see page 3985)

UTF8 (see page 3987)

Unicode (see page 3986)

BigEndianUnicode (see page 3985)

37.1.45.3.3 TEncoding.GetBufferEncoding Method

Determines the encoding for a given buffer.

Pascal

```
class function GetBufferEncoding(const Buffer: TBytes; var AEncoding: TEncoding): Integer;
static;
```

C++

```
static __fastcall int GetBufferEncoding(const TBytes Buffer, TEncoding AEncoding);
```

Description

GetBufferEncoding examines the Buffer parameter and determines its encoding by looking for known byte order marks (BOM). If the buffer doesn't contain a byte order mark for one of the standard encodings, the Default (see page 3985) standard encoding is returned.

Buffer refers to a TBytes (see page 4339) instance that will be examined.

AEncoding refers to an instance of TEncoding (see page 3969) that will be filled out with the encoding type of the buffer.

The **Return Value** of GetBufferEncoding is the length of the preamble in the Buffer of bytes.

Note: The `AEncoding` parameter should have the value NIL, otherwise its value is used to detect the encoding.

37.1.45.3.4 GetByteCount Method

This is the overview for the GetByteCount method overload.

Overload List

	Name	Description
	TEncoding.GetByteCount (PChar, Integer) (see page 3976)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (string) (see page 3976)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (string, Integer, Integer) (see page 3976)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (TCharArray) (see page 3976)	Returns the number of bytes generated by encoding Chars.
	TEncoding.GetByteCount (TCharArray, Integer, Integer) (see page 3976)	Returns the number of bytes generated by encoding Chars.

37.1.45.3.4.1 TEncoding.GetByteCount Method (PChar, Integer)

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; virtual; abstract; overload;
function GetByteCount(const Chars: TCharArray): Integer; overload;
function GetByteCount(const Chars: TCharArray; CharIndex: Integer; CharCount: Integer):
Integer; overload;
function GetByteCount(const S: string): Integer; overload;
function GetByteCount(const S: string; CharIndex: Integer; CharCount: Integer): Integer;
overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount) = 0;
__fastcall int GetByteCount(const TCharArray Chars);
__fastcall int GetByteCount(const TCharArray Chars, int CharIndex, int CharCount);
__fastcall int GetByteCount(const UnicodeString S);
__fastcall int GetByteCount(const UnicodeString S, int CharIndex, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString (see page 3381) from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

37.1.45.3.5 GetBytes Method

This is the overview for the GetBytes method overload.

Overload List

	Name	Description
	TEncoding.GetBytes (PChar, Integer, PByte, Integer) (see page 3977)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (string) (see page 3977)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (string, Integer, Integer, TBytes, Integer) (see page 3977)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (TCharArray) (see page 3977)	Encodes a set of characters into a sequence of bytes.
	TEncoding.GetBytes (TCharArray, Integer, Integer, TBytes, Integer) (see page 3977)	Encodes a set of characters into a sequence of bytes.

37.1.45.3.5.1 TEncoding.GetBytes Method (PChar, Integer, PByte, Integer)

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; virtual; abstract; overload;
function GetBytes(const Chars: TCharArray): TBytes; overload;
function GetBytes(const Chars: TCharArray; CharIndex: Integer; CharCount: Integer; var Bytes:
TBytes; ByteIndex: Integer): Integer; overload;
function GetBytes(const S: string): TBytes; overload;
function GetBytes(const S: string; CharIndex: Integer; CharCount: Integer; var Bytes: TBytes;
ByteIndex: Integer): Integer; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount)
= 0;
__fastcall TBytes GetBytes(const TCharArray Chars);
__fastcall int GetBytes(const TCharArray Chars, int CharIndex, int CharCount, TBytes Bytes,
int ByteIndex);
__fastcall TBytes GetBytes(const UnicodeString S);
__fastcall int GetBytes(const UnicodeString S, int CharIndex, int CharCount, TBytes Bytes, int
ByteIndex);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString (see page 3381) to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes (see page 4339) sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

37.1.45.3.6 GetCharCount Method

This is the overview for the GetCharCount method overload.

Overload List

	Name	Description
	TEncoding.GetCharCount (PByte, Integer) (see page 3978)	Calcuates a number of characters generated by decoding a sequence of bytes.
	TEncoding.GetCharCount (TBytes) (see page 3978)	Calcuates a number of characters generated by decoding a sequence of bytes.
	TEncoding.GetCharCount (TBytes, Integer, Integer) (see page 3978)	Calcuates a number of characters generated by decoding a sequence of bytes.

37.1.45.3.6.1 TEncoding.GetCharCount Method (PByte, Integer)

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; virtual; abstract; overload;
function GetCharCount(const Bytes: TBytes): Integer; overload;
function GetCharCount(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer): Integer;
overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount) = 0;
__fastcall int GetCharCount(const TBytes Bytes);
__fastcall int GetCharCount(const TBytes Bytes, int ByteIndex, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

37.1.45.3.7 GetChars Method

This is the overview for the GetChars method overload.

Overload List

	Name	Description
	TEncoding.GetChars (PByte, Integer, PChar, Integer) (see page 3979)	Decodes a set of bytes into a set of characters.
	TEncoding.GetChars (TBytes) (see page 3979)	Decodes a set of bytes into a set of characters.
	TEncoding.GetChars (TBytes, Integer, Integer) (see page 3979)	Decodes a set of bytes into a set of characters.
	TEncoding.GetChars (TBytes, Integer, Integer, TCharArray, Integer) (see page 3979)	Decodes a set of bytes into a set of characters.

37.1.45.3.7.1 TEncoding.GetChars Method (PByte, Integer, PChar, Integer)

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; virtual; abstract; overload;
function GetChars(const Bytes: TBytes): TCharArray; overload;
function GetChars(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer): TCharArray;
overload;
function GetChars(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer; var Chars:
TCharArray; CharIndex: Integer): Integer; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount)
= 0;
__fastcall TCharArray GetChars(const TBytes Bytes);
__fastcall TCharArray GetChars(const TBytes Bytes, int ByteIndex, int ByteCount);
__fastcall int GetChars(const TBytes Bytes, int ByteIndex, int ByteCount, TCharArray Chars,
int CharIndex);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

37.1.45.3.8 TEncoding.GetEncoding Method

Returns the encoding used for the code page supplied.

Pascal

```
class function GetEncoding(CodePage: Integer): TEncoding; static;
```

C++

```
static __fastcall TEncoding GetEncoding(int CodePage);
```

Description

TEncoding.GetEncoding returns the preferred encoding for CodePage.

CodePage must be a valid codepage identifier or one of the CP_XXX constants defined by the Windows API.

See Also

Encoding.GetEncoding (MSDN)

CodePage Identifiers (MSDN)

37.1.45.3.9 TEncoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount) = 0;
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

37.1.45.3.10 TEncoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount) = 0;
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

37.1.45.3.11 TEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; virtual; abstract;
```

C++

```
virtual __fastcall TBytes GetPreamble() = 0;
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes (see page 4339) that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

37.1.45.3.12 GetString Method

This is the overview for the GetString method overload.

Overload List

	Name	Description
	TEncoding.GetString (TBytes) (see page 3981)	Decodes the byte array into a string.
	TEncoding.GetString (TBytes, Integer, Integer) (see page 3981)	Decodes the byte array into a string.

37.1.45.3.12.1 TEncoding.GetString Method (TBytes)

Decodes the byte array into a string.

Pascal

```
function GetString(const Bytes: TBytes): string; overload;
function GetString(const Bytes: TBytes; ByteIndex: Integer; ByteCount: Integer): string;
overload;
```

C++

```
__fastcall UnicodeString GetString(const TBytes Bytes);
__fastcall UnicodeString GetString(const TBytes Bytes, int ByteIndex, int ByteCount);
```

Description

TEncoding.GetString decodes the Bytes byte array into a string.

ByteIndex specifies the first byte to decode.

ByteCount specifies the number of bytes to decode.

Delphi Examples:

```
{
This example converts various string formats to and from Unicode.
This example requires two text edits and a combobox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  LBuffer: TBytes;
  LByteOrderMark: TBytes;
  LFileStream: TFileStream;
  LOffset: Integer;
  LEncoding, DestEncoding: TEncoding;
  ComboIndex: Integer;
  ByteString: RawByteString;
  FileText: string;
  ucs4Text: UCS4String;
  wideStringText: WideString;
  utf8Text: UTF8String;
  L: Cardinal;

begin
  LEncoding:= nil;
  ComboIndex := ComboBox1.ItemIndex;
  LFileStream := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    // Read file into buffer
    SetLength(LBuffer, LFileStream.Size);
    LFileStream.ReadBuffer(Pointer(LBuffer)^, Length(LBuffer));
    // Get data encoding of data read
    LOffset := TEncoding.GetBufferEncoding(LBuffer, LEncoding);
  finally
    LFileStream.Free;
  end;

  // Process various types
  case ComboIndex of

    // AnsiToUtf8
    0:
    begin
      // Identify encoding and convert buffer to Ansi
      DestEncoding := TEncoding.ASCII;
      LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
        LOffset, Length(LBuffer) - LOffset);
      LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
      FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
      ByteString := AnsiToUtf8(FileText);
      ShowMessage(UTF8ToString(ByteString));

      // Write to file
      LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
      try
        // New destination encoding
        DestEncoding := TEncoding.UTF8;
        LByteOrderMark := DestEncoding.GetPreamble;
        LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
        LFileStream.Write(ByteString[1], Length(ByteString));
      finally
        LFileStream.Free;
      end;
    end;
```

```
      // UCS4StringToWideString
      1:
      begin
        // Identify encoding and convert buffer to UCS4
        DestEncoding := TEncoding.Unicode;
        LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
          LOffset, Length(LBuffer) - LOffset);
        LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
        FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
        ucs4Text := UnicodeStringToUCS4String(FileText);
        wideStringText := UCS4StringToWideString(ucs4Text);
        ShowMessage(wideStringText);

        // Write to file
        LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
        try
          // New destination encoding
//          DestEncoding := TEncoding.UTF8;
          LByteOrderMark := DestEncoding.GetPreamble;
          LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
          LFileStream.Write(wideStringText[1], Length(wideStringText) * SizeOf(Char));
        finally
          LFileStream.Free;
        end;
      end;

      // UnicodeToUtf8
      2:
      begin
        // Identify encoding and convert buffer to Unicode
        DestEncoding := TEncoding.Unicode;
        LBuffer := LEncoding.Convert(LEncoding, DestEncoding, LBuffer,
          LOffset, Length(LBuffer) - LOffset);
        LOffset := TEncoding.GetBufferEncoding(LBuffer, DestEncoding);
        FileText := DestEncoding.GetString(LBuffer, LOffset, Length(LBuffer) - LOffset);
        L := Length(FileText);
        SetLength(utf8Text, L * SizeOf(Char) + 1);
        L := UnicodeToUtf8(PAnsiChar(utf8Text), Length(utf8Text), PWideChar(FileText), L);
        ShowMessage(UTF8ToString(utf8Text));

        // Write to file
        LFileStream := TFileStream.Create(Edit2.Text, fmCreate);
        try
          // New destination encoding
          DestEncoding := TEncoding.UTF8;
          LByteOrderMark := DestEncoding.GetPreamble;
          LFileStream.Write(LByteOrderMark[0], Length(LByteOrderMark));
          LFileStream.Write(utf8Text[1], L);
        finally
          LFileStream.Free;
        end;
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    StringList: TStrings;
  begin
    Edit1.Text := 'Sample.txt';
    Edit2.Text := 'Dest.txt';
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('AnsiToUtf8');
```

```
        Add('UCS4StringToWideString');
        Add('UnicodeToUtf8');
      end;
      with ComboBox1 do begin
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

Encoding.GetString (MSDN)

37.1.45.3.13 TEncoding.IsStandardEncoding Method

Reports if this instance is a standard encoding instance.

Pascal

```
class function IsStandardEncoding(AEncoding: TEncoding): Boolean; static;
```

C++

```
static __fastcall Boolean IsStandardEncoding(TEncoding AEncoding);
```

Description

TEncoding.IsStandardEncoding compares AEncoding to the instances of TEncoding (see page 3969) provided in the TEncoding (see page 3969) class properties: Default (see page 3985), ASCII (see page 3985), BigEndianUnicode (see page 3985), Unicode (see page 3986), UTF7 (see page 3986) and UTF8 (see page 3987). The Return Value specifies if that search was successful.

See Also

ASCII (see page 3985)

BigEndianUnicode (see page 3985)

Default (see page 3985)

Unicode (see page 3986)

UTF7 (see page 3986)

UTF8 (see page 3987)

37.1.45.4 TEncoding Properties

The properties of the TEncoding class are listed here.

Public Properties

	Name	Description
R	ASCII (see page 3985)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 3985)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 3985)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 3986)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 3986)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 3986)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 3987)	Returns an encoding for the UTF-8 format.

37.1.45.4.1 TEncoding.ASCII Property

Returns an encoding for the ASCII character set.

Pascal

```
class property ASCII: TEncoding;
```

C++

```
__property TEncoding ASCII;
```

Description

TEncoding.ASCII returns an encoding for the ASCII character set.

The **Return Value** is an instance of type TMBCSEncoding (see page 3992).

See Also

Encoding.ASCII (MSDN)

37.1.45.4.2 TEncoding.BigEndianUnicode Property

Returns an encoding for the UTF-16 format with big-endian byte order.

Pascal

```
class property BigEndianUnicode: TEncoding;
```

C++

```
__property TEncoding BigEndianUnicode;
```

Description

TEncoding.BigEndianUnicode returns an encoding for the UTF-16 format with big-endian byte order.

The **Return Value** is an instance of type TBigEndianUnicodeEncoding (see page 3967). This encoding has a byte order mark of hexadecimal FEFF.

See Also

Encoding.BigEndianUnicode (MSDN)

37.1.45.4.3 TEncoding.Default Property

Returns an encoding for the operating system's default ANSI code page.

Pascal

```
class property Default: TEncoding;
```

C++

```
__property TEncoding Default;
```

Description

TEncoding.Default returns an encoding for the operating system's default ANSI code page.

The **Return Value** is an instance of type TMBCSEncoding (see page 3992).

See Also

Encoding.Default (MSDN)

37.1.45.4.4 TEncoding.IsSingleByte Property

This boolean specifies if the current encoding is of single-byte format.

Pascal

```
property IsSingleByte: Boolean;
```

C++

```
__property Boolean IsSingleByte;
```

Description

TEncoding.IsSingleByte specifies if the current encoding is of single-byte format.

See Also

Encoding.IsSingleByte (MSDN)

37.1.45.4.5 TEncoding.Unicode Property

Returns an encoding for the UTF-16 format with little-endian byte order.

Pascal

```
class property Unicode: TEncoding;
```

C++

```
__property TEncoding Unicode;
```

Description

TEncoding.Unicode returns an encoding for the UTF-16 format with little-endian byte order.

The **Return Value** is an instance of type TUnicodeEncoding (see page 4024). This encoding has a byte order mark of hexadecimal FFFE.

See Also

Encoding.Unicode (MSDN)

37.1.45.4.6 TEncoding.UTF7 Property

Returns an encoding for the UTF-7 format.

Pascal

```
class property UTF7: TEncoding;
```

C++

```
__property TEncoding UTF7;
```

Description

TEncoding.UTF7 returns an encoding for the UTF-7 format..

The **Return Value** is an instance of type TUTF7Encoding (see page 4029).

See Also

Encoding.UTF7 (MSDN)

37.1.45.4.7 TEncoding.UTF8 Property

Returns an encoding for the UTF-8 format.

Pascal

```
class property UTF8: TEncoding;
```

C++

```
__property TEncoding UTF8;
```

Description

TEncoding.UTF8 returns an encoding for the UTF-8 format.

The **Return Value** is an instance of type TUTF8Encoding (see page 4034). This encoding has a byte order mark of hexadecimal EFBBBF.

See Also

Encoding.UTF8 (MSDN)

37.1.46 TLanguages Class

TLanguages lists all available Windows locales.

Class Hierarchy

Pascal

```
TLanguages = class;
```

C++

```
class TLanguages;
```

File

SysUtils

Description

The Languages (see page 4223) function returns a TLanguages instance that lists the Windows locales supported by the current system. For each locale, TLanguages indicates the locale ID (see page 3990) (LCID), standard name, and recommended file-name extension. This information is obtained from the operating system; all TLanguages properties are read-only.

See Also

Languages (see page 4223)

37.1.46.1 TLanguages Members

The following tables list the members exposed by TLanguages.

Public Methods

	Name	Description
	Create (see page 3988)	Creates a new TLanguages (see page 3987) object.
	IndexOf (see page 3988)	Returns the index of the first locale in the array with the specified LCID.

Public Properties

	Name	Description
R	Count (see page 3989)	Contains the number of locales listed in the TLanguages (see page 3987) object.
R	Ext (see page 3989)	Contains the standard file-name extension at the specified position in the array.
R	ID (see page 3990)	Contains a locale ID as a string.
R	LocaleID (see page 3990)	Contains a locale ID (see page 3990) as an integer.
R	Name (see page 3991)	Contains the locale name at the specified position in the array.
R	NameFromLCID (see page 3991)	Contains the locale name specified by a string representation of a locale ID (see page 3990).
R	NameFromLocaleID (see page 3991)	Contains the locale name specified by an integer locale ID (see page 3990).

37.1.46.2 TLanguages.Create Constructor

Creates a new TLanguages (see page 3987) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TLanguages();
```

Description

Do not construct a TLanguages (see page 3987) object directly. Instead, call the Languages (see page 4223) function, which returns an instance of TLanguages (see page 3987) with locale information obtained from Windows.

See Also

Languages (see page 4223)

37.1.46.3 TLanguages Methods

The methods of the TLanguages class are listed here.

Public Methods

	Name	Description
	IndexOf (see page 3988)	Returns the index of the first locale in the array with the specified LCID.

37.1.46.3.1 TLanguages.IndexOf Method

Returns the index of the first locale in the array with the specified LCID.

Pascal

```
function IndexOf(ID: LCID): Integer;
```

C++

```
__fastcall int IndexOf(LCID ID);
```

Description

Call IndexOf to get the position in the array of a specified locale ID (see page 3990). For example, if U.S. English is the first locale listed in TLanguages (see page 3987), then IndexOf($0409) returns 0.

If the specified LCID is not found, IndexOf returns -1.

See Also

Ext (see page 3989)

ID (see page 3990)

LocaleID (see page 3990)

Name (see page 3991)

37.1.46.4 TLanguages Properties

The properties of the TLanguages class are listed here.

Public Properties

	Name	Description
	Count (see page 3989)	Contains the number of locales listed in the TLanguages (see page 3987) object.
	Ext (see page 3989)	Contains the standard file-name extension at the specified position in the array.
	ID (see page 3990)	Contains a locale ID as a string.
	LocaleID (see page 3990)	Contains a locale ID (see page 3990) as an integer.
	Name (see page 3991)	Contains the locale name at the specified position in the array.
	NameFromLCID (see page 3991)	Contains the locale name specified by a string representation of a locale ID (see page 3990).
	NameFromLocaleID (see page 3991)	Contains the locale name specified by an integer locale ID (see page 3990).

37.1.46.4.1 TLanguages.Count Property

Contains the number of locales listed in the TLanguages (see page 3987) object.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3634) Count to determine the number of locales supported by the current system.

See Also

ID (see page 3990)

LocaleID (see page 3990)

IndexOf (see page 3988)

37.1.46.4.2 TLanguages.Ext Property

Contains the standard file-name extension at the specified position in the array.

Pascal

```
property Ext [Index: Integer]: string;
```

C++

```
__property UnicodeString Ext[int Index];
```

Description

Read (see page 3634) Ext to get the standard three-letter file-name extension for a locale. For example, Ext[IndexOf (see page 3988)($0409)] returns the string ENU.

Ext is a zero-based array.

See Also

IndexOf (see page 3988)

37.1.46.4.3 TLanguages.ID Property

Contains a locale ID as a string.

Pascal

```
property ID [Index: Integer]: string;
```

C++

```
__property UnicodeString ID[int Index];
```

Description

Read (see page 3634) ID to get the locale ID at a specified position in the array as a string. (To get the same ID as an integer, use the LocaleID (see page 3990) property.) For example, if U.S. English is the first locale listed in TLanguages (see page 3987), then ID[0] returns the string $00000409. You can iterate through the ID array (from 0 to Count (see page 3989) - 1) to determine whether a specific locale is in the array.

See Also

Count (see page 3989)

LocaleID (see page 3990)

37.1.46.4.4 TLanguages.LocaleID Property

Contains a locale ID (see page 3990) as an integer.

Pascal

```
property LocaleID [Index: Integer]: LCID;
```

C++

```
__property LCID LocaleID[int Index];
```

Description

Read (see page 3634) LocaleID to get the locale ID (see page 3990) at a specified position in the array as an integer. (To get the same ID (see page 3990) as a string, use the ID (see page 3990) property.) For example, if U.S. English is the first locale listed in TLanguages (see page 3987), then LocaleID[0] returns the integer $409. You can iterate through the LocaleID array (from 0 to Count (see page 3989) - 1) to determine whether a specific locale is in the array.

See Also

Count (see page 3989)

ID (see page 3990)

37.1.46.4.5 TLanguages.Name Property

Contains the locale name at the specified position in the array.

Pascal

```
property Name [Index: Integer]: string;
```

C++

```
__property UnicodeString Name[int Index];
```

Description

Read (see page 3634) Name to get the standard Windows name for a locale. For example, Name[IndexOf (see page 3988)($0409)] returns the string English (United States).

Name is a zero-based array.

See Also

IndexOf (see page 3988)

NameFromLCID (see page 3991)

NameFromLocaleID (see page 3991)

37.1.46.4.6 TLanguages.NameFromLCID Property

Contains the locale name specified by a string representation of a locale ID (see page 3990).

Pascal

```
property NameFromLCID [const ID: string]: string;
```

C++

```
__property UnicodeString NameFromLCID[UnicodeString const ID];
```

Description

Read (see page 3634) NameFromLCID to get the standard Windows name for a locale using a locale ID (see page 3990) represented by a string.

See Also

Name (see page 3991)

NameFromLocaleID (see page 3991)

37.1.46.4.7 TLanguages.NameFromLocaleID Property

Contains the locale name specified by an integer locale ID (see page 3990).

Pascal

```
property NameFromLocaleID [ID: LCID]: string;
```

C++

```
__property UnicodeString NameFromLocaleID[LCID ID];
```

Description

Read (see page 3634) NameFromLocaleID to get the standard Windows name for a locale using an integer locale ID (see page 3990). For example, NameFromLCID (see page 3991)[$0409] returns the string English (United States).

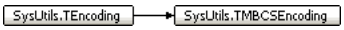
See Also

NameFromLCID (see page 3991)

Name (see page 3991)

37.1.47 TMBCSEncoding Class

Class Hierarchy



Pascal

```
TMBCSEncoding = class(TEncoding);
```

C++

```
class TMBCSEncoding : public TEncoding;
```

File

SysUtils

Description

This is class SysUtils.TMBCSEncoding.

37.1.47.1 TMBCSEncoding Members

The following tables list the members exposed by TMBCSEncoding.

Public Methods

	Name	Description
	Convert (see page 3974)	This is the overview for the Convert method overload.
V	Create (see page 3993)	This is the overview for the Create constructor overload.
	FreeEncodings (see page 3975)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 3976)	Determines the encoding for a given buffer.
	GetEncoding (see page 3980)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 3996)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 3996)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 3997)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 3981)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 3984)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 3985)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 3985)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 3985)	Returns an encoding for the operating system's default ANSI code page.

R	IsSingleByte (see page 3986)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 3986)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 3986)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 3987)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 3973)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 3974)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 3994)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 3976)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 3995)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 3977)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 3995)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 3978)	This is the overview for the GetCharCount method overload.
V	GetChars (see page 3996)	Decodes a set of bytes into a set of characters.
A	GetChars (see page 3979)	This is the overview for the GetChars method overload.

37.1.47.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
V	TMBCSEncoding.Create () (see page 3993)	This is Create, a member of class TMBCSEncoding.
V	TMBCSEncoding.Create (Integer) (see page 3993)	This is Create, a member of class TMBCSEncoding.
V	TMBCSEncoding.Create (Integer, Integer, Integer) (see page 3994)	This is Create, a member of class TMBCSEncoding.

37.1.47.2.1 TMBCSEncoding.Create Constructor ()

Pascal

```
constructor Create; virtual; overload;
```

C++

```
virtual __fastcall TMBCSEncoding();
```

Description

This is Create, a member of class TMBCSEncoding.

37.1.47.2.2 TMBCSEncoding.Create Constructor (Integer)

Pascal

```
constructor Create(CodePage: Integer); virtual; overload;
```

C++

```
virtual __fastcall TMBCSEncoding(int CodePage);
```

Description

This is Create, a member of class TMBCSEncoding.

37.1.47.2.3 TMBCSEncoding.Create Constructor (Integer, Integer, Integer)

Pascal

```
constructor Create(CodePage: Integer; MBToWCharFlags: Integer; WCharToMBFlags: Integer);
virtual; overload;
```

C++

```
virtual __fastcall TMBCSEncoding(int CodePage, int MBToWCharFlags, int WCharToMBFlags);
```

Description

This is Create, a member of class TMBCSEncoding.

37.1.47.3 TMBCSEncoding Methods

The methods of the TMBCSEncoding class are listed here.

Public Methods

	Name	Description
	GetMaxByteCount (see page 3996)	Calculates the maximum bytes generated to encode the specified number of characters.
	GetMaxCharCount (see page 3996)	Calculates the maximum number of characters generated to decode a specified number of bytes.
	GetPreamble (see page 3997)	Returns a preamble in a byte array that specifies the encoding used.

Strict protected Methods

	Name	Description
	GetByteCount (see page 3994)	Returns the number of bytes generated by encoding Chars.
	GetBytes (see page 3995)	Encodes a set of characters into a sequence of bytes.
	GetCharCount (see page 3995)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetChars (see page 3996)	Decodes a set of bytes into a set of characters.

37.1.47.3.1 TMBCSEncoding.GetByteCount Method

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

37.1.47.3.2 TMBCSEncoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

37.1.47.3.3 TMBCSEncoding.GetCharCount Method

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

37.1.47.3.4 TMBCSEncoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

37.1.47.3.5 TMBCSEncoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

37.1.47.3.6 TMBCSEncoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

37.1.47.3.7 TMBCSEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

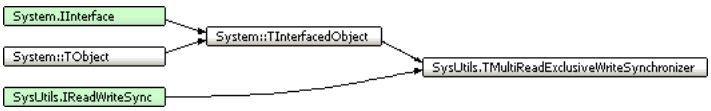
See Also

Encoding.GetPreamble (MSDN)

37.1.48 TMultiReadExclusiveWriteSynchronizer Class

TMultiReadExclusiveWriteSynchronizer protects memory in a multi-threaded application.

Class Hierarchy



Pascal

```
TMultiReadExclusiveWriteSynchronizer = class(TInterfacedObject, IReadWriteSync);
```

C++

```
class TMultiReadExclusiveWriteSynchronizer : public TInterfacedObject, public IReadWriteSync;
```

File

SysUtils

Description

Use TMultiReadExclusiveWriteSynchronizer to guard access to memory in a multi-threaded application. Unlike a critical section,

which blocks all other threads from reading or writing its associated memory, TMultiReadExclusiveWriteSynchronizer allows multiple threads to read from the protected memory simultaneously, while ensuring that any thread writing to the memory has exclusive access.

In applications where threads read from an object or variable frequently and only write to it occasionally, using the multi-read exclusive-write synchronizer instead of a critical section can result in considerable performance improvement.

All access to the protected memory must be bracketed by calls to the BeginRead (see page 4000) and EndRead (see page 4001) or BeginWrite (see page 4000) and EndWrite (see page 4001) methods. Any thread that reads from or writes to this memory without using these calls can introduce thread conflicts.

See Also

TCriticalSection (see page 3125)

TThread

Using the Multi-read Exclusive-write Synchronizer

37.1.48.1 TMultiReadExclusiveWriteSynchronizer Members

The following tables list the members exposed by TMultiReadExclusiveWriteSynchronizer.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	BeginRead (see page 4000)	Allows a thread to read from the protected memory when it is safe to do so.
	BeginWrite (see page 4000)	Allows a thread to write to the protected memory when it is safe to do so.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3999)	Creates and initializes an instance of TMultiReadExclusiveWriteSynchronizer (see page 3997).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.

	Destroy (see page 3999)	Destroys an instance of TMultiReadExclusiveWriteSynchronizer (see page 3997).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndRead (see page 4001)	Removes a thread from the internal list of threads that are accessing the protected memory.
	EndWrite (see page 4001)	Allows other threads to access the associated memory.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	RevisionLevel (see page 4002)	Changes revision level of protected memory.

37.1.48.2 TMultiReadExclusiveWriteSynchronizer.Create Constructor

Creates and initializes an instance of TMultiReadExclusiveWriteSynchronizer (see page 3997).

Pascal

```
constructor Create;
```

C++

```
__fastcall TMultiReadExclusiveWriteSynchronizer();
```

Description

Call Create to instantiate TMultiReadExclusiveWriteSynchronizer (see page 3997). The synchronizer must be created in the same thread as the memory it protects, so that it is accessible to all threads that need to access that memory.

37.1.48.3 TMultiReadExclusiveWriteSynchronizer.Destroy Destructor

Destroys an instance of TMultiReadExclusiveWriteSynchronizer (see page 3997).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TMultiReadExclusiveWriteSynchronizer();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TMultiReadExclusiveWriteSynchronizer (see page 3997) reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

37.1.48.4 TMultiReadExclusiveWriteSynchronizer Methods

The methods of the TMultiReadExclusiveWriteSynchronizer class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 4000)	Allows a thread to read from the protected memory when it is safe to do so.
	BeginWrite (see page 4000)	Allows a thread to write to the protected memory when it is safe to do so.
	EndRead (see page 4001)	Removes a thread from the internal list of threads that are accessing the protected memory.
	EndWrite (see page 4001)	Allows other threads to access the associated memory.

37.1.48.4.1 TMultiReadExclusiveWriteSynchronizer.BeginRead Method

Allows a thread to read from the protected memory when it is safe to do so.

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

Call BeginRead before reading from the memory protected by the multi-read exclusive-write synchronizer. BeginRead does not return until there are no other threads writing to the memory. After calling BeginRead, other threads can be granted read access to the protected memory by the BeginRead method. However, any thread asking for write permission by calling BeginWrite (see page 4000) must wait until all reading threads have called EndRead (see page 4001).

See Also

EndRead (see page 4001)

BeginWrite (see page 4000)

Using the Multi-read Exclusive-write Synchronizer

37.1.48.4.2 TMultiReadExclusiveWriteSynchronizer.BeginWrite Method

Allows a thread to write to the protected memory when it is safe to do so.

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

BeginWrite obtains an exclusive write lock on the memory protected by the synchronizer object. Call BeginWrite before writing to this memory. BeginWrite does not return until no other threads are reading from or writing to the memory. After a thread calls BeginWrite, any other thread asking to access that memory by calling BeginRead (see page 4000) or BeginWrite must wait until the writing thread calls EndWrite (see page 4001).

Note that a write lock operation is not atomic. Data in protected memory will always be in a consistent state when BeginWrite returns, but is not necessarily in the same state as when BeginWrite was called. As a rule, a thread should always discard previous samples from protected memory after promoting a read lock to a write lock. However, the calling thread can determine whether a state change has occurred by examining return value of BeginWrite: true if protected memory has not be written to by another thread, false if another thread may have modified protected memory.

See Also

EndWrite (see page 4001)

BeginRead (see page 4000)

Using the Multi-read Exclusive-write Synchronizer

37.1.48.4.3 TMultiReadExclusiveWriteSynchronizer.EndRead Method

Removes a thread from the internal list of threads that are accessing the protected memory.

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

Call EndRead after reading from the memory protected by the multi-read exclusive-write synchronizer. Each call to BeginRead (see page 4000) must be matched by a call to EndRead, or other threads can never return from the BeginWrite (see page 4000) method.

See Also

BeginRead (see page 4000)

BeginWrite (see page 4000)

Using the Multi-read Exclusive-write Synchronizer

37.1.48.4.4 TMultiReadExclusiveWriteSynchronizer.EndWrite Method

Allows other threads to access the associated memory.

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

Call EndWrite after writing to the memory that is protected by the multi-read exclusive-write synchronizer. Each call to BeginWrite (see page 4000) must be matched by a call to EndWrite, or other threads can never return from the BeginRead (see page 4000) or BeginWrite (see page 4000) methods.

See Also

BeginWrite (see page 4000)

BeginRead (see page 4000)

Using the Multi-read Exclusive-write Synchronizer

37.1.48.5 TMultiReadExclusiveWriteSynchronizer Properties

The properties of the TMultiReadExclusiveWriteSynchronizer class are listed here.

Public Properties

	Name	Description
	RevisionLevel (see page 4002)	Changes revision level of protected memory.

37.1.48.5.1 TMultiReadExclusiveWriteSynchronizer.RevisionLevel Property

Changes revision level of protected memory.

Pascal

```
property RevisionLevel: Cardinal;
```

C++

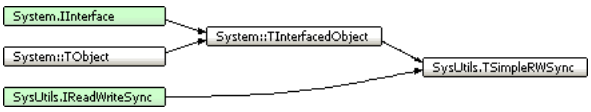
```
__property unsigned RevisionLevel;
```

Description

RevisionLevel changes each time a write lock is granted. This property should be used for identity comparisons only, not for determining revision sequence.

37.1.49 TSimpleRWSync Class

Class Hierarchy



Pascal

```
TSimpleRWSync = class(TInterfacedObject, IReadWriteSync);
```

C++

```
class TSimpleRWSync : public TInterfacedObject, public IReadWriteSync;
```

File

SysUtils

Description

This is class SysUtils.TSimpleRWSync.

37.1.49.1 TSimpleRWSync Members

The following tables list the members exposed by TSimpleRWSync.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	BeginRead (see page 4005)	This is BeginRead, a member of class TSimpleRWSync.
	BeginWrite (see page 4005)	This is BeginWrite, a member of class TSimpleRWSync.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 4004)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 4004)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndRead (see page 4006)	This is EndRead, a member of class TSimpleRWSync.
	EndWrite (see page 4006)	This is EndWrite, a member of class TSimpleRWSync.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.

	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

37.1.49.2 TSimpleRWSync.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TSimpleRWSync();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

37.1.49.3 TSimpleRWSync.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSimpleRWSync();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

37.1.49.4 TSimpleRWSync Methods

The methods of the TSimpleRWSync class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 4005)	This is BeginRead, a member of class TSimpleRWSync.
	BeginWrite (see page 4005)	This is BeginWrite, a member of class TSimpleRWSync.
	EndRead (see page 4006)	This is EndRead, a member of class TSimpleRWSync.
	EndWrite (see page 4006)	This is EndWrite, a member of class TSimpleRWSync.

37.1.49.4.1 TSimpleRWSync.BeginRead Method

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

This is BeginRead, a member of class TSimpleRWSync.

37.1.49.4.2 TSimpleRWSync.BeginWrite Method

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

This is BeginWrite, a member of class TSimpleRWSync.

37.1.49.4.3 TSimpleRWSync.EndRead Method

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

This is EndRead, a member of class TSimpleRWSync.

37.1.49.4.4 TSimpleRWSync.EndWrite Method

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

This is EndWrite, a member of class TSimpleRWSync.

37.1.50 TStringBuilder Class

An editable array of characters whose contents has all the functionality of a string object.

Class Hierarchy

Pascal

```
TStringBuilder = class;
```

C++

```
class TStringBuilder;
```

File

SysUtils

Description

Use a TStringBuilder as an editable string.

The TStringBuilder class is completely compatable with the .Net framework's StringBuilder (MSDN) class. Use a TStringBuilder instance to hold a character array. This array can be modified after creation. Substrings can be appended, searched and replaced, or inserted. The character array can be queried by index or converted to a string for comparison.

The character array can be referenced directly by using the Chars (see page 4022) property. The Length (see page 4022) property contains the current length of the character array.

builder.Length (see page 4022) = builder.ToString.Length (see page 4022);

The Capacity (see page 4021) property holds the current allocated storage space for this TStringBuilder instance. Capacity (see page 4021) can be increased up to the value of MaxCapacity (see page 4022) and down to the current value of Length (see page 4022). Capacity (see page 4021) is increased if Length (see page 4022) is increased, or the character array is altered by appending to increase it's length.

The Append (see page 4014) method converts a multitude of possible types to a string and appends that to the current character array. AppendFormat (see page 4015) allows you to append several objects to the character array, each object with it's own format.

The Equals (see page 4017) method compares a specified object or another TStringBuilder to this instance of TStringBuilder.

The Append (see page 4014), CopyTo (see page 4016), Insert (see page 4018), Remove (see page 4019) and Replace (see page 4020) methods access the character array by index or search the character as a string. The ToString (see page 4021) method converts the character array to a string to provide string functionality.

TStringBuilder is an implementation of the .NET StringBuilder (MSDN) class.

C++ Examples:

```
/*
This example exercises many of the members of the TStringBuilder class.
*/
template <class T>
void testAppend(T t)
{
    std::auto_ptr<TStringBuilder> builder(new TStringBuilder());
    builder->Append(t);
    UnicodeString s1 = builder->ToString();
    UnicodeString s2(t);
    assert(s1 == s2);
}


#if defined(_DELPHI_STRING_UNICODE)
const wchar_t* ext_chars =
    L"Ð• Ð‘ Ð’ Ð“ Ð„ Ð… Ð† Ð‡ Ðˆ Ð‰"
    L"Ðš Ð› Ðœ Ð• Ðž ÐŸ Ò€ Ð  Ð¡ Ð¢ Ñ„"
    L"Ð¤ Ð¥ Ñ  Ð¦ Ð§ Ð¨ Ð© Ðª Ð« Ð¬ Ñ¢"
    L"Ð® Ð†Ð• Ñ¦ Ñ¨ Ñª Ñ¬ Ñ® Ñ° Ñ² Ñ´ Ñ¤";
#else
const char* ext_chars =
    "Ã€ Ã• Ã‚ Ãƒ Ã„ Ã… Ã† Ã‡ Ãˆ Ã‰ Ãˆ Ã‹ Ãœ Ã• ÃŽ Ã•"
    "Ã• Ã‘ Ã’ Ã“ Ã” Ã• Ã– Ã— Ã˜ Ã™ Ãš Ã› Ãœ Ã• Ãž ÃŸ"
    "Ã  Ã¡ Ã¢ Ã£ Ã¤ Ã¥ Ã¦ Ã§ Ã¨ Ã© Ãª Ã« Ã¬ Ã- Ã® Ã¯"
    "Ã° Ã± Ã² Ã³ Ã´ Ãµ Ã¶ Ã· Ã¸ Ã¹ Ãº Ã» Ã¼ Ã½ Ã¾ Ã¿";
#endif

//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    {
        // Test starts out as empty
        std::auto_ptr<TStringBuilder> builder(new TStringBuilder());
        assert(builder->ToString() == System::String());
    }

    {
        // Append
        short s = 0xffe;
        testAppend(s);

        unsigned short us = 0xfffe;
        testAppend(us);
```

```
}

{
    const _DCHAR* p = ext_chars;

    std::auto_ptr<TStringBuilder> builder(new TStringBuilder());
    System::String str1(p);

    // Appending of wchar_t
    while (*p)
    {
        builder->Append(*p++);
    }

    // Length property
    assert(builder->Length == str1.Length());

    // Subscript operator
    for (int i=0; i<str1.Length(); i++)
    {
        // NOTE: strings are 1-based
        assert((*(builder.get()))[i] == str1[i+1]);
    }

    // ToString()
    System::String str2 = builder->ToString();
    Edit1->Text = str1;
    Edit2->Text = str2;
    assert(str2 == str1);

    // Replace
    builder->Replace(' ', '.');
    UnicodeString str3;
    str3 = builder->ToString();
    Edit3->Text = str3;
    assert(str3 != str2);

    // Replace back
    builder->Replace(L'.', L' ');
    System::String str4 = builder->ToString();
    Edit4->Text = str4;
    assert(str2 == str4);
}

{
    System::String str(ext_chars);

    std::auto_ptr<TStringBuilder> builder(new TStringBuilder());

    // Insert at start when empty, at start when not empty and at end
    builder->Insert(0, str);
    builder->Insert(0, str);
    builder->Insert(builder->Length, str);

    // Concatenate string 3 times
    System::String str2;
    str2 += str;
    str2 += str;
    str2 += str;

    str = builder->ToString();

    Edit5->Text = str;
    assert(builder->ToString() == str2);
}
MessageDlg("All assertions passed in the TStringBuilder Test test.",
```

```
        mtInformation, TMsgDlgButtons() << mbOK, 0);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    const _DCHAR* p = ext_chars;

    std::auto_ptr<TStringBuilder> builder(new TStringBuilder());

    // Appending of chars
    while (*p)
    {
        builder->Append(*p++);
    }

    System::String str1(ext_chars);
    System::String str2 = builder->ToString();
    assert(str1 == str2);

    builder->Replace(_D(" "), _D("."));
    System::String str3;
    str3 = builder->ToString();
    assert(str1 != str3);

    builder->Replace(_D("."), _D(" "));
    System::String str4(builder->ToString());
    assert(str1 == str4);
    MessageDlg("All assertions passed in the TStringBuilder Replace test.",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

See Also

Append (see page 4014)

AppendFormat (see page 4015)

AppendLine (see page 4016)

CopyTo (see page 4016)

EnsureCapacity (see page 4016)

Equals (see page 4017)

Insert (see page 4018)

Remove (see page 4019)

Replace (see page 4020)

ToString (see page 4021)

Capacity (see page 4021)

Chars (see page 4022)

Length (see page 4022)

MaxCapacity (see page 4022)

Andreano Lanusse's blog about TStringBuilder

StringBuilder (MSDN)

37.1.50.1 TStringBuilder Members

The following tables list the members exposed by TStringBuilder.

Protected Fields

	Name	Description
	FData (see page 4012)	This is FData, a member of class TStringBuilder.
	FLength (see page 4012)	This is FLength, a member of class TStringBuilder.
	FMaxCapacity (see page 4012)	This is FMaxCapacity, a member of class TStringBuilder.

Public Methods

	Name	Description
	Append (see page 4013)	This is the overview for the Append method overload.
	AppendFormat (see page 4015)	Appends a number of objects to the current character array in this TStringBuilder (see page 4006) instance.
	AppendLine (see page 4015)	This is the overview for the AppendLine method overload.
	CopyTo (see page 4016)	Copys a part of the instance character array into a destination string.
	Create (see page 4010)	This is the overview for the Create constructor overload.
	EnsureCapacity (see page 4016)	Sets the Capacity (see page 4021) property to the supplied value, if the new capacity is larger than the old capacity.
	Equals (see page 4017)	Compares the character array in the Chars (see page 4022) property, the Capacity (see page 4021) property, and the MaxCapacity (see page 4022) property of this instance to that of a supplied TStringBuilder (see page 4006) instance.
	Insert (see page 4017)	This is the overview for the Insert method overload.
	Remove (see page 4019)	Removes a substring from the character array of this instance.
	Replace (see page 4019)	This is the overview for the Replace method overload.
V	ToString (see page 4020)	This is the overview for the ToString method overload.

Public Properties

	Name	Description
	Capacity (see page 4021)	Used to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4006) instance.
	Chars (see page 4022)	Used to get or set the characters stored in this TStringBuilder (see page 4006) instance.
	Length (see page 4022)	Used to get or set the size of the character array stored in this TStringBuilder (see page 4006) instance.
R	MaxCapacity (see page 4022)	Used to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4006) instance.

37.1.50.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TStringBuilder.Create () (see page 4011)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4006).
	TStringBuilder.Create (Integer) (see page 4011)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4006).
	TStringBuilder.Create (Integer, Integer) (see page 4011)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4006).

	TStringBuilder.Create (string) (see page 4011)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4006).
	TStringBuilder.Create (string, Integer) (see page 4011)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4006).
	TStringBuilder.Create (string, Integer, Integer, Integer) (see page 4011)	Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4006).

37.1.50.2.1 TStringBuilder.Create Constructor ()

Allocates memory and constructs a safely initialized instance of a TStringBuilder (see page 4006).

Pascal

```
constructor Create; overload;
constructor Create(aCapacity: Integer); overload;
constructor Create(const Value: string); overload;
constructor Create(aCapacity: Integer; aMaxCapacity: Integer); overload;
constructor Create(const Value: string; aCapacity: Integer); overload;
constructor Create(const Value: string; StartIndex: Integer; Length: Integer; aCapacity:
Integer); overload;
```

C++

```
__fastcall TStringBuilder();
__fastcall TStringBuilder(int aCapacity);
__fastcall TStringBuilder(const UnicodeString Value);
__fastcall TStringBuilder(int aCapacity, int aMaxCapacity);
__fastcall TStringBuilder(const UnicodeString Value, int aCapacity);
__fastcall TStringBuilder(const UnicodeString Value, int StartIndex, int Length, int
aCapacity);
```

Description

Use Create to Create an instance of the TStringBuilder (see page 4006) class.

aCapacity specifies the initial value of the Capacity (see page 4021) property, the maximum number of characters this TStringBuilder (see page 4006) instance can hold. The value of the Capacity (see page 4021) property can be changed.

Value specifies the initial character array to be stored in this TStringBuilder (see page 4006) instance.

aMaxCapacity is the maximum value that the Capacity (see page 4021) property can be for this instance of the TStringBuilder (see page 4006) class.

StartIndex specifies the index of the first character in the Value string to be copied into the initial character array of this TStringBuilder (see page 4006) instance.

Length (see page 4022) is the number of characters to copy from the Value string into the initial character array of this TStringBuilder (see page 4006) instance.

See Also

Capacity (see page 4021)

StringBuilder.Create (MSDN)

37.1.50.3 TStringBuilder Fields

The fields of the TStringBuilder class are listed here.

Protected Fields

	Name	Description
	FData (see page 4012)	This is FData, a member of class TStringBuilder.
	FLength (see page 4012)	This is FLength, a member of class TStringBuilder.
	FMaxCapacity (see page 4012)	This is FMaxCapacity, a member of class TStringBuilder.

37.1.50.3.1 TStringBuilder.FData Field

Pascal

```
FData: TCharArray;
```

C++

```
TCharArray FData;
```

Description

This is FData, a member of class TStringBuilder.

37.1.50.3.2 TStringBuilder.FLength Field

Pascal

```
FLength: Integer;
```

C++

```
int FLength;
```

Description

This is FLength, a member of class TStringBuilder.

37.1.50.3.3 TStringBuilder.FMaxCapacity Field

Pascal

```
FMaxCapacity: Integer;
```

C++

```
int FMaxCapacity;
```

Description

This is FMaxCapacity, a member of class TStringBuilder.

37.1.50.4 TStringBuilder Methods

The methods of the TStringBuilder class are listed here.

Public Methods

	Name	Description
	Append (see page 4013)	This is the overview for the Append method overload.
	AppendFormat (see page 4015)	Appends a number of objects to the current character array in this TStringBuilder (see page 4006) instance.
	AppendLine (see page 4015)	This is the overview for the AppendLine method overload.
	CopyTo (see page 4016)	Copys a part of the instance character array into a destination string.

	EnsureCapacity (see page 4016)	Sets the Capacity (see page 4021) property to the supplied value, if the new capacity is larger than the old capacity.
	Equals (see page 4017)	Compares the character array in the Chars (see page 4022) property, the Capacity (see page 4021) property, and the MaxCapacity (see page 4022) property of this instance to that of a supplied TStringBuilder (see page 4006) instance.
	Insert (see page 4017)	This is the overview for the Insert method overload.
	Remove (see page 4019)	Removes a substring from the character array of this instance.
	Replace (see page 4019)	This is the overview for the Replace method overload.
	ToString (see page 4020)	This is the overview for the ToString method overload.

37.1.50.4.1 Append Method

This is the overview for the Append method overload.

Overload List

	Name	Description
	TStringBuilder.Append (Boolean) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Byte) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Cardinal) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Char) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Char, Integer) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Currency) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Double) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Int64) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Integer) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Shortint) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Single) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.

	TStringBuilder.Append (Smallint) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (string) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (string, Integer, Integer) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (TCharArray) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (TCharArray, Integer, Integer) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (TObject) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (UInt64) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.
	TStringBuilder.Append (Word) (see page 4014)	Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.

37.1.50.4.1.1 TStringBuilder.Append Method (Boolean)

Appends a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.

Pascal

```
function Append(const Value: Boolean): TStringBuilder; overload;
function Append(const Value: Byte): TStringBuilder; overload;
function Append(const Value: Char): TStringBuilder; overload;
function Append(const Value: Currency): TStringBuilder; overload;
function Append(const Value: Double): TStringBuilder; overload;
function Append(const Value: Smallint): TStringBuilder; overload;
function Append(const Value: Integer): TStringBuilder; overload;
function Append(const Value: Int64): TStringBuilder; overload;
function Append(const Value: TObject): TStringBuilder; overload;
function Append(const Value: Shortint): TStringBuilder; overload;
function Append(const Value: Single): TStringBuilder; overload;
function Append(const Value: string): TStringBuilder; overload;
function Append(const Value: UInt64): TStringBuilder; overload;
function Append(const Value: TCharArray): TStringBuilder; overload;
function Append(const Value: Word): TStringBuilder; overload;
function Append(const Value: Cardinal): TStringBuilder; overload;
function Append(const Value: Char; RepeatCount: Integer): TStringBuilder; overload;
function Append(const Value: TCharArray; StartIndex: Integer; CharCount: Integer):
TStringBuilder; overload;
function Append(const Value: string; StartIndex: Integer; Count: Integer): TStringBuilder;
overload;
```

C++

```
__fastcall TStringBuilder Append(const Boolean Value);
__fastcall TStringBuilder Append(const Byte Value);
```

```
__fastcall TStringBuilder Append(const Char Value);
__fastcall TStringBuilder Append(const Currency Value);
__fastcall TStringBuilder Append(const Double Value);
__fastcall TStringBuilder Append(const Smallint Value);
__fastcall TStringBuilder Append(const int Value);
__fastcall TStringBuilder Append(const Int64 Value);
__fastcall TStringBuilder Append(const TObject * Value);
__fastcall TStringBuilder Append(const Shortint Value);
__fastcall TStringBuilder Append(const Single Value);
__fastcall TStringBuilder Append(const UnicodeString Value);
__fastcall TStringBuilder Append(const UInt64 Value);
__fastcall TStringBuilder Append(const TCharArray Value);
__fastcall TStringBuilder Append(const Word Value);
__fastcall TStringBuilder Append(const unsigned Value);
__fastcall TStringBuilder Append(const Char Value, int RepeatCount);
__fastcall TStringBuilder Append(const TCharArray Value, int StartIndex, int CharCount);
__fastcall TStringBuilder Append(const UnicodeString Value, int StartIndex, int Count);
```

Description

Use Append to append a string representation of the supplied argument to the current character array in this TStringBuilder (see page 4006) instance.

In the case where **Value** is a Char, **Value** is a Unicode character and repeatedly appended to the end of theTStringBuilder string **Count** number of times.

The two overloads that use the parameters **CharCount** or **Count** append a substring. **StartIndex** indicates the first character is the string **Value** to append, and **CharCount** or **Count** specifies the number of characters to append.

See Also

StringBuilder.Append (MSDN)

37.1.50.4.2 TStringBuilder.AppendFormat Method

Appends a number of objects to the current character array in this TStringBuilder (see page 4006) instance.

Pascal

```
function AppendFormat(const Format: string; const Args: array of const): TStringBuilder;
overload;
```

C++

```
__fastcall TStringBuilder AppendFormat(const UnicodeString Format, const array of const Args);
```

Description

Use AppendFormat to append a number of objects to the current character array in this TStringBuilder (see page 4006) instance. The objects are represented by a string of format specifications and an array of objects to format. There is a one to one relationship between the format specifications and the objects in the objects array.

Format (see page 4188) is a string of format specifications.

Args is the corresponding array of objects to format.

See Also

StringBuilder.AppendFormat (MSDN)

37.1.50.4.3 AppendLine Method

This is the overview for the AppendLine method overload.

Overload List

	Name	Description
	TStringBuilder.AppendLine () (see page 4016)	Appends a line terminator or a string and a line terminator to the current character array in this instance.
	TStringBuilder.AppendLine (string) (see page 4016)	Appends a line terminator or a string and a line terminator to the current character array in this instance.

37.1.50.4.3.1 TStringBuilder.AppendLine Method ()

Appends a line terminator or a string and a line terminator to the current character array in this instance.

Pascal

```
function AppendLine: TStringBuilder; overload;
function AppendLine(const Value: string): TStringBuilder; overload;
```

C++

```
__fastcall TStringBuilder AppendLine();
__fastcall TStringBuilder AppendLine(const UnicodeString Value);
```

Description

Use AppendLine to append a line terminator or a string and a line terminator to the current character array in this instance.

See Also

StringBuilder.AppendLine (MSDN)

37.1.50.4.4 TStringBuilder.CopyTo Method

Copys a part of the instance character array into a destination string.

Pascal

```
procedure CopyTo(SourceIndex: Integer; const Destination: TCharArray; DestinationIndex:
Integer; Count: Integer);
```

C++

```
__fastcall CopyTo(int SourceIndex, const TCharArray Destination, int DestinationIndex, int
Count);
```

Description

Use CopyTo to copy a part of the instance character array into a destination string.

SourceIndex specifies the first character in the instance character array to copy.

Destination indicates the character array to copy to.

DestinationIndex specifies the position in the destination string to insert the substring.

Count is the number of characters to copy.

See Also

StringBuilder.CopyTo (MSDN)

37.1.50.4.5 TStringBuilder.EnsureCapacity Method

Sets the Capacity (see page 4021) property to the supplied value, if the new capacity is larger than the old capacity.

Pascal

```
function EnsureCapacity(aCapacity: Integer): Integer;
```

C++

```
__fastcall int EnsureCapacity(int aCapacity);
```

Description

Sets the Capacity (see page 4021) property to the supplied value, if the new capacity is larger than the old capacity. Also allocates more memory for the character array if needed. Returns the new capacity.

See Also

Capacity (see page 4021)

StringBuilder.EnsureCapacity (MSDN)

37.1.50.4.6 TStringBuilder.Equals Method

Compares the character array in the Chars (see page 4022) property, the Capacity (see page 4021) property, and the MaxCapacity (see page 4022) property of this instance to that of a supplied TStringBuilder (see page 4006) instance.

Pascal

```
function Equals(StringBuilder: TStringBuilder): Boolean; reintroduce;
```

C++

```
__fastcall Boolean Equals(TStringBuilder StringBuilder);
```

Description

Use Equals to compare the character array in the Chars (see page 4022) property, the Capacity (see page 4021) property, and the MaxCapacity (see page 4022) property of this instance to that of a supplied TStringBuilder (see page 4006) instance.

See Also

Capacity (see page 4021)

Chars (see page 4022)

MaxCapacity (see page 4022)

StringBuilder.Equals (MSDN)

37.1.50.4.7 Insert Method

This is the overview for the Insert method overload.

Overload List

	Name	Description
	TStringBuilder.Insert (Integer, Boolean) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Byte) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Cardinal) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.

	TStringBuilder.Insert (Integer, Char) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Currency) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Double) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Int64) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Integer) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Shortint) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Single) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Smallint) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, string) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, string, Integer) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, TCharArray) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, TCharArray, Integer, Integer) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, TObject) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, UInt64) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.
	TStringBuilder.Insert (Integer, Word) (see page 4018)	Inserts a string representation of the object parameter at the Index provided.

37.1.50.4.7.1 TStringBuilder.Insert Method (Integer, Boolean)

Inserts a string representation of the object parameter at the Index provided.

Pascal

```
function Insert(Index: Integer; const Value: Boolean): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Byte): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Char): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Currency): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Double): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Smallint): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Integer): TStringBuilder; overload;
function Insert(Index: Integer; const Value: TCharArray): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Int64): TStringBuilder; overload;
function Insert(Index: Integer; const Value: TObject): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Shortint): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Single): TStringBuilder; overload;
function Insert(Index: Integer; const Value: string): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Word): TStringBuilder; overload;
function Insert(Index: Integer; const Value: Cardinal): TStringBuilder; overload;
function Insert(Index: Integer; const Value: UInt64): TStringBuilder; overload;
```

```
function Insert(Index: Integer; const Value: string; count: Integer): TStringBuilder; overload;
function Insert(Index: Integer; const Value: TCharArray; startIndex: Integer; charCount:
Integer): TStringBuilder; overload;
```

C++

```
__fastcall TStringBuilder Insert(int Index, const Boolean Value);
__fastcall TStringBuilder Insert(int Index, const Byte Value);
__fastcall TStringBuilder Insert(int Index, const Char Value);
__fastcall TStringBuilder Insert(int Index, const Currency Value);
__fastcall TStringBuilder Insert(int Index, const Double Value);
__fastcall TStringBuilder Insert(int Index, const Smallint Value);
__fastcall TStringBuilder Insert(int Index, const int Value);
__fastcall TStringBuilder Insert(int Index, const TCharArray Value);
__fastcall TStringBuilder Insert(int Index, const Int64 Value);
__fastcall TStringBuilder Insert(int Index, const TObject * Value);
__fastcall TStringBuilder Insert(int Index, const Shortint Value);
__fastcall TStringBuilder Insert(int Index, const Single Value);
__fastcall TStringBuilder Insert(int Index, const UnicodeString Value);
__fastcall TStringBuilder Insert(int Index, const Word Value);
__fastcall TStringBuilder Insert(int Index, const unsigned Value);
__fastcall TStringBuilder Insert(int Index, const UInt64 Value);
__fastcall TStringBuilder Insert(int Index, const UnicodeString Value, int count);
__fastcall TStringBuilder Insert(int Index, const TCharArray Value, int startIndex, int
charCount);
```

Description

Use Insert to insert a string representation of the object parameter at the Index provided.

In the overload that uses the parameter **count**, **count** provides a way to insert the object **count** number of times.

startIndex and **charCount** provide a way to insert a substring of the **Value** TCharArray (see page 4340) into the character array of this instance.

See Also

StringBuilder.Insert (MSDN)

37.1.50.4.8 TStringBuilder.Remove Method

Removes a substring from the character array of this instance.

Pascal

```
function Remove(StartIndex: Integer; RemLength: Integer): TStringBuilder;
```

C++

```
__fastcall TStringBuilder Remove(int StartIndex, int RemLength);
```

Description

Use Remove to remove a substring from the character array of this instance.

StartIndex indicates the index of the first character to remove.

RemLength indicates the number of characters to remove.

See Also

StringBuilder.Remove (MSDN)

37.1.50.4.9 Replace Method

This is the overview for the Replace method overload.

Overload List

	Name	Description
	TStringBuilder.Replace (Char, Char) (see page 4020)	Replaces all occurences of a specified string with another.
	TStringBuilder.Replace (Char, Char, Integer, Integer) (see page 4020)	Replaces all occurences of a specified string with another.
	TStringBuilder.Replace (string, string) (see page 4020)	Replaces all occurences of a specified string with another.
	TStringBuilder.Replace (string, string, Integer, Integer) (see page 4020)	Replaces all occurences of a specified string with another.

37.1.50.4.9.1 TStringBuilder.Replace Method (Char, Char)

Replaces all occurences of a specified string with another.

Pascal

```
function Replace(const OldChar: Char; const NewChar: Char): TStringBuilder; overload;
function Replace(const OldValue: string; const NewValue: string): TStringBuilder; overload;
function Replace(const OldChar: Char; const NewChar: Char; StartIndex: Integer; Count:
Integer): TStringBuilder; overload;
function Replace(const OldValue: string; const NewValue: string; StartIndex: Integer; Count:
Integer): TStringBuilder; overload;
```

C++

```
__fastcall TStringBuilder Replace(const Char OldChar, const Char NewChar);
__fastcall TStringBuilder Replace(const UnicodeString OldValue, const UnicodeString NewValue);
__fastcall TStringBuilder Replace(const Char OldChar, const Char NewChar, int StartIndex, int
Count);
__fastcall TStringBuilder Replace(const UnicodeString OldValue, const UnicodeString NewValue,
int StartIndex, int Count);
```

Description

Use Replace to replace all occurences of a specified string with another. Replace can replace a single **OldChar** with a **NewChar** everywhere, an **OldValue string** with a **NewValue** string, an **OldChar** with a **NewChar** everywhere in a specified substring, or an **OldValue** string with a **NewValue** string everywhere in a specified substring.

See Also

StringBuilder.Replace (MSDN)

37.1.50.4.10 ToString Method

This is the overview for the ToString method overload.

Overload List

	Name	Description
	TStringBuilder.ToString () (see page 4021)	Returns a string representation of the character array contained in this TStringBuilder (see page 4006) instance.
	TStringBuilder.ToString (Integer, Integer) (see page 4021)	Returns a string representation of the character array contained in this TStringBuilder (see page 4006) instance.

37.1.50.4.10.1 TStringBuilder.ToString Method ()

Returns a string representation of the character array contained in this TStringBuilder (see page 4006) instance.

Pascal

```
function ToString: string; override; overload;
function ToString(StartIndex: Integer; StrLength: Integer): string; overload; reintroduce;
```

C++

```
virtual __fastcall UnicodeString ToString();
__fastcall UnicodeString ToString(int StartIndex, int StrLength);
```

Description

Use ToString to create a string representation of the character array contained in this TStringBuilder (see page 4006) instance.

StartIndex and **StrLength** specify a substing of the character array to convert.

See Also

StringBuilder.ToString (MSDN)

37.1.50.5 TStringBuilder Properties

The properties of the TStringBuilder class are listed here.

Public Properties

	Name	Description
	Capacity (see page 4021)	Used to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4006) instance.
	Chars (see page 4022)	Used to get or set the characters stored in this TStringBuilder (see page 4006) instance.
	Length (see page 4022)	Used to get or set the size of the character array stored in this TStringBuilder (see page 4006) instance.
R	MaxCapacity (see page 4022)	Used to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4006) instance.

37.1.50.5.1 TStringBuilder.Capacity Property

Used to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4006) instance.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Use Capacity to get or set the maximum number of characters that can be stored in this TStringBuilder (see page 4006) instance. Capacity cannot be set higher than the value of MaxCapacity (see page 4022) and cannot be set lower than the current value of Length (see page 4022).

See Also

MaxCapacity (see page 4022)

Length (see page 4022)

StringBuilder.Capacity (MSDN)

37.1.50.5.2 TStringBuilder.Chars Property

Used to get or set the characters stored in this TStringBuilder (see page 4006) instance.

Pascal

```
property Chars [index: Integer]: Char;
```

C++

```
__property Char Chars[int index];
```

Description

Use Chars to get or set the characters stored in this TStringBuilder (see page 4006) instance.

index specifies the position of the character to access.

The value of this property is the character stored at location index.

See Also

StringBuilder.Chars (MSDN)

37.1.50.5.3 TStringBuilder.Length Property

Used to get or set the size of the character array stored in this TStringBuilder (see page 4006) instance.

Pascal

```
property Length: Integer;
```

C++

```
__property int Length;
```

Description

Use Length to get or set the size of the character array stored in this TStringBuilder (see page 4006) instance. If the value of **Length** is lowered, the character array is truncated. If the value is raised, the character array is padded with spaces and the Capacity (see page 4021) property is increased. An error occurs if an attempt is made to increase **Length** larger than the MaxCapacity (see page 4022) property.

See Also

Capacity (see page 4021)

MaxCapacity (see page 4022)

StringBuilder.Length (MSDN)

37.1.50.5.4 TStringBuilder.MaxCapacity Property

Used to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4006) instance.

Pascal

```
property MaxCapacity: Integer;
```

C++

```
__property int MaxCapacity;
```

Description

Use MaxCapacity to get the maximum size allowed of the character array stored in this TStringBuilder (see page 4006)

instance. **MaxCapacity** is set when the instance of TStringBuilder (see page 4006) is created and determines the maximum value to which the Capacity (see page 4021) property can be set.

See Also

Capacity (see page 4021)

StringBuilder.MaxCapacity (MSDN)

37.1.51 TThreadLocalCounter Class

Class Hierarchy

Pascal

```
TThreadLocalCounter = class;
```

C++

```
class TThreadLocalCounter;
```

File

SysUtils

Description

This is class SysUtils.TThreadLocalCounter.

37.1.51.1 TThreadLocalCounter Members

The following tables list the members exposed by TThreadLocalCounter.

Public Methods

	Name	Description
	Close (see page 4024)	This is Close, a member of class TThreadLocalCounter.
	Delete (see page 4024)	This is Delete, a member of class TThreadLocalCounter.
	Destroy (see page 4023)	This is Destroy, a member of class TThreadLocalCounter.
	Open (see page 4024)	This is Open, a member of class TThreadLocalCounter.

37.1.51.2 TThreadLocalCounter.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThreadLocalCounter();
```

Description

This is Destroy, a member of class TThreadLocalCounter.

37.1.51.3 TThreadLocalCounter Methods

The methods of the TThreadLocalCounter class are listed here.

Public Methods

	Name	Description
	Close (see page 4024)	This is Close, a member of class TThreadLocalCounter.
	Delete (see page 4024)	This is Delete, a member of class TThreadLocalCounter.
	Open (see page 4024)	This is Open, a member of class TThreadLocalCounter.

37.1.51.3.1 TThreadLocalCounter.Close Method

Pascal

```
procedure Close(var Thread: PThreadInfo);
```

C++

```
__fastcall Close(PThreadInfo Thread);
```

Description

This is Close, a member of class TThreadLocalCounter.

37.1.51.3.2 TThreadLocalCounter.Delete Method

Pascal

```
procedure Delete(var Thread: PThreadInfo);
```

C++

```
__fastcall Delete(PThreadInfo Thread);
```

Description

This is Delete, a member of class TThreadLocalCounter.

37.1.51.3.3 TThreadLocalCounter.Open Method

Pascal

```
procedure Open(var Thread: PThreadInfo);
```

C++

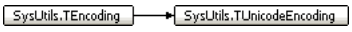
```
__fastcall Open(PThreadInfo Thread);
```

Description

This is Open, a member of class TThreadLocalCounter.

37.1.52 TUnicodeEncoding Class

Class Hierarchy



Pascal

```
TUnicodeEncoding = class(TEncoding);
```

C++

```
class TUnicodeEncoding : public TEncoding;
```

File

SysUtils

Description

This is class SysUtils.TUnicodeEncoding.

37.1.52.1 TUnicodeEncoding Members

The following tables list the members exposed by TUnicodeEncoding.

Public Methods

	Name	Description
	Convert (see page 3974)	This is the overview for the Convert method overload.
V	Create (see page 4026)	This is Create, a member of class TUnicodeEncoding.
	FreeEncodings (see page 3975)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 3976)	Determines the encoding for a given buffer.
	GetEncoding (see page 3980)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 4028)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4028)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4029)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 3981)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 3984)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 3985)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 3985)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 3985)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 3986)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 3986)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 3986)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 3987)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 3973)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 3974)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 4026)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 3976)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 4027)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 3977)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 4027)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 3978)	This is the overview for the GetCharCount method overload.

	GetChars (see page 4028)	Decodes a set of bytes into a set of characters.
	GetChars (see page 3979)	This is the overview for the GetChars method overload.

37.1.52.2 TUnicodeEncoding.Create Constructor

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TUnicodeEncoding();
```

Description

This is Create, a member of class TUnicodeEncoding.

37.1.52.3 TUnicodeEncoding Methods

The methods of the TUnicodeEncoding class are listed here.

Public Methods

	Name	Description
	GetMaxByteCount (see page 4028)	Calculates the maximum bytes generated to encode the specified number of characters.
	GetMaxCharCount (see page 4028)	Calculates the maximum number of characters generated to decode a specified number of bytes.
	GetPreamble (see page 4029)	Returns a preamble in a byte array that specifies the encoding used.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4026)	Returns the number of bytes generated by encoding Chars.
	GetBytes (see page 4027)	Encodes a set of characters into a sequence of bytes.
	GetCharCount (see page 4027)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetChars (see page 4028)	Decodes a set of bytes into a set of characters.

37.1.52.3.1 TUnicodeEncoding.GetByteCount Method

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

37.1.52.3.2 TUnicodeEncoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

37.1.52.3.3 TUnicodeEncoding.GetCharCount Method

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

37.1.52.3.4 TUnicodeEncoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

37.1.52.3.5 TUnicodeEncoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

37.1.52.3.6 TUnicodeEncoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

37.1.52.3.7 TUnicodeEncoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

37.1.53 TUTF7Encoding Class

Class Hierarchy

Pascal

```
TUTF7Encoding = class(TMBCSEncoding);
```

C++

```
class TUTF7Encoding : public TMBCSEncoding;
```

File

SysUtils

Description

This is class SysUtils.TUTF7Encoding.

37.1.53.1 TUTF7Encoding Members

The following tables list the members exposed by TUTF7Encoding.

Public Methods

	Name	Description
	Convert (see page 3974)	This is the overview for the Convert method overload.
V	Create (see page 4030)	This is Create, a member of class TUTF7Encoding.
V	Create (see page 3993)	This is the overview for the Create constructor overload.
	FreeEncodings (see page 3975)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 3976)	Determines the encoding for a given buffer.
	GetEncoding (see page 3980)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 4033)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4033)	Calculates the maximum number of characters generated to decode a specified number of bytes.
A	GetPreamble (see page 3981)	Returns a preamble in a byte array that specifies the encoding used.
V	GetPreamble (see page 3997)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 3981)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 3984)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 3985)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 3985)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 3985)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 3986)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 3986)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 3986)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 3987)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 3973)	This is FIsSingleByte, a member of class TEncoding.
	FMaxCharSize (see page 3974)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 4031)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 3976)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 4031)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 3977)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 4032)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 3978)	This is the overview for the GetCharCount method overload.
V	GetChars (see page 4032)	Decodes a set of bytes into a set of characters.
A	GetChars (see page 3979)	This is the overview for the GetChars method overload.

37.1.53.2 TUTF7Encoding.Create Constructor

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TUTF7Encoding();
```

Description

This is Create, a member of class TUTF7Encoding.

37.1.53.3 TUTF7Encoding Methods

The methods of the TUTF7Encoding class are listed here.

Public Methods

	Name	Description
	GetMaxByteCount (see page 4033)	Calculates the maximum bytes generated to encode the specified number of characters.
	GetMaxCharCount (see page 4033)	Calculates the maximum number of characters generated to decode a specified number of bytes.

Strict protected Methods

	Name	Description
	GetByteCount (see page 4031)	Returns the number of bytes generated by encoding Chars.
	GetBytes (see page 4031)	Encodes a set of characters into a sequence of bytes.
	GetCharCount (see page 4032)	Calcuates a number of characters generated by decoding a sequence of bytes.
	GetChars (see page 4032)	Decodes a set of bytes into a set of characters.

37.1.53.3.1 TUTF7Encoding.GetByteCount Method

Returns the number of bytes generated by encoding Chars.

Pascal

```
function GetByteCount(Chars: PChar; CharCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetByteCount(const char * Chars, int CharCount);
```

Description

Returns the number of bytes generated by encoding Chars. Note that the number of bytes in a string is not necessarily exactly proportional to the number of characters in a given character array or string.

The **Chars** parameter can be a character array or a character pointer containing the bytes to be counted.

The **S** parameter refers to a UnicodeString from which the Byte count will be extracted.

The **CharCount** parameter specifies the number of characters to encode.

The **CharIndex** parameter indicates the index within the Chars array where counting should begin.

The **CharCount** parameter indicates the number of characters that should be included when counting the bytes.

The Return Value is the number of bytes in the passed Chars or S parameter.

See Also

Encoding.GetByteCount (MSDN)

37.1.53.3.2 TUTF7Encoding.GetBytes Method

Encodes a set of characters into a sequence of bytes.

Pascal

```
function GetBytes(Chars: PChar; CharCount: Integer; Bytes: PByte; ByteCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetBytes(const char * Chars, int CharCount, PByte Bytes, int ByteCount);
```

Description

TEncoding.GetBytes encodes Chars parameter into a sequence of bytes.

Chars can be a character array or a character pointer which will be encoded into a sequence of bytes.

S is a string of type UnicodeString to be encoded into a sequence of bytes.

The **CharCount** parameter specifies the number of characters to encode.

Bytes specifies the location to write the sequence of bytes.

ByteCount is the maximum number of bytes that can be written to the given location.

The **Return Value** is the number of bytes actually encoded or the actual TBytes sequence of bytes.

See Also

Encoding.GetBytes (MSDN)

37.1.53.3.3 TUTF7Encoding.GetCharCount Method

Calcuates a number of characters generated by decoding a sequence of bytes.

Pascal

```
function GetCharCount(Bytes: PByte; ByteCount: Integer): Integer; override; overload;
```

C++

```
virtual __fastcall int GetCharCount(PByte Bytes, int ByteCount);
```

Description

TEncoding.GetCharCount generates a number of characters by decoding **Bytes**.

Bytes can be a byte array, or a byte pointer.

The **ByteCount** parameter specifies the number of bytes to decode.

The **Return Value** is the number of characters decoded.

See Also

Encoding.GetCharCount (MSDN)

37.1.53.3.4 TUTF7Encoding.GetChars Method

Decodes a set of bytes into a set of characters.

Pascal

```
function GetChars(Bytes: PByte; ByteCount: Integer; Chars: PChar; CharCount: Integer):
Integer; override; overload;
```

C++

```
virtual __fastcall int GetChars(PByte Bytes, int ByteCount, const char * Chars, int CharCount);
```

Description

TEncoding.GetChars generates a number of characters by decoding Bytes.

Bytes can be a byte pointer or a byte array.

ByteIndex specifies the first byte to decode. If not present, the index is 0.

ByteCount specifies the number of bytes to decode. If not present, all bytes will be decoded.

Chars is a pointer to the destination of the decoded characters.

CharIndex specifies the maximum number of characters to store.

The **Return Value** is either a character array of the decoded characters, or the number of characters decoded.

See Also

Encoding.GetChars (MSDN)

37.1.53.3.5 TUTF7Encoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

37.1.53.3.6 TUTF7Encoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

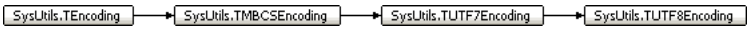
The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

37.1.54 TUTF8Encoding Class

Class Hierarchy



Pascal

```
TUTF8Encoding = class(TUTF7Encoding);
```

C++

```
class TUTF8Encoding : public TUTF7Encoding;
```

File

SysUtils

Description

This is class SysUtils.TUTF8Encoding.

37.1.54.1 TUTF8Encoding Members

The following tables list the members exposed by TUTF8Encoding.

Public Methods

	Name	Description
	Convert (see page 3974)	This is the overview for the Convert method overload.
V	Create (see page 4035)	This is Create, a member of class TUTF8Encoding.
V	Create (see page 3993)	This is the overview for the Create constructor overload.
	FreeEncodings (see page 3975)	Frees the internal standard encoding instances.
	GetBufferEncoding (see page 3976)	Determines the encoding for a given buffer.
	GetEncoding (see page 3980)	Returns the encoding used for the code page supplied.
V	GetMaxByteCount (see page 4035)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4036)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4036)	Returns a preamble in a byte array that specifies the encoding used.
	GetString (see page 3981)	This is the overview for the GetString method overload.
	IsStandardEncoding (see page 3984)	Reports if this instance is a standard encoding instance.

Public Properties

	Name	Description
R	ASCII (see page 3985)	Returns an encoding for the ASCII character set.
R	BigEndianUnicode (see page 3985)	Returns an encoding for the UTF-16 format with big-endian byte order.
R	Default (see page 3985)	Returns an encoding for the operating system's default ANSI code page.
R	IsSingleByte (see page 3986)	This boolean specifies if the current encoding is of single-byte format.
R	Unicode (see page 3986)	Returns an encoding for the UTF-16 format with little-endian byte order.
R	UTF7 (see page 3986)	Returns an encoding for the UTF-7 format.
R	UTF8 (see page 3987)	Returns an encoding for the UTF-8 format.

Strict protected Fields

	Name	Description
	FIsSingleByte (see page 3973)	This is FIsSingleByte, a member of class TEncoding.

	FMaxCharSize (see page 3974)	This is FMaxCharSize, a member of class TEncoding.

Strict protected Methods

	Name	Description
V	GetByteCount (see page 3994)	Returns the number of bytes generated by encoding Chars.
V	GetByteCount (see page 4031)	Returns the number of bytes generated by encoding Chars.
A	GetByteCount (see page 3976)	This is the overview for the GetByteCount method overload.
V	GetBytes (see page 3995)	Encodes a set of characters into a sequence of bytes.
V	GetBytes (see page 4031)	Encodes a set of characters into a sequence of bytes.
A	GetBytes (see page 3977)	This is the overview for the GetBytes method overload.
V	GetCharCount (see page 3995)	Calcuates a number of characters generated by decoding a sequence of bytes.
V	GetCharCount (see page 4032)	Calcuates a number of characters generated by decoding a sequence of bytes.
A	GetCharCount (see page 3978)	This is the overview for the GetCharCount method overload.
V	GetChars (see page 3996)	Decodes a set of bytes into a set of characters.
V	GetChars (see page 4032)	Decodes a set of bytes into a set of characters.
A	GetChars (see page 3979)	This is the overview for the GetChars method overload.

37.1.54.2 TUTF8Encoding.Create Constructor

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TUTF8Encoding();
```

Description

This is Create, a member of class TUTF8Encoding.

37.1.54.3 TUTF8Encoding Methods

The methods of the TUTF8Encoding class are listed here.

Public Methods

	Name	Description
V	GetMaxByteCount (see page 4035)	Calculates the maximum bytes generated to encode the specified number of characters.
V	GetMaxCharCount (see page 4036)	Calculates the maximum number of characters generated to decode a specified number of bytes.
V	GetPreamble (see page 4036)	Returns a preamble in a byte array that specifies the encoding used.

37.1.54.3.1 TUTF8Encoding.GetMaxByteCount Method

Calculates the maximum bytes generated to encode the specified number of characters.

Pascal

```
function GetMaxByteCount(CharCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxByteCount(int CharCount);
```

Description

TEncoding.GetMaxByteCount calculates the maximum bytes generated to encode CharCount number of characters.

CharCount must be a positive integer.

The **Return Value** is the number of bytes calculated.

See Also

Encoding.GetMaxByteCount (MSDN)

37.1.54.3.2 TUTF8Encoding.GetMaxCharCount Method

Calculates the maximum number of characters generated to decode a specified number of bytes.

Pascal

```
function GetMaxCharCount(ByteCount: Integer): Integer; override;
```

C++

```
virtual __fastcall int GetMaxCharCount(int ByteCount);
```

Description

TEncoding.GetMaxCharCount calculates the maximum number of characters generated to decode **ByteCount** number of bytes.

The **Return Value** is the number of characters calculated.

See Also

Encoding.GetMaxCharCount (MSDN)

37.1.54.3.3 TUTF8Encoding.GetPreamble Method

Returns a preamble in a byte array that specifies the encoding used.

Pascal

```
function GetPreamble: TBytes; override;
```

C++

```
virtual __fastcall TBytes GetPreamble();
```

Description

TEncoding.GetPreamble returns an array of bytes of type TBytes that contains the preamble required for the encoding used. If no preamble is required a zero length array is returned.

See Also

Encoding.GetPreamble (MSDN)

37.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IReadWriteSync (see page 4037)	This is class SysUtils.IReadWriteSync.

37.2.1 IReadWriteSync Interface

Class Hierarchy

SysUtils.IReadWriteSync

Pascal

```
IReadWriteSync = interface;
```

C++

```
__interface IReadWriteSync;
```

File

SysUtils

Description

This is class SysUtils.IReadWriteSync.

37.2.1.1 IReadWriteSync Members

The following tables list the members exposed by IReadWriteSync.

Public Methods

	Name	Description
	BeginRead (see page 4038)	This is BeginRead, a member of class IReadWriteSync.
	BeginWrite (see page 4038)	This is BeginWrite, a member of class IReadWriteSync.
	EndRead (see page 4038)	This is EndRead, a member of class IReadWriteSync.
	EndWrite (see page 4038)	This is EndWrite, a member of class IReadWriteSync.

37.2.1.2 IReadWriteSync Methods

The methods of the IReadWriteSync class are listed here.

Public Methods

	Name	Description
	BeginRead (see page 4038)	This is BeginRead, a member of class IReadWriteSync.
	BeginWrite (see page 4038)	This is BeginWrite, a member of class IReadWriteSync.
	EndRead (see page 4038)	This is EndRead, a member of class IReadWriteSync.

	EndWrite (see page 4038)	This is EndWrite, a member of class IReadWriteSync.

37.2.1.2.1 IReadWriteSync.BeginRead Method

Pascal

```
procedure BeginRead;
```

C++

```
__fastcall BeginRead();
```

Description

This is BeginRead, a member of class IReadWriteSync.

37.2.1.2.2 IReadWriteSync.BeginWrite Method

Pascal

```
function BeginWrite: Boolean;
```

C++

```
__fastcall Boolean BeginWrite();
```

Description

This is BeginWrite, a member of class IReadWriteSync.

37.2.1.2.3 IReadWriteSync.EndRead Method

Pascal

```
procedure EndRead;
```

C++

```
__fastcall EndRead();
```

Description

This is EndRead, a member of class IReadWriteSync.

37.2.1.2.4 IReadWriteSync.EndWrite Method

Pascal

```
procedure EndWrite;
```

C++

```
__fastcall EndWrite();
```

Description

This is EndWrite, a member of class IReadWriteSync.

37.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Abort (see page 4047)	Raises a silent exception.
	AddExitProc (see page 4048)	Adds a procedure to the exit list.
	AddTerminateProc (see page 4048)	Adds a terminate procedure to the system list of termination procedures.
	AdjustLineBreaks (see page 4049)	Adjusts line break characters to reflect Windows or Linux style.
	AnsiCompareFileName (see page 4049)	Compares file names based on the current locale.
	AnsiCompareStr (see page 4050)	Compares strings based on the current locale with case sensitivity.
	AnsiCompareText (see page 4051)	Compares strings based on the current locale without case sensitivity.
	AnsiDequotedStr (see page 4051)	Converts a quoted string into an unquoted string
	AnsiExtractQuotedStr (see page 4052)	Converts a quoted string into an unquoted string.
	AnsiExtractQuotedStr (see page 4052)	Converts a quoted string into an unquoted string.
	AnsiLastChar (see page 4052)	Returns a pointer to the last character in a string.
	AnsiLastChar (see page 4052)	Returns a pointer to the last character in a string.
	AnsiLowerCase (see page 4053)	Returns a string that is a copy of the given string converted to lower case.
	AnsiLowerCaseFileName (see page 4053)	Converts a file name to lower case.
	AnsiPos (see page 4054)	Locates the position of a substring.
	AnsiQuotedStr (see page 4054)	Returns the quoted version of a string.
	AnsiSameStr (see page 4055)	Compares strings based on the current locale with case sensitivity.
	AnsiSameText (see page 4055)	Compares strings based on the current locale without case sensitivity.
	AnsiStrAlloc (see page 4056)	This is function SysUtils.AnsiStrAlloc.
	AnsiStrComp (see page 4056)	Compares null-terminated character strings.
	AnsiStrComp (see page 4056)	Compares null-terminated character strings.
	AnsiStrIComp (see page 4057)	Compares null terminated character strings case insensitively.
	AnsiStrIComp (see page 4057)	Compares null terminated character strings case insensitively.
	AnsiStrLastChar (see page 4057)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLastChar (see page 4057)	Returns a pointer to the last character in a null-terminated string.
	AnsiStrLComp (see page 4058)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLComp (see page 4058)	Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.
	AnsiStrLIComp (see page 4058)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLIComp (see page 4058)	Compares two strings, case-insensitively, up to the first MaxLen bytes.
	AnsiStrLower (see page 4059)	Converts all characters in a null-terminated string to lower case.
	AnsiStrLower (see page 4059)	Converts all characters in a null-terminated string to lower case.
	AnsiStrPos (see page 4060)	Returns a pointer to the first occurrence of a substring.
	AnsiStrPos (see page 4060)	Returns a pointer to the first occurrence of a substring.
	AnsiStrRScan (see page 4060)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrRScan (see page 4060)	Returns a pointer to the last occurrence of a specified character in a specified string.
	AnsiStrScan (see page 4061)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrScan (see page 4061)	Returns a pointer to first occurrence of a character in a string.
	AnsiStrUpper (see page 4061)	Converts all characters in a null-terminated string to upper case.
	AnsiStrUpper (see page 4061)	Converts all characters in a null-terminated string to upper case.
	AnsiUpperCase (see page 4062)	Converts a string to upper case.
	AnsiUpperCaseFileName (see page 4062)	Converts a file name to upper case.
	AppendStr (see page 4063)	Appends a dynamically allocated string to an existing string.
	AssignStr (see page 4063)	Assigns a new dynamically allocated string to a specified pointer.

	Beep (see page 4063)	Emits a message beep.
	BoolToStr (see page 4066)	Converts a Boolean value to a string.
	ByteLength (see page 4068)	This is function SysUtils.ByteLength.
	BytesOf (see page 4068)	Converts a string into a byte array.
	BytesOf (see page 4068)	Converts a string into a byte array.
	BytesOf (see page 4068)	Converts a string into a byte array.
	BytesOf (see page 4068)	Converts a string into a byte array.
	ByteToCharIndex (see page 4069)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharIndex (see page 4069)	Returns the position of the character that contains a specified byte in a string.
	ByteToCharLen (see page 4070)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteToCharLen (see page 4070)	Returns the number of characters fully contained in the first MaxLen bytes of a string.
	ByteType (see page 4070)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	ByteType (see page 4070)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	CallTerminateProcs (see page 4071)	Calls all of the functions in the termination procedure list.
	ChangeFileExt (see page 4071)	Changes the extension of a file name.
	ChangeFilePath (see page 4073)	Changes the path of a file name.
	CharInSet (see page 4073)	CharInSet checks whether a given character is contained within a set.
	CharInSet (see page 4073)	CharInSet checks whether a given character is contained within a set.
	CharLength (see page 4074)	Returns number of bytes used by character.
	CharLength (see page 4074)	Returns number of bytes used by character.
	CharToByteIndex (see page 4075)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteIndex (see page 4075)	Returns the 1-based index of the first byte of a specified character in a string.
	CharToByteLen (see page 4076)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToByteLen (see page 4076)	Returns the length, in bytes, of the first MaxLen characters of a string.
	CharToElementIndex (see page 4076)	This is function SysUtils.CharToElementIndex.
	CharToElementIndex (see page 4076)	This is function SysUtils.CharToElementIndex.
	CharToElementLen (see page 4077)	This is function SysUtils.CharToElementLen.
	CharToElementLen (see page 4077)	This is function SysUtils.CharToElementLen.
	CheckWin32Version (see page 4077)	Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 3159).
	CompareMem (see page 4078)	Performs a binary comparison of two memory images.
	CompareStr (see page 4078)	Compares two strings case sensitively.
	CompareStr (see page 4078)	Compares two strings case sensitively.
	CompareText (see page 4079)	Compares two strings by ordinal value without case sensitivity.
	CompareText (see page 4079)	Compares two strings by ordinal value without case sensitivity.
	CreateDir (see page 4083)	Creates a new directory.
	CreateGUID (see page 4084)	Creates a globally unique identifier.
	CurrentYear (see page 4086)	Returns the current year.
	CurrToStr (see page 4086)	Formats a Currency (see page 3235) value as a string.
	CurrToStr (see page 4086)	Formats a Currency (see page 3235) value as a string.
	CurrToStrF (see page 4086)	Converts a Currency (see page 3235) value to a string, using a specified format.
	CurrToStrF (see page 4086)	Converts a Currency (see page 3235) value to a string, using a specified format.
	Date (see page 4087)	Returns the current date.
	DateTimeToFileDate (see page 4088)	Converts a TDateTime (see page 3323) value to an OS timestamp value.
	DateTimeToStr (see page 4089)	Converts a TDateTime (see page 3323) value to a string.
	DateTimeToStr (see page 4089)	Converts a TDateTime (see page 3323) value to a string.
	DateTimeToString (see page 4089)	Converts a TDateTime (see page 3323) value to a string using a specified Format (see page 4188).
	DateTimeToString (see page 4089)	Converts a TDateTime (see page 3323) value to a string using a specified Format (see page 4188).

	DateTimeToSystemTime (see page 4091)	Converts a TDateTime (see page 3323) value into the Win32 API's system time type.
	DateTimeToTimeStamp (see page 4092)	Converts a TDateTime (see page 3323) value into the corresponding TTimeStamp (see page 4334) value.
	DateToStr (see page 4092)	Converts a TDateTime (see page 3323) value to a string.
	DateToStr (see page 4092)	Converts a TDateTime (see page 3323) value to a string.
	DayOfWeek (see page 4094)	Returns the day of the week for a specified date.
	DecodeDate (see page 4095)	Returns Year, Month, and Day values for a TDateTime (see page 3323) value.
	DecodeDateFully (see page 4096)	Returns Year, Month, and Day, and Day-of-Week values for a TDateTime (see page 3323) value.
	DecodeTime (see page 4097)	Breaks a TDateTime (see page 3323) value into hours, minutes, seconds, and milliseconds.
	DelegatesEqual (see page 4098)	This is function SysUtils.DelegatesEqual.
	DeleteFile (see page 4098)	Deletes a file from disk.
	DirectoryExists (see page 4100)	Determines whether a specified directory exists.
	DiskFree (see page 4101)	Returns the number of free bytes on a specified drive.
	DiskSize (see page 4102)	Returns the size, in bytes, of a specified drive.
	DisposeStr (see page 4103)	Disposes of a string pointer that was allocated using NewStr (see page 4227).
	ElementToCharIndex (see page 4104)	This is function SysUtils.ElementToCharIndex.
	ElementToCharIndex (see page 4104)	This is function SysUtils.ElementToCharIndex.
	ElementToCharLen (see page 4104)	This is function SysUtils.ElementToCharLen.
	ElementToCharLen (see page 4105)	This is function SysUtils.ElementToCharLen.
	EncodeDate (see page 4105)	Returns a TDateTime (see page 3323) value that represents a specified Year, Month, and Day.
	EncodeTime (see page 4106)	Returns a TDateTime (see page 3323) value for a specified Hour, Min, Sec, and MSec.
	ExceptionErrorMessage (see page 4107)	Formats a standard error message.
	ExcludeTrailingBackslash (see page 4108)	Returns a path name without a trailing delimiter.
	ExcludeTrailingPathDelimiter (see page 4109)	Returns a path name without a trailing delimiter.
	ExpandFileName (see page 4109)	Returns the full path name for a relative file name.
	ExpandFileNameCase (see page 4110)	Returns the full path name for a relative file name on a case-sensitive file system.
	ExpandUNCFileName (see page 4111)	Returns the full path of a file name in UNC format, if appropriate.
	ExtractFileDir (see page 4112)	Extracts the drive and directory parts from FileName.
	ExtractFileDrive (see page 4112)	Returns the drive portion of a file name.
	ExtractFileExt (see page 4113)	Returns the extension portions of a file name.
	ExtractFileName (see page 4113)	Extracts the name and extension parts of a file name.
	ExtractFilePath (see page 4115)	Returns the drive and directory portions of a file name.
	ExtractRelativePath (see page 4117)	Returns a relative path name, relative to a specific base directory.
	ExtractShortPathName (see page 4118)	Converts a file name to the short 8.3 form.
	FileAge (see page 4119)	Returns the OS timestamp of a file.
	FileAge (see page 4119)	Returns the OS timestamp of a file.
	FileClose (see page 4121)	Closes a specified file.
	FileCreate (see page 4128)	Creates a new file.
	FileCreate (see page 4128)	Creates a new file.
	FileDateToDateTime (see page 4130)	Converts an OS timestamp value to TDateTime (see page 3323) value.
	FileExists (see page 4133)	Tests if a specified file exists.
	FileGetAttr (see page 4139)	Returns the file attributes of FileName.
	FileGetDate (see page 4142)	Returns an OS timestamp for a specified file.
	FileIsReadOnly (see page 4142)	Reports whether a file is read-only.
	FileOpen (see page 4143)	Opens a specified file using a specified access mode.
	FileRead (see page 4150)	Reads a specified number of bytes from a file.
	FileSearch (see page 4156)	Searches a specified directory path for a file.
	FileSeek (see page 4157)	Repositions read/write point.
	FileSeek (see page 4157)	Repositions read/write point.
	FileSetAttr (see page 4162)	Sets the file attributes of a specified file.
	FileSetDate (see page 4164)	Sets the OS time stamp for a specified file.

	FileSetDate (see page 4164)	Sets the OS time stamp for a specified file.
	FileSetReadOnly (see page 4165)	Sets the file permissions to read-only.
	FileWrite (see page 4166)	Writes the contents of a buffer to the current position in a file.
	FinalizePackage (see page 4172)	Finalizes the specified package (.bpl file).
	FindClose (see page 4172)	Releases memory allocated by FindFirst (see page 4175).
	FindCmdLineSwitch (see page 4174)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 4174)	Determines whether a string was passed as a command line argument to the application.
	FindCmdLineSwitch (see page 4174)	Determines whether a string was passed as a command line argument to the application.
	FindFirst (see page 4175)	Searches for the first instance of a file name with a given set of attributes in a specified directory.
	FindNext (see page 4177)	Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 4175).
	FloatToCurr (see page 4179)	Converts a floating-point value to a System::Currency (see page 3235) value.
	FloatToDateTime (see page 4179)	Converts a floating-point value to a TDateTime (see page 3323) value.
	FloatToDecimal (see page 4180)	Converts a floating-point value to a decimal representation.
	FloatToStr (see page 4180)	Converts a floating point value to a string.
	FloatToStr (see page 4180)	Converts a floating point value to a string.
	FloatToStrF (see page 4182)	Converts a floating point value to a string, using a specified Format (see page 4188), Precision, and Digits.
	FloatToStrF (see page 4182)	Converts a floating point value to a string, using a specified Format (see page 4188), Precision, and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToText (see page 4183)	Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FloatToTextFmt (see page 4184)	Converts a floating-point value to an unterminated character string, using a specified format.
	FmtLoadStr (see page 4185)	Returns formatted output using a resourced format string.
	FmtStr (see page 4185)	Assembles a formatted string using a format string and an array of arguments.
	FmtStr (see page 4185)	Assembles a formatted string using a format string and an array of arguments.
	ForceDirectories (see page 4186)	Creates a new directory, including the creation of parent directories as needed.
	Format (see page 4188)	Returns a formatted string assembled from a format string and an array of arguments.
	Format (see page 4188)	Returns a formatted string assembled from a format string and an array of arguments.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatBuf (see page 4193)	Formats the arguments from an array, placing the result in a buffer.
	FormatCurr (see page 4194)	Formats a System::Currency (see page 3235) object.
	FormatCurr (see page 4194)	Formats a System::Currency (see page 3235) object.
	FormatDateTime (see page 4195)	Formats a TDateTime (see page 3323) value.
	FormatDateTime (see page 4195)	Formats a TDateTime (see page 3323) value.
	FormatFloat (see page 4197)	Formats a floating point value.

	FormatFloat (see page 4197)	Formats a floating point value.
	FreeAndNil (see page 4201)	Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).
	GDAL (see page 4202)	Get Delphi Access Licence.
	GetCurrentDir (see page 4202)	Returns the name of the current directory.
	GetEnvironmentVariable (see page 4204)	Retrieves an environment variable value.
	GetFileVersion (see page 4205)	Returns file version.
	GetFormatSettings (see page 4206)	Resets the date and number format parameters to initial values.
	GetLocaleChar (see page 4206)	Retrieves the first character of a localisation value.
	GetLocaleFormatSettings (see page 4206)	Populates a TFormatSettings (see page 4326) data structure.
	GetLocaleStr (see page 4207)	Retrieves a localisation value.
	GetModuleName (see page 4207)	Returns the fully qualified name for a module, given its handle.
	GetPackageDescription (see page 4208)	Returns the description resource from a package.
	GetPackageInfo (see page 4209)	Enumerates all the units and required packages for a package.
	GetTime (see page 4209)	Returns the current time.
	GUIDToString (see page 4210)	Converts a class identifier GUID to a string.
	HashName (see page 4212)	This is function SysUtils.HashName.
	IncAMonth (see page 4212)	Increments date data by one month.
	IncludeTrailingBackslash (see page 4213)	Ensures path name ends with delimiter
	IncludeTrailingPathDelimiter (see page 4213)	Ensures path name ends with delimiter.
	IncMonth (see page 4214)	Returns a date shifted by a specified number of months.
	InitializePackage (see page 4215)	Validates and initializes a given package.
	InitializePackage (see page 4215)	Validates and initializes a given package.
	IntToHex (see page 4215)	Returns the hex representation of an integer.
	IntToHex (see page 4215)	Returns the hex representation of an integer.
	IntToStr (see page 4216)	Converts an integer to a string.
	IntToStr (see page 4216)	Converts an integer to a string.
	IsAssembly (see page 4217)	Determines whether a dll is an Assembly.
	IsDelimiter (see page 4218)	Indicates whether a specified character in a string matches one of a set of delimiters.
	IsEqualGUID (see page 4218)	Indicates whether two TGUID (see page 3718) values are the same.
	IsLeadChar (see page 4220)	This is function SysUtils.IsLeadChar.
	IsLeadChar (see page 4220)	This is function SysUtils.IsLeadChar.
	IsLeapYear (see page 4220)	Indicates whether a specified year is a leap year.
	IsPathDelimiter (see page 4222)	Indicates whether the byte at a specified position in a string is the path delimiter.
	IsValidIdent (see page 4222)	Determines whether a string conforms to Delphi identifier rules.
	Languages (see page 4223)	Lists the locales for which support is available.
	LastDelimiter (see page 4223)	Returns the byte index of the last character that matches any character in a specified set of delimiters.
	LoadPackage (see page 4224)	Loads a specified package.
	LoadPackage (see page 4224)	Loads a specified package.
	LoadStr (see page 4224)	Loads a string resource from the application's executable file.
	LowerCase (see page 4225)	Converts an ASCII string to lowercase.
	LowerCase (see page 4225)	Converts an ASCII string to lowercase.
	MSecsToTimeStamp (see page 4226)	Converts a specified number of milliseconds into a TTimeStamp (see page 4334) value.
	NewStr (see page 4227)	Allocates a string on the heap.
	NextCharIndex (see page 4227)	Returns the byte index of the next character.
	NextCharIndex (see page 4227)	Returns the byte index of the next character.
	Now (see page 4227)	Returns the current date and time.
	OutOfMemoryError (see page 4228)	Raises an EOutOfMemory (see page 3942) exception.
	PlatformBytesOf (see page 4229)	This is function SysUtils.PlatformBytesOf.
	PlatformStringOf (see page 4229)	This is function SysUtils.PlatformStringOf.
	QuotedStr (see page 4229)	Returns the quoted version of a string.
	RaiseLastOSError (see page 4230)	Raises an exception for the last occurring OS or system library error.
	RaiseLastOSError (see page 4230)	Raises an exception for the last occurring OS or system library error.

	RaiseLastWin32Error (see page 4231)	Raises an exception for the last occurring Win32 error.
	RCS (see page 4232)	Check for Delphi Client/Server Licence.
	RemoveDir (see page 4232)	Deletes an existing empty directory.
	RenameFile (see page 4233)	Changes a file name.
	ReplaceDate (see page 4236)	Replaces the date portion of a TDateTime (see page 3323) value with a specified date.
	ReplaceTime (see page 4236)	Replaces the time portion of a TDateTime (see page 3323) value with a specified time.
	RPR (see page 4237)	Check for Delphi Professional Licence.
	SafeLoadLibrary (see page 4238)	Safe and simple library loading.
	SameFileName (see page 4238)	Compares file names based on the current locale.
	SameStr (see page 4239)	Compares two strings by ordinal value with case sensitivity.
	SameStr (see page 4239)	Compares two strings by ordinal value with case sensitivity.
	SameText (see page 4239)	Compares two strings by ordinal value without case sensitivity.
	SameText (see page 4239)	Compares two strings by ordinal value without case sensitivity.
	SetCurrentDir (see page 4240)	Sets the current directory.
	ShowException (see page 4240)	Displays an exception message with its physical address
	Sleep (see page 4241)	Delay program execution for a specified number of milliseconds.
	StrAlloc (see page 4241)	Allocates a buffer for a null-terminated string and returns a pointer to its first character.
	StrBufSize (see page 4242)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4241).
	StrBufSize (see page 4242)	Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4241).
	StrByteType (see page 4242)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrByteType (see page 4242)	Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.
	StrCat (see page 4243)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCat (see page 4243)	Appends a copy of Source to the end of Dest and returns the concatenated string.
	StrCharLength (see page 4244)	Returns the length, in bytes, of a character.
	StrCharLength (see page 4244)	Returns the length, in bytes, of a character.
	StrComp (see page 4244)	Compares two strings with case sensitivity.
	StrComp (see page 4244)	Compares two strings with case sensitivity.
	StrCopy (see page 4246)	Copies a null-terminated string.
	StrCopy (see page 4246)	Copies a null-terminated string.
	StrDispose (see page 4247)	Disposes of a string.
	StrDispose (see page 4247)	Disposes of a string.
	StrECopy (see page 4248)	Copies null-terminated string.
	StrECopy (see page 4248)	Copies null-terminated string.
	StrEnd (see page 4249)	Returns a pointer to the end of a null terminated string.
	StrEnd (see page 4249)	Returns a pointer to the end of a null terminated string.
	StrFmt (see page 4250)	Formats entries in an array.
	StrFmt (see page 4250)	Formats entries in an array.
	StrFmt (see page 4250)	Formats entries in an array.
	StrFmt (see page 4250)	Formats entries in an array.
	StrIComp (see page 4251)	Compares two strings without case sensitivity.
	StrIComp (see page 4251)	Compares two strings without case sensitivity.
	StringOf (see page 4253)	This is function SysUtils.StringOf.
	StringReplace (see page 4253)	Replaces occurrences of a substring within a string.
	StringToGUID (see page 4254)	Converts a string to a GUID.
	StrLCat (see page 4255)	Appends up to a specified maximum number of characters to string.
	StrLCat (see page 4255)	Appends up to a specified maximum number of characters to string.
	StrLComp (see page 4257)	Compares up to a specified maximum number of characters in two strings.
	StrLComp (see page 4257)	Compares up to a specified maximum number of characters in two strings.
	StrLCopy (see page 4258)	Copies up to a specified maximum number of characters from Source to Dest.

	StrLCopy (see page 4258)	Copies up to a specified maximum number of characters from Source to Dest.
	StrLen (see page 4260)	Returns number of characters in a string excluding the null terminator.
	StrLen (see page 4260)	Returns number of characters in a string excluding the null terminator.
	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.
	StrLFmt (see page 4261)	Formats a series of arguments from a specified open array into a buffer.
	StrLIComp (see page 4262)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLIComp (see page 4262)	Compares strings up to a specified maximum number of characters, without case sensitivity.
	StrLower (see page 4264)	Converts a string to lowercase.
	StrLower (see page 4264)	Converts a string to lowercase.
	StrMove (see page 4265)	Copies specified number of characters to string.
	StrMove (see page 4265)	Copies specified number of characters to string.
	StrNew (see page 4266)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNew (see page 4266)	Allocates space on and copies a string to the heap; returning a pointer to the string.
	StrNextChar (see page 4268)	Returns pointer to following character.
	StrNextChar (see page 4268)	Returns pointer to following character.
	StrPas (see page 4268)	Converts null-terminated string to an AnsiString (long string).
	StrPas (see page 4269)	Converts null-terminated string to an AnsiString (long string).
	StrPCopy (see page 4271)	Copies an AnsiString (long string) to a null-terminated string.
	StrPCopy (see page 4271)	Copies an AnsiString (long string) to a null-terminated string.
	StrPLCopy (see page 4272)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPLCopy (see page 4272)	Copies characters from an AnsiString (long string) into a null-terminated string.
	StrPos (see page 4272)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrPos (see page 4273)	Returns a pointer to the first occurrence of STR2 in STR1.
	StrRScan (see page 4274)	Returns a pointer to the last occurrence of a specified character in a string.
	StrRScan (see page 4274)	Returns a pointer to the last occurrence of a specified character in a string.
	StrScan (see page 4275)	Returns a pointer to first occurrence of a specified character in a string.
	StrScan (see page 4275)	Returns a pointer to first occurrence of a specified character in a string.
	StrToBool (see page 4276)	Converts a string to a Boolean value.
	StrToBoolDef (see page 4277)	Converts a string to a Boolean value, with error default.
	StrToCurr (see page 4278)	Converts a string to a System::Currency (see page 3235) value.
	StrToCurr (see page 4278)	Converts a string to a System::Currency (see page 3235) value.
	StrToCurrDef (see page 4278)	Converts a string to a System::Currency (see page 3235) value, with error default.
	StrToCurrDef (see page 4278)	Converts a string to a System::Currency (see page 3235) value, with error default.
	StrToDate (see page 4279)	Converts a string to a TDateTime (see page 3323) value.
	StrToDate (see page 4279)	Converts a string to a TDateTime (see page 3323) value.
	StrToDateDef (see page 4280)	Converts a string to a TDateTime (see page 3323) value, with error default.
	StrToDateDef (see page 4280)	Converts a string to a TDateTime (see page 3323) value, with error default.
	StrToDateTime (see page 4281)	Converts a string to a TDateTime (see page 3323) value.
	StrToDateTime (see page 4281)	Converts a string to a TDateTime (see page 3323) value.
	StrToDateTimeDef (see page 4282)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrToDateTimeDef (see page 4282)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrToFloat (see page 4282)	Converts a given string to a floating-point value.
	StrToFloat (see page 4282)	Converts a given string to a floating-point value.
	StrToFloatDef (see page 4286)	Converts a given string to a floating-point value with error default.
	StrToFloatDef (see page 4286)	Converts a given string to a floating-point value with error default.
	StrToInt (see page 4286)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToInt64 (see page 4287)	Converts a string that represents an integer (decimal or hex notation) to a number.

	StrToInt64Def (see page 4288)	Converts a string that represents an integer (decimal or hex notation) to a number.
	StrToIntDef (see page 4288)	Converts a string that represents an integer (decimal or hex notation) to a number with error default.
	StrToTime (see page 4291)	Converts a string to a TDateTime (see page 3323) value.
	StrToTime (see page 4291)	Converts a string to a TDateTime (see page 3323) value.
	StrToTimeDef (see page 4292)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrToTimeDef (see page 4292)	Converts a string to a TDateTime (see page 3323) value with error default.
	StrUpper (see page 4292)	Returns a string in upper case.
	StrUpper (see page 4292)	Returns a string in upper case.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	Supports (see page 4294)	Indicates whether a given object or interface supports a specified interface.
	SysErrorMessage (see page 4294)	Converts OS error codes into strings.
	SystemTimeToDateTime (see page 4295)	Converts a system time value into a TDateTime (see page 3323) value.
	TextPos (see page 4296)	This is function SysUtils.TextPos.
	TextPos (see page 4296)	This is function SysUtils.TextPos.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	TextToFloat (see page 4296)	Converts a null-terminated string to a floating-point value.
	Time (see page 4297)	Returns the current time.
	TimeStampToDateTime (see page 4298)	Converts a TTimeStamp (see page 4334) value into the corresponding TDateTime (see page 3323) value.
	TimeStampToMSecs (see page 4299)	Converts a TTimeStamp (see page 4334) value into an absolute number of milliseconds.
	TimeToStr (see page 4299)	Returns a string that represents a TDateTime (see page 3323) value.
	TimeToStr (see page 4299)	Returns a string that represents a TDateTime (see page 3323) value.
	Trim (see page 4301)	Trims leading and trailing spaces and control characters from a string.
	TrimLeft (see page 4301)	Trims leading spaces and control characters from a string.
	TrimRight (see page 4302)	Trims trailing spaces and control characters from a string.
	TryEncodeDate (see page 4302)	Creates a TDateTime (see page 3323) value that represents a specified Year, Month, and Day.
	TryEncodeTime (see page 4303)	Creates a TDateTime (see page 3323) value for a specified Hour, Min, Sec, and MSec.
	TryFloatToCurr (see page 4304)	Converts a floating-point value to a System::Currency (see page 3235) value.
	TryFloatToDateTime (see page 4304)	Converts a floating-point value to a TDateTime (see page 3323) value.
	TryStrToBool (see page 4305)	Converts a string to a Boolean value, with a Boolean success code.
	TryStrToCurr (see page 4305)	Converts a string to a System::Currency (see page 3235) value, with a Boolean success code.
	TryStrToCurr (see page 4305)	Converts a string to a System::Currency (see page 3235) value, with a Boolean success code.
	TryStrToDate (see page 4306)	Converts a string to a TDateTime (see page 3323) value, with a Boolean success code.
	TryStrToDate (see page 4306)	Converts a string to a TDateTime (see page 3323) value, with a Boolean success code.
	TryStrToDateTime (see page 4307)	Converts a string to a TDateTime (see page 3323) value with a Boolean success code.
	TryStrToDateTime (see page 4307)	Converts a string to a TDateTime (see page 3323) value with a Boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.
	TryStrToFloat (see page 4307)	Converts a given string to a floating-point value with a boolean success code.

	TryStrToInt (see page 4308)	Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.
	TryStrToInt64 (see page 4309)	Converts a string that represents an integer (decimal or hex notation) to a number.
	TryStrToTime (see page 4309)	Converts a string to a TDateTime (see page 3323) value with an error default.
	TryStrToTime (see page 4309)	Converts a string to a TDateTime (see page 3323) value with an error default.
	UIntToStr (see page 4310)	This is function SysUtils.UIntToStr.
	UIntToStr (see page 4310)	This is function SysUtils.UIntToStr.
	UnloadPackage (see page 4310)	Unloads a package.
	UpperCase (see page 4311)	Returns a copy of a string in uppercase.
	UpperCase (see page 4311)	Returns a copy of a string in uppercase.
	WideBytesOf (see page 4312)	Converts an Unicode string to an array of bytes.
	WideCompareStr (see page 4313)	Compares Unicode strings based on the current locale with case sensitivity.
	WideCompareText (see page 4313)	Compares Unicode strings based on the current locale without case sensitivity.
	WideFmtStr (see page 4314)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFmtStr (see page 4314)	Assembles a formatted Unicode string using a format string and an array of arguments.
	WideFormat (see page 4314)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormat (see page 4314)	Returns a formatted Unicode string assembled from a format string and an array of arguments.
	WideFormatBuf (see page 4315)	Formats the arguments from an array, placing the result in a buffer.
	WideFormatBuf (see page 4315)	Formats the arguments from an array, placing the result in a buffer.
	WideLowerCase (see page 4316)	Returns Unicode string converted to lower case.
	WideSameStr (see page 4316)	Compares Unicode strings based on the current locale with case sensitivity.
	WideSameText (see page 4316)	Compares Unicode strings based on the current locale without case sensitivity.
	WideStrAlloc (see page 4317)	This is function SysUtils.WideStrAlloc.
	WideStringOf (see page 4317)	This is function SysUtils.WideStringOf.
	WideUpperCase (see page 4318)	Returns Unicode string converted to upper case.
	Win32Check (see page 4318)	Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.
	WrapText (see page 4318)	Splits a string into multiple lines as its length approaches a specified size.
	WrapText (see page 4318)	Splits a string into multiple lines as its length approaches a specified size.

37.3.1 SysUtils.Abort Function

Raises a silent exception.

Pascal

```
procedure Abort;
```

C++

```
Abort();
```

File

SysUtils

Description

Use Abort to escape from an execution path without reporting an error.

Abort raises a special "silent exception" (EAbort (see page 3916)), which operates like any other exception, but does not display an error message to the end user. Abort redirects execution to the end of the last exception block.

Delphi Examples:

```
{
This example checks for a valid entry in a TDBEdit control
and calls the Abort procedure if the control is empty; Abort
cancels the post before it happens.
}
procedure TForm1.ClientDataSet1BeforePost(DataSet: TDataSet);
begin
  if DBEdit1.Text = '' then
    Abort;
  DataSet.Fields[2].AsString := DBEdit1.Text;
end;
```

See Also

EAbort (see page 3916)

37.3.2 SysUtils.AddExitProc Function

Adds a procedure to the exit list.

Pascal

```
procedure AddExitProc(Proc: TProcedure);
```

C++

```
AddExitProc(TProcedure Proc);
```

File

SysUtils

Description

AddExitProc adds Proc to the Delphi exit procedure list, so that Proc is automatically executed when the application terminates. AddExitProc is not compatible with CLX package support and is provided for backward compatibility only. Do not use AddExitProc in new applications.

37.3.3 SysUtils.AddTerminateProc Function

Adds a terminate procedure to the system list of termination procedures.

Pascal

```
procedure AddTerminateProc(TermProc: TTerminateProc);
```

C++

```
AddTerminateProc(TTerminateProc TermProc);
```

File

SysUtils

Description

Call AddTerminateProc in Delphi code to add a procedure that is called when the application is about to terminate. The run time library calls all termination procedures before the application terminates. The supplied termination procedure (TermProc) should return true if the application can safely terminate or false if the application cannot safely terminate. If any termination procedure returns false, the application does not terminate.

See Also

CallTerminateProcs (see page 4071)

37.3.4 SysUtils.AdjustLineBreaks Function

Adjusts line break characters to reflect Windows or Linux style.

Pascal

```
function AdjustLineBreaks(const S: string; Style: TTextLineBreakStyle = tlbsCRLF): string;
overload;
```

C++

```
UnicodeString AdjustLineBreaks(const UnicodeString S, TTextLineBreakStyle Style = tlbsCRLF);
```

File

SysUtils

Description

AdjustLineBreaks returns a text string with its line breaks adjusted to fit a specified style.

S is the string to convert.

Style indicates whether line breaks should all be Windows-style line breaks with a carriage return and linefeed (tlbsCRLF), or Linux-style breaks with a single linefeed character (tlbsLF). If Style is omitted, the style for the compile target is used.

See Also

TrimLeft (see page 4301)

TrimRight (see page 4302)

37.3.5 SysUtils.AnsiCompareFileName Function

Compares file names based on the current locale.

Pascal

```
function AnsiCompareFileName(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareFileName(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiCompareFileName compares two file names. The current locale controls the compare operation. Comparison is case-insensitive under Windows, case-sensitive under Linux.

The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

AnsiCompareFileName supports the idiosyncrasies of MS-DOS file names that use Zenkaku Moji, a character encoding used in some Asian locales. Under Windows locales that do not use MBCS, AnsiCompareFileName is identical to AnsiCompareText (see page 4051). Under Linux, AnsiCompareFileName is identical to AnsiCompareStr (see page 4050).

See Also

SameFileName (see page 4238)

AnsiCompareText (see page 4051)

37.3.6 SysUtils.AnsiCompareStr Function

Compares strings based on the current locale with case sensitivity.

Pascal

```
function AnsiCompareStr(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareStr(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiCompareStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters. Thus, setting S1 to 'a' and S2 to 'A' causees AnsiCompareStr to return a value less than zero, while CompareStr (see page 4078), with the same arguments, returns a value greater than zero.

See Also

AnsiCompareText (see page 4051)

AnsiStrComp (see page 4056)

AnsiSameStr (see page 4055)

37.3.7 SysUtils.AnsiCompareText Function

Compares strings based on the current locale without case sensitivity.

Pascal

```
function AnsiCompareText(const S1: string; const S2: string): Integer;
```

C++

```
int AnsiCompareText(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiCompareText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. AnsiCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

AnsiCompareStr (see page 4050)

AnsiStrIComp (see page 4057)

AnsiSameText (see page 4055)

37.3.8 SysUtils.AnsiDequotedStr Function

Converts a quoted string into an unquoted string

Pascal

```
function AnsiDequotedStr(const S: string; AQuote: Char): string; overload;
```

C++

```
UnicodeString AnsiDequotedStr(const UnicodeString S, Char AQuote);
```

File

SysUtils

Description

AnsiDequotedStr removes the quote characters from the beginning and end of a quoted string, and reduces pairs of quote characters within the string to a single quote character. The AQuote parameter defines what character to use as a quote character. If the S parameter does not begin and end with the quote character, AnsiDequotedStr returns S unchanged.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiQuotedStr (see page 4054)

AnsiExtractQuotedStr (see page 4052)

37.3.9 SysUtils.AnsiExtractQuotedStr Function

Converts a quoted string into an unquoted string.

Pascal

```
function AnsiExtractQuotedStr(var Src: PAnsiChar; Quote: AnsiChar): AnsiString; overload;
function AnsiExtractQuotedStr(var Src: PWidechar; Quote: WideChar): UnicodeString; overload;
```

C++

```
AnsiString AnsiExtractQuotedStr(PAnsiChar Src, AnsiChar Quote);
UnicodeString AnsiExtractQuotedStr(PWidechar Src, WideChar Quote);
```

File

SysUtils

Description

AnsiExtractQuotedStr removes the quote characters from the beginning and end of a quoted string, and reduces pairs of quote characters within the string to a single quote character. The Quote parameter defines what character to use as a quote character. If the first character in Src is not the value of the Quote parameter, AnsiExtractQuotedStr returns an empty string.

The function copies characters from Src to the result string until the second solitary quote character or the first null character in Src. The Src parameter is updated to point to the first character following the quoted string. If Src does not contain a matching end quote character, the Src parameter is updated to point to the terminating null character.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiQuotedStr (see page 4054)

37.3.10 SysUtils.AnsiLastChar Function

Returns a pointer to the last character in a string.

Pascal

```
function AnsiLastChar(const S: AnsiString): PAnsiChar; overload;
function AnsiLastChar(const S: UnicodeString): PWideChar; overload;
```

C++

```
PAnsiChar AnsiLastChar(const AnsiString S);
PWideChar AnsiLastChar(const UnicodeString S);
```

File

SysUtils

Description

Call AnsiLastChar to obtain the last character in a string. This function supports multi-byte character sets (MBCS).

See Also

AnsiStrLastChar (see page 4057)

AnsiStrRScan (see page 4060)

37.3.11 SysUtils.AnsiLowerCase Function

Returns a string that is a copy of the given string converted to lower case.

Pascal

```
function AnsiLowerCase(const S: string): string; overload;
```

C++

```
UnicodeString AnsiLowerCase(const UnicodeString S);
```

File

SysUtils

Description

AnsiLowerCase returns a string that is a copy of the given string converted to lower case. The conversion uses the current locale. This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCaseFileName (see page 4053)

AnsiUpperCase (see page 4062)

LowerCase (see page 4225)

37.3.12 SysUtils.AnsiLowerCaseFileName Function

Converts a file name to lower case.

Pascal

```
function AnsiLowerCaseFileName(const S: string): string; overload;
```

C++

```
UnicodeString AnsiLowerCaseFileName(const UnicodeString S);
```

File

SysUtils

Description

AnsiLowerCaseFileName returns a string that is a copy of S converted to lower case. The conversion uses the current locale.

Note: On Windows, AnsiLowerCaseFileName supports lowercase conversion idiosyncrasies of file names in Asian locales (Zenkaku).

On Linux, or in locales that do not use multi byte character systems (MBCS), AnsiLowerCaseFileName is identical to AnsiLowerCase (see page 4053).

See Also

AnsiLowerCase (see page 4053)

AnsiUpperCaseFileName (see page 4062)

37.3.13 SysUtils.AnsiPos Function

Locates the position of a substring.

Pascal

```
function AnsiPos(const Substr: string; const S: string): Integer; overload;
```

C++

```
int AnsiPos(const UnicodeString Substr, const UnicodeString S);
```

File

SysUtils

Description

Call AnsiPos to obtain the byte offset of the Substr parameter, as it appears in the string S. For example, if Substr is the string "AB", and S is the string "ABCDE", AnsiPos returns 1. If Substr does not appear in S, AnsiPos returns 0.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiStrPos (see page 4060)

37.3.14 SysUtils.AnsiQuotedStr Function

Returns the quoted version of a string.

Pascal

```
function AnsiQuotedStr(const S: string; Quote: Char): string; overload;
```

C++

```
UnicodeString AnsiQuotedStr(const UnicodeString S, Char Quote);
```

File

SysUtils

Description

Use AnsiQuotedStr to convert a string (S) to a quoted string, using the provided Quote character. A Quote character is inserted at the beginning and end of S, and each Quote character in the string is doubled. To remove the quotes from a quoted string, use AnsiDequotedStr (see page 4051) routine.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiExtractQuotedStr (see page 4052)

QuotedStr (see page 4229)

37.3.15 SysUtils.AnsiSameStr Function

Compares strings based on the current locale with case sensitivity.

Pascal

```
function AnsiSameStr(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean AnsiSameStr(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiSameStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

AnsiSameText (see page 4055)

AnsiCompareStr (see page 4050)

AnsiSameCaption

37.3.16 SysUtils.AnsiSameText Function

Compares strings based on the current locale without case sensitivity.

Pascal

```
function AnsiSameText(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean AnsiSameText(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

AnsiSameText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

AnsiSameStr (see page 4055)

AnsiCompareText (see page 4051)

AnsiSameCaption

37.3.17 SysUtils.AnsiStrAlloc Function

Pascal

```
function AnsiStrAlloc(Size: Cardinal): PAnsiChar;
```

C++

```
PAnsiChar AnsiStrAlloc(unsigned Size);
```

File

SysUtils

Description

This is function SysUtils.AnsiStrAlloc.

37.3.18 SysUtils.AnsiStrComp Function

Compares null-terminated character strings.

Pascal

```
function AnsiStrComp(S1: PAnsiChar; S2: PAnsiChar): Integer;
function AnsiStrComp(S1: PWideChar; S2: PWideChar): Integer;
```

C++

```
int AnsiStrComp(PAnsiChar S1, PAnsiChar S2);
int AnsiStrComp(PWideChar S1, PWideChar S2);
```

File

SysUtils

Description

AnsiStrComp compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiCompareStr (see page 4050)

AnsiStrIComp (see page 4057)

AnsiStrLComp (see page 4058)

37.3.19 SysUtils.AnsiStrIComp Function

Compares null terminated character strings case insensitively.

Pascal

```
function AnsiStrIComp(S1: PAnsiChar; S2: PAnsiChar): Integer;
function AnsiStrIComp(S1: PWideChar; S2: PWideChar): Integer;
```

C++

```
int AnsiStrIComp(PAnsiChar S1, PAnsiChar S2);
int AnsiStrIComp(PWideChar S1, PWideChar S2);
```

File

SysUtils

Description

AnsiStrIComp compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is the same as for CompareStr (see page 4078).

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiCompareText (see page 4051)

AnsiStrComp (see page 4056)

37.3.20 SysUtils.AnsiStrLastChar Function

Returns a pointer to the last character in a null-terminated string.

Pascal

```
function AnsiStrLastChar(P: PAnsiChar): PAnsiChar; overload;
function AnsiStrLastChar(P: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar AnsiStrLastChar(PAnsiChar P);
PWideChar AnsiStrLastChar(PWideChar P);
```

File

SysUtils

Description

Call AnsiStrLastChar to obtain a pointer to the last full character in a null-terminated sequence. When working with multi-byte character sets (MBCS), the last character will not be the same as the last char value if the last two bytes represent a single double byte character.

See Also

AnsiLastChar (see page 4052)

37.3.21 SysUtils.AnsiStrLComp Function

Compares the first MaxLen bytes of two null-terminated strings, case-sensitively.

Pascal

```
function AnsiStrLComp(S1: PAnsiChar; S2: PAnsiChar; MaxLen: Cardinal): Integer; overload;
function AnsiStrLComp(S1: PWideChar; S2: PWideChar; MaxLen: Cardinal): Integer;
```

C++

```
int AnsiStrLComp(PAnsiChar S1, PAnsiChar S2, unsigned MaxLen);
int AnsiStrLComp(PWideChar S1, PWideChar S2, unsigned MaxLen);
```

File

SysUtils

Description

AnsiStrLComp compares S1 to S2, with case sensitivity. If S1 or S2 is longer than MaxLen bytes, AnsiStrLComp only compares the first MaxLen bytes. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2 (up to MaxLen characters)	= 0

Note: This function supports multi-byte character sets (MBCS).

Note: AnsiStrLComp can continue the comparison beyond a null terminating character if S1 and S2 are equal up to a null character and the null character occurs before MaxLen bytes.

Note: Most locales consider lowercase characters to come before corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters come after uppercase characters.

See Also

AnsiStrComp (see page 4056)

AnsiStrLIComp (see page 4058)

StrLComp (see page 4257)

37.3.22 SysUtils.AnsiStrLIComp Function

Compares two strings, case-insensitively, up to the first MaxLen bytes.

Pascal

```
function AnsiStrLIComp(S1: PAnsiChar; S2: PAnsiChar; MaxLen: Cardinal): Integer; overload;
function AnsiStrLIComp(S1: PWideChar; S2: PWideChar; MaxLen: Cardinal): Integer;
```

C++

```
int AnsiStrLIComp(PAnsiChar S1, PAnsiChar S2, unsigned MaxLen);
int AnsiStrLIComp(PWideChar S1, PWideChar S2, unsigned MaxLen);
```

File

SysUtils

Description

AnsiStrLIComp compares S1 to S2, without case sensitivity. If S1 or S2 is longer than MaxLen characters, AnsiStrLIComp only compares up to the first MaxLen characters. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2 (up to MaxLen characters)	= 0

Note: This function supports multi-byte character sets (MBCS).

Note: AnsiStrLIComp can continue the comparison beyond a null terminating character if S1 and S2 are equal up to a null character and the null character occurs before MaxLen bytes.

See Also

AnsiStrIComp (see page 4057)

AnsiStrLComp (see page 4058)

StrLIComp (see page 4262)

37.3.23 SysUtils.AnsiStrLower Function

Converts all characters in a null-terminated string to lower case.

Pascal

```
function AnsiStrLower(Str: PAnsiChar): PAnsiChar; overload;
function AnsiStrLower(Str: PWideChar): PWideChar;
```

C++

```
PAnsiChar AnsiStrLower(PAnsiChar Str);
PWideChar AnsiStrLower(PWideChar Str);
```

File

SysUtils

Description

AnsiStrLower returns a null-terminated character sequence where all characters are lower case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCase (see page 4053)

AnsiLowerCaseFileName (see page 4053)

AnsiStrUpper (see page 4061)

37.3.24 SysUtils.AnsiStrPos Function

Returns a pointer to the first occurrence of a substring.

Pascal

```
function AnsiStrPos(Str: PAnsiChar; SubStr: PAnsiChar): PAnsiChar; overload;
function AnsiStrPos(Str: PWideChar; SubStr: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar AnsiStrPos(PAnsiChar Str, PAnsiChar SubStr);
PWideChar AnsiStrPos(PWideChar Str, PWideChar SubStr);
```

File

SysUtils

Description

Call AnsiStrPos to locate the first occurrence of SubStr in Str (see page 3676). If SubStr does not occur in Str (see page 3676), AnsiStrPos returns nil (Delphi) or NULL (C++).

Unlike the StrPos (see page 4272) function, AnsiStrPos works with multi-byte characters sets (MBCS).

See Also

AnsiPos (see page 4054)

StrPos (see page 4272)

37.3.25 SysUtils.AnsiStrRScan Function

Returns a pointer to the last occurrence of a specified character in a specified string.

Pascal

```
function AnsiStrRScan(Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function AnsiStrRScan(Str: PWideChar; Chr: WideChar): PWideChar;
```

C++

```
PAnsiChar AnsiStrRScan(PAnsiChar Str, AnsiChar Chr);
PWideChar AnsiStrRScan(PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

Call AnsiStrRScan to obtain a pointer to the last occurrence of Chr (see page 3536) in Str (see page 3676). If Chr (see page 3536) does not occur in Str (see page 3676), AnsiStrRScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Unlike the StrRScan (see page 4274) function, AnsiStrRScan works with multi-byte character systems (MBCS).

See Also

AnsiStrScan (see page 4061)

StrRScan (see page 4274)

37.3.26 SysUtils.AnsiStrScan Function

Returns a pointer to first occurrence of a character in a string.

Pascal

```
function AnsiStrScan(Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function AnsiStrScan(Str: PWideChar; Chr: WideChar): PWideChar; overload;
```

C++

```
PAnsiChar AnsiStrScan(PAnsiChar Str, AnsiChar Chr);
PWideChar AnsiStrScan(PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

Call AnsiStrScan to obtain a pointer to the first occurrence of Chr (see page 3536) in Str (see page 3676). If Chr (see page 3536) does not occur in Str (see page 3676), AnsiStrScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Unlike the StrScan (see page 4275) function, AnsiStrScan works with multi-byte character systems (MBCS).

See Also

AnsiStrRScan (see page 4060)

StrScan (see page 4275)

37.3.27 SysUtils.AnsiStrUpper Function

Converts all characters in a null-terminated string to upper case.

Pascal

```
function AnsiStrUpper(Str: PAnsiChar): PAnsiChar; overload;
function AnsiStrUpper(Str: PWideChar): PWideChar;
```

C++

```
PAnsiChar AnsiStrUpper(PAnsiChar Str);
PWideChar AnsiStrUpper(PWideChar Str);
```

File

SysUtils

Description

AnsiStrUpper returns a null-terminated string where all characters are upper case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiUpperCase (see page 4062)

AnsiUpperCaseFileName (see page 4062)

AnsiStrLower (see page 4059)

37.3.28 SysUtils.AnsiUpperCase Function

Converts a string to upper case.

Pascal

```
function AnsiUpperCase(const S: string): string; overload;
```

C++

```
UnicodeString AnsiUpperCase(const UnicodeString S);
```

File

SysUtils

Description

AnsiUpperCase returns a string that is a copy of S, converted to upper case. The conversion uses the current locale.

Note: This function supports multi-byte character sets (MBCS).

See Also

AnsiLowerCase (see page 4053)

UpperCase (see page 4311)

37.3.29 SysUtils.AnsiUpperCaseFileName Function

Converts a file name to upper case.

Pascal

```
function AnsiUpperCaseFileName(const S: string): string; overload;
```

C++

```
UnicodeString AnsiUpperCaseFileName(const UnicodeString S);
```

File

SysUtils

Description

AnsiUpperCaseFileName returns a string that is a copy of S converted to upper case. The conversion uses the current locale.

Note: On Windows, AnsiUpperCaseFileName supports uppercase conversion idiosyncrasies of file names in Asian locales (Zenkaku).

On Linux or in locales that do not use multi byte character systems (MBCS), AnsiUpperCaseFileName is identical to AnsiUpperCase (see page 4062).

See Also

AnsiLowerCaseFileName (see page 4053)

AnsiUpperCase (see page 4062)

37.3.30 SysUtils.AppendStr Function

Appends a dynamically allocated string to an existing string.

Pascal

```
procedure AppendStr(var Dest: AnsiString; const S: AnsiString); deprecated;
```

C++

```
AppendStr(AnsiString Dest, const AnsiString S);
```

File

SysUtils

Description

AppendStr appends S to the end of Dest. It is provided for backwards compatibility only. Use the + operator instead:

Dest := Dest + S;

Dest = Dest + S;

37.3.31 SysUtils.AssignStr Function

Assigns a new dynamically allocated string to a specified pointer.

Pascal

```
procedure AssignStr(var P: PAnsiString; const S: AnsiString); deprecated;
```

C++

```
AssignStr(PAnsiString P, const AnsiString S);
```

File

SysUtils

Description

Do not use AssignStr in new applications. It is provided for backward compatibility only.

See Also

DisposeStr (see page 4103)

37.3.32 SysUtils.Beep Function

Emits a message beep.

Pascal

```
procedure Beep;
```

C++

```
Beep();
```

File

SysUtils

Description

Beep generates a conventional message beep.

Note: On Windows, Beep calls the Windows API MessageBeep.

Note: On Linux, to minimize library dependencies in the run time library, the SysUtils (see page 3901) version of Beep has limited Linux functionality. For cross-platform GUI applications, use the QControls version of Beep instead.

C++ Examples:

```
/*
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection. Notice that the
"Match whole word only" and "Match case" work.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
  FindDialog1->Execute();
}

void __fastcall TForm1::FindDialog1Find(TObject *Sender)
{
  int FoundAt, StartPos, ToEnd;
  TSearchTypes mySearchTypes = TSearchTypes();
  // begin the search after the current selection
  // if there is one
  // otherwise, begin at the start of the text
  if (FindDialog1->Options.Contains(frMatchCase))
    mySearchTypes << stMatchCase;
  if (FindDialog1->Options.Contains(frWholeWord))
    mySearchTypes << stWholeWord;
  if (RichEdit1->SelLength)
    StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
  else
    StartPos = 0;
  // ToEnd is the length from StartPos
  // to the end of the text in the rich edit control
  ToEnd = RichEdit1->Text.Length() - StartPos;
  FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd, mySearchTypes);
  if (FoundAt != -1)
  {
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
  }
  else Beep();
```

```
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  RichEdit1->PlainText = False;
  RichEdit1->Lines->LoadFromFile("../Overview.RTF");  // starts in Debug
  RichEdit1->ScrollBars = ssVertical;
}
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
This example requires a TRichEdit, a TButton, and a
TFindDialog.  Clicking the button click will display a Find
Dialog to the right of the edit control.  Filling in the
"Find what" text and pressing the Find Next button will
select the first matching string in the Rich Edit control
that follows the previous selection.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  FindDialog1.Position :=
    Point(RichEdit1.Left + RichEdit1.Width, RichEdit1.Top);
  FindDialog1.Execute;
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: Integer;
  mySearchTypes : TSearchTypes;
  myFindOptions : TFindOptions;
begin
  mySearchTypes := [];
  with RichEdit1 do
  begin
    if frMatchCase in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stMatchCase];
    if frWholeWord in FindDialog1.Options then
       mySearchTypes := mySearchTypes + [stWholeWord];
    { begin the search after the current selection if there is one }
    { otherwise, begin at the start of the text }
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
```

```
      { ToEnd is the length from StartPos to the end of the
        text in the rich edit control }
      ToEnd := Length(Text) - StartPos;
      FoundAt :=
        FindText(FindDialog1.FindText, StartPos, ToEnd, mySearchTypes);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end
      else Beep;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  const Path = 'OverView.RTF';
  begin
    RichEdit1.PlainText := False;
    RichEdit1.Lines.LoadFromFile(Path);
    RichEdit1.ScrollBars := ssVertical;
  end;
  {
  The following example beeps once for each "beep" passed in
  on the command line.  The example terminates the application
  if  "exit" is passed in on the command line.  Build the
  project to generate a .exe file and then execute that file
  from a command line: "ParamCount_proj beep beep exit".
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
  //  Use print statements since you can't use the debugger.
  //  MessageDlg(
  //    'System ParamCount = ' + IntToStr(System.ParamCount),
  //    mtConfirmation, [mbOK], 0);
    for i := 1 to System.ParamCount do
    begin
      if LowerCase(ParamStr(i)) = 'beep' then
        Beep
      else if LowerCase(ParamStr(i)) = 'exit' then
        Application.Terminate;
      Sleep(250);
    end;
  end;
```

37.3.33 SysUtils.BoolToStr Function

Converts a Boolean value to a string.

Pascal

```
function BoolToStr(B: Boolean; UseBoolStrs: Boolean = False): string;
```

C++

```
UnicodeString BoolToStr(Boolean B, Boolean UseBoolStrs = False);
```

File

SysUtils

Description

BoolToStr converts the given Boolean value to a string as follows:

Value of B	Value of UseBoolStrs	Value of returned string
true	false	'-1'
true	true	The first string in TrueBoolStrs (see page 4363) (default 'True');
false	false	'0'
false	true	The first string in FalseBoolStrs (see page 4352) (default 'False');

Note: The TrueBoolStrs (see page 4363) and FalseBoolStrs (see page 4352) variables are undefined when an application starts. Any use of StrToBool (see page 4276), BoolToStr or TryStrToBool (see page 4305) will cause these variables to be initialised with the default values above.

C++ Examples:

```
/*
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
*/
void __fastcall TForm1::ApplicationEvents1SettingChange(TObject *Sender, int Flag, const
AnsiString Section, int &Result)
{
  Memo1->Lines->BeginUpdate();
  try
  {
//    char buffer[64];
    AnsiString buffer;
    buffer.sprintf("Section = %s", Section.c_str());
    Memo1->Lines->Add(buffer);
    buffer.sprintf("Flag = %8x", Flag);
    Memo1->Lines->Add(buffer);
    if (strcmp(Section.c_str(), "intl") == 0)
    {
      buffer.sprintf("DefaultLCID = %8x", SysLocale.DefaultLCID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("PriLangID = %8x", SysLocale.PriLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("SubLangID = %8x", SysLocale.SubLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("FarEast = %s", BoolToStr(SysLocale.FarEast, true));
      Memo1->Lines->Add(buffer);
      buffer.sprintf("MiddleEast = %s", BoolToStr(SysLocale.MiddleEast, true));
      Memo1->Lines->Add(buffer);
      Memo1->Lines->Add("");
    }
  }
  __finally
  {
    Memo1->Lines->EndUpdate();
  }
}
```

Delphi Examples:

```
{
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
```

```
}
procedure TForm1.ApplicationEvents1SettingChange(Sender: TObject; Flag: Integer; const
Section: string; var Result: Integer);
begin
  Memo1.Lines.BeginUpdate;
  try
    Memo1.Lines.Add(Format('Section  = %s', [Section]));
    Memo1.Lines.Add(Format('Flags  = %.8x', [Flag]));
    if AnsiSameStr(Section, 'intl') then
    with SysLocale do
    begin
      Memo1.Lines.Add(Format('DefaultLCID  = %.8x', [DefaultLCID]));
      Memo1.Lines.Add(Format('PriLangID  = %.8x', [PriLangID]));
      Memo1.Lines.Add(Format('SubLangID  = %.8x', [SubLangID]));
      Memo1.Lines.Add(Format('FarEast  = %s', [BoolToStr(FarEast, True)]));
      Memo1.Lines.Add(Format('MiddleEast  = %s', [BoolToStr(MiddleEast, True)]));
    end;
    Memo1.Lines.Add('');
  finally
    Memo1.Lines.EndUpdate;
  end;
//  Result:= 0;
end;
```

See Also

StrToBool (see page 4276)

TryStrToBool (see page 4305)

TrueBoolStrs (see page 4363)

FalseBoolStrs (see page 4352)

37.3.34 SysUtils.ByteLength Function

Pascal

```
function ByteLength(const S: string): Integer;
```

C++

```
int ByteLength(const UnicodeString S);
```

File

SysUtils

Description

This is function SysUtils.ByteLength.

37.3.35 SysUtils.BytesOf Function

Converts a string into a byte array.

Pascal

```
function BytesOf(const Val: RawByteString): TBytes; overload;
function BytesOf(const Val: UnicodeString): TBytes; overload;
function BytesOf(const Val: WideChar): TBytes; overload;
```

```
function BytesOf(const Val: AnsiChar): TBytes; overload;
```

C++

```
TBytes BytesOf(const RawByteString Val);
TBytes BytesOf(const UnicodeString Val);
TBytes BytesOf(const WideChar Val);
TBytes BytesOf(const AnsiChar Val);
```

File

SysUtils

Description

Call BytesOf to convert a string or a character to an array of bytes. Unicode strings and WideStrings (see page 6807) are converted using the default system locale represented by the Default property of the TEncoding (see page 3969) class.

See Also

TEncoding (see page 3969)

GetBytes (see page 3977)

WideBytesOf (see page 4312)

StringCodePage (see page 3676)

37.3.36 SysUtils.ByteToCharIndex Function

Returns the position of the character that contains a specified byte in a string.

Pascal

```
function ByteToCharIndex(const S: AnsiString; Index: Integer): Integer; overload;
function ByteToCharIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int ByteToCharIndex(const AnsiString S, int Index);
int ByteToCharIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call ByteToCharIndex to determine which character contains a particular byte in a string, where 1 specifies the first byte, 2 specifies the second byte, and so on. If the Index parameter specifies a byte that is not in the AnsiString (Index < 0 or Index > Length (see page 3607)(S)), ByteToCharIndex returns 0.

If the system does not use a multi-byte character system (MBCS), ByteToCharIndex simply returns the value of Index, because there is a 1:1 correspondence between bytes and characters. Otherwise, ByteToCharIndex returns the position of the character, where 1 specifies the first character, 2 specifies the second character, and so on.

For example, if an AnsiString contains two characters, a single-byte character followed by a double byte character, then ByteToCharIndex returns 1 when Index is 1, it returns 2 when Index is 2 or 3, and it returns 0 for any other value of Index.

See Also

ByteToCharLen (see page 4070)

CharToByteIndex (see page 4075)

37.3.37 SysUtils.ByteToCharLen Function

Returns the number of characters fully contained in the first MaxLen bytes of a string.

Pascal

```
function ByteToCharLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
function ByteToCharLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int ByteToCharLen(const AnsiString S, int MaxLen);
int ByteToCharLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

Call ByteToCharLen to determine the number of characters represented by a portion of a string. ByteToCharLen counts characters until it reaches MaxLen bytes or the end of the string, whichever comes first.

When working with multi-byte character systems (MBCS), the value returned by ByteToCharLen may be less than MaxLen or the length of the string.

Note: ByteToCharLen will count a partial character if it appears as the last of the first MaxLen bytes.

See Also

ByteToCharIndex (see page 4069)

CharToByteIndex (see page 4075)

CharToByteLen (see page 4076)

37.3.38 SysUtils.ByteType Function

Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.

Pascal

```
function ByteType(const S: AnsiString; Index: Integer): TMbcsByteType; overload;
function ByteType(const S: UnicodeString; Index: Integer): TMbcsByteType; overload;
```

C++

```
TMbcsByteType ByteType(const AnsiString S, int Index);
TMbcsByteType ByteType(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call ByteType to determine whether a specified byte in a string is a single-byte character, the first byte of a multibyte character, or one of the trailing bytes.

AnsiString is the string that contains the byte in question.

Index identifies the byte for which you want to know the byte type. Bytes are numbered from 1.

If the system is not using a multi-byte character system (MBCS), ByteType always returns mbSingleByte. Otherwise, ByteType returns mbSingleByte if the indicated byte represents a complete character in S, mbLeadByte if it represents the first byte of a double byte character, and mbTrailByte if it represents a subsequent byte of a multibyte character.

Note: No checking is done to ensure that Index is less than the length of S. It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

LeadBytes (see page 4354)

StrByteType (see page 4242)

37.3.39 SysUtils.CallTerminateProcs Function

Calls all of the functions in the termination procedure list.

Pascal

```
function CallTerminateProcs: Boolean;
```

C++

```
Boolean CallTerminateProcs();
```

File

SysUtils

Description

CallTerminateProcs is called internally when an application is about to terminate. It returns true only if all of the functions in the application's terminate procedure list return true. Do not call this function directly. Its use is automatic in Delphi application.

See Also

AddTerminateProc (see page 4048)

37.3.40 SysUtils.ChangeFileExt Function

Changes the extension of a file name.

Pascal

```
function ChangeFileExt(const FileName: string; const Extension: string): string; overload;
```

C++

```
UnicodeString ChangeFileExt(const UnicodeString FileName, const UnicodeString Extension);
```

File

SysUtils

Description

ChangeFileExt takes the file name passed in FileName and changes the extension of the file name to the extension passed in Extension. Extension specifies the new extension, including the initial dot character.

ChangeFileExt does not rename the actual file, it just creates a new file name string.

Note: This function works with multi-byte character sets (MBCS).

C++ Examples:

```
/*
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::ConvertIcon2BitmapClick(TObject *Sender)
{
  OpenDialog1->DefaultExt = ".ICO";
  OpenDialog1->Filter = "icons (*.ico)|*.ICO";
  OpenDialog1->Options << ofOverwritePrompt << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    std::auto_ptr<TIcon> pi(new Graphics::TIcon);
    AnsiString as;
    pi->LoadFromFile(OpenDialog1->FileName);
    as = ChangeFileExt(OpenDialog1->FileName,".BMP");
    Image1->Width = pi->Width;
    Image1->Height = pi->Height;
    Image1->Canvas->Draw(0, 0, pi.get());
    Image1->Picture->SaveToFile(as);
    ShowMessage(OpenDialog1->FileName + " Saved to " + as);
  }
}
```

Delphi Examples:

```
{
This example converts a specified icon to a bitmap. To run
this example, add an image, a button, and an Open dialog to
a form. Name the button ConvertIcon2Bitmap, and add the
following code as its OnClick event handler.
}
procedure TForm1.ConvertIcon2BitmapClick(Sender: TObject);
var
  s : string;
  Icon: TIcon;
begin
  OpenDialog1.DefaultExt := '.ICO';
  OpenDialog1.Filter := 'icons (*.ico)|*.ICO';
  OpenDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly ];
  if OpenDialog1.Execute then
  begin
    Icon := TIcon.Create;
    try
      Icon.Loadfromfile(OpenDialog1.FileName);
      s:= ChangeFileExt(OpenDialog1.FileName,'.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0,0,Icon);
      Image1.Picture.SaveToFile(s);
      ShowMessage(OpenDialog1.FileName + ' Saved to ' + s);
    finally
      Icon.Free;
    end;
  end;
```

```
end;
```

See Also

ExtractFileExt (see page 4113)

37.3.41 SysUtils.ChangeFilePath Function

Changes the path of a file name.

Pascal

```
function ChangeFilePath(const FileName: string; const Path: string): string; overload;
```

C++

```
UnicodeString ChangeFilePath(const UnicodeString FileName, const UnicodeString Path);
```

File

SysUtils

Description

ChangeFilePath replaces the drive and directory parts of the passed FileName with the passed Path value. If Path is not blank, it must specify a drive (terminated by a colon) and/or a directory (terminated by a backslash).

ChangeFilePath does not rename the actual file, it just creates a new file name string.

Note: This function works with multi-byte character sets (MBCS).

See Also

ExtractFilePath (see page 4115)

37.3.42 SysUtils.CharInSet Function

CharInSet checks whether a given character is contained within a set.

Pascal

```
function CharInSet(C: AnsiChar; const CharSet: TSysCharSet): Boolean; overload;
function CharInSet(C: WideChar; const CharSet: TSysCharSet): Boolean; overload;
```

C++

```
Boolean CharInSet(AnsiChar C, const TSysCharSet CharSet);
Boolean CharInSet(WideChar C, const TSysCharSet CharSet);
```

File

SysUtils

Description

Use CharInSet to check whether a given character is contained within a set of characters. CharInSet checks whether the `C` parameter is contained within the `CharSet` set. `C` can either be a Unicode or an ANSI character, while `CharSet` can only be a set of ANSI characters. The function returns **True** if the character is contained within the set and **False** if the character is not contained within the set or if the character is Unicode.

Note: We recommend that the TCharacter class (which is Unicode enabled) be used whenever possible to check whether a

character enters a certain category, such as digits or letters.

See Also

inOpArray (see page 6871)

inOpSet (see page 6871)

TCharacter

37.3.43 SysUtils.CharLength Function

Returns number of bytes used by character.

Pascal

```
function CharLength(const S: AnsiString; Index: Integer): Integer; overload;
function CharLength(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int CharLength(const AnsiString S, int Index);
int CharLength(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call CharLength to determine the size in bytes of the character starting at `Index` in `S`. If the character does not start at `Index`, this function returns the size of the *remainder* of the character, not the full character length.

Note: `Index` is an *element* index into S, not a byte or character index.

If `S` is an **AnsiString** and the system is not using a multi-byte character system (MBCS), CharLength always returns 1.

The following example illustrates CharLength's operation.

```
type
  SJISString = type AnsiString(932);
var
  A: SJISString;
  L: Integer;
begin
  A := 'A' +
       'B' +
       #$82#$A0 +  // Japanese Hiragana 'A'
       #$82#$A2 +  // Japanese Hiragana 'I'
       #$82#$A4 +  // Japanese Hiragana 'U'
       'C' +
       'D';

  L := CharLength(A, 1);     //returns 1 ('A')
  L := CharLength(A, 2);     //returns 1 ('B')
  L := CharLength(A, 3);     //returns 2
  L := CharLength(A, 4);     //returns 1
end;
```

In this example, when the index is 1 or 2, it points to the beginning of a single byte character, so the function returns 1. When the index is 3, it points to the beginning of a two byte character, and the function returns 2. When the index is 4, it points to the second half of a two byte character and returns 1. Note that for this example, the element size is 1. Some characters require two elements, and some only need one element.

In general, CharLength is only useful when you start indexing at the beginning of a string and note where multibyte characters start as the index increases.

This function also works for Unicode characters:

```
var
  U: UnicodeString;
  L: Integer;
begin
  U := 'abc';
  L := SysUtils.CharLength(U,1); //returns 2
  L := SysUtils.CharLength(U,2); //returns 2

  U := #$20BB7; //surrogate pair
  L := SysUtils.CharLength(U,1); //returns 4
  L := SysUtils.CharLength(U,2); //returns 2
  end;
```

Note that the element size is 2 in this example, and the surrogate pair character consists of two elements.

When the string is 'abc', each character is a single two byte element, so the function returns 2. String literals are Unicode by default.

For the surrogate pair, when the index is 1, it points to the first element in the character, so the function returns 4. When the index is 2, it points to the second element in the character and returns 2.

See Also

CharLength

37.3.44 SysUtils.CharToByteIndex Function

Returns the 1-based index of the first byte of a specified character in a string.

Pascal

```
function CharToByteIndex(const S: AnsiString; Index: Integer): Integer; overload;
function CharToByteIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int CharToByteIndex(const AnsiString S, int Index);
int CharToByteIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call CharToByteIndex to obtain the byte offset for the character at a specified position in the string S, where 1 specifies the first character, 2 specifies the second character, and so on. CharToByteIndex returns zero if Index is out of range (Index <= 0 or S contains fewer than Index characters). If the Index parameter indicates a double-byte character, CharToByteIndex returns the offset of the first byte.

If the system is not using a multi-byte character system (MBCS), CharToByteIndex returns the value of Index, because there is a 1:1 correspondence between bytes and characters.

See Also

ByteToCharIndex (see page 4069)

CharToByteLen (see page 4076)

37.3.45 SysUtils.CharToByteLen Function

Returns the length, in bytes, of the first MaxLen characters of a string.

Pascal

```
function CharToByteLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
function CharToByteLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int CharToByteLen(const AnsiString S, int MaxLen);
int CharToByteLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

Call CharToByteLen to convert the length of a string in characters to the corresponding length in bytes. CharToByteLen returns the number of bytes required by the first MaxLen characters of S. If S contains fewer than MaxLen characters, CharToByteLen returns the byte length of S.

If the system is not using a multi-byte character system (MBCS), CharToByteLen returns the length of S or MaxLen, whichever is smaller.

See Also

ByteToCharLen (see page 4070)

CharToByteIndex (see page 4075)

37.3.46 SysUtils.CharToElementIndex Function

Pascal

```
function CharToElementIndex(const S: AnsiString; Index: Integer): Integer; overload;
```

C++

```
int CharToElementIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.CharToElementIndex.

37.3.47 SysUtils.CharToElementIndex Function

Pascal

```
function CharToElementIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int CharToElementIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.CharToElementIndex.

37.3.48 SysUtils.CharToElementLen Function

Pascal

```
function CharToElementLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
```

C++

```
int CharToElementLen(const AnsiString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.CharToElementLen.

37.3.49 SysUtils.CharToElementLen Function

Pascal

```
function CharToElementLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int CharToElementLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.CharToElementLen.

37.3.50 SysUtils.CheckWin32Version Function

Check to see if you are running on a specific level (or higher) of the Windows 32 bit Operating System (see page 3159).

Pascal

```
function CheckWin32Version(AMajor: Integer; AMinor: Integer = 0): Boolean;
```

C++

```
Boolean CheckWin32Version(int AMajor, int AMinor = 0);
```

File

SysUtils

Description

The Windows Operating System (see page 3159) Major.Minor version number is compared with the passed AMajor and AMinor values. AMinor defaults to 0 if not supplied.

CheckWin32Version Returns true if the Windows OS Major.Minor version number is >= the passed AMajor.AMinor value.

Note: Prior to Delphi 7, this routine returned a different result.

See Also

Win32MajorVersion (see page 4365)Win32MinorVersion (see page 4366)

37.3.51 SysUtils.CompareMem Function

Performs a binary comparison of two memory images.

Pascal

```
function CompareMem(P1: Pointer; P2: Pointer; Length: Integer): Boolean;
```

C++

```
Boolean CompareMem(void * P1, void * P2, int Length);
```

File

SysUtils

Description

CompareMem performs a binary compare of Length (see page 3607) bytes of memory referenced by P1 to that of P2. CompareMem returns true if the memory referenced by P1 is identical to that of P2.

See Also

CompareStr (see page 4078)

CompareText (see page 4079)

CollectionsEqual

37.3.52 SysUtils.CompareStr Function

Compares two strings case sensitively.

Pascal

```
function CompareStr(const S1: string; const S2: string): Integer; overload;
function CompareStr(const S1: string; const S2: string; LocaleOptions: TLocaleOptions):
Integer; overload;
```

C++

```
int CompareStr(const UnicodeString S1, const UnicodeString S2);
int CompareStr(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

CompareStr compares S1 to S2, with case-sensitivity. The return value is less than 0 if S1 is less than S2, 0 if S1 equals S2, or greater than 0 if S1 is greater than S2. The compare operation is based on the 8-bit ordinal value of each character and is not affected by the current locale.

C++ Examples:

```
/*
The following code compares String1, 'STEVE', to String2,
'STEVe'. Note that CompareStr returns a number less than 0
because the value of 'e' is greater than the value of 'E'.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString String1 = "STEVE";
  AnsiString String2 = "Steve";
  int I = CompareStr(String1, String2);  // the value of I is < 0
  Edit1->Text = IntToStr(I);
  if (I != 0)
    ShowMessage ("The strings are not equal");
}
```

Delphi Examples:

```
{
The following code compares String1, 'STEVE', to String2,
'STEVe'. Note that CompareStr returns a number less than 0
because the value of 'e' is greater than the value of 'E'.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  String1, String2 : string;
  I : Integer;
begin
  String1 := 'STEVE';
  String2 := 'STEVe';
  I := SysUtils.CompareStr(String1, String2);  { the value of I is < 0 }
  Edit1.Text := InttoStr(I);
  if I <> 0 then
    MessageDlg('The strings are not equal', mtWarning, [mbOK], 0)
end;
```

See Also

CompareText (see page 4079)

AnsiCompareStr (see page 4050)

37.3.53 SysUtils.CompareText Function

Compares two strings by ordinal value without case sensitivity.

Pascal

```
function CompareText(const S1: string; const S2: string): Integer; overload;
function CompareText(const S1: string; const S2: string; LocaleOptions: TLocaleOptions):
Integer; overload;
```

C++

```
int CompareText(const UnicodeString S1, const UnicodeString S2);
int CompareText(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

CompareText compares S1 and S2 and returns 0 if they are equal. If S1 is greater than S2, CompareText returns an integer greater than 0. If S1 is less than S2, CompareText returns an integer less than 0. CompareText is not case sensitive and is not affected by the current locale.

C++ Examples:

```
/*
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
*/
int __stdcall CustomSortProc(long Item1, long Item2, long ParamSort)
{
  return -CompareText((reinterpret_cast<TListItem *>(Item1))->Caption,
                      (reinterpret_cast<TListItem *>(Item2))->Caption);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListView1->CustomSort(CustomSortProc, 0);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *ListItem;
  ListView1->ViewStyle = vsList;
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Apples";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Oranges";
  ListItem = ListView1->Items->Add();
  ListItem->Caption = "Pears";
}
/*
This example shows how to use the OnColumnClick and
OnCompare events of a list view to let users sort the
columns in a report-style list view by clicking on the
column headers. This requires a global variable to keep
track of the column that was clicked.  The OnColumnClick
event handler sets the global variable to indicate the
column to sort and calls AlphaSort:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TListItem *Item;
  TListColumn *Column;
  const char imagenames[3][20] = {
    "C++ Image", "Borland Image", "Delphi Image"};
  const char Col2Array[3][40] = {
    "Documentation for the C++ icon.",
    "Borland icon.", "Delphi icon."
    };
  // Create a ListView item for each image in the ImageList
  ListView1->SmallImages = ImageList1;
  ListView1->LargeImages = ImageList1;
```

```
  for (int i = 0; i < ImageList1->Count; i++)
  {
    Item = ListView1->Items->Add();
    Item->Caption = imagenames[i];
    Item->ImageIndex = i;
    Item->SubItems->Add(Col2Array[i]);
  }
  // Create two columns to show during viewing as vsReport
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 1";
  Column->Width = 200;
  Column = ListView1->Columns->Add();
  Column->Caption = "Column 2";
  Column->Width = 200;
  ListView1->ViewStyle = vsReport;
}

int ColumnToSort = 0;

/*
The OnColumnClick event handler sets the global variable to
indicate the column to sort and calls AlphaSort:
*/

void __fastcall TForm1::ListView1ColumnClick(TObject *Sender,
      TListColumn *Column)
{
  ColumnToSort = Column->Index;
  dynamic_cast<TCustomListView *>(Sender)->AlphaSort();
}

/*
The OnCompare event handler causes the list view to sort on
the selected column:
*/

void __fastcall TForm1::ListView1Compare(TObject *Sender, TListItem *Item1,
      TListItem *Item2, int Data, int &Compare)
{
  if (ColumnToSort == 0)
    Compare = CompareText(Item1->Caption,Item2->Caption);
  else
  {
    int ix = ColumnToSort - 1;
    Compare =
      CompareText(Item1->SubItems->Strings[ix], Item2->SubItems->Strings[ix]);
  }
}
```

Delphi Examples:

```
{
The following code orders a list view in reverse
alphabetical order on the click of a button. The callback
function CustomSortProc calls the global CompareText
function and negates its return value.
}
function CustomSortProc(Item1, Item2: TListItem; ParamSort: integer): integer; stdcall;
begin
  Result := -CompareText(Item1.Caption,Item2.Caption);
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListView1.CustomSort(@CustomSortProc, 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
  var ListItem : TListItem;
  begin
    ListView1.ViewStyle := vsList;
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Apples';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Oranges';
    ListItem := ListView1.Items.Add;
    ListItem.Caption := 'Pears';
  end;
  {
  This example shows how to use the OnColumnClick and
  OnCompare events of a list view to let users sort the
  columns in a report-style list view by clicking on the
  column headers. This requires a global variable to keep
  track of the column that was clicked.  The OnColumnClick
  event handler sets the global variable to indicate the
  column to sort and calls AlphaSort:
  }
  var ColumnToSort: Integer;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    I: Integer;
    ListItem: TListItem;
    NewColumn: TListColumn;
  const
    imagenames: Array[0..2] of string = ('C++ Image', 'Borland Image', 'Delphi Image');
  begin
    Col2Array[0]:= 'Documentation for the C++ icon.';
    Col2Array[1]:= 'Borland icon.';
    Col2Array[2]:= 'Delphi icon.';
    // Create a ListView item for each image in the ImageList
    with ListView1 do
    begin
      SmallImages := ImageList1;
      LargeImages := ImageList1;
      for I := 0 to ImageList1.Count - 1 do
      begin
        ListItem := Items.Add;
        Listitem.Caption := imagenames[I];
        ListItem.ImageIndex := I;
        Listitem.SubItems.Add(Col2Array[I]);
      end;
      // Create two columns to show during viewing as vsReport
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 1';
      NewColumn.Width := 200;
      NewColumn := Columns.Add;
      NewColumn.Caption := 'Column 2';
      NewColumn.Width := 200;
      ListView1.ViewStyle := vsReport;
    end;
  end;

  procedure TForm1.ListView1ColumnClick(Sender: TObject; Column: TListColumn);
  begin
    ColumnToSort := Column.Index;
    (Sender as TCustomListView).AlphaSort;
  end;

  {
  The OnCompare event handler causes the list view to sort on
  the selected column:
  }
  procedure TForm1.ListView1Compare(Sender: TObject; Item1, Item2: TListItem; Data: Integer; var
```

```
  Compare: Integer);
  var
    ix: Integer;
  begin
    if ColumnToSort = 0 then
      Compare := CompareText(Item1.Caption,Item2.Caption)
    else begin
     ix := ColumnToSort - 1;
     Compare := CompareText(Item1.SubItems[ix],Item2.SubItems[ix]);
    end;
  end;
  {
  Note:   This OnCompare event handler uses the global
  CompareText function. An application may want to use
  AnsiCompareText, CompareStr, or AnsiCompareStr instead,
  depending on whether the comparison should be case-sensitive
  and whether the locale should be considered.
  }
```

See Also

CompareStr (see page 4078)

AnsiCompareText (see page 4051)

CompareMem (see page 4078)

SameText (see page 4239)

37.3.54 SysUtils.CreateDir Function

Creates a new directory.

Pascal

```
function CreateDir(const Dir: string): Boolean;
```

C++

```
Boolean CreateDir(const UnicodeString Dir);
```

File

SysUtils

Description

CreateDir creates a new directory. The return value is true if a new directory was successfully created, or false if an error occurred.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
```

```
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

See Also

ForceDirectories (see page 4186)

ExtractFileDir (see page 4112)

GetCurrentDir (see page 4202)

RemoveDir (see page 4232)

SetCurrentDir (see page 4240)

37.3.55 SysUtils.CreateGUID Function

Creates a globally unique identifier.

Pascal

```
function CreateGUID(out Guid: TGUID): HResult; stdcall;
```

C++

```
__stdcall HRESULT CreateGUID(TGUID Guid);
```

File

SysUtils

Description

CreateGUID sets Guid to a new-created Globally Unique Identifier.

Note: Under Windows, CreateGUID calls the Windows API CoCreateGUID.

Note: Under Linux, CreateGUID calls uuid_generate_time.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
```

```
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);
```

```
    { Testing if 2 guids are equal }
    if IsEqualGUID(MyGuid0, MyGuid1) then
       Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                       '2 randomly generated GUIDs cannot be equal!');
end;
```

37.3.56 SysUtils.CurrentYear Function

Returns the current year.

Pascal

```
function CurrentYear: Word;
```

C++

```
Word CurrentYear();
```

File

SysUtils

Description

CurrentYear returns the current year on the Gregorian calendar. The value returned is the complete four-digit year.

37.3.57 SysUtils.CurrToStr Function

Formats a Currency (see page 3235) value as a string.

Pascal

```
function CurrToStr(Value: Currency): string; overload;
function CurrToStr(Value: Currency; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString CurrToStr(Currency Value);
UnicodeString CurrToStr(Currency Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

CurrToStr converts the Currency (see page 3235) value given by Value to its string representation. The conversion uses the ffGeneral floating point conversion rules.

The first form of CurrToStr is not thread-safe, because it uses localization information contained in global variables. The second form of CurrToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of CurrToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.58 SysUtils.CurrToStrF Function

Converts a Currency (see page 3235) value to a string, using a specified format.

Pascal

```
function CurrToStrF(Value: Currency; Format: TFloatFormat; Digits: Integer): string; overload;
function CurrToStrF(Value: Currency; Format: TFloatFormat; Digits: Integer; const
FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString CurrToStrF(Currency Value, TFloatFormat Format, int Digits);
UnicodeString CurrToStrF(Currency Value, TFloatFormat Format, int Digits, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

CurrToStrF converts the Currency (see page 3235) value given by Value to its string representation using the indicated float format convention. A call to CurrToStrF corresponds to a call to FloatToStrF (see page 4182) with an implied precision of 19 digits.

The first form of CurrToStrF is not thread-safe, because it uses localization information contained in global variables. The second form of CurrToStrF, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of CurrToStrF, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.59 SysUtils.Date Function

Returns the current date.

Pascal

```
function Date: TDateTime;
```

C++

```
TDateTime Date();
```

File

SysUtils

Description

Use Date to obtain the current local date as a TDateTime (see page 3323) value. The time portion of the value is 0 (midnight).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Today is  ' + SysUtils.DateToStr(Date);
end;
```

See Also

DateToStr (see page 4092)

DecodeDate (see page 4095)

Now (see page 4227)

Time (see page 4297)

Today

Tomorrow

Yesterday

IsToday

37.3.60 SysUtils.DateTimeToFileDate Function

Converts a TDateTime (see page 3323) value to an OS timestamp value.

Pascal

```
function DateTimeToFileDate(DateTime: TDateTime): Integer;
```

C++

```
int DateTimeToFileDate(TDateTime DateTime);
```

File

SysUtils

Description

A timestamp is a signed 32-bit integer, used by the Operating System (see page 3159) to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use DateTimeToFileDate to convert a TDateTime (see page 3323) value to an OS timestamp value.

DateTimeToFileDate is particularly useful for converting the Time (see page 4297) field of the TSearchRec (see page 4330) type used by the FindFirst (see page 4175) and FindNext (see page 4177) functions.

See Also

FileDateToDateTime (see page 4130)

FindFirst (see page 4175)

FindNext (see page 4177)

TSearchRec (see page 4330)

37.3.61 SysUtils.DateTimeToStr Function

Converts a TDateTime (see page 3323) value to a string.

Pascal

```
function DateTimeToStr(const DateTime: TDateTime): string; overload;
function DateTimeToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings):
string; overload;
```

C++

```
UnicodeString DateTimeToStr(const TDateTime DateTime);
UnicodeString DateTimeToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

The first form of DateTimeToStr is not thread-safe, because it uses localization information contained in global variables. The second form of DateTimeToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateTimeToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(Now());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.DateTimeToStr(Now);
end;
```

37.3.62 SysUtils.DateTimeToString Function

Converts a TDateTime (see page 3323) value to a string using a specified Format (see page 4188).

Pascal

```
procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime);
overload;
```

```
procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime;
const FormatSettings: TFormatSettings); overload;
```

C++

```
DateTimeToString(UnicodeString Result, const UnicodeString Format, TDateTime DateTime);
DateTimeToString(UnicodeString Result, const UnicodeString Format, TDateTime DateTime, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

DateTimeToString converts the TDateTime (see page 3323) value given by DateTime using the format string given by Format (see page 4188) into the string variable given by Result. See the table below for information about the supported format strings.

The first form of DateTimeToString is not thread-safe, because it uses localization information contained in global variables. The second form of DateTimeToString, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateTimeToString, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

Date (see page 4087) Time (see page 4297) Format (see page 4188) Strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d"), simply format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables.

In the following table, specifiers are given in lower case. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat (see page 4358) global variable, followed by the time using the format given by the LongTimeFormat (see page 4356) global variable. The time is not displayed if the date-time value indicates midnight precisely.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames (see page 4359) global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames (see page 4355) global variable.
ddddd	Displays the date using the format given by the ShortDateFormat (see page 4358) global variable.
dddddd	Displays the date using the format given by the LongDateFormat (see page 4354) global variable.
e	(Windows only) Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee	(Windows only) Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	(Windows only) Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	(Windows only) Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec (see page 3550)) using the strings given by the ShortMonthNames (see page 4359) global variable.

mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames (see page 4355) global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).
t	Displays the time using the format given by the ShortTimeFormat (see page 4360) global variable.
tt\	Displays the time using the format given by the LongTimeFormat (see page 4356) global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString (see page 4362) global variable for any hour before noon, and the contents of the TimePMString (see page 4362) global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator (see page 4349) global variable.
:	Displays the time separator character given by the TimeSeparator (see page 4363) global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

See Also

DateTimeToStr (see page 4089)

GetLocaleFormatSettings (see page 4206)

37.3.63 SysUtils.DateTimeToSystemTime Function

Converts a TDateTime (see page 3323) value into the Win32 API's system time type.

Pascal

```
procedure DateTimeToSystemTime(const DateTime: TDateTime; var SystemTime: TSystemTime);
```

C++

```
DateTimeToSystemTime(const TDateTime DateTime, TSystemTime SystemTime);
```

File

SysUtils

Description

Use DateTimeToSystemTime to convert a TDateTime (see page 3323) value into a TSystemTime (Delphi) or _SYSTEMTIME (C++) value. This procedure is primarily used to convert TDateTime (see page 3323) values for making a Windows API call.

See Also

SystemTimeToDateTime (see page 4295)

37.3.64 SysUtils.DateTimeToTimeStamp Function

Converts a TDateTime (see page 3323) value into the corresponding TTimeStamp (see page 4334) value.

Pascal

```
function DateTimeToTimeStamp(DateTime: TDateTime): TTimeStamp;
```

C++

```
TTimeStamp DateTimeToTimeStamp(TDateTime DateTime);
```

File

SysUtils

Description

Call DateTimeToTimeStamp to convert a TDateTime (see page 3323) value into a TTimeStamp (see page 4334) value.

See Also

TimeStampToDateTime (see page 4298)

37.3.65 SysUtils.DateToStr Function

Converts a TDateTime (see page 3323) value to a string.

Pascal

```
function DateToStr(const DateTime: TDateTime): string; overload;
function DateToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings): string;
overload;
```

C++

```
UnicodeString DateToStr(const TDateTime DateTime);
UnicodeString DateToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use DateToStr to obtain a string representation of a date value that can be used for display purposes.

The first form of DateToStr is not thread-safe, because it uses localization information contained in global variables. The second form of DateToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of DateToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
  Label1->Caption = DateToStr(Date());
}
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateToStr(Date());
}
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeDate((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text));
    Label1->Caption = DateToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid day, month, or year.");
  }
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
{
This example uses a label and a button on a form. When the
user clicks the button, the current date is displayed in the
caption of the label:
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Today is  ' + SysUtils.DateToStr(Date);
end;
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeDate(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text));
  Label1.Caption := DateToStr(MyDate);
end;
```

37.3.66 SysUtils.DayOfWeek Function

Returns the day of the week for a specified date.

Pascal

```
function DayOfWeek(const DateTime: TDateTime): Word;
```

C++

```
Word DayOfWeek(const TDateTime DateTime);
```

File

SysUtils

Description

DayOfWeek returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

Note: DayOfWeek is not compliant with the ISO 8601 standard, which defines Monday as the first day of the week. For an ISO 8601 compliant version, use the DayOfTheWeek function instead.

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char days[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday" };
  TDateTime dtDate = StrToDate(Edit1->Text);
  ShowMessage(Edit1->Text + AnsiString(" is a ") + days[dtDate.DayOfWeek() - 1]);
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
  days: array[1..7] of string;
begin
  days[1] := 'Sunday';
  days[2] := 'Monday';
  days[3] := 'Tuesday';
  days[4] := 'Wednesday';
  days[5] := 'Thursday';
  days[6] := 'Friday';
  days[7] := 'Saturday';
  ADate := StrToDate(Edit1.Text);
  ShowMessage(Edit1.Text + ' is a ' + days[SysUtils.DayOfWeek(ADate)]);
end;
```

See Also

Date (see page 4087)

EncodeDate (see page 4105)

DayOfTheMonth

DayOfTheYear

DayOfTheWeek

37.3.67 SysUtils.DecodeDate Function

Returns Year, Month, and Day values for a TDateTime (see page 3323) value.

Pascal

```
procedure DecodeDate(const DateTime: TDateTime; var Year: Word; var Month: Word; var Day:
Word);
```

C++

```
DecodeDate(const TDateTime DateTime, Word Year, Word Month, Word Day);
```

File

SysUtils

Description

The DecodeDate procedure breaks the value specified as the Date (see page 4087) parameter into Year, Month, and Day values. If the given TDateTime (see page 3323) value has a negative (BC) year, the year, month, and day return parameters are all set to zero.

C++ Examples:

```
/*
This example uses a button and two labels on a form. When
```

```
  the user clicks the button, the current date and time are
  reported in the captions of the two labels.
  */
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    Word Year, Month, Day, Hour, Min, Sec, MSec;
    TDateTime dtPresent = Now();
    DecodeDate(dtPresent, Year, Month, Day);
    Label1->Caption = AnsiString("Today is Day ") + IntToStr(Day) + AnsiString(" of Month ") +
  IntToStr(Month) + AnsiString(" of Year ") + IntToStr(Year);
    DecodeTime(dtPresent, Hour, Min, Sec, MSec);
    Label2->Caption = AnsiString("The time is Minute ") + IntToStr(Min) + AnsiString(" of Hour
  ") + IntToStr(Hour);
  }
```

Delphi Examples:

```
  {
  This example uses a button and two labels on a form. When
  the user clicks the button, the current date and time are
  reported in the captions of the two labels.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Present: TDateTime;
    Year, Month, Day, Hour, Min, Sec, MSec: Word;
   begin
    Present:= Now;
    SysUtils.DecodeDate(Present, Year, Month, Day);
    Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
      + IntToStr(Month) + ' of Year ' + IntToStr(Year);
    SysUtils.DecodeTime(Present, Hour, Min, Sec, MSec);
    Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
      + IntToStr(Hour);
  end;
```

See Also

DecodeDateFully (see page 4096)

DecodeTime (see page 4097)

DayOf

YearOf

MonthOf

DecodeDateTime

37.3.68 SysUtils.DecodeDateFully Function

Returns Year, Month, and Day, and Day-of-Week values for a TDateTime (see page 3323) value.

Pascal

```
  function DecodeDateFully(const DateTime: TDateTime; var Year: Word; var Month: Word; var Day:
  Word; var DOW: Word): Boolean;
```

C++

```
  Boolean DecodeDateFully(const TDateTime DateTime, Word Year, Word Month, Word Day, Word DOW);
```

File

SysUtils

Description

DecodeDateFully procedure breaks the value specified as the Date (see page 4087) parameter into Year, Month, Day, and DOW values. If Year is a leap year, the function return value is True; otherwise the function return value is false. If the given TDateTime (see page 3323) value has a negative (BC) year, the return parameters are all set to zero.

See Also

DecodeDate (see page 4095)

DecodeTime (see page 4097)

DayOf

YearOf

MonthOf

DecodeDateTime

37.3.69 SysUtils.DecodeTime Function

Breaks a TDateTime (see page 3323) value into hours, minutes, seconds, and milliseconds.

Pascal

```
procedure DecodeTime(const DateTime: TDateTime; var Hour: Word; var Min: Word; var Sec: Word;
var MSec: Word);
```

C++

```
DecodeTime(const TDateTime DateTime, Word Hour, Word Min, Word Sec, Word MSec);
```

File

SysUtils

Description

DecodeTime breaks the object specified as the Time (see page 4297) parameter into hours, minutes, seconds, and milliseconds.

C++ Examples:

```
/*
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Word Year, Month, Day, Hour, Min, Sec, MSec;
  TDateTime dtPresent = Now();
  DecodeDate(dtPresent, Year, Month, Day);
  Label1->Caption = AnsiString("Today is Day ") + IntToStr(Day) + AnsiString(" of Month ") +
IntToStr(Month) + AnsiString(" of Year ") + IntToStr(Year);
  DecodeTime(dtPresent, Hour, Min, Sec, MSec);
  Label2->Caption = AnsiString("The time is Minute ") + IntToStr(Min) + AnsiString(" of Hour
") + IntToStr(Hour);
}
```

Delphi Examples:

```
{
This example uses a button and two labels on a form. When
the user clicks the button, the current date and time are
reported in the captions of the two labels.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day, Hour, Min, Sec, MSec: Word;
 begin
  Present:= Now;
  SysUtils.DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Today is Day ' + IntToStr(Day) + ' of Month '
    + IntToStr(Month) + ' of Year ' + IntToStr(Year);
  SysUtils.DecodeTime(Present, Hour, Min, Sec, MSec);
  Label2.Caption := 'The time is Minute ' + IntToStr(Min) + ' of Hour '
    + IntToStr(Hour);
end;
```

See Also

DecodeDate (see page 4095)

EncodeTime (see page 4106)

Time (see page 4297)

HourOf

MinuteOf

SecondOf

MilliSecondOf

DecodeDateTime

37.3.70 SysUtils.DelegatesEqual Function

Pascal

```
function DelegatesEqual(A: Pointer; B: Pointer): Boolean;
```

C++

```
Boolean DelegatesEqual(void * A, void * B);
```

File

SysUtils

Description

This is function SysUtils.DelegatesEqual.

37.3.71 SysUtils.DeleteFile Function

Deletes a file from disk.

Pascal

```
function DeleteFile(const FileName: string): Boolean;
```

C++

```
Boolean DeleteFile(const UnicodeString FileName);
```

File

SysUtils

Description

DeleteFile deletes the file named by FileName from the disk. If the file cannot be deleted or does not exist, the function returns false.

C++ Examples:

```
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
/*
Click the button to open a TOpenDialog, then select a file
to delete.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
      DeleteFile(OpenDialog1->FileName);
  }
}
```

Delphi Examples:

```
{
The following code prompts for confirmation before deleting
a file.  Create a file and then enter the path and file name
in the textfields.  Click the button to delete it.
}
procedure TForm1.Button1Click(Sender: TObject);
var FileName: string;
begin
```

```
  Filename:= Edit1.Text;
  if SysUtils.FileExists(FileName) then
  begin
    if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
      DeleteFile(FileName);
  end
  else
    MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
[mbOK], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit2.Text:= ExtractFilePath(Application.ExeName);
end;
```

37.3.72 SysUtils.DirectoryExists Function

Determines whether a specified directory exists.

Pascal

```
function DirectoryExists(const Directory: string): Boolean;
```

C++

```
Boolean DirectoryExists(const UnicodeString Directory);
```

File

SysUtils

Description

Call DirectoryExists to determine whether the directory specified by the **Directory** parameter exists. If the directory exists, the function returns true. If the directory does not exist, the function returns false.

If a full path name is entered, DirectoryExists searches for the directory along the designated path. Otherwise, the **Directory** parameter is interpreted as a relative path name from the current directory.

Note: The FileCtrl unit (Windows only) also contains a DirectoryExists function. However, the FileCtrl version is deprecated, and the SysUtils (see page 3901) version preferred, even if the code does not need to be cross-platform.

C++ Examples:

```
/*
The following example creates a directory 'C:\temp' if it
does not already exist.
}
*/
#include <Filectrl.hpp>
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (!DirectoryExists("c:\\temp"))
  {
    if (!CreateDir("C:\\temp"))
      throw Exception("Cannot create c:\\temp directory.");
  }
  if (!CreateDir("C:\\temp"))
    throw Exception("Cannot create C:\\temp again!");
}
```

Delphi Examples:

```
{
The following example creates a directory 'C:\temp' if it
does not already exist.
}
uses FileCtrl;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not SysUtils.DirectoryExists('C:\temp') then
    if not CreateDir('C:\temp') then
      raise Exception.Create('Cannot create C:\temp');
  if not CreateDir('C:\temp') then
    raise Exception.Create('Cannot create C:\temp again!');
end;
```

See Also

ForceDirectories (see page 4186)

SelectDirectory

FileExists (see page 4133)

37.3.73 SysUtils.DiskFree Function

Returns the number of free bytes on a specified drive.

Pascal

```
function DiskFree(Drive: Byte): Int64;
```

C++

```
Int64 DiskFree(Byte Drive);
```

File

SysUtils

Description

DiskFree returns the number of free bytes on the specified drive, where 0 = Current, 1 = A, 2 = B, and so on.

DiskFree returns -1 if the drive number is invalid.

Note: DiskFree is only available on Windows.

C++ Examples:

```
/*
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  __int64 AmtFree = DiskFree(0);
  __int64 Total = DiskSize(0);
  AnsiString S;
  S.sprintf("%I64d percent of the space on drive 0 is free: %I64d KB", AmtFree*100/Total,
AmtFree/1024 );
  Label1->Caption = S;
}
```

```
/*
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ShowMessage(FormatFloat("#,##0 bytes", DiskFree(0)));
}
```

Delphi Examples:

```
{
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  AmtFree: Int64;
  Total:   Int64;
begin
  AmtFree := SysUtils.DiskFree(0);
  Total := DiskSize(0);
  S := IntToStr((AmtFree  * 100) div Total) + ' percent of the space on drive 0 is free: ' +
IntToStr(AmtFree div 1024) + ' Kbytes free. ';
  Label1.Caption := S;
end;
{
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(FormatFloat('#,##0 bytes', DiskFree(0)));
end;
```

See Also

DiskSize (see page 4102)

37.3.74 SysUtils.DiskSize Function

Returns the size, in bytes, of a specified drive.

Pascal

```
function DiskSize(Drive: Byte): Int64;
```

C++

```
Int64 DiskSize(Byte Drive);
```

File

SysUtils

Description

DiskSize returns the size in bytes of the specified drive, where 0 = Current, 1 = A, 2 = B, etc. DiskSize returns -1 if the drive number is invalid.

Note: DiskSize is only available on Windows.

C++ Examples:

```
/*
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  __int64 AmtFree = DiskFree(0);
  __int64 Total = DiskSize(0);
  AnsiString S;
  S.sprintf("%I64d percent of the space on drive 0 is free: %I64d KB", AmtFree*100/Total,
AmtFree/1024 );
  Label1->Caption = S;
}
```

Delphi Examples:

```
{
This example uses a form with a label on it. When the
following code executes, it displays a message in the label
indicating the number of KB free, and what percentage of the
entire disk space that represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S: string;
  AmtFree: Int64;
  Total:   Int64;
begin
  AmtFree := SysUtils.DiskFree(0);
  Total := DiskSize(0);
  S := IntToStr((AmtFree  * 100) div Total) + ' percent of the space on drive 0 is free: ' +
IntToStr(AmtFree div 1024) + ' Kbytes free. ';
  Label1.Caption := S;
end;
```

See Also

DiskFree (see page 4101)

37.3.75 SysUtils.DisposeStr Function

Disposes of a string pointer that was allocated using NewStr (see page 4227).

Pascal

```
procedure DisposeStr(P: PAnsiString); deprecated;
```

C++

```
DisposeStr(PAnsiString P);
```

File

SysUtils

Description

DisposeStr is only provided for backward compatibility with existing code that allocated a string using NewStr (see page 4227). Both DisposeStr and NewStr (see page 4227) are obsolete.

See Also

NewStr (see page 4227)

37.3.76 SysUtils.ElementToCharIndex Function

Pascal

```
function ElementToCharIndex(const S: AnsiString; Index: Integer): Integer; overload;
```

C++

```
int ElementToCharIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharIndex.

37.3.77 SysUtils.ElementToCharIndex Function

Pascal

```
function ElementToCharIndex(const S: UnicodeString; Index: Integer): Integer; overload;
```

C++

```
int ElementToCharIndex(const UnicodeString S, int Index);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharIndex.

37.3.78 SysUtils.ElementToCharLen Function

Pascal

```
function ElementToCharLen(const S: AnsiString; MaxLen: Integer): Integer; overload;
```

C++

```
int ElementToCharLen(const AnsiString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharLen.

37.3.79 SysUtils.ElementToCharLen Function

Pascal

```
function ElementToCharLen(const S: UnicodeString; MaxLen: Integer): Integer; overload;
```

C++

```
int ElementToCharLen(const UnicodeString S, int MaxLen);
```

File

SysUtils

Description

This is function SysUtils.ElementToCharLen.

37.3.80 SysUtils.EncodeDate Function

Returns a TDateTime (see page 3323) value that represents a specified Year, Month, and Day.

Pascal

```
function EncodeDate(Year: Word; Month: Word; Day: Word): TDateTime;
```

C++

```
TDateTime EncodeDate(Word Year, Word Month, Word Day);
```

File

SysUtils

Description

EncodeDate returns a TDateTime (see page 3323) value from the values specified as the Year, Month, and Day parameters.

The year must be between 1 and 9999.

Valid Month values are 1 through 12.

Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

If the specified values are not within range, EncodeDate raises an EConvertError (see page 3919) exception.

C++ Examples:

```
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeDate((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
```

```
(Word)StrToInt(Edit3->Text));
    Label1->Caption = DateToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid day, month, or year.");
  }
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeDate(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text));
  Label1.Caption := DateToStr(MyDate);
end;
```

See Also

DateToStr (see page 4092)

DecodeDate (see page 4095)

EncodeTime (see page 4106)

EncodeDateTime

TryEncodeDate (see page 4302)

37.3.81 SysUtils.EncodeTime Function

Returns a TDateTime (see page 3323) value for a specified Hour, Min, Sec, and MSec.

Pascal

```
function EncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word): TDateTime;
```

C++

```
TDateTime EncodeTime(Word Hour, Word Min, Word Sec, Word MSec);
```

File

SysUtils

Description

EncodeTime encodes the given hour, minute, second, and millisecond into a TDateTime (see page 3323) value.

Valid Hour values are 0 through 23.

Valid Min and Sec values are 0 through 59.

Valid MSec values are 0 through 999.

If the specified values are not within range, EncodeTime raises an EConvertError (see page 3919) exception.

The resulting value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

C++ Examples:

```
/*
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeTime((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text), (Word)StrToInt(Edit4->Text));
    Label1->Caption = TimeToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid hour, minute, second, or millisecond.");
  }
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeTime(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text), StrToInt(Edit4.Text));
  Label1.Caption := TimeToStr(MyDate);
end;
```

See Also

DecodeTime (see page 4097)

EncodeDate (see page 4105)

EConvertError (see page 3919)

EncodeDateTime

37.3.82 SysUtils.ExceptionErrorMessage Function

Formats a standard error message.

Pascal

```
function ExceptionErrorMessage(ExceptObject: TObject; ExceptAddr: Pointer; Buffer: PChar;
```

```
  Size: Integer): Integer;
```

C++

```
  int ExceptionErrorMessage(TObject * ExceptObject, void * ExceptAddr, const char * Buffer, int
  Size);
```

File

SysUtils

Description

Call ExceptionErrorMessage to assemble an error message that indicates the object associated with the current exception and the location at which the exception was raised. The assembled message is written into the buffer provided by the Buffer parameter. Specify the size of this buffer using the Size parameter.

ExceptionErrorMessage returns the number of bytes in the assembled error message.

See Also

ExceptAddr (see page 3562)

ExceptObject (see page 3562)

ShowException (see page 4240)

37.3.83 SysUtils.ExcludeTrailingBackslash Function

Returns a path name without a trailing delimiter.

Pascal

```
  function ExcludeTrailingBackslash(const S: string): string; overload; platform;
```

C++

```
  UnicodeString ExcludeTrailingBackslash(const UnicodeString S);
```

File

SysUtils

Description

ExcludeTrailingBackslash returns S with any trailing path delimiter ('\' on Windows, '/' on Linux) removed. This function is included for backward compatibility only. New (see page 3619) programs should use ExcludeTrailingPathDelimiter (see page 4109).

Note: ExcludeTrailingBackslash works with multibyte character sets.

See Also

IncludeTrailingBackslash (see page 4213)

IsPathDelimiter (see page 4222)

ExcludeTrailingPathDelimiter (see page 4109)

37.3.84 SysUtils.ExcludeTrailingPathDelimiter Function

Returns a path name without a trailing delimiter.

Pascal

```
function ExcludeTrailingPathDelimiter(const S: string): string; overload;
```

C++

```
UnicodeString ExcludeTrailingPathDelimiter(const UnicodeString S);
```

File

SysUtils

Description

ExcludeTrailingPathDelimiter returns S with any trailing path delimiter ('\' on Windows, '/' on Linux) removed. If the last character in S is a delimiter, the return value is S with the last character removed. Otherwise, S is returned unchanged.

Note: ExcludeTrailingPathDelimiter works with multibyte character sets.

See Also

IncludeTrailingPathDelimiter (see page 4213)

IsPathDelimiter (see page 4222)

37.3.85 SysUtils.ExpandFileName Function

Returns the full path name for a relative file name.

Pascal

```
function ExpandFileName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExpandFileName(const UnicodeString FileName);
```

File

SysUtils

Description

ExpandFileName converts the relative file name into a fully qualified path name. ExpandFileName does not verify that the resulting fully qualified path name refers to an existing file, or even that the resulting path exists.

Note: ExpandFileName does not support UNC format. To expand to a UNC file name, use ExpandUNCFileName (see page 4111).

C++ Examples:

```
/*
The following example uses a listbox, an edit control, and a
button on a form. When the button is clicked, the filename
specified in the edit control is expanded to a fully
```

```
qualified filename and added to the list box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ListBox1->Items->Add(ExpandFileName(Edit1->Text));
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label2->Caption = ExtractFilePath(Application->ExeName);
}
```

Delphi Examples:

```
{
The following example uses a listbox, an edit control, and a
button on a form. When the button is clicked, the filename
specified in the edit control is expanded to a fully
qualified filename and added to the list box.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Add(SysUtils.ExpandFileName(Edit1.Text));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label2.Caption := ExtractFilePath(Application.ExeName);
end;
```

See Also

ExtractFileName (see page 4113)

ExpandUNCFileName (see page 4111)

ExpandFileNameCase (see page 4110)

37.3.86 SysUtils.ExpandFileNameCase Function

Returns the full path name for a relative file name on a case-sensitive file system.

Pascal

```
function ExpandFileNameCase(const FileName: string; out MatchFound: TFilenameCaseMatch):
string; overload;
```

C++

```
UnicodeString ExpandFileNameCase(const UnicodeString FileName, TFilenameCaseMatch MatchFound);
```

File

SysUtils

Description

ExpandFileNameCase expands a relative file name into a fully qualified path name, then attempts to find the file or a close case-insensitive match. This is useful when converting file name data from case-insensitive conventions (Windows) to case-sensitive conventions (Linux). It is also useful for validating file names entered by users unfamiliar with Linux conventions.

FileName is the relative file name to expand.

MatchFound indicates what match, if any, was found for FileName.

The return value is the fully qualified path name generated by ExpandFileNameCase. Its relationship to FileName depends on the value of MatchFound, as indicated in the following table:

MatchFound	Return value
mkNone	The fully qualified path name, as returned by ExpandFileName (see page 4109).
mkExactMatch	The fully qualified path name for the single case-sensitive match to FileName. This is the same value as the one returned when MatchFound is mkNone.
mkSingleMatch	(Linux only.) The fully qualified path name for the single file that matches FileName case-insensitively.
mkAmbiguous	(Linux only.) The fully qualified path name for one of the case-insensitve matches to FileName. This path name is not the same as the specified file, and not necessarily the closest match.

Note: ExpandFileNameCase is much slower than ExpandFileName (see page 4109), particularly when the specified file name is ambiguous or does not match an actual file.

See Also

ExpandFileName (see page 4109)

37.3.87 SysUtils.ExpandUNCFileName Function

Returns the full path of a file name in UNC format, if appropriate.

Pascal

```
function ExpandUNCFileName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExpandUNCFileName(const UnicodeString FileName);
```

File

SysUtils

Description

ExpandUNCFileName returns the fully-qualified file name for a specified file name.

Note: On Windows, ExpandUNCFileName returns a fully qualified path name using Universal Naming Convention for network files. For files on local disks, ExpandUNCFileName is equivalent to ExpandFileName (see page 4109). In a UNC file name, the drive prefix is replaced with the name of the network resource to which the drive is mapped. For example, if the working directory is J:\ourdir and drive J is mapped to \\ourserver\ourshare, ExpandUNCFileName('ourfile') returns "\\ourserver\ourshare\ourdir\ourfile".

Note: On Linux, ExpandUNCFileName is always equivalent to ExpandFileName (see page 4109).

See Also

ExpandFileName (see page 4109)

ExtractFileDrive (see page 4112)

37.3.88 SysUtils.ExtractFileDir Function

Extracts the drive and directory parts from FileName.

Pascal

```
function ExtractFileDir(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileDir(const UnicodeString FileName);
```

File

SysUtils

Description

The resulting string is a directory name suitable for passing to the CreateDir (see page 4083), GetCurrentDir (see page 4202), RemoveDir (see page 4232), and SetCurrentDir (see page 4240) functions. This string is empty if FileName contains no drive and directory parts.

Note: This function works for multi-byte character systems (MBCS).

See Also

CreateDir (see page 4083)

GetCurrentDir (see page 4202)

RemoveDir (see page 4232)

SetCurrentDir (see page 4240)

37.3.89 SysUtils.ExtractFileDrive Function

Returns the drive portion of a file name.

Pascal

```
function ExtractFileDrive(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileDrive(const UnicodeString FileName);
```

File

SysUtils

Description

Note: On Windows, ExtractFileDrive returns a string containing the drive portion of a fully qualified path name for the file passed in the FileName. For file names with drive letters, the result is in the form "'<drive>'". For file names with a UNC path the result is in the form "'\\<servername>\<sharename>'". If the given path contains neither style of path prefix, the result is an empty string.

Note: On Linux, ExtractFileDrive always returns a zero-length string.

See Also

ExpandFileName (see page 4109)

ExpandUNCFileName (see page 4111)

37.3.90 SysUtils.ExtractFileExt Function

Returns the extension portions of a file name.

Pascal

```
function ExtractFileExt(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileExt(const UnicodeString FileName);
```

File

SysUtils

Description

Use ExtractFileExt to obtain the extension from a file name. For example, the following code returns the extension of the file name specified by a variable named MyFileName:

MyFilesExtension := ExtractFileExt(MyFileName); // Delphi

MyFilesExtension = ExtractFileExt(MyFileName); // C++

The resulting string includes the period character that separates the name and extension parts. This string is empty if the given file name has no extension.

Note: This function works for multi-byte character systems (MBCS).

See Also

ExtractFileName (see page 4113)

37.3.91 SysUtils.ExtractFileName Function

Extracts the name and extension parts of a file name.

Pascal

```
function ExtractFileName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFileName(const UnicodeString FileName);
```

File

SysUtils

Description

The resulting string is the rightmost characters of FileName, starting with the first character after the colon or backslash that separates the path information from the name and extension. The resulting string is equal to FileName if FileName contains no drive and directory parts.

Note: This function works for multi-byte character systems (MBCS).

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
```

```
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
```

See Also

ExpandFileName (see page 4109)

ExtractFilePath (see page 4115)

37.3.92 SysUtils.ExtractFilePath Function

Returns the drive and directory portions of a file name.

Pascal

```
function ExtractFilePath(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractFilePath(const UnicodeString FileName);
```

File

SysUtils

Description

The resulting string is the leftmost characters of FileName, up to and including the colon or backslash that separates the path information from the name and extension. The resulting string is empty if FileName contains no drive and directory parts.

This function works for multi-byte character systems (MBCS).

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString temp1 = Edit4->Text;
  AnsiString temp2 = ExtractFileName(Edit1->Text);
  AnsiString NewFileName = temp1 + temp2;
  AnsiString OldFileName =
    Edit3->Text + ExtractFileName(Edit2->Text);
```

```
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, fmOpenRead |
fmShareDenyWrite));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, fmCreate |
fmShareDenyRead));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit3->Text = ExtractFilePath(Application->ExeName);
  Edit4->Text = Edit3->Text;
}
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit3.Text := ExtractFilePath(Application.ExeName);
  Edit4.Text := Edit3.Text;
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  temp1, temp2: string;
  Msg: string;
  NewFile: TFileStream;
  OldFile: TFileStream;
begin
  temp1 := Edit4.Text;
  temp2 := ExtractFileName(Edit1.Text);
```

```
    NewFileName := temp1 + temp2;
    OldFileName :=
      Edit3.Text + ExtractFileName(Edit2.Text);
    Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
    if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
    begin
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;
  {
  The following code prompts for confirmation before deleting
  a file.  Create a file and then enter the path and file name
  in the textfields.  Click the button to delete it.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var FileName: string;
  begin
    Filename:= Edit1.Text;
    if SysUtils.FileExists(FileName) then
    begin
      if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
  mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
        DeleteFile(FileName);
    end
    else
      MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
  [mbOK], 0);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Edit2.Text:= ExtractFilePath(Application.ExeName);
  end;
```

See Also

ExtractFileName (see page 4113)

37.3.93 SysUtils.ExtractRelativePath Function

Returns a relative path name, relative to a specific base directory.

Pascal

```
function ExtractRelativePath(const BaseName: string; const DestName: string): string; overload;
```

C++

```
UnicodeString ExtractRelativePath(const UnicodeString BaseName, const UnicodeString DestName);
```

File

SysUtils

Description

Call ExtractRelativePath to convert a fully qualified path name into a relative path name. The `DestName` parameter specifies file name (including path) to be converted. `BaseName` is the fully qualified name of the base directory to which the returned path name should be relative. `BaseName` may or may not include a file name, but it must include the final path delimiter.

Note: If you send it what should be a directory without the trailing backslash (c:\foo

instead of `c:\foo\`), the function treats `foo` as a file and returns undesirable results. You should correctly call it like this:

```
ExtractRelativePath('c:\foo\', 'c:\bar\')
```

so that it would return

```
'..\bar\'
```

.

```
ExtractRelativePath('c:\foo\', 'c:\bar')
```

would return the incorrect result, probably like

```
'..'
```

.

See Also

ExpandFileName (see page 4109)

37.3.94 SysUtils.ExtractShortPathName Function

Converts a file name to the short 8.3 form.

Pascal

```
function ExtractShortPathName(const FileName: string): string; overload;
```

C++

```
UnicodeString ExtractShortPathName(const UnicodeString FileName);
```

File

SysUtils

Description

ExtractShortPathName converts the file name, FileName, to the short 8.3 form. ExtractShortPathName returns an empty string if the file or directory does not exist.

The first parameter, FileName, can be a full path name. For example, ExtractShortPathName would convert the full path name:

C:\Program Files\MyCompany\MyApp\MyApp.exe

to the short form:

C:\Progra~1\MyComp~1\MyApp\MyApp.exe

Note: ExtractShortPathName is only available on Windows.

See Also

ExpandFileName (see page 4109)

ExpandUNCFileName (see page 4111)

37.3.95 SysUtils.FileAge Function

Returns the OS timestamp of a file.

Pascal

```
function FileAge(const FileName: string): Integer; overload; deprecated;
function FileAge(const FileName: string; out FileDateTime: TDateTime): Boolean; overload;
```

C++

```
int FileAge(const UnicodeString FileName);
Boolean FileAge(const UnicodeString FileName, TDateTime FileDateTime);
```

File

SysUtils

Description

Call FileAge to obtain the OS timestamp of the file specified by `FileName`. The `FileDateTime` output parameter can be later converted to a TDateTime (see page 3323) using the FileDateToDateTime (see page 4130) function. The return value is **True** if the operation succeeded, or **False** otherwise.

Note: The first overloaded version of FileAge is deprecated. Use the second version of FileAge.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
```

```
      NewAttributes = Attributes;
      if (FileAttrDlg->ReadOnly->Checked)
        NewAttributes = NewAttributes | faReadOnly;
      else
         NewAttributes = NewAttributes & ~faReadOnly;

      if (FileAttrDlg->Archive->Checked)
        NewAttributes = NewAttributes | faArchive;
      else
        NewAttributes = NewAttributes & ~faArchive;

      if (FileAttrDlg->System->Checked)
        NewAttributes = NewAttributes | faSysFile;
      else
        NewAttributes = NewAttributes & ~faSysFile;

      if (FileAttrDlg->Hidden->Checked)
        NewAttributes = NewAttributes | faHidden;
      else
        NewAttributes = NewAttributes  & ~faHidden;
      if (NewAttributes != Attributes)
        FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
```

```
        else
          NewAttributes := NewAttributes and not faSysFile;
        if Hidden.Checked then
          NewAttributes := NewAttributes or faHidden
        else
          NewAttributes := NewAttributes and not faHidden;
        if NewAttributes <> Attributes then { if anything changed... }
          FileSetAttr(FileDirName.Caption, NewAttributes);
           { ...write the new values }
      end;
    end;
  end;
```

See Also

FileDateToDateTime (see page 4130)

FileGetDate (see page 4142)

FileExists (see page 4133)

37.3.96 SysUtils.FileClose Function

Closes a specified file.

Pascal

```
procedure FileClose(Handle: Integer);
```

C++

```
FileClose(int Handle);
```

File

SysUtils

Description

FileClose closes a file given its handle. The handle is obtained when the file is opened using FileOpen (see page 4143) or FileCreate (see page 4128).

Note: : When working with Delphi language file variables, use the CloseFile (see page 3539) procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function. Verify
the file written after executing.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
```

```
    if (SaveDialog1->Execute())
    {
      if (FileExists(SaveDialog1->FileName))
      {
        fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
        strcat(szFileName, ".BAK");
        RenameFile(SaveDialog1->FileName, szFileName);
      }
      iFileHandle = FileCreate(SaveDialog1->FileName);
      // Write out the number of rows and columns in the grid.
      FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
  sizeof(StringGrid1->ColCount));
      FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
  sizeof(StringGrid1->RowCount));
      for (int x=0;x<StringGrid1->ColCount;x++)
      {
        for (int y=0;y<StringGrid1->RowCount;y++)
        {
          // Write out the length of each string, followed by the string itself.
          iLength = StringGrid1->Cells[x][y].Length();
          FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
          FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
  StringGrid1->Cells[x][y].Length());
        }
      }
      FileClose(iFileHandle);
    }
  }
  //---------------------------------------------------------------------------
  void __fastcall TForm1::FormCreate(TObject *Sender)
  {
    StringGrid1->Cells[1][0] = "Column 1";
    StringGrid1->Cells[2][0] = "Column 2";
    StringGrid1->Cells[3][0] = "Column 3";
    StringGrid1->Cells[4][0] = "Column 4";
    StringGrid1->Cells[0][1] = "Row 1";
    StringGrid1->Cells[1][1] = "Object";
    StringGrid1->Cells[2][1] = "Pascal";
    StringGrid1->Cells[3][1] = "is";
    StringGrid1->Cells[4][1] = "excellent";
    StringGrid1->Cells[0][2] = "Row 2";
    StringGrid1->Cells[1][2] = "Delphi";
    StringGrid1->Cells[2][2] = "is";
    StringGrid1->Cells[4][2] = "RAD";
  };
  /*
  The following example uses a button, a string grid, and an
  Open dialog box on a form. When the button is clicked, the
  user is prompted for a filename. When the user clicks OK,
  the specified file is opened, read into a buffer, and closed.
  Then the buffer is displayed in two columns of the string
  grid. The first time you use this app, you should do a "Fill
  Grid" and then "Write Grid to File".   Always use files
  generated in this way to do a "Read File into Grid".
  */
  // Read File to Grid

  #include <memory>       //for STL auto_ptr class

  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    int iFileHandle;
    int iFileLength;
    int iBytesRead;
    int iLength;
    if (OpenDialog1->Execute())
```

```
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
4);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
4);
      for (int x = 0; x < StringGrid1->ColCount; x++)
      {
        for (int y = 0; y < StringGrid1->RowCount; y++)
        {
          // Write out the length of each string, followed by the string itself.
          FileRead(iFileHandle, reinterpret_cast<char *>(&iLength), 4);
          std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iLength + 1]);
          FileRead(iFileHandle, pszBuffer.get(), iLength);
          pszBuffer.get()[iLength] = 0;
          StringGrid1->Cells[x][y] = pszBuffer.get();
        }
      }
      FileClose(iFileHandle);
    }
    catch(...)
    {
#if defined(_DELPHI_STRING_UNICODE)
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error",
        IDOK);
#else
      Application->MessageBox(
        "Can't perform one of the following file operations: Open, Seek, Read, Close.",
        "File Error",
        IDOK);
#endif
    }
  }
}

// Write Grid into File

#include <dir.h>

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
```

```
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (AnsiChar *)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
```

```
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
```

```
    { do something here }
    end;
  end;
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
// Read File to Grid
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  X, Y, I: Integer;
  StringLen, colcnt, rowcnt: Integer;
  cellString: String;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//      Buffer := PChar(System.AllocMem(iFileLength + 1));
//      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
//      FileClose(iFileHandle);
//      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
//      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
//      FileSeek(iFileHandle,0,0);
      colcnt := StringGrid1.ColCount;
      iBytesRead := SysUtils.FileRead(
        iFileHandle,
        colcnt,
        4);  // 4 because it's an integer
      rowcnt := StringGrid1.RowCount;
      iBytesRead := SysUtils.FileRead(
        iFileHandle,
        rowcnt,
        4);  // 4 because it's an integer
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          iBytesRead := SysUtils.FileRead(iFileHandle, StringLen, 4);  // 4 because it's an
integer
          Buffer := System.AllocMem(StringLen + 1);
//          GetMem(Buffer, StringLen); { allocate the buffer }
          iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, StringLen);
//          for I := 1 to StringLen do
//            cellString[I] := Buffer[I - 1];
          StringGrid1.Cells[X,Y] := AnsiString(Buffer);
        finally
          FreeMem(Buffer);
        end;
      end;
    end;
    finally
```

```
      FileClose(iFileHandle);
    end;
  end;
end;

// Write Grid into File
procedure TForm1.Button2Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PAnsiChar;
  cellString: AnsiString;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      PAnsiChar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PAnsiChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;
```

See Also

FileCreate (see page 4128)

FileOpen (see page 4143)

CloseFile (see page 3539)

37.3.97 SysUtils.FileCreate Function

Creates a new file.

Pascal

```
function FileCreate(const FileName: string): Integer; overload;
function FileCreate(const FileName: string; Rights: Integer): Integer; overload;
```

C++

```
int FileCreate(const UnicodeString FileName);
int FileCreate(const UnicodeString FileName, int Rights);
```

File

SysUtils

Description

FileCreate creates a new file with the specified name. If the return value is positive, the function was successful and the value is the file handle of the new file. A return value of -1 indicates that an error occurred.

Note: On Windows, the FileAccessRights variable and Rights parameter are ignored.

Note: On Linux, a file has a set of permission bits controlling access to the file. The first form of FileCreate takes the permission bits from the global FileAccessRights variable. The second form takes the permission bits from the Rights parameter. In either case, the permission bits are masked by the current process's umask value.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function. Verify
the file written after executing.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
```

```
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
```

```
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
```

See Also

FileClose (see page 4121)

FileOpen (see page 4143)

AssignFile (see page 3525)

37.3.98 SysUtils.FileDateToDateTime Function

Converts an OS timestamp value to TDateTime (see page 3323) value.

Pascal

```
function FileDateToDateTime(FileDate: Integer): TDateTime;
```

C++

```
TDateTime FileDateToDateTime(int FileDate);
```

File

SysUtils

Description

A timestamp is a signed 32-bit integer, used by the OS to record information such as the date and time a file was modified. The precise format of a timestamp depends on the OS. Use FileDateToDateTime to convert a timestamp to a TDateTime (see page 3323) value.

You may need to use FileDateToDateTime to convert the Time (see page 4297) field of the TSearchRec (see page 4330) type used by the FindFirst (see page 4175) and FindNext (see page 4177) functions.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
```

```
      if (NewAttributes != Attributes)
        FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
    }
  }
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then
        NewAttributes := NewAttributes or faHidden
      else
        NewAttributes := NewAttributes and not faHidden;
      if NewAttributes <> Attributes then { if anything changed... }
        FileSetAttr(FileDirName.Caption, NewAttributes);
         { ...write the new values }
    end;
  end;
end;
```

See Also

DateTimeToFileDate (see page 4088)

FindFirst (see page 4175)

FindNext (see page 4177)

37.3.99 SysUtils.FileExists Function

Tests if a specified file exists.

Pascal

```
function FileExists(const FileName: string): Boolean;
```

C++

```
Boolean FileExists(const UnicodeString FileName);
```

File

SysUtils

Description

FileExists returns true if the file specified by FileName exists. If the file does not exist, FileExists returns false.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function. Verify
the file written after executing.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
```

```
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
/*
The following code prompts for confirmation before deleting
a file.  Create a file and then click on the button to select
the file with a TOpenDialog.  Click Yes to delete it.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Title = "Delete File";
  if (OpenDialog1->Execute())
  {
    if (FileExists(OpenDialog1->FileName))
    {
      if (MessageDlg(
        ("Do you really want to delete " + ExtractFileName(OpenDialog1->FileName) + "?"),
        mtConfirmation,
        TMsgDlgButtons() << mbYes << mbNo,
        0,
        mbNo) == mrYes)
        DeleteFile(OpenDialog1->FileName);
    }
    else
      MessageDlg(("File " + ExtractFileName(OpenDialog1->FileName) + " does not exist."),
mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  }
}
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first time you use this app, you should do a "Fill
Grid" and then "Write Grid to File".   Always use files
generated in this way to do a "Read File into Grid".
*/
// Read File to Grid

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  int iLength;
  if (OpenDialog1->Execute())
```

```
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
4);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
4);
      for (int x = 0; x < StringGrid1->ColCount; x++)
      {
        for (int y = 0; y < StringGrid1->RowCount; y++)
        {
          // Write out the length of each string, followed by the string itself.
          FileRead(iFileHandle, reinterpret_cast<char *>(&iLength), 4);
          std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iLength + 1]);
          FileRead(iFileHandle, pszBuffer.get(), iLength);
          pszBuffer.get()[iLength] = 0;
          StringGrid1->Cells[x][y] = pszBuffer.get();
        }
      }
      FileClose(iFileHandle);
    }
    catch(...)
    {
#if defined(_DELPHI_STRING_UNICODE)
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error",
        IDOK);
#else
      Application->MessageBox(
        "Can't perform one of the following file operations: Open, Seek, Read, Close.",
        "File Error",
        IDOK);
#endif
    }
  }
}

// Write Grid into File

#include <dir.h>

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
```

```
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (AnsiChar *)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
```

```
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
    Cells[2,1] := 'Pascal';
    Cells[3,1] := 'is';
    Cells[4,1] := 'excellent';
    Cells[0,2] := 'Row 2';
    Cells[1,2] := 'Delphi';
    Cells[2,2] := 'is';
    Cells[4,2] := 'RAD';
  end;
end;
{
The following code prompts for confirmation before deleting
a file.  Create a file and then enter the path and file name
in the textfields.  Click the button to delete it.
}
procedure TForm1.Button1Click(Sender: TObject);
var FileName: string;
begin
  Filename:= Edit1.Text;
  if SysUtils.FileExists(FileName) then
  begin
    if MessageDlg(('Do you really want to delete ' + ExtractFileName(FileName) + '?'),
mtConfirmation, [mbYes, mbNo], 0, mbNo) = IDYes then
      DeleteFile(FileName);
  end
  else
    MessageDlg(('File ' + ExtractFileName(FileName) + ' does not exist.'), mtConfirmation,
[mbOK], 0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit2.Text:= ExtractFilePath(Application.ExeName);
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
// Read File to Grid
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  X, Y, I: Integer;
```

```
    StringLen, colcnt, rowcnt: Integer;
    cellString: String;
  begin
    if OpenDialog1.Execute then
    begin
      try
        iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
        iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
        FileSeek(iFileHandle,0,0);
  //      Buffer := PChar(System.AllocMem(iFileLength + 1));
  //      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
  //      FileClose(iFileHandle);
  //      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
  //      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
  //      FileSeek(iFileHandle,0,0);
        colcnt := StringGrid1.ColCount;
        iBytesRead := SysUtils.FileRead(
          iFileHandle,
          colcnt,
          4);  // 4 because it's an integer
        rowcnt := StringGrid1.RowCount;
        iBytesRead := SysUtils.FileRead(
          iFileHandle,
          rowcnt,
          4);  // 4 because it's an integer
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            iBytesRead := SysUtils.FileRead(iFileHandle, StringLen, 4);  // 4 because it's an
  integer
            Buffer := System.AllocMem(StringLen + 1);
  //          GetMem(Buffer, StringLen); { allocate the buffer }
            iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, StringLen);
  //          for I := 1 to StringLen do
  //            cellString[I] := Buffer[I - 1];
            StringGrid1.Cells[X,Y] := AnsiString(Buffer);
          finally
            FreeMem(Buffer);
          end;
        end;
      end;
      finally
        FileClose(iFileHandle);
      end;
    end;
  end;

  // Write Grid into File
  procedure TForm1.Button2Click(Sender: TObject);
  var
    BackupName: string;
    FileHandle: Integer;
    StringLen: Integer;
    X, Y, I: Integer;
    colCountLength, rowCountLength: Integer;
    Buffer: PAnsiChar;
    cellString: AnsiString;
  begin
    if SaveDialog1.Execute then
    begin
      if FileExists(SaveDialog1.FileName) then
      begin
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
```

```
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        PAnsiChar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PAnsiChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;
```

See Also

FileSearch (see page 4156)

DirectoryExists (see page 4100)

37.3.100 SysUtils.FileGetAttr Function

Returns the file attributes of FileName.

Pascal

```
function FileGetAttr(const FileName: string): Integer; platform;
```

C++

```
int FileGetAttr(const UnicodeString FileName);
```

File

SysUtils

Description

FileGetAttr returns the attributes of the file as a string of bits. This value is the same as the Attr field of a TSearchRec (see page 4330) type. Check for individual attributes with code such as the following:

Attrs := FileGetAttr('MyFile.sys');

```
if Attrs and faHidden <> 0 then
  FileSetAttr('MyFile.sys', Attrs – faHidden);
```

int Attrs = FileGetAttr("MyFile.sys");

```
if (Attrs & faHidden)
  FileSetAttr("MyFile.sys", Attrs & !faHidden);
```

A return value of INVALID_FILE_ATTRIBUTES indicates that an error occurred.

Note: See TSearchRec (see page 4330) for a description of the individual attribute constants.

Note: FileGetAttr is Windows-only.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
     if (NewAttributes != Attributes)
       FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
```

```
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then
        NewAttributes := NewAttributes or faHidden
      else
        NewAttributes := NewAttributes and not faHidden;
      if NewAttributes <> Attributes then { if anything changed... }
        FileSetAttr(FileDirName.Caption, NewAttributes);
         { ...write the new values }
    end;
  end;
end;
```

See Also

FileSetAttr (see page 4162)

37.3.101 SysUtils.FileGetDate Function

Returns an OS timestamp for a specified file.

Pascal

```
function FileGetDate(Handle: Integer): Integer;
```

C++

```
int FileGetDate(int Handle);
```

File

SysUtils

Description

Use FileGetDate to determine the timestamp of a file given its handle. The return value is -1 if the handle is invalid. The FileDateToDateTime (see page 4130) function can be used to convert the returned value to a TDateTime (see page 3323) value.

Note: :FileGetDate uses a file handle rather than a native Delphi file variable. To obtain the timestamp of a file that has not been opened using the FileOpen (see page 4143) or FileCreate (see page 4128) method, use the FileAge (see page 4119) function instead.

Note: :Under Linux, -1 is a valid timestamp. Use FileExists (see page 4133) to verify that the file does not exist.

See Also

FileSetDate (see page 4164)

FileDateToDateTime (see page 4130)

FileAge (see page 4119)

FileExists (see page 4133)

37.3.102 SysUtils.FileIsReadOnly Function

Reports whether a file is read-only.

Pascal

```
function FileIsReadOnly(const FileName: string): Boolean;
```

C++

```
Boolean FileIsReadOnly(const UnicodeString FileName);
```

File

SysUtils

Description

FileIsReadOnly returns **True** if FileName can be found and opened, but the current process does not have permission to write to it.

Note: On Windows XP, FileIsReadOnly returns **True** if the file does **not** exist.

37.3.103 SysUtils.FileOpen Function

Opens a specified file using a specified access mode.

Pascal

```
function FileOpen(const FileName: string; Mode: LongWord): Integer;
```

C++

```
int FileOpen(const UnicodeString FileName, LongWord Mode);
```

File

SysUtils

Description

Use FileOpen to open a file and obtain a file handle. The access mode value is constructed by or-ing one of the fmOpen constants with one of the fmShare constants defined in `File` open mode constants. If the return value is 0 or greater, the function was successful and the value is the file handle of the opened file. A return value of -1 indicates that an error occurred.

The Mode parameter consists of an open mode and (possibly) a share mode or'ed together. The open mode must be one of the following values:

Value	Meaning
fmOpenRead (see page 4377)	Open the file for reading only.
fmOpenWrite (see page 4378)	Open the file for writing only. Writing to the file completely replaces the current contents.
fmOpenReadWrite (see page 4378)	Open the file to modify the current contents rather than replace them.

The share mode must be one of the following values:

Value	Meaning
fmShareCompat (see page 4379)	Sharing is compatible with the way FCBs are opened.
fmShareExclusive (see page 4382)	Other applications can not open the file for any reason.
fmShareDenyWrite (see page 4381)	Other applications can open the file for reading but not for writing.
fmShareDenyRead (see page 4380)	Other applications can open the file for writing but not for reading.
fmShareDenyNone (see page 4380)	No attempt is made to prevent other applications from reading from or writing to the file.

Note: Use of the non-native Delphi language file handlers such as FileOpen is not encouraged. These routines map to system routines and return OS file handles, not normal Delphi file variables. These are low-level file access routines. For normal file operations use AssignFile (see page 3525), Rewrite (see page 3645), and Reset (see page 3643) instead.

C++ Examples:

```
/*
This example requires a TEdit, a button and a string grid.
Enter the file name in the TEdit and click the button to open
that file.  Notice that the file cannot be read.  Remove the
fmOpenWrite and the fmShareDenyNone access and the string
grid loads fine.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  try
  {
    iFileHandle = FileOpen(Edit1->Text, fmOpenRead | fmOpenWrite | fmShareDenyNone);
    if (iFileHandle > 0) {
      iFileLength = FileSeek(iFileHandle, 0, 2);
      FileSeek(iFileHandle,0,0);
      std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);

      for (int i = 0; i < iBytesRead; i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    else
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
  catch(...)
  {
    Application->MessageBox(
      L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
      L"File Error", IDOK);
  }
}
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first time you use this app, you should do a "Fill
Grid" and then "Write Grid to File".   Always use files
generated in this way to do a "Read File into Grid".
*/
// Read File to Grid

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  int iLength;
  if (OpenDialog1->Execute())
```

```
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
4);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
4);
      for (int x = 0; x < StringGrid1->ColCount; x++)
      {
        for (int y = 0; y < StringGrid1->RowCount; y++)
        {
          // Write out the length of each string, followed by the string itself.
          FileRead(iFileHandle, reinterpret_cast<char *>(&iLength), 4);
          std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iLength + 1]);
          FileRead(iFileHandle, pszBuffer.get(), iLength);
          pszBuffer.get()[iLength] = 0;
          StringGrid1->Cells[x][y] = pszBuffer.get();
        }
      }
      FileClose(iFileHandle);
    }
    catch(...)
    {
#if defined(_DELPHI_STRING_UNICODE)
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error",
        IDOK);
#else
      Application->MessageBox(
        "Can't perform one of the following file operations: Open, Seek, Read, Close.",
        "File Error",
        IDOK);
#endif
    }
  }
}

// Write Grid into File

#include <dir.h>

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
```

```
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (AnsiChar *)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
This example uses a file listbox and a regular listbox on a
form. The following routine scans through the files listed
in the file listbox and lists the sizes of any selected
files to the regular list box.  To exercise the error
condition create a file in the Debug directory, start this
application and then delete the file.  Now try to list the
```

```
size of the deleted file.  Set the MultiSelect and
ExtendedSelect properties on the FileListBox.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  F: File;
  i, filehandle: Integer;
begin
  for i := 0 to (FileListBox1.Items.Count - 1) do begin
  try
    if FileListBox1.Selected[i] then
    begin
      if not FileExists(FileListBox1.Items.Strings[i]) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' not found', mtError, [mbOk], 0);
        Continue;
      end;
      filehandle:=  FileOpen(FileListBox1.Items.Strings[i], fmOpenWrite);
      if (filehandle = -1) then
      begin
        MessageDlg('File: ' + FileListBox1.Items.Strings[i] +
                   ' cannot be opened with access mode fmOpenWrite.', mtError, [mbOk], 0);
        Continue;
      end
      else
        FileClose(filehandle);

      AssignFile(F, FileListBox1.Items.Strings[i]);
      Reset(F, 1);
      ListBox1.Items.Add(
        FileListBox1.Items.Strings[i] + ': ' + IntToStr(FileSize(F)));
      CloseFile(F);
    end;
   finally
   { do something here }
   end;
  end;
end;
{
This example requires a TEdit, a button and a string grid.
Enter the file name in the TEdit and click the button to open
that file.  Notice that the file cannot be read.  Remove the
fmOpenWrite and the fmShareDenyNone access and the string
grid loads fine.
}
procedure OpenForShare(const FileName: String);
var
  FileHandle : Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  FileHandle := SysUtils.FileOpen(FileName, fmOpenWrite or fmShareDenyNone or fmOpenRead);
  if FileHandle > 0 then
  begin
    try
      MessageDlg('Valid file handle.', mtInformation, [mbOk], 0);
      iFileLength := SysUtils.FileSeek(FileHandle,0,2);
      FileSeek(FileHandle, 0, 0);
      Buffer := PAnsiChar(System.AllocMem(iFileLength + 1));
      iBytesRead := SysUtils.FileRead(FileHandle, Buffer^, iFileLength);
      for i := 0 to iBytesRead-1 do
      begin
        Form1.StringGrid1.RowCount := Form1.StringGrid1.RowCount + 1;
```

```
        Form1.StringGrid1.Cells[1, i + 1] := Buffer[i];
        Form1.StringGrid1.Cells[2, i + 1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end
  else
    MessageDlg('Open error: FileHandle = negative DOS error code.', mtInformation, [mbOk], 0);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenForShare(Edit1.Text);
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
// Read File to Grid
procedure TForm1.Button1Click(Sender: TObject);
var
```

```
    iFileHandle: Integer;
    iFileLength: Integer;
    iBytesRead: Integer;
    Buffer: PAnsiChar;
    X, Y, I: Integer;
    StringLen, colcnt, rowcnt: Integer;
    cellString: String;
  begin
    if OpenDialog1.Execute then
    begin
      try
        iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
        iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
        FileSeek(iFileHandle,0,0);
  //      Buffer := PChar(System.AllocMem(iFileLength + 1));
  //      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
  //      FileClose(iFileHandle);
  //      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
  //      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
  //      FileSeek(iFileHandle,0,0);
        colcnt := StringGrid1.ColCount;
        iBytesRead := SysUtils.FileRead(
          iFileHandle,
          colcnt,
          4);  // 4 because it's an integer
        rowcnt := StringGrid1.RowCount;
        iBytesRead := SysUtils.FileRead(
          iFileHandle,
          rowcnt,
          4);  // 4 because it's an integer
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            iBytesRead := SysUtils.FileRead(iFileHandle, StringLen, 4);  // 4 because it's an
  integer
            Buffer := System.AllocMem(StringLen + 1);
  //          GetMem(Buffer, StringLen); { allocate the buffer }
            iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, StringLen);
  //          for I := 1 to StringLen do
  //            cellString[I] := Buffer[I - 1];
            StringGrid1.Cells[X,Y] := AnsiString(Buffer);
          finally
            FreeMem(Buffer);
          end;
        end;
      end;
      finally
        FileClose(iFileHandle);
      end;
    end;
  end;

  // Write Grid into File
  procedure TForm1.Button2Click(Sender: TObject);
  var
    BackupName: string;
    FileHandle: Integer;
    StringLen: Integer;
    X, Y, I: Integer;
    colCountLength, rowCountLength: Integer;
    Buffer: PAnsiChar;
    cellString: AnsiString;
  begin
    if SaveDialog1.Execute then
```

```
    begin
      if FileExists(SaveDialog1.FileName) then
      begin
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        PAnsiChar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PAnsiChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;
```

See Also

FileClose (see page 4121)

FileCreate (see page 4128)

AssignFile (see page 3525)

Rewrite (see page 3645)

Reset (see page 3643)

37.3.104 SysUtils.FileRead Function

Reads a specified number of bytes from a file.

Pascal

```
function FileRead(Handle: Integer; var Buffer; Count: LongWord): Integer;
```

C++

```
int FileRead(int Handle,  Buffer, LongWord Count);
```

File

SysUtils

Description

FileRead reads Count bytes from the file specified by Handle into the buffer. The Count parameter indicates the size, in bytes, of the buffer. The function result is the actual number of bytes read, which may be less than Count. The Handle that is passed to FileRead must be opened with FileOpen (see page 4143) or FileCreate (see page 4128).

Note: :In Delphi, do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To read from a file specified by a Delphi file variable, use the BlockRead (see page 3528) procedure or the Read (see page 3634) procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first time you use this app, you should do a "Fill
Grid" and then "Write Grid to File".   Always use files
generated in this way to do a "Read File into Grid".
*/
// Read File to Grid

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  int iLength;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
4);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
4);
      for (int x = 0; x < StringGrid1->ColCount; x++)
      {
        for (int y = 0; y < StringGrid1->RowCount; y++)
        {
          // Write out the length of each string, followed by the string itself.
          FileRead(iFileHandle, reinterpret_cast<char *>(&iLength), 4);
          std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iLength + 1]);
          FileRead(iFileHandle, pszBuffer.get(), iLength);
          pszBuffer.get()[iLength] = 0;
          StringGrid1->Cells[x][y] = pszBuffer.get();
        }
      }
      FileClose(iFileHandle);
    }
    catch(...)
    {
#if defined(_DELPHI_STRING_UNICODE)
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error",
```

```
        IDOK);
#else
      Application->MessageBox(
        "Can't perform one of the following file operations: Open, Seek, Read, Close.",
        "File Error",
        IDOK);
#endif
    }
  }
}

// Write Grid into File

#include <dir.h>

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (AnsiChar *)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
```

```
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
```

```
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
// Read File to Grid
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  X, Y, I: Integer;
  StringLen, colcnt, rowcnt: Integer;
  cellString: String;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//      Buffer := PChar(System.AllocMem(iFileLength + 1));
//      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
//      FileClose(iFileHandle);
//      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
//      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
//      FileSeek(iFileHandle,0,0);
      colcnt := StringGrid1.ColCount;
      iBytesRead := SysUtils.FileRead(
        iFileHandle,
        colcnt,
        4);  // 4 because it's an integer
      rowcnt := StringGrid1.RowCount;
      iBytesRead := SysUtils.FileRead(
        iFileHandle,
        rowcnt,
        4);  // 4 because it's an integer
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          iBytesRead := SysUtils.FileRead(iFileHandle, StringLen, 4);  // 4 because it's an
integer
          Buffer := System.AllocMem(StringLen + 1);
//          GetMem(Buffer, StringLen); { allocate the buffer }
          iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, StringLen);
//          for I := 1 to StringLen do
//            cellString[I] := Buffer[I - 1];
          StringGrid1.Cells[X,Y] := AnsiString(Buffer);
        finally
          FreeMem(Buffer);
        end;
      end;
    end;
    finally
      FileClose(iFileHandle);
    end;
  end;
```

```
end;

// Write Grid into File
procedure TForm1.Button2Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PAnsiChar;
  cellString: AnsiString;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      PAnsiChar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PAnsiChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;
```

See Also

BlockRead (see page 3528)

FileOpen (see page 4143)

FileCreate (see page 4128)

FileSeek (see page 4157)

FileWrite (see page 4166)

Read (see page 3634)

Write (see page 3707)

37.3.105 SysUtils.FileSearch Function

Searches a specified directory path for a file.

Pascal

```
function FileSearch(const Name: string; const DirList: string): string;
```

C++

```
UnicodeString FileSearch(const UnicodeString Name, const UnicodeString DirList);
```

File

SysUtils

Description

FileSearch searches through the directories passed in DirList for a file named Name. DirList is a list of path names delimited by semicolons on Windows, colons on Linux). If FileSearch locates a file matching Name, it returns a string specifying a path name for that file. If no matching file exists, FileSearch returns an empty string.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString asFileName = FileSearch(Edit1->Text, GetCurrentDir());
  if (asFileName.IsEmpty())
    ShowMessage(AnsiString("Couldn't find ") + Edit1->Text + ".");
  else
    ShowMessage(AnsiString("Found ") + asFileName + ".");
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Label2->Caption = GetCurrentDir();
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form. When the button is clicked, the current directory is
searched for the filename specified in the edit control. A
message box indicates whether the file is found.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FileToFind: string;
begin
  FileToFind := SysUtils.FileSearch(Edit1.Text, GetCurrentDir);
  if FileToFind = '' then
    ShowMessage('Couldn''t find ' + Edit1.Text + '.')
  else
    ShowMessage('Found ' + FileToFind + '.');
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  Label2.Caption := GetCurrentDir;
end;
```

See Also

FileExists (see page 4133)

37.3.106 SysUtils.FileSeek Function

Repositions read/write point.

Pascal

```
function FileSeek(Handle: Integer; Offset: Integer; Origin: Integer): Integer; overload;
function FileSeek(Handle: Integer; const Offset: Int64; Origin: Integer): Int64; overload;
```

C++

```
int FileSeek(int Handle, int Offset, int Origin);
Int64 FileSeek(int Handle, const Int64 Offset, int Origin);
```

File

SysUtils

Description

Use FileSeek to reposition the read/write point in a file that was opened with FileOpen (see page 4143) or FileCreate (see page 4128). Handle is the file handle that was returned by FileOpen (see page 4143) or FileCreate (see page 4128).

Offset specifies the number of bytes from Origin where the file pointer should be positioned. Origin is a code with three possible values, denoting the beginning of the file, the end of the file, and the current position of the file pointer.

Origin	Action
0	The file pointer is positioned Offset bytes from the beginning of the file.
1	The file pointer is positioned Offset bytes from its current position.
2	The file pointer is positioned Offset bytes from the end of the file.

If FileSeek is successful, it returns the new position of the file pointer; otherwise, it returns -1.

Note: :Do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To move the file pointer in a file specified by a Delphi file variable, use the Seek (see page 3650) procedure instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first time you use this app, you should do a "Fill
Grid" and then "Write Grid to File".   Always use files
generated in this way to do a "Read File into Grid".
*/
// Read File to Grid

#include <memory>       //for STL auto_ptr class
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  int iLength;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
4);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
4);
      for (int x = 0; x < StringGrid1->ColCount; x++)
      {
        for (int y = 0; y < StringGrid1->RowCount; y++)
        {
          // Write out the length of each string, followed by the string itself.
          FileRead(iFileHandle, reinterpret_cast<char *>(&iLength), 4);
          std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iLength + 1]);
          FileRead(iFileHandle, pszBuffer.get(), iLength);
          pszBuffer.get()[iLength] = 0;
          StringGrid1->Cells[x][y] = pszBuffer.get();
        }
      }
      FileClose(iFileHandle);
    }
    catch(...)
    {
#if defined(_DELPHI_STRING_UNICODE)
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error",
        IDOK);
#else
      Application->MessageBox(
        "Can't perform one of the following file operations: Open, Seek, Read, Close.",
        "File Error",
        IDOK);
#endif
    }
  }
}

// Write Grid into File

#include <dir.h>

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
```

```
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (AnsiChar *)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  char *pszBuffer;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//    pszBuffer = PChar(AllocMem(iFileLength+1));
      std::auto_ptr<char> pszBuffer(new char[iFileLength+1]);
      iBytesRead = FileRead(iFileHandle, pszBuffer.get(), iFileLength);
      FileClose(iFileHandle);
      for (int i=0;i<iBytesRead;i++)
      {
        StringGrid1->RowCount += 1;
        StringGrid1->Cells[1][i+1] = pszBuffer.get()[i];
        StringGrid1->Cells[2][i+1] = IntToStr((int)pszBuffer.get()[i]);
      }
    }
    catch(...)
    {
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error", IDOK);
    }
  }
}
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  i: Integer;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      Buffer := System.AllocMem(iFileLength + 1);
      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
      FileClose(iFileHandle);
      for i := 0 to iBytesRead-1 do
      begin
        StringGrid1.RowCount := StringGrid1.RowCount + 1;
        StringGrid1.Cells[1,i+1] := Buffer[i];
        StringGrid1.Cells[2,i+1] := IntToStr(Integer(Buffer[i]));
      end;
    finally
      FreeMem(Buffer);
    end;
  end;
end;
{
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first column contains the character values in the
buffer. The second column contains the numeric values of the
characters in the buffer.
}
// Read File to Grid
procedure TForm1.Button1Click(Sender: TObject);
var
  iFileHandle: Integer;
  iFileLength: Integer;
  iBytesRead: Integer;
  Buffer: PAnsiChar;
  X, Y, I: Integer;
  StringLen, colcnt, rowcnt: Integer;
  cellString: String;
begin
  if OpenDialog1.Execute then
  begin
    try
      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
//      Buffer := PChar(System.AllocMem(iFileLength + 1));
```

```
//      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
//      FileClose(iFileHandle);
//      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
//      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
//      FileSeek(iFileHandle,0,0);
      colcnt := StringGrid1.ColCount;
      iBytesRead := SysUtils.FileRead(
        iFileHandle,
        colcnt,
        4);  // 4 because it's an integer
      rowcnt := StringGrid1.RowCount;
      iBytesRead := SysUtils.FileRead(
        iFileHandle,
        rowcnt,
        4);  // 4 because it's an integer
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          iBytesRead := SysUtils.FileRead(iFileHandle, StringLen, 4);  // 4 because it's an
integer
          Buffer := System.AllocMem(StringLen + 1);
//          GetMem(Buffer, StringLen); { allocate the buffer }
          iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, StringLen);
//          for I := 1 to StringLen do
//            cellString[I] := Buffer[I - 1];
          StringGrid1.Cells[X,Y] := AnsiString(Buffer);
        finally
          FreeMem(Buffer);
        end;
      end;
    end;
    finally
      FileClose(iFileHandle);
    end;
  end;
end;

// Write Grid into File
procedure TForm1.Button2Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PAnsiChar;
  cellString: AnsiString;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      PAnsiChar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
```

```
        PAnsiChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;
```

See Also

FileOpen (see page 4143)

FileCreate (see page 4128)

FileRead (see page 4150)

FileWrite (see page 4166)

Seek (see page 3650)

37.3.107 SysUtils.FileSetAttr Function

Sets the file attributes of a specified file.

Pascal

```
function FileSetAttr(const FileName: string; Attr: Integer): Integer; platform;
```

C++

```
int FileSetAttr(const UnicodeString FileName, int Attr);
```

File

SysUtils

Description

FileSetAttr sets the file attributes of the file given by FileName to the value given by Attr. The value of Attr is formed by combining the appropriate file attribute constants, as in the following:

```
FileSetAttr('MyFile.sys', faReadOnly or faSysFile); // Delphi
FileSetAttr("MyFile.sys", faReadOnly | faSysFile); // C++
```

FileSetAttr returns zero if the function was successful. Otherwise the return value is an error code.

Note: See TSearchRec (see page 4330) for a description of the file attribute constants.

Note: FileSetAttr is only available on Windows.

C++ Examples:

```
/*
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
*/
#include "fattrdlg.h"

void __fastcall TForm1::FileListMouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

  unsigned short  Attributes;
  unsigned short  NewAttributes;

  FileAttrDlg->FileDirName->Caption = FileList->Items->Strings[FileList->ItemIndex];
  FileAttrDlg->FilePathName->Caption = FileList->Directory;
  FileAttrDlg->ChangeDate->Caption =
DateTimeToStr(FileDateToDateTime(FileAge(FileList->FileName)));
  Attributes = FileGetAttr(FileList->Items->Strings[FileList->ItemIndex]);
  FileAttrDlg->ReadOnly->Checked = Attributes & faReadOnly;
  FileAttrDlg->Archive->Checked = Attributes & faArchive;
  FileAttrDlg->System->Checked = Attributes & faSysFile;
  FileAttrDlg->Hidden->Checked = Attributes & faHidden;
  if (FileAttrDlg->ShowModal()!= mrCancel){
     NewAttributes = Attributes;
     if (FileAttrDlg->ReadOnly->Checked)
       NewAttributes = NewAttributes | faReadOnly;
     else
        NewAttributes = NewAttributes & ~faReadOnly;

     if (FileAttrDlg->Archive->Checked)
       NewAttributes = NewAttributes | faArchive;
     else
       NewAttributes = NewAttributes & ~faArchive;

     if (FileAttrDlg->System->Checked)
       NewAttributes = NewAttributes | faSysFile;
     else
       NewAttributes = NewAttributes & ~faSysFile;

     if (FileAttrDlg->Hidden->Checked)
       NewAttributes = NewAttributes | faHidden;
     else
       NewAttributes = NewAttributes  & ~faHidden;
     if (NewAttributes != Attributes)
       FileSetAttr(FileAttrDlg->FileDirName->Caption, NewAttributes);
  }
}
```

Delphi Examples:

```
{
The following code reads a file's attributes into a set
variable, sets the check boxes in a file-attribute dialog
box to represent the current attributes, then executes the
dialog box. If the user changes and accepts any dialog box
settings, the code sets the file attributes to match the
changed settings.  Click on a file to launch the dialog.
```

```
}
procedure TForm1.FileListMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  Attributes, NewAttributes: Word;
begin
  with FileAttrForm do
  begin
    FileDirName.Caption := FileList.Items[FileList.ItemIndex];
    { set box caption }
    FilePathName.Caption := FileList.Directory;
    { show directory name }
    ChangeDate.Caption := DateTimeToStr(FileDateToDateTime(FileAge(FileList.FileName)));
    Attributes := SysUtils.FileGetAttr(FileDirName.Caption);
    { read file attributes }
    ReadOnly.Checked := (Attributes and SysUtils.faReadOnly) = faReadOnly;
    Archive.Checked := (Attributes and faArchive) = faArchive;
    System.Checked := (Attributes and faSysFile) = faSysFile;
    Hidden.Checked := (Attributes and faHidden) = faHidden;
    if ShowModal <> id_Cancel then    { execute dialog box }
    begin
      NewAttributes := Attributes;
      { start with original attributes }
      if ReadOnly.Checked then
        NewAttributes := NewAttributes or SysUtils.faReadOnly
      else
        NewAttributes := NewAttributes and not SysUtils.faReadOnly;
      if Archive.Checked then
        NewAttributes := NewAttributes or faArchive
      else
        NewAttributes := NewAttributes and not faArchive;
      if System.Checked then
        NewAttributes := NewAttributes or faSysFile
      else
        NewAttributes := NewAttributes and not faSysFile;
      if Hidden.Checked then
        NewAttributes := NewAttributes or faHidden
      else
        NewAttributes := NewAttributes and not faHidden;
      if NewAttributes <> Attributes then { if anything changed... }
        FileSetAttr(FileDirName.Caption, NewAttributes);
         { ...write the new values }
    end;
  end;
end;
```

See Also

FileGetAttr (see page 4139)

TSearchRec (see page 4330)

37.3.108 SysUtils.FileSetDate Function

Sets the OS time stamp for a specified file.

Pascal

```
function FileSetDate(const FileName: string; Age: Integer): Integer; overload;
function FileSetDate(Handle: Integer; Age: Integer): Integer; overload; platform;
```

C++

```
int FileSetDate(const UnicodeString FileName, int Age);
```

```
int FileSetDate(int Handle, int Age);
```

File

SysUtils

Description

FileSetDate sets the OS timestamp of the specified file to the value given by Age. The DateTimeToFileDate (see page 4088) function can be used to convert a TDateTime (see page 3323) value to a OS timestamp.

Handle is the Windows file handle of the file to alter. This syntax is only available on Windows.

FileName is the name of the file to alter.

Age is the timestamp to apply to the specified file.

The return value is zero if the function was successful. Otherwise the return value is an error code.

See Also

FileCreate (see page 4128)

FileGetDate (see page 4142)

FileOpen (see page 4143)

DateTimeToFileDate (see page 4088)

37.3.109 SysUtils.FileSetReadOnly Function

Sets the file permissions to read-only.

Pascal

```
function FileSetReadOnly(const FileName: string; ReadOnly: Boolean): Boolean;
```

C++

```
Boolean FileSetReadOnly(const UnicodeString FileName, Boolean ReadOnly);
```

File

SysUtils

Description

FileSetReadOnly attempts to set the read-only status of the specified file.

FileName is the name of the file to alter.

ReadOnly indicates the status to assign to the file.

FileSetReadOnly returns true if the operation was successful and false if an error occurred.

Note: On Windows, the ReadOnly parameter is assigned to the file's read-only flag.

Note: On Linux, the inverse of the ReadOnly parameter is assigned to the file's user write, group write, and other write permission bits. The active user must have permission to perform these operations.

FileSetReadOnly is designed to support cross-platform code. To manipulate permissions more specifically than FileSetReadOnly allows, use platform-specific functions.

37.3.110 SysUtils.FileWrite Function

Writes the contents of a buffer to the current position in a file.

Pascal

```
function FileWrite(Handle: Integer; const Buffer; Count: LongWord): Integer;
```

C++

```
int FileWrite(int Handle, const  Buffer, LongWord Count);
```

File

SysUtils

Description

FileWrite writes Count bytes to the file given by Handle from the buffer specified by Buffer. Handle is a file handle returned by the FileOpen (see page 4143) or FileCreate (see page 4128) method.

The return value is the number of bytes actually written, or -1 if an error occurred.

Note: :Do not mix routines that take or return file handles with those that use Delphi language file variables (typically seen as var F). To write to a file specified by a Delphi file variable, use Write (see page 3707), Writeln, or BlockWrite (see page 3530) instead.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function. Verify
the file written after executing.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
```

```
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
/*
The following example uses a button, a string grid, and an
Open dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the specified file is opened, read into a buffer, and closed.
Then the buffer is displayed in two columns of the string
grid. The first time you use this app, you should do a "Fill
Grid" and then "Write Grid to File".   Always use files
generated in this way to do a "Read File into Grid".
*/
// Read File to Grid

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int iFileHandle;
  int iFileLength;
  int iBytesRead;
  int iLength;
  if (OpenDialog1->Execute())
  {
    try
    {
      iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
      iFileLength = FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
4);
      iBytesRead = FileRead(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
4);
      for (int x = 0; x < StringGrid1->ColCount; x++)
      {
        for (int y = 0; y < StringGrid1->RowCount; y++)
        {
          // Write out the length of each string, followed by the string itself.
```

```
          FileRead(iFileHandle, reinterpret_cast<char *>(&iLength), 4);
          std::auto_ptr<AnsiChar> pszBuffer(new AnsiChar[iLength + 1]);
          FileRead(iFileHandle, pszBuffer.get(), iLength);
          pszBuffer.get()[iLength] = 0;
          StringGrid1->Cells[x][y] = pszBuffer.get();
        }
      }
      FileClose(iFileHandle);
    }
    catch(...)
    {
#if defined(_DELPHI_STRING_UNICODE)
      Application->MessageBox(
        L"Can't perform one of the following file operations: Open, Seek, Read, Close.",
        L"File Error",
        IDOK);
#else
      Application->MessageBox(
        "Can't perform one of the following file operations: Open, Seek, Read, Close.",
        "File Error",
        IDOK);
#endif
    }
  }
}

// Write Grid into File

#include <dir.h>

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, (AnsiChar *)&iLength, sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
```

Delphi Examples:

```
{
```

```
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
  cellString: string;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      Pchar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  with StringGrid1 do
  begin
    Cells[1,0] := 'Column 1';
    Cells[2,0] := 'Column 2';
    Cells[3,0] := 'Column 3';
    Cells[4,0] := 'Column 4';
    Cells[0,1] := 'Row 1';
    Cells[1,1] := 'Object';
```

```
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
  {
  The following example uses a button, a string grid, and an
  Open dialog box on a form. When the button is clicked, the
  user is prompted for a filename. When the user clicks OK,
  the specified file is opened, read into a buffer, and closed.
  Then the buffer is displayed in two columns of the string
  grid. The first column contains the character values in the
  buffer. The second column contains the numeric values of the
  characters in the buffer.
  }
  // Read File to Grid
  procedure TForm1.Button1Click(Sender: TObject);
  var
    iFileHandle: Integer;
    iFileLength: Integer;
    iBytesRead: Integer;
    Buffer: PAnsiChar;
    X, Y, I: Integer;
    StringLen, colcnt, rowcnt: Integer;
    cellString: String;
  begin
    if OpenDialog1.Execute then
    begin
      try
        iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
        iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
        FileSeek(iFileHandle,0,0);
  //      Buffer := PChar(System.AllocMem(iFileLength + 1));
  //      iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, iFileLength);
  //      FileClose(iFileHandle);
  //      iFileHandle := SysUtils.FileOpen(OpenDialog1.FileName, fmOpenRead);
  //      iFileLength := SysUtils.FileSeek(iFileHandle,0,2);
  //      FileSeek(iFileHandle,0,0);
        colcnt := StringGrid1.ColCount;
        iBytesRead := SysUtils.FileRead(
          iFileHandle,
          colcnt,
          4);  // 4 because it's an integer
        rowcnt := StringGrid1.RowCount;
        iBytesRead := SysUtils.FileRead(
          iFileHandle,
          rowcnt,
          4);  // 4 because it's an integer
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            iBytesRead := SysUtils.FileRead(iFileHandle, StringLen, 4);  // 4 because it's an
  integer
            Buffer := System.AllocMem(StringLen + 1);
  //          GetMem(Buffer, StringLen); { allocate the buffer }
            iBytesRead := SysUtils.FileRead(iFileHandle, Buffer^, StringLen);
  //          for I := 1 to StringLen do
  //            cellString[I] := Buffer[I - 1];
            StringGrid1.Cells[X,Y] := AnsiString(Buffer);
          finally
```

```
          FreeMem(Buffer);
        end;
      end;
    end;
    finally
      FileClose(iFileHandle);
    end;
  end;
end;

// Write Grid into File
procedure TForm1.Button2Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PAnsiChar;
  cellString: AnsiString;
begin
  if SaveDialog1.Execute then
  begin
    if FileExists(SaveDialog1.FileName) then
    begin
      BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
      BackupName := ChangeFileExt(BackupName, '.BAK');
      if not RenameFile(SaveDialog1.FileName, BackupName) then
        raise Exception.Create('Unable to create backup file.');
    end;
    FileHandle := FileCreate(SaveDialog1.FileName);
    { Write out the number of rows and columns in the grid. }
    colCountLength := SizeOf(StringGrid1.ColCount);
    FileWrite(FileHandle,
      PAnsiChar(StringGrid1.ColCount), colCountLength);
    rowCountLength := SizeOf(StringGrid1.RowCount);
    FileWrite(FileHandle,
      PAnsiChar(StringGrid1.RowCount), rowCountLength);
    for X := 0 to StringGrid1.ColCount - 1 do
    begin
      for Y := 0 to StringGrid1.RowCount - 1 do
      begin
        try
          { Write out the length of each string, followed by the string itself. }
          StringLen := Length(StringGrid1.Cells[X,Y]);
          FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
          GetMem(Buffer, StringLen); { allocate the buffer }
          cellString := StringGrid1.Cells[X,Y];
          for I := 1 to StringLen do
             Buffer[I - 1] := cellString[I];
          FileWrite(FileHandle, Buffer^, StringLen);
        finally
          FreeMem(Buffer, StringLen);
        end;
      end;
    end;
    FileClose(FileHandle);
  end;
end;
```

See Also

FileRead (see page 4150)

FileSeek (see page 4157)

WriteLn (see page 3710)

Write (see page 3707)

BlockWrite (see page 3530)

37.3.111 SysUtils.FinalizePackage Function

Finalizes the specified package (.bpl file).

Pascal

```
procedure FinalizePackage(Module: HMODULE);
```

C++

```
FinalizePackage(HMODULE Module);
```

File

SysUtils

Description

Most applications do not need to call FinalizePackage, because UnloadPackage (see page 4310) calls FinalizePackage internally. FinalizePackage executes any finalization code in the specified package.

See Also

UnloadPackage (see page 4310)

37.3.112 SysUtils.FindClose Function

Releases memory allocated by FindFirst (see page 4175).

Pascal

```
procedure FindClose(var F: TSearchRec);
```

C++

```
FindClose(TSearchRec F);
```

File

SysUtils

Description

FindClose terminates a FindFirst (see page 4175)/FindNext (see page 4177) sequence.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
```

```
      RowCount := 0;
      if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
      begin
        repeat
          if (sr.Attr and FileAttrs) = sr.Attr then
          begin
          RowCount := RowCount + 1;
          Cells[1,RowCount-1] := sr.Name;
          Cells[2,RowCount-1] := IntToStr(sr.Size);
          end;
        until FindNext(sr) <> 0;
        FindClose(sr);
      end;
    end;
  end;
```

See Also

FindFirst (see page 4175)

FindNext (see page 4177)

37.3.113 SysUtils.FindCmdLineSwitch Function

Determines whether a string was passed as a command line argument to the application.

Pascal

```
function FindCmdLineSwitch(const Switch: string; const Chars: TSysCharSet; IgnoreCase:
Boolean): Boolean; overload;
function FindCmdLineSwitch(const Switch: string): Boolean; overload;
function FindCmdLineSwitch(const Switch: string; IgnoreCase: Boolean): Boolean; overload;
```

C++

```
Boolean FindCmdLineSwitch(const UnicodeString Switch, const TSysCharSet Chars, Boolean
IgnoreCase);
Boolean FindCmdLineSwitch(const UnicodeString Switch);
Boolean FindCmdLineSwitch(const UnicodeString Switch, Boolean IgnoreCase);
```

File

SysUtils

Description

FindCmdLineSwitch determines whether a string represents a command-line switch, based on the first character of the string.

Switch is a command line parameter, typically provided by the ParamStr (see page 3625) function.

Chars is the set of characters that distinguish switches from other parameters. If SwitchChars (see page 4398) is omitted, it defaults to '/' and '-' on Windows, and to '-' on Linux.

IgnoreCase determines whether case-insensitive character matching is used. The default is true on Windows, false on Linux.

See Also

ParamCount (see page 3623)

ParamStr (see page 3625)

37.3.114 SysUtils.FindFirst Function

Searches for the first instance of a file name with a given set of attributes in a specified directory.

Pascal

```
function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;
```

C++

```
int FindFirst(const UnicodeString Path, int Attr, TSearchRec F);
```

File

SysUtils

Description

FindFirst searches the directory specified by Path for the first file that matches the file name implied by Path and the attributes specified by the Attr parameter. The result is returned in the F parameter. Use the fields of this search record to extract the information needed. FindFirst returns 0 if a file was successfully located, otherwise, it returns an error code.

The Path constant parameter is the directory and file name mask, including wildcard characters. For example, '.\test*.*' specifies all files in the test subdirectory.

The Attr parameter specifies the special files to include in addition to all normal files. Choose from these file attribute constants when specifying the Attr parameter:

Constant	Description
faReadOnly (see page 4374)	Read (see page 3634)-only files
faHidden (see page 4373)	Hidden files
faSysFile (see page 4375)	System (see page 3159) files
faVolumeID (see page 4376)	Volume ID files
faDirectory (see page 4372)	Directory files
faArchive (see page 4372)	Archive files
faAnyFile (see page 4371)	Any file

Attributes can be combined by adding (Delphi) or or-ing (C++) their constants or values. For example, to search for read-only and hidden files in addition to normal files, pass (faReadOnly (see page 4374) + faHidden (see page 4373)) in Delphi or (faReadOnly (see page 4374) | faHidden (see page 4373)) in C++ as the Attr parameter. To include only normal files, pass zero for the Attr parameter.

Note: FindFirst allocates resources (memory) which must be released by calling FindClose (see page 4172).

Note: Some of the file attribute constants are not valid on all platforms. For example, faVolumeID (see page 4376) and faArchive (see page 4372) will not work on Linux.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
```

```
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
```

```
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
      repeat
        if (sr.Attr and FileAttrs) = sr.Attr then
        begin
        RowCount := RowCount + 1;
        Cells[1,RowCount-1] := sr.Name;
        Cells[2,RowCount-1] := IntToStr(sr.Size);
        end;
      until FindNext(sr) <> 0;
      FindClose(sr);
    end;
  end;
end;
```

See Also

FindClose (see page 4172)

FindNext (see page 4177)

37.3.115 SysUtils.FindNext Function

Returns the next entry matching the name and attributes specified in a previous call to FindFirst (see page 4175).

Pascal

```
function FindNext(var F: TSearchRec): Integer;
```

C++

```
int FindNext(TSearchRec F);
```

File

SysUtils

Description

FindNext returns the next entry that matches the name and attributes specified in a previous call to FindFirst (see page 4175). The search record must be one that was passed to FindFirst (see page 4175). The return value is zero if the function was successful. Otherwise the return value is an error code.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
  begin
    RowCount := 0;
    if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
    begin
```

```
        repeat
          if (sr.Attr and FileAttrs) = sr.Attr then
          begin
          RowCount := RowCount + 1;
          Cells[1,RowCount-1] := sr.Name;
          Cells[2,RowCount-1] := IntToStr(sr.Size);
          end;
        until FindNext(sr) <> 0;
        FindClose(sr);
      end;
    end;
  end;
```

See Also

FindFirst (see page 4175)

37.3.116 SysUtils.FloatToCurr Function

Converts a floating-point value to a System::Currency (see page 3235) value.

Pascal

```
function FloatToCurr(const Value: Extended): Currency;
```

C++

```
Currency FloatToCurr(const Extended Value);
```

File

SysUtils

Description

FloatToCurr returns the System::Currency (see page 3235) value equivalent to Value. If Value is not within the valid range for a System::Currency (see page 3235) value, FloatToCurr raises an EConvertError (see page 3919) exception.

See Also

TryFloatToCurr (see page 4304)

37.3.117 SysUtils.FloatToDateTime Function

Converts a floating-point value to a TDateTime (see page 3323) value.

Pascal

```
function FloatToDateTime(const Value: Extended): TDateTime;
```

C++

```
TDateTime FloatToDateTime(const Extended Value);
```

File

SysUtils

Description

FloatToDateTime returns the TDateTime (see page 3323) value equivalent to Value. If Value is not within the valid range for TDateTime (see page 3323), FloatToDateTime raises an EConvertError (see page 3919) exception.

See Also

TryFloatToDateTime (see page 4304)

MinDateTime (see page 4384)

EConvertError (see page 3919)

37.3.118 SysUtils.FloatToDecimal Function

Converts a floating-point value to a decimal representation.

Pascal

```
procedure FloatToDecimal(var Result: TFloatRec; const Value; ValueType: TFloatValue;
Precision: Integer; Decimals: Integer);
```

C++

```
FloatToDecimal(TFloatRec Result, const  Value, TFloatValue ValueType, int Precision, int
Decimals);
```

File

SysUtils

Description

FloatToDecimal converts a floating-point value to a decimal representation that is suited for further formatting.

The Value parameter must be a variable of type Extended or System::Currency (see page 3235), as indicated by the ValueType parameter.

For values of type Extended, the Precision parameter specifies the requested number of significant digits in the result--the allowed range is 1..18. For values of type System::Currency (see page 3235), the Precision parameter is ignored, and the implied precision of the conversion is 19 digits.

The Decimals parameter specifies the requested maximum number of digits to the left of the decimal point in the result. Precision and Decimals together control how the result is rounded. To produce a result that always has a given number of significant digits regardless of the magnitude of the number, specify 9999 for the Decimals parameter.

37.3.119 SysUtils.FloatToStr Function

Converts a floating point value to a string.

Pascal

```
function FloatToStr(Value: Extended): string; overload;
function FloatToStr(Value: Extended; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString FloatToStr(Extended Value);
UnicodeString FloatToStr(Extended Value, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToStr converts the floating-point value given by Value to its string representation. The conversion uses general number format with 15 significant digits.

For greater control over the formatting of the string, use the FloatToStrF (see page 4182) function.

The first form of FloatToStr is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

C++ Examples:

```
/*
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Extended ex = 16 * ArcTan(1.0/5.0) - 4.0 * ArcTan(1.0/239.0); // Machin's formula
  String mystring = FloatToStr(ex);
  Edit2->Text = mystring;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Text = "3.14159265358979";
}
```

Delphi Examples:

```
{
This example displays the compiler version number in the
text field.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Text := FloatToStr(System.CompilerVersion);
end;
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit3.Text := mystring;
  Str(ex:25:23, mystring);
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
```

```
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

See Also

FloatToStrF (see page 4182)

FloatToText (see page 4183)

GetLocaleFormatSettings (see page 4206)

37.3.120 SysUtils.FloatToStrF Function

Converts a floating point value to a string, using a specified Format (see page 4188), Precision, and Digits.

Pascal

```
function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision: Integer; Digits:
Integer): string; overload;
function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision: Integer; Digits:
Integer; const FormatSettings: TFormatSettings): string; overload;
```

C++

```
UnicodeString FloatToStrF(Extended Value, TFloatFormat Format, int Precision, int Digits);
UnicodeString FloatToStrF(Extended Value, TFloatFormat Format, int Precision, int Digits,
const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToStrF converts the floating-point value given by Value to its string representation.

The Value parameter is the value to convert.

The Precision parameter specifies the precision of the given value. It should be 7 or less for values of type Single, 15 or less for values of type Double, and 18 or less for values of type Extended.

The Digits and Format (see page 4188) parameters together control how the value is formatted into a string. For details, see the description of TFloatFormat (see page 4323).

If the given value is a NAN (not-a-number), the resulting string is 'NAN'. If the given value is positive infinity, the resulting string is 'INF'. If the given value is negative infinity, the resulting string is '-INF'.

The first form of FloatToStrF is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToStrF, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToStrF, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

Delphi Examples:

```
{
This example shows how to use the ArcTan and FloatToStr
functions to approximate Pi.
```

```
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ex : Extended;
  mystring : String;
begin
  ex := 16 * ArcTan(1/5) - 4 * ArcTan(1/239); // Machin's formula
  mystring := FloatToStr(ex);
  Edit3.Text := mystring;
  Str(ex:25:23, mystring);
  Edit4.Text := mystring;
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit2.Text := mystring;
  mystring := SysUtils.FormatFloat('#,##0.00000000000000000000000000;;Zero', ex);
  Edit5.Text := mystring;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  ex : Extended;
  myString : String;
begin
  ex := Pi;
//  mystring := FloatToStr(ex);
//  Str(ex:25:23, mystring);
  mystring := FloatToStrF(ex, ffFixed, 35, 33);
  Edit1.Text := mystring;
end;
```

37.3.121 SysUtils.FloatToText Function

Converts a floating-point value to an unterminated character string, using a specified Format (see page 4188), Precision and Digits.

Pascal

```
function FloatToText(BufferArg: PAnsiChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer): Integer; overload;
function FloatToText(BufferArg: PWideChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer): Integer; overload;
function FloatToText(BufferArg: PAnsiChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer; const FormatSettings: TFormatSettings):
Integer; overload;
function FloatToText(BufferArg: PWideChar; const Value; ValueType: TFloatValue; Format:
TFloatFormat; Precision: Integer; Digits: Integer; const FormatSettings: TFormatSettings):
Integer; overload;
```

C++

```
int FloatToText(PAnsiChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits);
int FloatToText(PWideChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits);
int FloatToText(PAnsiChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits, const TFormatSettings FormatSettings);
int FloatToText(PWideChar BufferArg, const  Value, TFloatValue ValueType, TFloatFormat Format,
int Precision, int Digits, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

FloatToText converts the given floating-point value to its decimal representation using the specified format, precision, and digits.

The Value parameter must be a variable of type Extended or System::Currency (see page 3235), as indicated by the ValueType parameter. The resulting string of characters is stored in the given buffer, and the returned value is the number of characters stored. The resulting string is not null-terminated.

The first form of FloatToText is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToText, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToText, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

FloatToStrF (see page 4182)

GetLocaleFormatSettings (see page 4206)

37.3.122 SysUtils.FloatToTextFmt Function

Converts a floating-point value to an unterminated character string, using a specified format.

Pascal

```
function FloatToTextFmt(Buf: PAnsiChar; const Value; ValueType: TFloatValue; Format:
PAnsiChar): Integer; overload;
function FloatToTextFmt(Buf: PAnsiChar; const Value; ValueType: TFloatValue; Format:
PAnsiChar; const FormatSettings: TFormatSettings): Integer; overload;
function FloatToTextFmt(Buf: PWideChar; const Value; ValueType: TFloatValue; Format:
PWideChar): Integer; overload;
function FloatToTextFmt(Buf: PWideChar; const Value; ValueType: TFloatValue; Format:
PWideChar; const FormatSettings: TFormatSettings): Integer; overload;
```

C++

```
int FloatToTextFmt(PAnsiChar Buf, const  Value, TFloatValue ValueType, PAnsiChar Format);
int FloatToTextFmt(PAnsiChar Buf, const  Value, TFloatValue ValueType, PAnsiChar Format, const
TFormatSettings FormatSettings);
int FloatToTextFmt(PWideChar Buf, const  Value, TFloatValue ValueType, PWideChar Format);
int FloatToTextFmt(PWideChar Buf, const  Value, TFloatValue ValueType, PWideChar Format, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use FloatToTextFmt to format a floating-point value using a format string rather than using a predefined format. To use a predefined format, use the FloatToText (see page 4183) or FloatToStrF (see page 4182) function instead.

The Value parameter must be a variable of type Extended or System::Currency (see page 3235), as indicated by the ValueType parameter. The resulting string of characters is stored in the given buffer, and the returned value is the number of characters stored. The resulting string is not null-terminated.

Formats supported are described as part of the FormatFloat (see page 4197) function.

The first form of FloatToTextFmt is not thread-safe, because it uses localization information contained in global variables. The second form of FloatToTextFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FloatToTextFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

FormatFloat (see page 4197)

FloatToStrF (see page 4182)

FloatToText (see page 4183)

GetLocaleFormatSettings (see page 4206)

37.3.123 SysUtils.FmtLoadStr Function

Returns formatted output using a resourced format string.

Pascal

```
function FmtLoadStr(Ident: Integer; const Args: array of const): string;
```

C++

```
UnicodeString FmtLoadStr(int Ident, const array of const Args);
```

File

SysUtils

Description

FmtLoadStr loads a string from a program's resource string table and uses that string, plus an array of arguments, as a parameter to Format (see page 4188).

Ident is the string resource ID of the desired format string.

Args is an array of arguments that replace the format specifiers in the string Ident designates.

Note: Args_Size is the index of the last element in the Args array (one less than the number of arguments).

FmtLoadStr returns the resourced string with the arguments applied. This is the same value that would result from calling Format (see page 4188), using the resourced string and the specified arguments.

See Also

Format (see page 4188)

37.3.124 SysUtils.FmtStr Function

Assembles a formatted string using a format string and an array of arguments.

Pascal

```
procedure FmtStr(var Result: string; const Format: string; const Args: array of const);
overload;
procedure FmtStr(var Result: string; const Format: string; const Args: array of const; const
FormatSettings: TFormatSettings); overload;
```

C++

```
FmtStr(UnicodeString Result, const UnicodeString Format, const array of const Args);
FmtStr(UnicodeString Result, const UnicodeString Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the specified open (untyped) array.

StrResult returns the results of the operation.

Format (see page 4188) is the format string that determines how to generate StrResult. For information on the format strings, see Format (see page 4188) Strings.

Args is an array of arguments that replace the format specifiers in Format (see page 4188).

Note: Args_Size specifies the index of the last element of Args (one less than the number of elements).

The first form of FmtStr is not thread-safe, because it uses localization information contained in global variables. The second form of FmtStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FmtStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

FormatBuf (see page 4193)

StrFmt (see page 4250)

StrLFmt (see page 4261)

Format (see page 4188)

FmtLoadStr (see page 4185)

GetLocaleFormatSettings (see page 4206)

37.3.125 SysUtils.ForceDirectories Function

Creates a new directory, including the creation of parent directories as needed.

Pascal

```
function ForceDirectories(Dir: string): Boolean;
```

C++

```
Boolean ForceDirectories(UnicodeString Dir);
```

File

SysUtils

Description

ForceDirectories creates a new directory as specified in Dir, which must be a fully-qualified path name. If the directories given in the path do not yet exist, ForceDirectories attempts to create them.

ForceDirectories returns true if it successfully creates all necessary directories, false if it could not create a needed directory.

Warning: Important:Do not call ForceDirectories with an empty string. Doing so causes ForceDirectories to raise an exception.

Note: The FileCtrl unit (Windows only) also contains a ForceDirectories function. However, the FileCtrl version is deprecated, and the SysUtils (see page 3901) version preferred, even if the code does not need to be cross-platform.

C++ Examples:

```
/*
The following sample code sets the directory of
DirectoryListBox1 to c:\\WINDOWS when the form is created.
When Button1 is pressed, a subdirectory called
c:\\WINDOWS\\mytemp is added, but note that it is not updated
in DirectoryListBox1 until Button2 is pressed and Update is
called.
Note: A EInOutError will occur if you add a directory that
already exists.  Also, you must add the new directory to
your current directory to see the magic happen.  Also,
capitalization matters in C++ and the file spearators must
be "\\".
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ForceDirectories(Edit1->Text);
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  DirectoryListBox1->Update();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  DirectoryListBox1->Directory = "c:\\WINDOWS";
}
/*
This example uses a label and a button on a form. When the
user clicks the button, all the directories along the
specified path that don't exist are created. The results are
reported in the caption of the label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString Dir = "C:\\Apps\\Sales\\Local";
  if (ForceDirectories(Dir))
    Label1->Caption = Dir + " was created";
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, all the directories along the
specified path that don't exist are created. The results are
reported in the caption of the label:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Dir: string;
begin
  Dir := 'C:\APPS\SALES\LOCAL';
  if SysUtils.ForceDirectories(Dir) then
    Label1.Caption := Dir + ' was created'
end;
```

See Also

ExtractFileDir (see page 4112)

GetCurrentDir (see page 4202)

RemoveDir (see page 4232)

SetCurrentDir (see page 4240)

CreateDir (see page 4083)

DirectoryExists (see page 4100)

SelectDirectory

37.3.126 SysUtils.Format Function

Returns a formatted string assembled from a format string and an array of arguments.

Pascal

```
function Format(const Format: string; const Args: array of const): string; overload;
function Format(const Format: string; const Args: array of const; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
UnicodeString Format(const UnicodeString Format, const array of const Args);
UnicodeString Format(const UnicodeString Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

The Format function formats the series of arguments in an open (untyped) array.

Format is the format string. For information on the format strings, see Format Strings, described in this topic.

Args is an array of arguments to apply to the format specifiers in Format. For C++, Args_Size specifies the index of the last element of Args (one less than the number of elements).

Format returns the results of applying the arguments in Args to the format string Format.

Two syntax forms are given for Format. The first form of Format is not thread-safe, because it uses localization information contained in global variables. The second form of Format, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of Format, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

Format Strings

Format strings specify required formats to general-purpose formatting routines. Format strings passed to the string formatting routines contain two types of objects -- literal characters and format specifiers. Literal characters are copied verbatim to the resulting string. Format specifiers fetch arguments from the argument list and apply formatting to them.

Format specifiers have the following form:

```
"%" [index ":"] ["-"] [width] ["." prec] type
```

A format specifier begins with a % character. After the % come the following elements, in this order:

An optional argument zero-offset index specifier (that is, the first item has index 0), `[index ":"]`

An optional left justification indicator, `["-"]`

An optional width specifier, `[width]`

An optional precision specifier, `["." prec]`

The conversion type character, `type`

The following table summarizes the possible values for type:

Value	Meaning
d	Decimal. The argument must be an integer value. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.
u	Unsigned decimal. Similar to 'd' but no sign is output.
e	Scientific. The argument must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.
f	Fixed. The argument must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative. The number of digits after the decimal point is given by the precision specifier in the format string—a default of 2 decimal digits is assumed if no precision specifier is present.
g	General. The argument must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string—a default precision of 15 is assumed if no precision specifier is present. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.
n	Number. The argument must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators.
m	Money. The argument must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString (see page 4349), CurrencyFormat (see page 4348), NegCurrFormat (see page 4356), ThousandSeparator (see page 4362), DecimalSeparator (see page 4350), and CurrencyDecimals (see page 4348) global variables or their equivalent in a TFormatSettings (see page 4326) data structure. If the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals (see page 4348) global variable or its TFormatSettings (see page 4326) equivalent.
p	Pointer. The argument must be a pointer value. The value is converted to an 8 character string that represents the pointers value in hexadecimal.
s	String. The argument must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the argument is a string that is longer than this maximum, the string is truncated.
x	Hexadecimal. The argument must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Conversion characters may be specified in uppercase as well as in lowercase—both produce the same results.

For all floating-point formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator (see page 4350) and ThousandSeparator (see page 4362) global variables or their TFormatSettings (see page 4326) equivalent.

Index, width, and precision specifiers can be specified directly using decimal digit string (for example "%10d"), or indirectly using an asterisk character (for example "%*.*f"). When using an asterisk, the next argument in the argument list (which must be an integer value) becomes the value that is actually used. For example,

```
Format ('%*.*f', [8, 2, 123.456]);
```

is equivalent to

```
Format ('%8.2f', [123.456]);
```

Similarly:

```
TVarRec args[3] = {8,2,123.456};
Format ("%*.*f", args, 2);
```

is equivalent to

```
TVarRec args[1] = {123.456};
Format ("%8.2f", args, 0);
```

A width specifier sets the minimum field width for a conversion. If the resulting string is shorter than the minimum field width, it is padded with blanks to increase the field width. The default is to right-justify the result by adding blanks in front of the value, but if the format specifier contains a left-justification indicator (a "-" character preceding the width specifier), the result is left-justified by adding blanks after the value.

An index specifier sets the current argument list index to the specified value. The index of the first argument in the argument list is 0. Using index specifiers, it is possible to format the same argument multiple times. For example "Format('%d %d %0:d %1:d', [10, 20])" produces the string '10 20 10 20'.

Note: Setting the index specifier affects all subsequent formatting. That is, Format('%d %d %d %0:d %d', [1, 2, 3, 4]) returns '1 2 3 1 2', not '1 2 3 1 4'. To get the latter result, you must use Format('%d %d %d %0:d %3:d', [1, 2, 3, 4])

C++ Examples:

```
/*
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Save1Click(TObject *Sender)
{
  AnsiString temp1 = Edit4->Text;
  AnsiString temp2 = ExtractFileName(Edit1->Text);
  AnsiString NewFileName = temp1 + temp2;
  AnsiString OldFileName =
    Edit3->Text + ExtractFileName(Edit2->Text);
  AnsiString Msg = Format("Copy %s to %s", ARRAYOFCONST((OldFileName, NewFileName)));
  if (MessageDlg(Msg, mtCustom, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrOk)
  {
    std::auto_ptr<TFileStream> OldFile(new TFileStream(OldFileName, fmOpenRead |
fmShareDenyWrite));
    std::auto_ptr<TFileStream> NewFile(new TFileStream(NewFileName, fmCreate |
fmShareDenyRead));
    NewFile->CopyFrom(OldFile.get(), OldFile->Size);
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit3->Text = ExtractFilePath(Application->ExeName);
  Edit4->Text = Edit3->Text;
}
/*
This example displays a message on the form's status bar
indicating the record count after a record is deleted.  Set
the StatusBar SimplePanel property to True.
*/
void __fastcall TForm1::Table1AfterDelete(TDataSet *DataSet)
{
  StatusBar1->SimpleText = Format(
    "There are now %d records in the table",
    ARRAYOFCONST(((int)DataSet->RecordCount)));
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Table1->Delete();
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Table1 = new TTable(Form1);
  Table1->Active = false; // The Table component must not be active
  Table1->DatabaseName = "DBDEMOS";
  Table1->TableType = ttParadox;
  Table1->TableName = "CustInfo";
  Table1->Active = False;
  if (!Table1->Exists) // Don't overwrite an existing table
  {
    Table1->Close();
    Table1->DeleteTable();
  }
  // The Table component must not be active
  // First, describe the type of table and give
  // it a name
  // Next, describe the fields in the table
  Table1->FieldDefs->Clear();
  TFieldDef *pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field1";
  pNewDef->DataType = ftInteger;
  pNewDef->Required = true;
  pNewDef = Table1->FieldDefs->AddFieldDef();
  pNewDef->Name = "Field2";
  pNewDef->DataType = ftString;
  pNewDef->Size = 30;
  // Next, describe any indexes
  Table1->IndexDefs->Clear();
  /* the 1st index has no name because it is a Paradox primary key */
  Table1->IndexDefs->Add("","Field1", TIndexOptions() <<ixPrimary << ixUnique);
  Table1->IndexDefs->Add("Fld2Index","Field2", TIndexOptions() << ixCaseInsensitive);
  // Now that we have specified what we want, create the table
  Table1->CreateTable();
  Table1->Active = True;
  for (int i = 1; i <= 20; i++)
    Table1->AppendRecord(ARRAYOFCONST((i*111, i*222)));
  DS2->DataSet = Table1;
  DBGrid2->DataSource->DataSet = Table1;
  Table1->AfterDelete = Table1AfterDelete;
  Table1->Active = True;
}
```

Delphi Examples:

```
{
This example copies a specified old file to a new file.  Add
a TSaveDialog to the form.  Also add two TEdits, two TLabels
and a TButton with the OnClick event named Save1Click. Change
the Old Path and New Path to move files outside the Debug
directory, Old File and New File will not take a relative path.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit3.Text := ExtractFilePath(Application.ExeName);
  Edit4.Text := Edit3.Text;
end;

procedure TForm1.Save1Click(Sender: TObject);
var
  NewFileName, OldFileName: string;
  temp1, temp2: string;
```

```
    Msg: string;
    NewFile: TFileStream;
    OldFile: TFileStream;
  begin
    temp1 := Edit4.Text;
    temp2 := ExtractFileName(Edit1.Text);
    NewFileName := temp1 + temp2;
    OldFileName :=
      Edit3.Text + ExtractFileName(Edit2.Text);
    Msg := Format('Copy %s to %s?', [OldFileName, NewFileName]);
    if MessageDlg(Msg, mtCustom, mbOKCancel, 0) = mrOK then
    begin
      OldFile := TFileStream.Create(
        OldFileName, fmOpenRead or fmShareDenyWrite);
      try
        NewFile := TFileStream.Create(
          NewFileName, fmCreate or fmShareDenyRead);
        try
          NewFile.CopyFrom(OldFile, OldFile.Size);
        finally
          FreeAndNil(NewFile);
        end;
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;
  {
  This example displays a message on the form's status bar
  indicating the record count after a record is deleted.  Set
  the StatusBar SimplePanel property to True.
  }
  procedure TForm1.Table1AfterDelete(DataSet: TDataSet);
  begin
    StatusBar1.SimpleText := Format('There are now %d records in the table',
  [DataSet.RecordCount]);
  end;

  procedure TForm1.Button1Click(Sender: TObject);
  begin
    Table1.Delete;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    Table1:= TTable.Create(Form1);
    with Table1 do
    begin
      DatabaseName := 'DBDEMOS';
      TableType := ttParadox;
      TableName := 'CustInfo';
      Name := 'CustInfo';
      Table1.Active := False;
      { Don't overwrite an existing table }
      if Table1.Exists then
      begin
          Table1.Close;
          Table1.DeleteTable;
      end;
      begin
        { The Table component must not be active }
        { First, describe the type of table and give }
        { it a name }
        { Next, describe the fields in the table }
```

```
        with FieldDefs do
        begin
          Clear;
          with AddFieldDef do
          begin
            Name := 'Field1';
            DataType := ftInteger;
            Required := True;
          end;
          with AddFieldDef do
          begin
            Name := 'Field2';
            DataType := ftString;
            Size := 30;
          end;
        end;
        { Next, describe any indexes }
        with IndexDefs do
        begin
          Clear;
          { The 1st index has no name because it is
          { a Paradox primary key }
          with AddIndexDef do
          begin
            Name := '';
            Fields := 'Field1';
            Options := [ixPrimary];
          end;
          with AddIndexDef do
          begin
            Name := 'Fld2Indx';
            Fields := 'Field2';
            Options := [ixCaseInsensitive];
          end;
        end;
        { Call the CreateTable method to create the table }
        CreateTable;
        Table1.Active:= True;
        for i := 1 to 20 do
          Table1.AppendRecord([i*111, i*222]);
      end;
    end;
    DS2.DataSet:= Table1;
    DBGrid2.DataSource.DataSet:= Table1;
    Table1.AfterDelete:= Table1AfterDelete;
    Table1.Active:= True;
  end;
```

See Also

Date (see page 4087)

37.3.127 SysUtils.FormatBuf Function

Formats the arguments from an array, placing the result in a buffer.

Pascal

```
function FormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const Args:
array of const): Cardinal; overload;
function FormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const Args:
array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
function FormatBuf(Buffer: PWideChar; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
```

```
Args: array of const): Cardinal; overload;
function FormatBuf(var Buffer: UnicodeString; BufLen: Cardinal; const Format; FmtLen:
Cardinal; const Args: array of const): Cardinal; overload;
function FormatBuf(Buffer: PWideChar; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
function FormatBuf(var Buffer: UnicodeString; BufLen: Cardinal; const Format; FmtLen:
Cardinal; const Args: array of const; const FormatSettings: TFormatSettings): Cardinal;
overload;
```

C++

```
unsigned FormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array of
const Args);
unsigned FormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array of
const Args, const TFormatSettings FormatSettings);
unsigned FormatBuf(PWideChar Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const
array of const Args);
unsigned FormatBuf(UnicodeString Buffer, unsigned BufLen, const  Format, unsigned FmtLen,
const array of const Args);
unsigned FormatBuf(PWideChar Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const
array of const Args, const TFormatSettings FormatSettings);
unsigned FormatBuf(UnicodeString Buffer, unsigned BufLen, const  Format, unsigned FmtLen,
const array of const Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the specified open array into a text buffer.

Buffer is the text buffer that receives the results.

BufLen is the number of bytes in Buffer.

Format (see page 4188) is the format string that determines how arguments are formatted. For information on the format strings, see Format (see page 4188) Strings.

FmtLen is the number of bytes in Format (see page 4188).

Args are the arguments that replace the format specifiers in Format (see page 4188).

Note: Args_Size is the index of the last element in Args (one less than the number of arguments).

FormatBuf function returns the length, in bytes, of the resulting string.

The first form of FormatBuf is not thread-safe, because it uses localization information contained in global variables. The second form of FormatBuf, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatBuf, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

GetLocaleFormatSettings (see page 4206)

37.3.128 SysUtils.FormatCurr Function

Formats a System::Currency (see page 3235) object.

Pascal

```
function FormatCurr(const Format: string; Value: Currency): string; overload;
function FormatCurr(const Format: string; Value: Currency; const FormatSettings:
```

```
TFormatSettings): string; overload;
```

C++

```
UnicodeString FormatCurr(const UnicodeString Format, Currency Value);
UnicodeString FormatCurr(const UnicodeString Format, Currency Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

FormatCurr formats the System::Currency (see page 3235) object given by Value using the format string given by Format (see page 4188).

For information on the format strings, see FormatFloat (see page 4197) Strings.

The first form of FormatCurr is not thread-safe, because it uses localization information contained in global variables. The second form of FormatCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

CurrToStr (see page 4086)

CurrToStrF (see page 4086)

StrToCurr (see page 4278)

FormatFloat (see page 4197)

GetLocaleFormatSettings (see page 4206)

37.3.129 SysUtils.FormatDateTime Function

Formats a TDateTime (see page 3323) value.

Pascal

```
function FormatDateTime(const Format: string; DateTime: TDateTime): string; overload;
function FormatDateTime(const Format: string; DateTime: TDateTime; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
UnicodeString FormatDateTime(const UnicodeString Format, TDateTime DateTime);
UnicodeString FormatDateTime(const UnicodeString Format, TDateTime DateTime, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

FormatDateTime formats the TDateTime (see page 3323) value given by DateTime using the format given by Format (see page 4188). See the table below for information about the supported format strings.

If the string specified by the Format (see page 4188) parameter is empty, the TDateTime (see page 3323) value is formatted as if a 'c' format specifier had been given.

The first form of FormatDateTime is not thread-safe, because it uses localization information contained in global variables. The

second form of FormatDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

Date (see page 4087) Time (see page 4297) Format (see page 4188) Strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d"), simply format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables.

In the following table, specifiers are given in lower case. Case is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Displays
c	Displays the date using the format given by the ShortDateFormat (see page 4358) global variable, followed by the time using the format given by the LongTimeFormat (see page 4356) global variable. The time is not displayed if the date-time value indicates midnight precisely.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames (see page 4359) global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames (see page 4355) global variable.
ddddd	Displays the date using the format given by the ShortDateFormat (see page 4358) global variable.
dddddd	Displays the date using the format given by the LongDateFormat (see page 4354) global variable.
e	(Windows only) Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee	(Windows only) Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	(Windows only) Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	(Windows only) Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec (see page 3550)) using the strings given by the ShortMonthNames (see page 4359) global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames (see page 4355) global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).

t	Displays the time using the format given by the ShortTimeFormat (see page 4360) global variable.
tt\	Displays the time using the format given by the LongTimeFormat (see page 4356) global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString (see page 4362) global variable for any hour before noon, and the contents of the TimePMString (see page 4362) global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator (see page 4349) global variable.
:	Displays the time separator character given by the TimeSeparator (see page 4363) global variable.
'xx'/"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

C++ Examples:

```
/*
The following example uses FormatDateTime to set the string
variable S to a sentence indicating a meeting time in 3
hours. The sentence has the form "The meeting is on
Wednesday, February 15, 1995 at 2:30 PM."
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString S = "The meeting is on ";
  S = S + FormatDateTime("dddd, mmmm d, yyyy ' at ' hh:mm AM/PM", IncHour(Now(), 3));
  Label1->Caption = S;
}
```

Delphi Examples:

```
{
The following example uses FormatDateTime to set the string
variable S to a sentence indicating a meeting time in 3
hours. The sentence has the form "The meeting is on
Wednesday, February 15, 2008 at 2:30 PM."
}
procedure TForm1.Button1Click(Sender: TObject);
var S : string;
begin
  S := SysUtils.FormatDateTime(
    '"The meeting is on " dddd, mmmm d, yyyy, " at " hh:mm AM/PM', IncHour(Now(), 3));
  Label1.Caption := S;
end;
```

37.3.130 SysUtils.FormatFloat Function

Formats a floating point value.

Pascal

```
function FormatFloat(const Format: string; Value: Extended): string; overload;
function FormatFloat(const Format: string; Value: Extended; const FormatSettings:
TFormatSettings): string; overload;
```

C++

```
UnicodeString FormatFloat(const UnicodeString Format, Extended Value);
```

```
UnicodeString FormatFloat(const UnicodeString Format, Extended Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

FormatFloat formats the floating-point value given by Value using the format string given by Format (see page 4188). The following format specifiers are supported in the format string:

Specifier	Represents
0	Digit place holder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. Otherwise, a '0' is stored in that position in the output string.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string.
.	Decimal point. The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The actual character used as a the decimal separator in the output string is determined by the DecimalSeparator (see page 4350) global variable or its TFormatSettings (see page 4326) equivalent.
,	Thousand separator. If the format string contains one or more ',' characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of ',' characters in the format string does not affect the output, except to indicate that thousand separators are wanted. The actual character used as a the thousand separator in the output is determined by the ThousandSeparator (see page 4362) global variable or its TFormatSettings (see page 4326) equivalent.
E+	Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
'xx'/"xx"	Characters enclosed in single or double quotes are output as-is, and do not affect formatting.
;	Separates sections for positive, negative, and zero numbers in the format string.

The locations of the leftmost '0' before the decimal point in the format string and the rightmost '0' after the decimal point in the format string determine the range of digits that are always present in the output string.

The number being formatted is always rounded to as many decimal places as there are digit placeholders ('0' or '#') to the right of the decimal point. If the format string contains no decimal point, the value being formatted is rounded to the nearest whole number.

If the number being formatted has more digits to the left of the decimal separator than there are digit placeholders to the left of the '.' character in the format string, the extra digits are output before the first digit placeholder.

To allow different formats for positive, negative, and zero values, the format string can contain between one and three sections separated by semicolons.

One section: The format string applies to all values.

Two sections: The first section applies to positive values and zeros, and the second section applies to negative values.

Three sections: The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

If the section for negative values or the section for zero values is empty, that is if there is nothing between the semicolons that delimit the section, the section for positive values is used instead.

If the section for positive values is empty, or if the entire format string is empty, the value is formatted using general floating-point formatting with 15 significant digits, corresponding to a call to FloatToStrF (see page 4182) with the ffGeneral format. General

floating-point formatting is also used if the value has more than 18 digits to the left of the decimal point and the format string does not specify scientific notation.

The first form of FormatFloat is not thread-safe, because it uses localization information contained in global variables. The second form of FormatFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of FormatFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

C++ Examples:

```
/*
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int x = 1; x < StringGrid1->ColCount; x++)
  {
    for (int y = 1; y < StringGrid1->RowCount; y++)
    {
      StringGrid1->Cells[x][y] = FormatFloat(
      StringGrid1->Cells[0][y], StrToFloat(StringGrid1->Cells[x][0]));
    }
  }
}

int currCellCol, currCellRow;

void __fastcall TForm1::StringGrid1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  StringGrid1->MouseToCell(X, Y, currCellCol, currCellRow);
}

// Alter Cell
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  StringGrid1->Cells[currCellCol][currCellRow] = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "1234";
  StringGrid1->Cells[2][0] = "-1234";
  StringGrid1->Cells[3][0] = "0.5";
  StringGrid1->Cells[4][0] = "0";
  StringGrid1->Cells[0][1] = "";
  StringGrid1->Cells[0][2] = "0";
  StringGrid1->Cells[0][3] = "0.00";
  StringGrid1->Cells[0][4] = "#.##";
  StringGrid1->Cells[0][5] = "#,##0.00";
  StringGrid1->Cells[0][6] = "#,##0.00;(#,##0.00)";
  StringGrid1->Cells[0][7] = "#,##0.00;;Zero";
  StringGrid1->Cells[0][8] = "0.000E+00";
  StringGrid1->Cells[0][9] = "#.###E-0";
}
/*
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  ShowMessage(FormatFloat("#,##0 bytes", DiskFree(0)));
}
```

Delphi Examples:

```
{
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, I: Integer;
begin
  for X := 1 to StringGrid1.ColCount - 1 do
  begin
    for Y := 0 to StringGrid1.RowCount - 1 do
    begin
      StringGrid1.Cells[X,Y] := SysUtils.FormatFloat(
        StringGrid1.Cells[0,Y], StrToFloat(StringGrid1.Cells[X,0]));
    end;
  end;

end;

var currCellCol, currCellRow: Integer;

// Alter Cell
procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid1.Cells[currCellCol,currCellRow] := Edit1.Text;
end;

procedure TForm1.StringGrid1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  StringGrid1.MouseToCell(X, Y, &currCellCol, &currCellRow);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[1,0] := '1234';
  StringGrid1.Cells[2,0] := '-1234';
  StringGrid1.Cells[3,0] := '0.5';
  StringGrid1.Cells[4,0] := '0';
  StringGrid1.Cells[0,1] := '';
  StringGrid1.Cells[0,2] := '0';
  StringGrid1.Cells[0,3] := '0.00';
  StringGrid1.Cells[0,4] := '#.##';
  StringGrid1.Cells[0,5] := '#,##0.00';
  StringGrid1.Cells[0,6] := '#,##0.00;(#,##0.00)';
  StringGrid1.Cells[0,7] := '#,##0.00;;Zero';
  StringGrid1.Cells[0,8] := '0.000E+00';
  StringGrid1.Cells[0,9] := '#.###E-0';
end;
{
The following example displays in a dialog the disk free
space of the current drive with commas as thousand separators.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage(FormatFloat('#,##0 bytes', DiskFree(0)));
end;
```

See Also

FloatToStrF (see page 4182)

FloatToText (see page 4183)

FloatToTextFmt (see page 4184)

GetLocaleFormatSettings (see page 4206)

37.3.131 SysUtils.FreeAndNil Function

Frees an object reference and replaces the reference with nil (Delphi) or NULL (C++).

Pascal

```
procedure FreeAndNil(var Obj);
```

C++

```
FreeAndNil( Obj);
```

File

SysUtils

Description

Use FreeAndNil to ensure that a variable is nil (Delphi) or NULL (C++) after you free the object it references. Pass any variable that represents an object as the Obj parameter.

Warning: Obj must be an instance of a TObject (see page 3354) descendant.

Delphi Examples:

```
{
This code displays a Save Picture dialog box with the
TBitmap default extension that is added automatically to
files that are given no extension. Confirm that the
bitmap file has been saved under the filename and path
specified.
}
procedure TForm1.Button1Click(Sender: TObject);
var NewFileName, OldFileName: string;
    NewFile, OldFile: TFileStream;
begin
  SavePictureDialog1.DefaultExt := GraphicExtension(TBitmap);
  SavePictureDialog1.Filter := GraphicFilter(TBitmap);
  OldFileName := 'factory.bmp';
  if SavePictureDialog1.Execute then
  begin
    NewFileName := SavePictureDialog1.FileName;
    OldFile := TFileStream.Create(
      OldFileName, fmOpenRead or fmShareDenyWrite);
    try
      NewFile := TFileStream.Create(
        NewFileName, fmCreate or fmShareDenyRead);
      try
        NewFile.CopyFrom(OldFile, OldFile.Size);
      finally
        FreeAndNil(NewFile);
      end;
```

```
      finally
        FreeAndNil(OldFile);
      end;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    Image1.Picture.LoadFromFile('factory.bmp');
  end;
```

See Also

System::TObject (see page 3354)

37.3.132 SysUtils.GDAL Function

Get Delphi Access Licence.

Pascal

```
function GDAL: LongWord;
```

C++

```
LongWord GDAL();
```

File

SysUtils

Description

GDAL retreives the access licences resource. It checks if it is valid, if not an Access Licence Violation exception with the message 'Application is not licenced to use this feature' is raised.

The returned value is the decrypted first Access Licence (AL1).

See Also

RCS (see page 4232)

RPR (see page 4237)

37.3.133 SysUtils.GetCurrentDir Function

Returns the name of the current directory.

Pascal

```
function GetCurrentDir: string;
```

C++

```
UnicodeString GetCurrentDir();
```

File

SysUtils

Description

GetCurrentDir returns the fully qualified name of the current directory.

C++ Examples:

```
/*
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
*/
void __fastcall TMyChDir::Button1Click(TObject *Sender)
{
  try
  {
    System::ChDir(Edit1->Text);
  }
  catch (...)
  {
    MessageDlg("Cannot find directory", mtWarning, TMsgDlgButtons() << mbOK, 0);
  }
  Edit2->Text = GetCurrentDir();
  FileListBox1->Directory = GetCurrentDir();
  DirectoryListBox1->Directory = GetCurrentDir();
}

void __fastcall TMyChDir::DirectoryListBox1Change(TObject *Sender)
{
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}

void __fastcall TMyChDir::DriveComboBox1Change(TObject *Sender)
{
  DirectoryListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Drive = DriveComboBox1->Drive;
  FileListBox1->Directory = DirectoryListBox1->Directory;
  Edit2->Text = GetCurrentDir();
}
```

Delphi Examples:

```
{
This example shows how to change and display the current
directory through text.  This example requires that a drive
combo box, a file list box, and a directory list box are on
a form. Add this code as the OnChange event handler for the
drive combo box and the OnChange event handler for the
directory list box.  When the user changes the drive using
the combo box, the directory list box and file list box will
update to reflect the new drive and the current directory on
that drive.  When the user double clicks on a directory in
the file list box will update to reflect the new directory.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  System.ChDir(Edit1.Text);
  if IOResult <> 0 then
    MessageDlg('Cannot find directory', mtWarning, [mbOk], 0);
  Edit2.Text := GetCurrentDir;
  FileListBox1.Directory := GetCurrentDir;
  DirectoryListBox1.Directory := GetCurrentDir;
end;
```

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
  DirectoryListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Drive := DriveComboBox1.Drive;
  FileListBox1.Directory := DirectoryListBox1.Directory;
  Edit2.Text := GetCurrentDir;
end;
```

See Also

GetDir (see page 3580)

CreateDir (see page 4083)

ExtractFileDir (see page 4112)

RemoveDir (see page 4232)

SetCurrentDir (see page 4240)

37.3.134 SysUtils.GetEnvironmentVariable Function

Retrieves an environment variable value.

Pascal

```
function GetEnvironmentVariable(const Name: string): string; overload;
```

C++

```
UnicodeString GetEnvironmentVariable(const UnicodeString Name);
```

File

SysUtils

Description

Call GetEnvironmentVariable to retrieve the value of an environment variable, passed as `Name`, for the current process.

Note: The case of the specified environment variable `Name` is ignored.

The following table defines the standard environment variables for Microsoft Windows.

Environment variable name	Environment value
ALLUSERSPROFILE	Generic user profile.
APPDATA	Path of the application data folder.
CLIENTNAME	Name of Client machine.
COMMONPROGRAMFILES	Path of common program files folder.
COMPUTERNAME	Name of Computer code is running on.
COMSPEC	Path of the cmd.exe program.
HOMEDRIVE	Current home drive designation, such as 'C:'
HOMEPATH	Path to current location for document storage.

LOGONSERVER	
NUMBER_OF_PROCESSORS	Number of processors on current machine.
OS	Base name of the Operating System (see page 3159). Note that Windows XP is given as Windows_NT.
PATH	The current program path.
PATHEXT	
PROCESSOR_ARCHITECTURE	Type of CPU architecture. For example, X86 for Intel Pentium processors.
PROCESSOR_IDENTIFIER	
PROCESSOR_LEVEL	More detailed description of the CPU architecture.
PROCESSOR_REVISION	Processor revision level.
PROGRAMFILES	Path of the program files folder.
SESSIONNAME	Name of the current OS session.
SYSTEMDRIVE	Drive the OS operates from.
SYSTEMROOT	
TEMP	Path of the temporary files folder.
TMP	
USERDOMAIN	
USERNAME	Name of the current user.
USERPROFILE	Path of the folder holding the current user's information.
WINDIR	Path of the Windows folder.

37.3.135 SysUtils.GetFileVersion Function

Returns file version.

Pascal

```
function GetFileVersion(const AFileName: string): Cardinal;
```

C++

```
unsigned GetFileVersion(const UnicodeString AFileName);
```

File

SysUtils

Description

GetFileVersion returns the most significant 32 bits of a file binary version number. AFileName specifies the name of the file, which is searched using the same path as the LoadLibrary function.

GetFileVersion is only available on Windows and must not be used in cross-platform code.

37.3.136 SysUtils.GetFormatSettings Function

Resets the date and number format parameters to initial values.

Pascal

```
procedure GetFormatSettings;
```

C++

```
GetFormatSettings();
```

File

SysUtils

Description

GetFormatSettings resets all date and number format variables to their initial values.

Note: On Windows, GetFormatSettings is called automatically in response to WM_WININICHANGE messages.

37.3.137 SysUtils.GetLocaleChar Function

Retrieves the first character of a localisation value.

Pascal

```
function GetLocaleChar(Locale: Integer; LocaleType: Integer; Default: Char): Char; platform;
```

C++

```
Char GetLocaleChar(int Locale, int LocaleType, Char Default);
```

File

SysUtils

Description

GetLocaleChar retrieves the first character of the passed LocaleType value, or the passed Default value if not found. The actual value is determined by the location specified by Locale. For example, on Windows, $0809 is the Locale for United Kingdom English, and $0409 is the Locale for US English. For the LOCALE_SCURRENCY LocaleType, these locales would yield '£' and '$' respectively.

Note: This is an OS specific function.

See Also

GetLocaleStr (see page 4207)

37.3.138 SysUtils.GetLocaleFormatSettings Function

Populates a TFormatSettings (see page 4326) data structure.

Pascal

```
procedure GetLocaleFormatSettings(LCID: Integer; var FormatSettings: TFormatSettings);
```

C++

```
GetLocaleFormatSettings(int LCID, TFormatSettings FormatSettings);
```

File

SysUtils

Description

GetLocaleFormatSettings populates FormatSettings based on the Windows locale ID specified in LCID. A TFormatSettings (see page 4326) data structure must be initialized by a call to GetLocaleFormatSettings before it can be used with any string formatting routines.

37.3.139 SysUtils.GetLocaleStr Function

Retrieves a localisation value.

Pascal

```
function GetLocaleStr(Locale: Integer; LocaleType: Integer; const Default: string): string;
platform;
```

C++

```
UnicodeString GetLocaleStr(int Locale, int LocaleType, const UnicodeString Default);
```

File

SysUtils

Description

GetLocaleStr retrieves the passed LocaleType value, or the passed Default value if not found. The actual value is determined by the location specified by Locale. For example, on Windows, $040c is the Locale for France French, and $0409 is the Locale for US English. For the LOCALE_SMONTHNAME1 LocaleType, these locales would yield 'janvier' and 'January' respectively.

Note: This is an OS specific function.

See Also

GetLocaleChar (see page 4206)

37.3.140 SysUtils.GetModuleName Function

Returns the fully qualified name for a module, given its handle.

Pascal

```
function GetModuleName(Module: HMODULE): string;
```

C++

```
UnicodeString GetModuleName(HMODULE Module);
```

File

SysUtils

Description

Call GetModuleName to obtain the fully qualified name of a module, given its handle.

Module is the handle of the module whose file name is required.

Warning: :On Linux, GetModuleName is not always successful. Depending on the way the application is called, the presence of the /proc file system, and other factors, there are times when GetModuleName may return a name that is not fully qualified or in some cases, no file name at all. Be sure to check the results of this function before using the returned value.

C++ Examples:

```
/*
This example requires a button and a label.  When the user clicks
the button, the GetModuleName function retieves the path and name
of the project executable using the file handle HInstance.
*/
AnsiString __fastcall GetUsersFileName(void)
{
  return GetModuleName(unsigned(HInstance));
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = GetUsersFileName();
}
```

Delphi Examples:

```
{
This example requires a button and a label.  When the user clicks
the button, the GetModuleName function retieves the path and name
of the project executable using the file handle HInstance.
}
function GetUsersFileName: string;
begin
  Result := SysUtils.GetModuleName(HInstance);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := GetUsersFileName;
end;
```

37.3.141 SysUtils.GetPackageDescription Function

Returns the description resource from a package.

Pascal

```
function GetPackageDescription(ModuleName: PChar): string;
```

C++

```
UnicodeString GetPackageDescription(const char * ModuleName);
```

File

SysUtils

Description

Call GetPackageDescription to obtain the description stored with the named package. If the package does not have a description

resource, GetPackageDescription returns an empty string.

See Also

GetPackageInfo (see page 4209)

37.3.142 SysUtils.GetPackageInfo Function

Enumerates all the units and required packages for a package.

Pascal

```
procedure GetPackageInfo(Module: HMODULE; Param: Pointer; var Flags: Integer; InfoProc:
TPackageInfoProc);
```

C++

```
GetPackageInfo(HMODULE Module, void * Param, int Flags, TPackageInfoProc InfoProc);
```

File

SysUtils

Description

Call GetPackageInfo to process the information in a package's information table. The InfoProc callback is called for every unit included in the package and for every package required by the package. Module is the module handle for the package. GetPackageInfo passes any application-supplied information passed in the Param parameter to the InfoProc callback. The package's information flags are passed to the InfoProc and returned in the Flags parameter.

See Also

LoadPackage (see page 4224)

GetPackageDescription (see page 4208)

37.3.143 SysUtils.GetTime Function

Returns the current time.

Pascal

```
function GetTime: TDateTime;
```

C++

```
TDateTime GetTime();
```

File

SysUtils

Description

GetTime and Time (see page 4297) return the current time as a TDateTime (see page 3323) value. The two functions are completely equivalent.

Note: Time (see page 4297) clashes with an Xlib function of the same name. Use GetTime in Linux and cross-platform code.

See Also

Time (see page 4297)

Date (see page 4087)

DecodeTime (see page 4097)

Now (see page 4227)

TimeToStr (see page 4299)

37.3.144 SysUtils.GUIDToString Function

Converts a class identifier GUID to a string.

Pascal

```
function GUIDToString(const GUID: TGUID): string;
```

C++

```
UnicodeString GUIDToString(const TGUID GUID);
```

File

SysUtils

Description

GUIDToString converts a GUID into a string of printable characters. Each GUID always converts to a unique string.

If GUIDToString succeeds it returns a string for the requested GUID. If GUIDToString fails, it raises EConvertError (see page 3919).

C++ Examples:

```
/*
The following example manufactures a GUID and displays it in
a dialog box after translating it to a string.
*/
#include <ActiveX.hpp>
#include <ComObj.hpp>
#include <Sysutils.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    TGUID g;
    OleCheck(CoCreateGuid(&g));
    ShowMessage(Sysutils::GUIDToString(g));
}
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
```

```
            "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
          ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
          myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
          myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
The following example manufactures a GUID and displays it in
a dialog box after translating it to a string.
}
uses ActiveX, ComObj;
procedure TForm1.Button1Click(Sender: TObject);
var
  G: TGUID;
begin
  OleCheck(CoCreateGuid(G));
  ShowMessage(SysUtils.GUIDToString(G));
end;
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
```

```
    if IsEqualGUID(MyGuid0, MyGuid1) then
       Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                       '2 randomly generated GUIDs cannot be equal!');
  end;
```

See Also

StringToGUID (see page 4254)

EConvertError (see page 3919)

37.3.145 SysUtils.HashName Function

Pascal

```
function HashName(Name: PAnsiChar): Cardinal;
```

C++

```
unsigned HashName(PAnsiChar Name);
```

File

SysUtils

Description

This is function SysUtils.HashName.

37.3.146 SysUtils.IncAMonth Function

Increments date data by one month.

Pascal

```
procedure IncAMonth(var Year: Word; var Month: Word; var Day: Word; NumberOfMonths: Integer =
1);
```

C++

```
IncAMonth(Word Year, Word Month, Word Day, int NumberOfMonths = 1);
```

File

SysUtils

Description

IncAMonth modifies Year, Month, and Day parameters so the date they describe is incremented by NumberOfMonths months. NumberOfMonths can be negative, to return a date N months previous.

If the input day of month is greater than the last day of the resulting month, the day is set to the last day of the resulting month.

See Also

IncMonth (see page 4214)

DecodeDate (see page 4095)

EncodeDate (see page 4105)

ReplaceDate (see page 4236)

IncDay

IncHour

IncMilliSecond

IncMinute

IncSecond

IncWeek

IncYear

37.3.147 SysUtils.IncludeTrailingBackslash Function

Ensures path name ends with delimiter

Pascal

```
function IncludeTrailingBackslash(const S: string): string; overload; platform;
```

C++

```
UnicodeString IncludeTrailingBackslash(const UnicodeString S);
```

File

SysUtils

Description

IncludeTrailingBackslash ensures that a path name ends with a trailing path delimiter ('\' on Windows, '/' on Linux). This function is included for backward-compatibility only. New (see page 3619) programs should use IncludeTrailingPathDelimiter (see page 4213).

Note: IncludeTrailingBackslash works with multibyte character sets.

See Also

ExcludeTrailingBackslash (see page 4108)

IsPathDelimiter (see page 4222)

37.3.148 SysUtils.IncludeTrailingPathDelimiter Function

Ensures path name ends with delimiter.

Pascal

```
function IncludeTrailingPathDelimiter(const S: string): string; overload;
```

C++

```
UnicodeString IncludeTrailingPathDelimiter(const UnicodeString S);
```

File

SysUtils

Description

IncludeTrailingPathDelimiter ensures that a path name ends with a trailing path delimiter ('\" on Windows, '/' on Linux). If S already ends with a trailing delimiter character, it is returned unchanged; otherwise S with appended delimiter character is returned.

Note: IncludeTrailingPathDelimiter works with multibyte character sets.

See Also

ExcludeTrailingPathDelimiter (see page 4109)

IsPathDelimiter (see page 4222)

37.3.149 SysUtils.IncMonth Function

Returns a date shifted by a specified number of months.

Pascal

```
function IncMonth(const DateTime: TDateTime; NumberOfMonths: Integer = 1): TDateTime;
```

C++

```
TDateTime IncMonth(const TDateTime DateTime, int NumberOfMonths = 1);
```

File

SysUtils

Description

IncMonth returns the value of the Date (see page 4087) parameter, incremented by NumberOfMonths months. NumberOfMonths can be negative, to return a date N months previous.

If the input day of month is greater than the last day of the resulting month, the day is set to the last day of the resulting month. The time of day specified by the Date (see page 4087) parameter is copied to the result.

See Also

IncAMonth (see page 4212)

DecodeDate (see page 4095)

EncodeDate (see page 4105)

ReplaceDate (see page 4236)

IncDay

IncHour

IncMilliSecond

IncMinute

IncSecond

IncWeek

IncYear

37.3.150 SysUtils.InitializePackage Function

Validates and initializes a given package.

Pascal

```
procedure InitializePackage(Module: HMODULE); overload;
procedure InitializePackage(Module: HMODULE; AValidatePackage: TValidatePackageProc); overload;
```

C++

```
InitializePackage(HMODULE Module);
InitializePackage(HMODULE Module, TValidatePackageProc AValidatePackage);
```

File

SysUtils

Description

InitializePackage checks for duplicate units in a package and calls the initialization blocks of the contained units.

Note: InitializePackage is called internally by the LoadPackage (see page 4224) function.

See Also

LoadPackage (see page 4224)

37.3.151 SysUtils.IntToHex Function

Returns the hex representation of an integer.

Pascal

```
function IntToHex(Value: Integer; Digits: Integer): string; overload;
function IntToHex(Value: Int64; Digits: Integer): string; overload;
```

C++

```
UnicodeString IntToHex(int Value, int Digits);
UnicodeString IntToHex(Int64 Value, int Digits);
```

File

SysUtils

Description

IntToHex converts a number into a string containing the number's hexadecimal (base 16) representation. Value is the number to convert. Digits indicates the minimum number of hexadecimal digits to return.

C++ Examples:

```
/*
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = "";
```

```
    for (int i = 1; i <= Edit1->Text.Length(); i++)
    {
      try
      {
        Label1->Caption =
          Label1->Caption +
          IntToHex(Byte(Edit1->Text[i]),2) + ' ';
      }
      catch (...)
      {
        Beep();
      }
    }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, and a
label on a form. When the button is clicked, the hexadecimal
value of each character in the edit control is displayed in
the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
begin
  Label1.Caption := '';
  for i := 1 to Length(Edit1.Text) do
  begin
    try
      Label1.Caption :=
        Label1.Caption +
        SysUtils.IntToHex(Byte(Edit1.Text[i]),2) + ' ';
    except
      Beep;
    end;
  end;
end;
```

See Also

IntToStr (see page 4216)

37.3.152 SysUtils.IntToStr Function

Converts an integer to a string.

Pascal

```
function IntToStr(Value: Integer): string; overload;
function IntToStr(Value: Int64): string; overload;
```

C++

```
UnicodeString IntToStr(int Value);
UnicodeString IntToStr(Int64 Value);
```

File

SysUtils

Description

IntToStr converts an integer into a string containing the decimal representation of that number.

C++ Examples:

```
/*
The following example uses two edit controls, a button, and
a label on a form. When the button is clicked, the integers
in the edit controls are multiplied together and the result
is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  try
  {
    Label1->Caption = IntToStr(StrToInt(Edit1->Text) * StrToInt(Edit2->Text));
  }
  catch(...)
  {
    ShowMessage("You must specify integer values. Please try again.");
  }
}
```

Delphi Examples:

```
{
The following example uses two edit controls, a button, and
a label on a form. When the button is clicked, the integers
in the edit controls are multiplied together and the result
is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  try
    Label1.Caption :=
      SysUtils.IntToStr(StrToInt(Edit1.Text) * StrToInt(Edit2.Text));
  except
    ShowMessage('You must specify integer values. Please try again.');
  end;
end;
```

See Also

IntToHex (see page 4215)

StrToInt (see page 4286)

StrToIntDef (see page 4288)

FloatToStr (see page 4180)

DateToStr (see page 4092)

TimeToStr (see page 4299)

37.3.153 SysUtils.IsAssembly Function

Determines whether a dll is an Assembly.

Pascal

```
function IsAssembly(const FileName: string): Boolean;
```

C++

```
Boolean IsAssembly(const UnicodeString FileName);
```

File

SysUtils

Description

Call IsAssembly to determine whether the passed FileName is that of a .Net Assembly or a traditional WIN32 DLL. The result is true if FileName is an Assembly.

37.3.154 SysUtils.IsDelimiter Function

Indicates whether a specified character in a string matches one of a set of delimiters.

Pascal

```
function IsDelimiter(const Delimiters: string; const S: string; Index: Integer): Boolean;
overload;
```

C++

```
Boolean IsDelimiter(const UnicodeString Delimiters, const UnicodeString S, int Index);
```

File

SysUtils

Description

Call IsDelimiter to determine whether the character at byte offset Index in the string S is one of the delimiters in the string Delimiters. Index is the index of the byte in question, where 1 is the first byte of the string, 2 is the second byte, and so on.

When working with a multi-byte character system (MBCS), IsDelimiter checks to make sure the indicated byte is not part of a double byte character. The delimiters in the Delimiters parameter must all be single byte characters.

See Also

AnsiStrRScan (see page 4060)

AnsiStrScan (see page 4061)

IsPathDelimiter (see page 4222)

LastDelimiter (see page 4223)

37.3.155 SysUtils.IsEqualGUID Function

Indicates whether two TGUID (see page 3718) values are the same.

Pascal

```
function IsEqualGUID(const guid1: TGUID; const guid2: TGUID): Boolean; stdcall;
```

C++

```
__stdcall Boolean IsEqualGUID(const TGUID guid1, const TGUID guid2);
```

File

SysUtils

Description

IsEqualGUID returns true if the TGUID (see page 3718) values are equal.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7])));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));

    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));
```

```
    { Auto-generate a random GUID at runtime }
    if CreateGUID(MyGuid1) <> 0 then
       Memo1.Lines.Add('Creating GUID failed!')
    else
       Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

    { Generating second random GUID }
    CreateGUID(MyGuid0);

    { Testing if 2 guids are equal }
    if IsEqualGUID(MyGuid0, MyGuid1) then
       Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                       '2 randomly generated GUIDs cannot be equal!');
end;
```

37.3.156 SysUtils.IsLeadChar Function

Pascal

```
function IsLeadChar(C: AnsiChar): Boolean; overload;
```

C++

```
Boolean IsLeadChar(AnsiChar C);
```

File

SysUtils

Description

This is function SysUtils.IsLeadChar.

37.3.157 SysUtils.IsLeadChar Function

Pascal

```
function IsLeadChar(C: WideChar): Boolean; overload;
```

C++

```
Boolean IsLeadChar(WideChar C);
```

File

SysUtils

Description

This is function SysUtils.IsLeadChar.

37.3.158 SysUtils.IsLeapYear Function

Indicates whether a specified year is a leap year.

Pascal

```
function IsLeapYear(Year: Word): Boolean;
```

C++

```
Boolean IsLeapYear(Word Year);
```

File

SysUtils

Description

Call IsLeapYear to determine whether the year specified by the Year parameter is a leap year. Year specifies the calendar year.

Use YearOf to obtain the value of Year for IsLeapYear from a TDateTime (see page 3323) value.

C++ Examples:

```
/*
This example requires a button and a text edit. When the
user presses the button a message indicates if the year
specified in the text edit is a leap year.
*/
bool __fastcall IsThisLeapYear(int year)
{
  return IsLeapYear(year);
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int year = StrToInt(Edit1->Text);
  if (IsThisLeapYear(year))
    ShowMessage(Edit1->Text + " is a leap year.");
  else
    ShowMessage(Edit1->Text + " is not a leap year.");
}
```

Delphi Examples:

```
{
This example requires a button and a text edit. When the
user presses the button a message indicates if the year
specified in the text edit is a leap year.
}
function IsThisLeapYear(year : Word): Boolean;
begin
  Result := SysUtils.IsLeapYear(year);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  year : integer;
begin
  year := StrToInt(Edit1.Text);
  if (IsThisLeapYear(year)) then
    ShowMessage(Edit1.Text + ' is a leap year.')
  else
    ShowMessage(Edit1.Text + ' is not a leap year.');
end;
```

See Also

Date (see page 4087)

YearOf

IsInLeapYear

37.3.159 SysUtils.IsPathDelimiter Function

Indicates whether the byte at a specified position in a string is the path delimiter.

Pascal

```
function IsPathDelimiter(const S: string; Index: Integer): Boolean; overload;
```

C++

```
Boolean IsPathDelimiter(const UnicodeString S, int Index);
```

File

SysUtils

Description

Call IsPathDelimiter to determine whether a position in the string S contains the path delimiter character ('/' on Linux, '\' on Windows). Positions are numbered from 1.

When working with a multi-byte character system (MBCS), IsPathDelimiter distinguishes between a true delimiter character, and the byte of the same value that can appear as the second byte of a double byte character.

To test for more characters than just the delimiter, use IsDelimiter (see page 4218).

See Also

IsDelimiter (see page 4218)

LastDelimiter (see page 4223)

37.3.160 SysUtils.IsValidIdent Function

Determines whether a string conforms to Delphi identifier rules.

Pascal

```
function IsValidIdent(const Ident: string; AllowDots: Boolean = False): Boolean;
```

C++

```
Boolean IsValidIdent(const UnicodeString Ident, Boolean AllowDots = False);
```

File

SysUtils

Description

Call IsValidIdent to determine whether the passed Ident is valid for use as an identifier, such as the name of a component.

An identifier is a sequence of letters, digits or underscores. Only the first 63 characters are used uniquely. Letters are limited to A-Z and a-z, although the case is ignored. The first character cannot be a digit.

37.3.161 SysUtils.Languages Function

Lists the locales for which support is available.

Pascal

```
function Languages: TLanguages;
```

C++

```
TLanguages Languages();
```

File

SysUtils

Description

Use the value returned by Languages to obtain information about the locales that the system supports. This function is only available on Windows.

See Also

TSysLocale (see page 4333)

37.3.162 SysUtils.LastDelimiter Function

Returns the byte index of the last character that matches any character in a specified set of delimiters.

Pascal

```
function LastDelimiter(const Delimiters: string; const S: string): Integer; overload;
```

C++

```
int LastDelimiter(const UnicodeString Delimiters, const UnicodeString S);
```

File

SysUtils

Description

Call LastDelimiter to locate the last delimiter in a specified string.

Delimiters is a string where each character is a valid delimiter.

S is the string to search for delimiters.

For example, the line

MyIndex := LastDelimiter('\.:','c:\filename.ext'); // Delphi

MyIndex = LastDelimiter("\\.:","c:\\filename.ext"); // C++

sets MyIndex to 12.

When working with multi-byte character sets (MBCS), S may contain double byte characters, but the delimiters listed in the Delimiters parameter must all be single byte non-null characters.

See Also

IsDelimiter (see page 4218)

AnsiStrRScan (see page 4060)

37.3.163 SysUtils.LoadPackage Function

Loads a specified package.

Pascal

```
function LoadPackage(const Name: string): HMODULE; overload;
function LoadPackage(const Name: string; AValidatePackage: TValidatePackageProc): HMODULE;
overload;
```

C++

```
HMODULE LoadPackage(const UnicodeString Name);
HMODULE LoadPackage(const UnicodeString Name, TValidatePackageProc AValidatePackage);
```

File

SysUtils

Description

LoadPackage loads the package specified by the Name parameter, checks for duplicate units, and calls the initialization blocks of all units contained in the package.

See Also

UnloadPackage (see page 4310)

37.3.164 SysUtils.LoadStr Function

Loads a string resource from the application's executable file.

Pascal

```
function LoadStr(Ident: Integer): string;
```

C++

```
UnicodeString LoadStr(int Ident);
```

File

SysUtils

Description

LoadStr loads the string resource given by Ident from the application's executable file or shared library. If the string resource does not exist, LoadStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

Note: :LoadStr is provided primarily for backward compatibility. To isolate string resources in Delphi, use the resourcestring keyword in new applications.

37.3.165 SysUtils.LowerCase Function

Converts an ASCII string to lowercase.

Pascal

```
function LowerCase(const S: string): string; overload;
function LowerCase(const S: string; LocaleOptions: TLocaleOptions): string; overload;
```

C++

```
UnicodeString LowerCase(const UnicodeString S);
UnicodeString LowerCase(const UnicodeString S, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

LowerCase returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'. To convert 8-bit international characters, use AnsiLowerCase (see page 4053).

C++ Examples:

```
/*
This example uses an edit control, a label, and a button on a
form. When the user clicks the button, the label shows the text of
the edit control in lower case.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = LowerCase(Edit1->Text);
}
/*
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    "System ParamStr = " + AnsiString(System::ParamStr(i)),
//    mtConfirmation, TMsgDlgButtons() << mbOK, 0);
  for (int i=1;i<=ParamCount();i++)
  {
    if (LowerCase(ParamStr(i)) == "beep")
      Beep();
    else if (LowerCase(ParamStr(i)) == "exit")
      Application->Terminate();
  }
}
```

Delphi Examples:

```
{
This example uses an edit control, a label, and a button on a
form. When the user clicks the button, the label shows the text of
the edit control in lower case.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.LowerCase(Edit1.Text);
end;
{
The following example beeps once for each "beep" passed in
on the command line.  The example terminates the application
if  "exit" is passed in on the command line.  Build the
project to generate a .exe file and then execute that file
from a command line: "ParamCount_proj beep beep exit".
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
//  Use print statements since you can't use the debugger.
//  MessageDlg(
//    'System ParamCount = ' + IntToStr(System.ParamCount),
//    mtConfirmation, [mbOK], 0);
  for i := 1 to System.ParamCount do
  begin
    if LowerCase(ParamStr(i)) = 'beep' then
      Beep
    else if LowerCase(ParamStr(i)) = 'exit' then
      Application.Terminate;
    Sleep(250);
  end;
end;
```

See Also

AnsiLowerCase (see page 4053)

UpperCase (see page 4311)

37.3.166 SysUtils.MSecsToTimeStamp Function

Converts a specified number of milliseconds into a TTimeStamp (see page 4334) value.

Pascal

```
function MSecsToTimeStamp(MSecs: Comp): TTimeStamp;
```

C++

```
TTimeStamp MSecsToTimeStamp(Comp MSecs);
```

File

SysUtils

Description

Call MSecsToTimeStamp to convert MSecs, a number of milliseconds, into the number of days represented, plus the number of milliseconds remaining. The days and milliseconds are expressed as a TTimeStamp (see page 4334) value, where the Date (see page 4087) field is the number of days and the Time (see page 4297) field is the remainder in milliseconds.

See Also

TimeStampToMSecs (see page 4299)

TTimeStamp (see page 4334)

37.3.167 SysUtils.NewStr Function

Allocates a string on the heap.

Pascal

```
function NewStr(const S: AnsiString): PAnsiString; deprecated;
```

C++

```
PAnsiString NewStr(const AnsiString S);
```

File

SysUtils

Description

NewStr allocates a new string on the heap. Do not use NewStr in new applications. This function is provided for backward compatibility only.

Use normal Delphi operators instead. The string allocation will be performed automatically.

37.3.168 SysUtils.NextCharIndex Function

Returns the byte index of the next character.

Pascal

```
function NextCharIndex(const S: UnicodeString; Index: Integer): Integer; overload;
function NextCharIndex(const S: AnsiString; Index: Integer): Integer; overload;
```

C++

```
int NextCharIndex(const UnicodeString S, int Index);
int NextCharIndex(const AnsiString S, int Index);
```

File

SysUtils

Description

NextCharIndex returns the byte index of the first byte of the character following the character starting at S[Index].

37.3.169 SysUtils.Now Function

Returns the current date and time.

Pascal

```
function Now: TDateTime;
```

C++

```
TDateTime Now();
```

File

SysUtils

Description

Returns the current date and time, corresponding to the sum of the value returned by the global Date (see page 4087) and Time (see page 4297) functions.

Note: Although TDateTime (see page 3323) values can represent milliseconds, Now is accurate only to the nearest second.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(Now());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date and time is
converted to a string and displayed as the caption of the
label:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.DateTimeToStr(Now);
end;
```

See Also

Date (see page 4087)

DateTimeToStr (see page 4089)

Time (see page 4297)

37.3.170 SysUtils.OutOfMemoryError Function

Raises an EOutOfMemory (see page 3942) exception.

Pascal

```
procedure OutOfMemoryError;
```

C++

```
OutOfMemoryError();
```

File

SysUtils

Description

Call OutOfMemoryError to raise the EOutOfMemory (see page 3942) exception. OutOfMemoryError runs without allocating additional memory.

See Also

EOutOfMemory (see page 3942)

37.3.171 SysUtils.PlatformBytesOf Function

Pascal

```
function PlatformBytesOf(const Value: string): TBytes;
```

C++

```
TBytes PlatformBytesOf(const UnicodeString Value);
```

File

SysUtils

Description

This is function SysUtils.PlatformBytesOf.

37.3.172 SysUtils.PlatformStringOf Function

Pascal

```
function PlatformStringOf(const Value: TBytes): UnicodeString;
```

C++

```
UnicodeString PlatformStringOf(const TBytes Value);
```

File

SysUtils

Description

This is function SysUtils.PlatformStringOf.

37.3.173 SysUtils.QuotedStr Function

Returns the quoted version of a string.

Pascal

```
function QuotedStr(const S: string): string; overload;
```

C++

```
UnicodeString QuotedStr(const UnicodeString S);
```

File

SysUtils

Description

Use QuotedStr to convert the string S to a quoted string. A single quote character (') is inserted at the beginning and end of S, and each single quote character in the string is repeated. To remove the quotes from a quoted string, use AnsiDequotedStr (see page 4051) routine.

Note: When working with multi-byte character systems (MBCS), use AnsiQuotedStr (see page 4054) instead.

See Also

QuotedStr

AnsiQuotedStr (see page 4054)

AnsiExtractQuotedStr (see page 4052)

37.3.174 SysUtils.RaiseLastOSError Function

Raises an exception for the last occurring OS or system library error.

Pascal

```
procedure RaiseLastOSError; overload;
procedure RaiseLastOSError(LastError: Integer); overload;
```

C++

```
RaiseLastOSError();
RaiseLastOSError(int LastError);
```

File

SysUtils

Description

Call RaiseLastOSError to raise an EOSError (see page 3939) exception for the last OS API call that failed. RaiseLastOSError retrieves the code for the last occurring API call error, if any. If this belongs to a previously occurring error, RaiseLastOSError raises an EOSError (see page 3939) exception with the error code and message associated with that error.

C++ Examples:

```
//
//This code demonstrates the use of system error codes
//and how to use transform them into C++ exceptions.
//
void __fastcall TForm2::btRaiseLastClick(TObject *Sender)
{
    /* Set the last OS error to a bogus value */
    System::SetLastError(ERROR_ACCESS_DENIED);

    try
    {
        RaiseLastOSError();
    }
    catch(EOSError* ex)
    {
        MessageDlg(String("Caught an OS error with code: ") +
            IntToStr((int)ex->ErrorCode), mtError,
            TMsgDlgButtons() << mbOK, 0);
```

```
    }


    /* Let the Delphi's Exception dialog pop-up */
    RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

    /* Finally set the last error to none */
    System::SetLastError(ERROR_SUCCESS);

    if (GetLastError() != ERROR_SUCCESS)
        MessageDlg(String("Whoops, something went wrong ") +
          " in the mean time!", mtError, TMsgDlgButtons() << mbOK, 0);

    /* No exception should be thrown here */
    RaiseLastOSError();
}
```

Delphi Examples:

```
{
This code demonstrates the use of system error codes
and how to transform them into Delphi exceptions.
}
procedure TForm2.btRaiseLastClick(Sender: TObject);
begin
  { Set the last OS error to a bogus value }
  System.SetLastError(ERROR_ACCESS_DENIED);

  try
    RaiseLastOSError();
  except
    on Ex : EOSError do
       MessageDlg('Caught an OS error with code: ' +
          IntToStr(Ex.ErrorCode), mtError, [mbOK], 0);
  end;

  { Let the Delphi's Exception dialog pop-up }
  RaiseLastOSError(ERROR_NOT_ENOUGH_MEMORY);

  { Finally set the last error to none }
  System.SetLastError(ERROR_SUCCESS);

  if GetLastError() <> ERROR_SUCCESS then
     MessageDlg('Whoops, something went wrong in the ' +
                'mean time!', mtError, [mbOK], 0);

  { No exception should be thrown here }
  RaiseLastOSError();
end;
```

See Also

EOSError (see page 3939)

37.3.175 SysUtils.RaiseLastWin32Error Function

Raises an exception for the last occurring Win32 error.

Pascal

```
procedure RaiseLastWin32Error; deprecated;
```

C++

```
RaiseLastWin32Error();
```

File

SysUtils

Description

Call RaiseLastWin32Error to raise an EWin32Error (see page 3949) exception for the last Windows API call that failed. RaiseLastWin32Error retrieves the code for the last occurring Win32 error, if any. If Windows returns a previously occurring error code, RaiseLastWin32Error raises an EWin32Error (see page 3949) exception with the error code and message associated with that error.

Note: RaiseLastWin32Error is a deprecated Windows-only procedure. New (see page 3619) applications should use RaiseLastOSError (see page 4230) instead.

See Also

EWin32Error (see page 3949)

Win32Check (see page 4318)

RaiseLastOSError (see page 4230)

37.3.176 SysUtils.RCS Function

Check for Delphi Client/Server Licence.

Pascal

```
procedure RCS;
```

C++

```
RCS();
```

File

SysUtils

Description

RCS is used by VCL components to check whether there is a Client/Server licence, raising an exception of not.

See Also

GDAL (see page 4202)

RPR (see page 4237)

37.3.177 SysUtils.RemoveDir Function

Deletes an existing empty directory.

Pascal

```
function RemoveDir(const Dir: string): Boolean;
```

C++

```
Boolean RemoveDir(const UnicodeString Dir);
```

File

SysUtils

Description

Call RemoveDir to remove the directory specified by the Dir parameter. The return value is true if a new directory was successfully deleted, false if an error occurred. The directory must be empty before it can be successfully deleted.

See Also

ExtractFileDir (see page 4112)

GetCurrentDir (see page 4202)

SetCurrentDir (see page 4240)

37.3.178 SysUtils.RenameFile Function

Changes a file name.

Pascal

```
function RenameFile(const OldName: string; const NewName: string): Boolean;
```

C++

```
Boolean RenameFile(const UnicodeString OldName, const UnicodeString NewName);
```

File

SysUtils

Description

RenameFile attempts to change the name of the file specified by OldFile to NewFile. If the operation succeeds, RenameFile returns true. If RenameFile cannot rename the file (for example, if the application does not have permission to modify the file), it returns false.

Note: If NewFile represents the name of an existing file, RenameFile behaves differently under Windows and Linux. Under Windows, the rename operation fails. Under Linux, RenameFile silently removes the other file.

C++ Examples:

```
/*
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function. Verify
the file written after executing.
*/

#include <dir.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  char szFileName[MAXFILE+4];
  int iFileHandle;
  int iLength;
  if (SaveDialog1->Execute())
  {
    if (FileExists(SaveDialog1->FileName))
    {
      fnsplit(AnsiString(SaveDialog1->FileName).c_str(), 0, 0, szFileName, 0);
      strcat(szFileName, ".BAK");
      RenameFile(SaveDialog1->FileName, szFileName);
    }
    iFileHandle = FileCreate(SaveDialog1->FileName);
    // Write out the number of rows and columns in the grid.
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->ColCount)),
sizeof(StringGrid1->ColCount));
    FileWrite(iFileHandle, reinterpret_cast<char *>(&(StringGrid1->RowCount)),
sizeof(StringGrid1->RowCount));
    for (int x=0;x<StringGrid1->ColCount;x++)
    {
      for (int y=0;y<StringGrid1->RowCount;y++)
      {
        // Write out the length of each string, followed by the string itself.
        iLength = StringGrid1->Cells[x][y].Length();
        FileWrite(iFileHandle, reinterpret_cast<char *>(&iLength), sizeof(iLength));
        FileWrite(iFileHandle, StringGrid1->Cells[x][y].t_str(),
StringGrid1->Cells[x][y].Length());
      }
    }
    FileClose(iFileHandle);
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "Column 1";
  StringGrid1->Cells[2][0] = "Column 2";
  StringGrid1->Cells[3][0] = "Column 3";
  StringGrid1->Cells[4][0] = "Column 4";
  StringGrid1->Cells[0][1] = "Row 1";
  StringGrid1->Cells[1][1] = "Object";
  StringGrid1->Cells[2][1] = "Pascal";
  StringGrid1->Cells[3][1] = "is";
  StringGrid1->Cells[4][1] = "excellent";
  StringGrid1->Cells[0][2] = "Row 2";
  StringGrid1->Cells[1][2] = "Delphi";
  StringGrid1->Cells[2][2] = "is";
  StringGrid1->Cells[4][2] = "RAD";
};
```

Delphi Examples:

```
{
The following example uses a button, a string grid, and a
Save dialog box on a form. When the button is clicked, the
user is prompted for a filename. When the user clicks OK,
the contents of the string grid are written to the specified
file. Additional information is also written to the file so
that it can be read easily with the FileRead function.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  BackupName: string;
  FileHandle: Integer;
  StringLen: Integer;
  X, Y, I: Integer;
  colCountLength, rowCountLength: Integer;
  Buffer: PChar;
```

```
    cellString: string;
  begin
    if SaveDialog1.Execute then
    begin
      if FileExists(SaveDialog1.FileName) then
      begin
        BackupName := SysUtils.ExtractFileName(SaveDialog1.FileName);
        BackupName := ChangeFileExt(BackupName, '.BAK');
        if not RenameFile(SaveDialog1.FileName, BackupName) then
          raise Exception.Create('Unable to create backup file.');
      end;
      FileHandle := FileCreate(SaveDialog1.FileName);
      { Write out the number of rows and columns in the grid. }
      colCountLength := SizeOf(StringGrid1.ColCount);
      FileWrite(FileHandle,
        Pchar(StringGrid1.ColCount), colCountLength);
      rowCountLength := SizeOf(StringGrid1.RowCount);
      FileWrite(FileHandle,
        PChar(StringGrid1.RowCount), rowCountLength);
      for X := 0 to StringGrid1.ColCount - 1 do
      begin
        for Y := 0 to StringGrid1.RowCount - 1 do
        begin
          try
            { Write out the length of each string, followed by the string itself. }
            StringLen := Length(StringGrid1.Cells[X,Y]);
            FileWrite(FileHandle, PChar(StringLen), SizeOf(StringLen));
            GetMem(Buffer, StringLen); { allocate the buffer }
            cellString := StringGrid1.Cells[X,Y];
            for I := 1 to StringLen do
               Buffer[I - 1] := cellString[I];
            FileWrite(FileHandle, Buffer^, StringLen);
          finally
            FreeMem(Buffer, StringLen);
          end;
        end;
      end;
      FileClose(FileHandle);
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    with StringGrid1 do
    begin
      Cells[1,0] := 'Column 1';
      Cells[2,0] := 'Column 2';
      Cells[3,0] := 'Column 3';
      Cells[4,0] := 'Column 4';
      Cells[0,1] := 'Row 1';
      Cells[1,1] := 'Object';
      Cells[2,1] := 'Pascal';
      Cells[3,1] := 'is';
      Cells[4,1] := 'excellent';
      Cells[0,2] := 'Row 2';
      Cells[1,2] := 'Delphi';
      Cells[2,2] := 'is';
      Cells[4,2] := 'RAD';
    end;
  end;
```

See Also

DeleteFile (see page 4098)

Rename (see page 3642)

37.3.179 SysUtils.ReplaceDate Function

Replaces the date portion of a TDateTime (see page 3323) value with a specified date.

Pascal

```
procedure ReplaceDate(var DateTime: TDateTime; const NewDate: TDateTime);
```

C++

```
ReplaceDate(TDateTime DateTime, const TDateTime NewDate);
```

File

SysUtils

Description

ReplaceDate changes the value of the DateTime parameter so that it has the same date as the NewDate parameter. The time portion of DateTime remains the same. ReplaceDate makes any necessary adjustments for negative dates (before 1900).

See Also

DecodeDate (see page 4095)

EncodeDate (see page 4105)

ReplaceTime (see page 4236)

IncMonth (see page 4214)

RecodeDay

RecodeHour

RecodeMilliSecond

RecodeMinute

RecodeMonth

RecodeSecond

RecodeYear

RecodeDate

37.3.180 SysUtils.ReplaceTime Function

Replaces the time portion of a TDateTime (see page 3323) value with a specified time.

Pascal

```
procedure ReplaceTime(var DateTime: TDateTime; const NewTime: TDateTime);
```

C++

```
ReplaceTime(TDateTime DateTime, const TDateTime NewTime);
```

File

SysUtils

Description

ReplaceTime changes the value of the DateTime parameter so that it has the same time as the NewTime parameter. The date portion of DateTime remains the same. ReplaceTime makes any necessary adjustments for negative dates. (before 1900)

See Also

DecodeTime (see page 4097)

EncodeTime (see page 4106)

ReplaceDate (see page 4236)

RecodeDay

RecodeHour

RecodeMilliSecond

RecodeMinute

RecodeMonth

RecodeSecond

RecodeYear

StartOfADay

EndOfADay

RecodeTime

37.3.181 SysUtils.RPR Function

Check for Delphi Professional Licence.

Pascal

```
procedure RPR;
```

C++

```
RPR();
```

File

SysUtils

Description

RPR is used by VCL components to check whether there is a Professional licence, raising an exception of not.

See Also

GDAL (see page 4202)

RCS (see page 4232)

37.3.182 SysUtils.SafeLoadLibrary Function

Safe and simple library loading.

Pascal

```
function SafeLoadLibrary(const FileName: string; ErrorMode: UINT = SEM_NOOPENFILEERRORBOX):
HMODULE;
```

C++

```
HMODULE SafeLoadLibrary(const UnicodeString FileName, UINT ErrorMode = SEM_NOOPENFILEERRORBOX);
```

File

SysUtils

Description

SafeLoadLibrary loads a Windows DLL or Linux shared object file, as specified by Filename. SafeLoadLibrary preserves the current FPU control word, preventing library initialization code from permanently overwriting precision and exception masks.

Note: On Windows, SafeLoadLibrary temporarily sets the system error mode to ErrorMode. The default, SEM_NOOPENFILEERRORBOX, suppresses error dialogs. The previous error mode is restored before SafeLoadLibrary exits. For a list of error modes, refer to SetErrorMode in the Microsoft documentation.

Note: On Linux, the Dummy argument is ignored.

See Also

LoadPackage (see page 4224)

37.3.183 SysUtils.SameFileName Function

Compares file names based on the current locale.

Pascal

```
function SameFileName(const S1: string; const S2: string): Boolean;
```

C++

```
Boolean SameFileName(const UnicodeString S1, const UnicodeString S2);
```

File

SysUtils

Description

SameFileName compares two file names in the same way as AnsiCompareFileName (see page 4049). SameFileName returns true if the file names match, false otherwise. Like AnsiCompareFileName (see page 4049), SameFileName supports locales and MBCS strings; see the AnsiCompareFileName (see page 4049) topic for more details.

See Also

AnsiCompareText (see page 4051)

37.3.184 SysUtils.SameStr Function

Compares two strings by ordinal value with case sensitivity.

Pascal

```
function SameStr(const S1: string; const S2: string): Boolean; overload;
function SameStr(const S1: string; const S2: string; LocaleOptions: TLocaleOptions): Boolean;
overload;
```

C++

```
Boolean SameStr(const UnicodeString S1, const UnicodeString S2);
Boolean SameStr(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

SameStr compares S1 and S2 and returns true if they are identical. SameStr is case sensitive and is not affected by the current locale.

See Also

SameText (see page 4239)

CompareStr (see page 4078)

CompareText (see page 4079)

AnsiCompareStr (see page 4050)

AnsiCompareText (see page 4051)

CompareMem (see page 4078)

37.3.185 SysUtils.SameText Function

Compares two strings by ordinal value without case sensitivity.

Pascal

```
function SameText(const S1: string; const S2: string): Boolean; overload;
function SameText(const S1: string; const S2: string; LocaleOptions: TLocaleOptions): Boolean;
overload;
```

C++

```
Boolean SameText(const UnicodeString S1, const UnicodeString S2);
Boolean SameText(const UnicodeString S1, const UnicodeString S2, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

SameText compares S1 and S2 and returns true if they are equal. SameText is not case sensitive and is not affected by the current locale.

See Also

CompareStr (see page 4078)

AnsiCompareText (see page 4051)

CompareMem (see page 4078)

CompareText (see page 4079)

37.3.186 SysUtils.SetCurrentDir Function

Sets the current directory.

Pascal

```
function SetCurrentDir(const Dir: string): Boolean;
```

C++

```
Boolean SetCurrentDir(const UnicodeString Dir);
```

File

SysUtils

Description

The SetCurrentDir function sets the current directory. The return value is true if the current directory was successfully changed, or false if an error occurred.

See Also

ChDir (see page 3534)

CreateDir (see page 4083)

ExtractFileDir (see page 4112)

GetCurrentDir (see page 4202)

RemoveDir (see page 4232)

37.3.187 SysUtils.ShowException Function

Displays an exception message with its physical address

Pascal

```
procedure ShowException(ExceptObject: TObject; ExceptAddr: Pointer);
```

C++

```
ShowException(TObject * ExceptObject, void * ExceptAddr);
```

File

SysUtils

Description

ShowException displays the message associated with an exception, together with the exception's physical address. The

exception's address can be used with Search | Find Error (see page 3560) menu command to find the statement that raised the exception.

See Also

ExceptionErrorMessage (see page 4107)

ExceptObject (see page 3562)

ExceptAddr (see page 3562)

37.3.188 SysUtils.Sleep Function

Delay program execution for a specified number of milliseconds.

Pascal

```
procedure Sleep(milliseconds: Cardinal); stdcall;
```

C++

```
__stdcall Sleep(unsigned milliseconds);
```

File

SysUtils

Description

Sleep pauses program executions as specified by the milliseconds parameter. Under Windows, Sleep is just a link to the Sleep function in the system API.

37.3.189 SysUtils.StrAlloc Function

Allocates a buffer for a null-terminated string and returns a pointer to its first character.

Pascal

```
function StrAlloc(Size: Cardinal): PChar;
```

C++

```
const char * StrAlloc(unsigned Size);
```

File

SysUtils

Description

StrAlloc allocates a buffer for a null-terminated string with a maximum length of Size - 1 (1 byte must be reserved for the termination character). The result points to the location where the first character of the string is to be stored. A 32-bit number giving the total amount of memory allocated is stored in the four bytes preceding the first character; it is equal to Size + 4. If space for a string is allocated with StrAlloc, it should be deallocated via StrDispose (see page 4247).

Because AnsiStrings (long strings) are implicitly null terminated and dynamically allocated, the use of StrAlloc is deprecated.

See Also

StrDispose (see page 4247)

StrNew (see page 4266)

StrBufSize (see page 4242)

37.3.190 SysUtils.StrBufSize Function

Returns the maximum number of characters that can be stored in a buffer allocated by StrAlloc (see page 4241).

Pascal

```
function StrBufSize(const Str: PAnsiChar): Cardinal; overload;
function StrBufSize(const Str: PWideChar): Cardinal; overload;
```

C++

```
unsigned StrBufSize(const PAnsiChar Str);
unsigned StrBufSize(const PWideChar Str);
```

File

SysUtils

Description

StrBufSize returns the number of bytes in Str (see page 3676), including the byte for the termination character.

Note: If Str (see page 3676) does not point to a string buffer allocated by StrAlloc (see page 4241) or StrNew (see page 4266), no error message is returned, and the result is unpredictable.

See Also

StrAlloc (see page 4241)

StrNew (see page 4266)

37.3.191 SysUtils.StrByteType Function

Indicates whether a byte in a string is a single byte character, the first byte of a double byte character, or the second byte of a double byte character.

Pascal

```
function StrByteType(Str: PAnsiChar; Index: Cardinal): TMbcsByteType; overload;
function StrByteType(Str: PWideChar; Index: Cardinal): TMbcsByteType; overload;
```

C++

```
TMbcsByteType StrByteType(PAnsiChar Str, unsigned Index);
TMbcsByteType StrByteType(PWideChar Str, unsigned Index);
```

File

SysUtils

Description

Call StrByteType to determine the type of the byte specified by the Index parameter, where 0 specifies the first byte in the string Str (see page 3676), 1 specifies the second byte, and so on.

If the system is not using a multi-byte character system (MBCS), StrByteType always returns mbSingleByte. Otherwise, StrByteType returns mbSingleByte if the indicated byte represents a complete character in Str (see page 3676), mbLeadByte if

it represents the first byte of a double byte character, and mbTrailByte if it represents the second byte of a double byte character.

Note: No checking is done to ensure that Index is less than the length of Str (see page 3676). It is the caller's responsibility to ensure that Index is not out of bounds.

See Also

ByteType (see page 4070)

LeadBytes (see page 4354)

37.3.192 SysUtils.StrCat Function

Appends a copy of Source to the end of Dest and returns the concatenated string.

Pascal

```
function StrCat(Dest: PAnsiChar; const Source: PAnsiChar): PAnsiChar; overload;
function StrCat(Dest: PWideChar; const Source: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrCat(PAnsiChar Dest, const PAnsiChar Source);
PWideChar StrCat(PWideChar Dest, const PWideChar Source);
```

File

SysUtils

Description

Use StrCat to concatenate Source to the end of Dest. StrCat does not perform any length checking. The destination buffer must have room for at least StrLen (see page 4260)(Dest)+StrLen (see page 4260)(Source)+1 characters.

To check length, use the StrLCat (see page 4255) function.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szBuffer(new char[Label1->Caption.Length() + Edit1->Text.Length() + 1]);
  // The following 2 lines are used in VCL applications
  StrCopy(szBuffer.get(), AnsiString(Label1->Caption).c_str());
  StrCat(szBuffer.get(), AnsiString(Edit1->Text).c_str());
  Label1->Caption = szBuffer.get();
  Edit1->Clear();
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: PChar;
begin
  GetMem(Buffer,Length(Label1.Caption) + Length(Edit1.Text) + 1);
  StrCopy(Buffer, PChar(Label1.Caption));
  SysUtils.StrCat(Buffer, PChar(Edit1.Text));
  Label1.Caption := Buffer;
  Edit1.Clear;
  FreeMem(Buffer);
end;
```

See Also

StrLCat (see page 4255)

StrLen (see page 4260)

37.3.193 SysUtils.StrCharLength Function

Returns the length, in bytes, of a character.

Pascal

```
function StrCharLength(const Str: PAnsiChar): Integer; overload;
function StrCharLength(const Str: PWideChar): Integer; overload;
```

C++

```
int StrCharLength(const PAnsiChar Str);
int StrCharLength(const PWideChar Str);
```

File

SysUtils

Description

StrCharLength returns the size, in bytes, of the first character in Str (see page 3676). This value can be 1 or 2 under Windows, 1 to 6 under Linux.

See Also

ByteType (see page 4070)

LeadBytes (see page 4354)

37.3.194 SysUtils.StrComp Function

Compares two strings with case sensitivity.

Pascal

```
function StrComp(const Str1: PAnsiChar; const Str2: PAnsiChar): Integer; overload;
function StrComp(const Str1: PWideChar; const Str2: PWideChar): Integer; overload;
```

C++

```
int StrComp(const PAnsiChar Str1, const PAnsiChar Str2);
int StrComp(const PWideChar Str1, const PWideChar Str2);
```

File

SysUtils

Description

Call StrComp to compare two null-terminated strings, with case sensitivity. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1is the same as Str2
>0	Str1 sorts after Str2

Note: When working with International characters, use AnsiStrComp (see page 4056) instead.

C++ Examples:

```
/*
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 20]);
  int iResult = StrComp(Edit1->Text.t_str(), Edit2->Text.t_str());
  StrCopy(szResult.get(), Edit1->Text.t_str());
  if (iResult < 0)
    StrCat(szResult.get(), " is less than ");
  else if (iResult > 0)
    StrCat(szResult.get(), " is greater than ");
  else
    StrCat(szResult.get(), " is equal to ");

  StrCat(szResult.get(), AnsiString(Edit2->Text).c_str());
  ShowMessage(AnsiString(szResult.get()));
}
```

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Msg: string;
  CompResult: Integer;
begin
  Msg := Edit1.Text;
  CompResult := SysUtils.StrComp(PChar(Edit1.Text), PChar(Edit2.Text));
  if CompResult < 0 then
    Msg := Msg + ' is less than '
  else if CompResult > 0 then
    Msg := Msg + ' is greater than '
  else
    Msg := Msg + ' is equal to ';
  Msg := Msg + Edit2.Text;
```

```
  ShowMessage(Msg);
end;
```

See Also

AnsiStrComp (see page 4056)

StrIComp (see page 4251)

StrLComp (see page 4257)

StrLIComp (see page 4262)

37.3.195 SysUtils.StrCopy Function

Copies a null-terminated string.

Pascal

```
function StrCopy(Dest: PAnsiChar; const Source: PAnsiChar): PAnsiChar; overload;
function StrCopy(Dest: PWideChar; const Source: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrCopy(PAnsiChar Dest, const PAnsiChar Source);
PWideChar StrCopy(PWideChar Dest, const PWideChar Source);
```

File

SysUtils

Description

Use StrCopy to copy Source to Dest. StrCopy returns Dest.

StrCpy does not perform any length checking. The destination buffer must have room for at least StrLen (see page 4260)(Source)+1 characters.

For length checking, use the StrLCopy (see page 4258) function.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a button
on a form. When the button is clicked, the label's caption and the
edit control's text are combined in a buffer. Then the buffer is
displayed in the label's caption.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szBuffer(new char[Label1->Caption.Length() + Edit1->Text.Length() + 1]);
  // The following 2 lines are used in VCL applications
  StrCopy(szBuffer.get(), AnsiString(Label1->Caption).c_str());
  StrCat(szBuffer.get(), AnsiString(Edit1->Text).c_str());
  Label1->Caption = szBuffer.get();
  Edit1->Clear();
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a button
```

```
  on a form. When the button is clicked, the label's caption and the
  edit control's text are combined in a buffer. Then the buffer is
  displayed in the label's caption.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Buffer: PChar;
  begin
    GetMem(Buffer,Length(Label1.Caption) + Length(Edit1.Text) + 1);
    StrCopy(Buffer, PChar(Label1.Caption));
    SysUtils.StrCat(Buffer, PChar(Edit1.Text));
    Label1.Caption := Buffer;
    Edit1.Clear;
    FreeMem(Buffer);
  end;
```

See Also

StrECopy (see page 4248)

StrLCopy (see page 4258)

37.3.196 SysUtils.StrDispose Function

Disposes of a string.

Pascal

```
procedure StrDispose(Str: PAnsiChar); overload;
procedure StrDispose(Str: PWideChar); overload;
```

C++

```
StrDispose(PAnsiChar Str);
StrDispose(PWideChar Str);
```

File

SysUtils

Description

StrDispose is provided for backward compatibility. StrDispose disposes of a string on a heap that was previously allocated with StrAlloc (see page 4241) or StrNew (see page 4266).

If Str (see page 3676) is nil (Delphi) or NULL (C++), StrDispose does nothing.

C++ Examples:

```
/*
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Allocate memory.
  char* psz = StrNew(Edit1->Text.t_str());
  Application->MessageBox(
    Edit1->Text.c_str(),
    L"StrNew, StrDispose example",
    MB_OK);
  // Deallocate memory.
  StrDispose(psz);
```

```
}
```

Delphi Examples:

```
{
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Temp: PChar;
begin
  // Allocate memory.
  Temp := StrNew(PChar(Edit1.Text));
  Application.MessageBox(Temp, 'StrNew, StrDispose example', MB_OK);
  // Deallocate memory.
  SysUtils.StrDispose(Temp);
end;
```

See Also

StrAlloc (see page 4241)

StrNew (see page 4266)

37.3.197 SysUtils.StrECopy Function

Copies null-terminated string.

Pascal

```
function StrECopy(Dest: PAnsiChar; const Source: PAnsiChar): PAnsiChar; overload;
function StrECopy(Dest: PWideChar; const Source: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrECopy(PAnsiChar Dest, const PAnsiChar Source);
PWideChar StrECopy(PWideChar Dest, const PWideChar Source);
```

File

SysUtils

Description

StrECopy copies Source to Dest and returns StrEnd (see page 4249)(Dest).

StrECopy does not perform any length checking. The destination buffer must have room for at least StrLen (see page 4260)(Source)+1 characters.

Nested calls to StrECopy to concatenate a sequence of strings run more efficiently than multiple calls to StrCat (see page 4243).

C++ Examples:

```
/*
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the edit
control's text is appended to a static buffer. Then the
buffer is displayed in the label's caption.
*/
const static int MAX_LENGTH = 15;
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  static char szBuffer[MAX_LENGTH];
  static char* pszBuffer = szBuffer;

  if (StrLen(pszBuffer) + StrLen(Edit1->Text.c_str()) < MAX_LENGTH)
    pszBuffer = StrECopy(pszBuffer, Edit1->Text.t_str());
  else
    ShowMessage("Can't fit new string in allocated memory space.");
  Label1->Caption = szBuffer;
}
```

Delphi Examples:

```
{
This example uses a button on a form.  Since StrECopy returns
the destination char array, nested calls are possible and
actually more efficient than repeated calls.  StrECopy does
no bounds checking. When the button is clicked, the inline
text is appended to a static buffer. Then the buffer is
displayed in the form's canvas.
}

procedure TForm1.Button1Click(Sender: TObject);
const
  Turbo: PChar = 'Object';
  Pascal: PChar = 'Pascal';
 var
  S: array[0..15] of Char;
begin
  SysUtils.StrECopy(SysUtils.StrECopy(SysUtils.StrECopy(S, Turbo), ' '), Pascal);
  Canvas.TextOut(10, 10, string(S));
end;
```

See Also

StrCat (see page 4243)

StrCopy (see page 4246)

StrEnd (see page 4249)

StrLen (see page 4260)

37.3.198 SysUtils.StrEnd Function

Returns a pointer to the end of a null terminated string.

Pascal

```
function StrEnd(const Str: PAnsiChar): PAnsiChar; overload;
function StrEnd(const Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrEnd(const PAnsiChar Str);
PWideChar StrEnd(const PWideChar Str);
```

File

SysUtils

Description

StrEnd returns a pointer to the null character at the end of Str (see page 3676).

C++ Examples:

```
/*
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the edit
control's text is copied into a buffer. Then the buffer is
displayed backwards in the label.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> pszEditText(new char[Edit1->Text.Length()+1]);
  char *psz;
  StrCopy(pszEditText.get(), Edit1->Text.t_str());
  psz = StrEnd(pszEditText.get());
  Label1->Caption = "";
  while (psz >= &pszEditText.get()[0])
  {
    psz--;
    Label1->Caption = Label1->Caption + *psz;
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the edit
control's text is copied into a buffer. Then the buffer is
displayed backwards in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  TextBuffer: PChar;
  Ptr: PChar;
begin
  GetMem(TextBuffer, Length(Edit1.Text)+1);
  StrCopy(TextBuffer, PChar(Edit1.Text));
  Ptr := StrEnd(TextBuffer);
  Label1.Caption := '';
  while Ptr > TextBuffer do
  begin
    Dec(Ptr);
    Label1.Caption := Label1.Caption + Ptr^;
  end;
  FreeMem(TextBuffer);
end;
```

See Also

AnsiStrLastChar (see page 4057)

StrLen (see page 4260)

37.3.199 SysUtils.StrFmt Function

Formats entries in an array.

Pascal

```
function StrFmt(Buffer: PAnsiChar; Format: PAnsiChar; const Args: array of const): PAnsiChar;
overload;
```

```
function StrFmt(Buffer: PAnsiChar; Format: PAnsiChar; const Args: array of const; const
FormatSettings: TFormatSettings): PAnsiChar; overload;
function StrFmt(Buffer: PWideChar; Format: PWideChar; const Args: array of const): PWideChar;
overload;
function StrFmt(Buffer: PWideChar; Format: PWideChar; const Args: array of const; const
FormatSettings: TFormatSettings): PWideChar; overload;
```

C++

```
PAnsiChar StrFmt(PAnsiChar Buffer, PAnsiChar Format, const array of const Args);
PAnsiChar StrFmt(PAnsiChar Buffer, PAnsiChar Format, const array of const Args, const
TFormatSettings FormatSettings);
PWideChar StrFmt(PWideChar Buffer, PWideChar Format, const array of const Args);
PWideChar StrFmt(PWideChar Buffer, PWideChar Format, const array of const Args, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

StrFmt formats the series of arguments from an open array into a buffer.

Buffer is a buffer that receives the results.

Format (see page 4188) is a null-terminated format string that indicates how to generate the results. For more information about how the result is formatted, see Format (see page 4188) Strings.

Args is an array of arguments that replace the format specifiers in Format (see page 4188).

Note: Args_Size gives the index of the last argument (one less than the number of elements in Args).

StrFmt returns a pointer to the destination buffer.

The first form of StrFmt is not thread-safe, because it uses localization information contained in global variables. The second form of StrFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

Format (see page 4188)

GetLocaleFormatSettings (see page 4206)

37.3.200 SysUtils.StrIComp Function

Compares two strings without case sensitivity.

Pascal

```
function StrIComp(const Str1: PAnsiChar; const Str2: PAnsiChar): Integer; overload;
function StrIComp(const Str1: PWideChar; const Str2: PWideChar): Integer; overload;
```

C++

```
int StrIComp(const PAnsiChar Str1, const PAnsiChar Str2);
int StrIComp(const PWideChar Str1, const PWideChar Str2);
```

File

SysUtils

Description

Call StrIComp to compare two strings without case sensitivity. StrIComp returns a value greater than 0 if Str1 > Str2, less than 0 if Str1 < Str2, and returns 0 if the strings are equal except for differences in case.

When working with international characters, use AnsiStrIComp (see page 4057) instead.

C++ Examples:

```
/*
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared case-insensitively.
*/
#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 20]);
  int iResult = StrIComp(Edit1->Text.c_str(), Edit2->Text.c_str());

  StrCopy(szResult.get(), Edit1->Text.t_str());
  if (iResult < 0)
    StrCat(szResult.get(), " is less than ");
  else if (iResult > 0)
    StrCat(szResult.get(), " is greater than ");
  else
    StrCat(szResult.get(), " is equal to ");
  StrCat(szResult.get(), Edit2->Text.t_str());
  ShowMessage(AnsiString(szResult.get()));
}
```

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the edit
controls is compared case-insensitively.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Msg: string;
  CompResult: Integer;
begin
  Msg := Edit1.Text;
  CompResult := StrIComp(PChar(Edit1.Text), PChar(Edit2.Text));
  if CompResult < 0 then
    Msg := Msg + ' is less than '
  else if CompResult > 0 then
    Msg := Msg + ' is greater than '
  else
    Msg := Msg + ' is equal to ';
  Msg := Msg + Edit2.Text;
  ShowMessage(Msg);
end;
```

See Also

AnsiStrIComp (see page 4057)

StrComp (see page 4244)

StrLComp (see page 4257)

StrLIComp (see page 4262)

37.3.201 SysUtils.StringOf Function

Pascal

```
function StringOf(const Bytes: TBytes): UnicodeString;
```

C++

```
UnicodeString StringOf(const TBytes Bytes);
```

File

SysUtils

Description

This is function SysUtils.StringOf.

37.3.202 SysUtils.StringReplace Function

Replaces occurrences of a substring within a string.

Pascal

```
function StringReplace(const S: string; const OldPattern: string; const NewPattern: string;
Flags: TReplaceFlags): string; overload;
```

C++

```
UnicodeString StringReplace(const UnicodeString S, const UnicodeString OldPattern, const
UnicodeString NewPattern, TReplaceFlags Flags);
```

File

SysUtils

Description

StringReplace replaces occurrences of the substring specified by `OldPattern` with the substring specified by `NewPattern` in the string S.

`Flags` is a TReplaceFlags (see page 4343) type parameter. If `rfIgnoreCase` is set, the replacement is case-sensitive; otherwise case is ignored. If `rfReplaceAll` is on, all occurrences of `OldPattern` are replaced; if not, only the first occurrence is replaced.

Note: The parameters `S`, `OldPattern`, `NewPattern`, and the return value are of type `UnicodeString` (see page 3381). To do the replacement in an `AnsiString` context, use the StringReplace function. Also, to do the replacement in a `WideString` (see page 3467) context, use the WideStringReplace function.

Note: Recursive replacement of substrings is not supported. This means that if the substitution of `OldPattern` results in a new match for `NewPattern`, that match is not replaced.

To replace all occurrences of the substring within the string, you may also use the ReplaceStr function to do a case-sensitive search, or ReplaceText to do a case-insensitive search.

See Also

TReplaceFlags (see page 4343)

StringReplace

WideStringReplace (see page 6880)

ReplaceStr (see page 3044)

ReplaceText (see page 3045)

PosEx (see page 3043)

StuffString (see page 3053)

37.3.203 SysUtils.StringToGUID Function

Converts a string to a GUID.

Pascal

```
function StringToGUID(const S: string): TGUID;
```

C++

```
TGUID StringToGUID(const UnicodeString S);
```

File

SysUtils

Description

StringToGUID converts the string representation of a GUID, S, into an equivalent TGUID (see page 3718) data structure.

If StringToGUID succeeds it returns the requested GUID. If StringToGUID fails an EConvertError (see page 3919) exception is raised.

C++ Examples:

```
//
//This example demonstrates the usage of some GUID related routines along with
//the type itself.
//
void __fastcall TForm2::FormCreate(TObject *Sender)
{
    TGUID myGuid0, myGuid1;

    /* Create a new GUID from the string representation */
    myGuid0 = StringToGUID("{00020400-0000-0000-C000-000000000046}");
    Memo1->Lines->Add("The GUID is: " + GUIDToString(myGuid0));

    /*
    Accessing GUID's internal fields. Using Format function to
    obtain the same output as GUIDToString
    */
    Memo1->Lines->Add(Format(String("GUID using formatting is: ") +
              "{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}",
         ARRAYOFCONST((myGuid0.Data1, myGuid0.Data2, myGuid0.Data3,
         myGuid0.Data4[0], myGuid0.Data4[1], myGuid0.Data4[2], myGuid0.Data4[3],
         myGuid0.Data4[4], myGuid0.Data4[5], myGuid0.Data4[6], myGuid0.Data4[7]))));

    /* Auto-generate a random GUID at runtime */
    if (CreateGUID(myGuid1) != 0)
        Memo1->Lines->Add("Creating GUID failed!");
    else
        Memo1->Lines->Add("The generated guid is: " + GUIDToString(myGuid1));
```

```
    /* Generating second random GUID */
    CreateGUID(myGuid0);

    /* Testing if 2 guids are equal */
    if (IsEqualGUID(myGuid0, myGuid1))
       Memo1->Lines->Add("This cannot happen! CreateGUID guarantees that 2 " +
                "String("randomly generated GUIDs cannot be equal!"));
}
```

Delphi Examples:

```
{
This example demonstrates the usage of some GUID related routines along with
the type itself.
}
procedure TForm2.FormCreate(Sender: TObject);
var
  MyGuid0, MyGuid1 : TGUID;

begin
  { Create a new GUID from the string representation }
  MyGuid0 := StringToGUID('{00020400-0000-0000-C000-000000000046}');
  Memo1.Lines.Add('The GUID is: ' + GUIDToString(MyGuid0));

  {
  Accessing GUID's internal fields. Using Format function to obtain the
  same output as GUIDToString
  }
  Memo1.Lines.Add(Format('GUID using formatting is: ' +
       '{%0.8X-%0.4X-%0.4X-%0.2X%0.2X-%0.2X%0.2X%0.2X%0.2X%0.2X%0.2X}',
       [MyGuid0.D1, MyGuid0.D2, MyGuid0.D3,
       MyGuid0.D4[0], MyGuid0.D4[1], MyGuid0.D4[2], MyGuid0.D4[3],
       MyGuid0.D4[4], MyGuid0.D4[5], MyGuid0.D4[6], MyGuid0.D4[7]]));

  { Auto-generate a random GUID at runtime }
  if CreateGUID(MyGuid1) <> 0 then
     Memo1.Lines.Add('Creating GUID failed!')
  else
     Memo1.Lines.Add('The generated guid is: ' + GUIDToString(MyGuid1));

  { Generating second random GUID }
  CreateGUID(MyGuid0);

  { Testing if 2 guids are equal }
  if IsEqualGUID(MyGuid0, MyGuid1) then
     Memo1.Lines.Add('This cannot happen! CreateGUID guarantees that ' +
                     '2 randomly generated GUIDs cannot be equal!');
end;
```

See Also

GUIDToString (see page 4210)

37.3.204 SysUtils.StrLCat Function

Appends up to a specified maximum number of characters to string.

Pascal

```
function StrLCat(Dest: PAnsiChar; const Source: PAnsiChar; MaxLen: Cardinal): PAnsiChar;
overload;
function StrLCat(Dest: PWideChar; const Source: PWideChar; MaxLen: Cardinal): PWideChar;
```

```
  overload;
```

C++

```
PAnsiChar StrLCat(PAnsiChar Dest, const PAnsiChar Source, unsigned MaxLen);
PWideChar StrLCat(PWideChar Dest, const PWideChar Source, unsigned MaxLen);
```

File

SysUtils

Description

StrLCat appends at most MaxLen - StrLen (see page 4260)(Dest) characters from Source to the end of Dest and returns Dest. That is, MaxLen indicates the maximum length that is allowed in the result string.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString EditText = AnsiString(Edit1->Text);
  char* szFirstHalf = new char[StrLen(EditText.c_str())/2+2];
  char* szSecondHalf = new char[StrLen(EditText.c_str())/2+2];
  char* pszSecondHalf = &EditText[StrLen(EditText.c_str())/2+1];

  *szFirstHalf = 0;
  *szSecondHalf = 0;

  StrLCat(szFirstHalf, EditText.c_str(), StrLen(EditText.c_str())/2);
  StrCat(szSecondHalf, pszSecondHalf);

  Application->MessageBox(WideString(szFirstHalf).c_bstr(), L"First Half", MB_OKCANCEL);
  Application->MessageBox(WideString(szSecondHalf).c_bstr(), L"Second Half", MB_OKCANCEL);

  delete [] szFirstHalf;
  delete [] szSecondHalf;
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FirstHalf: PChar;
  SecondHalf: PChar;
  HalfLen: Integer;
begin
  HalfLen := StrLen(PChar(string(Edit1.Text))) div 2;
  GetMem(FirstHalf,HalfLen+2);
  GetMem(SecondHalf,HalfLen+2);
  FirstHalf^ := Chr(0);
  SecondHalf^ := Chr(0);
  StrLCat(FirstHalf, PChar(string(Edit1.Text)), HalfLen);
  StrCat(SecondHalf, PChar(string(Edit1.Text)) + HalfLen);
//  Application.MessageBox(FirstHalf, 'First Half', [smbOK], smsInformation, smbOK, smbCancel);
```

```
  Application.MessageBox(FirstHalf, 'First Half', MB_OKCANCEL);
  Application.MessageBox(SecondHalf, 'Second Half', MB_OKCANCEL);
  FreeMem(FirstHalf);
  FreeMem(SecondHalf);
end;
```

See Also

SizeOf (see page 3670)

StrCat (see page 4243)

StrLen (see page 4260)

37.3.205 SysUtils.StrLComp Function

Compares up to a specified maximum number of characters in two strings.

Pascal

```
function StrLComp(const Str1: PAnsiChar; const Str2: PAnsiChar; MaxLen: Cardinal): Integer;
overload;
function StrLComp(const Str1: PWideChar; const Str2: PWideChar; MaxLen: Cardinal): Integer;
overload;
```

C++

```
int StrLComp(const PAnsiChar Str1, const PAnsiChar Str2, unsigned MaxLen);
int StrLComp(const PWideChar Str1, const PWideChar Str2, unsigned MaxLen);
```

File

SysUtils

Description

StrLComp compares Str1 to Str2, up to a maximum length of MaxLen characters. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1 is the same as Str2
>0	Str1 sorts after Str2

Note: When working with international characters, use AnsiStrLComp (see page 4058) instead.

C++ Examples:

```
/*
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings.
*/
#include <algorithm>  // for the min function
#include <memory>       // for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 65]);
  int len = std::min(Edit1->Text.Length(), Edit2->Text.Length());
```

```
  int iResult = Sysutils::StrLComp(
    Edit1->Text.c_str(), Edit2->Text.c_str(), len + 1);
  StrCopy(szResult.get(), "The first ");
  StrCat(szResult.get(), IntToStr(len).t_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit1->Text.t_str());
  if (iResult < 0)
    StrCat(szResult.get(), " are less than the first ");
  else if (iResult > 0)
    StrCat(szResult.get(), " are greater than the first ");
  else
    StrCat(szResult.get(), " are equal to the first ");
  StrCat(szResult.get(), IntToStr(len).t_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit2->Text.t_str());

  ShowMessage(szResult.get());
}
```

Delphi Examples:

```
{
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S1, S2: String;
  Pos: Integer;
  Equal: Boolean;
begin
  Pos := 1;
  Equal := True;
  S1 := Edit1.Text;
  S2 := Edit2.Text;
  Form1.Repaint;
  while (Equal = True) do
  begin
    if StrLComp(PChar(S1), PChar(S2), Pos) = 0 then
      Inc(Pos)
    else
      Equal := False;
  end;
  Canvas.TextOut(10, 10, 'The first ' + IntToStr(Pos - 1) +
    ' characters are equal.');
end;
```

See Also

AnsiStrLComp (see page 4058)

StrComp (see page 4244)

StrIComp (see page 4251)

StrLIComp (see page 4262)

37.3.206 SysUtils.StrLCopy Function

Copies up to a specified maximum number of characters from Source to Dest.

Pascal

```
function StrLCopy(Dest: PAnsiChar; const Source: PAnsiChar; MaxLen: Cardinal): PAnsiChar;
overload;
function StrLCopy(Dest: PWideChar; const Source: PWideChar; MaxLen: Cardinal): PWideChar;
overload;
```

C++

```
PAnsiChar StrLCopy(PAnsiChar Dest, const PAnsiChar Source, unsigned MaxLen);
PWideChar StrLCopy(PWideChar Dest, const PWideChar Source, unsigned MaxLen);
```

File

SysUtils

Description

StrLCopy copies at most MaxLen characters from Source to Dest, then adds a null terminator to Dest and returns Dest. The SizeOf (see page 3670) standard function (Delphi) or the sizeof operator (C++) can be used to determine the MaxLen parameter. Usually MaxLen equals SizeOf (see page 3670)(Dest)-1.

Note: StrLCopy copies a null terminator as well as MaxLen characters. Be sure that Dest is large enough to accommodate the null terminator at the end.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the first X bytes of the
edit control are copied into a buffer, where X is a
predefined number.
*/
const MAX_BUFFER = 10;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char szBuffer[MAX_BUFFER+1];
  StrLCopy(szBuffer, Edit1->Text.t_str(), MAX_BUFFER);
  Application->MessageBox(WideString(szBuffer).c_bstr(), L"StrLCopy example", MB_OKCANCEL);
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form. When the button is clicked, the first X bytes of the
edit control are copied into a buffer, where X is a
predefined number.
}
const MAX_BUFFER = 10;
procedure TForm1.Button1Click(Sender: TObject);
var
  Buffer: array [0..MAX_BUFFER] of char;
begin
  StrLCopy(Buffer, PChar(Edit1.Text), MAX_BUFFER);
  Application.MessageBox(Buffer, 'StrLCopy example', MB_OKCANCEL);
end;
```

See Also

SizeOf (see page 3670)

StrCopy (see page 4246)

StrLen (see page 4260)

37.3.207 SysUtils.StrLen Function

Returns number of characters in a string excluding the null terminator.

Pascal

```
function StrLen(const Str: PAnsiChar): Cardinal; overload;
function StrLen(const Str: PWideChar): Cardinal; overload;
```

C++

```
unsigned StrLen(const PAnsiChar Str);
unsigned StrLen(const PWideChar Str);
```

File

SysUtils

Description

StrLen returns the number of characters in Str (see page 3676), not counting the null terminator.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString EditText = AnsiString(Edit1->Text);
  char* szFirstHalf = new char[StrLen(EditText.c_str())/2+2];
  char* szSecondHalf = new char[StrLen(EditText.c_str())/2+2];
  char* pszSecondHalf = &EditText[StrLen(EditText.c_str())/2+1];

  *szFirstHalf = 0;
  *szSecondHalf = 0;

  StrLCat(szFirstHalf, EditText.c_str(), StrLen(EditText.c_str())/2);
  StrCat(szSecondHalf, pszSecondHalf);

  Application->MessageBox(WideString(szFirstHalf).c_bstr(), L"First Half", MB_OKCANCEL);
  Application->MessageBox(WideString(szSecondHalf).c_bstr(), L"Second Half", MB_OKCANCEL);

  delete [] szFirstHalf;
  delete [] szSecondHalf;
}
```

Delphi Examples:

```
{
The following example uses an edit control and a button on a
form in an application. When the button is clicked, the edit
control's text is split in halves that are copied into
separate buffers. Then the contents of the buffers are
displayed in a message box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  FirstHalf: PChar;
  SecondHalf: PChar;
  HalfLen: Integer;
```

```
begin
  HalfLen := StrLen(PChar(string(Edit1.Text))) div 2;
  GetMem(FirstHalf,HalfLen+2);
  GetMem(SecondHalf,HalfLen+2);
  FirstHalf^ := Chr(0);
  SecondHalf^ := Chr(0);
  StrLCat(FirstHalf, PChar(string(Edit1.Text)), HalfLen);
  StrCat(SecondHalf, PChar(string(Edit1.Text)) + HalfLen);
//  Application.MessageBox(FirstHalf, 'First Half', [smbOK], smsInformation, smbOK, smbCancel);
  Application.MessageBox(FirstHalf, 'First Half', MB_OKCANCEL);
  Application.MessageBox(SecondHalf, 'Second Half', MB_OKCANCEL);
  FreeMem(FirstHalf);
  FreeMem(SecondHalf);
end;
```

See Also

StrEnd (see page 4249)

37.3.208 SysUtils.StrLFmt Function

Formats a series of arguments from a specified open array into a buffer.

Pascal

```
function StrLFmt(Buffer: PAnsiChar; MaxBufLen: Cardinal; Format: PAnsiChar; const Args: array
of const): PAnsiChar; overload;
function StrLFmt(Buffer: PAnsiChar; MaxBufLen: Cardinal; Format: PAnsiChar; const Args: array
of const; const FormatSettings: TFormatSettings): PAnsiChar; overload;
function StrLFmt(Buffer: PWideChar; MaxBufLen: Cardinal; Format: PWideChar; const Args: array
of const): PWideChar; overload;
function StrLFmt(Buffer: PWideChar; MaxBufLen: Cardinal; Format: PWideChar; const Args: array
of const; const FormatSettings: TFormatSettings): PWideChar; overload;
```

C++

```
PAnsiChar StrLFmt(PAnsiChar Buffer, unsigned MaxBufLen, PAnsiChar Format, const array of const
Args);
PAnsiChar StrLFmt(PAnsiChar Buffer, unsigned MaxBufLen, PAnsiChar Format, const array of const
Args, const TFormatSettings FormatSettings);
PWideChar StrLFmt(PWideChar Buffer, unsigned MaxBufLen, PWideChar Format, const array of const
Args);
PWideChar StrLFmt(PWideChar Buffer, unsigned MaxBufLen, PWideChar Format, const array of const
Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

StrLFmt formats the series of arguments in and open array and writes the results to a buffer.

Buffer is the buffer that receives the results.

MaxLen is the maximum number of bytes that can be written to Buffer. This value does not include the terminating null character.

Format (see page 4188) is the format string that governs the operation.

Args is an array of arguments that replace the format specifiers in Format (see page 4188).

Note: Args_Size is the index of the last argument in Args (one less than the number of arguments).

StrLFmt returns a pointer to the destination buffer.

The first form of StrLFmt is not thread-safe, because it uses localization information contained in global variables. The second form of StrLFmt, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrLFmt, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

FormatBuf (see page 4193)

StrFmt (see page 4250)

FmtStr (see page 4185)

GetLocaleFormatSettings (see page 4206)

37.3.209 SysUtils.StrLIComp Function

Compares strings up to a specified maximum number of characters, without case sensitivity.

Pascal

```
function StrLIComp(const Str1: PAnsiChar; const Str2: PAnsiChar; MaxLen: Cardinal): Integer;
overload;
function StrLIComp(const Str1: PWideChar; const Str2: PWideChar; MaxLen: Cardinal): Integer;
overload;
```

C++

```
int StrLIComp(const PAnsiChar Str1, const PAnsiChar Str2, unsigned MaxLen);
int StrLIComp(const PWideChar Str1, const PWideChar Str2, unsigned MaxLen);
```

File

SysUtils

Description

StrLIComp compares Str1 to Str2, up to a maximum length of MaxLen characters, without case sensitivity. The return value is indicated in the following table:

Return value	Condition
<0	Str1 sorts before Str2
=0	Str1 is the same as Str2
>0	Str1 sorts after Str2

Note: When working with international characters, use AnsiStrLIComp (see page 4058) instead.

C++ Examples:

```
/*
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings without case sensitivity.
*/
#include <algorithm>  // for the min function
#include <memory>       // for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
  std::auto_ptr<char> szResult(new char[Edit1->Text.Length() + Edit2->Text.Length() + 65]);
  int len = std::min(Edit1->Text.Length(), Edit2->Text.Length());
  int iResult = StrLIComp(
    Edit1->Text.c_str(), Edit2->Text.c_str(), len);

  StrCopy(szResult.get(), "The first ");
  StrCat(szResult.get(), IntToStr(len).t_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit1->Text.t_str());
  if (iResult < 0)
    StrCat(szResult.get(), " are less than the first ");
  else if (iResult > 0)
    StrCat(szResult.get(), " are greater than the first ");
  else
    StrCat(szResult.get(), " are equal to the first ");
  StrCat(szResult.get(), IntToStr(len).t_str());
  StrCat(szResult.get(), " characters of ");
  StrCat(szResult.get(), Edit2->Text.t_str());

  ShowMessage(szResult.get());
}
```

Delphi Examples:

```
{
This example requires two text edits and a button.
Enter text in the two text edits and click the button
to compare the strings, without case sensitivity.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  S1, S2: String;
  Pos: Integer;
  Equal: Boolean;
begin
  Pos := 1;
  Equal := True;
  S1 := Edit1.Text;
  S2 := Edit2.Text;
  Form1.Repaint;
  while (Equal = True) do
  begin
    if StrLIComp(PChar(S1), PChar(S2), Pos) = 0 then
      Inc(Pos)
    else
      Equal := False;
  end;
  Canvas.TextOut(10, 10, 'The first ' + IntToStr(Pos - 1) +
    ' characters are equal.');
end;
```

See Also

AnsiStrLIComp (see page 4058)

StrComp (see page 4244)

StrIComp (see page 4251)

StrLComp (see page 4257)

37.3.210 SysUtils.StrLower Function

Converts a string to lowercase.

Pascal

```
function StrLower(Str: PAnsiChar): PAnsiChar; overload;
function StrLower(Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrLower(PAnsiChar Str);
PWideChar StrLower(PWideChar Str);
```

File

SysUtils

Description

StrLower converts Str (see page 3676) to lowercase and returns Str (see page 3676).

Note: If the source string contains international characters, use AnsiStrLower (see page 4059) instead. If the source string is an AnsiString (long string), use LowerCase (see page 4225) (or AnsiLowerCase (see page 4053)) instead.

C++ Examples:

```
/*
The following example uses an edit box, two labels, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in label1's
caption and upper case in label2's caption.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = StrLower(Edit1->Text.c_str());
  Label2->Caption = StrUpper(Edit1->Text.c_str());
}
```

Delphi Examples:

```
{
The following example uses an edit box, a label, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in the label's
caption.
}
var
  S: array[0..20] of Char = 'A fUnNy StRiNg';

procedure TForm1.Button1Click(Sender: TObject);
var
  lower, upper : string;
begin
  lower := string(StrLower(S));
  upper := string(StrUpper(S));
  Canvas.TextOut(5, 10, lower + ' ' + upper);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  Label1.Caption := S;
end;
```

See Also

AnsiLowerCase (see page 4053)

AnsiStrLower (see page 4059)

LowerCase (see page 4225)

StrUpper (see page 4292)

37.3.211 SysUtils.StrMove Function

Copies specified number of characters to string.

Pascal

```
function StrMove(Dest: PAnsiChar; const Source: PAnsiChar; Count: Cardinal): PAnsiChar;
overload;
function StrMove(Dest: PWideChar; const Source: PWideChar; Count: Cardinal): PWideChar;
overload;
```

C++

```
PAnsiChar StrMove(PAnsiChar Dest, const PAnsiChar Source, unsigned Count);
PWideChar StrMove(PWideChar Dest, const PWideChar Source, unsigned Count);
```

File

SysUtils

Description

StrMove copies exactly Count characters from Source to Dest and returns Dest. Source and Dest can overlap.

C++ Examples:

```
/*
This example shows two functions that do the same thing as
StrNew and StrDispose.
*/
char * AHeapaString(char *S)
{
// Allocate string on heap
  if ((S != NULL) && (S[0] != '\0'))
  {
    int L = StrLen(S) + 1;
    char *p = (char *)malloc(L);
    return StrMove(p, S, L);
  }
  return NULL;
}
void DisposeDaString(char *S)
{
// Dispose string on heap
  if (S != NULL)
    free(S);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char *S = "Now you see it.";
  char *otherS = AHeapaString(S);
  Canvas->TextOut(5, 10, otherS);
```

```
  Canvas->Refresh();
  Sleep(5000);
  DisposeDaString(otherS);
}
```

Delphi Examples:

```
{
This example shows two functions that do the same thing as
StrNew and StrDispose.
}
function AHeapaString(S: PChar): PChar;
{ Allocate string on heap }
 var
  L: Cardinal;
  P: PChar;
  StrNew: PChar;
begin
  StrNew := nil;
  if (S <> nil) and (S[0] <> #0) then
  begin
    L := StrLen(S) + 1;
    GetMem(P, L);
    StrNew := StrMove(P, S, L);
  end;
  Result := StrNew;
end;

procedure DisposeDaString(S: PChar);
{ Dispose string on heap }
begin
  if S <> nil then FreeMem(S, StrLen(S) + 1);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  S, otherS: PChar;
begin
  S := 'Now you see it';
  otherS := AHeapaString(S);
  Canvas.TextOut(5, 10, otherS);
  Canvas.Refresh;
  Sleep(5000);
  DisposeDaString(otherS);
end;
```

See Also

Move (see page 3617)

37.3.212 SysUtils.StrNew Function

Allocates space on and copies a string to the heap; returning a pointer to the string.

Pascal

```
function StrNew(const Str: PAnsiChar): PAnsiChar; overload;
function StrNew(const Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrNew(const PAnsiChar Str);
PWideChar StrNew(const PWideChar Str);
```

File

SysUtils

Description

StrNew allocates a copy of Str (see page 3676) on the heap.

If Str (see page 3676) is nil (Delphi) or NULL (C++) or points to an empty string, StrNew returns a pointer to a new empty string.

Otherwise, StrNew makes a duplicate of Str (see page 3676), obtaining space with a call to StrAlloc (see page 4241), and returns a pointer to the duplicated string.

The allocated space is the length of Str (see page 3676) + 5 bytes.

C++ Examples:

```
/*
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Allocate memory.
  char* psz = StrNew(Edit1->Text.t_str());
  Application->MessageBox(
    Edit1->Text.c_str(),
    L"StrNew, StrDispose example",
    MB_OK);
  // Deallocate memory.
  StrDispose(psz);
}
```

Delphi Examples:

```
{
This example uses an edit control and a button on a form.
When the button is clicked, memory is allocated for a copy
of the text in the edit control, the text is displayed in a
message box, and then the memory is deallocated.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Temp: PChar;
begin
  // Allocate memory.
  Temp := StrNew(PChar(Edit1.Text));
  Application.MessageBox(Temp, 'StrNew, StrDispose example', MB_OK);
  // Deallocate memory.
  SysUtils.StrDispose(Temp);
end;
```

See Also

GetMem (see page 3584)

StrDispose (see page 4247)

StrAlloc (see page 4241)

37.3.213 SysUtils.StrNextChar Function

Returns pointer to following character.

Pascal

```
function StrNextChar(const Str: PAnsiChar): PAnsiChar; overload;
function StrNextChar(const Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrNextChar(const PAnsiChar Str);
PWideChar StrNextChar(const PWideChar Str);
```

File

SysUtils

Description

Note: StrNextChar returns a pointer to the second character in Str (see page 3676), allowing for multi-byte characters.

37.3.214 SysUtils.StrPas Function

Converts null-terminated string to an AnsiString (long string).

Pascal

```
function StrPas(const Str: PAnsiChar): AnsiString; overload;
```

C++

```
AnsiString StrPas(const PAnsiChar Str);
```

File

SysUtils

Description

This function is provided for backwards compatibility only. To convert a null terminated string to an AnsiString or native Delphi language string, use a typecast or an assignment.

C++ Examples:

```
/*
The following code locks the memory for text on the Clipboard, then
reads the text.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
      AnsiString MyString;

      int TextHandle;
      PChar pText;

      Clipboard()->Open();
      try
      {
```

```
            TextHandle = Clipboard()->GetAsHandle(CF_UNICODETEXT);
            pText = (PChar)GlobalLock((HGLOBAL)TextHandle);
            MyString = pText;
            ListBox1->Items->Add(StrPas(pText));
            GlobalUnlock((HGLOBAL)TextHandle);
      }
      catch (...)
      {
            Clipboard()->Close();
            throw;
      }
      Clipboard()->Close();
}
```

Delphi Examples:

```
{
The following code locks the memory for text on the Clipboard, then reads the text.
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
      MyHandle: THandle;
      TextPtr: PChar;
begin
      ClipBoard.Open;
      try
      {$IFNDEF UNICODE}
            MyHandle := Clipboard.GetAsHandle(CF_TEXT);
      {$ELSE}
            MyHandle := Clipboard.GetAsHandle(CF_UNICODETEXT);
      {$ENDIF}

            TextPtr := GlobalLock(MyHandle);
            ListBox1.Items.Add(StrPas(TextPtr));
            GlobalUnlock(MyHandle);
      finally
            Clipboard.Close;
      end;
end;
```

See Also

StrPCopy (see page 4271)

37.3.215 SysUtils.StrPas Function

Converts null-terminated string to an AnsiString (long string).

Pascal

```
function StrPas(const Str: PWideChar): UnicodeString; overload;
```

C++

```
UnicodeString StrPas(const PWideChar Str);
```

File

SysUtils

Description

This function is provided for backwards compatibility only. To convert a null terminated string to an AnsiString or native Delphi language string, use a typecast or an assignment.

C++ Examples:

```
/*
The following code locks the memory for text on the Clipboard, then
reads the text.
*/
#include <Clipbrd.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
      AnsiString MyString;

      int TextHandle;
      PChar pText;

      Clipboard()->Open();
      try
      {
            TextHandle = Clipboard()->GetAsHandle(CF_UNICODETEXT);
            pText = (PChar)GlobalLock((HGLOBAL)TextHandle);
            MyString = pText;
            ListBox1->Items->Add(StrPas(pText));
            GlobalUnlock((HGLOBAL)TextHandle);
      }
      catch (...)
      {
            Clipboard()->Close();
            throw;
      }
      Clipboard()->Close();
}
```

Delphi Examples:

```
{
The following code locks the memory for text on the Clipboard, then reads the text.
}

uses clipbrd;

procedure TForm1.Button1Click(Sender: TObject);
var
      MyHandle: THandle;
      TextPtr: PChar;
begin
      ClipBoard.Open;
      try
      {$IFNDEF UNICODE}
            MyHandle := Clipboard.GetAsHandle(CF_TEXT);
      {$ELSE}
            MyHandle := Clipboard.GetAsHandle(CF_UNICODETEXT);
      {$ENDIF}

            TextPtr := GlobalLock(MyHandle);
            ListBox1.Items.Add(StrPas(TextPtr));
            GlobalUnlock(MyHandle);
      finally
            Clipboard.Close;
      end;
end;
```

See Also

StrPCopy (see page 4271)

37.3.216 SysUtils.StrPCopy Function

Copies an AnsiString (long string) to a null-terminated string.

Pascal

```
function StrPCopy(Dest: PAnsiChar; const Source: AnsiString): PAnsiChar; overload;
function StrPCopy(Dest: PWideChar; const Source: UnicodeString): PWideChar; overload;
```

C++

```
PAnsiChar StrPCopy(PAnsiChar Dest, const AnsiString Source);
PWideChar StrPCopy(PWideChar Dest, const UnicodeString Source);
```

File

SysUtils

Description

StrPCopy copies Source into a null-terminated string Dest. It returns a pointer to Dest.

StrPCopy does not perform any length checking.

The destination buffer must have room for at least Length (see page 3607)(Source)+1 characters.

C++ Examples:

```
/*
The following example uses an edit control, a label, and a
button on a form. When the button is clicked, the label's
caption and the edit control's text are combined in a buffer.
Then the buffer is displayed in the label's caption.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<char> szBuffer(new char[Label1->Caption.Length() + Edit1->Text.Length() + 1]);
  StrPCopy(szBuffer.get(), Label1->Caption);
  StrCat(szBuffer.get(), Edit1->Text.t_str());
  Label1->Caption = szBuffer.get();
  Edit1->Clear();
}
```

Delphi Examples:

```
{
This example requires a button on a form and uses StrPCopy
to copy a string into an array and then display the character
array on the form canvas.  StrPCopy does not do length checking.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  A: array[0..79] of Char;
  S: String;
begin
  S := 'Honk if you know Blaise.';
  StrPCopy(A, S);
```

```
  Canvas.TextOut(10, 10, string(A));
end;
```

See Also

StrCopy (see page 4246)

37.3.217 SysUtils.StrPLCopy Function

Copies characters from an AnsiString (long string) into a null-terminated string.

Pascal

```
function StrPLCopy(Dest: PAnsiChar; const Source: AnsiString; MaxLen: Cardinal): PAnsiChar;
overload;
function StrPLCopy(Dest: PWideChar; const Source: UnicodeString; MaxLen: Cardinal): PWideChar;
overload;
```

C++

```
PAnsiChar StrPLCopy(PAnsiChar Dest, const AnsiString Source, unsigned MaxLen);
PWideChar StrPLCopy(PWideChar Dest, const UnicodeString Source, unsigned MaxLen);
```

File

SysUtils

Description

StrPLCopy copies up to a maximum of MaxLen characters from Source into the null-terminated string Dest. StrPLCopy returns a pointer to Dest.

Note: Dest should point to a location with room for MaxLen + 1 characters, including the null terminator.

37.3.218 SysUtils.StrPos Function

Returns a pointer to the first occurrence of STR2 in STR1.

Pascal

```
function StrPos(const Str1: PAnsiChar; const Str2: PAnsiChar): PAnsiChar; overload;
```

C++

```
PAnsiChar StrPos(const PAnsiChar Str1, const PAnsiChar Str2);
```

File

SysUtils

Description

StrPos returns a pointer to the first occurrence of Str2 in Str1. If Str2 does not occur in Str1, StrPos returns nil (Delphi) or NULL (C++).

Note: If the source string contains international characters, use AnsiStrPos (see page 4060) instead.

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the first
```

```
edit control is searched for the text in the second edit control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
      if StrPos(PChar(Edit1.Text), PChar(Edit2.Text)) <> nil then
         ShowMessage('Substring found')
      else
         ShowMessage('Substring not found');
end;
```

C++ Examples:

```
/*
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the first
edit control is searched for the text in the second edit control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      if (StrPos(Edit1->Text.c_str(), Edit2->Text.c_str()))
         ShowMessage("Substring found.");
      else
         ShowMessage("Substring not found.");
}
```

See Also

AnsiStrPos (see page 4060)

37.3.219 SysUtils.StrPos Function

Returns a pointer to the first occurrence of STR2 in STR1.

Pascal

```
function StrPos(const Str1: PWideChar; const Str2: PWideChar): PWideChar; overload;
```

C++

```
PWideChar StrPos(const PWideChar Str1, const PWideChar Str2);
```

File

SysUtils

Description

StrPos returns a pointer to the first occurrence of Str2 in Str1. If Str2 does not occur in Str1, StrPos returns nil (Delphi) or NULL (C++).

Note: If the source string contains international characters, use AnsiStrPos (see page 4060) instead.

Delphi Examples:

```
{
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the first
edit control is searched for the text in the second edit control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
      if StrPos(PChar(Edit1.Text), PChar(Edit2.Text)) <> nil then
         ShowMessage('Substring found')
      else
         ShowMessage('Substring not found');
```

```
end;
```

C++ Examples:

```
/*
The following example uses two edit controls and a button on
a form. When the button is clicked, the text in the first
edit control is searched for the text in the second edit control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
      if (StrPos(Edit1->Text.c_str(), Edit2->Text.c_str()))
         ShowMessage("Substring found.");
      else
         ShowMessage("Substring not found.");
}
```

See Also

AnsiStrPos (see page 4060)

37.3.220 SysUtils.StrRScan Function

Returns a pointer to the last occurrence of a specified character in a string.

Pascal

```
function StrRScan(const Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function StrRScan(const Str: PWideChar; Chr: WideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrRScan(const PAnsiChar Str, AnsiChar Chr);
PWideChar StrRScan(const PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

StrRScan returns a pointer to the last occurrence of Chr (see page 3536) in Str (see page 3676).

If Chr (see page 3536) does not occur in Str (see page 3676), StrRScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Note: If the source string contains international characters, use AnsiStrRScan (see page 4060) instead.

C++ Examples:

```
/*
The following example uses an Open dialog box and a button
on a form. When the button is clicked, the user is prompted
for a filename. Then the filename, minus its path, is
displayed in a message box.
Note:   This technique will not work with multi-byte
character sets (MBCS). To be more general, use ExtractFileName.
*/
char *__fastcall GetFileName(char *pszFilePath)
{
  char *pszFileName = StrRScan(pszFilePath, '\\');
  if (!pszFileName)
    pszFileName = pszFilePath;
```

```
    else
      pszFileName++;
    return pszFileName;
  }
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    if (OpenDialog1->Execute())
    {
      char *name = GetFileName(OpenDialog1->FileName.t_str());
      Application->MessageBox(
        WideString(name).c_bstr(),
        L"File Name",
        MB_OK);
    }
  }
```

Delphi Examples:

```
{
This function behaves much like ExtractFileName except that
it uses null-terminated strings and it will not work with
multi-byte character sets (MBCS).
}
function NamePart(FileName: PAnsiChar): PAnsiChar;
 var
  P: PAnsiChar;
  sep: Char;
begin
  sep := '/';
  P := StrRScan(FileName, '/');
  if P = nil then
  begin
    P := StrRScan(FileName, ':');
    if P = nil then P := FileName;
  end;
  NamePart := P;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  S : string;
begin
  S := string(NamePart('C:\Test.fil'));
  Canvas.TextOut(10, 10, S);
end;
```

See Also

AnsiStrRScan (see page 4060)

StrScan (see page 4275)

37.3.221 SysUtils.StrScan Function

Returns a pointer to first occurrence of a specified character in a string.

Pascal

```
function StrScan(const Str: PAnsiChar; Chr: AnsiChar): PAnsiChar; overload;
function StrScan(const Str: PWideChar; Chr: WideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrScan(const PAnsiChar Str, AnsiChar Chr);
```

```
PWideChar StrScan(const PWideChar Str, WideChar Chr);
```

File

SysUtils

Description

StrScan returns a pointer to the first occurrence of Chr (see page 3536) in Str (see page 3676). If Chr (see page 3536) does not occur in Str (see page 3676), StrScan returns nil (Delphi) or NULL (C++). The null terminator is considered to be part of the string.

Note: If the source string contains international characters, use AnsiStrScan (see page 4061) instead.

C++ Examples:

```
/*
The following example uses an edit control and a button on a
form. When the button is clicked, the text in the edit
control is searched for a wildcard (asterisk character).
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (StrScan(Edit1->Text.c_str(), '*'))
    ShowMessage("Wildcard found.");
  else
    ShowMessage("Wildcard not found.");
}
```

Delphi Examples:

```
{
The following example uses a button on a form. When the
button is clicked, the text is searched for a wildcard
(asterisk character).
}
function HasWildcards(FileName: PChar): Boolean;
{ Return true if file name has wildcards in it }
begin
  HasWildcards := (StrScan(FileName, '*') <> nil) or
    (StrScan(FileName, '?') <> nil);
end;

procedure TForm1.Button1Click(Sender: TObject);
const
  P: PChar = 'C:\Test.* ';
begin
  if HasWildcards(P) then
    ShowMessage('The string has wildcards')
  else
    ShowMessage('The string does not have wildcards');
end;
```

See Also

AnsiStrScan (see page 4061)

StrRScan (see page 4274)

37.3.222 SysUtils.StrToBool Function

Converts a string to a Boolean value.

Pascal

```
function StrToBool(const S: string): Boolean; overload;
```

C++

```
Boolean StrToBool(const UnicodeString S);
```

File

SysUtils

Description

StrToBool converts the string specified by S to a Boolean. If S can be converted to a number, StrToBool returns false if that number is 0, true otherwise. If S can't be converted to a number, StrToBool returns true if S is any of the strings listed in TrueBoolStrs (see page 4363) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 4352) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 4363) or FalseBoolStrs (see page 4352), StrToBool raises an EConvertError (see page 3919) exception.

Note: If you do not assign a value to TrueBoolStrs (see page 4363), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 4352), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 4363) and FalseBoolStrs (see page 4352) variables are undefined when an application starts. Any use of StrToBool, BoolToStr (see page 4066) or TryStrToBool (see page 4305) will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 4066)

TrueBoolStrs (see page 4363)

FalseBoolStrs (see page 4352)

37.3.223 SysUtils.StrToBoolDef Function

Converts a string to a Boolean value, with error default.

Pascal

```
function StrToBoolDef(const S: string; const Default: Boolean): Boolean; overload;
```

C++

```
Boolean StrToBoolDef(const UnicodeString S, const Boolean Default);
```

File

SysUtils

Description

StrToBoolDef converts the string specified by S to a Boolean. If S can be converted to a number, StrToBoolDef returns false if that number is 0, true otherwise. If S can't be converted to a number, StrToBoolDef returns true if S is any of the strings listed in TrueBoolStrs (see page 4363) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 4352) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 4363) or FalseBoolStrs (see page 4352), StrToBoolDef returns Default.

Note: If you do not assign a value to TrueBoolStrs (see page 4363), it is treated as if it contains the single string 'True'.

Similarly, if you do not assign a value to FalseBoolStrs (see page 4352), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 4363) and FalseBoolStrs (see page 4352) variables are undefined when an application starts. Any use of StrToBool (see page 4276), BoolToStr (see page 4066) or TryStrToBool (see page 4305) will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 4066)

TrueBoolStrs (see page 4363)

FalseBoolStrs (see page 4352)

StrToBool (see page 4276)

37.3.224 SysUtils.StrToCurr Function

Converts a string to a System::Currency (see page 3235) value.

Pascal

```
function StrToCurr(const S: string): Currency; overload;
function StrToCurr(const S: string; const FormatSettings: TFormatSettings): Currency; overload;
```

C++

```
Currency StrToCurr(const UnicodeString S);
Currency StrToCurr(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToCurr to convert a string that represents a floating-point value to the corresponding System::Currency (see page 3235) value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, StrToCurr raises an EConvertError (see page 3919) exception.

The first form of StrToCurr is not thread-safe, because it uses localization information contained in global variables. The second form of StrToCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.225 SysUtils.StrToCurrDef Function

Converts a string to a System::Currency (see page 3235) value, with error default.

Pascal

```
function StrToCurrDef(const S: string; const Default: Currency): Currency; overload;
function StrToCurrDef(const S: string; const Default: Currency; const FormatSettings:
```

```
TFormatSettings): Currency; overload;
```

C++

```
Currency StrToCurrDef(const UnicodeString S, const Currency Default);
Currency StrToCurrDef(const UnicodeString S, const Currency Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToCurrDef to convert a string that represents a floating-point value to the corresponding System::Currency (see page 3235) value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, StrToCurrDef returns Default.

The first form of StrToCurrDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToCurrDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToCurrDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.226 SysUtils.StrToDate Function

Converts a string to a TDateTime (see page 3323) value.

Pascal

```
function StrToDate(const S: string): TDateTime; overload;
function StrToDate(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToDate(const UnicodeString S);
TDateTime StrToDate(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDate to parse a string that specifies a date. If S does not contain a valid date, StrToDate raises an EConvertError (see page 3919) exception.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 4349) global variable or its TFormatSettings (see page 4326) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 4358) global variable or its TFormatSettings (see page 4326) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4364). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3235) and Date (see page 4087)-Time (see page 4297) Formatting Variables" for more information.

The first form of StrToDate is not thread-safe, because it uses localization information contained in global variables. The second

form of StrToDate, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDate, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  char days[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday" };
  TDateTime dtDate = StrToDate(Edit1->Text);
  ShowMessage(Edit1->Text + AnsiString(" is a ") + days[dtDate.DayOfWeek() - 1]);
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a date in the edit box in the format
associated with the current locale (for example MM/DD/YY
format in the US), the string entered is converted to a
TDateTime value. This value is used to indicate the day of
the week the date represents.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ADate: TDateTime;
  days: array[1..7] of string;
begin
  days[1] := 'Sunday';
  days[2] := 'Monday';
  days[3] := 'Tuesday';
  days[4] := 'Wednesday';
  days[5] := 'Thursday';
  days[6] := 'Friday';
  days[7] := 'Saturday';
  ADate := StrToDate(Edit1.Text);
  ShowMessage(Edit1.Text + ' is a ' + days[SysUtils.DayOfWeek(ADate)]);
end;
```

37.3.227 SysUtils.StrToDateDef Function

Converts a string to a TDateTime (see page 3323) value, with error default.

Pascal

```
function StrToDateDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToDateDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToDateDef(const UnicodeString S, const TDateTime Default);
TDateTime StrToDateDef(const UnicodeString S, const TDateTime Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToDateDef to parse a string that specifies a date. If S does not contain a valid date, StrToDateDef returns Default.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 4349) global variable or its TFormatSettings (see page 4326) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 4358) global variable or its TFormatSettings (see page 4326) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4364). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3235) and Date (see page 4087)-Time (see page 4297) Formatting Variables" for more information.

The first form of StrToDateDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.228 SysUtils.StrToDateTime Function

Converts a string to a TDateTime (see page 3323) value.

Pascal

```
function StrToDateTime(const S: string): TDateTime; overload;
function StrToDateTime(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToDateTime(const UnicodeString S);
TDateTime StrToDateTime(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDateTime to parse a string that specifies a date and time value. If S does not contain a valid date, StrToDateTime raises an EConvertError (see page 3919) exception.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4364). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3235) and Date (see page 4087)-Time (see page 4297) Formatting Variables" for more information.

The first form of StrToDateTime is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.229 SysUtils.StrToDateTimeDef Function

Converts a string to a TDateTime (see page 3323) value with error default.

Pascal

```
function StrToDateTimeDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToDateTimeDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToDateTimeDef(const UnicodeString S, const TDateTime Default);
TDateTime StrToDateTimeDef(const UnicodeString S, const TDateTime Default, const
TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToDateTimeDef to parse a string that specifies a date and time value. If S does not contain a valid date, StrToDateTimeDef returns Default.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4364). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3235) and Date (see page 4087)-Time (see page 4297) Formatting Variables" for more information.

The first form of StrToDateTimeDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToDateTimeDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToDateTimeDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

DateTimeToStr (see page 4089)

StrToDate (see page 4279)

StrToTime (see page 4291)

StrToDateTime (see page 4281)

37.3.230 SysUtils.StrToFloat Function

Converts a given string to a floating-point value.

Pascal

```
function StrToFloat(const S: string): Extended; overload;
function StrToFloat(const S: string; const FormatSettings: TFormatSettings): Extended;
overload;
```

C++

```
Extended StrToFloat(const UnicodeString S);
Extended StrToFloat(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Use StrToFloat to convert a string, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4350) global variable or its TFormatSettings (see page 4326) equivalent defines the character that is used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, StrToFloat raises an EConvertError (see page 3919) exception.

The first form of StrToFloat is not thread-safe, because it uses localization information contained in global variables. The second form of StrToFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

C++ Examples:

```
/*
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int x = 1; x < StringGrid1->ColCount; x++)
  {
    for (int y = 1; y < StringGrid1->RowCount; y++)
    {
      StringGrid1->Cells[x][y] = FormatFloat(
      StringGrid1->Cells[0][y], StrToFloat(StringGrid1->Cells[x][0]));
    }
  }
}

int currCellCol, currCellRow;

void __fastcall TForm1::StringGrid1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
  StringGrid1->MouseToCell(X, Y, currCellCol, currCellRow);
}

// Alter Cell
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  StringGrid1->Cells[currCellCol][currCellRow] = Edit1->Text;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[1][0] = "1234";
  StringGrid1->Cells[2][0] = "-1234";
  StringGrid1->Cells[3][0] = "0.5";
```

```
  StringGrid1->Cells[4][0] = "0";
  StringGrid1->Cells[0][1] = "";
  StringGrid1->Cells[0][2] = "0";
  StringGrid1->Cells[0][3] = "0.00";
  StringGrid1->Cells[0][4] = "#.##";
  StringGrid1->Cells[0][5] = "#,##0.00";
  StringGrid1->Cells[0][6] = "#,##0.00;(#,##0.00)";
  StringGrid1->Cells[0][7] = "#,##0.00;;Zero";
  StringGrid1->Cells[0][8] = "0.000E+00";
  StringGrid1->Cells[0][9] = "#.###E-0";
}
```

Delphi Examples:

```
{
The following example uses two buttons, a string grid, and
text edit on a form. When the conversion button is clicked,
the values across the top of the string grid are converted
using the formats along the left side.  Select a cell and
use the Alter Cell button and text edit to change the
values converted and the conversion formats.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, I: Integer;
begin
  for X := 1 to StringGrid1.ColCount - 1 do
  begin
    for Y := 0 to StringGrid1.RowCount - 1 do
    begin
      StringGrid1.Cells[X,Y] := SysUtils.FormatFloat(
        StringGrid1.Cells[0,Y], StrToFloat(StringGrid1.Cells[X,0]));
    end;
  end;

end;

var currCellCol, currCellRow: Integer;

// Alter Cell
procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid1.Cells[currCellCol,currCellRow] := Edit1.Text;
end;

procedure TForm1.StringGrid1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  StringGrid1.MouseToCell(X, Y, &currCellCol, &currCellRow);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[1,0] := '1234';
  StringGrid1.Cells[2,0] := '-1234';
  StringGrid1.Cells[3,0] := '0.5';
  StringGrid1.Cells[4,0] := '0';
  StringGrid1.Cells[0,1] := '';
  StringGrid1.Cells[0,2] := '0';
  StringGrid1.Cells[0,3] := '0.00';
  StringGrid1.Cells[0,4] := '#.##';
  StringGrid1.Cells[0,5] := '#,##0.00';
  StringGrid1.Cells[0,6] := '#,##0.00;(#,##0.00)';
  StringGrid1.Cells[0,7] := '#,##0.00;;Zero';
  StringGrid1.Cells[0,8] := '0.000E+00';
  StringGrid1.Cells[0,9] := '#.###E-0';
end;
```

```
{
This example uses a form with a combo box, two list boxes,
an edit control, and a button.  The user selects a
conversion family from the combo box, which causes the list
boxes to fill with the conversion types registered with that
family. When the user clicks the button, a message box
appears showing the value in the edit control converted from
the units selected in the first list box to the units
selected in the second list box.
Note:   If you have not registered your own conversion families,
include the stdconvs unit in your uses clause to use the
standard conversion families.
}

{
The following OnCreate event handler for the form initializes
the combo box and list boxes:
}

procedure TForm1.FormCreate(Sender: TObject);
var
  FamilyList: TConvFamilyArray;
  i: Integer;
begin
  GetConvFamilies(FamilyList);
  for i := 0 to (Length(FamilyList) - 1) do
    ComboBox1.Items.Add(ConvFamilyToDescription(FamilyList[i]));
  ComboBox1.ItemIndex := 0; { select the first item }
  ComboBox1Select(ComboBox1); {trigger the event to initialize lists}
end;

{
The following OnClick event handler for the button reads the
value entered in the edit box and converts it between the
measurement units selected in the list boxes:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  newVal: Double;
  CurFamily: TConvFamily;
  FromType, ToType: TConvType;
begin
  DescriptionToConvFamily(ComboBox1.Items[ComboBox1.ItemIndex], CurFamily);
  DescriptionToConvType(CurFamily, ListBox1.Items[ListBox1.ItemIndex], FromType);
  DescriptionToConvType(CurFamily, ListBox2.Items[ListBox2.ItemIndex], ToType);
  newVal := Convert(StrToFloat(Edit1.Text), FromType, ToType);
  ShowMessage(Format('%g %s', [newVal, ConvTypeToDescription(ToType)]));
end;

{
The following OnSelect event handler for the combo box
initializes the list boxes with the registered conversion
types for the selected family:
}
procedure TForm1.ComboBox1Select(Sender: TObject);
var
  TypeList: TConvTypeArray;
  i: Integer;
  CurFamily: TConvFamily;Description:string;
beginDescription := ComboBox1.Items[ComboBox1.ItemIndex];
  if DescriptionToConvFamily(Description, CurFamily) then
  begin
    GetConvTypes(CurFamily, TypeList);
    ListBox1.Items.Clear;
    ListBox2.Items.Clear;
    for i := 0 to Length(TypeList) -1 do
```

```
      ListBox1.Items.Add(ConvTypeToDescription(TypeList[i]));
    ListBox2.Items := ListBox1.Items;
    ListBox1.ItemIndex := 0;
    ListBox2.ItemIndex := 0;
  end;
end;
```

37.3.231 SysUtils.StrToFloatDef Function

Converts a given string to a floating-point value with error default.

Pascal

```
function StrToFloatDef(const S: string; const Default: Extended): Extended; overload;
function StrToFloatDef(const S: string; const Default: Extended; const FormatSettings:
TFormatSettings): Extended; overload;
```

C++

```
Extended StrToFloatDef(const UnicodeString S, const Extended Default);
Extended StrToFloatDef(const UnicodeString S, const Extended Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use StrToFloatDef to convert a string, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4350) global variable or its TFormatSettings (see page 4326) equivalent defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, StrToFloatDef returns Default.

The first form of StrToFloatDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToFloatDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToFloatDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.232 SysUtils.StrToInt Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt(const S: string): Integer; overload;
```

C++

```
int StrToInt(const UnicodeString S);
```

File

SysUtils

Description

StrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a

number. If S does not represent a valid number, StrToInt raises an EConvertError (see page 3919) exception.

C++ Examples:

```
/*
This example uses two edit boxes and a button on a form. When the user clicks
the button, the code converts the values in the edit boxes to numbers, adds them,
and displays a message indicating the sum:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int i1, i2;
  i1 = StrToInt(Edit1->Text);
  i2 = StrToInt(Edit2->Text);
  ShowMessage(IntToStr(i1 + i2));
}
```

Delphi Examples:

```
{
This example uses two edit boxes and a button on a form. When the user clicks
the button, the code converts the values in the edit boxes to numbers, adds them,
and displays a message indicating the sum:
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  J: Integer;
begin
  I := StrToInt(Edit1.Text);
  J := StrToInt(Edit2.Text);
  ShowMessage(IntToStr(I + J));
end;
```

See Also

IntToHex (see page 4215)

IntToStr (see page 4216)

StrToIntDef (see page 4288)

TryStrToInt (see page 4308)

37.3.233 SysUtils.StrToInt64 Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt64(const S: string): Int64; overload;
```

C++

```
Int64 StrToInt64(const UnicodeString S);
```

File

SysUtils

Description

StrToInt64 converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt64 raises an EConvertError (see page 3919) exception.

See Also

IntToHex (see page 4215)

IntToStr (see page 4216)

StrToIntDef (see page 4288)

StrToInt64Def (see page 4288)

StrToInt (see page 4286)

TryStrToInt (see page 4308)

37.3.234 SysUtils.StrToInt64Def Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function StrToInt64Def(const S: string; const Default: Int64): Int64; overload;
```

C++

```
Int64 StrToInt64Def(const UnicodeString S, const Int64 Default);
```

File

SysUtils

Description

StrToInt64Def converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToInt64Def returns the number passed in Default.

See Also

IntToStr (see page 4216)

StrToInt (see page 4286)

StrToInt64 (see page 4287)

TryStrToInt64 (see page 4309)

37.3.235 SysUtils.StrToIntDef Function

Converts a string that represents an integer (decimal or hex notation) to a number with error default.

Pascal

```
function StrToIntDef(const S: string; Default: Integer): Integer; overload;
```

C++

```
int StrToIntDef(const UnicodeString S, int Default);
```

File

SysUtils

Description

StrToIntDef converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number. If S does not represent a valid number, StrToIntDef returns Default.

C++ Examples:

```
/*
This example uses two edit boxes and a button on a form. The
user enters a number in the first edit box and clicks the
button. If the number entered was a valid integer, the same
value appears in the second edit box. If the number was not
a valid integer, the default value of 1000 appears in the
second edit box:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int Number = StrToIntDef(Edit1->Text, 1000);
  Edit2->Text = IntToStr(Number);
}
```

Delphi Examples:

```
{
  This example uses the FetchParams method to sync the params
  with the server.
}
procedure TForm1.SaveParamsClick(Sender: TObject);
begin
  { Save the parameters for the TDataSet }
  SavedParams.Assign(CDS.Params);
end;

procedure TForm1.AlterParamsClick(Sender: TObject);
var
  I : Integer;
  ListItem: string;
begin
  CDS.Params.CreateParam(ftString, 'stringParam', ptInput);  // you can also edit the params
in the CDS dynamically
  CDS.Params.CreateParam(ftSmallInt, 'smallIntParam', ptInput);
  CDS.Params.CreateParam(ftInteger, 'intParam', ptInput);
  CDS.Params.CreateParam(ftWord, 'wordParam', ptInput);
  CDS.Params.CreateParam(ftBoolean, 'boolParam', ptInput);
  CDS.Params.CreateParam(ftFloat, 'floatParam', ptInput);
  CDS.Params.CreateParam(ftCurrency, 'currencyParam', ptInput);
  CDS.Params.CreateParam(ftBCD, 'bcdParam', ptInput);
  CDS.Params.CreateParam(ftDate, 'dateParam', ptInput);
  CDS.Params.CreateParam(ftTime, 'timeParam', ptInput);
  CDS.Params.CreateParam(ftDateTime, 'dateTimeParam', ptInput);
  { Call FetchParams to ensure parameters reflect server metadata }
  CDS.FetchParams;
  with CDS.Params do
  begin
    for I := 0 to Count - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case Items[I].DataType of
        ftString:
          Items[I].AsString := ListItem;
        ftSmallInt:
          Items[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          Items[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          Items[I].AsWord := StrToIntDef(ListItem, 0);
```

```
          ftBoolean:
            begin
              if ListItem = 'True' then
                Items[I].AsBoolean := True
              else
                Items[I].AsBoolean := False;
            end;
          ftFloat:
            Items[I].AsFloat := StrToFloat(ListItem);
          ftCurrency:
            Items[I].AsCurrency := StrToFloat(ListItem);
          ftBCD:
            Items[I].AsBCD := StrToCurr(ListItem);
          ftDate:
            Items[I].AsDate := StrToDate(ListItem);
          ftTime:
            Items[I].AsTime := StrToTime(ListItem);
          ftDateTime:
            Items[I].AsDateTime := StrToDateTime(ListItem);
        end;
      end;
    end;
  end;

  procedure TForm1.RestoreParamsClick(Sender: TObject);
  begin
    { Restore the parameters to TDataSet }
    CDS.Params.AssignValues(SavedParams);
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { Initialize SavedParams }
    SavedParams := TParams.Create;
    ListBox1.Items.Add('Here is a string');
    ListBox1.Items.Add('4');
    ListBox1.Items.Add('3456789');
    ListBox1.Items.Add('34');
    ListBox1.Items.Add('0');
    ListBox1.Items.Add('12.56');
    ListBox1.Items.Add('4.23');
    ListBox1.Items.Add('99.5');
    ListBox1.Items.Add('4/12/53');
    ListBox1.Items.Add('7:45:00');
    ListBox1.Items.Add('4/12/53 7:45:00');
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  begin
    SavedParams.Free;
  end;
  {
  This example uses two edit boxes and a button on a form. The
  user enters a number in the first edit box and clicks the
  button. If the number entered was a valid integer, the same
  value appears in the second edit box. If the number was not
  a valid integer, the default value of 1000 appears in the
  second edit box:
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    Number: Integer;
  begin
    Number := StrToIntDef(Edit1.Text, 1000);
    Edit2.Text := IntToStr(Number);
  end;
```

See Also

IntToHex (see page 4215)

IntToStr (see page 4216)

StrToInt (see page 4286)

TryStrToInt (see page 4308)

37.3.236 SysUtils.StrToTime Function

Converts a string to a TDateTime (see page 3323) value.

Pascal

```
function StrToTime(const S: string): TDateTime; overload;
function StrToTime(const S: string; const FormatSettings: TFormatSettings): TDateTime;
overload;
```

C++

```
TDateTime StrToTime(const UnicodeString S);
TDateTime StrToTime(const UnicodeString S, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

Call StrToTime to parse a string that specifies a time value. If S does not contain a valid time, StrToTime raises an EConvertError (see page 3919) exception.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 4363) global variable or its TFormatSettings (see page 4326) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of StrToTime is not thread-safe, because it uses localization information contained in global variables. The second form of StrToTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

C++ Examples:

```
/*
This example uses an edit box and a button on a form. When
the user enters a time in the edit box in the HH:MM:SS
format, the string entered is converted to a TDateTime value.
A message then appears that says "Good Morning" or "Good
Afternoon", depending on whether the time entered was in the
morning or afternoon.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dtTime = StrToTime(Edit1->Text);
  if (dtTime < (TDateTime) 0.50 )
    ShowMessage("Good Morning");
  else
    ShowMessage("Good Afternoon");
}
```

Delphi Examples:

```
{
This example uses an edit box and a button on a form. When
the user enters a time in the edit box in the HH:MM:SS
format, the string entered is converted to a TDateTime value.
A message then appears that says "Good Morning" or "Good
Afternoon", depending on whether the time entered was in the
morning or afternoon.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  ATime: TDateTime;
begin
  ATime := StrToTime(Edit1.Text);
  if ATime < 0.50 then
    ShowMessage('Good Morning')
  else
    ShowMessage('Good Afternoon');
end;
```

37.3.237 SysUtils.StrToTimeDef Function

Converts a string to a TDateTime (see page 3323) value with error default.

Pascal

```
function StrToTimeDef(const S: string; const Default: TDateTime): TDateTime; overload;
function StrToTimeDef(const S: string; const Default: TDateTime; const FormatSettings:
TFormatSettings): TDateTime; overload;
```

C++

```
TDateTime StrToTimeDef(const UnicodeString S, const TDateTime Default);
TDateTime StrToTimeDef(const UnicodeString S, const TDateTime Default, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call StrToTimeDef to parse a string that specifies a time value. If S does not contain a valid time, StrToTimeDef returns Default.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 4363) global variable or its TFormatSettings (see page 4326) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of StrToTimeDef is not thread-safe, because it uses localization information contained in global variables. The second form of StrToTimeDef, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of StrToTimeDef, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.238 SysUtils.StrUpper Function

Returns a string in upper case.

Pascal

```
function StrUpper(Str: PAnsiChar): PAnsiChar; overload;
function StrUpper(Str: PWideChar): PWideChar; overload;
```

C++

```
PAnsiChar StrUpper(PAnsiChar Str);
PWideChar StrUpper(PWideChar Str);
```

File

SysUtils

Description

StrUpper converts Str (see page 3676) to uppercase and returns Str (see page 3676).

Note: If the source string contains international characters, use AnsiStrUpper (see page 4061) instead. If the source string is an AnsiString (long string), use UpperCase (see page 4311) (or AnsiUpperCase (see page 4062)) instead.

C++ Examples:

```
/*
The following example uses an edit box, two labels, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in label1's
caption and upper case in label2's caption.
*/

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = StrLower(Edit1->Text.c_str());
  Label2->Caption = StrUpper(Edit1->Text.c_str());
}
```

Delphi Examples:

```
{
The following example uses an edit box, a label, and a
button on a form. When the button is clicked, the text in
the edit control is displayed in lower case in the label's
caption.
}
var
  S: array[0..20] of Char = 'A fUnNy StRiNg';

procedure TForm1.Button1Click(Sender: TObject);
var
  lower, upper : string;
begin
  lower := string(StrLower(S));
  upper := string(StrUpper(S));
  Canvas.TextOut(5, 10, lower + ' ' + upper);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.Caption := S;
end;
```

See Also

AnsiUpperCase (see page 4062)

UpperCase (see page 4311)

StrLower (see page 4264)

37.3.239 SysUtils.Supports Function

Indicates whether a given object or interface supports a specified interface.

Pascal

```
function Supports(const Instance: IInterface; const IID: TGUID; out Intf): Boolean; overload;
function Supports(const Instance: TObject; const IID: TGUID; out Intf): Boolean; overload;
function Supports(const Instance: IInterface; const IID: TGUID): Boolean; overload;
function Supports(const Instance: TObject; const IID: TGUID): Boolean; overload;
function Supports(const AClass: TClass; const IID: TGUID): Boolean; overload;
```

C++

```
Boolean Supports(const IInterface Instance, const TGUID IID,  Intf);
Boolean Supports(const TObject * Instance, const TGUID IID,  Intf);
Boolean Supports(const IInterface Instance, const TGUID IID);
Boolean Supports(const TObject * Instance, const TGUID IID);
Boolean Supports(const TClass AClass, const TGUID IID);
```

File

SysUtils

Description

Call Supports to determine whether the object or interface specified by Instance, or the class specified by AClass, supports the interface identified by the IID parameter. If Instance supports the interface, Supports returns the interface as the Intf parameter and returns true. If AClass supports the interface, Supports does not return an interface, but still returns true. If the interface specified by IID is not supported, Supports returns false.

On the Win32 platform, to determine whether the interface specified by IID is supported, Supports calls the QueryInterface method of the supplied interface (or the interface of the supplied object). However, unlike calling QueryInterface directly, Supports lets you pass a nil (Delphi) or NULL (C++) value as the instance.

On the .NET platform, Supports uses the typecasting facilities of the .NET framework.

See Also

TComponent

37.3.240 SysUtils.SysErrorMessage Function

Converts OS error codes into strings.

Pascal

```
function SysErrorMessage(ErrorCode: Cardinal): string;
```

C++

```
UnicodeString SysErrorMessage(unsigned ErrorCode);
```

File

SysUtils

Description

SysErrorMessage returns an error message string that corresponds to the specified OS error code.

See Also

RaiseLastOSError (see page 4230)

EWin32Error (see page 3949)

37.3.241 SysUtils.SystemTimeToDateTime Function

Converts a system time value into a TDateTime (see page 3323) value.

Pascal

```
function SystemTimeToDateTime(const SystemTime: TSystemTime): TDateTime;
```

C++

```
TDateTime SystemTimeToDateTime(const TSystemTime SystemTime);
```

File

SysUtils

Description

Call SystemTimeToDateTime to convert a value from Windows 32's TSystemTime (Delphi) or _SYSTEMTIME (C++) API format to a TDateTime (see page 3323) value. SystemTimeToDateTime is available on Windows only.

C++ Examples:

```
/*
The following code retrieves the current date and time using
the Windows GetLocalTime API call, and converts this value
to a TDateTime value.
*/
TDateTime __fastcall GetCurrentDateTime()
{
  SYSTEMTIME SystemTime;
  GetLocalTime(&SystemTime);
  return SystemTimeToDateTime(SystemTime);
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = DateTimeToStr(GetCurrentDateTime());
}
```

Delphi Examples:

```
{
The following code retrieves the current date and time using
the Windows GetLocalTime API call, and converts this value
to a TDateTime value.
}
function GetCurrentDateTime: TDateTime;
var
  SystemTime: TSystemTime;
begin
  GetLocalTime(SystemTime);
  Result := SysUtils.SystemTimeToDateTime(SystemTime);
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := SysUtils.DateTimeToStr(GetCurrentDateTime);
end;
```

See Also

DateTimeToSystemTime (see page 4091)

37.3.242 SysUtils.TextPos Function

Pascal

```
function TextPos(Str: PAnsiChar; SubStr: PAnsiChar): PAnsiChar; overload;
```

C++

```
PAnsiChar TextPos(PAnsiChar Str, PAnsiChar SubStr);
```

File

SysUtils

Description

This is function SysUtils.TextPos.

37.3.243 SysUtils.TextPos Function

Pascal

```
function TextPos(Str: PWideChar; SubStr: PWideChar): PWideChar; overload;
```

C++

```
PWideChar TextPos(PWideChar Str, PWideChar SubStr);
```

File

SysUtils

Description

This is function SysUtils.TextPos.

37.3.244 SysUtils.TextToFloat Function

Converts a null-terminated string to a floating-point value.

Pascal

```
function TextToFloat(Buffer: PAnsiChar; var Value; ValueType: TFloatValue): Boolean; overload;
function TextToFloat(Buffer: PAnsiChar; var Value; ValueType: TFloatValue; const
FormatSettings: TFormatSettings): Boolean; overload;
function TextToFloat(Buffer: PWideChar; var Value; ValueType: TFloatValue): Boolean; overload;
function TextToFloat(Buffer: PWideChar; var Value; ValueType: TFloatValue; const
FormatSettings: TFormatSettings): Boolean; overload;
```

C++

```
Boolean TextToFloat(PAnsiChar Buffer,  Value, TFloatValue ValueType);
Boolean TextToFloat(PAnsiChar Buffer,  Value, TFloatValue ValueType, const TFormatSettings
FormatSettings);
Boolean TextToFloat(PWideChar Buffer,  Value, TFloatValue ValueType);
Boolean TextToFloat(PWideChar Buffer,  Value, TFloatValue ValueType, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use TextToFloat to convert a null-terminated string, Buffer, to a floating-point value. The string must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The converted value is assigned to the Value parameter, which must be of type Extended or System::Currency (see page 3235), as indicated by the ValueType parameter.

TextToFloat returns true if the conversion is successful, or false if the Buffer does not contain a valid floating-point value.

The first form of TextToFloat is not thread-safe, because it uses localization information contained in global variables. The second form of TextToFloat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TextToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.245 SysUtils.Time Function

Returns the current time.

Pascal

```
function Time: TDateTime;
```

C++

```
TDateTime Time();
```

File

SysUtils

Description

Time and GetTime (see page 4209) return the current time as a TDateTime (see page 3323) value. The two functions are completely equivalent.

Note: Time clashes with an Xlib function of the same name. Use GetTime (see page 4209) in Linux and cross-platform code.

C++ Examples:

```
/*
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
```

```
  TDateTime DateTime = Time();  // store the current date and time
  AnsiString str = TimeToStr(DateTime); // convert the time to a string
  Caption = str;  // display the time on the form's caption
  // Note This could have been done with the following line of code:
  //  Caption = TimeToStr(Time());
}
```

Delphi Examples:

```
{
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
}
procedure TForm1.Timer1Timer(Sender: TObject);
var
  DateTime : TDateTime;
  str : string;
begin
  DateTime := Time;  // store the current date and time
  str := TimeToStr(DateTime); // convert the time into a string
  Caption := str;  // display the time on the form's caption
  { Note This could have been done with the following line of code:
    Caption := TimeToStr(Time); }
end;
```

See Also

Date (see page 4087)

DecodeTime (see page 4097)

Now (see page 4227)

TimeToStr (see page 4299)

37.3.246 SysUtils.TimeStampToDateTime Function

Converts a TTimeStamp (see page 4334) value into the corresponding TDateTime (see page 3323) value.

Pascal

```
function TimeStampToDateTime(const TimeStamp: TTimeStamp): TDateTime;
```

C++

```
TDateTime TimeStampToDateTime(const TTimeStamp TimeStamp);
```

File

SysUtils

Description

Call TimeStampToDateTime to convert a TTimeStamp (see page 4334) value into a TDateTime (see page 3323) value. TTimeStamp (see page 4334) values represent time as separate date and time values, where the date is the number of calendar days since the start of the current calendar (that is, January 1, 0001 would have a value of 1), and time is the number of milliseconds since midnight. TDateTime (see page 3323) values represent time as the number of days (including fractional days) that have elapsed since 12:00 am on December 30, 1899.

See Also

DateTimeToTimeStamp (see page 4092)

37.3.247 SysUtils.TimeStampToMSecs Function

Converts a TTimeStamp (see page 4334) value into an absolute number of milliseconds.

Pascal

```
function TimeStampToMSecs(const TimeStamp: TTimeStamp): Comp;
```

C++

```
Comp TimeStampToMSecs(const TTimeStamp TimeStamp);
```

File

SysUtils

Description

Call TimeStampToMSecs to convert a TTimeStamp (see page 4334) value, into the total number of milliseconds that comprise the Date (see page 4087) field plus the number of milliseconds given in the Time (see page 4297) field.

See Also

MSecsToTimeStamp (see page 4226)

TimeStampToDateTime (see page 4298)

37.3.248 SysUtils.TimeToStr Function

Returns a string that represents a TDateTime (see page 3323) value.

Pascal

```
function TimeToStr(const DateTime: TDateTime): string; overload;
function TimeToStr(const DateTime: TDateTime; const FormatSettings: TFormatSettings): string;
overload;
```

C++

```
UnicodeString TimeToStr(const TDateTime DateTime);
UnicodeString TimeToStr(const TDateTime DateTime, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

TimeToStr converts the Time (see page 4297) parameter, a TDateTime (see page 3323) value, to a string. The conversion uses the format specified by the LongTimeFormat (see page 4356) global variable or its TFormatSettings (see page 4326) equivalent. Change the format of the string by changing the values of some of the date and time variables.

The first form of TimeToStr is not thread-safe, because it uses localization information contained in global variables. The second form of TimeToStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TimeToStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

C++ Examples:

```
/*
```

```
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TDateTime dt;
  try
  {
    dt = EncodeTime((Word)StrToInt(Edit1->Text), (Word)StrToInt(Edit2->Text),
(Word)StrToInt(Edit3->Text), (Word)StrToInt(Edit4->Text));
    Label1->Caption = TimeToStr(dt);
  }
  catch(...)
  {
    ShowMessage("Invalid hour, minute, second, or millisecond.");
  }
}
/*
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
*/
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  TDateTime DateTime = Time();  // store the current date and time
  AnsiString str = TimeToStr(DateTime); // convert the time to a string
  Caption = str;  // display the time on the form's caption
  // Note This could have been done with the following line of code:
  //  Caption = TimeToStr(Time());
}
```

Delphi Examples:

```
{
This example uses three edit controls, a label, and a button
on a form. When the button is clicked, the day, month, and
year specified in the edit controls is stored in a TDateTime
object. Then the TDateTime object is converted to an
AnsiString that is displayed in the label.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  MyDate: TDateTime;
begin
  MyDate := SysUtils.EncodeTime(StrToInt(Edit1.Text), StrToInt(Edit2.Text),
StrToInt(Edit3.Text), StrToInt(Edit4.Text));
  Label1.Caption := TimeToStr(MyDate);
end;
{
This procedure displays the current time in the form's
caption every time the Timer's OnClick event fires, based upon
the Interval Property (default: 1000 = 1 sec)
}
procedure TForm1.Timer1Timer(Sender: TObject);
var
  DateTime : TDateTime;
  str : string;
begin
  DateTime := Time;  // store the current date and time
  str := TimeToStr(DateTime); // convert the time into a string
  Caption := str;  // display the time on the form's caption
  { Note This could have been done with the following line of code:
    Caption := TimeToStr(Time); }
end;
```

See Also

DateTimeToStr (see page 4089)

DateToStr (see page 4092)

StrToDateTime (see page 4281)

Time (see page 4297)

GetLocaleFormatSettings (see page 4206)

37.3.249 SysUtils.Trim Function

Trims leading and trailing spaces and control characters from a string.

Pascal

```
function Trim(const S: string): string; overload;
```

C++

```
UnicodeString Trim(const UnicodeString S);
```

File

SysUtils

Description

Trim removes leading and trailing spaces and control characters from the given string S.

In the 7-bit ASCII character set defined in ANSI X3.4-1977 (C0 and G0), "control codes" are defined as all characters whose code is between 0 and 31.

See Also

TrimLeft (see page 4301)

TrimRight (see page 4302)

37.3.250 SysUtils.TrimLeft Function

Trims leading spaces and control characters from a string.

Pascal

```
function TrimLeft(const S: string): string; overload;
```

C++

```
UnicodeString TrimLeft(const UnicodeString S);
```

File

SysUtils

Description

TrimLeft returns a copy of the string S with leading spaces and control characters removed.

See Also

Trim (see page 4301)

TrimRight (see page 4302)

37.3.251 SysUtils.TrimRight Function

Trims trailing spaces and control characters from a string.

Pascal

```
function TrimRight(const S: string): string; overload;
```

C++

```
UnicodeString TrimRight(const UnicodeString S);
```

File

SysUtils

Description

TrimRight returns a copy of the string S with trailing spaces and control characters removed.

See Also

TrimLeft (see page 4301)

Trim (see page 4301)

37.3.252 SysUtils.TryEncodeDate Function

Creates a TDateTime (see page 3323) value that represents a specified Year, Month, and Day.

Pascal

```
function TryEncodeDate(Year: Word; Month: Word; Day: Word; out Date: TDateTime): Boolean;
```

C++

```
Boolean TryEncodeDate(Word Year, Word Month, Word Day, TDateTime Date);
```

File

SysUtils

Description

TryEncodeDate generates a TDateTime (see page 3323) value from the values specified as the Year, Month, and Day parameters.

The year must be between 1 and 9999.

Valid Month values are 1 through 12.

Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month value. For example, the possible Day values for month 2 (February) are 1 through 28 or 1 through 29, depending on whether or not the Year value specifies a leap year.

The Date (see page 4087) parameter returns the generated value if TryEncodeDate returns true.

If the specified values are not within range, TryEncodeDate returns false.

See Also

DateToStr (see page 4092)

DecodeDate (see page 4095)

TryEncodeTime (see page 4303)

TryEncodeDateTime

EncodeDate (see page 4105)

37.3.253 SysUtils.TryEncodeTime Function

Creates a TDateTime (see page 3323) value for a specified Hour, Min, Sec, and MSec.

Pascal

```
function TryEncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word; out Time: TDateTime):
Boolean;
```

C++

```
Boolean TryEncodeTime(Word Hour, Word Min, Word Sec, Word MSec, TDateTime Time);
```

File

SysUtils

Description

EncodeTime (see page 4106) encodes the given hour, minute, second, and millisecond into a TDateTime (see page 3323) value.

Valid Hour values are 0 through 24. If Hour is 24, Min, Sec, and MSec must all be 0, and the resulting TDateTime (see page 3323) value represents midnight (12:00:00:000 AM) of the following day.

Valid Min and Sec values are 0 through 59.

Valid MSec values are 0 through 999.

The Time (see page 4297) paremter returns the generated TDateTime (see page 3323) value of the specifies values are in range. This value is a number between 0 and 1 (inclusive) that indicates the fractional part of a day given by the specified time or (if 1.0) midnight on the following day. The value 0 corresponds to midnight, 0.5 corresponds to noon, 0.75 corresponds to 6:00 pm, and so on.

See Also

DecodeTime (see page 4097)

TryEncodeDate (see page 4302)

TryEncodeDateTime

EncodeTime (see page 4106)

37.3.254 SysUtils.TryFloatToCurr Function

Converts a floating-point value to a System::Currency (see page 3235) value.

Pascal

```
function TryFloatToCurr(const Value: Extended; out AResult: Currency): Boolean;
```

C++

```
Boolean TryFloatToCurr(const Extended Value, Currency AResult);
```

File

SysUtils

Description

TryFloatToCurr converts the specified floating-point value into a System::Currency (see page 3235) value.

Value is the floating-point value to convert.

AResult returns the equivalent System::Currency (see page 3235) value, if possible.

If TryFloatToCurr is successful, it returns true. If Value is not within the valid range for a System::Currency (see page 3235) value, TryFloatToCurr returns false.

See Also

FloatToCurr (see page 4179)

37.3.255 SysUtils.TryFloatToDateTime Function

Converts a floating-point value to a TDateTime (see page 3323) value.

Pascal

```
function TryFloatToDateTime(const Value: Extended; out AResult: TDateTime): Boolean;
```

C++

```
Boolean TryFloatToDateTime(const Extended Value, TDateTime AResult);
```

File

SysUtils

Description

TryFloatToDateTime converts a floating-point value into the corresponding TDateTime (see page 3323) value.

Value is the floating-point value to convert.

AResult returns the corresponding TDateTime (see page 3323) value if possible.

TryFloatToDateTime returns true if the conversion was successful, false otherwise.

See Also

MinDateTime (see page 4384)

FloatToDateTime (see page 4179)

37.3.256 SysUtils.TryStrToBool Function

Converts a string to a Boolean value, with a Boolean success code.

Pascal

```
function TryStrToBool(const S: string; out Value: Boolean): Boolean; overload;
```

C++

```
Boolean TryStrToBool(const UnicodeString S, Boolean Value);
```

File

SysUtils

Description

TryStrToBool converts the string specified by S to a Boolean and sets Value to the result. If S can be converted to a number, TryStrToBool sets Value to false if that number is 0, true otherwise. If S can't be converted to a number, TryStrToBool returns true if S is any of the strings listed in TrueBoolStrs (see page 4363) (or differs from one only by case) and false if it is any of the strings listed in FalseBoolStrs (see page 4352) (or differs from one only by case). If S is not a number and not one of the strings listed in TrueBoolStrs (see page 4363) or FalseBoolStrs (see page 4352), TryStrToBool returns false to indicate that the conversion failed. If the conversion is successful, TryStrToBool returns true.

Note: If you do not assign a value to TrueBoolStrs (see page 4363), it is treated as if it contains the single string 'True'. Similarly, if you do not assign a value to FalseBoolStrs (see page 4352), it is treated as if it contains the single string 'False';

Note: The TrueBoolStrs (see page 4363) and FalseBoolStrs (see page 4352) variables are undefined when an application starts. Any use of StrToBool (see page 4276), BoolToStr (see page 4066) or TryStrToBool will cause these variables to be initialised with the default values above.

See Also

BoolToStr (see page 4066)

TrueBoolStrs (see page 4363)

FalseBoolStrs (see page 4352)

StrToBool (see page 4276)

37.3.257 SysUtils.TryStrToCurr Function

Converts a string to a System::Currency (see page 3235) value, with a Boolean success code.

Pascal

```
function TryStrToCurr(const S: string; out Value: Currency): Boolean; overload;
function TryStrToCurr(const S: string; out Value: Currency; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToCurr(const UnicodeString S, Currency Value);
Boolean TryStrToCurr(const UnicodeString S, Currency Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToCurr to convert a string that represents a floating-point value to the corresponding System::Currency (see page 3235) value. The source string, S, must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional 'E' or 'e' followed by a signed integer. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4350) global variable or its TFormatSettings (see page 4326) equivalent defines the character that is used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If the string doesn't contain a valid value, TryStrToCurr returns Default.

The first form of TryStrToCurr is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToCurr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToCurr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.258 SysUtils.TryStrToDate Function

Converts a string to a TDateTime (see page 3323) value, with a Boolean success code.

Pascal

```
function TryStrToDate(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToDate(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToDate(const UnicodeString S, TDateTime Value);
Boolean TryStrToDate(const UnicodeString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToDate to parse a string that specifies a date. If S does not contain a valid date, TryStrToDate returns false.

S must consist of two or three numbers, separated by the character defined by the DateSeparator (see page 4349) global variable or its TFormatSettings (see page 4326) equivalent. The order for month, day, and year is determined by the ShortDateFormat (see page 4358) global variable or its TFormatSettings (see page 4326) equivalent--possible combinations are m/d/y, d/m/y, and y/m/d.

If S contains only two numbers, it is interpreted as a date (m/d or d/m) in the current year.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4364). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3235) and Date (see page 4087)-Time (see page 4297) Formatting Variables" for more information.

The first form of TryStrToDate is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToDate, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToDate, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.259 SysUtils.TryStrToDateTime Function

Converts a string to a TDateTime (see page 3323) value with a Boolean success code.

Pascal

```
function TryStrToDateTime(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToDateTime(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToDateTime(const UnicodeString S, TDateTime Value);
Boolean TryStrToDateTime(const UnicodeString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToDateTime to parse a string that specifies a date and time value. If S does not contain a valid date, TryStrToDateTime returns false.

The S parameter must use the current locale's date/time format. In the US, this is commonly MM/DD/YY HH:MM:SS format. Specifying AM or PM as part of the time is optional, as are the seconds. Use 24-hour time (7:45 PM is entered as 19:45, for example) if AM or PM is not specified.

Year values between 0 and 99 are converted using the TwoDigitYearCenturyWindow (see page 4364). This value is stored either in a global variable (first form) or as a field in the FormatSettings parameter (second form) See "Currency (see page 3235) and Date (see page 4087)-Time (see page 4297) Formatting Variables" for more information.

The first form of TryStrToDateTime is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToDateTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToDateTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.260 SysUtils.TryStrToFloat Function

Converts a given string to a floating-point value with a boolean success code.

Pascal

```
function TryStrToFloat(const S: string; out Value: Extended): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Extended; const FormatSettings:
TFormatSettings): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Double): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Double; const FormatSettings:
TFormatSettings): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Single): Boolean; overload;
function TryStrToFloat(const S: string; out Value: Single; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToFloat(const UnicodeString S, Extended Value);
Boolean TryStrToFloat(const UnicodeString S, Extended Value, const TFormatSettings
```

```
FormatSettings);
Boolean TryStrToFloat(const UnicodeString S, Double Value);
Boolean TryStrToFloat(const UnicodeString S, Double Value, const TFormatSettings
FormatSettings);
Boolean TryStrToFloat(const UnicodeString S, Single Value);
Boolean TryStrToFloat(const UnicodeString S, Single Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Use TryStrToFloat to convert a astring, S, to a floating-point value. S must consist of an optional sign (+ or -), a string of digits with an optional decimal point, and an optional mantissa. The mantissa consists of 'E' or 'e' followed by an optional sign (+ or -) and a whole number. Leading and trailing blanks are ignored.

The DecimalSeparator (see page 4350) global variable defines the character that must be used as a decimal point. Thousand separators and currency symbols are not allowed in the string. If S doesn't contain a valid value, TryStrToFloat returns Default.

TryStrToFloat places the result in Value, and returns true if the conversion was successful, false otherwise.

The first three forms of TryStrToFloat are not thread-safe, because they use localization information contained in global variables. The second three forms of TryStrToFloat, which are thread-safe, refer to localization information contained in the FormatSettings parameter. Before calling the thread-safe forms of TryStrToFloat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.261 SysUtils.TryStrToInt Function

Converts a string that represents an integer (decimal or hex notation) to a number with Boolean success code.

Pascal

```
function TryStrToInt(const S: string; out Value: Integer): Boolean; overload;
```

C++

```
Boolean TryStrToInt(const UnicodeString S, int Value);
```

File

SysUtils

Description

TryStrToInt converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number, which is assigned to Value. If S does not represent a valid number, TryStrToInt returns false; otherwise TryStrToInt returns true.

To accept decimal but not hexadecimal values in the input string, you may use code like this:

```
function TryDecimalStrToInt( const S: string; out Value: Integer): Boolean;
begin
   result := ( pos( '$', S ) = 0 ) and TryStrToInt( S, Value );
end;
```

See Also

IntToHex (see page 4215)

IntToStr (see page 4216)

StrToInt (see page 4286)

StrToIntDef (see page 4288)

37.3.262 SysUtils.TryStrToInt64 Function

Converts a string that represents an integer (decimal or hex notation) to a number.

Pascal

```
function TryStrToInt64(const S: string; out Value: Int64): Boolean; overload;
```

C++

```
Boolean TryStrToInt64(const UnicodeString S, Int64 Value);
```

File

SysUtils

Description

TryStrToInt64 converts the string S, which represents an integer-type number in either decimal or hexadecimal notation, into a number, which is assigned to Value. If S does not represent a valid number, TryStrToInt64 returns false; otherwise TryStrToInt64 returns true.

To accept decimal but not hexadecimal values in the input string, you may use code like this:

```
function TryDecimalStrToInt( const S: string; out Value: Integer): Boolean;
begin
   result := ( pos( '$', S ) = 0 ) and TryStrToInt( S, Value );
end;
```

See Also

IntToStr (see page 4216)

StrToInt (see page 4286)

StrToInt64 (see page 4287)

37.3.263 SysUtils.TryStrToTime Function

Converts a string to a TDateTime (see page 3323) value with an error default.

Pascal

```
function TryStrToTime(const S: string; out Value: TDateTime): Boolean; overload;
function TryStrToTime(const S: string; out Value: TDateTime; const FormatSettings:
TFormatSettings): Boolean; overload;
```

C++

```
Boolean TryStrToTime(const UnicodeString S, TDateTime Value);
Boolean TryStrToTime(const UnicodeString S, TDateTime Value, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

Call TryStrToTime to parse a string that specifies a time value. If S does not contain a valid time, TryStrToTime returns false.

The S parameter must consist of two or three numbers, separated by the character defined by the TimeSeparator (see page 4363) global variable or its TFormatSettings (see page 4326) equivalent, optionally followed by an AM or PM indicator. The numbers represent hour, minute, and (optionally) second, in that order. If the time is followed by AM or PM, it is assumed to be in 12-hour clock format. If no AM or PM indicator is included, the time is assumed to be in 24-hour clock format.

The first form of TryStrToTime is not thread-safe, because it uses localization information contained in global variables. The second form of TryStrToTime, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of TryStrToTime, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.264 SysUtils.UIntToStr Function

Pascal

```
function UIntToStr(Value: Cardinal): string; overload;
```

C++

```
UnicodeString UIntToStr(unsigned Value);
```

File

SysUtils

Description

This is function SysUtils.UIntToStr.

37.3.265 SysUtils.UIntToStr Function

Pascal

```
function UIntToStr(Value: UInt64): string; overload;
```

C++

```
UnicodeString UIntToStr(UInt64 Value);
```

File

SysUtils

Description

This is function SysUtils.UIntToStr.

37.3.266 SysUtils.UnloadPackage Function

Unloads a package.

Pascal

```
procedure UnloadPackage(Module: HMODULE);
```

C++

```
UnloadPackage(HMODULE Module);
```

File

SysUtils

Description

Call UnloadPackage to unload a package that was loaded using the LoadPackage (see page 4224) function. UnloadPackage performs any finalization of units contained in the package and then unloads the package BPL.

See Also

LoadPackage (see page 4224)

37.3.267 SysUtils.UpperCase Function

Returns a copy of a string in uppercase.

Pascal

```
function UpperCase(const S: string): string; overload;
function UpperCase(const S: string; LocaleOptions: TLocaleOptions): string; overload;
```

C++

```
UnicodeString UpperCase(const UnicodeString S);
UnicodeString UpperCase(const UnicodeString S, TLocaleOptions LocaleOptions);
```

File

SysUtils

Description

UpperCase returns a copy of the string S, with the same text but with all 7-bit ASCII characters between 'a' and 'z' converted to uppercase. To convert 8-bit international characters, use AnsiUpperCase (see page 4062) instead.

C++ Examples:

```
/*
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int I = 0; I < ListBox1->Items->Count; I++)
    ListBox1->Items->Strings[I] = UpperCase(ListBox1->Items->Strings[I]);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");
```

```
}
```

Delphi Examples:

```
{
This example uses a list box and a button on a form. Use the
Items property editor in the Object Inspector to enter a
list of strings in the list box. When the application runs
and the button is clicked, the strings in the list box
become uppercase.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to ListBox1.Items.Count -1 do
    ListBox1.Items[I] := SysUtils.UpperCase(ListBox1.Items[I]);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  Listbox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;
```

See Also

AnsiUpperCase (see page 4062)

LowerCase (see page 4225)

37.3.268 SysUtils.WideBytesOf Function

Converts an Unicode string to an array of bytes.

Pascal

```
function WideBytesOf(const Value: UnicodeString): TBytes;
```

C++

```
TBytes WideBytesOf(const UnicodeString Value);
```

File

SysUtils

Description

Call WideBytesOf to convert an Unicode string to an array of bytes. Strings are converted using the UTF-16 encoding, represented by the Unicode property of the TEncoding (see page 3969) class.

Unlike the BytesOf (see page 4068) routine, the resulting array of bytes contains two bytes per character in the input string.

See Also

BytesOf (see page 4068)

StringCodePage (see page 3676)

TEncoding (see page 3969)

37.3.269 SysUtils.WideCompareStr Function

Compares Unicode strings based on the current locale with case sensitivity.

Pascal

```
function WideCompareStr(const S1: WideString; const S2: WideString): Integer;
```

C++

```
int WideCompareStr(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideCompareStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is:

Condition	Return Value
S1 > S2	> 0
S1 < S2	< 0
S1 = S2	= 0

Note: Most locales consider lowercase characters to be less than the corresponding uppercase characters. This is in contrast to ASCII order, in which lowercase characters are greater than uppercase characters. with S1 set to 'a' and S2 set to 'A', WideCompareStr returns a value lsess than zero, while CompareStr (see page 4078) returns a value greater than zero.

See Also

WideCompareText (see page 4313)

WideSameStr (see page 4316)

37.3.270 SysUtils.WideCompareText Function

Compares Unicode strings based on the current locale without case sensitivity.

Pascal

```
function WideCompareText(const S1: WideString; const S2: WideString): Integer;
```

C++

```
int WideCompareText(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideCompareText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. WideCompareText returns a value less than 0 if S1 < S2, a value greater than 0 if S1 > S2, and returns 0 if S1 = S2.

See Also

WideCompareStr (see page 4313)

WideSameText (see page 4316)

37.3.271 SysUtils.WideFmtStr Function

Assembles a formatted Unicode string using a format string and an array of arguments.

Pascal

```
procedure WideFmtStr(var Result: WideString; const Format: WideString; const Args: array of
const); overload;
procedure WideFmtStr(var Result: WideString; const Format: WideString; const Args: array of
const; const FormatSettings: TFormatSettings); overload;
```

C++

```
WideFmtStr(BSTR Result, const BSTR Format, const array of const Args);
WideFmtStr(BSTR Result, const BSTR Format, const array of const Args, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args into a Unicode string. Formatting is controlled by the format string Format (see page 4188); the results are returned in the parameter Result.

For information on the format strings, see Format (see page 4188) Strings.

The first form of WideFmtStr is not thread-safe, because it uses localization information contained in global variables. The second form of WideFmtStr, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFmtStr, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

GetLocaleFormatSettings (see page 4206)

37.3.272 SysUtils.WideFormat Function

Returns a formatted Unicode string assembled from a format string and an array of arguments.

Pascal

```
function WideFormat(const Format: WideString; const Args: array of const): WideString;
overload;
function WideFormat(const Format: WideString; const Args: array of const; const
FormatSettings: TFormatSettings): WideString; overload;
```

C++

```
BSTR WideFormat(const BSTR Format, const array of const Args);
BSTR WideFormat(const BSTR Format, const array of const Args, const TFormatSettings
FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the format string Format (see page 4188); the results are returned in the function result as a Unicode string.

For information on the format strings, see Format (see page 4188) Strings.

The first form of WideFormat is not thread-safe, because it uses localization information contained in global variables. The second form of WideFormat, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFormat, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

See Also

GetLocaleFormatSettings (see page 4206)

37.3.273 SysUtils.WideFormatBuf Function

Formats the arguments from an array, placing the result in a buffer.

Pascal

```
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const): Cardinal; overload;
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Format; FmtLen: Cardinal; const
Args: array of const; const FormatSettings: TFormatSettings): Cardinal; overload;
```

C++

```
unsigned WideFormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array
of const Args);
unsigned WideFormatBuf( Buffer, unsigned BufLen, const  Format, unsigned FmtLen, const array
of const Args, const TFormatSettings FormatSettings);
```

File

SysUtils

Description

This function formats the series of arguments in the open array Args. Formatting is controlled by the Unicode format string Format (see page 4188) (whose length is given by FmtLen); the results are returned in Buffer (whose length is given by BufLen). The function returns the number of Unicode characters in the formatted string.

Note: Both buffers contain wide (Unicode) characters. BufLen and FmtLen must specify the number of Unicode characters in their respective buffers, not the number of bytes.

For information on the format strings, see Format (see page 4188) Strings.

The first form of WideFormatBuf is not thread-safe, because it uses localization information contained in global variables. The second form of WideFormatBuf, which is thread-safe, refers to localization information contained in the FormatSettings parameter. Before calling the thread-safe form of WideFormatBuf, you must populate FormatSettings with localization information. To populate FormatSettings with a set of default locale values, call GetLocaleFormatSettings (see page 4206).

37.3.274 SysUtils.WideLowerCase Function

Returns Unicode string converted to lower case.

Pascal

```
function WideLowerCase(const S: WideString): WideString;
```

C++

```
BSTR WideLowerCase(const BSTR S);
```

File

SysUtils

Description

WideLowerCase returns S converted to lower case. The conversion uses the current locale.

See Also

WideUpperCase (see page 4318)

LowerCase (see page 4225)

37.3.275 SysUtils.WideSameStr Function

Compares Unicode strings based on the current locale with case sensitivity.

Pascal

```
function WideSameStr(const S1: WideString; const S2: WideString): Boolean;
```

C++

```
Boolean WideSameStr(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideSameStr compares S1 to S2, with case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, Ffalse otherwise.

See Also

WideSameText (see page 4316)

WideCompareStr (see page 4313)

37.3.276 SysUtils.WideSameText Function

Compares Unicode strings based on the current locale without case sensitivity.

Pascal

```
function WideSameText(const S1: WideString; const S2: WideString): Boolean;
```

C++

```
Boolean WideSameText(const BSTR S1, const BSTR S2);
```

File

SysUtils

Description

WideSameText compares S1 to S2, without case sensitivity. The compare operation is controlled by the current locale. The return value is true if the two strings have the same value, false otherwise.

See Also

WideSameStr (see page 4316)

WideCompareText (see page 4313)

37.3.277 SysUtils.WideStrAlloc Function

Pascal

```
function WideStrAlloc(Size: Cardinal): PWideChar;
```

C++

```
PWideChar WideStrAlloc(unsigned Size);
```

File

SysUtils

Description

This is function SysUtils.WideStrAlloc.

37.3.278 SysUtils.WideStringOf Function

Pascal

```
function WideStringOf(const Value: TBytes): UnicodeString;
```

C++

```
UnicodeString WideStringOf(const TBytes Value);
```

File

SysUtils

Description

This is function SysUtils.WideStringOf.

37.3.279 SysUtils.WideUpperCase Function

Returns Unicode string converted to upper case.

Pascal

```
function WideUpperCase(const S: WideString): WideString;
```

C++

```
BSTR WideUpperCase(const BSTR S);
```

File

SysUtils

Description

WideUpperCase returns S converted to upper case. The conversion uses the current locale.

See Also

WideLowerCase (see page 4316)

UpperCase (see page 4311)

37.3.280 SysUtils.Win32Check Function

Checks the return value of a Windows API call and raises an appropriate exception when it indicates failure.

Pascal

```
function Win32Check(RetVal: BOOL): BOOL; platform;
```

C++

```
BOOL Win32Check(BOOL RetVal);
```

File

SysUtils

Description

Call Win32Check with the return value of a Windows API call that returns a Boolean to indicate success or failure. If the Windows API returns false (indicating failure), Win32Check calls RaiseLastOSError (see page 4230) to raise an exception. If the Windows function returns true, Win32Check returns true.

Win32Check is available on Windows only.

See Also

RaiseLastOSError (see page 4230)

37.3.281 SysUtils.WrapText Function

Splits a string into multiple lines as its length approaches a specified size.

Pascal

```
function WrapText(const Line: string; const BreakStr: string; const BreakChars: TSysCharSet;
MaxCol: Integer): string; overload;
function WrapText(const Line: string; MaxCol: Integer = 45): string; overload;
```

C++

```
UnicodeString WrapText(const UnicodeString Line, const UnicodeString BreakStr, const
TSysCharSet BreakChars, int MaxCol);
UnicodeString WrapText(const UnicodeString Line, int MaxCol = 45);
```

File

SysUtils

Description

WrapText scans a string for occurrences of any of the characters specified by nBreakChars and inserts a line-break, specified by BreakStr, at the last occurrence of a character in nBreakChars before MaxCol. Line is the text WrapText scans. MaxCol is the maximum line length.

If the BreakStr and nBreakChars parameters are omitted, WrapText searches for space, hyphen, or tab characters on which to break the line and inserts a carriage return/line feed pair (Windows) or a linefeed (Linux) at the break points.

WrapText does not insert a break into an embedded quoted string (both single quotes and double quotes are supported).

For example, the following call wraps the text into two lines at the last space character:

WrapText('The rain in Spain falls mainly on the plain.', #13#10, ['.',' ',#9,'-'], 42);

TSysCharSet (see page 4344) bchars

bchars << '.' << '-' << ' ';

WrapText("The rain in Spain falls mainly on the plain.", "\r\n", bchars, 42);

The result:

The rain in Spain falls mainly on the

plain.

37.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFilenameCaseMatch (see page 4323)	TFilenameCaseMatch indicates how close a match was found to a specified file name.
	TFloatFormat (see page 4323)	TFloatFormat is an enumerated list of formatting codes for float functions.
	TFloatValue (see page 4325)	TFloatValue is an enumeration of two basic floating point types.
	TLocaleOptions (see page 4328)	TLocaleOptions defines a choice of dependent and independent locale options.
	TMbcsByteType (see page 4329)	TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).
	TNameType (see page 4329)	This is record SysUtils.TNameType.

Records

	Name	Description
	Int64Rec (see page 4320)	Int64Rec declares a utility record to provide access to the bytes of an Int64 value.
	LongRec (see page 4321)	LongRec declares a utility record to provide access to the bytes of a LongWord value.
	TCharSearch (see page 4321)	This is record SysUtils.TCharSearch.
	TExceptionRecord (see page 4322)	TExceptionRecord holds information about an exception in an application.
	TFloatRec (see page 4325)	TFloatRec declares a utility record that provides access to the attributes of a floating point number.
	TFormatSettings (see page 4326)	TFormatSettings defines a thread-safe string formatting context.
	TLangRec (see page 4327)	TLangRec declares a record used to describe a Windows locale.
	TSearchRec (see page 4330)	TSearchRec defines file information searched for by FindFirst (see page 4175) or FindNext (see page 4177).
	TSysLocale (see page 4333)	TSysLocale represents locale information.
	TThreadInfo (see page 4334)	This is record SysUtils.TThreadInfo.
	TTimeStamp (see page 4334)	TTimeStamp represents time and date values.
	WordRec (see page 4335)	WordRec declares a utility record to store high and low order bytes of a variable.

37.4.1 SysUtils.Int64Rec Record

Int64Rec declares a utility record to provide access to the bytes of an Int64 value.

Pascal

```
Int64Rec = packed record
  case Integer of
    0: (Hi: Cardinal;
        Lo: Cardinal;);
    1: (Cardinals: array [0..1] of Cardinal);
        Words: array [0..3] of Word);
        Bytes: array [0..7] of Byte););
end;
```

C++

```
struct Int64Rec {
  unsigned Hi;
  unsigned Lo;
  array [0..1] of Cardinal) Cardinals;
```

```
  array [0..3] of Word) Words;
  array [0..7] of Byte) Bytes;
};
```

File

SysUtils

Description

The Int64Rec type declares a utility record that stores 8 contiguous (packed) bytes of data. Int64Rec is used primarily for typecasting, where Int64 data needs to be dissected.

The 8 bytes may be accessed individually via the `Bytes` array, or as an array of 4 `Words`, or an array of 2 `Cardinals`. And finally, the value may be accessed via the `Hi` and `Lo` order Cardinals.

37.4.2 SysUtils.LongRec Record

LongRec declares a utility record to provide access to the bytes of a LongWord value.

Pascal

```
LongRec = packed record
  case Integer of
    0: (Hi: Word;
        Lo: Word;);
    1: (Words: array [0..1] of Word);
        Bytes: array [0..3] of Byte););
end;
```

C++

```
struct LongRec {
  Word Hi;
  Word Lo;
  array [0..1] of Word) Words;
  array [0..3] of Byte) Bytes;
};
```

File

SysUtils

Description

The LongRec type declares a utility record that stores 4 contiguous (packed) bytes of data. LongRec is used primarily for typecasting, where LongWord data needs to be dissected.

The 4 bytes may be accessed individually via the `Bytes` array, or as an array of 2 `Words`, or the value may be accessed via the `Hi` and `Lo` order Words.

37.4.3 SysUtils.TCharSearch Record

Pascal

```
TCharSearch = record
  ArrayPtr: PChar;
  MatchPtr: PChar;
end;
```

C++

```
struct TCharSearch {
  const char * ArrayPtr;
  const char * MatchPtr;
};
```

File

SysUtils

Description

This is record SysUtils.TCharSearch.

37.4.4 SysUtils.TExceptionRecord Record

TExceptionRecord holds information about an exception in an application.

Pascal

```
TExceptionRecord = record
  ExceptionCode: Cardinal;
  ExceptionFlags: Cardinal;
  ExceptionRecord: PExceptionRecord;
  ExceptionAddress: Pointer;
  NumberParameters: Cardinal;
  case Boolean of
    True: (ExceptionInformation: array [0..14] of Longint);
          False: (ExceptAddr: Pointer; ExceptObject: Pointer;);
end;
```

C++

```
struct TExceptionRecord {
  unsigned ExceptionCode;
  unsigned ExceptionFlags;
  PExceptionRecord ExceptionRecord;
  void * ExceptionAddress;
  unsigned NumberParameters;
  array [0..14] of Longint) ExceptionInformation;
  (ExceptAddr: Pointer; ExceptObject: Pointer False;
};
```

File

SysUtils

Description

TExceptionRecord holds information about an exception in an application. It is furnished by the Delphi exception handling process.

The record comprises the following elements :

Element	Meaning
ExceptionCode	The exception code, such as EXCEPTION_INT_OVERFLOW.
ExceptionFlags	Comprises one flag : EXCEPTION_NONCONTINUABLE_EXCEPTION. If 0, then the exception is allowed to continue.
ExceptionRecord	Pointer to subsequent exception records, if any.
ExceptionAddress	Address where the exception occured.

`NumberParameters`	Number of elements in the `ExceptionInformation` array.
`ExceptionInformation`	An array of numbers that describe the exception.

37.4.5 SysUtils.TFilenameCaseMatch Enumeration

TFilenameCaseMatch indicates how close a match was found to a specified file name.

Pascal

```
TFilenameCaseMatch = (
  mkNone,
  mkExactMatch,
  mkSingleMatch,
  mkAmbiguous
);
```

C++

```
enum TFilenameCaseMatch {
  mkNone,
  mkExactMatch,
  mkSingleMatch,
  mkAmbiguous
};
```

File

SysUtils

Description

These are the possible values of the TFilenameCaseMatch type:

Value	Meaning
mkNone	No actual file matches the file name.
mkExactMatch	A single file matches the file name.
mkSingleMatch	No file matches the file name, but a single match can be found by ignoring differences in case.
mkAmbiguous	No file matches the file name, but two or more matches can be found by ignoring differences in case.

37.4.6 SysUtils.TFloatFormat Enumeration

TFloatFormat is an enumerated list of formatting codes for float functions.

Pascal

```
TFloatFormat = (
  ffGeneral,
  ffExponent,
  ffFixed,
  ffNumber,
  ffCurrency
);
```

C++

```
enum TFloatFormat {
  ffGeneral,
  ffExponent,
  ffFixed,
  ffNumber,
  ffCurrency
};
```

File

SysUtils

Description

TFloatFormat defines an enumerated list of formatting codes for the float functions.

Format (see page 4188)	Defines
ffGeneral	General number format. The value is converted to the shortest possible decimal string using scientific format. Trailing zeros are removed from the resulting string.
ffExponent	Scientific format. The value is converted to a string of the form "-d.ddd...E+dddd". The resulting string starts with a minus sign if the number is negative, and one digit always precedes the decimal point. The total number of digits before the exponent in the resulting string (including the one before the decimal point) is given by the Precision parameter. The "E" exponent character in the resulting string is always followed by a plus or minus sign and up to four digits. The Digits parameter specifies the minimum number of digits in the exponent (between 0 and 4).
ffFixed	Fixed point format. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative, and at least one digit always precedes the decimal point. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18. If the number of digits to the left of the decimal point is greater than the specified precision, the resulting value will use scientific format. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffFixed with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.
ffNumber	Number format. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The ffNumber format corresponds to the ffFixed format, except that the resulting string contains thousandth separators. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffNumber with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.
ffCurrency	Currency (see page 3235) format. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString (see page 4349), CurrencyFormat (see page 4348), NegCurrFormat (see page 4356), ThousandSeparator (see page 4362), and DecimalSeparator (see page 4350) global variables, all of which are initialized from the System::Currency (see page 3235) Format (see page 4188) in the International section of the Windows Control Panel. The number of digits after the decimal point is given by the Digits parameter--it must be between 0 and 18. The resulting values is padded with zeros when the Digits parameter is higher than the number of digits dictated by precision. For example, given ffCurrency with Precision set to 5 and digits set to 3, fomatting 345.6789 gives the string '345.680', padding with zeros after the precision is met.

See Also

FloatToDecimal (see page 4180)

FloatToStr (see page 4180)

FloatToStrF (see page 4182)

FloatToText (see page 4183)

FloatToTextFmt (see page 4184)

37.4.7 SysUtils.TFloatRec Record

TFloatRec declares a utility record that provides access to the attributes of a floating point number.

Pascal

```
TFloatRec = packed record
  Exponent: Smallint;
  Negative: Boolean;
  Digits: array[0..20] of AnsiChar;
end;
```

C++

```
struct TFloatRec {
  Smallint Exponent;
  Boolean Negative;
  array[0..20] of AnsiChar Digits;
};
```

File

SysUtils

Description

The TFloatRec type declares a record that holds the attributes of a floating point number. The record makes the attributes accessible for easy formatting or specialised numerical processing.

`Exponent` holds the floating point number exponent value.

`Negative` is set true if the floating point number has a negative value.

`Digits` is a 21 element array of numeric characters representing the floating point number mantissa value : its significant digits.

TFloatRec is used by the FloatToDecimal (see page 4180) function to hold the results of the conversion of a floating point value to its decimal representation.

See Also

FloatToDecimal (see page 4180)

37.4.8 SysUtils.TFloatValue Enumeration

TFloatValue is an enumeration of two basic floating point types.

Pascal

```
TFloatValue = (
  fvExtended,
  fvCurrency
);
```

C++

```
enum TFloatValue {
  fvExtended,
  fvCurrency
};
```

File

SysUtils

Description

TFloatValue is used by a number of data conversion routines, where it is used to indicate what type of floating point number is being handled.

These are the possible values of the TFloatValue type:

Value	Meaning
fvExtended	An extended floating point number. These are the highest capacity floating point numbers in Delphi : they support the largest mantissa and exponent sizes.
fvDecimal	A decimal floating point number. Decimals have a fixed exponent size of 4 digits.

See Also

FloatToDecimal (see page 4180)

FloatToText (see page 4183)

FloatToTextFmt (see page 4184)

TextToFloat (see page 4296)

37.4.9 SysUtils.TFormatSettings Record

TFormatSettings defines a thread-safe string formatting context.

Pascal

```
TFormatSettings = record
  CurrencyFormat: Byte;
  NegCurrFormat: Byte;
  ThousandSeparator: Char;
  DecimalSeparator: Char;
  CurrencyDecimals: Byte;
  DateSeparator: Char;
  TimeSeparator: Char;
  ListSeparator: Char;
  CurrencyString: string;
  ShortDateFormat: string;
  LongDateFormat: string;
  TimeAMString: string;
  TimePMString: string;
  ShortTimeFormat: string;
  LongTimeFormat: string;
  ShortMonthNames: array[1..12] of string;
  LongMonthNames: array[1..12] of string;
  ShortDayNames: array[1..7] of string;
  LongDayNames: array[1..7] of string;
  TwoDigitYearCenturyWindow: Word;
end;
```

C++

```
struct TFormatSettings {
  Byte CurrencyFormat;
  Byte NegCurrFormat;
  Char ThousandSeparator;
```

```
    Char DecimalSeparator;
    Byte CurrencyDecimals;
    Char DateSeparator;
    Char TimeSeparator;
    Char ListSeparator;
    UnicodeString CurrencyString;
    UnicodeString ShortDateFormat;
    UnicodeString LongDateFormat;
    UnicodeString TimeAMString;
    UnicodeString TimePMString;
    UnicodeString ShortTimeFormat;
    UnicodeString LongTimeFormat;
    array[1..12] of string ShortMonthNames;
    array[1..12] of string LongMonthNames;
    array[1..7] of string ShortDayNames;
    array[1..7] of string LongDayNames;
    Word TwoDigitYearCenturyWindow;
  };
```

File

SysUtils

Description

TFormatSettings defines a data structure containing locale information used by string formatting routine. Each member of TFormatSettings is equivalent to the global variable with the same name. A variable of type TFormatSettings defines a thread-safe context that formatting functions can use in place of the default global context, which is not thread-safe.

To create and use the thread-safe environment defined by TFormatSettings, follow these steps:

Define a variable of type TFormatSettings

Call GetLocaleFormatSettings (see page 4206) to populate the TFormatSettings variable with locale information.

Pass the TFormatSettings variable as the last parameter of the string formatting routine.

Each routine that accepts a TFormatSettings parameter is thread-safe, and is an overload of an equivalent function that refers to the global formatting variables.

37.4.10 SysUtils.TLangRec Record

TLangRec declares a record used to describe a Windows locale.

Pascal

```
  TLangRec = packed record
    FName: string;
    FLCID: LCID;
    FExt: string;
  end;
```

C++

```
  struct TLangRec {
    UnicodeString FName;
    LCID FLCID;
    UnicodeString FExt;
  };
```

File

SysUtils

Description

The TLangRec type declares a record that holds the attributes of a Windows language. The record is principally used internally by TLanguages (see page 3987).

`FName` holds the Windows locale name.

`FLCID` holds the local id for the Windows locale.

`FExt` holds the recommended filename extension for the Windows locale.

See Also

TLanguages (see page 3987)

37.4.11 SysUtils.TLocaleOptions Enumeration

TLocaleOptions defines a choice of dependent and independent locale options.

Pascal

```
TLocaleOptions = (
  loInvariantLocale,
  loUserLocale
);
```

C++

```
enum TLocaleOptions {
  loInvariantLocale,
  loUserLocale
};
```

File

SysUtils

Description

The TLocaleOptions type is an enumeration of two choices for dealing with the operating locale of an application. This Microsoft Windows specific type is used in a number of string handling routines, where the locale language may have a bearing on the routine.

Format (see page 4188)	Defines
loInvariantLocale	Use this option where a consistent inter-application operation is desired, inedependent of the actual locale.
loUserLocale	Use this option where the user locale is taken into account.

See Also

CompareStr (see page 4078)

CompareText (see page 4079)

LowerCase (see page 4225)

SameStr (see page 4239)

SameText (see page 4239)

UpperCase (see page 4311)

37.4.12 SysUtils.TMbcsByteType Enumeration

TMbcsByteType represents the use of a single byte in a string that uses a multi-byte character set (MBCS).

Pascal

```
TMbcsByteType = (
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
);
```

C++

```
enum TMbcsByteType {
  mbSingleByte,
  mbLeadByte,
  mbTrailByte
};
```

File

SysUtils

Description

TMbcsByteType represents the possible return values of the ByteType (see page 4070) and StrByteType (see page 4242) functions. Possible values are as follows:

Values	Meaning
mbSingleByte	The char is used to represent an entire character in the string. The value of Char can not be included in the LeadBytes (see page 4354) set.
mbLeadByte	The char is used to represent the first byte of a multi-byte character. The value of Char must be included in the LeadBytes (see page 4354) set.
mbTrailByte	The char is used to represent one of the trailing bytes in a multi-byte character. There are no a priori restrictions on the value of char.

See Also

ByteType (see page 4070)

LeadBytes (see page 4354)

StrByteType (see page 4242)

37.4.13 SysUtils.TNameType Enumeration

Pascal

```
TNameType = (
  ntContainsUnit,
  ntRequiresPackage,
  ntDcpBpiName
);
```

C++

```
enum TNameType {
  ntContainsUnit,
  ntRequiresPackage,
  ntDcpBpiName
};
```

File

SysUtils

Description

This is record SysUtils.TNameType.

37.4.14 SysUtils.TSearchRec Record

TSearchRec defines file information searched for by FindFirst (see page 4175) or FindNext (see page 4177).

Pascal

```
TSearchRec = record
  Time: Integer;
  Size: Int64;
  Attr: Integer;
  Name: TFileName;
  ExcludeAttr: Integer;
  FindHandle: THandle platform;
  FindData: TWin32FindData platform;
end;
```

C++

```
struct TSearchRec {
  int Time;
  Int64 Size;
  int Attr;
  TFileName Name;
  int ExcludeAttr;
  THandle platform FindHandle;
  TWin32FindData platform FindData;
};
```

File

SysUtils

Description

The TSearchRec type defines file information searched for by a FindFirst (see page 4175) or FindNext (see page 4177) function call. If a file is found, the fields of the TSearchRec type parameter are modified to specify the found file.

Attr represents the file attributes of the file. Test Attr against the following attribute constants or values to determine if a file attribute matches the file's properties.

Note: On Windows, attribute constants map directly to DOS file attributes.

Note: On Linux, attribute constants map to sets of file characteristics; use the Mode field to access the specific permission mode.

Constant	Value	Description	Linux Meaning
faReadOnly (see page 4374)	1	Read (see page 3634)-only files	Current user does not have write access.
faHidden (see page 4373)	2	Hidden files	File name begins with ".".
faSysFile (see page 4375)	4	System (see page 3159) files	File is socket, symbolic link, device file, or FIFO.
faVolumeID (see page 4376)	8	Volume ID files	Not used.
faDirectory (see page 4372)	16	Directory files	Directory.
faArchive (see page 4372)	32	Archive files	Not used.
faSymLink (see page 4375)	64	Symbolic link	File is a symbolic link.
faAnyFile (see page 4371)	71	Any file	Any file.

The faReadOnly (see page 4374) constant has the same name as the enumerated value that is defined by the TFieldAttribute type. If both the SysUtils (see page 3901) and the Db units are used in your source files, you must disambiguate by specifying the unit to qualify the use of faReadOnly (see page 4374). That is, write SysUtils.faReadOnly (see page 4374) (Delphi) or SysUtils::faReadOnly (see page 4374) (C++).

To test for an attribute, combine the value of the Attr field with the attribute constant with the and operator. If the file has that attribute, the result will be greater than 0. For example, if the found file is a hidden file, the following expression will evaluate to true:

(SearchRec.Attr and faHidden (see page 4373)) <> 0.

(SearchRec.Attr & faHidden (see page 4373)) != 0.

Time contains the time stamp of the file. It can be converted to a TDateTime (see page 3323) value using FileDateToDateTime (see page 4130).

Size contains the size of the file in bytes.

Name contains the base file name, including extension.

FindHandle is an internal handle used to track find state.

FindData (Windows only) contains additional information such as the file creation time, last access time, and both the long and short file names.

Mode (Linux only) is the file's permission mask.

C++ Examples:

```
/*
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
```

```
directory. Click at least one box before clicking the button.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TSearchRec sr;
  int iAttributes = 0;
  StringGrid1->RowCount = 1;
  iAttributes |= faReadOnly * CheckBox1->Checked;
  iAttributes |= faHidden * CheckBox2->Checked;
  iAttributes |= faSysFile * CheckBox3->Checked;
  iAttributes |= faVolumeID * CheckBox4->Checked;
  iAttributes |= faDirectory * CheckBox5->Checked;
  iAttributes |= faArchive * CheckBox6->Checked;
  iAttributes |= faAnyFile * CheckBox7->Checked;
  StringGrid1->RowCount = 0;
  if (FindFirst(Edit1->Text, iAttributes, sr) == 0)
  {
    do
    {
      if ((sr.Attr & iAttributes) == sr.Attr)
      {
        StringGrid1->RowCount = StringGrid1->RowCount + 1;
        StringGrid1->Cells[1][StringGrid1->RowCount-1] = sr.Name;
        StringGrid1->Cells[2][StringGrid1->RowCount-1] = IntToStr(sr.Size);
      }
    } while (FindNext(sr) == 0);
    FindClose(sr);
  }
}
```

Delphi Examples:

```
{
The following example uses an edit control, a button, a
string grid, and seven check boxes. The check boxes
correspond to the seven possible file attributes. When the
button is clicked, the path specified in the edit control is
searched for files matching the checked file attributes. The
names and sizes of the matching files are inserted into the
string grid.  The Path constant parameter is the directory
and file name mask, including wildcard characters. For
example, '.\test\*.*' specifies all files in the current
directory. Click at least one box before clicking the button.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sr: TSearchRec;
  FileAttrs: Integer;
begin
  StringGrid1.RowCount := 1;
  if CheckBox1.Checked then
    FileAttrs := faReadOnly
  else
    FileAttrs := 0;
  if CheckBox2.Checked then
    FileAttrs := FileAttrs + faHidden;
  if CheckBox3.Checked then
    FileAttrs := FileAttrs + faSysFile;
  if CheckBox4.Checked then
    FileAttrs := FileAttrs + faVolumeID;
  if CheckBox5.Checked then
    FileAttrs := FileAttrs + faDirectory;
  if CheckBox6.Checked then
    FileAttrs := FileAttrs + faArchive;
  if CheckBox7.Checked then
    FileAttrs := FileAttrs + faAnyFile;
  with StringGrid1 do
```

```
    begin
      RowCount := 0;
      if SysUtils.FindFirst(Edit1.Text, FileAttrs, sr) = 0 then
      begin
        repeat
          if (sr.Attr and FileAttrs) = sr.Attr then
          begin
          RowCount := RowCount + 1;
          Cells[1,RowCount-1] := sr.Name;
          Cells[2,RowCount-1] := IntToStr(sr.Size);
          end;
        until FindNext(sr) <> 0;
        FindClose(sr);
      end;
    end;
  end;
```

37.4.15 SysUtils.TSysLocale Record

TSysLocale represents locale information.

Pascal

```
  TSysLocale = packed record
    DefaultLCID: Integer;
    PriLangID: Integer;
    SubLangID: Integer;
    FarEast: Boolean;
    MiddleEast: Boolean;
  end;
```

C++

```
  struct TSysLocale {
    int DefaultLCID;
    int PriLangID;
    int SubLangID;
    Boolean FarEast;
    Boolean MiddleEast;
  };
```

File

SysUtils

Description

The TSysLocale type represents a locale. The locale determines how dates and times are formatted, how items are alphabetically sorted, and how strings are compared.

The **DefaultLCID** field represents the locale identifier. This is a 32-bit value that specifies a default sort order and a default language identifier.

The **PirLangID** field specifies the primary type of the language identifier. This is one of the constants that identify a language group, such as **LANG_ENGLISH** or **LANG_FRENCH**.

The **SubLangID** field specifies the subtype of the language identifier. This is one of the constants that identify a language subgroup, such as **SUBLANG_ENGLISH_US** or **SUBLANG_FRENCH_CANADIAN**.

FarEast is true or nonzero if User32.dll supports DBCS; false or zero otherwise. In other words, **FarEast** is true or nonzero if the double-byte character-set (DBCS) version of User.exe is installed; false or zero otherwise.

MiddleEast is true if the system is enabled for Hebrew and Arabic languages; false otherwise.

See Also

TLocale

Languages (see page 4223)

TLocaleCode (see page 1708)

SysLocale (see page 4360)

37.4.16 SysUtils.TThreadInfo Record

Pascal

```
TThreadInfo = record
  Next: PThreadInfo;
  ThreadID: Cardinal;
  Active: Integer;
  RecursionCount: Cardinal;
end;
```

C++

```
struct TThreadInfo {
  PThreadInfo Next;
  unsigned ThreadID;
  int Active;
  unsigned RecursionCount;
};
```

File

SysUtils

Description

This is record SysUtils.TThreadInfo.

37.4.17 SysUtils.TTimeStamp Record

TTimeStamp represents time and date values.

Pascal

```
TTimeStamp = record
  Time: Integer;
  Date: Integer;
end;
```

C++

```
struct TTimeStamp {
  int Time;
  int Date;
};
```

File

SysUtils

Description

Use TTimeStamp to represent date and time values when a great deal of accuracy is required for the time portion. When the time values do not need to be precise to the millisecond, use the more compact TDateTime (see page 3323) representation. If additional precision is needed, use TSQLTimeStamp (but do NOT assign it to a Variant (see page 3421)).

The Time field indicates the number of milliseconds that have elapsed since midnight.

The Date field indicates the number of calendar days since the start of the calendar (the number of days since 1/1/0001 plus one).

See Also

DateTimeToTimeStamp (see page 4092)

TDateTime (see page 3787)

TimeStampToDateTime (see page 4298)

TSQLTimeStamp (see page 1737)

37.4.18 SysUtils.WordRec Record

WordRec declares a utility record to store high and low order bytes of a variable.

Pascal

```
WordRec = packed record
  case Integer of
    0: (Hi: Byte;
        Lo: Byte;);
    1: (Bytes: array [0..1] of Byte););
end;
```

C++

```
struct WordRec {
  Byte Hi;
  Byte Lo;
  array [0..1] of Byte) Bytes;
};
```

File

SysUtils

Description

The WordRec type declares a utility record that stores the high and low order bytes of the specified variable as type Byte. WordRec is used primarily for typecasting.

37.5 Types

The following table lists types in this documentation.

Types

Name	Description
ExceptClass (see page 4336)	ExceptClass declares an Exception (see page 3949) class.
PByteArray (see page 4337)	PByteArray declares a pointer to an array of Bytes.
PDayTable (see page 4337)	PDayTable is a pointer to a TDayTable (see page 4340) type.
PExceptionRecord (see page 4338)	PExceptionRecord is a pointer to a TExceptionRecord (see page 4322) type.
PThreadInfo (see page 4338)	This is type SysUtils.PThreadInfo.
PWordArray (see page 4338)	PWordArray is a pointer to a TWordArray (see page 4345) type.
TByteArray (see page 4339)	TByteArray declares an array of Bytes.
TBytes (see page 4339)	TBytes declares an array of Bytes.
TCharArray (see page 4340)	This is type SysUtils.TCharArray.
TDayTable (see page 4340)	TDayTable is used to hold the number of days of each month of a year.
TFileName (see page 4340)	TFileName is a string that represents a file name.
TFunc (see page 4341)	This is type SysUtils.TFunc.
TIntegerSet (see page 4341)	TIntegerSet is used to treat an integer as a set of bits.
TMREWSync (see page 4341)	TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 3997).
TPackageInfoProc (see page 4342)	TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 4209).
TPredicate (see page 4342)	This is type SysUtils.TPredicate.
TProc (see page 4343)	This is type SysUtils.TProc.
TProcedure (see page 4343)	TProcedure declares a procedure.
TReplaceFlags (see page 4343)	TReplaceFlags indicates how to perform a search-and-replace operation
TSysCharSet (see page 4344)	TSysCharSet is a set or characters.
TTerminateProc (see page 4344)	TTerminateProc is a call-back function invoked by CallTerminateProcs (see page 4071).
TValidatePackageProc (see page 4345)	TValidatePackageProc is a call-back function invoked by LoadPackage (see page 4224).
TWordArray (see page 4345)	TWordArray declares an array of Words.

37.5.1 SysUtils.ExceptClass Type

ExceptClass declares an Exception (see page 3949) class.

Pascal

```
ExceptClass = class of Exception;
```

C++

```
Exception ExceptClass;
```

File

SysUtils

Description

The ExceptClass type declares an Exception (see page 3949) class. This class encapsulates the fundamental properties and methods for all exceptions. Use Exception (see page 3949) as the base class for creating custom exceptions.

See Also

Exception (see page 3949)

37.5.2 SysUtils.PByteArray Type

PByteArray declares a pointer to an array of Bytes.

Pascal

```
PByteArray = ^TByteArray;
```

C++

```
^TByteArray PByteArray;
```

File

SysUtils

Description

The PByteArray type declares a pointer to a TByteArray (see page 4339) : a static array of Bytes. The array size is 32,768 (2**15) elements.

Note: Static arrays are declared with an immutable size.

See Also

TByteArray (see page 4339)

37.5.3 SysUtils.PDayTable Type

PDayTable is a pointer to a TDayTable (see page 4340) type.

Pascal

```
PDayTable = ^TDayTable;
```

C++

```
^TDayTable PDayTable;
```

File

SysUtils

Description

The TDayTable (see page 4340) type points to a TDayTable (see page 4340) variable. TDayTable (see page 4340) declares a static array of 12 Words. It is principally used by the MonthDays (see page 4386) constant to hold the number of days in a month for normal and leap years.

Note: Static arrays are declared with an immutable size.

See Also

TDayTable (see page 4340)

37.5.4 SysUtils.PExceptionRecord Type

PExceptionRecord is a pointer to a TExceptionRecord (see page 4322) type.

Pascal

```
PExceptionRecord = ^TExceptionRecord;
```

C++

```
^TExceptionRecord PExceptionRecord;
```

File

SysUtils

Description

The PExceptionRecord type points to a TExceptionRecord (see page 4322) variable. TExceptionRecord (see page 4322) holds information about an exception in an application.

See Also

TExceptionRecord (see page 4322)

37.5.5 SysUtils.PThreadInfo Type

Pascal

```
PThreadInfo = ^TThreadInfo;
```

C++

```
^TThreadInfo PThreadInfo;
```

File

SysUtils

Description

This is type SysUtils.PThreadInfo.

37.5.6 SysUtils.PWordArray Type

PWordArray is a pointer to a TWordArray (see page 4345) type.

Pascal

```
PWordArray = ^TWordArray;
```

C++

```
^TWordArray PWordArray;
```

File

SysUtils

Description

The PWordArray type points to a TWordArray (see page 4345) variable. The TWordArray (see page 4345) type declares a static array of Words. The array size is 16,384 (2**14) elements.

See Also

TWordArray (see page 4345)

37.5.7 SysUtils.TByteArray Type

TByteArray declares an array of Bytes.

Pascal

```
TByteArray = array[0..32767] of Byte;
```

C++

```
array[0..32767] of Byte TByteArray;
```

File

SysUtils

Description

The TByteArray type declares a static array of Bytes. The array size is 32,768 (2**15) elements.

Note: Static arrays are declared with an immutable size.

37.5.8 SysUtils.TBytes Type

TBytes declares an array of Bytes.

Pascal

```
TBytes = array of Byte;
```

C++

```
array of Byte TBytes;
```

File

SysUtils

Description

The TBytes type declares a dynamic array of Bytes.

Note: Dynamic arrays have no intrinsic size. SetLength (see page 3657) is used to allocate storage for the defined array size desired. Dynamic arrays are always integer-indexed, starting at 0.

37.5.9 SysUtils.TCharArray Type

Pascal

```
TCharArray = array of Char;
```

C++

```
array of Char TCharArray;
```

File

SysUtils

Description

This is type SysUtils.TCharArray.

37.5.10 SysUtils.TDayTable Type

TDayTable is used to hold the number of days of each month of a year.

Pascal

```
TDayTable = array[1..12] of Word;
```

C++

```
array[1..12] of Word TDayTable;
```

File

SysUtils

Description

The TDayTable type declares a static array of 12 Words. It is principally used by the MonthDays (see page 4386) constant to hold the number of days in a month for normal and leap years.

Note: Static arrays are declared with an immutable size.

See Also

MonthDays (see page 4386)

37.5.11 SysUtils.TFileName Type

TFileName is a string that represents a file name.

Pascal

```
TFileName = type string;
```

C++

```
type string TFileName;
```

File

SysUtils

Description

Use TFileName to represent strings that are only used for file names.

37.5.12 SysUtils.TFunc Type

Pascal

```
TFunc<T1,T2,T3,T4,TResult> = reference to function (Arg1: T1; Arg2: T2; Arg3: T3; Arg4: T4):
TResult;
```

C++

```
reference to function (Arg1: T1; Arg2: T2; Arg3: T3; Arg4: T4): TResult TFunc;
```

File

SysUtils

Description

This is type SysUtils.TFunc.

37.5.13 SysUtils.TIntegerSet Type

TIntegerSet is used to treat an integer as a set of bits.

Pascal

```
TIntegerSet = set of 0..SizeOf(Integer) * 8 - 1;
```

C++

```
0..SizeOf(Integer) * 8 - 1 TIntegerSet;
```

File

SysUtils

Description

Type TIntegerSet = set of 0 .. Integer-bit-size-1;

Assuming that an integer is 4 bytes (=32 bits), the set will 0 .. 31. TIntegerSet is typically used to treat an integer as a set of bits. To do so, an integer variable value is assigned to a TIntegerSet variable using TIntegerSet casting.

37.5.14 SysUtils.TMREWSync Type

TMREWSync is a short name equivalent to TMultiReadExclusiveWriteSynchronizer (see page 3997).

Pascal

```
TMREWSync = TMultiReadExclusiveWriteSynchronizer;
```

C++

```
TMultiReadExclusiveWriteSynchronizer TMREWSync;
```

File

SysUtils

Description

The TMREWSync type is used instead of TMultiReadExclusiveWriteSynchronizer (see page 3997).

Note: TMultiReadExclusiveWriteSynchronizer (see page 3997) is a class that minimizes thread serialization to gain read access to a resource shared among threads while still providing complete exclusivity to callers needing write access to the shared resource.

37.5.15 SysUtils.TPackageInfoProc Type

TPackageInfoProc is a call-back procedure invoked by GetPackageInfo (see page 4209).

Pascal

```
TPackageInfoProc = procedure (const Name: string; NameType: TNameType; Flags: Byte; Param:
Pointer);
```

C++

```
(const Name: string; NameType: TNameType; Flags: Byte; Param: Pointer) ( TPackageInfoProc)();
```

File

SysUtils

Description

The TPackageInfoProc type declares the procedure that is invoked by GetPackageInfo (see page 4209) to retrieve package information. It is called for every unit in the package, and every package required by that package.

The following table lists the information provided in these call back invocations of your TPackageInfoProc variable.

Parameter name	Parameter value
Name	Unit or required package name
NameType	Indicates if this call is for a contained unit, required package or DcpBpiName.
Flags	
Param	

See Also

GetPackageInfo (see page 4209)

37.5.16 SysUtils.TPredicate Type

Pascal

```
TPredicate<T> = reference to function (Arg1: T): Boolean;
```

C++

```
reference to function (Arg1: T): Boolean TPredicate;
```

File

SysUtils

Description

This is type SysUtils.TPredicate.

37.5.17 SysUtils.TProc Type

Pascal

```
TProc<T1,T2,T3,T4> = reference to procedure (Arg1: T1; Arg2: T2; Arg3: T3; Arg4: T4);
```

C++

```
reference to procedure (Arg1: T1; Arg2: T2; Arg3: T3; Arg4: T4) TProc;
```

File

SysUtils

Description

This is type SysUtils.TProc.

37.5.18 SysUtils.TProcedure Type

TProcedure declares a procedure.

Pascal

```
TProcedure = procedure;
```

C++

```
( TProcedure)();
```

File

SysUtils

Description

Use the TProcedure type as a generic Procedure pointer.

37.5.19 SysUtils.TReplaceFlags Type

TReplaceFlags indicates how to perform a search-and-replace operation

Pascal

```
TReplaceFlags = set of (rfReplaceAll, rfIgnoreCase);
```

C++

```
(rfReplaceAll, rfIgnoreCase) TReplaceFlags;
```

File

SysUtils

Description

TReplaceFlags is a set of flags that govern how to locate and replace a substring within a string. This set can contain the following values:

Value	Meaning
rfReplaceAll	Replace all occurrences. If this flag is not present, only the first occurrence of the target substring is replaced.
rfIgnoreCase	Match occurrences of the substring case-insensitively. If this flag is not present, only case-sensitive matches are considered.

See Also

StringReplace (see page 4253)

37.5.20 SysUtils.TSysCharSet Type

TSysCharSet is a set or characters.

Pascal

```
TSysCharSet = set of AnsiChar;
```

C++

```
AnsiChar TSysCharSet;
```

File

SysUtils

Description

TSysCharSet is a set containing ANSI characters. TSysCharSet is useful when passing a set of characters to various functions that require them. TSysCharSet is primarily used in the CharInSet (see page 4073) routine, which detects whether a given character is contained within the set.

Note: You cannot use TSysCharSet to group Unicode characters. This means that TSysCharSet cannot be used to check whether an Unicode character is a part of TSysCharSet.

See Also

CharInSet (see page 4073)

TCharacter

37.5.21 SysUtils.TTerminateProc Type

TTerminateProc is a call-back function invoked by CallTerminateProcs (see page 4071).

Pascal

```
TTerminateProc = function : Boolean;
```

C++

```
: Boolean ( TTerminateProc)();
```

File

SysUtils

Description

CallTerminateProcs (see page 4071) is called internally when an application is about to terminate. It invokes TTerminateProc for each defined termination function. Termination functions are added using AddTerminateProc (see page 4048). The application only terminates if all termination routines return true.

See Also

AddTerminateProc (see page 4048)

CallTerminateProcs (see page 4071)

37.5.22 SysUtils.TValidatePackageProc Type

TValidatePackageProc is a call-back function invoked by LoadPackage (see page 4224).

Pascal

```
TValidatePackageProc = function (Module: HMODULE): Boolean;
```

C++

```
(Module: HMODULE): Boolean ( TValidatePackageProc)();
```

File

SysUtils

Description

TValidatePackageProc is optionally invoked by LoadPackage (see page 4224) when a package is loaded. The package is not loaded if TValidatePackageProc determines that it is not valid and hence returns false.

See Also

LoadPackage (see page 4224)

37.5.23 SysUtils.TWordArray Type

TWordArray declares an array of Words.

Pascal

```
TWordArray = array[0..16383] of Word;
```

C++

```
array[0..16383] of Word TWordArray;
```

File

SysUtils

Description

The TWordArray type declares a static array of Words. The array size is 16,384 (2**14) elements.

Note: Static arrays are declared with an immutable size.

37.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
CurrencyDecimals (see page 4348)	CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.
CurrencyFormat (see page 4348)	CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 \| 1 = 1$ | 2 = $ 1 | 3 = 1 $
CurrencyString (see page 4349)	CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.
DateSeparator (see page 4349)	DateSeparator is the character used to separate the year, month, and day parts of a date value.
DecimalSeparator (see page 4350)	DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.
EmptyAnsiStr (see page 4351)	This is variable SysUtils.EmptyAnsiStr.
EmptyStr (see page 4351)	An empty string constant.
EmptyWideStr (see page 4351)	An empty string version of the WideString (see page 3467) constant.
EraNames (see page 4352)	Defines the names of calendar eras for East Asian locales.
EraYearOffsets (see page 4352)	Controls the DateTimeToStr (see page 4089) function (for East Asian locales).
FalseBoolStrs (see page 4352)	Lists strings that can represent the boolean value false.
GetDiskFreeSpaceEx (see page 4353)	This is variable SysUtils.GetDiskFreeSpaceEx.
HexDisplayPrefix (see page 4353)	HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.
LeadBytes (see page 4354)	Lists all values that can be the first byte of a double byte character on the local system.
ListSeparator (see page 4354)	The character used to separate items in a list.
LongDateFormat (see page 4354)	LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.
LongDayNames (see page 4355)	LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 4195) causes a long day name to be substituted.
LongMonthNames (see page 4355)	LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 4195) causes a long month name to be substituted.
LongTimeFormat (see page 4356)	LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.
NegCurrFormat (see page 4356)	NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.
NullAnsiStr (see page 4357)	This is variable SysUtils.NullAnsiStr.
NullStr (see page 4357)	A pointer to an empty string.
NullWideStr (see page 4358)	A pointer to an empty wide string.
ShortDateFormat (see page 4358)	ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.
ShortDayNames (see page 4359)	ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 4195) causes a short day name to be substituted.
ShortMonthNames (see page 4359)	ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 4195) causes a short month name to be substituted.
ShortTimeFormat (see page 4360)	ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.
SysLocale (see page 4360)	SysLocale holds locale information.
ThousandSeparator (see page 4362)	ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.

TimeAMString (see page 4362)	TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.
TimePMString (see page 4362)	TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.
TimeSeparator (see page 4363)	TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.
TrueBoolStrs (see page 4363)	Lists strings that can represent the boolean value false.
TwoDigitYearCenturyWindow (see page 4364)	Determines what century is added to two-digit years when string dates are converted to numeric dates.
Win32BuildNumber (see page 4364)	Specifies which the build level of the Windows Operating System (see page 3159) operating.
Win32CSDVersion (see page 4365)	Specifies the Win32 latest installed service pack.
Win32MajorVersion (see page 4365)	Specifies which major Windows Operating System (see page 3159) version is operating.
Win32MinorVersion (see page 4366)	Specifies which minor Windows Operating System (see page 3159) version is operating.
Win32Platform (see page 4366)	Specifies the Windows Operating System (see page 3159) platform identifier.

37.6.1 SysUtils.CurrencyDecimals Variable

CurrencyDecimals is the number of digits to the right of the decimal point in a currency amount.

Pascal

```
CurrencyDecimals: Byte;
```

C++

```
Byte CurrencyDecimals;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_ICURRDIGITS.

37.6.2 SysUtils.CurrencyFormat Variable

CurrencyFormat defines the currency symbol placement and separation used in floating-point–to–decimal conversions. Possible values are: 0 = $1 | 1 = 1$ | 2 = $ 1 | 3 = 1 $

Pascal

```
CurrencyFormat: Byte;
```

C++

```
Byte CurrencyFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_ICURRENCY.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
```

37.6.3 SysUtils.CurrencyString Variable

CurrencyString defines the currency symbol (or characters) used in floating-point to decimal conversions.

Pascal

```
CurrencyString: string;
```

C++

```
UnicodeString CurrencyString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SCURRENCY.

37.6.4 SysUtils.DateSeparator Variable

DateSeparator is the character used to separate the year, month, and day parts of a date value.

Pascal

```
DateSeparator: Char;
```

C++

```
Char DateSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SDATE.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
```

37.6.5 SysUtils.DecimalSeparator Variable

DecimalSeparator is the character used to separate the integer part from the fractional part of a number. This variable must have a value other than #0.

Pascal

```
DecimalSeparator: Char;
```

C++

```
Char DecimalSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SDECIMAL.

37.6.6 SysUtils.EmptyAnsiStr Variable

Pascal

```
EmptyAnsiStr: AnsiString = '';
```

C++

```
AnsiString EmptyAnsiStr = '';
```

File

SysUtils

Description

This is variable SysUtils.EmptyAnsiStr.

37.6.7 SysUtils.EmptyStr Variable

An empty string constant.

Pascal

```
EmptyStr: string = '';
```

C++

```
UnicodeString EmptyStr = '';
```

File

SysUtils

Description

Included for backward compatibility only.

37.6.8 SysUtils.EmptyWideStr Variable

An empty string version of the WideString (see page 3467) constant.

Pascal

```
EmptyWideStr: WideString = '';
```

C++

```
BSTR EmptyWideStr = '';
```

File

SysUtils

Description

Included for backward compatibility only.

37.6.9 SysUtils.EraNames Variable

Defines the names of calendar eras for East Asian locales.

Pascal

```
EraNames: array [1..MaxEraCount] of string;
```

C++

```
array [1..MaxEraCount] of string EraNames;
```

File

SysUtils

Description

EraNames defines an array of 7 era names for Japanese and Taiwanese locales. It is principally used for date formatting.

See Also

FormatDateTime (see page 4195)

DateTimeToString (see page 4089)

37.6.10 SysUtils.EraYearOffsets Variable

Controls the DateTimeToStr (see page 4089) function (for East Asian locales).

Pascal

```
EraYearOffsets: array [1..MaxEraCount] of Integer;
```

C++

```
array [1..MaxEraCount] of Integer EraYearOffsets;
```

File

SysUtils

37.6.11 SysUtils.FalseBoolStrs Variable

Lists strings that can represent the boolean value false.

Pascal

```
FalseBoolStrs: array of String;
```

C++

```
array of String FalseBoolStrs;
```

File

SysUtils

Description

FalseBoolStrs is an array of strings that are used in conversions between Boolean values and strings.

When using the BoolToStr (see page 4066) function, false values are converted to the first string in the list.

When using the StrToBool (see page 4276) function, any string found in the list, regardless of case, is converted to false. If no string match is found, TrueBoolStrs (see page 4363) is checked, true being returned if a match is found. No matches result in the EConvertError (see page 3919) exception being thrown.

See Also

StrToBool (see page 4276)

TryStrToBool (see page 4305)

TrueBoolStrs (see page 4363)

BoolToStr (see page 4066)

37.6.12 SysUtils.GetDiskFreeSpaceEx Variable

Pascal

```
GetDiskFreeSpaceEx: function (Directory: PChar; var FreeAvailable, TotalSpace: TLargeInteger;
TotalFree: PLargeInteger): Bool stdcall = nil;
```

C++

```
function (Directory: PChar; var FreeAvailable, TotalSpace: TLargeInteger; TotalFree:
PLargeInteger): Bool stdcall GetDiskFreeSpaceEx = nil;
```

File

SysUtils

Description

This is variable SysUtils.GetDiskFreeSpaceEx.

37.6.13 SysUtils.HexDisplayPrefix Variable

HexDisplayPrefix contains the prefix to display on hexadecimal values - `$` for Pascal (Delphi) syntax, `0x` for C++ syntax.

Pascal

```
HexDisplayPrefix: string = '$';
```

C++

```
UnicodeString HexDisplayPrefix = '$';
```

File

SysUtils

Description

This is for display only - this does not affect the string-to-integer conversion routines.

37.6.14 SysUtils.LeadBytes Variable

Lists all values that can be the first byte of a double byte character on the local system.

Pascal

```
LeadBytes: set of AnsiChar = [];
```

C++

```
set of AnsiChar LeadBytes = [];
```

File

SysUtils

Description

Use LeadBytes to test whether a value can represent the lead byte in a double byte character. If an individual value is in the LeadBytes set, it is not necessarily used as the lead byte of a double byte character. It may also appear as the second byte of a double byte character. To determine whether the character is used as a lead byte, use the global ByteType (see page 4070) or StrByteType (see page 4242) function.

Note: LeadBytes is always empty when running on a system with a locale that does not use a multi byte character system (MBCS).

See Also

ByteType (see page 4070)

StrByteType (see page 4242)

37.6.15 SysUtils.ListSeparator Variable

The character used to separate items in a list.

Pascal

```
ListSeparator: Char;
```

C++

```
Char ListSeparator;
```

File

SysUtils

Description

On Windows, the default value is fetched from LOCALE_SLIST.

37.6.16 SysUtils.LongDateFormat Variable

LongDateFormat is the format string used to convert a date value to a long string suitable for display but not for editing.

Pascal

```
LongDateFormat: string;
```

C++

```
UnicodeString LongDateFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SLONGDATE.

37.6.17 SysUtils.LongDayNames Variable

LongDayNames is the array of strings containing long day names. The dddd format specifier in a format string passed to FormatDateTime (see page 4195) causes a long day name to be substituted.

Pascal

```
LongDayNames: array[1..7] of string;
```

C++

```
array[1..7] of string LongDayNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SDAYNAME system locale entries.

37.6.18 SysUtils.LongMonthNames Variable

LongMonthNames is the array of strings containing long month names. The mmmm format specifier in a format string passed to FormatDateTime (see page 4195) causes a long month name to be substituted.

Pascal

```
LongMonthNames: array[1..12] of string;
```

C++

```
array[1..12] of string LongMonthNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SMONTHNAME system locale entries.

37.6.19 SysUtils.LongTimeFormat Variable

LongTimeFormat is the format string used to convert a time value to a string with hours, minutes, and seconds.

Pascal

```
LongTimeFormat: string;
```

C++

```
UnicodeString LongTimeFormat;
```

File

SysUtils

Description

On Windows, the default value is computed from LOCALE_ITIME and LOCALE_ITLZERO.

37.6.20 SysUtils.NegCurrFormat Variable

NegCurrFormat defines the currency format used in floating-point to decimal conversions of negative numbers.

Pascal

```
NegCurrFormat: Byte;
```

C++

```
Byte NegCurrFormat;
```

File

SysUtils

Description

Possible values are:

Value	Format (see page 4188)
0	($1)
1	-$1
2	$-1
3	$1-
4	(1$)
5	-1$
6	1-$
7	1$-
8	-1 $
9	-$ 1
10	1 $-
11	$ 1-

12	$ -1
13	1- $
14	($ 1)
15	(1 $)

On Windows, the initial value is fetched from LOCALE_INEGCURR.

37.6.21 SysUtils.NullAnsiStr Variable

Pascal

```
NullAnsiStr: PAnsiString = @EmptyAnsiStr;
```

C++

```
PAnsiString NullAnsiStr = @EmptyAnsiStr;
```

File

SysUtils

Description

This is variable SysUtils.NullAnsiStr.

37.6.22 SysUtils.NullStr Variable

A pointer to an empty string.

Pascal

```
NullStr: PString = @EmptyStr;
```

C++

```
PString NullStr = @EmptyStr;
```

File

SysUtils

Description

NullStr defines a pointer to the empty string EmptyStr (see page 4351).

See Also

EmptyStr (see page 4351)

NullWideStr (see page 4358)

PString (see page 3776)

37.6.23 SysUtils.NullWideStr Variable

A pointer to an empty wide string.

Pascal

```
NullWideStr: PWideString = @EmptyWideStr;
```

C++

```
PWideString NullWideStr = @EmptyWideStr;
```

File

SysUtils

Description

NullWideStr defines a pointer to the empty wide string EmptyWideStr (see page 4351).

See Also

EmptyWideStr (see page 4351)

NullStr (see page 4357)

PWideString (see page 3782)

37.6.24 SysUtils.ShortDateFormat Variable

ShortDateFormat is the format string used to convert a date value to a short string suitable for editing.

Pascal

```
ShortDateFormat: string;
```

C++

```
UnicodeString ShortDateFormat;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_SSHORTDATE.

C++ Examples:

```
/*
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  DateSeparator = '-';
  ShortDateFormat = "m/d/yyyy";
```

```
  Label1->Caption = DateToStr(Date());
}
```

Delphi Examples:

```
{
This example uses a label and a button on a form. When the
user clicks the button, the current date displays in the
caption of the label. Because some of the date variables are
assigned new values, the format of the date in the label
changes. For example, if the date is 9/15/94, the date
displays as 9-15-1994.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  DateSeparator := '-';
  ShortDateFormat := 'm/d/yyyy';
  Label1.Caption := DateToStr(Date);
end;
```

37.6.25 SysUtils.ShortDayNames Variable

ShortDayNames is the array of strings containing short day names. The ddd format specifier in a format string passed to FormatDateTime (see page 4195) causes a short day name to be substituted.

Pascal

```
ShortDayNames: array[1..7] of string;
```

C++

```
array[1..7] of string ShortDayNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SABBREVDAYNAME system locale entries.

37.6.26 SysUtils.ShortMonthNames Variable

ShortMonthNames is the array of strings containing short month names. The mmm format specifier in a format string passed to FormatDateTime (see page 4195) causes a short month name to be substituted.

Pascal

```
ShortMonthNames: array[1..12] of string;
```

C++

```
array[1..12] of string ShortMonthNames;
```

File

SysUtils

Description

On Windows, the default values are fetched from the LOCALE_SABBREVMONTHNAME system locale entries.

37.6.27 SysUtils.ShortTimeFormat Variable

ShortTimeFormat is the format string used to convert a time value to a short string with only hours and minutes.

Pascal

```
ShortTimeFormat: string;
```

C++

```
UnicodeString ShortTimeFormat;
```

File

SysUtils

Description

On Windows, the default value is computed from LOCALE_ITIME and LOCALE_ITLZERO.

37.6.28 SysUtils.SysLocale Variable

SysLocale holds locale information.

Pascal

```
SysLocale: TSysLocale;
```

C++

```
TSysLocale SysLocale;
```

File

SysUtils

Description

The SysLocale holds the attributes of the current locale. The locale determines how dates and times are formatted, how items are alphabetically sorted, and how strings are compared.

The **DefaultLCID** field represents the locale identifier. This is a 32-bit value that specifies a default sort order and a default language identifier.

The **PirLangID** field specifies the primary type of the language identifier. This is one of the constants that identify a language group, such as **LANG_ENGLISH** or **LANG_FRENCH**.

The **SubLangID** field specifies the subtype of the language identifier. This is one of the constants that identify a language subgroup, such as **SUBLANG_ENGLISH_US** or **SUBLANG_FRENCH_CANADIAN**.

FarEast is true or nonzero if User32.dll supports DBCS; false or zero otherwise. In other words, **FarEast** is true or nonzero if the double-byte character-set (DBCS) version of User.exe is installed; false or zero otherwise.

MiddleEast is true if the system is enabled for Hebrew and Arabic languages; false otherwise.

C++ Examples:

```
/*
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
```

```
Regional and Language Options in the Windows Control Panel.
*/
void __fastcall TForm1::ApplicationEvents1SettingChange(TObject *Sender, int Flag, const
AnsiString Section, int &Result)
{
  Memo1->Lines->BeginUpdate();
  try
  {
//    char buffer[64];
    AnsiString buffer;
    buffer.sprintf("Section = %s", Section.c_str());
    Memo1->Lines->Add(buffer);
    buffer.sprintf("Flag = %8x", Flag);
    Memo1->Lines->Add(buffer);
    if (strcmp(Section.c_str(), "intl") == 0)
    {
      buffer.sprintf("DefaultLCID = %8x", SysLocale.DefaultLCID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("PriLangID = %8x", SysLocale.PriLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("SubLangID = %8x", SysLocale.SubLangID);
      Memo1->Lines->Add(buffer);
      buffer.sprintf("FarEast = %s", BoolToStr(SysLocale.FarEast, true));
      Memo1->Lines->Add(buffer);
      buffer.sprintf("MiddleEast = %s", BoolToStr(SysLocale.MiddleEast, true));
      Memo1->Lines->Add(buffer);
      Memo1->Lines->Add("");
    }
  }
  __finally
  {
    Memo1->Lines->EndUpdate();
  }
}
```

Delphi Examples:

```
{
This example requires that a memo and a TApplicationEvent be
added to the form.  Change the Windows Locale by using the
Regional and Language Options in the Windows Control Panel.
}
procedure TForm1.ApplicationEvents1SettingChange(Sender: TObject; Flag: Integer; const
Section: string; var Result: Integer);
begin
  Memo1.Lines.BeginUpdate;
  try
    Memo1.Lines.Add(Format('Section  = %s', [Section]));
    Memo1.Lines.Add(Format('Flags  = %.8x', [Flag]));
    if AnsiSameStr(Section, 'intl') then
    with SysLocale do
    begin
      Memo1.Lines.Add(Format('DefaultLCID  = %.8x', [DefaultLCID]));
      Memo1.Lines.Add(Format('PriLangID  = %.8x', [PriLangID]));
      Memo1.Lines.Add(Format('SubLangID  = %.8x', [SubLangID]));
      Memo1.Lines.Add(Format('FarEast  = %s', [BoolToStr(FarEast, True)]));
      Memo1.Lines.Add(Format('MiddleEast  = %s', [BoolToStr(MiddleEast, True)]));
    end;
    Memo1.Lines.Add('');
  finally
    Memo1.Lines.EndUpdate;
  end;
//  Result:= 0;
end;
```

See Also

TSysLocale (see page 4333)

Languages (see page 4223)

37.6.29 SysUtils.ThousandSeparator Variable

ThousandSeparator is the character used to separate thousands in numbers with more than three digits to the left of the decimal separator.

Pascal

```
ThousandSeparator: Char;
```

C++

```
Char ThousandSeparator;
```

File

SysUtils

Description

A value of #0 indicates that no thousand separator character should be output, even if the format string specifies it.

On Windows, the initial value is fetched from LOCALE_STHOUSAND.

37.6.30 SysUtils.TimeAMString Variable

TimeAMString is the suffix string used for time values between 00:00 and 11:59 in 12-hour clock format.

Pascal

```
TimeAMString: string;
```

C++

```
UnicodeString TimeAMString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_S1159.

37.6.31 SysUtils.TimePMString Variable

TimePMString is the suffix string used for time values between 12:00 and 23:59 in 12-hour clock format.

Pascal

```
TimePMString: string;
```

C++

```
UnicodeString TimePMString;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_S2359.

37.6.32 SysUtils.TimeSeparator Variable

TimeSeparator is the character used to separate the hour, minute, and second parts of a time value.

Pascal

```
TimeSeparator: Char;
```

C++

```
Char TimeSeparator;
```

File

SysUtils

Description

On Windows, the initial value is fetched from LOCALE_STIME.

37.6.33 SysUtils.TrueBoolStrs Variable

Lists strings that can represent the boolean value false.

Pascal

```
TrueBoolStrs: array of String;
```

C++

```
array of String TrueBoolStrs;
```

File

SysUtils

Description

TrueBoolStrs is an array of strings that are used in conversions between Boolean values and strings.

When using the BoolToStr (see page 4066) function, false values are converted to the first string in the list.

When using the StrToBool (see page 4276) function, any string found in the list, regardless of case, is converted to true. If no string match is found, FalseBoolStrs (see page 4352) is checked, false being returned if a match is found. No matches result in the EConvertError (see page 3919) exception being thrown.

See Also

StrToBool (see page 4276)

TryStrToBool (see page 4305)

FalseBoolStrs (see page 4352)

BoolToStr (see page 4066)

37.6.34 SysUtils.TwoDigitYearCenturyWindow Variable

Determines what century is added to two-digit years when string dates are converted to numeric dates.

Pascal

```
TwoDigitYearCenturyWindow: Word = 50;
```

C++

```
Word TwoDigitYearCenturyWindow = 50;
```

File

SysUtils

Description

The value of TwoDigitYearCenturyWindow, if it is nonzero, is subtracted from the current year to calculate the floating "pivot" prior to which two-digit years are interpreted as falling in the next century. The default value is 50. The table below shows some examples.

37.6.35 SysUtils.Win32BuildNumber Variable

Specifies which the build level of the Windows Operating System (see page 3159) operating.

Pascal

```
Win32BuildNumber: Integer = 0;
```

C++

```
int Win32BuildNumber = 0;
```

File

SysUtils

Description

Win32BuildNumber specifies the build number of the Operating System (see page 3159). The unique identifier for a Windows Operating System (see page 3159) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber

Note: Win32BuildNumber is available on Windows only.

See Also

Win32CSDVersion (see page 4365)

Win32MinorVersion (see page 4366)

Win32MajorVersion (see page 4365)

Win32Platform (see page 4366)

37.6.36 SysUtils.Win32CSDVersion Variable

Specifies the Win32 latest installed service pack.

Pascal

```
Win32CSDVersion: string = '';
```

C++

```
UnicodeString Win32CSDVersion = '';
```

File

SysUtils

Description

Win32CSDVersion specifies the name of the most recent service pack.

Note: Win32CSDVersion is available on Windows only.

See Also

Win32BuildNumber (see page 4364)

Win32MajorVersion (see page 4365)

Win32MinorVersion (see page 4366)

Win32Platform (see page 4366)

37.6.37 SysUtils.Win32MajorVersion Variable

Specifies which major Windows Operating System (see page 3159) version is operating.

Pascal

```
Win32MajorVersion: Integer = 0;
```

C++

```
int Win32MajorVersion = 0;
```

File

SysUtils

Description

Win32MajorVersion specifies the Major version of the Operating System (see page 3159). The unique identifier for a Windows Operating System (see page 3159) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber (see page 4364)

Note: Win32MajorVersion is available on Windows only.

See Also

Win32BuildNumber (see page 4364)

Win32CSDVersion (see page 4365)

Win32MinorVersion (see page 4366)

Win32Platform (see page 4366)

37.6.38 SysUtils.Win32MinorVersion Variable

Specifies which minor Windows Operating System (see page 3159) version is operating.

Pascal

```
Win32MinorVersion: Integer = 0;
```

C++

```
int Win32MinorVersion = 0;
```

File

SysUtils

Description

Win32MinorVersion specifies the Minor version of the Operating System (see page 3159). The unique identifier for a Windows Operating System (see page 3159) is Win32MajorVersion.Win32MinorVersion.Win32BuildNumber (see page 4364)

Note: Win32MinorVersion is available on Windows only.

See Also

Win32BuildNumber (see page 4364)

Win32CSDVersion (see page 4365)

Win32MajorVersion (see page 4365)

Win32Platform (see page 4366)

37.6.39 SysUtils.Win32Platform Variable

Specifies the Windows Operating System (see page 3159) platform identifier.

Pascal

```
Win32Platform: Integer = 0;
```

C++

```
int Win32Platform = 0;
```

File

SysUtils

Description

Win32Platform is used to identify the type of Win32 platform the system is running. This will be one of the following values:

Value	Meaning
VER_PLATFORM_WIN32s = 0	System (see page 3159) is Win32s.
VER_PLATFORM_WIN32_WINDOWS = 1	System (see page 3159) is Windows 95.

VER_PLATFORM_WIN32_NT = 2	System (see page 3159) is Windows NT. This includes Windows XP and Vista.

Note: Win32Platform is available on Windows only.

See Also

Win32BuildNumber (see page 4364)

Win32CSDVersion (see page 4365)

Win32MajorVersion (see page 4365)

Win32MinorVersion (see page 4366)

37.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
DateDelta (see page 4369)	Specifies the difference between TDateTime (see page 3323) and TTimeStamp (see page 4334) starting values.
DefaultFalseBoolStr (see page 4370)	Specifies the default string representation of the boolean false value.
DefaultTrueBoolStr (see page 4370)	Specifies the default string representation of the boolean true value.
DriveDelim (see page 4371)	Specifies the character used to separate the Windows drive from the path in a file name.
faAnyFile (see page 4371)	Specifies any file type.
faArchive (see page 4372)	Specifies only Windows archived file types.
faDirectory (see page 4372)	Specifies only directory 'file' types.
faHidden (see page 4373)	Specifies only hidden file types.
faNormal (see page 4374)	This is constant SysUtils.faNormal.
faReadOnly (see page 4374)	Specifies only read only file types.
faSymLink (see page 4375)	Specifies only symbolic link file types.
faSysFile (see page 4375)	Specifies only system file types.
faTemporary (see page 4376)	This is constant SysUtils.faTemporary.
faVolumeID (see page 4376)	Specifies Volume ID file types.
fmOpenRead (see page 4377)	Specifies a file should be opened for reading only.
fmOpenReadWrite (see page 4378)	Specifies a file should be opened for reading and writing.
fmOpenWrite (see page 4378)	Specifies a file should be opened for writing only.
fmShareCompat (see page 4379)	Specifies file sharing that is FCB compatible.
fmShareDenyNone (see page 4380)	Specifies file sharing that does not prevent file access by other applications.
fmShareDenyRead (see page 4380)	Specifies file sharing that prevents read file access by other applications.
fmShareDenyWrite (see page 4381)	Specifies file sharing that prevents write file access by other applications.
fmShareExclusive (see page 4382)	Specifies file sharing that prevents any file access by other applications.
HoursPerDay (see page 4382)	Specifies the number of hours in a day.
MaxCurrency (see page 4383)	Defines the largest value supported for Currency (see page 3235) data types.
MaxDateTime (see page 4383)	Specifies the highest supported TDateTime (see page 3323) value.
MaxEraCount (see page 4384)	This is constant SysUtils.MaxEraCount.
MinCurrency (see page 4384)	Defines the largest negative value supported for Currency (see page 3235) data types.
MinDateTime (see page 4384)	Specifies the earliest supported TDateTime (see page 3323) value.
MinsPerDay (see page 4385)	Specifies the number of minutes in a day.
MinsPerHour (see page 4386)	Specifies the number of minutes in an hour.
MonthDays (see page 4386)	Defines the number of days in each month for normal and leap years.
MSecsPerDay (see page 4387)	Specifies the number of milliseconds in a day.
MSecsPerSec (see page 4387)	Specifies the number of milliseconds in a second.
PathDelim (see page 4388)	Specifies the character used to separate each directory name in a file path.
PathSep (see page 4388)	Specifies the character used to separate a sequence of file or directory names.
pfBCB4Produced (see page 4389)	Indicates that a package was produced by via C++.
pfConsumerBCB (see page 4389)	Indicates that C++ can be a consumer of a package.
pfConsumerCompat (see page 4390)	This is constant SysUtils.pfConsumerCompat.
pfConsumerDelphi (see page 4390)	Indicates that Delphi can be a consumer of a package.
pfConsumerMask (see page 4391)	Indicates the comsumer of a package.
pfDelphi4Produced (see page 4391)	Indicates that a package was produced by Delphi 4 or later.
pfDesignOnly (see page 4392)	Indicates that a package was compiled with {$DESIGNONLY ON}.

pfExeModule (see page 4392)	Indicates that a module is an executable.
pfIgnoreDupUnits (see page 4392)	Indicates that a package was compiled with {$DUPLICATEENABLE ON}.
pfLibraryModule (see page 4393)	Indicates that a module is a library.
pfModuleTypeMask (see page 4393)	Indicates that a package is a module.
pfNeverBuild (see page 4394)	Indicates that a package was compiled with the -z flag.
pfPackageModule (see page 4394)	Indicates that a module is a package.
pfProducerMask (see page 4395)	Indicates the producer of a package.
pfProducerUndefined (see page 4395)	Indicates that a package was produced by an unknown developmemt environment.
pfRunOnly (see page 4396)	Indicates that a package was compiled with {$RUNONLY ON}.
pfV3Produced (see page 4396)	Indicates that a package was produced by Delphi 3 or later.
SecsPerDay (see page 4397)	Specifies the number of seconds in a day.
SecsPerHour (see page 4397)	This is constant SysUtils.SecsPerHour.
SecsPerMin (see page 4398)	Specifies the number of seconds in a minute.
SwitchChars (see page 4398)	Specifies the prefix character(s) for program command line arguments.
ufImplicitUnit (see page 4399)	Indicates that a unit was implicitly imported into a package.
ufMainUnit (see page 4399)	Indicates that a unit is the main one in a project.
ufOrgWeakUnit (see page 4400)	This is constant SysUtils.ufOrgWeakUnit.
ufPackageUnit (see page 4400)	Indicates that a unit is a package.
ufWeakPackageUnit (see page 4400)	Indicates that a unit is a either a weak unit or a package unit.
ufWeakUnit (see page 4401)	Indicates that a unit is a weak unit.
UnixDateDelta (see page 4401)	Specifies the difference between TDateTime (see page 3323) and Lunix date/time origins.

37.7.1 SysUtils.DateDelta Constant

Specifies the difference between TDateTime (see page 3323) and TTimeStamp (see page 4334) starting values.

Pascal

```
DateDelta = 693594;
```

C++

```
DateDelta = 693594;
```

File

SysUtils

Description

DateDelta is the number of days (693594) between January 1, 0001 (starting TTimeStamp (see page 4334) value), and December 31, 1899 (starting TDateTime (see page 3323) value). This value is useful for converting date-time values between TTimeStamp (see page 4334) and TDateTime (see page 3323).

See Also

TDateTime (see page 3787)

TTimeStamp (see page 4334)

UnixDateDelta (see page 4401)

37.7.2 SysUtils.DefaultFalseBoolStr Constant

Specifies the default string representation of the boolean false value.

Pascal

```
DefaultFalseBoolStr = 'False';
```

C++

```
DefaultFalseBoolStr = 'False';
```

File

SysUtils

Description

DefaultFalseBoolStr is 'False'. It is used internally by the StrToBool (see page 4276) and BoolToStr (see page 4066) routines. They assign DefaultFalseBoolStr to the 1st element of the FalseBoolStrs (see page 4352) array, used by these routines as a default string representation of the boolean false value.

See Also

DefaultTrueBoolStr (see page 4370)

StrToBool (see page 4276)

FalseBoolStrs (see page 4352)

TrueBoolStrs (see page 4363)

BoolToStr (see page 4066)

37.7.3 SysUtils.DefaultTrueBoolStr Constant

Specifies the default string representation of the boolean true value.

Pascal

```
DefaultTrueBoolStr = 'True';
```

C++

```
DefaultTrueBoolStr = 'True';
```

File

SysUtils

Description

DefaultTrueBoolStr is 'True'. It is used internally by the StrToBool (see page 4276) and BoolToStr (see page 4066) routines. They assign DefaultTrueBoolStr to the 1st element of the TrueBoolStrs (see page 4363) array, used by these routines as a default string representation of the boolean true value.

See Also

DefaultFalseBoolStr (see page 4370)

StrToBool (see page 4276)

FalseBoolStrs (see page 4352)

TrueBoolStrs (see page 4363)

BoolToStr (see page 4066)

37.7.4 SysUtils.DriveDelim Constant

Specifies the character used to separate the Windows drive from the path in a file name.

Pascal

```
DriveDelim = ':';
```

C++

```
DriveDelim = ':';
```

File

SysUtils

Description

DriveDelim is the ':' character that follows the drive character ('A', 'B' and so on) in a full filename on a Windows Operating System (see page 3159). The value is a null character for Linux Operating Systems since they do not support drive designations.

See Also

PathDelim (see page 4388)

PathSep (see page 4388)

37.7.5 SysUtils.faAnyFile Constant

Specifies any file type.

Pascal

```
faAnyFile = $000001FF;
```

C++

```
faAnyFile = 0x000001FF;
```

File

SysUtils

Description

faAnyFile is principally used by FindFirst (see page 4175) to indicate that files of any type encountered sould be returned. Namely, all files.

See Also

faArchive (see page 4372)

faDirectory (see page 4372)

faHidden (see page 4373)

faReadOnly (see page 4374)

faSymLink (see page 4375)

faSysFile (see page 4375)

FindFirst (see page 4175)

TSearchRec (see page 4330)

37.7.6 SysUtils.faArchive Constant

Specifies only Windows archived file types.

Pascal

```
faArchive = $00000020 platform;
```

C++

```
faArchive = 0x00000020 platform;
```

File

SysUtils

Description

faArchive is principally used by FindFirst (see page 4175) to indicate that only archived files sould be returned. It is Windows specific : Linux does not support archiving.

See Also

faAnyFile (see page 4371)

faDirectory (see page 4372)

faHidden (see page 4373)

faReadOnly (see page 4374)

faSymLink (see page 4375)

faSysFile (see page 4375)

FindFirst (see page 4175)

TSearchRec (see page 4330)

37.7.7 SysUtils.faDirectory Constant

Specifies only directory 'file' types.

Pascal

```
faDirectory = $00000010;
```

C++

```
faDirectory = 0x00000010;
```

File

SysUtils

Description

faDirectory is principally used by FindFirst (see page 4175) to indicate that only directories sould be returned. Whilst FindFirst (see page 4175) is generally used to search for files, it can also return directories within the search directory.

See Also

faAnyFile (see page 4371)

faArchive (see page 4372)

faHidden (see page 4373)

faReadOnly (see page 4374)

faSymLink (see page 4375)

faSysFile (see page 4375)

FindFirst (see page 4175)

TSearchRec (see page 4330)

37.7.8 SysUtils.faHidden Constant

Specifies only hidden file types.

Pascal

```
faHidden = $00000002 platform;
```

C++

```
faHidden = 0x00000002 platform;
```

File

SysUtils

Description

faHidden is principally used by FindFirst (see page 4175) to indicate that only hidden files sould be returned. On Linux systems, hidden files are those that start with a "." character.

See Also

faAnyFile (see page 4371)

faArchive (see page 4372)

faDirectory (see page 4372)

faReadOnly (see page 4374)

faSymLink (see page 4375)

faSysFile (see page 4375)

FindFirst (see page 4175)

TSearchRec (see page 4330)

37.7.9 SysUtils.faNormal Constant

Pascal

```
faNormal = $00000080 platform;
```

C++

```
faNormal = 0x00000080 platform;
```

File

SysUtils

Description

This is constant SysUtils.faNormal.

37.7.10 SysUtils.faReadOnly Constant

Specifies only read only file types.

Pascal

```
faReadOnly = $00000001 platform;
```

C++

```
faReadOnly = 0x00000001 platform;
```

File

SysUtils

Description

faReadOnly is principally used by FindFirst (see page 4175) to indicate that only read only files sould be returned. On Linux systems, read only is not necessarily a system wide attribute - access rights are user and group dependent. Read (see page 3634) only here means that the current application does read but not write to a file.

See Also

faAnyFile (see page 4371)

faArchive (see page 4372)

faDirectory (see page 4372)

faHidden (see page 4373)

faSymLink (see page 4375)

faSysFile (see page 4375)

FindFirst (see page 4175)

TSearchRec (see page 4330)

37.7.11 SysUtils.faSymLink Constant

Specifies only symbolic link file types.

Pascal

```
faSymLink = $00000040 platform;
```

C++

```
faSymLink = 0x00000040 platform;
```

File

SysUtils

Description

faSymLink is principally used by FindFirst (see page 4175) to indicate that only symbolic link files sould be returned. Symbolic links are a feature of Linux systems, with no Windows counterpart.

See Also

faAnyFile (see page 4371)

faArchive (see page 4372)

faDirectory (see page 4372)

faHidden (see page 4373)

faReadOnly (see page 4374)

faSysFile (see page 4375)

FindFirst (see page 4175)

TSearchRec (see page 4330)

37.7.12 SysUtils.faSysFile Constant

Specifies only system file types.

Pascal

```
faSysFile = $00000004 platform;
```

C++

```
faSysFile = 0x00000004 platform;
```

File

SysUtils

Description

faSysFile is principally used by FindFirst (see page 4175) to indicate that only system files sould be returned. On Linux systems, a system file is one that is treated as a socket, symbolic link, device file, or FIFO.

See Also

faAnyFile (see page 4371)

faArchive (see page 4372)

faDirectory (see page 4372)

faHidden (see page 4373)

faReadOnly (see page 4374)

faSymLink (see page 4375)

FindFirst (see page 4175)

TSearchRec (see page 4330)

37.7.13 SysUtils.faTemporary Constant

Pascal

```
faTemporary = $00000100 platform;
```

C++

```
faTemporary = 0x00000100 platform;
```

File

SysUtils

Description

This is constant SysUtils.faTemporary.

37.7.14 SysUtils.faVolumeID Constant

Specifies Volume ID file types.

Pascal

```
faVolumeID = $00000008 platform deprecated;
```

C++

```
faVolumeID = 0x00000008 platform deprecated;
```

File

SysUtils

Description

faVolumeID is deprecated. Use your operating system functions to obtain this information.

Note: faVolumeID was used only for DOS/9x compatibility.

See Also

faArchive (see page 4372)

faDirectory (see page 4372)

faHidden (see page 4373)

faReadOnly (see page 4374)

faSymLink (see page 4375)

faSysFile (see page 4375)

FindFirst (see page 4175)

37.7.15 SysUtils.fmOpenRead Constant

Specifies a file should be opened for reading only.

Pascal

```
fmOpenRead = $0000;
```

C++

```
fmOpenRead = 0x0000;
```

File

SysUtils

Description

fmOpenRead is one of the values you can assign to FileMode (see page 3824). It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenRead, subsequent calls to Reset (see page 3643) open files for reading only.

Note: You must use this value when you plan to read a read only file - using fmOpenReadWrite (see page 4378) will cause the open to fail because it will attempt and fail to get write access.

See Also

fmOpenReadWrite (see page 4378)

fmOpenWrite (see page 4378)

fmShareCompat (see page 4379)

fmShareDenyNone (see page 4380)

fmShareDenyRead (see page 4380)

fmShareDenyWrite (see page 4381)

fmShareExclusive (see page 4382)

Reset (see page 3643)

FileMode (see page 3824)

Create

37.7.16 SysUtils.fmOpenReadWrite Constant

Specifies a file should be opened for reading and writing.

Pascal

```
fmOpenReadWrite = $0002;
```

C++

```
fmOpenReadWrite = 0x0002;
```

File

SysUtils

Description

fmOpenReadWrite is one of the values you can assign to FileMode (see page 3824). It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenReadWrite, subsequent calls to Reset (see page 3643) open files for reading and writing.

Note: Do not assign this value when you plan to read a read only file - the open will fail because write access will be attempted but be denied.

See Also

fmOpenRead (see page 4377)

fmOpenWrite (see page 4378)

fmShareCompat (see page 4379)

fmShareDenyNone (see page 4380)

fmShareDenyRead (see page 4380)

fmShareDenyWrite (see page 4381)

fmShareExclusive (see page 4382)

Reset (see page 3643)

FileMode (see page 3824)

Create

37.7.17 SysUtils.fmOpenWrite Constant

Specifies a file should be opened for writing only.

Pascal

```
fmOpenWrite = $0001;
```

C++

```
fmOpenWrite = 0x0001;
```

File

SysUtils

Description

fmOpenWrite is one of the values you can assign to FileMode (see page 3824). It is also one of the attributes assignable in the Create. The TFileStream is used to read and write files. When you assign the value fmOpenWrite, subsequent calls to Reset (see page 3643) open files for writing only.

See Also

fmOpenRead (see page 4377)

fmOpenReadWrite (see page 4378)

fmShareCompat (see page 4379)

fmShareDenyNone (see page 4380)

fmShareDenyRead (see page 4380)

fmShareDenyWrite (see page 4381)

fmShareExclusive (see page 4382)

Reset (see page 3643)

FileMode (see page 3824)

Create

37.7.18 SysUtils.fmShareCompat Constant

Specifies file sharing that is FCB compatible.

Pascal

```
fmShareCompat = $0000 platform;
```

C++

```
fmShareCompat = 0x0000 platform;
```

File

SysUtils

Description

fmShareCompat is one of the attributes assignable in the Create. The TFileStream is used to read and write files. It specifies file sharing that is compatible with the way that FCB's (File Control Blocks) are opened.

See Also

fmOpenRead (see page 4377)

fmOpenReadWrite (see page 4378)

fmOpenWrite (see page 4378)

fmShareDenyNone (see page 4380)

fmShareDenyRead (see page 4380)

fmShareDenyWrite (see page 4381)

fmShareExclusive (see page 4382)

Create

37.7.19 SysUtils.fmShareDenyNone Constant

Specifies file sharing that does not prevent file access by other applications.

Pascal

```
fmShareDenyNone = $0040;
```

C++

```
fmShareDenyNone = 0x0040;
```

File

SysUtils

Description

fmShareDenyNone is one of the attributes assignable in the Create. The TFileStream is used to read and write files. It specifies file sharing that does not prevent any other applications from reading from or writing to the file.

See Also

fmOpenRead (see page 4377)

fmOpenReadWrite (see page 4378)

fmOpenWrite (see page 4378)

fmShareCompat (see page 4379)

fmShareDenyRead (see page 4380)

fmShareDenyWrite (see page 4381)

fmShareExclusive (see page 4382)

Create

37.7.20 SysUtils.fmShareDenyRead Constant

Specifies file sharing that prevents read file access by other applications.

Pascal

```
fmShareDenyRead = $0030 platform;
```

C++

```
fmShareDenyRead = 0x0030 platform;
```

File

SysUtils

Description

fmShareDenyRead is one of the attributes assignable in the Create. The TFileStream is used to read and write files. It specifies file sharing that prevents other applications from reading the file. Other applications may open the file for writing only.

See Also

fmOpenRead (see page 4377)

fmOpenReadWrite (see page 4378)

fmOpenWrite (see page 4378)

fmShareCompat (see page 4379)

fmShareDenyNone (see page 4380)

fmShareDenyWrite (see page 4381)

fmShareExclusive (see page 4382)

Create

37.7.21 SysUtils.fmShareDenyWrite Constant

Specifies file sharing that prevents write file access by other applications.

Pascal

```
fmShareDenyWrite = $0020;
```

C++

```
fmShareDenyWrite = 0x0020;
```

File

SysUtils

Description

fmShareDenyWrite is one of the attributes assignable in the Create. The TFileStream is used to read and write files. It specifies file sharing that prevents other applications from writing to the file. Other applications may open the file for reading only.

See Also

fmOpenRead (see page 4377)

fmOpenReadWrite (see page 4378)

fmOpenWrite (see page 4378)

fmShareCompat (see page 4379)

fmShareDenyNone (see page 4380)

fmShareDenyRead (see page 4380)

fmShareExclusive (see page 4382)

Create

37.7.22 SysUtils.fmShareExclusive Constant

Specifies file sharing that prevents any file access by other applications.

Pascal

```
fmShareExclusive = $0010;
```

C++

```
fmShareExclusive = 0x0010;
```

File

SysUtils

Description

fmShareExclusive is one of the attributes assignable in the Create. The TFileStream is used to read and write files. It specifies file sharing that prevents other applications from opening the file at all.

See Also

fmOpenRead (see page 4377)

fmOpenReadWrite (see page 4378)

fmOpenWrite (see page 4378)

fmShareCompat (see page 4379)

fmShareDenyNone (see page 4380)

fmShareDenyRead (see page 4380)

fmShareDenyWrite (see page 4381)

Create

37.7.23 SysUtils.HoursPerDay Constant

Specifies the number of hours in a day.

Pascal

```
HoursPerDay = 24;
```

C++

```
HoursPerDay = 24;
```

File

SysUtils

Description

HoursPerDay is the number of hours (24) in a day. Using this constant rather than hard coding 24 makes for more readable code.

See Also

DateDelta (see page 4369)

MinsPerDay (see page 4385)

MinsPerHour (see page 4386)

MSecsPerDay (see page 4387)

MSecsPerSec (see page 4387)

SecsPerDay (see page 4397)

SecsPerMin (see page 4398)

37.7.24 SysUtils.MaxCurrency Constant

Defines the largest value supported for Currency (see page 3235) data types.

Pascal

```
MaxCurrency: Currency = 922337203685477.5807;
```

C++

```
Currency MaxCurrency = 922337203685477.5807;
```

File

SysUtils

Description

MaxCurrency is the largest value that can be held by a Currency (see page 3235) data variable on the Windows Opersting System (see page 3159).

Note: Linux supports a value 1.0 less than this maximum (922337203685476.5807).

See Also

MinCurrency (see page 4384)

37.7.25 SysUtils.MaxDateTime Constant

Specifies the highest supported TDateTime (see page 3323) value.

Pascal

```
MaxDateTime: TDateTime = 2958465.99999;
```

C++

```
TDateTime MaxDateTime = 2958465.99999;
```

File

SysUtils

Description

MaxDateTime is the most future date : 12/31/9999 23:59:59.999 that the TDateTime (see page 3323) type supports. The actual shown value of 2958465.99999 reflects the fact that TDateTime (see page 3323) is actually a floating point number, rather than a structured type. The mantissa holds the number of days relative to 12/30/1899 00:00:00.000, the exponent holds a fraction of a day.

See Also

TDateTime (see page 3787)

MinDateTime (see page 4384)

37.7.26 SysUtils.MaxEraCount Constant

Pascal

```
MaxEraCount = 7;
```

C++

```
MaxEraCount = 7;
```

File

SysUtils

Description

This is constant SysUtils.MaxEraCount.

37.7.27 SysUtils.MinCurrency Constant

Defines the largest negative value supported for Currency (see page 3235) data types.

Pascal

```
MinCurrency: Currency = -922337203685477.5807;
```

C++

```
Currency MinCurrency = -922337203685477.5807;
```

File

SysUtils

Description

MinCurrency is the largest negative value that can be held by a Currency (see page 3235) data variable on the Windows Opersting System (see page 3159).

Note: Linux supports a value 1.0 less negative than this minimum (-922337203685476.5807).

See Also

MaxCurrency (see page 4383)

37.7.28 SysUtils.MinDateTime Constant

Specifies the earliest supported TDateTime (see page 3323) value.

Pascal

```
MinDateTime: TDateTime = -657434.0;
```

C++

```
TDateTime MinDateTime = -657434.0;
```

File

SysUtils

Description

MinDateTime is the most historic date : 01/01/0100 00:00:00.000 that the TDateTime (see page 3323) type supports. The actual shown value of -657434.0 reflects the fact that TDateTime (see page 3323) is actually a floating point number, rather than a structured type. The mantissa holds the number of days relative to 12/30/1899 00:00:00.000, the exponent holds a fraction of a day.

Note: The fractional part of a TDateTime (see page 3323) is always treated in a positive sense. It always means the elapsed part of a day - if the TDateTime (see page 3323) value is negative, this refers only to a negative offset of the integral day value from 12/30/1899.

See Also

TDateTime (see page 3787)

MaxDateTime (see page 4383)

37.7.29 SysUtils.MinsPerDay Constant

Specifies the number of minutes in a day.

Pascal

```
MinsPerDay = HoursPerDay * MinsPerHour;
```

C++

```
MinsPerDay = HoursPerDay * MinsPerHour;
```

File

SysUtils

Description

MinsPerDay is the number of minutes (1440) in a day. It is the product of HoursPerDay (see page 4382) and MinsPerHour (see page 4386).

Using this constant rather than hard coding 1440 makes for more readable code.

See Also

DateDelta (see page 4369)

HoursPerDay (see page 4382)

MinsPerHour (see page 4386)

MSecsPerDay (see page 4387)

MSecsPerSec (see page 4387)

SecsPerDay (see page 4397)

SecsPerMin (see page 4398)

37.7.30 SysUtils.MinsPerHour Constant

Specifies the number of minutes in an hour.

Pascal

```
MinsPerHour = 60;
```

C++

```
MinsPerHour = 60;
```

File

SysUtils

Description

MinsPerHour is the number of minutes (60) in an hour. Using this constant rather than hard coding 60 makes for more readable code.

See Also

DateDelta (see page 4369)

HoursPerDay (see page 4382)

MSecsPerDay (see page 4387)

MSecsPerSec (see page 4387)

SecsPerDay (see page 4397)

SecsPerMin (see page 4398)

37.7.31 SysUtils.MonthDays Constant

Defines the number of days in each month for normal and leap years.

Pascal

```
MonthDays: array [Boolean] of TDayTable = ((31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31));
```

C++

```
array [Boolean] of TDayTable MonthDays = ((31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31));
```

File

SysUtils

Description

MonthDays defines a 2 dimensional array of TDayTable (see page 4340) values. Namely, an array of two sets of 12 words. The first set holds the number of days in each month in a normal year, the second set the number in a leap year.

Use MonthDays[IsLeapYear (see page 4220)(Y), M] to find the number of days in Year 'Y', month 'M'.

See Also

DaysInMonth

IsLeapYear (see page 4220)

TDayTable (see page 4340)

37.7.32 SysUtils.MSecsPerDay Constant

Specifies the number of milliseconds in a day.

Pascal

```
MSecsPerDay = SecsPerDay * MSecsPerSec;
```

C++

```
MSecsPerDay = SecsPerDay * MSecsPerSec;
```

File

SysUtils

Description

MSecsPerDay is the number of milliseconds (86,400,000) in a day. It is the product of SecsPerDay (see page 4397) and MSecsPerSec (see page 4387).

Using this constant rather than hard coding 86400000 makes for more readable code.

See Also

DateDelta (see page 4369)

HoursPerDay (see page 4382)

MinsPerDay (see page 4385)

MinsPerHour (see page 4386)

MSecsPerSec (see page 4387)

SecsPerDay (see page 4397)

SecsPerMin (see page 4398)

37.7.33 SysUtils.MSecsPerSec Constant

Specifies the number of milliseconds in a second.

Pascal

```
MSecsPerSec = 1000;
```

C++

```
MSecsPerSec = 1000;
```

File

SysUtils

Description

MSecsPerSec is the number of milliseconds (1000) in a second.

Using this constant rather than hard coding 1000 makes for more readable code.

See Also

DateDelta (see page 4369)

HoursPerDay (see page 4382)

MinsPerDay (see page 4385)

MinsPerHour (see page 4386)

MSecsPerDay (see page 4387)

SecsPerMin (see page 4398)

SecsPerDay (see page 4397)

37.7.34 SysUtils.PathDelim Constant

Specifies the character used to separate each directory name in a file path.

Pascal

```
PathDelim = '\';
```

C++

```
PathDelim = '\';
```

File

SysUtils

Description

PathDelim is the '\' character for Windows, or '/' for Linux that is used to separate each directory in the hierarchy of directories in a path. A path is used to navigate to a specific directory or file(s).

See Also

DriveDelim (see page 4371)

PathSep (see page 4388)

37.7.35 SysUtils.PathSep Constant

Specifies the character used to separate a sequence of file or directory names.

Pascal

```
PathSep = ';';
```

C++

```
PathSep = ';';
```

File

SysUtils

Description

PathSep is the ';' character for Windows, or ':' for Linux that is used to separate filenames in a sequence of file or directory names. FileSearch (see page 4156), for example, expects PathSep to be used to separate the values in the string of directory names.

See Also

DriveDelim (see page 4371)

FileSearch (see page 4156)

37.7.36 SysUtils.pfBCB4Produced Constant

Indicates that a package was produced by via C++.

Pascal

```
pfBCB4Produced = $08000000;
```

C++

```
pfBCB4Produced = 0x08000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfBCB4Produced is one of these flag values, indicating that a package was produced by C++. BCB stands for Borland C++ Builder.

See Also

GetPackageInfo (see page 4209)

37.7.37 SysUtils.pfConsumerBCB Constant

Indicates that C++ can be a consumer of a package.

Pascal

```
pfConsumerBCB = $00200000;
```

C++

```
pfConsumerBCB = 0x00200000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfConsumerBCB is one of these flag values, indicating that C++ is a legitimate consumer, or user, of a package. The C++ consumer is the Borland C++ Builder compiler, linking in the package.

A package will be marked with the pfConsumerBCB when the -JL flag has been set during compilation. This ensures that the extra support files (.lib, .bpi, .hpp)required by a C++ consumer are generated and included in the package.

See Also

GetPackageInfo (see page 4209)

37.7.38 SysUtils.pfConsumerCompat Constant

Pascal

```
pfConsumerCompat = $00000000;
```

C++

```
pfConsumerCompat = 0x00000000;
```

File

SysUtils

Description

This is constant SysUtils.pfConsumerCompat.

37.7.39 SysUtils.pfConsumerDelphi Constant

Indicates that Delphi can be a consumer of a package.

Pascal

```
pfConsumerDelphi = $00100000;
```

C++

```
pfConsumerDelphi = 0x00100000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfConsumerDelphi is one of these flag values, indicating that Delphi is a legitimate consumer, or user, of a package. The Delphi consumer is the Delphi compiler, linking in the package.

See Also

GetPackageInfo (see page 4209)

37.7.40 SysUtils.pfConsumerMask Constant

Indicates the comsumer of a package.

Pascal

```
pfConsumerMask = $00F00000;
```

C++

```
pfConsumerMask = 0x00F00000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfConsumerMask is a mask that you can use to detect the consumers can use a package. This mask covers these flags : pfConsumerBCB (see page 4389), pfConsumerCompat (see page 4390) and pfConsumerDelphi (see page 4390) flags.

Note: pfConsumerMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 4209)

pfConsumerBCB (see page 4389)

pfConsumerCompat (see page 4390)

pfConsumerDelphi (see page 4390)

37.7.41 SysUtils.pfDelphi4Produced Constant

Indicates that a package was produced by Delphi 4 or later.

Pascal

```
pfDelphi4Produced = $0C000000;
```

C++

```
pfDelphi4Produced = 0x0C000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfDelphi4Produced is one of these flag values, indicating that the a package was produced by Delphi 4 or later. Packages were introduced in Delphi 3.

See Also

GetPackageInfo (see page 4209)

37.7.42 SysUtils.pfDesignOnly Constant

Indicates that a package was compiled with {$DESIGNONLY ON}.

Pascal

```
pfDesignOnly = $00000002;
```

C++

```
pfDesignOnly = 0x00000002;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfDesignOnly is one of these flag values, indicating that the package was compiled with the $DESIGNONLY compiler directive set ON. This means that the for package is intended for installation in the IDE.

See Also

GetPackageInfo (see page 4209)

37.7.43 SysUtils.pfExeModule Constant

Indicates that a module is an executable.

Pascal

```
pfExeModule = $00000000;
```

C++

```
pfExeModule = 0x00000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfExeModule is one of these flag values, indicating that the module is an executable.

See Also

GetPackageInfo (see page 4209)

37.7.44 SysUtils.pfIgnoreDupUnits Constant

Indicates that a package was compiled with {$DUPLICATEENABLE ON}.

Pascal

```
pfIgnoreDupUnits = $00000008;
```

C++

```
pfIgnoreDupUnits = 0x00000008;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfIgnoreDupUnits is one of these flag values, indicating that the package was compiled with the $DUPLICATEENABLE compiler directive set ON.

See Also

GetPackageInfo (see page 4209)

37.7.45 SysUtils.pfLibraryModule Constant

Indicates that a module is a library.

Pascal

```
pfLibraryModule = $80000000;
```

C++

```
pfLibraryModule = 0x80000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfLibraryModule is one of these flag values, indicating that the module is a library (.dll file extension).

See Also

GetPackageInfo (see page 4209)

37.7.46 SysUtils.pfModuleTypeMask Constant

Indicates that a package is a module.

Pascal

```
pfModuleTypeMask = $C0000000;
```

C++

```
pfModuleTypeMask = 0xC0000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfModuleTypeMask is a mask that you can use to detect whether the package is a module. This mask is the binary AND of the pfExeModule (see page 4392), pfPackageModule (see page 4394) and pfLibraryModule (see page 4393) flags.

Note: pfModuleTypeMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 4209)

37.7.47 SysUtils.pfNeverBuild Constant

Indicates that a package was compiled with the -z flag.

Pascal

```
pfNeverBuild = $00000001;
```

C++

```
pfNeverBuild = 0x00000001;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfNeverBuild is one of these flag values, indicating that the package was compiled with the -z flag.

See Also

GetPackageInfo (see page 4209)

37.7.48 SysUtils.pfPackageModule Constant

Indicates that a module is a package.

Pascal

```
pfPackageModule = $40000000;
```

C++

```
pfPackageModule = 0x40000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfPackageModule is one of these flag values, indicating that the module is a package (.bpl file extension).

See Also

GetPackageInfo (see page 4209)

37.7.49 SysUtils.pfProducerMask Constant

Indicates the producer of a package.

Pascal

```
pfProducerMask = $0C000000;
```

C++

```
pfProducerMask = 0x0C000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfProducerMask is a mask that you can use to detect the producer of a package. This mask is the binary AND of the pfBCB4Produced (see page 4389), pfDelphi4Produced (see page 4391), pfProducerUndefined (see page 4395) and pfV3Produced (see page 4396) flags.

Note: pfProducerMask is not itself a returned flag - it is simply a mask.

See Also

GetPackageInfo (see page 4209)

pfBCB4Produced (see page 4389)

pfDelphi4Produced (see page 4391)

pfProducerUndefined (see page 4395)

pfV3Produced (see page 4396)

37.7.50 SysUtils.pfProducerUndefined Constant

Indicates that a package was produced by an unknown developmemt environment.

Pascal

```
pfProducerUndefined = $04000000;
```

C++

```
pfProducerUndefined = 0x04000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfProducerUndefined is one of these flag values, indicating that the producer of a package was undefined.

See Also

GetPackageInfo (see page 4209)

37.7.51 SysUtils.pfRunOnly Constant

Indicates that a package was compiled with {$RUNONLY ON}.

Pascal

```
pfRunOnly = $00000004;
```

C++

```
pfRunOnly = 0x00000004;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfRunOnly is one of these flag values, indicating that the package was compiled with the $RUNONLY compiler directive set ON. The {$RUNONLY ON} directive causes the package where it occurs to be compiled as runtime only. Packages compiled with {$RUNONLY ON} cannot be installed as design-time packages in the IDE.

See Also

GetPackageInfo (see page 4209)

37.7.52 SysUtils.pfV3Produced Constant

Indicates that a package was produced by Delphi 3 or later.

Pascal

```
pfV3Produced = $00000000;
```

C++

```
pfV3Produced = 0x00000000;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about a package. It returns a set of flags.

pfV3Produced is one of these flag values, indicating that the a package was produced by Delphi 3 or later. Packages were introduced in Delphi 3.

See Also

GetPackageInfo (see page 4209)

37.7.53 SysUtils.SecsPerDay Constant

Specifies the number of seconds in a day.

Pascal

```
SecsPerDay = MinsPerDay * SecsPerMin;
```

C++

```
SecsPerDay = MinsPerDay * SecsPerMin;
```

File

SysUtils

Description

SecsPerDay is the number of seconds (86,400) in a day. It is the product of MinsPerDay (see page 4385) and SecsPerMin (see page 4398).

Using this constant rather than hard coding 86400 makes for more readable code.

See Also

DateDelta (see page 4369)

HoursPerDay (see page 4382)

MinsPerDay (see page 4385)

MinsPerHour (see page 4386)

MSecsPerDay (see page 4387)

MSecsPerSec (see page 4387)

SecsPerMin (see page 4398)

37.7.54 SysUtils.SecsPerHour Constant

Pascal

```
SecsPerHour = SecsPerMin * MinsPerHour;
```

C++

```
SecsPerHour = SecsPerMin * MinsPerHour;
```

File

SysUtils

Description

This is constant SysUtils.SecsPerHour.

37.7.55 SysUtils.SecsPerMin Constant

Specifies the number of seconds in a minute.

Pascal

```
SecsPerMin = 60;
```

C++

```
SecsPerMin = 60;
```

File

SysUtils

Description

SecsPerMin is the number of seconds (60) in a minute.

Using this constant rather than hard coding 60 makes for more readable code.

See Also

DateDelta (see page 4369)

HoursPerDay (see page 4382)

MinsPerDay (see page 4385)

MinsPerHour (see page 4386)

MSecsPerDay (see page 4387)

MSecsPerSec (see page 4387)

SecsPerDay (see page 4397)

37.7.56 SysUtils.SwitchChars Constant

Specifies the prefix character(s) for program command line arguments.

Pascal

```
SwitchChars = ['/','-'];
```

C++

```
SwitchChars = ['/','-'];
```

File

SysUtils

Description

When a program is started, an optional 'command line' string may be supplied. Such a string may specify program operational arguments, or 'switches', prefixed by a separator character. SwitchChars defines these separator characters as either '-' or '/' on a Windows system, and as '-' on Linux systems.

See Also

FindCmdLineSwitch (see page 4174)

37.7.57 SysUtils.ufImplicitUnit Constant

Indicates that a unit was implicitly imported into a package.

Pascal

```
ufImplicitUnit = $10;
```

C++

```
ufImplicitUnit = 0x10;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about the units within a package. TPackageInfoProc (see page 4342) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufImplicitUnit is set, then the unit was implicitly imported into a package. This means that the unit is not used in another package required by this package, but is used by one of the units this package contains.

See Also

GetPackageInfo (see page 4209)

TPackageInfoProc (see page 4342)

37.7.58 SysUtils.ufMainUnit Constant

Indicates that a unit is the main one in a project.

Pascal

```
ufMainUnit = $01;
```

C++

```
ufMainUnit = 0x01;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about the units within a package. TPackageInfoProc (see page 4342) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufMainUnit is set, then the unit is a Main unit.

See Also

GetPackageInfo (see page 4209)

TPackageInfoProc (see page 4342)

37.7.59 SysUtils.ufOrgWeakUnit Constant

Pascal

```
ufOrgWeakUnit = $08;
```

C++

```
ufOrgWeakUnit = 0x08;
```

File

SysUtils

Description

This is constant SysUtils.ufOrgWeakUnit.

37.7.60 SysUtils.ufPackageUnit Constant

Indicates that a unit is a package.

Pascal

```
ufPackageUnit = $02;
```

C++

```
ufPackageUnit = 0x02;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about the units within a package. TPackageInfoProc (see page 4342) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufPackageUnit is set, then the unit is a package unit.

See Also

GetPackageInfo (see page 4209)

TPackageInfoProc (see page 4342)

37.7.61 SysUtils.ufWeakPackageUnit Constant

Indicates that a unit is a either a weak unit or a package unit.

Pascal

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit;
```

C++

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit;
```

File

SysUtils

Description

ufWeakPackageUnit is used internally in Delphi. It is equal to the logical or of ufWeakUnit (see page 4401) and ufPackageUnit (see page 4400)

37.7.62 SysUtils.ufWeakUnit Constant

Indicates that a unit is a weak unit.

Pascal

```
ufWeakUnit = $04;
```

C++

```
ufWeakUnit = 0x04;
```

File

SysUtils

Description

The GetPackageInfo (see page 4209) function returns information about the units within a package. TPackageInfoProc (see page 4342) is one of the parameters to that function : it is a callback procedure that is invoked for each unit in the package. It passes a set of flags associated with each unit.

If the ufWeakUnit is set, then the unit was compiled with the $WEAKPACKAGEUNIT compiler directive set.

See Also

GetPackageInfo (see page 4209)

TPackageInfoProc (see page 4342)

37.7.63 SysUtils.UnixDateDelta Constant

Specifies the difference between TDateTime (see page 3323) and Lunix date/time origins.

Pascal

```
UnixDateDelta = 25569;
```

C++

```
UnixDateDelta = 25569;
```

File

SysUtils

Description

UnixDateDelta is the number of days (25569) between December 30, 1899 (starting TDateTime (see page 3323) value), and Jamuary 1, 1970 (starting value for Linux date time values). This value is useful for converting date-time values between TDateTime (see page 3323) and Linux file date time values.

See Also

TDateTime (see page 3787)

TTimeStamp (see page 4334)

DateDelta (see page 4369)

38 TabNotBk Namespace

This is namespace TabNotBk.

Topics

Name	Description
Classes (see page 4404)	The following table lists classes in this documentation.
Types (see page 4454)	The following table lists types in this documentation.
Constants (see page 4455)	The following table lists constants in this documentation.

Classes

	Name	Description
	TTabbedNotebook (see page 4404)	TTabbedNotebook contains multiple pages, each with its own set of controls.
	TTabPage (see page 4443)	TTabPage implements an individual tab page of a tabbed notebook control.

Constants

Name	Description
CM_TABFONTCHANGED (see page 4455)	This is constant TabNotBk.CM_TABFONTCHANGED.

Types

Name	Description
TPageChangeEvent (see page 4454)	TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 4404) control.

38.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TTabbedNotebook (see page 4404)	TTabbedNotebook contains multiple pages, each with its own set of controls.
	TTabPage (see page 4443)	TTabPage implements an individual tab page of a tabbed notebook control.

38.1.1 TTabbedNotebook Class

TTabbedNotebook contains multiple pages, each with its own set of controls.

Class Hierarchy

Pascal

```
TTabbedNotebook = class(TCustomTabControl);
```

C++

```
class TTabbedNotebook : public TCustomTabControl;
```

File

TabNotBk

Description

TTabbedNotebook is provided for backwards compatibility. New (see page 3619) applications should use TPageControl instead.

38.1.1.1 TTabbedNotebook Members

The following tables list the members exposed by TTabbedNotebook.

Protected Methods

	Name	Description
	AlignControls (see page 4407)	Aligns any controls for which the control is the parent within a specified area of the control.
	Change (see page 4407)	Pending tab change event dispatcher.
	Click (see page 4407)	Respond to user click.
	CreateHandle (see page 4408)	Creates underlying screen object.
	CreateParams (see page 4409)	Sets up the window-creation parameters for the tab control.
	GetChildOwner (see page 4409)	Returns the owner of a child component being read from a stream.
	KeyDown (see page 4410)	Respond to key press events.
	Loaded (see page 4411)	Form load event dispatcher.
	ReadState (see page 4411)	Prepares the control for having its properties assigned values from a stream.
	SetPageIndex (see page 4412)	This is SetPageIndex, a member of class TTabbedNotebook.
	ShowControl (see page 4412)	Ensures that a specified child control is visible.

Protected Properties

	Name	Description
	Tabs (see page 4431)	Contains the list of text strings that label the tabs of the tab control.

Public Methods

	Name	Description
	CMTabFontChanged (see page 4408)	This is CMTabFontChanged, a member of class TTabbedNotebook.
V	Create (see page 4405)	Constructs a tabbed notebook component.
V	Destroy (see page 4406)	Destroys a tabbed notebook component.
V	GetChildren (see page 4409)	Calls a specified method for each child of the control.
	GetIndexForPage (see page 4410)	Returns the PageIndex (see page 4421) value of the specified page.
	TabFontChanged (see page 4412)	Invalidates the tabs when the TabFont (see page 4430) property changes.

Public Properties

	Name	Description
R	TopFont (see page 4433)	Returns the current value of font used on the tabs of the tabbed notebook.

Published Events

	Name	Description
	OnChange (see page 4435)	Occurs when the page of a tabbed notebook changes.
	OnClick (see page 4436)	Occurs when a tab of a tabbed notebook is selected.
	OnContextPopup (see page 4436)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 4437)	Occurs when a control receives the input focus.
	OnExit (see page 4439)	Occurs when the input focus shifts away from one control to another.

Published Properties

	Name	Description
	ActivePage (see page 4413)	Determines which page displays in the tabbed notebook control.
	Align (see page 4414)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4417)	Specifies how the control is anchored to its parent.
	Constraints (see page 4418)	Specifies the size constraints for the control.
	DoubleBuffered (see page 4418)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 4420)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4420)	Controls the attributes of text written on or in the control.
	PageIndex (see page 4421)	Determines which page displays in the tabbed notebook component.
	Pages (see page 4422)	Contains the strings that identify the individual pages of the tabbed notebook control.
	ParentDoubleBuffered (see page 4422)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4423)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4423)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4428)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabFont (see page 4430)	Determines the font used on the tabs of the tabbed notebook control.
	TabOrder (see page 4430)	Indicates the position of the control in its parent's tab order.
	TabsPerRow (see page 4432)	TabsPerRow is obsolete and retained only for backwards compatibility.
	TabStop (see page 4433)	Determines if the user can tab to a control.
	Visible (see page 4433)	Determines whether the component appears onscreen.

38.1.1.2 TTabbedNotebook.Create Constructor

Constructs a tabbed notebook component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabbedNotebook(TComponent * AOwner);
```

Description

Call Create TTabbedNotebook (see page 4404) indirectly, using the new keyword, to instantiate a tabbed notebook at runtime. Tabbed notebooks added to forms at design time are created automatically.

The AOwner parameter assigns a value to the notebook's Owner property.

See Also

Owner

38.1.1.3 TTabbedNotebook.Destroy Destructor

Destroys a tabbed notebook component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabbedNotebook();
```

Description

Do not call Destroy directly in an application, Instead, use Free method, which checks that the notebook is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

38.1.1.4 TTabbedNotebook Methods

The methods of the TTabbedNotebook class are listed here.

Protected Methods

	Name	Description
	AlignControls (see page 4407)	Aligns any controls for which the control is the parent within a specified area of the control.
	Change (see page 4407)	Pending tab change event dispatcher.
	Click (see page 4407)	Respond to user click.
	CreateHandle (see page 4408)	Creates underlying screen object.
	CreateParams (see page 4409)	Sets up the window-creation parameters for the tab control.
	GetChildOwner (see page 4409)	Returns the owner of a child component being read from a stream.
	KeyDown (see page 4410)	Respond to key press events.
	Loaded (see page 4411)	Form load event dispatcher.
	ReadState (see page 4411)	Prepares the control for having its properties assigned values from a stream.
	SetPageIndex (see page 4412)	This is SetPageIndex, a member of class TTabbedNotebook.
	ShowControl (see page 4412)	Ensures that a specified child control is visible.

Public Methods

	Name	Description
	CMTabFontChanged (see page 4408)	This is CMTabFontChanged, a member of class TTabbedNotebook.
V	GetChildren (see page 4409)	Calls a specified method for each child of the control.
	GetIndexForPage (see page 4410)	Returns the PageIndex (see page 4421) value of the specified page.
	TabFontChanged (see page 4412)	Invalidates the tabs when the TabFont (see page 4430) property changes.

38.1.1.4.1 TTabbedNotebook.AlignControls Method

Aligns any controls for which the control is the parent within a specified area of the control.

Pascal

```
procedure AlignControls(AControl: TControl; var Rect: TRect); override;
```

C++

```
virtual __fastcall AlignControls(TControl * AControl, TRect Rect);
```

Description

Use AlignControls to align all controls within an area of a form, panel, group box, scroll box, or any container control. Specify the area in which to align the controls as the value of the Rect parameter.

AlignControls uses the Align property value for each child control to determine how to align it. The AControl parameter can be nil (Delphi) or NULL (C++). If you specify a control in AControl, that control takes precedence in alignment over other, similarly-aligned controls.

See Also

Align

38.1.1.4.2 TTabbedNotebook.Change Method

Pending tab change event dispatcher.

Pascal

```
procedure Change; override;
```

C++

```
virtual __fastcall Change();
```

Description

Change is called automatically when the selected tab changes. Override Change to specify additional processing before the OnChange event handler executes, or to block the OnChange event.

See Also

OnChange

38.1.1.4.3 TTabbedNotebook.Click Method

Respond to user click.

Pascal

```
procedure Click; override;
```

C++

```
virtual __fastcall Click();
```

Description

Click is called automatically when the user left-clicks the control. Component or application code can call Click to simulate a user mouse click. This is often done in menu actions and hotkey handlers.

As implemented in TControl, Click queries whether the OnClick event handler exists and is different from the OnExecute handler for the control's Action. If this is true, the OnClick event handler is called. If this is false, and the Action OnExecute event handler exists, the Action's Execute method is called. Override Click to provide additional behavior.

See Also

Action

OnExecute

Execute

DblClick

OnClick

38.1.1.4.4 TTabbedNotebook.CMTabFontChanged Method

Pascal

```
procedure CMTabFontChanged(var Message: TMessage); message CM_TABFONTCHANGED;
```

C++

```
__fastcall CMTabFontChanged(TMessage Message);
```

Description

This is CMTabFontChanged, a member of class TTabbedNotebook.

38.1.1.4.5 TTabbedNotebook.CreateHandle Method

Creates underlying screen object.

Pascal

```
procedure CreateHandle; override;
```

C++

```
virtual __fastcall CreateHandle();
```

Description

CreateHandle creates the Microsoft Windows screen object that underlies the control. if it does not already exist. The new window handle becomes the value of the Handle property.

CreateHandle creates or locates the parent control's window before it creates a window for the control itself.

Note: If the Parent property is nil (Delphi) or NULL (C++), CreateHandle raises an exception.

See Also

CreateWindowHandle

CreateWnd

Handle

Parent

38.1.1.4.6 TTabbedNotebook.CreateParams Method

Sets up the window-creation parameters for the tab control.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

Tab controls call CreateParams internally when the window for the tab control must be generated. CreateParams overrides the inherited method to specify a tab control window with the current property settings.

See Also

CreateWnd

38.1.1.4.7 TTabbedNotebook.GetChildOwner Method

Returns the owner of a child component being read from a stream.

Pascal

```
function GetChildOwner: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetChildOwner();
```

Description

GetChildOwner is used internally by the component streaming system. It is rarely necessary to call it directly.

In TComponent, GetChildOwner always returns nil (Delphi) or NULL (C++), indicating that the owner is the root component currently being read (usually a form or data module). The Owner of a component is responsible for destroying it.

See Also

Create

Owner

38.1.1.4.8 TTabbedNotebook.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

38.1.1.4.9 TTabbedNotebook.GetIndexForPage Method

Returns the PageIndex (see page 4421) value of the specified page.

Pascal

```
function GetIndexForPage(const PageName: string): Integer;
```

C++

```
__fastcall int GetIndexForPage(const UnicodeString PageName);
```

Description

The PageIndex (see page 4421) property value is determined by the page's position in the Pages (see page 4422) property array. Specify the name of the page as the value of the PageName parameter. The name you specify must be one of the strings in the Pages (see page 4422) property.

See Also

PageIndex (see page 4421)

Pages (see page 4422)

38.1.1.4.10 TTabbedNotebook.KeyDown Method

Respond to key press events.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

When a windowed control receives a key-down message (WM_KEYDOWN) from Windows, its message handler calls the DoKeyDown method. If DoKeyDown determines that the control should, in fact, process the character, it calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

See Also

KeyUp

OnKeyDown

DoKeyDown

DoKeyPress

DoKeyUp

38.1.1.4.11 TTabbedNotebook.Loaded Method

Form load event dispatcher.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

The protected Loaded method is called automatically when the form that contains the tab control is loaded into memory. It provides an opportunity for the tab control to perform any initializations after the control and all other components in the form are loaded into memory, but before the first paint operation.

In TCustomTabControl, Loaded ensures that the layout of the tabs accommodate the images specified by the Images property.

See Also

Images

38.1.1.4.12 TTabbedNotebook.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

38.1.1.4.13 TTabbedNotebook.SetPageIndex Method

Pascal

```
procedure SetPageIndex(Value: Integer);
```

C++

```
__fastcall SetPageIndex(int Value);
```

Description

This is SetPageIndex, a member of class TTabbedNotebook.

38.1.1.4.14 TTabbedNotebook.ShowControl Method

Ensures that a specified child control is visible.

Pascal

```
procedure ShowControl(AControl: TControl); override;
```

C++

```
virtual __fastcall ShowControl(TControl * AControl);
```

Description

Call ShowControl to display a child control. Specify the control you want to ensure will be visible as the value of the AControl parameter.

As implemented in TWinControl, ShowControl simply calls the ShowControl method of the control's parent. Derived objects override ShowControl to perform whatever actions are necessary to allow the indicated child object to be visible. Possible actions include scrolling the particular child object into view, or changing to the appropriate notebook page.

See Also

Show

Parent

38.1.1.4.15 TTabbedNotebook.TabFontChanged Method

Invalidates the tabs when the TabFont (see page 4430) property changes.

Pascal

```
procedure TabFontChanged(Sender: TObject);
```

C++

```
__fastcall TabFontChanged(TObject * Sender);
```

Description

TabFontChanged is called automatically when the TabFont (see page 4430) property changes. It invalidates the notebook so that the tabs are repainted in the new font.

See Also

TabFont (see page 4430)

Invalidate

38.1.1.5 TTabbedNotebook Properties

The properties of the TTabbedNotebook class are listed here.

Protected Properties

	Name	Description
	Tabs (see page 4431)	Contains the list of text strings that label the tabs of the tab control.

Public Properties

	Name	Description
R	TopFont (see page 4433)	Returns the current value of font used on the tabs of the tabbed notebook.

Published Properties

	Name	Description
	ActivePage (see page 4413)	Determines which page displays in the tabbed notebook control.
	Align (see page 4414)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4417)	Specifies how the control is anchored to its parent.
	Constraints (see page 4418)	Specifies the size constraints for the control.
	DoubleBuffered (see page 4418)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	Enabled (see page 4420)	Controls whether the control responds to mouse, keyboard, and timer events.
	Font (see page 4420)	Controls the attributes of text written on or in the control.
	PageIndex (see page 4421)	Determines which page displays in the tabbed notebook component.
	Pages (see page 4422)	Contains the strings that identify the individual pages of the tabbed notebook control.
	ParentDoubleBuffered (see page 4422)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4423)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4423)	Identifies the pop-up menu associated with the control.
	ShowHint (see page 4428)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	TabFont (see page 4430)	Determines the font used on the tabs of the tabbed notebook control.
	TabOrder (see page 4430)	Indicates the position of the control in its parent's tab order.
	TabsPerRow (see page 4432)	TabsPerRow is obsolete and retained only for backwards compatibility.
	TabStop (see page 4433)	Determines if the user can tab to a control.
	Visible (see page 4433)	Determines whether the component appears onscreen.

38.1.1.5.1 TTabbedNotebook.ActivePage Property

Determines which page displays in the tabbed notebook control.

Pascal

```
property ActivePage: string;
```

C++

```
__property UnicodeString ActivePage;
```

Description

The value of ActivePage must be one of the strings contained in the Pages (see page 4422) property.

Note: To indicate the page that appears in the tabbed notebook by index rather than text, use the PageIndex (see page 4421) property instead.

See Also

PageIndex (see page 4421)

Pages (see page 4422)

38.1.1.5.2 TTabbedNotebook.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;
```

```
  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
```

```
  it is important to set its position to the appropriate side
  of the control it will resize.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    Split: TSplitter;
    Files: TFileListBox;
    Dirs: TDirectoryListBox;
  begin
    { first add a directory list box to the form }
    Dirs := TDirectoryListBox.Create(Form1);
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane - a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
```

```
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

38.1.1.5.3 TTabbedNotebook.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

38.1.1.5.4 TTabbedNotebook.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

38.1.1.5.5 TTabbedNotebook.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
```

```
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

38.1.1.5.6 TTabbedNotebook.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

38.1.1.5.7 TTabbedNotebook.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

38.1.1.5.8 TTabbedNotebook.PageIndex Property

Determines which page displays in the tabbed notebook component.

Pascal

```
property PageIndex: Integer;
```

C++

```
__property int PageIndex;
```

Description

Changing the PageIndex value changes the page in the control.

Each string in the Pages (see page 4422) property is automatically assigned a PageIndex value when the page is created. The first page receives a value of 0, the second has a value of 1, and so on. If you delete a string from the Pages (see page 4422) property, the PageIndex values are reassigned so that the values always begin with 0 and continue to increase without any gaps between values.

See Also

ActivePage (see page 4413)

Pages (see page 4422)

38.1.1.5.9 TTabbedNotebook.Pages Property

Contains the strings that identify the individual pages of the tabbed notebook control.

Pascal

```
property Pages: TStrings;
```

C++

```
__property TStrings Pages;
```

Description

Tabbed notebook controls create a separate page for each string in the Pages property. For example, if Pages contains three strings, First, Second, and Third, the control has three separate pages. Each string in Pages corresponds to a TTabPage (see page 4443) object whose Caption property is the same as the string.

You can change the active page in a tabbed notebook control with either the ActivePage (see page 4413) or PageIndex (see page 4421) property.

See Also

ActivePage (see page 4413)

PageIndex (see page 4421)

38.1.1.5.10 TTabbedNotebook.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of

the parent's DoubleBuffered property.

38.1.1.5.11 TTabbedNotebook.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

38.1.1.5.12 TTabbedNotebook.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
```

```
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
```

```
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
```

```
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
  else
    Beep;
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.PasteFromClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.PasteFromClipboard
  else
    Beep;
end;
{
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
```

```
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
```

```
var
  ItemToEdit: TMenuItem;
begin
  ItemToEdit := PopupMenu.Items[1];
  ItemToEdit.Caption := 'Changed Caption';
end;

procedure TForm1.DestroyButtonClick(Sender: TObject);
var
  ItemToDelete: TMenuItem;
begin
  ItemToDelete := PopupMenu.Items[2];
  ItemToDelete.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  index: Integer;
  NewItem: TMenuItem;
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

38.1.1.5.13 TTabbedNotebook.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

38.1.1.5.14 TTabbedNotebook.TabFont Property

Determines the font used on the tabs of the tabbed notebook control.

Pascal

```
property TabFont: TFont;
```

C++

```
__property TFont TabFont;
```

Description

Use TabFont to specify the font characteristics used to display the label on the tabs of the notebook.

Note: The text on the selected tab is boldfaced if the selected font for the TabFont property is not also boldfaced.

See Also

TopFont (see page 4433)

38.1.1.5.15 TTabbedNotebook.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

38.1.1.5.16 TTabbedNotebook.Tabs Property

Contains the list of text strings that label the tabs of the tab control.

Pascal

```
property Tabs;
```

C++

```
__property Tabs;
```

Description

Tabs contains a list of strings that label the tabs in the tab control. Tabs can also contain objects associated with the labels, using the TStrings Objects property. Add new strings to the Tabs property to add new tabs to the tab control. Rearrange the list to rearrange the tabs.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
    TabControl1Change(Sender);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
```

```
  several files. To run the example, place a tab control on a
  form and add a memo control that fits into its client area.
  Be sure to leave enough room for the tabs when they appear.
  Then add an OpenDialog and a button to the form.  The
  "Add a file" button adds a single file to the tabcontrol.
  The "Assign files" button removes previous files and can be
  used to assign multiple files.  To assign multiple files,
  use CNTL Select or SHIFT Select to select files in the
  OpenDialog.
  }
  procedure TForm1.Add_a_fileClick(Sender: TObject);
  var index : integer;
  begin
    OpenDialog1.Options :=
      [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
    if OpenDialog1.Execute then
    begin
      index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
      Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
      TabControl1Change(Sender);
    end;
  end;

  procedure TForm1.Assign_filesClick(Sender: TObject);
  var index : integer;
  begin
    OpenDialog1.Options :=
      [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
    if OpenDialog1.Execute then
    begin
      TabControl1.Tabs.Assign(OpenDialog1.Files);
      Memo1.Lines.LoadFromFile(
        TabControl1.Tabs[TabControl1.TabIndex]);
    end;
  end;

  {
  Place the following code in the tab control's OnChange event
  handler:
  }
  procedure TForm1.TabControl1Change(Sender: TObject);
  begin
    with TabControl1 do
      Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
  end;
```

See Also

TabIndex

Objects

TTabControl

38.1.1.5.17 TTabbedNotebook.TabsPerRow Property

TabsPerRow is obsolete and retained only for backwards compatibility.

Pascal

```
property TabsPerRow: Integer;
```

C++

```
__property int TabsPerRow;
```

Description

The TabsPerRow property determines the number of tabs that can appear in a row at the top of the tabbed notebook control. If there are more tabbed pages in the control than can fit in one row, multiple rows are displayed. The number of tabs you specify affects the size of the tabs--the more tabs per row--the smaller the tabs.

TabsPerRow can't be set to a value smaller than 3.

See Also

TabFont (see page 4430)

RowCount

38.1.1.5.18 TTabbedNotebook.TabStop Property

Determines if the user can tab to a control.

Pascal

```
property TabStop;
```

C++

```
__property TabStop;
```

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

See Also

TabOrder

38.1.1.5.19 TTabbedNotebook.TopFont Property

Returns the current value of font used on the tabs of the tabbed notebook.

Pascal

```
property TopFont: TFont;
```

C++

```
__property TFont TopFont;
```

Description

TopFont is a read-only property that reports the TabFont (see page 4430) value.

See Also

TabFont (see page 4430)

38.1.1.5.20 TTabbedNotebook.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

38.1.1.6 TTabbedNotebook Events

The events of the TTabbedNotebook class are listed here.

Published Events

	Name	Description
	OnChange (see page 4435)	Occurs when the page of a tabbed notebook changes.
	OnClick (see page 4436)	Occurs when a tab of a tabbed notebook is selected.
	OnContextPopup (see page 4436)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnEnter (see page 4437)	Occurs when a control receives the input focus.
	OnExit (see page 4439)	Occurs when the input focus shifts away from one control to another.

38.1.1.6.1 TTabbedNotebook.OnChange Event

Occurs when the page of a tabbed notebook changes.

Pascal

```
property OnChange: TPageChangeEvent;
```

C++

```
__property TPageChangeEvent OnChange;
```

Description

Write (see page 3707) an OnPage event handler to respond to a page change in a tabbed notebook.

See Also

OnClick (see page 4436)

38.1.1.6.2 TTabbedNotebook.OnClick Event

Occurs when a tab of a tabbed notebook is selected.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write (see page 3707) an OnClick event handler to respond when the user selects a tab in the tabbed notebook. Immediately after the OnClick event occurs, the OnChange (see page 4435) event occurs.

See Also

OnChange (see page 4435)

38.1.1.6.3 TTabbedNotebook.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

38.1.1.6.4 TTabbedNotebook.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
```

```
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
```

```
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

38.1.1.6.5 TTabbedNotebook.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
```

```
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';
```

```
    { use the Courier font in the maskedit boxes }
    MaskEdit1.Font.Name := 'Courier';
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
```

```
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

38.1.2 TTabPage Class

TTabPage implements an individual tab page of a tabbed notebook control.

Class Hierarchy

Pascal

```
TTabPage = class(TWinControl);
```

C++

```
class TTabPage : public TWinControl;
```

File

TabNotBk

Description

Each instance of this class holds the controls to be displayed when it is the active page of a tabbed notebook. TTabPage objects are associated with the strings in the corresponding tabbed notebook's Pages property by setting their Caption (see page 4445) property.

See Also

TTabbedNotebook (see page 4404)

38.1.2.1 TTabPage Members

The following tables list the members exposed by TTabPage.

Protected Methods

	Name	Description
	ReadState (see page 4445)	Prepares the control for having its properties assigned values from a stream.

Public Methods

	Name	Description
	Create (see page 4444)	Creates an instance of TWinControl.

Published Properties

	Name	Description
	Caption (see page 4445)	Specifies a text string that identifies the control to the user.
	Enabled (see page 4448)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 4449)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 4450)	Indicates the position of the control in its parent's tab order.
	Visible (see page 4450)	Determines whether the component appears onscreen.
	Width (see page 4452)	Specifies the horizontal size of the control or form in pixels.

38.1.2.2 TTabPage.Create Constructor

Creates an instance of TWinControl.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabPage(TComponent * AOwner);
```

Description

Call Create to construct and initialize a new control and insert the newly-constructed control into its owner, as specified by the AOwner parameter. Create inserts the control in the owner by calling the owner's InsertComponent method.

Most controls override Create to initialize their unique properties. Objects that override the Create method must always call the inherited Create method first, and then proceed with the component-specific initialization. Specify the override directive when overriding the Create method.

If a component's Create method allocates resources or memory, override the Destroy method to free those resources.

See Also

Brush

TabOrder

CreateParented

InsertComponent

Create

Owner

38.1.2.3 TTabPage Methods

The methods of the TTabPage class are listed here.

Protected Methods

	Name	Description
	ReadState (see page 4445)	Prepares the control for having its properties assigned values from a stream.

38.1.2.3.1 TTabPage.ReadState Method

Prepares the control for having its properties assigned values from a stream.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

Override ReadState to change the preparations the control makes as it readies itself to have its property values assigned from a stream. For example, a control might destroy temporary internal data structures or objects before new instances are loaded from the stream. Be sure to include a call to the inherited method when overriding ReadState.

As implemented in TWinControl, ReadState disables control alignment until all the child controls have been read from the stream, calls the inherited ReadState method, and re-enables the alignment of all the controls. The method then builds the tab order list of controls and synchronizes its visibility with that of its parent control.

See Also

Parent

ReadState

38.1.2.4 TTabPage Properties

The properties of the TTabPage class are listed here.

Published Properties

	Name	Description
	Caption (see page 4445)	Specifies a text string that identifies the control to the user.
	Enabled (see page 4448)	Controls whether the control responds to mouse, keyboard, and timer events.
	Height (see page 4449)	Specifies the vertical size of the control in pixels.
	TabOrder (see page 4450)	Indicates the position of the control in its parent's tab order.
	Visible (see page 4450)	Determines whether the component appears onscreen.
	Width (see page 4452)	Specifies the horizontal size of the control or form in pixels.

38.1.2.4.1 TTabPage.Caption Property

Specifies a text string that identifies the control to the user.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Note: The Caption property inheritted from TControl is a protected property. Access to the Caption requires casting the class to an inheritted one where Caption is a published or public property.

C++ Examples:

```
/*
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->AutoSize = false;
  Label1->Caption = "This string is too long to be the caption of this label";
}
/*
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would make a TTabSheet variable unnecessary since
PageControl1.->ActivePage is already of type TTabSheet.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TTabSheet *sheet = PageControl1->ActivePage;
  sheet->Caption = Edit1->Text;
}
/*
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
*/
#include <comctrls.hpp>
#include <memory>

void AddButtons(TToolBar *toolBar, TStringList *captions)
{
  for (int i = 0; i < captions->Count; i++)
  {
    TToolButton *button = new TToolButton(toolBar);  // The owner (toolBar) will clean this up.
    button->Parent = toolBar;
    button->Caption = captions->Strings[i];
    if (button->Caption == "|")
```

```
      button->Style = tbsSeparator;
    else
      button->Style = tbsButton;
  }
}

TToolBar *myToolBar;

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  myToolBar = new TToolBar(this); // The owner (this) will clean this up.
  myToolBar->Parent = this;
//  TStringList *captions = new TStringList();
  std::auto_ptr<TStringList> captions(new TStringList());

  captions->Add("New");
  captions->Add("Save");
  captions->Add("|");
  captions->Add("Cut");
  captions->Add("Copy");
  captions->Add("Paste");
  AddButtons(myToolBar, captions.get());
//  delete captions;
  myToolBar->ShowCaptions = True;
  myToolBar->Height = 40;
}
```

Delphi Examples:

```
{
The following code keeps the size of the label control
constant, even though the length of the label's caption
changes. As a result, the caption of the label is probably
too long to display in the label when the user clicks the
button:
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.AutoSize := False;
  Label1.Caption :=
    'This string is too long to be the caption of this label';
end;
{
This example requires a TPageControl populated by several
TabSheets, a TEdit and a TButton. When the button is
clicked, the caption of the active tab changes to that of
the TEdit.  These lines of code could also be written:
PageControl1.ActivePage.Caption := Edit1.Text;
This would obviate the need for a TTabSheet variable since
PageControl1.->ActivePage is already of type TTabSheet.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  sheet: TTabSheet;
begin
  sheet := PageControl1.ActivePage;
  sheet.Caption := Edit1.Text;
end;
{
This example creates a ToolBar and adds ToolButtons to it.
Include ComCtlrs in the uses clause.
}
procedure AddButtons(ToolBar: TToolBar;
  Const ButtonCaptions: array of String);
var
  I: Integer;
begin
```

```
    for I := 0 to High(ButtonCaptions) do
      begin
        with TToolButton.Create(ToolBar) do
        begin
          Parent := ToolBar;
          Caption := ButtonCaptions[I];
          if (ButtonCaptions[I] = '|') then
            Style := tbsSeparator
          else
            Style := tbsButton;
        end;
      end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ToolBar1 := TToolBar.Create(Self);
    ToolBar1.Parent := Self;
    AddButtons(ToolBar1, ['New', 'Save', '|', 'Cut', 'Copy', 'Paste']);
    ToolBar1.ShowCaptions := True;
    ToolBar1.Height := 40;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    I: Integer;
  begin
    for I := ToolBar1.ButtonCount - 1 downto 0 do
      ToolBar1.Buttons[I].Free;
    ToolBar1.Free;
  end;
```

38.1.2.4.2 TTabPage.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

38.1.2.4.3 TTabPage.Height Property

Specifies the vertical size of the control in pixels.

Pascal

```
property Height;
```

C++

```
__property Height;
```

Description

Use the Height property to read or change the height of the control.

C++ Examples:

```
/*
This example needs only a blank form.  Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
*/
#include <ComCtrls.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int I = 0; I <= 2; I++)
  {
    TRichEdit *pRich = new TRichEdit(this);
    pRich->Parent = this;
    pRich->Top = pRich->Height * I;
    pRich->Text = "This text will take the focus.";
    pRich->SelStart = 0;
    pRich->SelLength = pRich->Text.Length();
    if (I == 1)
      pRich->HideSelection = false;
  }
}
```

Delphi Examples:

```
{
This example needs only a blank form. Three Rich Edit
controls are placed vertically on the form, each contain a
line of text, and the second form has HideSelection set to
false.  Try tabbing between the three Rich Edit controls and
observe the effect on the selected text.
}
uses ComCtrls;

procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to 2 do
    with TRichEdit.Create(Self) do
    begin
      Parent := Self;
      Top := Height * I;
      Text := 'This text will take the focus.';
      SelStart := 0;
      SelLength := Length(Text);
      if (I = 1) then
        HideSelection := False;
```

```
    end;
end;
```

See Also

ClientHeight

Width

Using TDBListBox and TDBComboBox

38.1.2.4.4 TTabPage.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

38.1.2.4.5 TTabPage.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false,

the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
```

```
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

38.1.2.4.6 TTabPage.Width Property

Specifies the horizontal size of the control or form in pixels.

Pascal

```
property Width;
```

C++

```
__property Width;
```

Description

Use the Width property to read or change the width of the control.

Note: For tab sheet controls, changing this property at runtime has no effect.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
```

```
    ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
    ps->Align = pdlb->Align; // Give same alignment as directory list.

    // Each pane must be at least one quarter of the form's width.
    ps->MinSize = Form1->ClientWidth/4;

    // Finally, create the last pane - a file list box.
    pflb->Parent = Form1;
    pflb->Align = alClient;
    pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane – a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
```

See Also

ClientWidth

Height

Left

38.2 Types

The following table lists types in this documentation.

Types

Name	Description
TPageChangeEvent (see page 4454)	TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 4404) control.

38.2.1 TabNotBk.TPageChangeEvent Type

TPageChangeEvent is the type for event handlers that respond when the current page changes in a TTabbedNotebook (see page 4404) control.

Pascal

```
TPageChangeEvent = procedure (Sender: TObject; NewTab: Integer; var AllowChange: Boolean) of object;
```

C++

```
(Sender: TObject; NewTab: Integer; var AllowChange: Boolean) ( TPageChangeEvent)();
```

File

TabNotBk

Description

Sender is the TTabbedNotebook (see page 4404) control.

NewTab is the tab that is about to become the selected or current tab.

AllowChange specifies whether the change to a new page is allowed. If the event handler sets AllowChange to false, a new page change doesn't occur.

See Also

Procedural types

Creating events

38.3 Constants

The following table lists constants in this documentation.

Constants

Name	Description
CM_TABFONTCHANGED (see page 4455)	This is constant TabNotBk.CM_TABFONTCHANGED.

38.3.1 TabNotBk.CM_TABFONTCHANGED Constant

Pascal

```
CM_TABFONTCHANGED = CM_BASE + 100;
```

C++

```
CM_TABFONTCHANGED = CM_BASE + 100;
```

File

TabNotBk

Description

This is constant TabNotBk.CM_TABFONTCHANGED.

39 Tabs Namespace

This is namespace Tabs.

Topics

Name	Description
Classes (see page 4458)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4538)	The following table lists structs, records, enums in this documentation.
Types (see page 4541)	The following table lists types in this documentation.

Classes

	Name	Description
	TScroller (see page 4458)	TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.
	TTabList (see page 4463)	This is class Tabs.TTabList.
	TTabSet (see page 4470)	The TTabSet component presents horizontal tabs users can click to initiate actions.

Structs, Records, Enums

	Name	Description
	TEdgePart (see page 4538)	This is record Tabs.TEdgePart.
	TEdgeType (see page 4538)	This is record Tabs.TEdgeType.
	TScrollBtn (see page 4539)	This is record Tabs.TScrollBtn.
	TScrollOrientation (see page 4539)	This is record Tabs.TScrollOrientation.
	TTabSetTabStyle (see page 4540)	This is record Tabs.TTabSetTabStyle.

Types

Name	Description
TDrawTabEvent (see page 4541)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4470) control.
TMeasureTabEvent (see page 4541)	TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 4470) control.
TTabChangeEvent (see page 4542)	TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 4470) control changes.
TTabSetDrawTabEvent (see page 4542)	TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4470) control.
TTabStyle (see page 4543)	TTabStyle indicates whether a TTabSet (see page 4470) control is owner-drawn.

39.1 Classes

The following table lists classes in this documentation.

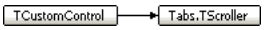
Classes

	Name	Description
	TScroller (see page 4458)	TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.
	TTabList (see page 4463)	This is class Tabs.TTabList.
	TTabSet (see page 4470)	The TTabSet component presents horizontal tabs users can click to initiate actions.

39.1.1 TScroller Class

TScroller is a horizontal scrolling control that provides basic horizontal scrolling capabilities.

Class Hierarchy



Pascal

```
TScroller = class(TCustomControl);
```

C++

```
class TScroller : public TCustomControl;
```

File

Tabs

Description

The TTabSet (see page 4470) object uses TScroller to scroll its visible tabs. TScroller is provided for backward compatibility only. TScroller is similar to an up-down control (TUpDown) with a horizontal orientation, except that it does not work with a companion control.

See Also

TTabSet (see page 4470)

TUpDown

39.1.1.1 TScroller Members

The following tables list the members exposed by TScroller.

Protected Methods

	Name	Description
	Paint (see page 4460)	Paints the image of the TScroller (see page 4458) object into its canvas.

Public Methods

	Name	Description
	Create (see page 4459)	Creates an instance of TScroller (see page 4458).
	Destroy (see page 4460)	Frees the memory associated with the TScroller (see page 4458) object.

Published Events

	Name	Description
	OnClick (see page 4463)	OnClick occurs after the Position (see page 4462) property has been updated in response to the user clicking one of the arrow buttons.

Published Properties

	Name	Description
	Change (see page 4461)	Specifies the number of units to scroll when the user clicks the arrow buttons.
	Max (see page 4461)	Specifies the maximum value the Position (see page 4462) property can take.
	Min (see page 4462)	Specifies the minimum value the Position (see page 4462) property can take.
	Position (see page 4462)	Position is the current position in the scroll range.
	ScrollOrientation (see page 4462)	This is ScrollOrientation, a member of class TScroller.

39.1.1.2 TScroller.Create Constructor

Creates an instance of TScroller (see page 4458).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TScroller(TComponent * AOwner);
```

Description

Call TScroller (see page 4458) to create an instance of TScroller (see page 4458) at runtime. TScroller (see page 4458) allocates the memory for the TScroller (see page 4458) object and initializes

Width to 24 and Height to 13

Min (see page 4462) and Max (see page 4461) to 0

Position (see page 4462) to 0

Change (see page 4461) to 1

The AOwner parameter specifies the component (usually the form) that is responsible for freeing the scroller.

See Also

Change (see page 4461)

Height

Max (see page 4461)

Min (see page 4462)

Position (see page 4462)

Width

39.1.1.3 TScroller.Destroy Destructor

Frees the memory associated with the TScroller (see page 4458) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScroller();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks whether the TScroller (see page 4458) reference is nil before calling Destroy.

See Also

Create (see page 4459)

System::TObject::Free (see page 3372)

39.1.1.4 TScroller Methods

The methods of the TScroller class are listed here.

Protected Methods

	Name	Description
	Paint (see page 4460)	Paints the image of the TScroller (see page 4458) object into its canvas.

39.1.1.4.1 TScroller.Paint Method

Paints the image of the TScroller (see page 4458) object into its canvas.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

Paint is called automatically in response to WM_PAINT messages. Paint paints the arrow buttons, graying them out when Position (see page 4462) cannot be moved in the associated direction.

See Also

Position (see page 4462)

Canvas

39.1.1.5 TScroller Properties

The properties of the TScroller class are listed here.

Published Properties

	Name	Description
	Change (see page 4461)	Specifies the number of units to scroll when the user clicks the arrow buttons.
	Max (see page 4461)	Specifies the maximum value the Position (see page 4462) property can take.
	Min (see page 4462)	Specifies the minimum value the Position (see page 4462) property can take.
	Position (see page 4462)	Position is the current position in the scroll range.
	ScrollOrientation (see page 4462)	This is ScrollOrientation, a member of class TScroller.

39.1.1.5.1 TScroller.Change Property

Specifies the number of units to scroll when the user clicks the arrow buttons.

Pascal

```
property Change: Integer;
```

C++

```
__property int Change;
```

Description

Set Change to indicate the number of units between Min (see page 4462) and Max (see page 4461) the Position (see page 4462) moves whenever the user clicks on one of the arrow buttons. Change represents the mapping between the units used to specify Min (see page 4462), Max (see page 4461), and Position (see page 4462), and the logical units represented by the scroller.

For example, if the values of Min (see page 4462), Max (see page 4461), and Position (see page 4462) are given in screen pixels, Change would be the number of pixels to scroll whenever the user clicks the scroll arrow.

See Also

Max (see page 4461)

Min (see page 4462)

Position (see page 4462)

39.1.1.5.2 TScroller.Max Property

Specifies the maximum value the Position (see page 4462) property can take.

Pascal

```
property Max: Longint;
```

C++

```
__property Longint Max;
```

Description

Use Max to set the upper limit of the scrolling range. The Max and Min (see page 4462) properties define the range of values over which the scroller can scroll.

See Also

Change (see page 4461)

Min (see page 4462)

Position (see page 4462)

39.1.1.5.3 TScroller.Min Property

Specifies the minimum value the Position (see page 4462) property can take.

Pascal

```
property Min: Longint;
```

C++

```
__property Longint Min;
```

Description

Use Min to set the lower limit of the scrolling range. The Min and Max (see page 4461) properties define the range of values over which the scroller can scroll.

See Also

Change (see page 4461)

Max (see page 4461)

Position (see page 4462)

39.1.1.5.4 TScroller.Position Property

Position is the current position in the scroll range.

Pascal

```
property Position: Longint;
```

C++

```
__property Longint Position;
```

Description

Read (see page 3634) Position to determine the current position the scroller represents. Position is a value between Min (see page 4462) and Max (see page 4461). Use an OnClick (see page 4463) event handler to notify the object being scrolled when the Position changes.

See Also

Max (see page 4461)

Min (see page 4462)

OnClick (see page 4463)

39.1.1.5.5 TScroller.ScrollOrientation Property

Pascal

```
property ScrollOrientation: TScrollOrientation;
```

C++

```
__property TScrollOrientation ScrollOrientation;
```

Description

This is ScrollOrientation, a member of class TScroller.

39.1.1.6 TScroller Events

The events of the TScroller class are listed here.

Published Events

	Name	Description
	OnClick (see page 4463)	OnClick occurs after the Position (see page 4462) property has been updated in response to the user clicking one of the arrow buttons.

39.1.1.6.1 TScroller.OnClick Event

OnClick occurs after the Position (see page 4462) property has been updated in response to the user clicking one of the arrow buttons.

Pascal

```
property OnClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnClick;
```

Description

Write (see page 3707) an OnClick event handler to implement the scrolling action of the object that uses the TScroller (see page 4458) to control scrolling. When the user clicks one of the arrow buttons, the Position (see page 4462) gets updated by Change (see page 4461) (unless the Max (see page 4461) or Min (see page 4462) value is hit), and then an OnClick event occurs.

See Also

Change (see page 4461)

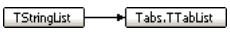
Max (see page 4461)

Min (see page 4462)

Position (see page 4462)

39.1.2 TTabList Class

Class Hierarchy



Pascal

```
TTabList = class(TStringList);
```

C++

```
class TTabList : public TStringList;
```

File

Tabs

Description

This is class Tabs.TTabList.

39.1.2.1 TTabList Members

The following tables list the members exposed by TTabList.

Protected Methods

	Name	Description
	Put (see page 4470)	Changes the value of the string with a specified index.

Public Methods

	Name	Description
	Add (see page 4464)	Adds a new string to the list.
	AddStrings (see page 4467)	Adds a group of strings to the list.
	Clear (see page 4468)	Deletes all the strings from the list.
	Create (see page 4464)	This is Create, a member of class TTabList.
	Delete (see page 4469)	Removes the string specified by the `Index` parameter.
	Insert (see page 4469)	Inserts a string to the list at the position specified by `Index`.

39.1.2.2 TTabList.Create Constructor

Pascal

```
constructor Create(const ATabSet: TTabSet);
```

C++

```
__fastcall TTabList(const TTabSet ATabSet);
```

Description

This is Create, a member of class TTabList.

39.1.2.3 TTabList Methods

The methods of the TTabList class are listed here.

Protected Methods

	Name	Description
	Put (see page 4470)	Changes the value of the string with a specified index.

Public Methods

	Name	Description
	Add (see page 4464)	Adds a new string to the list.
	AddStrings (see page 4467)	Adds a group of strings to the list.
	Clear (see page 4468)	Deletes all the strings from the list.
	Delete (see page 4469)	Removes the string specified by the `Index` parameter.
	Insert (see page 4469)	Inserts a string to the list at the position specified by `Index`.

39.1.2.3.1 TTabList.Add Method

Adds a new string to the list.

Pascal

```
function Add(const S: string): Integer; override;
```

C++

```
virtual __fastcall int Add(const UnicodeString S);
```

Description

Call Add to add the string S to the list. If the list is sorted, S is added to the appropriate position in the sort order. If the list is not sorted, S is added to the end of the list. Add returns the position of the item in the list, where the first item in the list has a value of 0.

Note: For sorted lists, Add will raise an EListError exception if the string S already appears in the list and Duplicates is set to dupError. If Duplicates is set to dupIgnore, trying to add a duplicate string causes Add to return the index of the existing entry.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  std::auto_ptr<TStringList> MyList(new TStringList());
  MyList->Add("Animals");
  MyList->Add("Flowers");
  MyList->Add("Ocean");
  MyList->Sort(); // Find will only work on sorted lists
  if (MyList->Find("Flowers", Index))
  {
    ListBox1->Items->AddStrings(MyList.get());
    Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
  }
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TStrings> StringList(new TStringList());
  StringList->Add("This example uses A string List. ");
  StringList->Add("It is the easiest way to add strings");
  StringList->Add("to a combobox's list of strings. ");
  StringList->Add("Always remember TStrings is abstract,");
  StringList->Add("So create a TStringList instead.");
  ComboBox1->Width = 210;
  ComboBox1->Items->Assign(StringList.get());
  ComboBox1->ItemIndex = 0;
}
```

Delphi Examples:

```
{
```

```
  This example uses a list box and a label on a form. When the
  application runs, a string list object is created and three
  strings are added to it. The Find method searches the
  strings to look for a match with the string 'Flowers'. If
  the string is found, all the strings in the string list are
  added to the list box, and the index value of the 'Flowers'
  string appears in the caption of the label control.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  var
    MyList: TStringList;
    Index: Integer;
  begin
    MyList := TStringList.Create;
    try
      MyList.Add('Animals');
      MyList.Add('Flowers');
      MyList.Add('Cars');
      MyList.Sort;   { Find will only work on sorted lists! }
      if MyList.Find('Flowers', Index) then
      begin
        ListBox1.Items.AddStrings(MyList);
        Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
      end;
    finally
      MyList.Free;
    end;
  end;
  {
  This example creates a stringlist, adds elements to it and
  assigns the strings to items of a combo box.
  }
  procedure TForm1.Button1Click(Sender: TObject);
  var
    StringList: TStringList;
  begin
    StringList := TStringList.Create;
    try
      with StringList do begin
        Add('This example uses A string List.');
        Add('It is the easiest way to add strings');
        Add('to a combobox''s list of strings.');
        Add('Always remember TStrings.Create method');
        Add('is abstract; So use TStringList.Create');
        Add('method instead.');
      end;
      with ComboBox1 do begin
        Width := 210;
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

AddStrings

Append

Delete

Duplicates

Insert

Sorted

39.1.2.3.2 TTabList.AddStrings Method

Adds a group of strings to the list.

Pascal

```
procedure AddStrings(Strings: TStrings); override;
```

C++

```
virtual __fastcall AddStrings(TStrings Strings);
```

Description

Call AddStrings to add the strings from another TStrings object to the list. If both the source and destination TStrings objects support objects associated with their strings, references to the associated objects will be added as well.

C++ Examples:

```
/*
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  int Index;
  std::auto_ptr<TStringList> MyList(new TStringList());
  MyList->Add("Animals");
  MyList->Add("Flowers");
  MyList->Add("Ocean");
  MyList->Sort(); // Find will only work on sorted lists
  if (MyList->Find("Flowers", Index))
  {
    ListBox1->Items->AddStrings(MyList.get());
    Label1->Caption = "Flowers has an index value of " + AnsiString(Index);
  }
}
/*
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  std::auto_ptr<TStringList> list(new TStringList());
  list->Add("Plants");
  list->Add("Animals");
  list->Add("Minerals");
  list->Sorted = true;
  ListBox1->Items->AddStrings(list.get());
}
```

Delphi Examples:

```
{
This example uses a list box and a label on a form. When the
application runs, a string list object is created and three
strings are added to it. The Find method searches the
strings to look for a match with the string 'Flowers'. If
the string is found, all the strings in the string list are
added to the list box, and the index value of the 'Flowers'
string appears in the caption of the label control.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
  Index: Integer;
begin
  MyList := TStringList.Create;
  try
    MyList.Add('Animals');
    MyList.Add('Flowers');
    MyList.Add('Cars');
    MyList.Sort;   { Find will only work on sorted lists! }
    if MyList.Find('Flowers', Index) then
    begin
      ListBox1.Items.AddStrings(MyList);
      Label1.Caption := 'Flowers has an index value of ' + IntToStr(Index);
    end;
  finally
    MyList.Free;
  end;
end;
{
This example uses a list box on a form. When the application
runs, a string list object is created and three strings are
added to it. The strings are sorted and added to the list
box, where they appear in their sorted order:
}
procedure TForm1.FormCreate(Sender: TObject);
var
  MyList: TStringList;
begin
  MyList := TStringList.Create;
  MyList.Add('Plants');
  MyList.Add('Animals');
  MyList.Add('Minerals');
  MyList.Sorted := True;
  ListBox1.Items.AddStrings(MyList);
  MyList.Free;
end;
```

See Also

Add

AddObject

Assign

Count

Objects

39.1.2.3.3 TTabList.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete

Objects

Strings

39.1.2.3.4 TTabList.Delete Method

Removes the string specified by the `Index` parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add

AddObject

Clear

IndexOf

InsertObject

Objects

Strings

39.1.2.3.5 TTabList.Insert Method

Inserts a string to the list at the position specified by `Index`.

Pascal

```
procedure Insert(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Insert(int Index, const UnicodeString S);
```

Description

Call Insert to add the string S to the list at the position specified by `Index`. If `Index` is 0, the string is inserted at the beginning of the list. If `Index` is 1, the string is put in the second position of the list, and so on.

If the string has an associated object, use the InsertObject method instead.

Note: If the list is sorted, calling Insert or InsertObject will raise an EListError exception. Use Add or AddObject with sorted lists.

See Also

Add

AddObject

EListError

InsertObject

Sorted

Strings

39.1.2.3.6 TTabList.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: string); override;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by `Index` to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

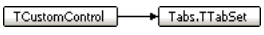
Strings

Objects

39.1.3 TTabSet Class

The TTabSet component presents horizontal tabs users can click to initiate actions.

Class Hierarchy



Pascal

```
TTabSet = class(TCustomControl);
```

C++

```
class TTabSet : public TCustomControl;
```

File

Tabs

Description

Tab set controls are commonly used to display tabbed pages within a dialog box. TTabSet is provided for backward compatibility. Use TTabControl component in 32-bit Windows applications.

You create a set of tabs for the tab set control when you specify a list of strings as the value of the Tabs (see page 4506) property. One tab is created for each string.

To determine which tab is currently selected or to use code to select a tab, use the TabIndex (see page 4506) property. To find out which tab is the first visible tab in the tab set control or to make a tab the first visible tab, use the FirstIndex (see page 4494) property.

39.1.3.1 TTabSet Members

The following tables list the members exposed by TTabSet.

Protected Methods

	Name	Description
	CanChange (see page 4474)	This is CanChange, a member of class TTabSet.
	CMHintShow (see page 4474)	This is CMHintShow, a member of class TTabSet.
V	CreateParams (see page 4475)	Initializes a window-creation parameter data structure.
V	DrawTab (see page 4475)	This is DrawTab, a member of class TTabSet.
V	GetImageIndex (see page 4476)	This is GetImageIndex, a member of class TTabSet.
V	MeasureTab (see page 4477)	This is MeasureTab, a member of class TTabSet.
V	MouseDown (see page 4478)	OnMouseDown event dispatcher.
V	Notification (see page 4479)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 4479)	Renders the image of a custom control.
V	SetTabIndex (see page 4480)	This is SetTabIndex, a member of class TTabSet.

Protected Properties

	Name	Description
R	Scroller (see page 4502)	This is Scroller, a member of class TTabSet.

Public Methods

	Name	Description
V	Create (see page 4473)	Constructs a tab set control.
V	Destroy (see page 4473)	Frees the memory associated with the TTabSet (see page 4470) object.
V	GetChildren (see page 4475)	Calls a specified method for each child of the control.
	ItemAtPos (see page 4476)	Returns the index of the tab indicated by the coordinates of a point on the control.
	ItemRect (see page 4476)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ItemWidth (see page 4477)	Returns the width, in pixels, of a specified tab in the set.
	MinClientRect (see page 4477)	This is the overview for the MinClientRect method overload.
	SelectNext (see page 4480)	Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

Public Properties

	Name	Description
	Canvas (see page 4486)	Specifies the TCanvas object that presents a drawing surface for the control.
	FirstIndex (see page 4494)	Specifies the tab that appears in the leftmost visible position in the tab set control.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 4509)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 4510)	Occurs when an object with custom alignment is aligned.
	OnChange (see page 4511)	Occurs just before a new tab is selected.
	OnClick (see page 4511)	Occurs when the user clicks the control.
	OnDragDrop (see page 4513)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4516)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4519)	Occurs when a tab in an owner-drawn tab set control needs to redisplay.
	OnEndDock (see page 4519)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4520)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4521)	Occurs when a control receives the input focus.
	OnExit (see page 4523)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4527)	This is OnGetImageIndex, a member of class TTabSet.
	OnMeasureTab (see page 4527)	Occurs when an application needs to redisplay a tab in an owner-draw tab set.
	OnMouseActivate (see page 4528)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4529)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4531)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4531)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4532)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4534)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4536)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4537)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Published Properties

	Name	Description
	Align (see page 4481)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4485)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 4485)	Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.
	BackgroundColor (see page 4486)	Determines the background color of the tab set control.
	Constraints (see page 4488)	Specifies the size constraints for the control.
	DitherBackground (see page 4489)	Determines whether the background color set with the BackgroundColor (see page 4486) property is dithered.
	DoubleBuffered (see page 4489)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4491)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4491)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4492)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4493)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndMargin (see page 4494)	Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.
	Font (see page 4494)	Controls the attributes of text written on or in the control.
	Images (see page 4496)	This is Images, a member of class TTabSet.
	ParentBackground (see page 4496)	Determines whether control uses parent's theme background.
	ParentDoubleBuffered (see page 4496)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4496)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4497)	Identifies the pop-up menu associated with the control.
	SelectedColor (see page 4502)	Determines the color of the selected tab in the tab set control.

	ShowHint (see page 4502)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShrinkToFit (see page 4504)	This is ShrinkToFit, a member of class TTabSet.
	SoftTop (see page 4504)	Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.
	StartMargin (see page 4504)	Determines how far, in pixels, the first tab appears from the left edge of the tab set control.
	Style (see page 4505)	Determines whether the tab set is owner-drawn.
	TabHeight (see page 4505)	Determines the height of the tabs in an owner-drawn tab set control.
	TabIndex (see page 4506)	Determines which tab of a tab set control is currently selected.
	TabPosition (see page 4506)	This is TabPosition, a member of class TTabSet.
	Tabs (see page 4506)	Contains the list of text strings that appear on the tabs.
	UnselectedColor (see page 4507)	Determines the color of the tabs that aren't currently selected in the tab set control.
	Visible (see page 4507)	Determines whether the component appears onscreen.
R	VisibleTabs (see page 4508)	Contains the number of tabs currently visible in the tab set control.

39.1.3.2 TTabSet.Create Constructor

Constructs a tab set control.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TTabSet(TComponent * AOwner);
```

Description

Create sets the initial values of the control, including creating a TTabList (see page 4463) object that holds the list of tabs for the control. The AOwner parameter specifies the component (usually a form) that is responsible for freeing the tab set.

See Also

Destroy (see page 4473)

39.1.3.3 TTabSet.Destroy Destructor

Frees the memory associated with the TTabSet (see page 4470) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTabSet();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the tab set is not nil before calling Destroy. Destroy frees the TTabList (see page 4463) object that holds the list of tabs.

See Also

Create (see page 4473)

System::TObject::Free (see page 3372)

39.1.3.4 TTabSet Methods

The methods of the TTabSet class are listed here.

Protected Methods

	Name	Description
	CanChange (see page 4474)	This is CanChange, a member of class TTabSet.
	CMHintShow (see page 4474)	This is CMHintShow, a member of class TTabSet.
V	CreateParams (see page 4475)	Initializes a window-creation parameter data structure.
V	DrawTab (see page 4475)	This is DrawTab, a member of class TTabSet.
V	GetImageIndex (see page 4476)	This is GetImageIndex, a member of class TTabSet.
V	MeasureTab (see page 4477)	This is MeasureTab, a member of class TTabSet.
V	MouseDown (see page 4478)	OnMouseDown event dispatcher.
V	Notification (see page 4479)	Responds to notifications that components are being created or destroyed.
V	Paint (see page 4479)	Renders the image of a custom control.
V	SetTabIndex (see page 4480)	This is SetTabIndex, a member of class TTabSet.

Public Methods

	Name	Description
V	GetChildren (see page 4475)	Calls a specified method for each child of the control.
	ItemAtPos (see page 4476)	Returns the index of the tab indicated by the coordinates of a point on the control.
	ItemRect (see page 4476)	Returns the rectangle that surrounds the item specified in the Item parameter.
	ItemWidth (see page 4477)	Returns the width, in pixels, of a specified tab in the set.
	MinClientRect (see page 4477)	This is the overview for the MinClientRect method overload.
	SelectNext (see page 4480)	Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

39.1.3.4.1 TTabSet.CanChange Method

Pascal

```
function CanChange(NewIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CanChange(int NewIndex);
```

Description

This is CanChange, a member of class TTabSet.

39.1.3.4.2 TTabSet.CMHintShow Method

Pascal

```
procedure CMHintShow(var Message: TCMHintShow); message CM_HINTSHOW;
```

C++

```
__fastcall CMHintShow(TCMHintShow Message);
```

Description

This is CMHintShow, a member of class TTabSet.

39.1.3.4.3 TTabSet.CreateParams Method

Initializes a window-creation parameter data structure.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

The CreateWnd method calls CreateParams to initialize the parameters it passes to CreateWindowHandle. Override CreateParams to customize the way a control creates its Windows representation. When overriding CreateParams, always call the inherited method first to set the default values, then make any desired adjustments.

CreateParams fills in the parameter that is passed as the value of the Params parameter. This data structure holds information needed when telling Windows to create a window handle. The fields of a TCreateParams record become the parameters to a call to the CreateWindowEx API function.

TWinControl implements CreateParams by setting all the fields of Params to generic base values. A number of the standard controls override CreateParams to change one or more of the default values in Params.

See Also

CreateWindowHandle

CreateWnd

AddBiDiModeExStyle

39.1.3.4.4 TTabSet.DrawTab Method

Pascal

```
procedure DrawTab(TabCanvas: TCanvas; R: TRect; Index: Integer; Selected: Boolean); virtual;
```

C++

```
virtual __fastcall DrawTab(TCanvas TabCanvas, TRect R, int Index, Boolean Selected);
```

Description

This is DrawTab, a member of class TTabSet.

39.1.3.4.5 TTabSet.GetChildren Method

Calls a specified method for each child of the control.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren is called by the streaming system that loads and saves components. Applications seldom need to call this routine.

GetChildren executes the callback specified by the Proc parameter for every child control listed by the Controls property that lists the Root parameter as its Owner.

Override GetChildren to limit or augment which child controls are saved with the control. When overriding GetChildren, call the procedure passed as the Proc parameter for every child control that should be saved. The Root parameter indicates the component (usually a form) that owns the control.

See Also

Controls

Owner

39.1.3.4.6 TTabSet.GetImageIndex Method

Pascal

```
function GetImageIndex(TabIndex: Integer): Integer; virtual;
```

C++

```
virtual __fastcall int GetImageIndex(int TabIndex);
```

Description

This is GetImageIndex, a member of class TTabSet.

39.1.3.4.7 TTabSet.ItemAtPos Method

Returns the index of the tab indicated by the coordinates of a point on the control.

Pascal

```
function ItemAtPos(Pos: TPoint; IgnoreTabHeight: Boolean = False): Integer;
```

C++

```
__fastcall int ItemAtPos(TPoint Pos, Boolean IgnoreTabHeight = False);
```

Description

Call ItemAtPos to identify the tab under the point specified by the Pos (see page 3627) parameter. Pos (see page 3627) is the point in the control in client coordinates. If the returned index is 0, the tab is the first tab in the tab set, if the index is 1, the tab is the second tab, and so on. ItemAtPos returns –1 if there is no tab at the specified point.

See Also

ItemRect (see page 4476)

39.1.3.4.8 TTabSet.ItemRect Method

Returns the rectangle that surrounds the item specified in the Item parameter.

Pascal

```
function ItemRect(Index: Integer): TRect;
```

C++

```
__fastcall TRect ItemRect(int Index);
```

Description

Use ItemRect to obtain the rectangle that surrounds the tab when you are drawing on an owner-draw tab set.

See Also

ItemAtPos (see page 4476)

Style (see page 4505)

OnDrawTab (see page 4519)

OnMeasureTab (see page 4527)

39.1.3.4.9 TTabSet.ItemWidth Method

Returns the width, in pixels, of a specified tab in the set.

Pascal

```
function ItemWidth(Index: Integer): Integer;
```

C++

```
__fastcall int ItemWidth(int Index);
```

Description

Use ItemWidth to obtain the width, in pixels of a specified tab.

Index is the index of the tab, where 0 indicates the first tab, 1 indicates the second tab, and so on.

See Also

ItemAtPos (see page 4476)

ItemRect (see page 4476)

39.1.3.4.10 TTabSet.MeasureTab Method

Pascal

```
procedure MeasureTab(Index: Integer; var TabWidth: Integer); virtual;
```

C++

```
virtual __fastcall MeasureTab(int Index, int TabWidth);
```

Description

This is MeasureTab, a member of class TTabSet.

39.1.3.4.11 MinClientRect Method

This is the overview for the MinClientRect method overload.

Overload List

	Name	Description
	TTabSet.MinClientRect () (see page 4477)	Returns the boundaries that are required to display the tabs in the tab set.
	TTabSet.MinClientRect (Boolean) (see page 4477)	Returns the boundaries that are required to display the tabs in the tab set.
	TTabSet.MinClientRect (Integer, Boolean) (see page 4477)	Returns the boundaries that are required to display the tabs in the tab set.

39.1.3.4.11.1 TTabSet.MinClientRect Method ()

Returns the boundaries that are required to display the tabs in the tab set.

Pascal

```
function MinClientRect: TRect; overload;
function MinClientRect(IncludeScroller: Boolean): TRect; overload;
function MinClientRect(TabCount: Integer; IncludeScroller: Boolean = False): TRect; overload;
```

C++

```
__fastcall TRect MinClientRect();
__fastcall TRect MinClientRect(Boolean IncludeScroller);
__fastcall TRect MinClientRect(int TabCount, Boolean IncludeScroller = False);
```

Description

Use MinClientRect to obtain the minimum size that the tab set's client area must be to display its tabs and (if specified) a scroller for scrolling hidden tabs. MinClientRect calculates the width based on the widths of all the tabs up to a specified count, adding in the values of StartMargin (see page 4504) and EndMargin (see page 4494) and the space that appears between tabs. If requested, it also adds the width of a scroller for the tabs. It calculates the height based on the value of the TabHeight (see page 4505) property.

TabCount specifies the number of tabs to display. MinClientRect uses the widths of these tabs to calculate the width of the client rectangle. If you use a syntax that does not include a TabCount parameter, MinClientRect assumes that all tabs must be visible.

IncludeScroller indicates whether the client rectangle must include space for the scroller control that allows users to scroll the tabs, displaying any tabs that are not included in TabCount. If IncludeScroller is false, or if this parameter is omitted, MinClientRect does not include the width for the scroller in the resulting rectange.

MinClientRect returns the dimensions, in pixels, needed to display the specified number of tabs and, if specified, the scroller. The returned rectangle has an origin of 0,0.

See Also

TabHeight (see page 4505)

ItemWidth (see page 4477)

StartMargin (see page 4504)

EndMargin (see page 4494)

39.1.3.4.12 TTabSet.MouseDown Method

OnMouseDown event dispatcher.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown event handler when the user presses the mouse button down while the cursor's hotspot is over the control.

The Button parameter determines which mouse button the user pressed. Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button. X and Y are the pixel coordinates of the mouse pointer within the client area of the control.

A control calls MouseDown in response to any of the Windows mouse-down messages (WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN), decoding the message parameters into the shift-key state and position, which

it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was pressed down: left, right, or middle.

See Also

MouseMove

MouseUp

OnMouseDown

39.1.3.4.13 TTabSet.Notification Method

Responds to notifications that components are being created or destroyed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification allows TControl to update its PopupMenu property when the pop-up menu it refers to is destroyed.

Override the Notification method to respond to notifications that other components are about to be destroyed or have just been created. Use the Notification method to update controls that rely on other objects. Remove references to objects that are going away so that the control does not refer to objects that have been destroyed. For example, data-aware objects override the Notification method to update themselves when their data source is removed. Some objects respond to notifications that other objects of a specific type are being created. For example, the session component sets the Session property of new data-aware controls to itself when the AutoSessionName property is true.

By default, components pass along the notification to their owned components, if any.

See Also

PopupMenu

Notification

39.1.3.4.14 TTabSet.Paint Method

Renders the image of a custom control.

Pascal

```
procedure Paint; override;
```

C++

```
virtual __fastcall Paint();
```

Description

The Paint method for TCustomControl does nothing other than provide the interface for a method that responds to WM_PAINT messages. Paint is called by the PaintWindow method, after that method has supplied the Canvas with the handle to a device context. When creating a custom control, always override Paint to draw the image of the control.

Tip: To determine which portions of the control's canvas need to be repainted when Paint is called, use the ClipRect property of the canvas.

See Also

Canvas

PaintWindow

GetDeviceContext

ClipRect

39.1.3.4.15 TTabSet.SelectNext Method

Selects the next tab in a tab set control, scrolling if necessary to bring the selected tab into view.

Pascal

```
procedure SelectNext(Direction: Boolean);
```

C++

```
__fastcall SelectNext(Boolean Direction);
```

Description

The value of the Direction parameters determines if the tab to the left or right is selected. If Direction is true, the tab to the right is selected. If Direction is false, the tab to the left is selected. When the last tab in either direction is selected, calling SelectNext using the same direction wraps around to the beginning of the tab order. For example, if your application has three tabs, First, Second, and Third, and Third is the current tab, calling SelectNext(true) selects First. Likewise, if First is the current tab, SelectNext(false) selects Third.

When SelectNext is called, the OnClick (see page 4511) event of the tab set occurs, followed by the OnChange (see page 4511) event, just as if the user had clicked on a new tab.

See Also

Tabs (see page 4506)

OnChange (see page 4511)

39.1.3.4.16 TTabSet.SetTabIndex Method

Pascal

```
procedure SetTabIndex(Value: Integer); virtual;
```

C++

```
virtual __fastcall SetTabIndex(int Value);
```

Description

This is SetTabIndex, a member of class TTabSet.

39.1.3.5 TTabSet Properties

The properties of the TTabSet class are listed here.

Protected Properties

	Name	Description
	Scroller (see page 4502)	This is Scroller, a member of class TTabSet.

Public Properties

	Name	Description
	Canvas (see page 4486)	Specifies the TCanvas object that presents a drawing surface for the control.
	FirstIndex (see page 4494)	Specifies the tab that appears in the leftmost visible position in the tab set control.

Published Properties

	Name	Description
	Align (see page 4481)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4485)	Specifies how the control is anchored to its parent.
	AutoScroll (see page 4485)	Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.
	BackgroundColor (see page 4486)	Determines the background color of the tab set control.
	Constraints (see page 4488)	Specifies the size constraints for the control.
	DitherBackground (see page 4489)	Determines whether the background color set with the BackgroundColor (see page 4486) property is dithered.
	DoubleBuffered (see page 4489)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4491)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4491)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4492)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	Enabled (see page 4493)	Controls whether the control responds to mouse, keyboard, and timer events.
	EndMargin (see page 4494)	Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.
	Font (see page 4494)	Controls the attributes of text written on or in the control.
	Images (see page 4496)	This is Images, a member of class TTabSet.
	ParentBackground (see page 4496)	Determines whether control uses parent's theme background.
	ParentDoubleBuffered (see page 4496)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentShowHint (see page 4496)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4497)	Identifies the pop-up menu associated with the control.
	SelectedColor (see page 4502)	Determines the color of the selected tab in the tab set control.
	ShowHint (see page 4502)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	ShrinkToFit (see page 4504)	This is ShrinkToFit, a member of class TTabSet.
	SoftTop (see page 4504)	Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.
	StartMargin (see page 4504)	Determines how far, in pixels, the first tab appears from the left edge of the tab set control.
	Style (see page 4505)	Determines whether the tab set is owner-drawn.
	TabHeight (see page 4505)	Determines the height of the tabs in an owner-drawn tab set control.
	TabIndex (see page 4506)	Determines which tab of a tab set control is currently selected.
	TabPosition (see page 4506)	This is TabPosition, a member of class TTabSet.
	Tabs (see page 4506)	Contains the list of text strings that appear on the tabs.
	UnselectedColor (see page 4507)	Determines the color of the tabs that aren't currently selected in the tab set control.
	Visible (see page 4507)	Determines whether the component appears onscreen.
R	VisibleTabs (see page 4508)	Contains the number of tabs currently visible in the tab set control.

39.1.3.5.1 TTabSet.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form’s implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);

  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
```

```
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
```

```
    { line it up on the left of the form }
    Dirs.Parent := Form1;
    Dirs.Align := alLeft;
    Dirs.Width := Form1.ClientWidth div 3;
  { now add the splitter to divide the directory pane from the file pane }
    Split := TSplitter.Create(Form1);
    Split.Parent := Form1;
    { make sure the splitter is to the right of the directory list box! }
    Split.Left := Dirs.Left + Dirs.Width + 1;
    Split.Align := Dirs.Align; { give it the same alignment as the directory }
    { each pane must be at least one quarter of the form's width }
    Split.MinSize := Form1.ClientWidth div 4;
    { Finally, create the last pane – a file list box }
    Files := TFileListBox.Create(Form1);
    Files.Parent := Form1;
    Files.Align := alClient;
    Dirs.FileList := Files; { link the file list box to the directory list box }
  end;
  {
  Dynamically create a TProgressBar control and align it to
  the bottom of the form.
  }

  var
    I: Integer;

  procedure TForm1.Button1Click(Sender: TObject);
  var
    ProgressBar: TProgressBar;
  begin
    ProgressBar := TProgressBar.Create(Self);
    with ProgressBar do
    begin
      Parent := Self;
      Align := alBottom;
    end;
  end;
  {
  This example dynamically creates a Page Control, then a
  series of Tab Sheets on the Page Control.
  }
  var
    PageControl1: TPageControl;
    pts: array[0..3] of TTabSheet;
  const
    TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

  procedure TForm1.FormCreate(Sender: TObject);
  const
    colorpalette: Array[0..11] of TColor = (
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
```

```
        Brush.Color := colorPalette[i];
    end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  for i := Low(TabTitles) to High(TabTitles) do
  begin
    pts[i].Free;
  end;
  PageControl1.Free;
end;
```

See Also

Parent

Anchors

39.1.3.5.2 TTabSet.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

39.1.3.5.3 TTabSet.AutoScroll Property

Determines if scroll buttons automatically appear in a tab set control if there isn't room in the control to display all the tabs.

Pascal

```
property AutoScroll: Boolean;
```

C++

```
__property Boolean AutoScroll;
```

Description

If AutoScroll is false, the application can still access tabs that aren't visible by using the FirstIndex (see page 4494) or TabIndex (see page 4506) properties at design time or runtime, but the user can't click on the tabs with the mouse at runtime.

See Also

FirstIndex (see page 4494)

TabIndex (see page 4506)

39.1.3.5.4 TTabSet.BackgroundColor Property

Determines the background color of the tab set control.

Pascal

```
property BackgroundColor: TColor;
```

C++

```
__property TColor BackgroundColor;
```

Description

Use BackgroundColor to indicate the color that the tab set control uses to paint the area between the tabs and the border of the control.

See Also

DitherBackground (see page 4489)

SelectedColor (see page 4502)

Color

39.1.3.5.5 TTabSet.Canvas Property

Specifies the TCanvas object that presents a drawing surface for the control.

Pascal

```
property Canvas;
```

C++

```
__property Canvas;
```

Description

Use the properties of the TCanvas object to draw or paint on the surface of the control. Canvas encapsulates a Windows device context, providing all the tools and methods needed for drawing and painting.

Canvas is a protected property that is usually redeclared as public in descendants of TCustomControl.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
```

```
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  */
  void __fastcall TForm1::DrawGrid1DrawCell(
    TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
  {
    long index = ARow * DrawGrid1->ColCount + ACol;
    DrawGrid1->Canvas->Brush->Color = clBackground;
    DrawGrid1->Canvas->FillRect(Rect);
    if (State.Contains(gdFocused))
    {
      DrawGrid1->Canvas->DrawFocusRect(Rect);
      Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
    }
    ImageList1->Draw(
      DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  }

  void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
        bool &CanSelect)
  {
    CanSelect= True;
  }
  /*
  The following code uses ClientRect to find and draw a line
  from the top left to the bottom right of the current
  control.  Select a TControl in the ListBox list.  This
  example assumes that the current object is a descendent of
  TControl and has a public Canvas property.
  */
  {
    TPoint APoint = Point(X, Y);
    int Index = ListBox1->ItemAtPos(APoint, True);
    TStringGrid *myStringGrid;
    TClass ClassRef = ListBox1->Items->Objects[Index]->ClassType();
    if (String(ClassRef->ClassName()) == "TStringGrid")
    {
      myStringGrid = dynamic_cast<TStringGrid *>(ListBox1->Items->Objects[Index]);
      // make sure we are using the TControls canvas and not the from's!
      myStringGrid->Canvas->MoveTo(myStringGrid->ClientRect.Left, myStringGrid->ClientRect.Top);
      myStringGrid->Canvas->LineTo(myStringGrid->ClientRect.Right,
  myStringGrid->ClientRect.Bottom);
      myStringGrid = NULL;
    }
  }
```

Delphi Examples:

```
  {
  The following code uses the bitmaps in an image list
  component to draw the contents of each cell in a draw grid.
  It draws a focus rectangle around the cell that has focus.
  goDrawFocusSelect in the DrawGrid Options parameter must be
  True to set focus on a cell.  The ImageList Draw method must
  be called after DrawFocusRect. The OnSelectCell event handler
  must implemented to return true.
  }
  procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
    Rect: TRect; State: TGridDrawState);
  var
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clWhite;
    DrawGrid1.Canvas.FillRect(Rect);
    if (gdFocused in State) then
    begin
```

```
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
{
The following code uses ClientRect to find and draw a line
from the top left to the bottom right of the current
control.  Select a TControl in the ListBox list.  This
example assumes that the current object is a descendent of
TControl and has a public Canvas property.
}
  if (ListBox1.Items.Objects[Index] is TStringGrid) then
  begin
    myStringGrid:= (ListBox1.Items.Objects[Index] as TStringGrid);
    with myStringGrid.ClientRect do
    begin // make sure we are using the TControl's canvas and not the form's!
      myStringGrid.Canvas.MoveTo(Left, Top);
      myStringGrid.Canvas.LineTo(Right, Bottom);
    end;
  myStringGrid:= nil;
  end;
```

See Also

Paint

39.1.3.5.6 TTabSet.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

39.1.3.5.7 TTabSet.DitherBackground Property

Determines whether the background color set with the BackgroundColor (see page 4486) property is dithered.

Pascal

```
property DitherBackground: Boolean;
```

C++

```
__property Boolean DitherBackground;
```

Description

Dithering means the background is lightened by 50%, which is intended to make the tabs easier to see. If DitherBackground is true, the tab set control background is dithered. If it is false, there is no dithering.

See Also

BackgroundColor (see page 4486)

SelectedColor (see page 4502)

39.1.3.5.8 TTabSet.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
```

```
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
```

```
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

39.1.3.5.9 TTabSet.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

39.1.3.5.10 TTabSet.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

39.1.3.5.11 TTabSet.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
```

```
    HT: THitTests;
  begin
    if TreeView1.Selected = nil then Exit;
    HT := TreeView1.GetHitTestInfoAt(X, Y);
    AnItem := TreeView1.GetNodeAt(X, Y);
    if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
    begin
      if (htOnItem in HT) or (htOnIcon in HT) then
        AttachMode := naAddChild
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

39.1.3.5.12 TTabSet.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

39.1.3.5.13 TTabSet.EndMargin Property

Determines how far, in pixels, the rightmost tab appears from the right edge of the tab set control.

Pascal

```
property EndMargin: Integer;
```

C++

```
__property int EndMargin;
```

Description

EndMargin determines how far in pixels the rightmost tab appears from the right edge of the tab set control. Together with the StartMargin (see page 4504) property, EndMargin can play a role in determining how many tabs can fit within the tab set control.

If AutoScroll (see page 4485) is true and scroll buttons appear in the tab set control, EndMargin determines how far in pixels the rightmost tab appears from the left edge of the scroll buttons, rather than the edge of the tab set control.

See Also

AutoScroll (see page 4485)

StartMargin (see page 4504)

39.1.3.5.14 TTabSet.FirstIndex Property

Specifies the tab that appears in the leftmost visible position in the tab set control.

Pascal

```
property FirstIndex: Integer;
```

C++

```
__property int FirstIndex;
```

Description

Any tabs with a lower value in the FirstIndex property scroll to the left in the tab set control and don't appear until the user scrolls the tabs.

The default value of FirstIndex is 0 indicating that the tab with an index of 0 is in the leftmost position. For example, if you have three tabs labeled First, Second, and Third with TabIndex (see page 4506) values of 0, 1, and 2, respectively, First appears first, by default, because it has an index value of 0. If you want to shift the tabs so the Second or Third tab appears leftmost in the tab set control, change the FirstIndex value to 1 or 2.

See Also

Tabs (see page 4506)

39.1.3.5.15 TTabSet.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

39.1.3.5.16 TTabSet.Images Property

Pascal

```
property Images: TCustomImageList;
```

C++

```
__property TCustomImageList Images;
```

Description

This is Images, a member of class TTabSet.

39.1.3.5.17 TTabSet.ParentBackground Property

Determines whether control uses parent's theme background.

Pascal

```
property ParentBackground;
```

C++

```
__property ParentBackground;
```

Description

If ParentBackground is true, the control uses the parent's theme background to draw its own background. ParentBackground has no effect unless XP themes are enabled.

39.1.3.5.18 TTabSet.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

39.1.3.5.19 TTabSet.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

39.1.3.5.20 TTabSet.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
```

```
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Cut1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }

  void __fastcall TForm1::Paste1Click(TObject *Sender)
  {
    TComponent *pComponent = PopupMenu1->PopupComponent;
    if (pComponent)
    {
      if (pComponent->ClassNameIs("TEdit"))
        (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
      else
        MessageBeep(0);
    }
    else
      MessageBeep(0);
  }
  /*
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  */
  void __fastcall TForm1::AddButtonClick(TObject *Sender)
  {
      // The owner (PopupMenu1) will cleanup this menu item.
      TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
      int index = PopupMenu1->Items->Count;
      PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
      NewItem->Caption = "Menu Item " + IntToStr(index);
      NewItem->Tag = index;
      NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
      TNotifyEvent();
  }

  void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
  {
      TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
      if (ClickedItem)
      {
          switch (ClickedItem->Tag)
```

```
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
    PopupMenu1.AutoPopup := True;
    Edit1.PopupMenu := PopupMenu1;
    Edit2.PopupMenu := PopupMenu1;
  end;

  {
  These are the cut, copy, and paste OnClick events for the commands
  on the pop-up menu. Double click on the pop-up menu and add three
  TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
  The names will be automatically set to "Cut1", "Copy1", and
  "Paste1".  Then double click on the OnClick events and they will
  be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
  Note: only the selected portion of the string in the edit box will
  be copied, cut or replaced.
  }
  procedure TForm1.Copy1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
     Edit1.CopyToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CopyToClipboard
    else
      Beep;
  end;

  procedure TForm1.Cut1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.CutToClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.CutToClipboard
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
```

```
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
```

```
begin
  for index := 0 to 3 do
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;
end;
```

See Also

AutoPopup

Popup

OnContextPopup

39.1.3.5.21 TTabSet.Scroller Property

Pascal

```
property Scroller: TScroller;
```

C++

```
__property TScroller Scroller;
```

Description

This is Scroller, a member of class TTabSet.

39.1.3.5.22 TTabSet.SelectedColor Property

Determines the color of the selected tab in the tab set control.

Pascal

```
property SelectedColor: TColor;
```

C++

```
__property TColor SelectedColor;
```

Description

SelectedColor determines the color of the selected tab in the tab set control.

See Also

BackgroundColor (see page 4486)

DitherBackground (see page 4489)

Color

39.1.3.5.23 TTabSet.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
```

```
    begin
      for I := 1 to 10 do
        Items.Add('Item ' + IntToStr(I));
      Hint := 'Select an item';
      ShowHint := True;
    end;
    Application.ShowHint := True;
  end;
  {
  Note: To see an example that displays the hints of controls
  in some place other than in a Help Hint box, see the OnHint
  example.
  }
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

39.1.3.5.24 TTabSet.ShrinkToFit Property

Pascal

```
property ShrinkToFit: Boolean;
```

C++

```
__property Boolean ShrinkToFit;
```

Description

This is ShrinkToFit, a member of class TTabSet.

39.1.3.5.25 TTabSet.SoftTop Property

Specifies whether a line appears at the edge of the tab set, setting off the non-current tabs.

Pascal

```
property SoftTop: Boolean;
```

C++

```
__property Boolean SoftTop;
```

Description

When SoftTop is false (the default), a line appears at the edge of the non-current tabs. When SoftTop is true, this line is not drawn, softening the border of the tab set.

39.1.3.5.26 TTabSet.StartMargin Property

Determines how far, in pixels, the first tab appears from the left edge of the tab set control.

Pascal

```
property StartMargin: Integer;
```

C++

```
__property int StartMargin;
```

Description

Together with the EndMargin (see page 4494) property, StartMargin determines how many tabs fit in the tab set control.

See Also

EndMargin (see page 4494)

39.1.3.5.27 TTabSet.Style Property

Determines whether the tab set is owner-drawn.

Pascal

```
property Style: TTabStyle;
```

C++

```
__property TTabStyle Style;
```

Description

Owner-draw tabs can display objects other than strings, such as graphical images. Owner-draw tabs require more programming, however, because the application needs information on how to render the image for each tab in the tab set control.

Each time an item is displayed in an owner-drawn tab, two events occur. The first is the OnMeasureTab (see page 4527) event. In the OnMeasureTab (see page 4527) event handler, you write the code that calculates the width of the tab needed to hold the text or graphical image. After the OnMeasureTab (see page 4527) event, the OnDrawTab (see page 4519) event occurs. The code you write for the OnDrawTab (see page 4519) event draws the tab and its contents using the width found with the OnMeasureTab (see page 4527) event and the height specified as the value of the TabHeight (see page 4505) property.

Note: The TTabStyle (see page 4543) type defined in the tabs unit for the Style property differs from the TTabStyle (see page 4543) type defined in the comctrls unit. Do not to use or include the comctrls unit in source files where you are working with the tab control. Doing so can lead to compiler errors because of type mismatches.

See Also

TabHeight (see page 4505)

OnDrawTab (see page 4519)

OnMeasureTab (see page 4527)

39.1.3.5.28 TTabSet.TabHeight Property

Determines the height of the tabs in an owner-drawn tab set control.

Pascal

```
property TabHeight: Integer;
```

C++

```
__property int TabHeight;
```

Description

If the Style (see page 4505) property value is tsStandard, the TabHeight property value is ignored. If the Style (see page 4505) property value is tsOwnerDraw, TabHeight is the height of the tabs.

See Also

Style (see page 4505)

39.1.3.5.29 TTabSet.TabIndex Property

Determines which tab of a tab set control is currently selected.

Pascal

```
property TabIndex: Integer;
```

C++

```
__property int TabIndex;
```

Description

The value of TabIndex can be from –1 to one less than the number of strings in the Tabs (see page 4506) property array (Count – 1). TabIndex is an index to the Tabs (see page 4506) array property: A value of 0 indicates the first tab in the tab set control, 1 is the second tab, and so on. If no tabs are selected, TabIndex has a value of –1.

When a value is assigned to TabIndex, the OnClick (see page 4511) event for the tab set occurs, followed by the OnChange (see page 4511) event, just as if the user had clicked on a new tab.

See Also

Tabs (see page 4506)

OnChange (see page 4511)

OnClick

39.1.3.5.30 TTabSet.TabPosition Property

Pascal

```
property TabPosition: TTabPosition;
```

C++

```
__property TTabPosition TabPosition;
```

Description

This is TabPosition, a member of class TTabSet.

39.1.3.5.31 TTabSet.Tabs Property

Contains the list of text strings that appear on the tabs.

Pascal

```
property Tabs: TStrings;
```

C++

```
__property TStrings Tabs;
```

Description

For each string in the Tabs property, a new tab is created to display the text string. At design time, you enter the text strings you want to appear on the tabs using the String List editor. Double-click the value column of the Tabs property to display the String List editor. You can also change strings in the Tabs property at runtime.

See Also

VisibleTabs (see page 4508)

39.1.3.5.32 TTabSet.UnselectedColor Property

Determines the color of the tabs that aren't currently selected in the tab set control.

Pascal

```
property UnselectedColor: TColor;
```

C++

```
__property TColor UnselectedColor;
```

Description

UnselectedColor determines the color of the tabs that aren't currently selected in the tab set control.

See Also

BackgroundColor (see page 4486)

SelectedColor (see page 4502)

39.1.3.5.33 TTabSet.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
```

```
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

Enabled

Hide

Show

VisibleChanging

39.1.3.5.34 TTabSet.VisibleTabs Property

Contains the number of tabs currently visible in the tab set control.

Pascal

```
property VisibleTabs: Integer;
```

C++

```
__property int VisibleTabs;
```

Description

Use VisibleTabs to determine how many tabs are visible in the tab set.

See Also

Tabs (see page 4506)

39.1.3.6 TTabSet Events

The events of the TTabSet class are listed here.

Published Events

	Name	Description
	OnAlignInsertBefore (see page 4509)	Occurs when an object with custom alignment is aligned.
	OnAlignPosition (see page 4510)	Occurs when an object with custom alignment is aligned.
	OnChange (see page 4511)	Occurs just before a new tab is selected.
	OnClick (see page 4511)	Occurs when the user clicks the control.
	OnDragDrop (see page 4513)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4516)	Occurs when the user drags an object over a control.
	OnDrawTab (see page 4519)	Occurs when a tab in an owner-drawn tab set control needs to redisplay.
	OnEndDock (see page 4519)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4520)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4521)	Occurs when a control receives the input focus.
	OnExit (see page 4523)	Occurs when the input focus shifts away from one control to another.
	OnGetImageIndex (see page 4527)	This is OnGetImageIndex, a member of class TTabSet.
	OnMeasureTab (see page 4527)	Occurs when an application needs to redisplay a tab in an owner-draw tab set.
	OnMouseActivate (see page 4528)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4529)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4531)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4531)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4532)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4534)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnStartDock (see page 4536)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4537)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

39.1.3.6.1 TTabSet.OnAlignInsertBefore Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignInsertBefore;
```

C++

```
__property OnAlignInsertBefore;
```

Description

OnAlignInsertBefore occurs for each pair of child controls with an Align property of **alCustom**.

It can determine the order in which custom-aligned child controls are placed. OnAlignInsertBefore returns true if **C2** should be presented before **C1**, and false otherwise. It holds a TAlignInsertBeforeEvent type.

CustomAlignInsertBefore triggers the OnAlignInsertBefore event. If this event is defined, CustomAlignInsertBefore returns the value returned by OnAlignInsertBefore. Defining this event allows users to define the order of control alignment without overriding CustomAlignInsertBefore.

See Also

CustomAlignInsertBefore

Align

TAlignInsertBeforeEvent

39.1.3.6.2 TTabSet.OnAlignPosition Event

Occurs when an object with custom alignment is aligned.

Pascal

```
property OnAlignPosition;
```

C++

```
__property OnAlignPosition;
```

Description

OnAlignPosition occurs when child controls with an Align property of **alCustom** are aligned.

CustomAlignPosition triggers the OnAlignPosition event. If this event is defined, CustomAlignPosition uses the alignment parameters it obtains from OnAlignPosition. Defining this event allows users to set the alignment parameters without overriding CustomAlignPosition. It holds a TAlignPositionEvent type.

These are the parameters:

Parameter	Meaning
NewLeft | NewTop | NewWidth | NewHeight	Location and size of child control, as determined by container's size and the Anchors TControl_Anchors constraints for the control. OnAlignPosition can modify these values to reposition the control.
AlignRect	The client area in which the control is aligned. OnAlignPosition can modify this value.
AlignInfo	Alignment information in a TAlignInfo.
As implemented in TWinControl, OnAlignPosition does nothing.	

See Also

CustomAlignPosition

Align

Anchors

TAlignInfo

TAlignPositionEvent

39.1.3.6.3 TTabSet.OnChange Event

Occurs just before a new tab is selected.

Pascal

```
property OnChange: TTabChangeEvent;
```

C++

```
__property TTabChangeEvent OnChange;
```

Description

OnChange occurs just as the TabIndex (see page 4506) value is about to change. To prevent the TabIndex (see page 4506) value from changing, set AllowChange to false in the OnChange event handler.

39.1.3.6.4 TTabSet.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
```

```
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

39.1.3.6.5 TTabSet.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
```

```
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
```

```
    TForm1(Source).Left := X;
    TForm1(Source).Top := Y;
  end;

  procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := False;
    if (Source is TLabel) and (State = dsDragEnter) then
      (Source as TLabel).EndDrag(False);
  end;
  {
  This code requires a list box and three labels, each with a
  different font and color.  The DragMode property for each of
  the labels is dmAutomatic.  The user can select a label and
  drag it to a list box and drop it. When the label is dropped,
  the items in the list box assume the color and font of the
  dropped label.
  }

  procedure TForm1.FormCreate(Sender: TObject);
  begin
    ListBox1.Items.Add('Not');
    ListBox1.Items.Add('In');
    ListBox1.Items.Add('Alphabetical');
    ListBox1.Items.Add('Order');
  end;

  // This OnDragOver event handler permits the list box to
  // accept a dropped label:

  procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TLabel;
  end;

  // This OnDragDrop event handler implements the drop behavior.

  procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
  begin
    if (Sender is TListBox) and (Source is TLabel) then
    begin
      with Sender as TListBox do
      begin
        Font := (Source as TLabel).Font;
        Color := (Source as TLabel).Color;
        DoubleBuffered := true;
        Color := clWindow;
      end;
    end;
  end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

39.1.3.6.6 TTabSet.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
```

```
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
```

```
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

39.1.3.6.7 TTabSet.OnDrawTab Event

Occurs when a tab in an owner-drawn tab set control needs to redisplay.

Pascal

```
property OnDrawTab: TDrawTabEvent;
```

C++

```
__property TDrawTabEvent OnDrawTab;
```

Description

OnDrawTab occurs when a tab redisplays if the Style (see page 4505) property value is tsOwnerDraw. For example, it happens when the user selects a tab or scrolls the tabs.

You must write the code in the OnDrawTab event handler to draw the tab.

OnDrawTab occurs just after the OnMeasureTab (see page 4527) event, which contains the code to calculate the width of the tab needed. The height of the tab is determined by the value of the TabHeight (see page 4505) property of the tab set control. The code you write in the OnDrawTab event handler, therefore, must use the width determined with the OnMeasureTab (see page 4527) event to draw the tab.

See Also

TabHeight (see page 4505)

OnMeasureTab (see page 4527)

Style (see page 4505)

39.1.3.6.8 TTabSet.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

39.1.3.6.9 TTabSet.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
```

```
    else
      S := S + 'rejected!';
    Status.Caption := S;
  end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

39.1.3.6.10 TTabSet.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}
```

```
void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
```

```
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

39.1.3.6.11 TTabSet.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit

event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
```

```
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
```

```
  subnet mask edit box, the subnet address is automatically calculated
  using bit-wise AND operations and displayed in the third mask edit box.
  An additional test is made to assure that the values in the IP address
  and in the subnet mask are in the interval from 0 to 255,
  as this validation test cannot be accomplished
  using the edit mask property alone.
  }
  procedure TForm1.FormCreate(Sender: TObject);
  begin
    { initialize the maskedit boxes to contain the empty string }
    MaskEdit1.Text := '';
    MaskEdit2.Text := '';
    MaskEdit3.Text := '';

    { use the Courier font in the maskedit boxes }
    MaskEdit1.Font.Name := 'Courier';
    MaskEdit2.Font.Name := 'Courier';
    MaskEdit3.Font.Name := 'Courier';

    {
    initialize the mask of each maskedit box
    according to IP address coding conventions
    }
    MaskEdit1.EditMask := '!099.099.099.099;1; ';
    MaskEdit2.EditMask := '!099.099.099.099;1; ';
    MaskEdit3.EditMask := '!099.099.099.099;1; ';

    MaskEdit2.OnExit := MaskEdit2Exit;
  end;

  procedure TForm1.MaskEdit2Exit(Sender: TObject);
  var
    net1, net2, host1, host2,
    netmask1, netmask2, hostmask1, hostmask2,
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');
```

```
    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;

  procedure TForm1.Memo1Exit(Sender: TObject);
  begin
    Memo1.Color := clWindow;
  end;
```

See Also

OnEnter

ActiveControl

39.1.3.6.12 TTabSet.OnGetImageIndex Event

Pascal

```
property OnGetImageIndex: TTabGetImageEvent;
```

C++

```
__property TTabGetImageEvent OnGetImageIndex;
```

Description

This is OnGetImageIndex, a member of class TTabSet.

39.1.3.6.13 TTabSet.OnMeasureTab Event

Occurs when an application needs to redisplay a tab in an owner-draw tab set.

Pascal

```
property OnMeasureTab: TMeasureTabEvent;
```

C++

```
__property TMeasureTabEvent OnMeasureTab;
```

Description

OnMeasureTab occurs only if the Style (see page 4505) property is tsOwnerDraw. In the OnMeasureTab event handler, write the code to calculate the width needed to draw the tab. Your code is responsible for calculating and returning the tab width, depending on what you have drawn in the tab (if the tab is of Style (see page 4505) tsOwnerDraw). After the OnMeasureTab event occurs, the OnDrawTab (see page 4519) event occurs, when the OnDrawTab (see page 4519) event handler draws the tab using the width calculated in OnMeasureTab.

See Also

TabHeight (see page 4505)

OnDrawTab (see page 4519)

39.1.3.6.14 TTabSet.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

39.1.3.6.15 TTabSet.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
```

```
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;
```

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

39.1.3.6.16 TTabSet.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

39.1.3.6.17 TTabSet.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

39.1.3.6.18 TTabSet.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
```

```
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
```

```
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

39.1.3.6.19 TTabSet.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
```

```
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

39.1.3.6.20 TTabSet.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

39.1.3.6.21 TTabSet.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

39.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEdgePart (see page 4538)	This is record Tabs.TEdgePart.
	TEdgeType (see page 4538)	This is record Tabs.TEdgeType.
	TScrollBtn (see page 4539)	This is record Tabs.TScrollBtn.
	TScrollOrientation (see page 4539)	This is record Tabs.TScrollOrientation.
	TTabSetTabStyle (see page 4540)	This is record Tabs.TTabSetTabStyle.

39.2.1 Tabs.TEdgePart Enumeration

Pascal

```
TEdgePart = (
  epSelectedLeft,
  epUnselectedLeft,
  epSelectedRight,
  epUnselectedRight
);
```

C++

```
enum TEdgePart {
  epSelectedLeft,
  epUnselectedLeft,
  epSelectedRight,
  epUnselectedRight
};
```

File

Tabs

Description

This is record Tabs.TEdgePart.

39.2.2 Tabs.TEdgeType Enumeration

Pascal

```
TEdgeType = (
  etNone,
  etFirstIsSel,
  etFirstNotSel,
  etLastIsSel,
  etLastNotSel,
  etNotSelToSel,
  etSelToNotSel,
  etNotSelToNotSel
```

```
);
```

C++

```
enum TEdgeType {
  etNone,
  etFirstIsSel,
  etFirstNotSel,
  etLastIsSel,
  etLastNotSel,
  etNotSelToSel,
  etSelToNotSel,
  etNotSelToNotSel
};
```

File

Tabs

Description

This is record Tabs.TEdgeType.

39.2.3 Tabs.TScrollBtn Enumeration

Pascal

```
TScrollBtn = (
  sbLeft,
  sbRight
);
```

C++

```
enum TScrollBtn {
  sbLeft,
  sbRight
};
```

File

Tabs

Description

This is record Tabs.TScrollBtn.

39.2.4 Tabs.TScrollOrientation Enumeration

Pascal

```
TScrollOrientation = (
  soLeftRight,
  soUpDown
);
```

C++

```
enum TScrollOrientation {
  soLeftRight,
  soUpDown
};
```

File

Tabs

Description

This is record Tabs.TScrollOrientation.

39.2.5 Tabs.TTabSetTabStyle Enumeration

Pascal

```
TTabSetTabStyle = (
  tsStandard,
  tsOwnerDraw,
  tsSoftTabs,
  tsModernTabs,
  tsModernPopout
);
```

C++

```
enum TTabSetTabStyle {
  tsStandard,
  tsOwnerDraw,
  tsSoftTabs,
  tsModernTabs,
  tsModernPopout
};
```

File

Tabs

Description

This is record Tabs.TTabSetTabStyle.

39.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDrawTabEvent (see page 4541)	TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4470) control.
TMeasureTabEvent (see page 4541)	TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 4470) control.
TTabChangeEvent (see page 4542)	TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 4470) control changes.
TTabSetDrawTabEvent (see page 4542)	TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4470) control.
TTabStyle (see page 4543)	TTabStyle indicates whether a TTabSet (see page 4470) control is owner-drawn.

39.3.1 Tabs.TDrawTabEvent Type

TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4470) control.

Pascal

```
TDrawTabEvent = TTabSetDrawTabEvent;
```

C++

```
TTabSetDrawTabEvent TDrawTabEvent;
```

File

Tabs

Description

TDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4470) control.

Note: The TDrawTabEvent type defined in the tabs unit for the OnDrawTab event handler differs from the TDrawTabEvent type defined in the comctrls unit. Do not use the comctrls unit in source files where you are working with the tab control.

See Also

Procedural types

Creating events

39.3.2 Tabs.TMeasureTabEvent Type

TMeasureTabEvent is the type for event handlers that indicate the size of the tabs in an owner-drawn TTabSet (see page 4470) control.

Pascal

```
TMeasureTabEvent = procedure (Sender: TObject; Index: Integer; var TabWidth: Integer) of
```

```
object;
```

C++

```
(Sender: TObject; Index: Integer; var TabWidth: Integer) ( TMeasureTabEvent)();
```

File

Tabs

Description

Sender is the TTabSet (see page 4470) control whose tab is about to be drawn.

Index identifies the position of the tab in the tab set control.

TabWidth is the width of the tab.

See Also

Procedural types

Creating events

39.3.3 Tabs.TTabChangeEvent Type

TTabChangeEvent is the type for event handlers that respond when the current tab of a TTabSet (see page 4470) control changes.

Pascal

```
TTabChangeEvent = procedure (Sender: TObject; NewTab: Integer; var AllowChange: Boolean) of
object;
```

C++

```
(Sender: TObject; NewTab: Integer; var AllowChange: Boolean) ( TTabChangeEvent)();
```

File

Tabs

Description

Sender is the TTabSet (see page 4470) control whose tab is about to change.

NewTab is the tab that is about to become the selected tab.

AllowChange determines whether the change is permitted. If the event handler sets AllowChange to false, the user won't be able to select the new tab, in effect disabling it.

See Also

Procedural types

Creating events

39.3.4 Tabs.TTabSetDrawTabEvent Type

TTabSetDrawTabEvent is the type for event handlers that render the tab of an owner-drawn TTabSet (see page 4470) control.

Pascal

```
TTabSetDrawTabEvent = procedure (Sender: TObject; TabCanvas: TCanvas; R: TRect; Index:
Integer; Selected: Boolean) of object;
```

C++

```
(Sender: TObject; TabCanvas: TCanvas; R: TRect; Index: Integer; Selected: Boolean) (
TTabSetDrawTabEvent)();
```

File

Tabs

Description

TabCanvas is the canvas on which the item is drawn.

Index is the position of the tab in the tab set control.

R is the area in the tab where the item is to be drawn.

Selected indicates whether the tab is currently selected.

See Also

Procedural types

Creating events

39.3.5 Tabs.TTabStyle Type

TTabStyle indicates whether a TTabSet (see page 4470) control is owner-drawn.

Pascal

```
TTabStyle = TTabSetTabStyle;
```

C++

```
TTabSetTabStyle TTabStyle;
```

File

Tabs

Description

These are the possible values:

Value	Meaning
tsStandard	Each tab has the standard size and look.

tsOwnerDrawEach tab has the height specified with the TabHeight property and width needed to hold the text or glyph.

40 TConnect Namespace

This is namespace TConnect.

Topics

Name	Description
Classes (see page 4546)	The following table lists classes in this documentation.

Classes

	Name	Description
	TLocalConnection (see page 4546)	TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

40.1 Classes

The following table lists classes in this documentation.

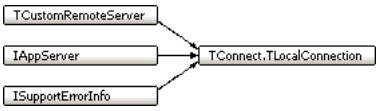
Classes

	Name	Description
	TLocalConnection (see page 4546)	TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

40.1.1 TLocalConnection Class

TLocalConnection represents the connection between a client dataset or XML broker and a provider that resides in the same application.

Class Hierarchy



Pascal

```
TLocalConnection = class(TCustomRemoteServer, IAppServer, ISupportErrorInfo);
```

C++

```
class TLocalConnection : public TCustomRemoteServer, public IAppServer, public
ISupportErrorInfo;
```

File

TConnect

Description

Use TLocalConnection to manage the connection between one or more client datasets or XML brokers and the providers that reside in the same application. When working with local providers, it is not necessary to set the RemoteServer property of a client dataset or XML broker to a TLocalConnection component. However, using TLocalConnection lets your application communicate directly with local providers through the AppServer (see page 4553) property. This provides access to methods that would otherwise be unavailable, and makes it easier to scale up to a multi-tiered application at a later time.

TLocalConnection implements the IAppServer interface.

Warning: TLocalConnection only links clients to providers that are instances of TDataSetProvider.

See Also

TCustomClientDataSet

TSocketConnection (see page 221)

TWebConnection (see page 254)

TDCOMConnection

IAppServer

TXMLBroker (see page 7303)

TDataSetProvider

TSoapConnection (see page 1188)

Specifying a Provider

Communicating with the Client Dataset

40.1.1.1 TLocalConnection Members

The following tables list the members exposed by TLocalConnection.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 4549)	This is AS_ApplyUpdates, a member of class TLocalConnection.
	AS_DataRequest (see page 4549)	This is AS_DataRequest, a member of class TLocalConnection.
	AS_Execute (see page 4549)	This is AS_Execute, a member of class TLocalConnection.
	AS_GetParams (see page 4549)	This is AS_GetParams, a member of class TLocalConnection.
	AS_GetProviderNames (see page 4550)	This is AS_GetProviderNames, a member of class TLocalConnection.
	AS_GetRecords (see page 4550)	This is AS_GetRecords, a member of class TLocalConnection.
	AS_RowRequest (see page 4550)	This is AS_RowRequest, a member of class TLocalConnection.
V	GetConnected (see page 4550)	Returns the value of the Connected property.
V	GetProvider (see page 4551)	This is GetProvider, a member of class TLocalConnection.
	InterfaceSupportsErrorInfo (see page 4552)	This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.
V	Notification (see page 4552)	Forwards notification messages to all owned components.
	RegisterProvider (see page 4552)	This is RegisterProvider, a member of class TLocalConnection.
	UnRegisterProvider (see page 4553)	This is UnRegisterProvider, a member of class TLocalConnection.

Public Methods

	Name	Description
V	Create (see page 4547)	Creates and initializes a local connection component.
V	Destroy (see page 4548)	Frees an instance of TLocalConnection (see page 4546).
V	GetProviderNames (see page 4551)	Executes a callback for all provider objects on the application server.
V	GetServer (see page 4551)	Returns the IAppServer interface for communicating with local providers.
V	SafeCallException (see page 4553)	Handles exceptions in methods declared using the safecall calling convention.

Public Properties

	Name	Description
R	AppServer (see page 4553)	Provides access to the IAppServer interface that handles communication with providers.
R	ProviderCount (see page 4554)	Indicates the number of providers in the Providers (see page 4554) property list.
R	Providers (see page 4554)	Lists the local providers with which this local connection component can communicate.

40.1.1.2 TLocalConnection.Create Constructor

Creates and initializes a local connection component.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLocalConnection(TComponent * AOwner);
```

Description

Call Create to instantiate TLocalConnection (see page 4546) at runtime. TLocalConnection (see page 4546) components placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies the component (usually the form or data module) that is responsible for freeing the TLocalConnection (see page 4546) instance.

See Also

Owner

40.1.1.3 TLocalConnection.Destroy Destructor

Frees an instance of TLocalConnection (see page 4546).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TLocalConnection();
```

Description

Do not call Destroy directly in an application. Instead, call Free, which checks that the TLocalConnection (see page 4546) reference is not nil before calling the destructor.

See Also

System::TObject::Free (see page 3372)

40.1.1.4 TLocalConnection Methods

The methods of the TLocalConnection class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 4549)	This is AS_ApplyUpdates, a member of class TLocalConnection.
	AS_DataRequest (see page 4549)	This is AS_DataRequest, a member of class TLocalConnection.
	AS_Execute (see page 4549)	This is AS_Execute, a member of class TLocalConnection.
	AS_GetParams (see page 4549)	This is AS_GetParams, a member of class TLocalConnection.
	AS_GetProviderNames (see page 4550)	This is AS_GetProviderNames, a member of class TLocalConnection.
	AS_GetRecords (see page 4550)	This is AS_GetRecords, a member of class TLocalConnection.
	AS_RowRequest (see page 4550)	This is AS_RowRequest, a member of class TLocalConnection.
V	GetConnected (see page 4550)	Returns the value of the Connected property.
V	GetProvider (see page 4551)	This is GetProvider, a member of class TLocalConnection.
	InterfaceSupportsErrorInfo (see page 4552)	This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.
V	Notification (see page 4552)	Forwards notification messages to all owned components.
	RegisterProvider (see page 4552)	This is RegisterProvider, a member of class TLocalConnection.
	UnRegisterProvider (see page 4553)	This is UnRegisterProvider, a member of class TLocalConnection.

Public Methods

	Name	Description
	GetProviderNames (see page 4551)	Executes a callback for all provider objects on the application server.
	GetServer (see page 4551)	Returns the IAppServer interface for communicating with local providers.
	SafeCallException (see page 4553)	Handles exceptions in methods declared using the safecall calling convention.

40.1.1.4.1 TLocalConnection.AS_ApplyUpdates Method

Pascal

```
function AS_ApplyUpdates(const ProviderName: WideString; Delta: OleVariant; MaxErrors:
Integer; out ErrorCount: Integer; var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(const BSTR ProviderName, OleVariant Delta, int
MaxErrors, int ErrorCount, OleVariant OwnerData);
```

Description

This is AS_ApplyUpdates, a member of class TLocalConnection.

40.1.1.4.2 TLocalConnection.AS_DataRequest Method

Pascal

```
function AS_DataRequest(const ProviderName: WideString; Data: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_DataRequest(const BSTR ProviderName, OleVariant Data);
```

Description

This is AS_DataRequest, a member of class TLocalConnection.

40.1.1.4.3 TLocalConnection.AS_Execute Method

Pascal

```
procedure AS_Execute(const ProviderName: WideString; const CommandText: WideString; var
Params: OleVariant; var OwnerData: OleVariant); safecall;
```

C++

```
__fastcall AS_Execute(const BSTR ProviderName, const BSTR CommandText, OleVariant Params,
OleVariant OwnerData);
```

Description

This is AS_Execute, a member of class TLocalConnection.

40.1.1.4.4 TLocalConnection.AS_GetParams Method

Pascal

```
function AS_GetParams(const ProviderName: WideString; var OwnerData: OleVariant): OleVariant;
safecall;
```

C++

```
__fastcall OleVariant AS_GetParams(const BSTR ProviderName, OleVariant OwnerData);
```

Description

This is AS_GetParams, a member of class TLocalConnection.

40.1.1.4.5 TLocalConnection.AS_GetProviderNames Method

Pascal

```
function AS_GetProviderNames: OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetProviderNames();
```

Description

This is AS_GetProviderNames, a member of class TLocalConnection.

40.1.1.4.6 TLocalConnection.AS_GetRecords Method

Pascal

```
function AS_GetRecords(const ProviderName: WideString; Count: Integer; out RecsOut: Integer;
Options: Integer; const CommandText: WideString; var Params: OleVariant; var OwnerData:
OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_GetRecords(const BSTR ProviderName, int Count, int RecsOut, int
Options, const BSTR CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is AS_GetRecords, a member of class TLocalConnection.

40.1.1.4.7 TLocalConnection.AS_RowRequest Method

Pascal

```
function AS_RowRequest(const ProviderName: WideString; Row: OleVariant; RequestType: Integer;
var OwnerData: OleVariant): OleVariant; safecall;
```

C++

```
__fastcall OleVariant AS_RowRequest(const BSTR ProviderName, OleVariant Row, int RequestType,
OleVariant OwnerData);
```

Description

This is AS_RowRequest, a member of class TLocalConnection.

40.1.1.4.8 TLocalConnection.GetConnected Method

Returns the value of the Connected property.

Pascal

```
function GetConnected: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetConnected();
```

Description

GetConnected is the protected read implementation of the Connected property. Descendant classes override this method to return true if a connection is open and false otherwise. As defined in TCustomConnection, GetConnected always returns false.

See Also

Connected

SetConnected

40.1.1.4.9 TLocalConnection.GetProvider Method

Pascal

```
function GetProvider(const ProviderName: string): TCustomProvider; virtual;
```

C++

```
virtual __fastcall TCustomProvider GetProvider(const UnicodeString ProviderName);
```

Description

This is GetProvider, a member of class TLocalConnection.

40.1.1.4.10 TLocalConnection.GetProviderNames Method

Executes a callback for all provider objects on the application server.

Pascal

```
procedure GetProviderNames(Proc: TGetStrProc); override;
```

C++

```
virtual __fastcall GetProviderNames(TGetStrProc Proc);
```

Description

Call GetProviderNames to execute the procedure specified by the Proc parameter for every provider exposed by the application server. For every provider listed by the remote data module's AS_GetProviderNames method, Proc is called with the name of the provider.

See Also

AS_GetProviderNames

AppServer

ProviderName

40.1.1.4.11 TLocalConnection.GetServer Method

Returns the IAppServer interface for communicating with local providers.

Pascal

```
function GetServer: IAppServer; override;
```

C++

```
virtual __fastcall IAppServer GetServer();
```

Description

GetServer returns the value of the AppServer (see page 4553) property.

See Also

AppServer (see page 4553)

40.1.1.4.12 TLocalConnection.InterfaceSupportsErrorInfo Method

Pascal

```
function InterfaceSupportsErrorInfo(const iid: TIID): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT InterfaceSupportsErrorInfo(const TIID iid);
```

Description

This is InterfaceSupportsErrorInfo, a member of class TLocalConnection.

40.1.1.4.13 TLocalConnection.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

40.1.1.4.14 TLocalConnection.RegisterProvider Method

Pascal

```
procedure RegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall RegisterProvider(TCustomProvider Prov);
```

Description

This is RegisterProvider, a member of class TLocalConnection.

40.1.1.4.15 TLocalConnection.SafeCallException Method

Handles exceptions in methods declared using the safecall calling convention.

Pascal

```
function SafeCallException(ExceptObject: TObject; ExceptAddr: Pointer): HResult; override;
```

C++

```
virtual __fastcall HRESULT SafeCallException(TObject * ExceptObject, void * ExceptAddr);
```

Description

SafeCallException handles exceptions in methods that use the safecall calling convention. Some classes that implement interfaces override this method to handle errors that might occur. TComponent calls the implementation of this method for the interface supported by the component, if it exists. If the component does not support interfaces, this method calls the SafeCallException method inherited from TObject, which returns E_UNEXPECTED. This is a default return value that is appropriate for classes that do not support any interfaces.

See Also

System::TObject::SafeCallException (see page 3380)

40.1.1.4.16 TLocalConnection.UnRegisterProvider Method

Pascal

```
procedure UnRegisterProvider(Prov: TCustomProvider);
```

C++

```
__fastcall UnRegisterProvider(TCustomProvider Prov);
```

Description

This is UnRegisterProvider, a member of class TLocalConnection.

40.1.1.5 TLocalConnection Properties

The properties of the TLocalConnection class are listed here.

Public Properties

	Name	Description
	AppServer (see page 4553)	Provides access to the IAppServer interface that handles communication with providers.
	ProviderCount (see page 4554)	Indicates the number of providers in the Providers (see page 4554) property list.
	Providers (see page 4554)	Lists the local providers with which this local connection component can communicate.

40.1.1.5.1 TLocalConnection.AppServer Property

Provides access to the IAppServer interface that handles communication with providers.

Pascal

```
property AppServer: IAppServer;
```

C++

```
__property IAppServer AppServer;
```

Description

Use AppServer to make explicit IAppServer calls to communicate with the local providers. The IAppServer interface lets you call provider methods that are otherwise unavailable because they are protected.

See Also

Providers (see page 4554)

AppServer

40.1.1.5.2 TLocalConnection.ProviderCount Property

Indicates the number of providers in the Providers (see page 4554) property list.

Pascal

```
property ProviderCount: integer;
```

C++

```
__property int ProviderCount;
```

Description

Read (see page 3634) ProviderCount to determine the number of local providers. This is the number of entries in the Providers (see page 4554) property list.

See Also

Providers (see page 4554)

40.1.1.5.3 TLocalConnection.Providers Property

Lists the local providers with which this local connection component can communicate.

Pascal

```
property Providers [const ProviderName: string]: TCustomProvider;
```

C++

```
__property TCustomProvider Providers[UnicodeString const ProviderName];
```

Description

Providers provides access to the local providers that are available through the IAppServer interface. Providers only lists the providers whose Exported property is set to true.

ProviderName is the name of a provider. If you specify a name of a provider that is not listed, TLocalConnection (see page 4546) raises an exception.

See Also

ProviderCount (see page 4554)

AppServer (see page 4553)

Exported

41 ThemedActnCtrls Namespace

This is namespace ThemedActnCtrls.

Topics

Name	Description
Classes (see page 4556)	The following table lists classes in this documentation.

Classes

	Name	Description
	TThemedAddRemoveItem (see page 4556)	This is class ThemedActnCtrls.TThemedAddRemoveItem.
	TThemedButtonControl (see page 4559)	This is class ThemedActnCtrls.TThemedButtonControl.
	TThemedCustomizePopup (see page 4561)	This is class ThemedActnCtrls.TThemedCustomizePopup.
	TThemedDropDownButton (see page 4563)	This is class ThemedActnCtrls.TThemedDropDownButton.
	TThemedExpandBtn (see page 4566)	This is class ThemedActnCtrls.TThemedExpandBtn.
	TThemedMenuButton (see page 4567)	This is class ThemedActnCtrls.TThemedMenuButton.
	TThemedMenuItem (see page 4569)	This is class ThemedActnCtrls.TThemedMenuItem.
	TThemedPopupMenu (see page 4573)	This is class ThemedActnCtrls.TThemedPopupMenu.
	TThemedToolScrollBtn (see page 4575)	This is class ThemedActnCtrls.TThemedToolScrollBtn.

41.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TThemedAddRemoveItem (see page 4556)	This is class ThemedActnCtrls.TThemedAddRemoveItem.
	TThemedButtonControl (see page 4559)	This is class ThemedActnCtrls.TThemedButtonControl.
	TThemedCustomizePopup (see page 4561)	This is class ThemedActnCtrls.TThemedCustomizePopup.
	TThemedDropDownButton (see page 4563)	This is class ThemedActnCtrls.TThemedDropDownButton.
	TThemedExpandBtn (see page 4566)	This is class ThemedActnCtrls.TThemedExpandBtn.
	TThemedMenuButton (see page 4567)	This is class ThemedActnCtrls.TThemedMenuButton.
	TThemedMenuItem (see page 4569)	This is class ThemedActnCtrls.TThemedMenuItem.
	TThemedPopupMenu (see page 4573)	This is class ThemedActnCtrls.TThemedPopupMenu.
	TThemedToolScrollBtn (see page 4575)	This is class ThemedActnCtrls.TThemedToolScrollBtn.

41.1.1 TThemedAddRemoveItem Class

Class Hierarchy

Pascal

```
TThemedAddRemoveItem = class(TCustomAddRemoveItem);
```

C++

```
class TThemedAddRemoveItem : public TCustomAddRemoveItem;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedAddRemoveItem.

41.1.1.1 TThemedAddRemoveItem Members

The following tables list the members exposed by TThemedAddRemoveItem.

Protected Methods

	Name	Description
	DrawBackground (see page 4557)	Draws a background image on the control.
	DrawGlyph (see page 4558)	Draws a symbol on the control.
	DrawSeparator (see page 4558)	This is DrawSeparator, a member of class TThemedAddRemoveItem.
	DrawText (see page 4558)	Draws text on the control.
	DrawUnusedEdges (see page 4559)	This is DrawUnusedEdges, a member of class TThemedAddRemoveItem.
	GetImageSize (see page 4559)	This is GetImageSize, a member of class TThemedAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 4557)	Calculates the size of the add/remove menu item.

41.1.1.2 TThemedAddRemoveItem Methods

The methods of the TThemedAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4557)	Draws a background image on the control.
	DrawGlyph (see page 4558)	Draws a symbol on the control.
	DrawSeparator (see page 4558)	This is DrawSeparator, a member of class TThemedAddRemoveItem.
	DrawText (see page 4558)	Draws text on the control.
	DrawUnusedEdges (see page 4559)	This is DrawUnusedEdges, a member of class TThemedAddRemoveItem.
	GetImageSize (see page 4559)	This is GetImageSize, a member of class TThemedAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 4557)	Calculates the size of the add/remove menu item.

41.1.1.2.1 TThemedAddRemoveItem.CalcBounds Method

Calculates the size of the add/remove menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

CalcBounds updates the Height, Width, and TextBounds properties to reflect current conditions (such as the item's caption and the current font).

See Also

Height

Width

TextBounds

41.1.1.2.2 TThemedAddRemoveItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

41.1.1.2.3 TThemedAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

41.1.1.2.4 TThemedAddRemoveItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TThemedAddRemoveItem.

41.1.1.2.5 TThemedAddRemoveItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

41.1.1.2.6 TThemedAddRemoveItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TThemedAddRemoveItem.

41.1.1.2.7 TThemedAddRemoveItem.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TThemedAddRemoveItem.

41.1.2 TThemedButtonControl Class

Class Hierarchy

Pascal

```
TThemedButtonControl = class(TCustomButtonControl);
```

C++

```
class TThemedButtonControl : public TCustomButtonControl;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedButtonControl.

41.1.2.1 TThemedButtonControl Members

The following tables list the members exposed by TThemedButtonControl.

Protected Methods

	Name	Description
	DrawBackground (see page 4560)	Draws a background image on the control.
	DrawFrame (see page 4560)	This is DrawFrame, a member of class TThemedButtonControl.
	DrawGlyph (see page 4560)	Draws a symbol on the control.

	DrawLargeGlyph (see page 4561)	This is DrawLargeGlyph, a member of class TThemedButtonControl.
	DrawText (see page 4561)	Draws text on the control.

41.1.2.2 TThemedButtonControl Methods

The methods of the TThemedButtonControl class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4560)	Draws a background image on the control.
	DrawFrame (see page 4560)	This is DrawFrame, a member of class TThemedButtonControl.
	DrawGlyph (see page 4560)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4561)	This is DrawLargeGlyph, a member of class TThemedButtonControl.
	DrawText (see page 4561)	Draws text on the control.

41.1.2.2.1 TThemedButtonControl.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

41.1.2.2.2 TThemedButtonControl.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TThemedButtonControl.

41.1.2.2.3 TThemedButtonControl.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

41.1.2.2.4 TThemedButtonControl.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TThemedButtonControl.

41.1.2.2.5 TThemedButtonControl.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

41.1.3 TThemedCustomizePopup Class

Class Hierarchy

Pascal

```
TThemedCustomizePopup = class(TCustomizeActionToolBar);
```

C++

```
class TThemedCustomizePopup : public TCustomizeActionToolBar;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedCustomizePopup.

41.1.3.1 TThemedCustomizePopup Members

The following tables list the members exposed by TThemedCustomizePopup.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 4562)	This is GetAddRemoveItemClass, a member of class TThemedCustomizePopup.
	GetDefaultColorMapClass (see page 4563)	This is GetDefaultColorMapClass, a member of class TThemedCustomizePopup.
	NCPaint (see page 4563)	This is NCPaint, a member of class TThemedCustomizePopup.

Public Methods

	Name	Description
	Create (see page 4562)	Creates and initializes a TCustomActionPopupMenu instance.

41.1.3.2 TThemedCustomizePopup.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TThemedCustomizePopup(TComponent * AOwner);
```

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

41.1.3.3 TThemedCustomizePopup Methods

The methods of the TThemedCustomizePopup class are listed here.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 4562)	This is GetAddRemoveItemClass, a member of class TThemedCustomizePopup.
	GetDefaultColorMapClass (see page 4563)	This is GetDefaultColorMapClass, a member of class TThemedCustomizePopup.
	NCPaint (see page 4563)	This is NCPaint, a member of class TThemedCustomizePopup.

41.1.3.3.1 TThemedCustomizePopup.GetAddRemoveItemClass Method

Pascal

```
function GetAddRemoveItemClass: TCustomAddRemoveItemClass; override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass();
```

Description

This is GetAddRemoveItemClass, a member of class TThemedCustomizePopup.

41.1.3.3.2 TThemedCustomizePopup.GetDefaultColorMapClass Method

Pascal

```
function GetDefaultColorMapClass: TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetDefaultColorMapClass();
```

Description

This is GetDefaultColorMapClass, a member of class TThemedCustomizePopup.

41.1.3.3.3 TThemedCustomizePopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TThemedCustomizePopup.

41.1.4 TThemedDropDownButton Class

Class Hierarchy

Pascal

```
TThemedDropDownButton = class(TCustomDropDownButton);
```

C++

```
class TThemedDropDownButton : public TCustomDropDownButton;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedDropDownButton.

41.1.4.1 TThemedDropDownButton Members

The following tables list the members exposed by TThemedDropDownButton.

Protected Methods

	Name	Description
	DrawBackground (see page 4564)	Draws a background image on the control.
	DrawFrame (see page 4564)	This is DrawFrame, a member of class TThemedDropDownButton.
	DrawGlyph (see page 4565)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4565)	This is DrawLargeGlyph, a member of class TThemedDropDownButton.
	GetDropDownButtonWidth (see page 4565)	This is GetDropDownButtonWidth, a member of class TThemedDropDownButton.
	GetPopupClass (see page 4565)	This is GetPopupClass, a member of class TThemedDropDownButton.

41.1.4.2 TThemedDropDownButton Methods

The methods of the TThemedDropDownButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4564)	Draws a background image on the control.
	DrawFrame (see page 4564)	This is DrawFrame, a member of class TThemedDropDownButton.
	DrawGlyph (see page 4565)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4565)	This is DrawLargeGlyph, a member of class TThemedDropDownButton.
	GetDropDownButtonWidth (see page 4565)	This is GetDropDownButtonWidth, a member of class TThemedDropDownButton.
	GetPopupClass (see page 4565)	This is GetPopupClass, a member of class TThemedDropDownButton.

41.1.4.2.1 TThemedDropDownButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

41.1.4.2.2 TThemedDropDownButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TThemedDropDownButton.

41.1.4.2.3 TThemedDropDownButton.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

41.1.4.2.4 TThemedDropDownButton.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TThemedDropDownButton.

41.1.4.2.5 TThemedDropDownButton.GetDropDownButtonWidth Method

Pascal

```
function GetDropDownButtonWidth: Integer; override;
```

C++

```
virtual __fastcall int GetDropDownButtonWidth();
```

Description

This is GetDropDownButtonWidth, a member of class TThemedDropDownButton.

41.1.4.2.6 TThemedDropDownButton.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; override;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TThemedDropDownButton.

41.1.5 TThemedExpandBtn Class

Class Hierarchy

Pascal

```
TThemedExpandBtn = class(TCustomMenuExpandBtn);
```

C++

```
class TThemedExpandBtn : public TCustomMenuExpandBtn;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedExpandBtn.

41.1.5.1 TThemedExpandBtn Members

The following tables list the members exposed by TThemedExpandBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 4566)	Draws a background image on the control.
	DrawFrame (see page 4567)	This is DrawFrame, a member of class TThemedExpandBtn.
	GetImageSize (see page 4567)	This is GetImageSize, a member of class TThemedExpandBtn.

41.1.5.2 TThemedExpandBtn Methods

The methods of the TThemedExpandBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4566)	Draws a background image on the control.
	DrawFrame (see page 4567)	This is DrawFrame, a member of class TThemedExpandBtn.
	GetImageSize (see page 4567)	This is GetImageSize, a member of class TThemedExpandBtn.

41.1.5.2.1 TThemedExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

41.1.5.2.2 TThemedExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TThemedExpandBtn.

41.1.5.2.3 TThemedExpandBtn.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TThemedExpandBtn.

41.1.6 TThemedMenuButton Class

Class Hierarchy

Pascal

```
TThemedMenuButton = class(TCustomMenuButton);
```

C++

```
class TThemedMenuButton : public TCustomMenuButton;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedMenuButton.

41.1.6.1 TThemedMenuButton Members

The following tables list the members exposed by TThemedMenuButton.

Protected Methods

	Name	Description
	DrawBackground (see page 4568)	Draws a background image on the control.
	DrawText (see page 4569)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 4568)	Calculates the size of the control.

41.1.6.2 TThemedMenuButton Methods

The methods of the TThemedMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4568)	Draws a background image on the control.
	DrawText (see page 4569)	Draws text on the control.

Public Methods

	Name	Description
	CalcBounds (see page 4568)	Calculates the size of the control.

41.1.6.2.1 TThemedMenuButton.CalcBounds Method

Calculates the size of the control.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

This method is largely the same as CalcBounds (TCustomActionControl), with one minor exception. The width of the control is set to be one pixel wider. For more information, see the documentation for the inherited method.

See Also

CalcBounds

41.1.6.2.2 TThemedMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

41.1.6.2.3 TThemedMenuButton.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

41.1.7 TThemedMenuItem Class

Class Hierarchy

Pascal

```
TThemedMenuItem = class(TCustomMenuItem);
```

C++

```
class TThemedMenuItem : public TCustomMenuItem;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedMenuItem.

41.1.7.1 TThemedMenuItem Members

The following tables list the members exposed by TThemedMenuItem.

Protected Methods

	Name	Description
	DrawBackground (see page 4570)	Draws a background image on the control.
	DrawGlyph (see page 4571)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4571)	This is DrawLargeGlyph, a member of class TThemedMenuItem.
	DrawSeparator (see page 4571)	This is DrawSeparator, a member of class TThemedMenuItem.

	DrawShadowedText (see page 4572)	This is DrawShadowedText, a member of class TThemedMenuItem.
	DrawSubMenuGlyph (see page 4572)	This is DrawSubMenuGlyph, a member of class TThemedMenuItem.
	DrawText (see page 4572)	Draws text on the control.
	DrawUnusedEdges (see page 4572)	This is DrawUnusedEdges, a member of class TThemedMenuItem.
	GetImageSize (see page 4573)	This is GetImageSize, a member of class TThemedMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 4570)	Calculates the size of the menu item.

41.1.7.2 TThemedMenuItem Methods

The methods of the TThemedMenuItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4570)	Draws a background image on the control.
	DrawGlyph (see page 4571)	Draws a symbol on the control.
	DrawLargeGlyph (see page 4571)	This is DrawLargeGlyph, a member of class TThemedMenuItem.
	DrawSeparator (see page 4571)	This is DrawSeparator, a member of class TThemedMenuItem.
	DrawShadowedText (see page 4572)	This is DrawShadowedText, a member of class TThemedMenuItem.
	DrawSubMenuGlyph (see page 4572)	This is DrawSubMenuGlyph, a member of class TThemedMenuItem.
	DrawText (see page 4572)	Draws text on the control.
	DrawUnusedEdges (see page 4572)	This is DrawUnusedEdges, a member of class TThemedMenuItem.
	GetImageSize (see page 4573)	This is GetImageSize, a member of class TThemedMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 4570)	Calculates the size of the menu item.

41.1.7.2.1 TThemedMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

41.1.7.2.2 TThemedMenuItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

41.1.7.2.3 TThemedMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

41.1.7.2.4 TThemedMenuItem.DrawLargeGlyph Method

Pascal

```
procedure DrawLargeGlyph(Location: TPoint); override;
```

C++

```
virtual __fastcall DrawLargeGlyph(TPoint Location);
```

Description

This is DrawLargeGlyph, a member of class TThemedMenuItem.

41.1.7.2.5 TThemedMenuItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TThemedMenuItem.

41.1.7.2.6 TThemedMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: string; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TThemedMenuItem.

41.1.7.2.7 TThemedMenuItem.DrawSubMenuGlyph Method

Pascal

```
procedure DrawSubMenuGlyph; override;
```

C++

```
virtual __fastcall DrawSubMenuGlyph();
```

Description

This is DrawSubMenuGlyph, a member of class TThemedMenuItem.

41.1.7.2.8 TThemedMenuItem.DrawText Method

Draws text on the control.

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: string); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

Use DrawText to draw the text specified by Text on the control within the area specified by ARect.

See Also

DrawBackground

DrawGlyph

41.1.7.2.9 TThemedMenuItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TThemedMenuItem.

41.1.7.2.10 TThemedMenuItem.GetImageSize Method

Pascal

```
function GetImageSize: TPoint; override;
```

C++

```
virtual __fastcall TPoint GetImageSize();
```

Description

This is GetImageSize, a member of class TThemedMenuItem.

41.1.8 TThemedPopupMenu Class

Class Hierarchy

Pascal

```
TThemedPopupMenu = class(TCustomActionPopupMenu);
```

C++

```
class TThemedPopupMenu : public TCustomActionPopupMenu;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedPopupMenu.

41.1.8.1 TThemedPopupMenu Members

The following tables list the members exposed by TThemedPopupMenu.

Protected Methods

	Name	Description
	GetExpandBtnClass (see page 4574)	This is GetExpandBtnClass, a member of class TThemedPopupMenu.
	NCPaint (see page 4574)	This is NCPaint, a member of class TThemedPopupMenu.
	PositionPopup (see page 4574)	This is PositionPopup, a member of class TThemedPopupMenu.
	VisibleChanging (see page 4575)	Provides the interface for a method that responds to changes in the control's visibility.

Public Methods

	Name	Description
	Create (see page 4573)	Creates and initializes a TCustomActionPopupMenu instance.

41.1.8.2 TThemedPopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TThemedPopupMenu(TComponent * AOwner);
```

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

41.1.8.3 TThemedPopupMenu Methods

The methods of the TThemedPopupMenu class are listed here.

Protected Methods

	Name	Description
	GetExpandBtnClass (see page 4574)	This is GetExpandBtnClass, a member of class TThemedPopupMenu.
	NCPaint (see page 4574)	This is NCPaint, a member of class TThemedPopupMenu.
	PositionPopup (see page 4574)	This is PositionPopup, a member of class TThemedPopupMenu.
	VisibleChanging (see page 4575)	Provides the interface for a method that responds to changes in the control's visibility.

41.1.8.3.1 TThemedPopupMenu.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TThemedPopupMenu.

41.1.8.3.2 TThemedPopupMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TThemedPopupMenu.

41.1.8.3.3 TThemedPopupMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TThemedPopupMenu.

41.1.8.3.4 TThemedPopupMenu.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible

41.1.9 TThemedToolScrollBtn Class

Class Hierarchy

Pascal

```
TThemedToolScrollBtn = class(TCustomToolScrollBtn);
```

C++

```
class TThemedToolScrollBtn : public TCustomToolScrollBtn;
```

File

ThemedActnCtrls

Description

This is class ThemedActnCtrls.TThemedToolScrollBtn.

41.1.9.1 TThemedToolScrollBtn Members

The following tables list the members exposed by TThemedToolScrollBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 4576)	Draws a background image on the control.

41.1.9.2 TThemedToolScrollBtn Methods

The methods of the TThemedToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 4576)	Draws a background image on the control.

41.1.9.2.1 TThemedToolScrollBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

42 Themes Namespace

This is namespace Themes.

Topics

Name	Description
Classes (see page 4579)	The following table lists classes in this documentation.
Functions (see page 4593)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4595)	The following table lists structs, records, enums in this documentation.
Types (see page 4629)	The following table lists types in this documentation.
Variables (see page 4631)	The following table lists variables in this documentation.

Classes

	Name	Description
	TThemeServices (see page 4579)	TThemeServices is a class used for communicating with the OS theme engine.

Functions

	Name	Description
	ThemeControl (see page 4593)	Checks whether a control should be themed.
	ThemeServices (see page 4593)	Returns an instance of the TThemeServices (see page 4579) class.
	UnthemedDesigner (see page 4594)	Checks whether the VCL Forms designer supports theming for a control.

Structs, Records, Enums

	Name	Description
	TThemedButton (see page 4595)	TThemedButton describes the possible themable parts of a button.
	TThemedClock (see page 4598)	TThemedClock describes the possible themable parts of a clock.
	TThemedComboBox (see page 4598)	TThemedComboBox describes the possible themable parts of a combo box.
	TThemedEdit (see page 4599)	TThemedEdit describes the possible themable parts of an edit.
	TThemedElement (see page 4600)	TThemedElement describes the possible themed user interface elements.
	TThemedElementDetails (see page 4602)	TThemedElementDetails describes a themed user interface element.
	TThemedExplorerBar (see page 4603)	TThemedExplorerBar describes the possible themable parts of an explorer bar control.
	TThemedHeader (see page 4604)	TThemedHeader describes the possible themable parts of a header control.
	TThemedListview (see page 4605)	TThemedListview describes the possible themable parts of a list view.
	TThemedMenu (see page 4606)	TThemedMenu describes the possible themable parts of a menu.
	TThemedPage (see page 4607)	TThemedPage describes the possible themable parts of a tab page.
	TThemedProgress (see page 4608)	TThemedProgress describes the possible themable parts of a progress bar.

	TThemedRebar (see page 4609)	TThemedRebar describes the possible themable parts of a rebar control.
	TThemedScrollBar (see page 4610)	TThemedScrollBar describes the possible themable parts of a scroll bar.
	TThemedSpin (see page 4612)	TThemedSpin describes the possible themable parts of a spin control.
	TThemedStartPanel (see page 4613)	TThemedStartPanel describes the possible themable parts of a start panel.
	TThemedStatus (see page 4614)	TThemedStatus describes the possible themable parts of a status bar.
	TThemedTab (see page 4615)	TThemedTab describes the possible themable parts of a tab.
	TThemedTaskBand (see page 4616)	TThemedTaskBand describes the possible themable parts of a task band.
	TThemedTaskBar (see page 4617)	TThemedTaskBar describes the possible themable parts of a system task bar.
	TThemedToolBar (see page 4618)	TThemedToolBar describes the possible themable parts of a tool bar.
	TThemedToolTip (see page 4620)	TThemedToolTip describes the possible themable parts of a tool tip.
	TThemedTrackBar (see page 4621)	TThemedTrackBar describes the possible themable parts of a track bar.
	TThemedTrayNotify (see page 4622)	TThemedTrayNotify describes the possible themable parts of a system tray notification area.
	TThemedTreeview (see page 4623)	TThemedTreeview describes the possible themable parts of a tree view.
	TThemedWindow (see page 4624)	TThemedWindow describes the possible themable parts of a window.

Types

Name	Description
PThemedElementDetails (see page 4629)	PThemedElementDetails is pointer to a TThemedElementDetails (see page 4602) record.
TThemeData (see page 4629)	TThemeData is an array of theme handles.
TThemeServicesClass (see page 4630)	TThemeServicesClass defines the metaclass for TThemeServices (see page 4579).

Variables

Name	Description
ThemeServicesClass (see page 4631)	ThemeServicesClass contains the metaclass reference for TThemeServices (see page 4579).

42.1 Classes

The following table lists classes in this documentation.

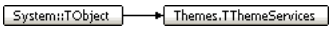
Classes

	Name	Description
	TThemeServices (see page 4579)	TThemeServices is a class used for communicating with the OS theme engine.

42.1.1 TThemeServices Class

TThemeServices is a class used for communicating with the OS theme engine.

Class Hierarchy



Pascal

```
TThemeServices = class(TObject);
```

C++

```
class TThemeServices : public TObject;
```

File

Themes

Description

Use properties and methods of TThemeServices to communicate with the theme engine of the underlying Operating System (see page 3159). Do not create instances of TThemeServices, since only one instance is allowed to be created in the VCL. Rather, use the ThemeServices (see page 4593) routine to retrieve a singleton instance of the TThemeServices class.

See Also

ThemeServices (see page 4593)

42.1.1.1 TThemeServices Members

The following tables list the members exposed by TThemeServices.

Protected Methods

	Name	Description
	DoOnThemeChange (see page 4583)	DoOnThemeChange triggers the OnThemeChange (see page 4592) event.
	UnloadThemeData (see page 4590)	Clears all the internal data for the current theme.

Public Events

	Name	Description
	OnThemeChange (see page 4592)	Occurs when the Operating System (see page 3159) theme changes.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	ApplyThemeChange (see page 4582)	Resets the application theme.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ColorToRGB (see page 4582)	This is the overview for the ColorToRGB method overload.
	ContentRect (see page 4583)	Returns the rectangle of the background area of a themed control.
	Create (see page 4581)	Creates and initializes an instance of the TThemeServices (see page 4579) class.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 4581)	Destroys an instance of TThemeServices (see page 4579) class.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DrawEdge (see page 4584)	This is the overview for the DrawEdge method overload.
	DrawElement (see page 4584)	This is the overview for the DrawElement method overload.
	DrawIcon (see page 4585)	Draws an icon over a themed user interface element.
	DrawParentBackground (see page 4586)	This is the overview for the DrawParentBackground method overload.
	DrawText (see page 4586)	Draws text over a themed user interface element.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetElementDetails (see page 4587)	This is the overview for the GetElementDetails method overload.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	HasTransparentParts (see page 4589)	Checks whether a given user interface element has transparent parts.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	PaintBorder (see page 4590)	Draws the border of a control.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.
	UpdateThemes (see page 4590)	Resets the application theme.

Public Properties

	Name	Description
	Theme (see page 4591)	Represents a set of handles associated with each user interface element type.

	ThemesAvailable (see page 4591)	Specifies whether the current Operation System (see page 3159) supports themes.
	ThemesEnabled (see page 4592)	Specifies whether the themes are enabled for this application.

42.1.1.2 TThemeServices.Create Constructor

Creates and initializes an instance of the TThemeServices (see page 4579) class.

Pascal

```
constructor Create; virtual;
```

C++

```
virtual __fastcall TThemeServices();
```

Description

Use Create to create and initialize an instance of the TThemeServices (see page 4579) class. Do not call this method directly. TThemeServices (see page 4579) is designed as a singleton, which means that only one instance should be created at a time. Use ThemeServices (see page 4593) routine to retrieve the instance of the TThemeServices (see page 4579) class.

See Also

ThemeServices (see page 4593)

42.1.1.3 TThemeServices.Destroy Destructor

Destroys an instance of TThemeServices (see page 4579) class.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TThemeServices();
```

Description

Do not call Destroy. There is only one instance of TThemeServices (see page 4579) present at run-time. Calling Destroy destroys that instance and crashes the application.

See Also

System::TObject::Free (see page 3372)

42.1.1.4 TThemeServices Methods

The methods of the TThemeServices class are listed here.

Protected Methods

	Name	Description
	DoOnThemeChange (see page 4583)	DoOnThemeChange triggers the OnThemeChange (see page 4592) event.
	UnloadThemeData (see page 4590)	Clears all the internal data for the current theme.

Public Methods

	Name	Description
	ApplyThemeChange (see page 4582)	Resets the application theme.

	ColorToRGB (see page 4582)	This is the overview for the ColorToRGB method overload.
	ContentRect (see page 4583)	Returns the rectangle of the background area of a themed control.
	DrawEdge (see page 4584)	This is the overview for the DrawEdge method overload.
	DrawElement (see page 4584)	This is the overview for the DrawElement method overload.
	DrawIcon (see page 4585)	Draws an icon over a themed user interface element.
	DrawParentBackground (see page 4586)	This is the overview for the DrawParentBackground method overload.
	DrawText (see page 4586)	Draws text over a themed user interface element.
	GetElementDetails (see page 4587)	This is the overview for the GetElementDetails method overload.
	HasTransparentParts (see page 4589)	Checks whether a given user interface element has transparent parts.
	PaintBorder (see page 4590)	Draws the border of a control.
	UpdateThemes (see page 4590)	Resets the application theme.

42.1.1.4.1 TThemeServices.ApplyThemeChange Method

Resets the application theme.

Pascal

```
procedure ApplyThemeChange;
```

C++

```
__fastcall ApplyThemeChange();
```

Description

ApplyThemeChange is called automatically by the TApplication object whenever the Operating System (see page 3159) theme is changed.

Note: ApplyThemeChange is used internally by the VCL and should not be called manually by applications.

See Also

ThemeServices (see page 4593)

Application

42.1.1.4.2 ColorToRGB Method

This is the overview for the ColorToRGB method overload.

Overload List

	Name	Description
	TThemeServices.ColorToRGB (TColor) (see page 4582)	Converts a TColor to its RGB representation.
	TThemeServices.ColorToRGB (TColor, PThemedElementDetails) (see page 4582)	Converts a TColor to its RGB representation.

42.1.1.4.2.1 TThemeServices.ColorToRGB Method (TColor)

Converts a TColor to its RGB representation.

Pascal

```
function ColorToRGB(Color: TColor): COLORREF; overload;
function ColorToRGB(Color: TColor; Details: PThemedElementDetails): COLORREF; overload;
```

C++

```
__fastcall COLORREF ColorToRGB(TColor Color);
__fastcall COLORREF ColorToRGB(TColor Color, PThemedElementDetails Details);
```

Description

Call ColorToRGB to convert a TColor to its RGB representation. ColorToRGB is only useful to applications that require the colors of controls when theming is enabled; otherwise applications should use the ColorToRGB routine located in Graphics unit.

See Also

ThemeServices (see page 4593)

ColorToRGB

42.1.1.4.3 TThemeServices.ContentRect Method

Returns the rectangle of the background area of a themed control.

Pascal

```
function ContentRect(DC: HDC; Details: TThemedElementDetails; BoundingRect: TRect): TRect;
```

C++

```
__fastcall TRect ContentRect(HDC DC, TThemedElementDetails Details, TRect BoundingRect);
```

Description

Call ContentRect to retrieve the rectangle that contains the background area of a control in a themed environment. ContentRect expects a TThemedElementDetails (see page 4602) parameter that describes the user interface element from which the informations is being retrieved.

See Also

ThemeServices (see page 4593)

DrawEdge (see page 4584)

42.1.1.4.4 TThemeServices.DoOnThemeChange Method

DoOnThemeChange triggers the OnThemeChange (see page 4592) event.

Pascal

```
procedure DoOnThemeChange; virtual;
```

C++

```
virtual __fastcall DoOnThemeChange();
```

Description

DoOnThemeChange is called whenever the Operating System (see page 3159) theme is changed. DoOnThemeChange simply calls the user supplied OnThemeChange (see page 4592) event, otherwise nothing happens.

See Also

OnThemeChange (see page 4592)

TThemeServices (see page 4579)

42.1.1.4.5 DrawEdge Method

This is the overview for the DrawEdge method overload.

Overload List

	Name	Description
	TThemeServices.DrawEdge (HDC, TThemedElementDetails, TRect, Cardinal, Cardinal) (see page 4584)	Draws the edge of a themed user interface element.
	TThemeServices.DrawEdge (HDC, TThemedElementDetails, TRect, Cardinal, Cardinal, PRect) (see page 4584)	Draws the edge of a themed user interface element.

42.1.1.4.5.1 TThemeServices.DrawEdge Method (HDC, TThemedElementDetails, TRect, Cardinal, Cardinal)

Draws the edge of a themed user interface element.

Pascal

```
procedure DrawEdge(DC: HDC; Details: TThemedElementDetails; const R: TRect; Edge: Cardinal;
Flags: Cardinal); overload;
procedure DrawEdge(DC: HDC; Details: TThemedElementDetails; const R: TRect; Edge: Cardinal;
Flags: Cardinal; ContentRect: PRect); overload;
```

C++

```
__fastcall DrawEdge(HDC DC, TThemedElementDetails Details, const TRect R, unsigned Edge,
unsigned Flags);
__fastcall DrawEdge(HDC DC, TThemedElementDetails Details, const TRect R, unsigned Edge,
unsigned Flags, PRect ContentRect);
```

Description

Call DrawEdge to draw the edge of a themed user interface element. The area is drawn using the current Operation System (see page 3159) theme.

See Also

ThemeServices (see page 4593)

DrawElement (see page 4585)

DrawIcon (see page 4585)

42.1.1.4.6 DrawElement Method

This is the overview for the DrawElement method overload.

Overload List

	Name	Description
	TThemeServices.DrawElement (HDC, TThemedElementDetails, TRect) (see page 4585)	Draws a themed user interface element.

	TThemeServices.DrawElement (HDC, TThemedElementDetails, TRect, PRect) (see page 4585)	Draws a themed user interface element.
	TThemeServices.DrawElement (HDC, TThemedElementDetails, TRect, TRect) (see page 4585)	Draws a themed user interface element.

42.1.1.4.6.1 TThemeServices.DrawElement Method (HDC, TThemedElementDetails, TRect)

Draws a themed user interface element.

Pascal

```
procedure DrawElement(DC: HDC; Details: TThemedElementDetails; const R: TRect); overload;
procedure DrawElement(DC: HDC; Details: TThemedElementDetails; const R: TRect; ClipRect:
TRect); overload;
procedure DrawElement(DC: HDC; Details: TThemedElementDetails; const R: TRect; ClipRect:
PRect); overload;
```

C++

```
__fastcall DrawElement(HDC DC, TThemedElementDetails Details, const TRect R);
__fastcall DrawElement(HDC DC, TThemedElementDetails Details, const TRect R, TRect ClipRect);
__fastcall DrawElement(HDC DC, TThemedElementDetails Details, const TRect R, PRect ClipRect);
```

Description

Call DrawElement to draw a themed user interface element given by the `Details` parameter on a device context. The element is drawn using the current Operation System (see page 3159) theme.

See Also

ThemeServices (see page 4593)

DrawIcon (see page 4585)

DrawEdge (see page 4584)

42.1.1.4.7 TThemeServices.DrawIcon Method

Draws an icon over a themed user interface element.

Pascal

```
procedure DrawIcon(DC: HDC; Details: TThemedElementDetails; const R: TRect; himl: HIMAGELIST;
Index: Integer);
```

C++

```
__fastcall DrawIcon(HDC DC, TThemedElementDetails Details, const TRect R, HIMAGELIST himl, int
Index);
```

Description

Call DrawIcon to draw an icon over a themed user interface element. The icon is drawn using the current Operation System (see page 3159) theme constraints.

See Also

ThemeServices (see page 4593)

DrawElement (see page 4585)

DrawEdge (see page 4584)

42.1.1.4.8 DrawParentBackground Method

This is the overview for the DrawParentBackground method overload.

Overload List

	Name	Description
	TThemeServices.DrawParentBackground (HWND, HDC, PThemedElementDetails, Boolean, PRect) (see page 4586)	Draws the parent background of the themed user interface element.
	TThemeServices.DrawParentBackground (HWND, HDC, PThemedElementDetails, Boolean, TRect) (see page 4586)	Draws the parent background of the themed user interface element.
	TThemeServices.DrawParentBackground (HWND, HDC, TThemedElementDetails, Boolean, PRect) (see page 4586)	Draws the parent background of the themed user interface element.

42.1.1.4.8.1 TThemeServices.DrawParentBackground Method (HWND, HDC, PThemedElementDetails, Boolean, PRect)

Draws the parent background of the themed user interface element.

Pascal

```
procedure DrawParentBackground(Window: HWND; Target: HDC; Details: PThemedElementDetails;
OnlyIfTransparent: Boolean; Bounds: PRect = nil); overload;
procedure DrawParentBackground(Window: HWND; Target: HDC; Details: PThemedElementDetails;
OnlyIfTransparent: Boolean; const Bounds: TRect); overload;
procedure DrawParentBackground(Window: HWND; Target: HDC; const Details:
TThemedElementDetails; OnlyIfTransparent: Boolean; Bounds: PRect = nil); overload;
```

C++

```
__fastcall DrawParentBackground(HWND Window, HDC Target, PThemedElementDetails Details,
Boolean OnlyIfTransparent, PRect Bounds = nil);
__fastcall DrawParentBackground(HWND Window, HDC Target, PThemedElementDetails Details,
Boolean OnlyIfTransparent, const TRect Bounds);
__fastcall DrawParentBackground(HWND Window, HDC Target, const TThemedElementDetails Details,
Boolean OnlyIfTransparent, PRect Bounds = nil);
```

Description

Call DrawParentBackground to draw the background of a themed user interface element's parent. The area is drawn using the current Operation System (see page 3159) theme.

See Also

ThemeServices (see page 4593)

DrawElement (see page 4585)

DrawIcon (see page 4585)

DrawEdge (see page 4584)

42.1.1.4.9 TThemeServices.DrawText Method

Draws text over a themed user interface element.

Pascal

```
procedure DrawText(DC: HDC; Details: TThemedElementDetails; const S: WideString; R: TRect;
Flags: Cardinal; Flags2: Cardinal);
```

C++

```
__fastcall DrawText(HDC DC, TThemedElementDetails Details, const BSTR S, TRect R, unsigned
Flags, unsigned Flags2);
```

Description

Call DrawText to draw text over a themed user interface element. The text is drawn using the current Operation System (see page 3159) theme constraints.

See Also

ThemeServices (see page 4593)

DrawElement (see page 4585)

DrawEdge (see page 4584)

DrawIcon (see page 4585)

42.1.1.4.10 GetElementDetails Method

This is the overview for the GetElementDetails method overload.

Overload List

	Name	Description
	TThemeServices.GetElementDetails (TThemedButton) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedClock) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedComboBox) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedEdit) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedExplorerBar) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedHeader) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedListView) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedMenu) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedPage) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedProgress) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedRebar) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedScrollBar) (see page 4588)	Retrieves the details of a themed user interface element.

	TThemeServices.GetElementDetails (TThemedSpin) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedStartPanel) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedStatus) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedTab) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedTaskBand) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedTaskBar) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedToolBar) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedToolTip) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedTrackBar) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedTrayNotify) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedTreeview) (see page 4588)	Retrieves the details of a themed user interface element.
	TThemeServices.GetElementDetails (TThemedWindow) (see page 4588)	Retrieves the details of a themed user interface element.

42.1.1.4.10.1 TThemeServices.GetElementDetails Method (TThemedButton)

Retrieves the details of a themed user interface element.

Pascal

```
function GetElementDetails(Detail: TThemedButton): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedClock): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedComboBox): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedEdit): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedExplorerBar): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedHeader): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedListView): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedMenu): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedPage): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedProgress): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedRebar): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedScrollBar): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedSpin): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedStartPanel): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedStatus): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedTab): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedTaskBand): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedTaskBar): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedToolBar): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedToolTip): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedTrackBar): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedTrayNotify): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedTreeview): TThemedElementDetails; overload;
function GetElementDetails(Detail: TThemedWindow): TThemedElementDetails; overload;
```

C++

```
__fastcall TThemedElementDetails GetElementDetails(TThemedButton Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedClock Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedComboBox Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedEdit Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedExplorerBar Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedHeader Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedListView Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedMenu Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedPage Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedProgress Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedRebar Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedScrollBar Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedSpin Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedStartPanel Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedStatus Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedTab Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedTaskBand Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedTaskBar Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedToolBar Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedToolTip Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedTrackBar Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedTrayNotify Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedTreeview Detail);
__fastcall TThemedElementDetails GetElementDetails(TThemedWindow Detail);
```

Description

Call GetElementDetails to retrieve the details of a themed user interface element. Each overloaded variant of GetElementDetails retrieves the details for a different user interface element. The retrieved value is a TThemedElementDetails (see page 4602) record associated with the input element.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

DrawElement (see page 4585)

DrawIcon (see page 4585)

42.1.1.4.11 TThemeServices.HasTransparentParts Method

Checks whether a given user interface element has transparent parts.

Pascal

```
function HasTransparentParts(Details: TThemedElementDetails): Boolean;
```

C++

```
__fastcall Boolean HasTransparentParts(TThemedElementDetails Details);
```

Description

Call HasTransparentParts to check whether a given user interface element has transparent parts under the current Operating System (see page 3159) theme.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

GetElementDetails (see page 4588)

42.1.1.4.12 TThemeServices.PaintBorder Method

Draws the border of a control.

Pascal

```
procedure PaintBorder(Control: TWinControl; EraseLRCorner: Boolean);
```

C++

```
__fastcall PaintBorder(TWinControl * Control, Boolean EraseLRCorner);
```

Description

Call PaintBorder to draw the border of a control. The area is drawn using the current Operation System (see page 3159) theme.

See Also

ThemeServices (see page 4593)

DrawElement (see page 4585)

DrawIcon (see page 4585)

42.1.1.4.13 TThemeServices.UnloadThemeData Method

Clears all the internal data for the current theme.

Pascal

```
procedure UnloadThemeData;
```

C++

```
__fastcall UnloadThemeData();
```

Description

UnloadThemeData is automatically called by the TApplication object whenever the Operating System (see page 3159) theme is changed.

Note: UnloadThemeData is used internally by the VCL and should not be called manually by applications.

See Also

ThemeServices (see page 4593)

Application

42.1.1.4.14 TThemeServices.UpdateThemes Method

Resets the application theme.

Pascal

```
procedure UpdateThemes;
```

C++

```
__fastcall UpdateThemes();
```

Description

UpdateThemes is automatically called by the TApplication object whenever the Operating System (see page 3159) theme is changed.

Note: UpdateThemes is used internally by the VCL and should not be called manually by applications.

See Also

ThemeServices (see page 4593)

Application

42.1.1.5 TThemeServices Properties

The properties of the TThemeServices class are listed here.

Public Properties

	Name	Description
	Theme (see page 4591)	Represents a set of handles associated with each user interface element type.
	ThemesAvailable (see page 4591)	Specifies whether the current Operation System (see page 3159) supports themes.
	ThemesEnabled (see page 4592)	Specifies whether the themes are enabled for this application.

42.1.1.5.1 TThemeServices.Theme Property

Represents a set of handles associated with each user interface element type.

Pascal

```
property Theme [Element: TThemedElement]: HTHEME;
```

C++

```
__property HTHEME Theme[TThemedElement Element];
```

Description

Use Theme to retrieve Windows handles for each user interface element. Windows API uses the returned handles to draw specific UI elements. The property is only used internally, so normally you do not need to use it.

See Also

ThemeServices (see page 4593)

ThemesAvailable (see page 4591)

42.1.1.5.2 TThemeServices.ThemesAvailable Property

Specifies whether the current Operation System (see page 3159) supports themes.

Pascal

```
property ThemesAvailable: Boolean;
```

C++

```
__property Boolean ThemesAvailable;
```

Description

Use ThemesAvailable to check whether the current Operating System (see page 3159) support themes.

See Also

ThemeServices (see page 4593)

ThemesEnabled (see page 4592)

42.1.1.5.3 TThemeServices.ThemesEnabled Property

Specifies whether the themes are enabled for this application.

Pascal

```
property ThemesEnabled: Boolean;
```

C++

```
__property Boolean ThemesEnabled;
```

Description

Use ThemesEnabled to check whether the themes are enabled for this application.

See Also

ThemeServices (see page 4593)

ThemesAvailable (see page 4591)

42.1.1.6 TThemeServices Events

The events of the TThemeServices class are listed here.

Public Events

	Name	Description
	OnThemeChange (see page 4592)	Occurs when the Operating System (see page 3159) theme changes.

42.1.1.6.1 TThemeServices.OnThemeChange Event

Occurs when the Operating System (see page 3159) theme changes.

Pascal

```
property OnThemeChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnThemeChange;
```

Description

Use the OnThemeChange event to implement custom processing that should occur whenever the Operation System (see page 3159) theme changes.

See Also

ThemeServices (see page 4593)

DoOnThemeChange (see page 4583)

42.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ThemeControl (see page 4593)	Checks whether a control should be themed.
	ThemeServices (see page 4593)	Returns an instance of the TThemeServices (see page 4579) class.
	UnthemedDesigner (see page 4594)	Checks whether the VCL Forms designer supports theming for a control.

42.2.1 Themes.ThemeControl Function

Checks whether a control should be themed.

Pascal

```
function ThemeControl(AControl: TControl): Boolean;
```

C++

```
Boolean ThemeControl(TControl * AControl);
```

File

Themes

Description

Call ThemeControl to check whether a control should be themed. `AControl` is the control that needs to be checked. ThemeControl returns **True** if the control can be themed and **False** otherwise.

Note: ThemeControl is used by the VCL Forms designer at design time. Applications should not call this routine directly.

See Also

UnthemedDesigner (see page 4594)

ThemeServices (see page 4593)

TThemeServices (see page 4579)

42.2.2 Themes.ThemeServices Function

Returns an instance of the TThemeServices (see page 4579) class.

Pascal

```
function ThemeServices: TThemeServices;
```

C++

```
TThemeServices ThemeServices();
```

File

Themes

Description

Call ThemeServices to obtain an instance of the TThemeServices (see page 4579) class. Normally, only one instance of TThemeServices (see page 4579) should be created at a time. ThemeServices makes sure that only one instance is created. If an instance that you want to create was already created, ThemeServices returns it.

See Also

TThemedElement (see page 4600)

TThemeServices (see page 4579)

42.2.3 Themes.UnthemedDesigner Function

Checks whether the VCL Forms designer supports theming for a control.

Pascal

```
function UnthemedDesigner(AControl: TControl): Boolean;
```

C++

```
Boolean UnthemedDesigner(TControl * AControl);
```

File

Themes

Description

Call UnthemedDesigner to check whether the VCL Forms designer supports theming for a control. `AControl` is the control that needs to be checked.

Note: UnthemedDesigner is used by the VCL Forms designer at design time. Applications should not call this routine directly.

See Also

ThemeControl (see page 4593)

ThemeServices (see page 4593)

TThemeServices (see page 4579)

42.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TThemedButton (see page 4595)	TThemedButton describes the possible themable parts of a button.
	TThemedClock (see page 4598)	TThemedClock describes the possible themable parts of a clock.
	TThemedComboBox (see page 4598)	TThemedComboBox describes the possible themable parts of a combo box.
	TThemedEdit (see page 4599)	TThemedEdit describes the possible themable parts of an edit.
	TThemedElement (see page 4600)	TThemedElement describes the possible themed user interface elements.
	TThemedExplorerBar (see page 4603)	TThemedExplorerBar describes the possible themable parts of an explorer bar control.
	TThemedHeader (see page 4604)	TThemedHeader describes the possible themable parts of a header control.
	TThemedListview (see page 4605)	TThemedListview describes the possible themable parts of a list view.
	TThemedMenu (see page 4606)	TThemedMenu describes the possible themable parts of a menu.
	TThemedPage (see page 4607)	TThemedPage describes the possible themable parts of a tab page.
	TThemedProgress (see page 4608)	TThemedProgress describes the possible themable parts of a progress bar.
	TThemedRebar (see page 4609)	TThemedRebar describes the possible themable parts of a rebar control.
	TThemedScrollBar (see page 4610)	TThemedScrollBar describes the possible themable parts of a scroll bar.
	TThemedSpin (see page 4612)	TThemedSpin describes the possible themable parts of a spin control.
	TThemedStartPanel (see page 4613)	TThemedStartPanel describes the possible themable parts of a start panel.
	TThemedStatus (see page 4614)	TThemedStatus describes the possible themable parts of a status bar.
	TThemedTab (see page 4615)	TThemedTab describes the possible themable parts of a tab.
	TThemedTaskBand (see page 4616)	TThemedTaskBand describes the possible themable parts of a task band.
	TThemedTaskBar (see page 4617)	TThemedTaskBar describes the possible themable parts of a system task bar.
	TThemedToolBar (see page 4618)	TThemedToolBar describes the possible themable parts of a tool bar.
	TThemedToolTip (see page 4620)	TThemedToolTip describes the possible themable parts of a tool tip.
	TThemedTrackBar (see page 4621)	TThemedTrackBar describes the possible themable parts of a track bar.
	TThemedTrayNotify (see page 4622)	TThemedTrayNotify describes the possible themable parts of a system tray notification area.
	TThemedTreeview (see page 4623)	TThemedTreeview describes the possible themable parts of a tree view.
	TThemedWindow (see page 4624)	TThemedWindow describes the possible themable parts of a window.

Records

	Name	Description
	TThemedElementDetails (see page 4602)	TThemedElementDetails describes a themed user interface element.

42.3.1 Themes.TThemedButton Enumeration

TThemedButton describes the possible themable parts of a button.

Pascal

```
TThemedButton = (
  tbButtonDontCare,
  tbButtonRoot,
  tbPushButtonNormal,
  tbPushButtonHot,
  tbPushButtonPressed,
  tbPushButtonDisabled,
```

```
    tbPushButtonDefaulted,
    tbRadioButtonUncheckedNormal,
    tbRadioButtonUncheckedHot,
    tbRadioButtonUncheckedPressed,
    tbRadioButtonUncheckedDisabled,
    tbRadioButtonCheckedNormal,
    tbRadioButtonCheckedHot,
    tbRadioButtonCheckedPressed,
    tbRadioButtonCheckedDisabled,
    tbCheckBoxUncheckedNormal,
    tbCheckBoxUncheckedHot,
    tbCheckBoxUncheckedPressed,
    tbCheckBoxUncheckedDisabled,
    tbCheckBoxCheckedNormal,
    tbCheckBoxCheckedHot,
    tbCheckBoxCheckedPressed,
    tbCheckBoxCheckedDisabled,
    tbCheckBoxMixedNormal,
    tbCheckBoxMixedHot,
    tbCheckBoxMixedPressed,
    tbCheckBoxMixedDisabled,
    tbGroupBoxNormal,
    tbGroupBoxDisabled,
    tbUserButton
  );
```

C++

```
  enum TThemedButton {
    tbButtonDontCare,
    tbButtonRoot,
    tbPushButtonNormal,
    tbPushButtonHot,
    tbPushButtonPressed,
    tbPushButtonDisabled,
    tbPushButtonDefaulted,
    tbRadioButtonUncheckedNormal,
    tbRadioButtonUncheckedHot,
    tbRadioButtonUncheckedPressed,
    tbRadioButtonUncheckedDisabled,
    tbRadioButtonCheckedNormal,
    tbRadioButtonCheckedHot,
    tbRadioButtonCheckedPressed,
    tbRadioButtonCheckedDisabled,
    tbCheckBoxUncheckedNormal,
    tbCheckBoxUncheckedHot,
    tbCheckBoxUncheckedPressed,
    tbCheckBoxUncheckedDisabled,
    tbCheckBoxCheckedNormal,
    tbCheckBoxCheckedHot,
    tbCheckBoxCheckedPressed,
    tbCheckBoxCheckedDisabled,
    tbCheckBoxMixedNormal,
    tbCheckBoxMixedHot,
    tbCheckBoxMixedPressed,
    tbCheckBoxMixedDisabled,
    tbGroupBoxNormal,
    tbGroupBoxDisabled,
    tbUserButton
  };
```

File

Themes

Description

TThemedButton describes the possible states of a themed button. The following table lists the possible values of

TThemedButton and their meaning.

Value	Description
tbButtonDontCare	The content that is shared across all the button parts.
tbButtonRoot	The content that is shared across all the button states.
tbPushButtonNormal	A push button in its normal state.
tbPushButtonHot	A push button in its hovered state.
tbPushButtonPressed	A push button in its pressed state.
tbPushButtonDisabled	A push button in its disabled state.
tbPushButtonDefaulted	A push button in its defaulted state.
tbRadioButtonUncheckedNormal	An unchecked radio button in its normal state.
tbRadioButtonUncheckedHot	An unchecked radio button in its hovered state.
tbRadioButtonUncheckedPressed	An unchecked radio button in its pressed state.
tbRadioButtonUncheckedDisabled	An unchecked radio button in its disabled state.
tbRadioButtonCheckedNormal	A checked radio button in its normal state.
tbRadioButtonCheckedHot	A checked radio button in its hovered state.
tbRadioButtonCheckedPressed	An unchecked radio button in its pressed state.
tbRadioButtonCheckedDisabled	An unchecked radio button in its disabled state.
tbCheckBoxUncheckedNormal	An unchecked check box in its normal state.
tbCheckBoxUncheckedHot	An unchecked check box in its hovered state.
tbCheckBoxUncheckedPressed	An unchecked check box in its pressed state.
tbCheckBoxUncheckedDisabled	An unchecked check box in its disabled state.
tbCheckBoxCheckedNormal	A checked check box in its normal state.
tbCheckBoxCheckedHot	A checked check box in its hovered state.
tbCheckBoxCheckedPressed	A checked check box in its pressed state.
tbCheckBoxCheckedDisabled	A checked check box in its disabled state.
tbCheckBoxMixedNormal	A mixed check box in its normal state.
tbCheckBoxMixedHot	A mixed check box in its hovered state.
tbCheckBoxMixedPressed	A mixed check box in its pressed state.
tbCheckBoxMixedDisabled	A mixed check box in its disabled state.
tbGroupBoxNormal	A group box in its normal state.
tbGroupBoxDisabled	A group box in its disabled state.
tbUserButton	An user-drawn button.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.2 Themes.TThemedClock Enumeration

TThemedClock describes the possible themable parts of a clock.

Pascal

```
TThemedClock = (
  tcClockDontCare,
  tcClockRoot,
  tcTimeNormal
);
```

C++

```
enum TThemedClock {
  tcClockDontCare,
  tcClockRoot,
  tcTimeNormal
};
```

File

Themes

Description

TThemedClock describes the possible themable parts of a clock. The following table lists the possible values of TThemedClock and their meaning.

Value	Description
tcClockDontCare	The content that is shared across all the clock parts.
tcClockRoot	The content that is shared across all the clock states.
tcTimeNormal	A clock in its normal state.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.3 Themes.TThemedComboBox Enumeration

TThemedComboBox describes the possible themable parts of a combo box.

Pascal

```
TThemedComboBox = (
  tcComboBoxDontCare,
  tcComboBoxRoot,
  tcDropDownButtonNormal,
  tcDropDownButtonHot,
  tcDropDownButtonPressed,
  tcDropDownButtonDisabled
);
```

C++

```
enum TThemedComboBox {
  tcComboBoxDontCare,
  tcComboBoxRoot,
  tcDropDownButtonNormal,
  tcDropDownButtonHot,
  tcDropDownButtonPressed,
  tcDropDownButtonDisabled
};
```

File

Themes

Description

TThemedComboBox describes the possible themable parts of a combo box. The following table lists the possible values of TThemedComboBox and their meaning.

Value	Description
tcComboBoxDontCare	The content that is shared across all the combo box parts.
tcComboBoxRoot	The content that is shared across all the combo box states.
tcDropDownButtonNormal	A combo box drop-down button in its normal state.
tcDropDownButtonHot	A combo box drop-down button in its hovered state.
tcDropDownButtonPressed	A combo box drop-down button in its pressed state.
tcDropDownButtonDisabled	A combo box drop-down button in its disabled state.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.4 Themes.TThemedEdit Enumeration

TThemedEdit describes the possible themable parts of an edit.

Pascal

```
TThemedEdit = (
  teEditDontCare,
  teEditRoot,
  teEditTextNormal,
  teEditTextHot,
  teEditTextSelected,
  teEditTextDisabled,
  teEditTextFocused,
  teEditTextReadOnly,
  teEditTextAssist,
  teEditCaret
);
```

C++

```
enum TThemedEdit {
  teEditDontCare,
```

```
    teEditRoot,
    teEditTextNormal,
    teEditTextHot,
    teEditTextSelected,
    teEditTextDisabled,
    teEditTextFocused,
    teEditTextReadOnly,
    teEditTextAssist,
    teEditCaret
  };
```

File

Themes

Description

TThemedEdit describes the possible themable parts of an edit. The following table lists the possible values of TThemedEdit and their meaning.

Value	Description
teEditDontCare	The content that is shared across all edit parts.
teEditRoot	The content that is shared across all edit states.
teEditTextNormal	The text in the edit control in its normal state.
teEditTextHot	The text in the edit control in its hovered state.
teEditTextSelected	The text in the edit control in its selected state.
teEditTextDisabled	The text in the edit control in disabled state.
teEditTextFocused	The text in the edit control in focused state.
teEditTextReadOnly	The text in the edit control in read-only state.
teEditTextAssist	The text in the edit control in its accessible state.
teEditCaret	The caret of an edit control.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.5 Themes.TThemedElement Enumeration

TThemedElement describes the possible themed user interface elements.

Pascal

```
  TThemedElement = (
    teButton,
    teClock,
    teComboBox,
    teEdit,
    teExplorerBar,
    teHeader,
    teListView,
    teMenu,
    tePage,
```

```
    teProgress,
    teRebar,
    teScrollBar,
    teSpin,
    teStartPanel,
    teStatus,
    teTab,
    teTaskBand,
    teTaskBar,
    teToolBar,
    teToolTip,
    teTrackBar,
    teTrayNotify,
    teTreeview,
    teWindow
  );
```

C++

```
  enum TThemedElement {
    teButton,
    teClock,
    teComboBox,
    teEdit,
    teExplorerBar,
    teHeader,
    teListView,
    teMenu,
    tePage,
    teProgress,
    teRebar,
    teScrollBar,
    teSpin,
    teStartPanel,
    teStatus,
    teTab,
    teTaskBand,
    teTaskBar,
    teToolBar,
    teToolTip,
    teTrackBar,
    teTrayNotify,
    teTreeview,
    teWindow
  };
```

File

Themes

Description

TThemedElement describes the possible themed user interface elements. The following table lists the possible values of TThemedElement and their meaning.

Value	Description
teButton	Represents a button.
teClock	Represents a clock.
teComboBox	Represents a combo box.
teEdit	Represents an edit box.
teExplorerBar	Represents an explorer bar.
teHeader	Represents a list header.

teListView	Represents a list view.
teMenu	Represents a menu.
tePage	Represents a tab or a property page.
teProgress	Represents a progress bar.
teRebar	Represents a rebar.
teScrollBar	Represents a scroll bar.
teSpin	Represents a spin button.
teStartPanel	Represents a start panel.
teStatus	Represents a status bar.
teTab	Represents a tab in a tab control.
teTaskBand	Represents a task band.
teTaskBar	Represents a task bar.
teToolBar	Represents a tool bar.
teToolTip	Represents a tool tip.
teTrackBar	Represents a track bar.
teTrayNotify	Represents a tray notify area.
teTreeview	Represents a tree view.
teWindow	Represents a window.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.6 Themes.TThemedElementDetails Record

TThemedElementDetails describes a themed user interface element.

Pascal

```
TThemedElementDetails = record
  Element: TThemedElement;
  Part: Integer;
  State: Integer;
end;
```

C++

```
struct TThemedElementDetails {
  TThemedElement Element;
  int Part;
  int State;
};
```

File

Themes

Description

TThemedElementDetails is a record that contains information useful when applying themes to a user interface element. The following table lists each member of TThemedElementDetails and its meaning.

Member	Description
`Element`	The type of the user interface element.
`Part`	The part of the control being themed.
`State`	The state of the user interface element.

See Also

TThemedElement (see page 4600)

ThemeServices (see page 4593)

TThemeData (see page 4629)

42.3.7 Themes.TThemedExplorerBar Enumeration

TThemedExplorerBar describes the possible themable parts of an explorer bar control.

Pascal

```
TThemedExplorerBar = (
  tebExplorerBarDontCare,
  tebExplorerBarRoot,
  tebHeaderBackgroundNormal,
  tebHeaderBackgroundHot,
  tebHeaderBackgroundPressed,
  tebHeaderCloseNormal,
  tebHeaderCloseHot,
  tebHeaderClosePressed,
  tebHeaderPinNormal,
  tebHeaderPinHot,
  tebHeaderPinPressed,
  tebHeaderPinSelectedNormal,
  tebHeaderPinSelectedHot,
  tebHeaderPinSelectedPressed,
  tebIEBarMenuNormal,
  tebIEBarMenuHot,
  tebIEBarMenuPressed,
  tebNormalGroupBackground,
  tebNormalGroupCollapseNormal,
  tebNormalGroupCollapseHot,
  tebNormalGroupCollapsePressed,
  tebNormalGroupExpandNormal,
  tebNormalGroupExpandHot,
  tebNormalGroupExpandPressed,
  tebNormalGroupHead,
  tebSpecialGroupBackground,
  tebSpecialGroupCollapseSpecial,
  tebSpecialGroupCollapseHot,
  tebSpecialGroupCollapsePressed,
  tebSpecialGroupExpandSpecial,
  tebSpecialGroupExpandHot,
  tebSpecialGroupExpandPressed,
  tebSpecialGroupHead
);
```

C++

```
enum TThemedExplorerBar {
  tebExplorerBarDontCare,
  tebExplorerBarRoot,
  tebHeaderBackgroundNormal,
  tebHeaderBackgroundHot,
  tebHeaderBackgroundPressed,
  tebHeaderCloseNormal,
  tebHeaderCloseHot,
  tebHeaderClosePressed,
  tebHeaderPinNormal,
  tebHeaderPinHot,
  tebHeaderPinPressed,
  tebHeaderPinSelectedNormal,
  tebHeaderPinSelectedHot,
  tebHeaderPinSelectedPressed,
  tebIEBarMenuNormal,
  tebIEBarMenuHot,
  tebIEBarMenuPressed,
  tebNormalGroupBackground,
  tebNormalGroupCollapseNormal,
  tebNormalGroupCollapseHot,
  tebNormalGroupCollapsePressed,
  tebNormalGroupExpandNormal,
  tebNormalGroupExpandHot,
  tebNormalGroupExpandPressed,
  tebNormalGroupHead,
  tebSpecialGroupBackground,
  tebSpecialGroupCollapseSpecial,
  tebSpecialGroupCollapseHot,
  tebSpecialGroupCollapsePressed,
  tebSpecialGroupExpandSpecial,
  tebSpecialGroupExpandHot,
  tebSpecialGroupExpandPressed,
  tebSpecialGroupHead
};
```

File

Themes

Description

TThemedExplorerBar describes the possible themable parts of an explorer bar control.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.8 Themes.TThemedHeader Enumeration

TThemedHeader describes the possible themable parts of a header control.

Pascal

```
TThemedHeader = (
  thHeaderDontCare,
  thHeaderRoot,
  thHeaderItemNormal,
```

```
    thHeaderItemHot,
    thHeaderItemPressed,
    thHeaderItemLeftNormal,
    thHeaderItemLeftHot,
    thHeaderItemLeftPressed,
    thHeaderItemRightNormal,
    thHeaderItemRightHot,
    thHeaderItemRightPressed,
    thHeaderSortArrowSortedUp,
    thHeaderSortArrowSortedDown
  );
```

C++

```
  enum TThemedHeader {
    thHeaderDontCare,
    thHeaderRoot,
    thHeaderItemNormal,
    thHeaderItemHot,
    thHeaderItemPressed,
    thHeaderItemLeftNormal,
    thHeaderItemLeftHot,
    thHeaderItemLeftPressed,
    thHeaderItemRightNormal,
    thHeaderItemRightHot,
    thHeaderItemRightPressed,
    thHeaderSortArrowSortedUp,
    thHeaderSortArrowSortedDown
  };
```

File

Themes

Description

TThemedHeader describes the possible themable parts of a header control.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.9 Themes.TThemedListview Enumeration

TThemedListview describes the possible themable parts of a list view.

Pascal

```
  TThemedListview = (
    tlListviewDontCare,
    tlListviewRoot,
    tlListItemNormal,
    tlListItemHot,
    tlListItemSelected,
    tlListItemDisabled,
    tlListItemSelectedNotFocus,
    tlListGroup,
    tlListDetail,
    tlListSortDetail,
    tlEmptyText
  );
```

C++

```
enum TThemedListview {
  tlListviewDontCare,
  tlListviewRoot,
  tlListItemNormal,
  tlListItemHot,
  tlListItemSelected,
  tlListItemDisabled,
  tlListItemSelectedNotFocus,
  tlListGroup,
  tlListDetail,
  tlListSortDetail,
  tlEmptyText
};
```

File

Themes

Description

TThemedListview describes the possible themable parts of a list view.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.10 Themes.TThemedMenu Enumeration

TThemedMenu describes the possible themable parts of a menu.

Pascal

```
TThemedMenu = (
  tmMenuDontCare,
  tmMenuRoot,
  tmMenuItemNormal,
  tmMenuItemSelected,
  tmMenuItemDemoted,
  tmMenuDropDown,
  tmMenuBarItem,
  tmMenuBarDropDown,
  tmChevron,
  tmSeparator
);
```

C++

```
enum TThemedMenu {
  tmMenuDontCare,
  tmMenuRoot,
  tmMenuItemNormal,
  tmMenuItemSelected,
  tmMenuItemDemoted,
  tmMenuDropDown,
  tmMenuBarItem,
  tmMenuBarDropDown,
  tmChevron,
  tmSeparator
```

```
};
```

File

Themes

Description

TThemedMenu describes the possible themable parts of a menu.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.11 Themes.TThemedPage Enumeration

TThemedPage describes the possible themable parts of a tab page.

Pascal

```
TThemedPage = (
  tpPageDontCare,
  tpPageRoot,
  tpUpNormal,
  tpUpHot,
  tpUpPressed,
  tpUpDisabled,
  tpDownNormal,
  tpDownHot,
  tpDownPressed,
  tpDownDisabled,
  tpUpHorzNormal,
  tpUpHorzHot,
  tpUpHorzPressed,
  tpUpHorzDisabled,
  tpDownHorzNormal,
  tpDownHorzHot,
  tpDownHorzPressed,
  tpDownHorzDisabled
);
```

C++

```
enum TThemedPage {
  tpPageDontCare,
  tpPageRoot,
  tpUpNormal,
  tpUpHot,
  tpUpPressed,
  tpUpDisabled,
  tpDownNormal,
  tpDownHot,
  tpDownPressed,
  tpDownDisabled,
  tpUpHorzNormal,
  tpUpHorzHot,
  tpUpHorzPressed,
  tpUpHorzDisabled,
  tpDownHorzNormal,
  tpDownHorzHot,
  tpDownHorzPressed,
```

```
  tpDownHorzDisabled
};
```

File

Themes

Description

TThemedPage describes the possible themable parts of a tab page.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.12 Themes.TThemedProgress Enumeration

TThemedProgress describes the possible themable parts of a progress bar.

Pascal

```
TThemedProgress = (
  tpProgressDontCare,
  tpProgressRoot,
  tpBar,
  tpBarVert,
  tpChunk,
  tpChunkVert
);
```

C++

```
enum TThemedProgress {
  tpProgressDontCare,
  tpProgressRoot,
  tpBar,
  tpBarVert,
  tpChunk,
  tpChunkVert
};
```

File

Themes

Description

TThemedProgress describes the possible themable parts of a progress bar. The following table lists the possible values of TThemedProgress and their meaning.

Value	Description
tpProgressDontCare	The content that is shared across all the progress bar parts.
tpProgressRoot	The content that is shared across all the progress bar states.
tpBar	The progress bar itself in horizontal orientation.
tpBarVert	The progress bar itself in vertical orientation.
tpChunk	A horizontal progress chunk.

tpChunkVert	A vertical progress chunk.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.13 Themes.TThemedRebar Enumeration

TThemedRebar describes the possible themable parts of a rebar control.

Pascal

```
TThemedRebar = (
  trRebarDontCare,
  trRebarRoot,
  trGripper,
  trGripperVert,
  trBandNormal,
  trBandHot,
  trBandPressed,
  trBandDisabled,
  trBandChecked,
  trBandHotChecked,
  trChevronNormal,
  trChevronHot,
  trChevronPressed,
  trChevronDisabled,
  trChevronVertNormal,
  trChevronVertHot,
  trChevronVertPressed,
  trChevronVertDisabled
);
```

C++

```
enum TThemedRebar {
  trRebarDontCare,
  trRebarRoot,
  trGripper,
  trGripperVert,
  trBandNormal,
  trBandHot,
  trBandPressed,
  trBandDisabled,
  trBandChecked,
  trBandHotChecked,
  trChevronNormal,
  trChevronHot,
  trChevronPressed,
  trChevronDisabled,
  trChevronVertNormal,
  trChevronVertHot,
  trChevronVertPressed,
  trChevronVertDisabled
};
```

File

Themes

Description

TThemedRebar describes the possible themable parts of a rebar control.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.14 Themes.TThemedScrollBar Enumeration

TThemedScrollBar describes the possible themable parts of a scroll bar.

Pascal

```
TThemedScrollBar = (
  tsScrollBarDontCare,
  tsScrollBarRoot,
  tsArrowBtnUpNormal,
  tsArrowBtnUpHot,
  tsArrowBtnUpPressed,
  tsArrowBtnUpDisabled,
  tsArrowBtnDownNormal,
  tsArrowBtnDownHot,
  tsArrowBtnDownPressed,
  tsArrowBtnDownDisabled,
  tsArrowBtnLeftNormal,
  tsArrowBtnLeftHot,
  tsArrowBtnLeftPressed,
  tsArrowBtnLeftDisabled,
  tsArrowBtnRightNormal,
  tsArrowBtnRightHot,
  tsArrowBtnRightPressed,
  tsArrowBtnRightDisabled,
  tsThumbBtnHorzNormal,
  tsThumbBtnHorzHot,
  tsThumbBtnHorzPressed,
  tsThumbBtnHorzDisabled,
  tsThumbBtnVertNormal,
  tsThumbBtnVertHot,
  tsThumbBtnVertPressed,
  tsThumbBtnVertDisabled,
  tsLowerTrackHorzNormal,
  tsLowerTrackHorzHot,
  tsLowerTrackHorzPressed,
  tsLowerTrackHorzDisabled,
  tsUpperTrackHorzNormal,
  tsUpperTrackHorzHot,
  tsUpperTrackHorzPressed,
  tsUpperTrackHorzDisabled,
  tsLowerTrackVertNormal,
  tsLowerTrackVertHot,
  tsLowerTrackVertPressed,
  tsLowerTrackVertDisabled,
  tsUpperTrackVertNormal,
  tsUpperTrackVertHot,
  tsUpperTrackVertPressed,
  tsUpperTrackVertDisabled,
  tsGripperHorzNormal,
  tsGripperHorzHot,
```

```
    tsGripperHorzPressed,
    tsGripperHorzDisabled,
    tsGripperVertNormal,
    tsGripperVertHot,
    tsGripperVertPressed,
    tsGripperVertDisabled,
    tsSizeBoxRightAlign,
    tsSizeBoxLeftAlign
  );
```

C++

```
  enum TThemedScrollBar {
    tsScrollBarDontCare,
    tsScrollBarRoot,
    tsArrowBtnUpNormal,
    tsArrowBtnUpHot,
    tsArrowBtnUpPressed,
    tsArrowBtnUpDisabled,
    tsArrowBtnDownNormal,
    tsArrowBtnDownHot,
    tsArrowBtnDownPressed,
    tsArrowBtnDownDisabled,
    tsArrowBtnLeftNormal,
    tsArrowBtnLeftHot,
    tsArrowBtnLeftPressed,
    tsArrowBtnLeftDisabled,
    tsArrowBtnRightNormal,
    tsArrowBtnRightHot,
    tsArrowBtnRightPressed,
    tsArrowBtnRightDisabled,
    tsThumbBtnHorzNormal,
    tsThumbBtnHorzHot,
    tsThumbBtnHorzPressed,
    tsThumbBtnHorzDisabled,
    tsThumbBtnVertNormal,
    tsThumbBtnVertHot,
    tsThumbBtnVertPressed,
    tsThumbBtnVertDisabled,
    tsLowerTrackHorzNormal,
    tsLowerTrackHorzHot,
    tsLowerTrackHorzPressed,
    tsLowerTrackHorzDisabled,
    tsUpperTrackHorzNormal,
    tsUpperTrackHorzHot,
    tsUpperTrackHorzPressed,
    tsUpperTrackHorzDisabled,
    tsLowerTrackVertNormal,
    tsLowerTrackVertHot,
    tsLowerTrackVertPressed,
    tsLowerTrackVertDisabled,
    tsUpperTrackVertNormal,
    tsUpperTrackVertHot,
    tsUpperTrackVertPressed,
    tsUpperTrackVertDisabled,
    tsGripperHorzNormal,
    tsGripperHorzHot,
    tsGripperHorzPressed,
    tsGripperHorzDisabled,
    tsGripperVertNormal,
    tsGripperVertHot,
    tsGripperVertPressed,
    tsGripperVertDisabled,
    tsSizeBoxRightAlign,
    tsSizeBoxLeftAlign
  };
```

File

Themes

Description

TThemedScrollBar describes the possible themable parts of a scroll bar.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.15 Themes.TThemedSpin Enumeration

TThemedSpin describes the possible themable parts of a spin control.

Pascal

```
TThemedSpin = (
  tsSpinDontCare,
  tsSpinRoot,
  tsUpNormal,
  tsUpHot,
  tsUpPressed,
  tsUpDisabled,
  tsDownNormal,
  tsDownHot,
  tsDownPressed,
  tsDownDisabled,
  tsUpHorzNormal,
  tsUpHorzHot,
  tsUpHorzPressed,
  tsUpHorzDisabled,
  tsDownHorzNormal,
  tsDownHorzHot,
  tsDownHorzPressed,
  tsDownHorzDisabled
);
```

C++

```
enum TThemedSpin {
  tsSpinDontCare,
  tsSpinRoot,
  tsUpNormal,
  tsUpHot,
  tsUpPressed,
  tsUpDisabled,
  tsDownNormal,
  tsDownHot,
  tsDownPressed,
  tsDownDisabled,
  tsUpHorzNormal,
  tsUpHorzHot,
  tsUpHorzPressed,
  tsUpHorzDisabled,
  tsDownHorzNormal,
  tsDownHorzHot,
  tsDownHorzPressed,
  tsDownHorzDisabled
```

```
};
```

File

Themes

Description

TThemedSpin describes the possible themable parts of a spin control.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.16 Themes.TThemedStartPanel Enumeration

TThemedStartPanel describes the possible themable parts of a start panel.

Pascal

```
TThemedStartPanel = (
  tspStartPanelDontCare,
  tspStartPanelRoot,
  tspUserPane,
  tspMorePrograms,
  tspMoreProgramsArrowNormal,
  tspMoreProgramsArrowHot,
  tspMoreProgramsArrowPressed,
  tspProgList,
  tspProgListSeparator,
  tspPlacesList,
  tspPlacesListSeparator,
  tspLogOff,
  tspLogOffButtonsNormal,
  tspLogOffButtonsHot,
  tspLogOffButtonsPressed,
  tspUserPicture,
  tspPreview
);
```

C++

```
enum TThemedStartPanel {
  tspStartPanelDontCare,
  tspStartPanelRoot,
  tspUserPane,
  tspMorePrograms,
  tspMoreProgramsArrowNormal,
  tspMoreProgramsArrowHot,
  tspMoreProgramsArrowPressed,
  tspProgList,
  tspProgListSeparator,
  tspPlacesList,
  tspPlacesListSeparator,
  tspLogOff,
  tspLogOffButtonsNormal,
  tspLogOffButtonsHot,
  tspLogOffButtonsPressed,
  tspUserPicture,
  tspPreview
};
```

File

Themes

Description

TThemedStartPanel describes the possible themable parts of a start panel.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.17 Themes.TThemedStatus Enumeration

TThemedStatus describes the possible themable parts of a status bar.

Pascal

```
TThemedStatus = (
  tsStatusDontCare,
  tsStatusRoot,
  tsPane,
  tsGripperPane,
  tsGripper
);
```

C++

```
enum TThemedStatus {
  tsStatusDontCare,
  tsStatusRoot,
  tsPane,
  tsGripperPane,
  tsGripper
};
```

File

Themes

Description

TThemedStatus describes the possible themable parts of a status bar. The following table lists the possible values of TThemedStatus and their meaning.

Value	Description
tsStatusDontCare	The content that is shared across all the status bar parts.
tsStatusRoot	The content that is shared across all the status bar states.
tsPane	The pane on the status bar.
tsGripperPane	The gripper pane on the status bar.
tsGripper	The gripper located on the right side of the status bar.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.18 Themes.TThemedTab Enumeration

TThemedTab describes the possible themable parts of a tab.

Pascal

```
TThemedTab = (
  ttTabDontCare,
  ttTabRoot,
  ttTabItemNormal,
  ttTabItemHot,
  ttTabItemSelected,
  ttTabItemDisabled,
  ttTabItemFocused,
  ttTabItemLeftEdgeNormal,
  ttTabItemLeftEdgeHot,
  ttTabItemLeftEdgeSelected,
  ttTabItemLeftEdgeDisabled,
  ttTabItemLeftEdgeFocused,
  ttTabItemRightEdgeNormal,
  ttTabItemRightEdgeHot,
  ttTabItemRightEdgeSelected,
  ttTabItemRightEdgeDisabled,
  ttTabItemRightEdgeFocused,
  ttTabItemBothEdgeNormal,
  ttTabItemBothEdgeHot,
  ttTabItemBothEdgeSelected,
  ttTabItemBothEdgeDisabled,
  ttTabItemBothEdgeFocused,
  ttTopTabItemNormal,
  ttTopTabItemHot,
  ttTopTabItemSelected,
  ttTopTabItemDisabled,
  ttTopTabItemFocused,
  ttTopTabItemLeftEdgeNormal,
  ttTopTabItemLeftEdgeHot,
  ttTopTabItemLeftEdgeSelected,
  ttTopTabItemLeftEdgeDisabled,
  ttTopTabItemLeftEdgeFocused,
  ttTopTabItemRightEdgeNormal,
  ttTopTabItemRightEdgeHot,
  ttTopTabItemRightEdgeSelected,
  ttTopTabItemRightEdgeDisabled,
  ttTopTabItemRightEdgeFocused,
  ttTopTabItemBothEdgeNormal,
  ttTopTabItemBothEdgeHot,
  ttTopTabItemBothEdgeSelected,
  ttTopTabItemBothEdgeDisabled,
  ttTopTabItemBothEdgeFocused,
  ttPane,
  ttBody
);
```

C++

```
enum TThemedTab {
  ttTabDontCare,
  ttTabRoot,
  ttTabItemNormal,
```

```
    ttTabItemHot,
    ttTabItemSelected,
    ttTabItemDisabled,
    ttTabItemFocused,
    ttTabItemLeftEdgeNormal,
    ttTabItemLeftEdgeHot,
    ttTabItemLeftEdgeSelected,
    ttTabItemLeftEdgeDisabled,
    ttTabItemLeftEdgeFocused,
    ttTabItemRightEdgeNormal,
    ttTabItemRightEdgeHot,
    ttTabItemRightEdgeSelected,
    ttTabItemRightEdgeDisabled,
    ttTabItemRightEdgeFocused,
    ttTabItemBothEdgeNormal,
    ttTabItemBothEdgeHot,
    ttTabItemBothEdgeSelected,
    ttTabItemBothEdgeDisabled,
    ttTabItemBothEdgeFocused,
    ttTopTabItemNormal,
    ttTopTabItemHot,
    ttTopTabItemSelected,
    ttTopTabItemDisabled,
    ttTopTabItemFocused,
    ttTopTabItemLeftEdgeNormal,
    ttTopTabItemLeftEdgeHot,
    ttTopTabItemLeftEdgeSelected,
    ttTopTabItemLeftEdgeDisabled,
    ttTopTabItemLeftEdgeFocused,
    ttTopTabItemRightEdgeNormal,
    ttTopTabItemRightEdgeHot,
    ttTopTabItemRightEdgeSelected,
    ttTopTabItemRightEdgeDisabled,
    ttTopTabItemRightEdgeFocused,
    ttTopTabItemBothEdgeNormal,
    ttTopTabItemBothEdgeHot,
    ttTopTabItemBothEdgeSelected,
    ttTopTabItemBothEdgeDisabled,
    ttTopTabItemBothEdgeFocused,
    ttPane,
    ttBody
  };
```

File

Themes

Description

TThemedTab describes the possible themable parts of a tab.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.19 Themes.TThemedTaskBand Enumeration

TThemedTaskBand describes the possible themable parts of a task band.

Pascal

```
TThemedTaskBand = (
  ttbTaskBandDontCare,
  ttbTaskBandRoot,
  ttbGroupCount,
  ttbFlashButton,
  ttpFlashButtonGroupMenu
);
```

C++

```
enum TThemedTaskBand {
  ttbTaskBandDontCare,
  ttbTaskBandRoot,
  ttbGroupCount,
  ttbFlashButton,
  ttpFlashButtonGroupMenu
};
```

File

Themes

Description

TThemedTaskBand describes the possible themable parts of a task band. The following table lists the possible values of TThemedTaskBand and their meaning.

Value	Description
ttbTaskBandDontCare	The content that is shared across all the task band parts.
ttbTaskBandRoot	The content that is shared across all the task band states.
ttbGroupCount	The group count part of a task band.
ttbFlashButton	The flashing button of a task band.
ttpFlashButtonGroupMenu	The flashing button's menu on a task band.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.20 Themes.TThemedTaskBar Enumeration

TThemedTaskBar describes the possible themable parts of a system task bar.

Pascal

```
TThemedTaskBar = (
  ttTaskBarDontCare,
  ttTaskBarRoot,
  ttbTimeNormal
);
```

C++

```
enum TThemedTaskBar {
  ttTaskBarDontCare,
```

```
    ttTaskBarRoot,
    ttbTimeNormal
  };
```

File

Themes

Description

TThemedTaskBar describes the possible themable parts of a system task bar. The following table lists the possible values of TThemedTaskBar and their meaning.

Value	Description
ttTaskBarDontCare	The content that is shared across all the task bar parts.
ttTaskBarRoot	The content that is shared across all the task bar states.
ttbTimeNormal	The task bar in its normal state.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.21 Themes.TThemedToolBar Enumeration

TThemedToolBar describes the possible themable parts of a tool bar.

Pascal

```
TThemedToolBar = (
  ttbToolBarDontCare,
  ttbToolBarRoot,
  ttbButtonNormal,
  ttbButtonHot,
  ttbButtonPressed,
  ttbButtonDisabled,
  ttbButtonChecked,
  ttbButtonCheckedHot,
  ttbDropDownButtonNormal,
  ttbDropDownButtonHot,
  ttbDropDownButtonPressed,
  ttbDropDownButtonDisabled,
  ttbDropDownButtonChecked,
  ttbDropDownButtonCheckedHot,
  ttbSplitButtonNormal,
  ttbSplitButtonHot,
  ttbSplitButtonPressed,
  ttbSplitButtonDisabled,
  ttbSplitButtonChecked,
  ttbSplitButtonCheckedHot,
  ttbSplitButtonDropDownNormal,
  ttbSplitButtonDropDownHot,
  ttbSplitButtonDropDownPressed,
  ttbSplitButtonDropDownDisabled,
  ttbSplitButtonDropDownChecked,
  ttbSplitButtonDropDownCheckedHot,
  ttbSeparatorNormal,
  ttbSeparatorHot,
```

```
    ttbSeparatorPressed,
    ttbSeparatorDisabled,
    ttbSeparatorChecked,
    ttbSeparatorCheckedHot,
    ttbSeparatorVertNormal,
    ttbSeparatorVertHot,
    ttbSeparatorVertPressed,
    ttbSeparatorVertDisabled,
    ttbSeparatorVertChecked,
    ttbSeparatorVertCheckedHot
  );
```

C++

```
  enum TThemedToolBar {
    ttbToolBarDontCare,
    ttbToolBarRoot,
    ttbButtonNormal,
    ttbButtonHot,
    ttbButtonPressed,
    ttbButtonDisabled,
    ttbButtonChecked,
    ttbButtonCheckedHot,
    ttbDropDownButtonNormal,
    ttbDropDownButtonHot,
    ttbDropDownButtonPressed,
    ttbDropDownButtonDisabled,
    ttbDropDownButtonChecked,
    ttbDropDownButtonCheckedHot,
    ttbSplitButtonNormal,
    ttbSplitButtonHot,
    ttbSplitButtonPressed,
    ttbSplitButtonDisabled,
    ttbSplitButtonChecked,
    ttbSplitButtonCheckedHot,
    ttbSplitButtonDropDownNormal,
    ttbSplitButtonDropDownHot,
    ttbSplitButtonDropDownPressed,
    ttbSplitButtonDropDownDisabled,
    ttbSplitButtonDropDownChecked,
    ttbSplitButtonDropDownCheckedHot,
    ttbSeparatorNormal,
    ttbSeparatorHot,
    ttbSeparatorPressed,
    ttbSeparatorDisabled,
    ttbSeparatorChecked,
    ttbSeparatorCheckedHot,
    ttbSeparatorVertNormal,
    ttbSeparatorVertHot,
    ttbSeparatorVertPressed,
    ttbSeparatorVertDisabled,
    ttbSeparatorVertChecked,
    ttbSeparatorVertCheckedHot
  };
```

File

Themes

Description

TThemedToolBar describes the possible themable parts of a tool bar.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.22 Themes.TThemedToolTip Enumeration

TThemedToolTip describes the possible themable parts of a tool tip.

Pascal

```
TThemedToolTip = (
  tttToolTipDontCare,
  tttToolTipRoot,
  tttStandardNormal,
  tttStandardLink,
  tttStandardTitleNormal,
  tttStandardTitleLink,
  tttBaloonNormal,
  tttBaloonLink,
  tttBaloonTitleNormal,
  tttBaloonTitleLink,
  tttBaloonStemUpLeft,
  tttBaloonStemDownLeft,
  tttCloseNormal,
  tttCloseHot,
  tttClosePressed
);
```

C++

```
enum TThemedToolTip {
  tttToolTipDontCare,
  tttToolTipRoot,
  tttStandardNormal,
  tttStandardLink,
  tttStandardTitleNormal,
  tttStandardTitleLink,
  tttBaloonNormal,
  tttBaloonLink,
  tttBaloonTitleNormal,
  tttBaloonTitleLink,
  tttBaloonStemUpLeft,
  tttBaloonStemDownLeft,
  tttCloseNormal,
  tttCloseHot,
  tttClosePressed
};
```

File

Themes

Description

TThemedToolTip describes the possible themable parts of a tool tip.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.23 Themes.TThemedTrackBar Enumeration

TThemedTrackBar describes the possible themable parts of a track bar.

Pascal

```
  TThemedTrackBar = (
    ttbTrackBarDontCare,
    ttbTrackBarRoot,
    ttbTrack,
    ttbTrackVert,
    ttbThumbNormal,
    ttbThumbHot,
    ttbThumbPressed,
    ttbThumbFocused,
    ttbThumbDisabled,
    ttbThumbBottomNormal,
    ttbThumbBottomHot,
    ttbThumbBottomPressed,
    ttbThumbBottomFocused,
    ttbThumbBottomDisabled,
    ttbThumbTopNormal,
    ttbThumbTopHot,
    ttbThumbTopPressed,
    ttbThumbTopFocused,
    ttbThumbTopDisabled,
    ttbThumbVertNormal,
    ttbThumbVertHot,
    ttbThumbVertPressed,
    ttbThumbVertFocused,
    ttbThumbVertDisabled,
    ttbThumbLeftNormal,
    ttbThumbLeftHot,
    ttbThumbLeftPressed,
    ttbThumbLeftFocused,
    ttbThumbLeftDisabled,
    ttbThumbRightNormal,
    ttbThumbRightHot,
    ttbThumbRightPressed,
    ttbThumbRightFocused,
    ttbThumbRightDisabled,
    ttbThumbTics,
    ttbThumbTicsVert
  );
```

C++

```
  enum TThemedTrackBar {
    ttbTrackBarDontCare,
    ttbTrackBarRoot,
    ttbTrack,
    ttbTrackVert,
    ttbThumbNormal,
    ttbThumbHot,
    ttbThumbPressed,
    ttbThumbFocused,
    ttbThumbDisabled,
    ttbThumbBottomNormal,
    ttbThumbBottomHot,
    ttbThumbBottomPressed,
    ttbThumbBottomFocused,
    ttbThumbBottomDisabled,
    ttbThumbTopNormal,
```

```
    ttbThumbTopHot,
    ttbThumbTopPressed,
    ttbThumbTopFocused,
    ttbThumbTopDisabled,
    ttbThumbVertNormal,
    ttbThumbVertHot,
    ttbThumbVertPressed,
    ttbThumbVertFocused,
    ttbThumbVertDisabled,
    ttbThumbLeftNormal,
    ttbThumbLeftHot,
    ttbThumbLeftPressed,
    ttbThumbLeftFocused,
    ttbThumbLeftDisabled,
    ttbThumbRightNormal,
    ttbThumbRightHot,
    ttbThumbRightPressed,
    ttbThumbRightFocused,
    ttbThumbRightDisabled,
    ttbThumbTics,
    ttbThumbTicsVert
  };
```

File

Themes

Description

TThemedTrackBar describes the possible themable parts of a track bar.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.24 Themes.TThemedTrayNotify Enumeration

TThemedTrayNotify describes the possible themable parts of a system tray notification area.

Pascal

```
  TThemedTrayNotify = (
    ttnTrayNotifyDontCare,
    ttnTrayNotifyRoot,
    ttnBackground,
    ttnAnimBackground
  );
```

C++

```
  enum TThemedTrayNotify {
    ttnTrayNotifyDontCare,
    ttnTrayNotifyRoot,
    ttnBackground,
    ttnAnimBackground
  };
```

File

Themes

Description

TThemedTrayNotify describes the possible themable parts of a system tray notification area. The following table lists the possible values of TThemedTrayNotify and their meaning.

Value	Description
`ttnTrayNotifyDontCare`	The content that is shared across all the tray notification parts.
`ttnTrayNotifyRoot`	The content that is shared across all the tray notification states.
`ttnBackground`	The background of a tray notification area.
`ttnAnimBackground`	The background of a tray notification area when animating.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.25 Themes.TThemedTreeview Enumeration

TThemedTreeview describes the possible themable parts of a tree view.

Pascal

```
TThemedTreeview = (
  ttTreeviewDontCare,
  ttTreeviewRoot,
  ttItemNormal,
  ttItemHot,
  ttItemSelected,
  ttItemDisabled,
  ttItemSelectedNotFocus,
  ttGlyphClosed,
  ttGlyphOpened,
  ttBranch
);
```

C++

```
enum TThemedTreeview {
  ttTreeviewDontCare,
  ttTreeviewRoot,
  ttItemNormal,
  ttItemHot,
  ttItemSelected,
  ttItemDisabled,
  ttItemSelectedNotFocus,
  ttGlyphClosed,
  ttGlyphOpened,
  ttBranch
};
```

File

Themes

Description

TThemedTreeview describes the possible themable parts of a tree view. The following table lists the possible values of

TThemedTreeview and their meaning.

Value	Description
ttTreeviewDontCare	The content that is shared across all the tree view parts.
ttTreeviewRoot	The content that is shared across all the tree view states.
ttItemNormal	A tree view item in its normal state.
ttItemHot	A tree view item in its hovered state.
ttItemSelected	A tree view item in its selected state.
ttItemDisabled	A tree view item in its disabled state.
ttItemSelectedNotFocus	A tree view item in its selected, but not focused state.
ttGlyphClosed	The glyph representing a collapsed tree view node.
ttGlyphOpened	The glyph representing an expanded tree view node.
ttBranch	Tree view branch that links all nodes together.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.3.26 Themes.TThemedWindow Enumeration

TThemedWindow describes the possible themable parts of a window.

Pascal

```
TThemedWindow = (
  twWindowDontCare,
  twWindowRoot,
  twCaptionActive,
  twCaptionInactive,
  twCaptionDisabled,
  twSmallCaptionActive,
  twSmallCaptionInactive,
  twSmallCaptionDisabled,
  twMinCaptionActive,
  twMinCaptionInactive,
  twMinCaptionDisabled,
  twSmallMinCaptionActive,
  twSmallMinCaptionInactive,
  twSmallMinCaptionDisabled,
  twMaxCaptionActive,
  twMaxCaptionInactive,
  twMaxCaptionDisabled,
  twSmallMaxCaptionActive,
  twSmallMaxCaptionInactive,
  twSmallMaxCaptionDisabled,
  twFrameLeftActive,
  twFrameLeftInactive,
  twFrameRightActive,
  twFrameRightInactive,
  twFrameBottomActive,
  twFrameBottomInactive,
```

```
    twSmallFrameLeftActive,
    twSmallFrameLeftInactive,
    twSmallFrameRightActive,
    twSmallFrameRightInactive,
    twSmallFrameBottomActive,
    twSmallFrameBottomInactive,
    twSysButtonNormal,
    twSysButtonHot,
    twSysButtonPushed,
    twSysButtonDisabled,
    twMDISysButtonNormal,
    twMDISysButtonHot,
    twMDISysButtonPushed,
    twMDISysButtonDisabled,
    twMinButtonNormal,
    twMinButtonHot,
    twMinButtonPushed,
    twMinButtonDisabled,
    twMDIMinButtonNormal,
    twMDIMinButtonHot,
    twMDIMinButtonPushed,
    twMDIMinButtonDisabled,
    twMaxButtonNormal,
    twMaxButtonHot,
    twMaxButtonPushed,
    twMaxButtonDisabled,
    twCloseButtonNormal,
    twCloseButtonHot,
    twCloseButtonPushed,
    twCloseButtonDisabled,
    twSmallCloseButtonNormal,
    twSmallCloseButtonHot,
    twSmallCloseButtonPushed,
    twSmallCloseButtonDisabled,
    twMDICloseButtonNormal,
    twMDICloseButtonHot,
    twMDICloseButtonPushed,
    twMDICloseButtonDisabled,
    twRestoreButtonNormal,
    twRestoreButtonHot,
    twRestoreButtonPushed,
    twRestoreButtonDisabled,
    twMDIRestoreButtonNormal,
    twMDIRestoreButtonHot,
    twMDIRestoreButtonPushed,
    twMDIRestoreButtonDisabled,
    twHelpButtonNormal,
    twHelpButtonHot,
    twHelpButtonPushed,
    twHelpButtonDisabled,
    twMDIHelpButtonNormal,
    twMDIHelpButtonHot,
    twMDIHelpButtonPushed,
    twMDIHelpButtonDisabled,
    twHorzScrollNormal,
    twHorzScrollHot,
    twHorzScrollPushed,
    twHorzScrollDisabled,
    twHorzThumbNormal,
    twHorzThumbHot,
    twHorzThumbPushed,
    twHorzThumbDisabled,
    twVertScrollNormal,
    twVertScrollHot,
    twVertScrollPushed,
    twVertScrollDisabled,
```

```
    twVertThumbNormal,
    twVertThumbHot,
    twVertThumbPushed,
    twVertThumbDisabled,
    twDialog,
    twCaptionSizingTemplate,
    twSmallCaptionSizingTemplate,
    twFrameLeftSizingTemplate,
    twSmallFrameLeftSizingTemplate,
    twFrameRightSizingTemplate,
    twSmallFrameRightSizingTemplate,
    twFrameBottomSizingTemplate,
    twSmallFrameBottomSizingTemplate
  );
```

C++

```
  enum TThemedWindow {
    twWindowDontCare,
    twWindowRoot,
    twCaptionActive,
    twCaptionInactive,
    twCaptionDisabled,
    twSmallCaptionActive,
    twSmallCaptionInactive,
    twSmallCaptionDisabled,
    twMinCaptionActive,
    twMinCaptionInactive,
    twMinCaptionDisabled,
    twSmallMinCaptionActive,
    twSmallMinCaptionInactive,
    twSmallMinCaptionDisabled,
    twMaxCaptionActive,
    twMaxCaptionInactive,
    twMaxCaptionDisabled,
    twSmallMaxCaptionActive,
    twSmallMaxCaptionInactive,
    twSmallMaxCaptionDisabled,
    twFrameLeftActive,
    twFrameLeftInactive,
    twFrameRightActive,
    twFrameRightInactive,
    twFrameBottomActive,
    twFrameBottomInactive,
    twSmallFrameLeftActive,
    twSmallFrameLeftInactive,
    twSmallFrameRightActive,
    twSmallFrameRightInactive,
    twSmallFrameBottomActive,
    twSmallFrameBottomInactive,
    twSysButtonNormal,
    twSysButtonHot,
    twSysButtonPushed,
    twSysButtonDisabled,
    twMDISysButtonNormal,
    twMDISysButtonHot,
    twMDISysButtonPushed,
    twMDISysButtonDisabled,
    twMinButtonNormal,
    twMinButtonHot,
    twMinButtonPushed,
    twMinButtonDisabled,
    twMDIMinButtonNormal,
    twMDIMinButtonHot,
    twMDIMinButtonPushed,
    twMDIMinButtonDisabled,
    twMaxButtonNormal,
```

```
    twMaxButtonHot,
    twMaxButtonPushed,
    twMaxButtonDisabled,
    twCloseButtonNormal,
    twCloseButtonHot,
    twCloseButtonPushed,
    twCloseButtonDisabled,
    twSmallCloseButtonNormal,
    twSmallCloseButtonHot,
    twSmallCloseButtonPushed,
    twSmallCloseButtonDisabled,
    twMDICloseButtonNormal,
    twMDICloseButtonHot,
    twMDICloseButtonPushed,
    twMDICloseButtonDisabled,
    twRestoreButtonNormal,
    twRestoreButtonHot,
    twRestoreButtonPushed,
    twRestoreButtonDisabled,
    twMDIRestoreButtonNormal,
    twMDIRestoreButtonHot,
    twMDIRestoreButtonPushed,
    twMDIRestoreButtonDisabled,
    twHelpButtonNormal,
    twHelpButtonHot,
    twHelpButtonPushed,
    twHelpButtonDisabled,
    twMDIHelpButtonNormal,
    twMDIHelpButtonHot,
    twMDIHelpButtonPushed,
    twMDIHelpButtonDisabled,
    twHorzScrollNormal,
    twHorzScrollHot,
    twHorzScrollPushed,
    twHorzScrollDisabled,
    twHorzThumbNormal,
    twHorzThumbHot,
    twHorzThumbPushed,
    twHorzThumbDisabled,
    twVertScrollNormal,
    twVertScrollHot,
    twVertScrollPushed,
    twVertScrollDisabled,
    twVertThumbNormal,
    twVertThumbHot,
    twVertThumbPushed,
    twVertThumbDisabled,
    twDialog,
    twCaptionSizingTemplate,
    twSmallCaptionSizingTemplate,
    twFrameLeftSizingTemplate,
    twSmallFrameLeftSizingTemplate,
    twFrameRightSizingTemplate,
    twSmallFrameRightSizingTemplate,
    twFrameBottomSizingTemplate,
    twSmallFrameBottomSizingTemplate
  };
```

File

Themes

Description

TThemedWindow describes the possible themable parts of a window.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.4 Types

The following table lists types in this documentation.

Types

Name	Description
PThemedElementDetails (see page 4629)	PThemedElementDetails is pointer to a TThemedElementDetails (see page 4602) record.
TThemeData (see page 4629)	TThemeData is an array of theme handles.
TThemeServicesClass (see page 4630)	TThemeServicesClass defines the metaclass for TThemeServices (see page 4579).

42.4.1 Themes.PThemedElementDetails Type

PThemedElementDetails is pointer to a TThemedElementDetails (see page 4602) record.

Pascal

```
PThemedElementDetails = ^TThemedElementDetails;
```

C++

```
^TThemedElementDetails PThemedElementDetails;
```

File

Themes

Description

PThemedElementDetails is pointer to a TThemedElementDetails (see page 4602) record. PThemedElementDetails variables are used by the TThemeServices (see page 4579) class to store theme related information.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

TThemeData (see page 4629)

42.4.2 Themes.TThemeData Type

TThemeData is an array of theme handles.

Pascal

```
TThemeData = array[TThemedElement] of HTHEME;
```

C++

```
array[TThemedElement] of HTHEME TThemeData;
```

File

Themes

Description

TThemeData is an array used internally to hold theme element handles. Each element of TThemeData is a handle assigned to the specific user interface element supported by the theme engine. TThemeServices (see page 4579) instances use an array of this type.

See Also

ThemeServices (see page 4593)

TThemedElementDetails (see page 4602)

42.4.3 Themes.TThemeServicesClass Type

TThemeServicesClass defines the metaclass for TThemeServices (see page 4579).

Pascal

```
TThemeServicesClass = class of TThemeServices;
```

C++

```
TThemeServices TThemeServicesClass;
```

File

Themes

Description

TThemeServicesClass is the metaclass for TThemeServices (see page 4579). Its value is the class reference for TThemeServices (see page 4579) or for one of its descendants.

See Also

ThemeServicesClass (see page 4631)

TThemeServices (see page 4579)

ThemeServices (see page 4593)

Class References

42.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ThemeServicesClass (see page 4631)	ThemeServicesClass contains the metaclass reference for TThemeServices (see page 4579).

42.5.1 Themes.ThemeServicesClass Variable

ThemeServicesClass contains the metaclass reference for TThemeServices (see page 4579).

Pascal

```
ThemeServicesClass: TThemeServicesClass = TThemeServices;
```

C++

```
TThemeServicesClass ThemeServicesClass = TThemeServices;
```

File

Themes

Description

ThemeServicesClass is used internally by the theme services. When the singleton instance of TThemeServices (see page 4579) is created, ThemeServicesClass is used. You are allowed to change the value of ThemeServicesClass, but you must do that before the application loads any VCL unit. All VCL units invoke the ThemeServices (see page 4593) routine that creates the instance of TThemeServices (see page 4579). Changing the value of the ThemeServicesClass variable afterwards does not yield any result.

See Also

TThemeServices (see page 4579)

TThemeServicesClass (see page 4630)

ThemeServices (see page 4593)

Class References

43 ToolWin Namespace

This is namespace ToolWin.

Topics

Name	Description
Classes (see page 4634)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4644)	The following table lists structs, records, enums in this documentation.
Types (see page 4647)	The following table lists types in this documentation.

Classes

	Name	Description
	TToolDockForm (see page 4634)	TToolDockForm is a default floating dock site class for tool buttons.
	TToolDockObject (see page 4637)	Handles drag-and-dock operations for toolbar controls.
	TToolWindow (see page 4639)	TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

Structs, Records, Enums

	Name	Description
	TEdgeBorder (see page 4644)	TEdgeBorders (see page 4647) is the type of the EdgeBorders property.
	TEdgeStyle (see page 4645)	TEdgeStyle describes the appearance of an edge border.
	TSizingOrientation (see page 4645)	TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

Types

Name	Description
TEdgeBorders (see page 4647)	This is type ToolWin.TEdgeBorders.

43.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TToolDockForm (see page 4634)	TToolDockForm is a default floating dock site class for tool buttons.
	TToolDockObject (see page 4637)	Handles drag-and-dock operations for toolbar controls.
	TToolWindow (see page 4639)	TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

43.1.1 TToolDockForm Class

TToolDockForm is a default floating dock site class for tool buttons.

Class Hierarchy

Pascal

```
TToolDockForm = class(TCustomDockForm);
```

C++

```
class TToolDockForm : public TCustomDockForm;
```

File

ToolWin

Description

Assign TToolDockForm to the FloatingDockSiteClass property of tool buttons to get appropriate docking behavior.

43.1.1.1 TToolDockForm Members

The following tables list the members exposed by TToolDockForm.

Protected Methods

	Name	Description
	CanResize (see page 4635)	Specifies whether the control can resize itself.
	CreateParams (see page 4636)	Initializes the window-creation parameter record when the form window is created.
	DoAddDockClient (see page 4636)	Performs actions when a control is docked to the form.
	WMNCCreate (see page 4636)	This is WMNCCreate, a member of class TToolDockForm.
	WMNCHitTest (see page 4636)	This is WMNCHitTest, a member of class TToolDockForm.
	WMNCLButtonDown (see page 4637)	This is WMNCLButtonDown, a member of class TToolDockForm.
	WMSize (see page 4637)	This is WMSize, a member of class TToolDockForm.
	WMSysCommand (see page 4637)	This is WMSysCommand, a member of class TToolDockForm.

Public Methods

	Name	Description
	Create (see page 4635)	Creates an instance of TToolDockForm (see page 4634).

43.1.1.2 TToolDockForm.Create Constructor

Creates an instance of TToolDockForm (see page 4634).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TToolDockForm(TComponent * AOwner);
```

Description

AOwner is the dock form's owner, which is responsible for freeing the tool dock form.

The constructor initializes the BorderWidth property to 1.

See Also

BorderWidth

Owner

43.1.1.3 TToolDockForm Methods

The methods of the TToolDockForm class are listed here.

Protected Methods

	Name	Description
	CanResize (see page 4635)	Specifies whether the control can resize itself.
	CreateParams (see page 4636)	Initializes the window-creation parameter record when the form window is created.
	DoAddDockClient (see page 4636)	Performs actions when a control is docked to the form.
	WMNCCreate (see page 4636)	This is WMNCCreate, a member of class TToolDockForm.
	WMNCHitTest (see page 4636)	This is WMNCHitTest, a member of class TToolDockForm.
	WMNCLButtonDown (see page 4637)	This is WMNCLButtonDown, a member of class TToolDockForm.
	WMSize (see page 4637)	This is WMSize, a member of class TToolDockForm.
	WMSysCommand (see page 4637)	This is WMSysCommand, a member of class TToolDockForm.

43.1.1.3.1 TToolDockForm.CanResize Method

Specifies whether the control can resize itself.

Pascal

```
function CanResize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean CanResize(int NewWidth, int NewHeight);
```

Description

CanResize is called automatically when an attempt is made to resize the control, after any autosizing has occurred. The NewWidth and NewHeight parameters specify the attempted new values for the control's height and width. CanResize optionally

adjusts the values of NewWidth and NewHeight, and returns true if the control can be resized to the returned values. CanResize returns false if the control can't be resized.

43.1.1.3.2 TToolDockForm.CreateParams Method

Initializes the window-creation parameter record when the form window is created.

Pascal

```
procedure CreateParams(var Params: TCreateParams); override;
```

C++

```
virtual __fastcall CreateParams(TCreateParams Params);
```

Description

CreateParams calls the inherited CreateParams method, and then adds WS_POPUP to the Style property.

43.1.1.3.3 TToolDockForm.DoAddDockClient Method

Performs actions when a control is docked to the form.

Pascal

```
procedure DoAddDockClient(Client: TControl; const ARect: TRect); override;
```

C++

```
virtual __fastcall DoAddDockClient(TControl * Client, const TRect ARect);
```

Description

DoAddDockClient sets the client's Align property to alNone and its BoundsRect property to ClientRect. Then DoAddDockClient calls the inherited DoAddDockClient method and sets the form's AutoSize property to true.

43.1.1.3.4 TToolDockForm.WMNCCreate Method

Pascal

```
procedure WMNCCreate(var Message: TWMNCCreate); message WM_NCCREATE;
```

C++

```
__fastcall WMNCCreate(TWMNCCreate Message);
```

Description

This is WMNCCreate, a member of class TToolDockForm.

43.1.1.3.5 TToolDockForm.WMNCHitTest Method

Pascal

```
procedure WMNCHitTest(var Message: TWMNCHitTest); message WM_NCHITTEST;
```

C++

```
__fastcall WMNCHitTest(TWMNCHitTest Message);
```

Description

This is WMNCHitTest, a member of class TToolDockForm.

43.1.1.3.6 TToolDockForm.WMNCLButtonDown Method

Pascal

```
procedure WMNCLButtonDown(var Message: TWMNCLButtonDown); message WM_NCLBUTTONDOWN;
```

C++

```
__fastcall WMNCLButtonDown(TWMNCLButtonDown Message);
```

Description

This is WMNCLButtonDown, a member of class TToolDockForm.

43.1.1.3.7 TToolDockForm.WMSize Method

Pascal

```
procedure WMSize(var Message: TWMSize); message WM_SIZE;
```

C++

```
__fastcall WMSize(TWMSize Message);
```

Description

This is WMSize, a member of class TToolDockForm.

43.1.1.3.8 TToolDockForm.WMSysCommand Method

Pascal

```
procedure WMSysCommand(var Message: TWMSysCommand); message WM_SYSCOMMAND;
```

C++

```
__fastcall WMSysCommand(TWMSysCommand Message);
```

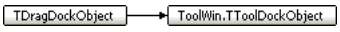
Description

This is WMSysCommand, a member of class TToolDockForm.

43.1.2 TToolDockObject Class

Handles drag-and-dock operations for toolbar controls.

Class Hierarchy



Pascal

```
TToolDockObject = class(TDragDockObject);
```

C++

```
class TToolDockObject : public TDragDockObject;
```

File

ToolWin

Description

Use TToolDockObject to handle drag-and-dock operations for toolbar controls that are docked to a TControlBar object.

Note: TToolDockObject is designed for tool bars that are docked to TControlBar. It may not work as expected when the application is docking a tool bar to a form or panel.

See Also

TToolBar

TControlBar

43.1.2.1 TToolDockObject Members

The following tables list the members exposed by TToolDockObject.

Protected Methods

	Name	Description
	AdjustDockRect (see page 4638)	Adjusts the value of DockRect to reflect the dragged control's current status.
	DrawDragDockImage (see page 4639)	Called when the dock image for a toolbar control is drawn.
	EraseDragDockImage (see page 4639)	Called when the dock image for a toolbar control is erased.

Public Methods

	Name	Description
	Create (see page 4638)	Creates and initializes an instance of TDragDockObject.

43.1.2.2 TToolDockObject.Create Constructor

Creates and initializes an instance of TDragDockObject.

Pascal

```
constructor Create(AControl: TControl); override;
```

C++

```
virtual __fastcall TToolDockObject(TControl * AControl);
```

Description

Create calls the inherited constructor and allocates memory for the Brush property.

43.1.2.3 TToolDockObject Methods

The methods of the TToolDockObject class are listed here.

Protected Methods

	Name	Description
	AdjustDockRect (see page 4638)	Adjusts the value of DockRect to reflect the dragged control's current status.
	DrawDragDockImage (see page 4639)	Called when the dock image for a toolbar control is drawn.
	EraseDragDockImage (see page 4639)	Called when the dock image for a toolbar control is erased.

43.1.2.3.1 TToolDockObject.AdjustDockRect Method

Adjusts the value of DockRect to reflect the dragged control's current status.

Pascal

```
procedure AdjustDockRect(ARect: TRect); override;
```

C++

```
virtual __fastcall AdjustDockRect(TRect ARect);
```

Description

AdjustDockRect is called automatically at points during the drag operation to adjust the value of the DockRect property. The adjustment ensures that the position where the control is docked reflects the current mouse position, and, if the control is floating, reflects the size and shape of the control's floating dock site.

See Also

DockRect

Floating

43.1.2.3.2 TToolDockObject.DrawDragDockImage Method

Called when the dock image for a toolbar control is drawn.

Pascal

```
procedure DrawDragDockImage; override;
```

C++

```
virtual __fastcall DrawDragDockImage();
```

Description

DrawDragDockImage draws a control that is moved in a toolbar.

43.1.2.3.3 TToolDockObject.EraseDragDockImage Method

Called when the dock image for a toolbar control is erased.

Pascal

```
procedure EraseDragDockImage; override;
```

C++

```
virtual __fastcall EraseDragDockImage();
```

Description

EraseDragDockImage erases a control that is moved in a toolbar.

43.1.3 TToolWindow Class

TToolWindow is the ancestor for windowed controls that require a custom client area and edge borders.

Class Hierarchy

Pascal

```
TToolWindow = class(TWinControl);
```

C++

```
class TToolWindow : public TWinControl;
```

File

ToolWin

Description

TToolWindow is the ancestor of the TCoolBar and TToolBar components, which facilitate the arrangement of buttons and other visual controls.

See Also

TCoolBar

TToolBar

43.1.3.1 TToolWindow Members

The following tables list the members exposed by TToolWindow.

Protected Methods

	Name	Description
	CMBorderChanged (see page 4641)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 4641)	This is CMCtl3DChanged, a member of class TToolWindow.
V	NCPaint (see page 4641)	This is NCPaint, a member of class TToolWindow.
	WMNCCalcSize (see page 4641)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 4642)	This is WMNCPaint, a member of class TToolWindow.

Public Methods

	Name	Description
V	Create (see page 4640)	Creates and initializes a TToolWindow (see page 4639) instance.

Public Properties

	Name	Description
	EdgeBorders (see page 4642)	Determines which sides of the control have visible borders.
	EdgeInner (see page 4642)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 4643)	Determines the border style for the control's outer edge.

43.1.3.2 TToolWindow.Create Constructor

Creates and initializes a TToolWindow (see page 4639) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TToolWindow(TComponent * AOwner);
```

Description

Create generates a TToolWindow (see page 4639) and initializes its properties.

43.1.3.3 TToolWindow Methods

The methods of the TToolWindow class are listed here.

Protected Methods

	Name	Description
	CMBorderChanged (see page 4641)	This is CMBorderChanged, a member of class TToolWindow.
	CMCtl3DChanged (see page 4641)	This is CMCtl3DChanged, a member of class TToolWindow.
	NCPaint (see page 4641)	This is NCPaint, a member of class TToolWindow.
	WMNCCalcSize (see page 4641)	This is WMNCCalcSize, a member of class TToolWindow.
	WMNCPaint (see page 4642)	This is WMNCPaint, a member of class TToolWindow.

43.1.3.3.1 TToolWindow.CMBorderChanged Method

Pascal

```
procedure CMBorderChanged(var Message: TMessage); message CM_BORDERCHANGED;
```

C++

```
__fastcall CMBorderChanged(TMessage Message);
```

Description

This is CMBorderChanged, a member of class TToolWindow.

43.1.3.3.2 TToolWindow.CMCtl3DChanged Method

Pascal

```
procedure CMCtl3DChanged(var Message: TMessage); message CM_CTL3DCHANGED;
```

C++

```
__fastcall CMCtl3DChanged(TMessage Message);
```

Description

This is CMCtl3DChanged, a member of class TToolWindow.

43.1.3.3.3 TToolWindow.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); virtual;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TToolWindow.

43.1.3.3.4 TToolWindow.WMNCCalcSize Method

Pascal

```
procedure WMNCCalcSize(var Message: TWMNCCalcSize); message WM_NCCALCSIZE;
```

C++

```
__fastcall WMNCCalcSize(TWMNCCalcSize Message);
```

Description

This is WMNCCalcSize, a member of class TToolWindow.

43.1.3.3.5 TToolWindow.WMNCPaint Method

Pascal

```
procedure WMNCPaint(var Message: TWMNCPaint); message WM_NCPAINT;
```

C++

```
__fastcall WMNCPaint(TWMNCPaint Message);
```

Description

This is WMNCPaint, a member of class TToolWindow.

43.1.3.4 TToolWindow Properties

The properties of the TToolWindow class are listed here.

Public Properties

	Name	Description
	EdgeBorders (see page 4642)	Determines which sides of the control have visible borders.
	EdgeInner (see page 4642)	Determines the border style for the control's inner edge.
	EdgeOuter (see page 4643)	Determines the border style for the control's outer edge.

43.1.3.4.1 TToolWindow.EdgeBorders Property

Determines which sides of the control have visible borders.

Pascal

```
property EdgeBorders: TEdgeBorders;
```

C++

```
__property TEdgeBorders EdgeBorders;
```

Description

Use the EdgeBorders property to add or remove borders. The appearance of the borders can be set with the EdgeInner (see page 4642) and EdgeOuter (see page 4643) properties.

See Also

EdgeInner (see page 4642)

EdgeOuter (see page 4643)

43.1.3.4.2 TToolWindow.EdgeInner Property

Determines the border style for the control's inner edge.

Pascal

```
property EdgeInner: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeInner;
```

Description

Use EdgeInner to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeInner specifies the appearance of the inner border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner property has an effect only on the sides specified by the EdgeBorders (see page 4642) property.

Use EdgeInner with EdgeOuter (see page 4643) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 4642)

EdgeOuter (see page 4643)

43.1.3.4.3 TToolWindow.EdgeOuter Property

Determines the border style for the control's outer edge.

Pascal

```
property EdgeOuter: TEdgeStyle;
```

C++

```
__property TEdgeStyle EdgeOuter;
```

Description

Use EdgeOuter to provide the tool window with a three-dimensional beveled look. A tool window has two borders on each side, an outer border drawn next to the outer edge, and an inner border drawn inside the outer border. EdgeOuter specifies the appearance of the outer border. It can be raised (esRaised), lowered (esLowered), or left off altogether (esNone).

Note: The EdgeInner (see page 4642) property has an effect only on the sides specified by the EdgeBorders (see page 4642) property.

Use EdgeOuter with EdgeInner (see page 4642) to give the component a three-dimensional look.

See Also

EdgeBorders (see page 4642)

EdgeInner (see page 4642)

43.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEdgeBorder (see page 4644)	TEdgeBorders (see page 4647) is the type of the EdgeBorders property.
	TEdgeStyle (see page 4645)	TEdgeStyle describes the appearance of an edge border.
	TSizingOrientation (see page 4645)	TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

43.2.1 ToolWin.TEdgeBorder Enumeration

TEdgeBorders (see page 4647) is the type of the EdgeBorders property.

Pascal

```
TEdgeBorder = (
  ebLeft,
  ebTop,
  ebRight,
  ebBottom
);
```

C++

```
enum TEdgeBorder {
  ebLeft,
  ebTop,
  ebRight,
  ebBottom
};
```

File

ToolWin

Description

TEdgeBorders (see page 4647) is a set of TEdgeBorder values. Each TEdgeBorder value indicates whether a beveled edge appears on a particular side of a control. The following table lists the TEdgeBorder values:

Value	Meaning
ebLeft	The control has a beveled left edge.
ebTop	The control has a beveled top edge.
ebRight	The control has a beveled right edge.
ebBottom	The control has a beveled bottom edge.

In addition to the values in the preceding table, the QGraphics unit defines the ebRect constant to represent all the edges of a control:

const ebRect = [ebLeft, ebTop, ebRight, ebBottom];

#define ebRect (System::Set<TEdgeBorder, ebLeft, ebBottom> () << TEdgeBorder(0) << TEdgeBorder(1) << TEdgeBorder(2)

<< TEdgeBorder(3))

43.2.2 ToolWin.TEdgeStyle Enumeration

TEdgeStyle describes the appearance of an edge border.

Pascal

```
TEdgeStyle = (
  esNone,
  esRaised,
  esLowered
);
```

C++

```
enum TEdgeStyle {
  esNone,
  esRaised,
  esLowered
};
```

File

ToolWin

Description

TEdgeStyle indicates the appearance of a particular border. The following table lists the possible values:

Value	Meaning
esNone	The border is not drawn.
esRaised	The border appears raised.
esLowered	The border appears lowered.

43.2.3 ToolWin.TSizingOrientation Enumeration

TSizingOrientation specifies the dimension along which a control's size is adjusted when docked.

Pascal

```
TSizingOrientation = (
  soNone,
  soHorizontal,
  soVertical
);
```

C++

```
enum TSizingOrientation {
  soNone,
  soHorizontal,
  soVertical
};
```

File

ToolWin

Description

TSizingOrientation can have one of the following values:

Value	Meaning
soNone	The control's size is not readjusted when it is docked.
soHorizontal	The control's width is adjusted when it is docked.
soVertical	The control's height is adjusted when it is docked.

43.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEdgeBorders (see page 4647)	This is type ToolWin.TEdgeBorders.

43.3.1 ToolWin.TEdgeBorders Type

Pascal

```
TEdgeBorders = set of TEdgeBorder;
```

C++

```
TEdgeBorder TEdgeBorders;
```

File

ToolWin

Description

This is type ToolWin.TEdgeBorders.

44 Types Namespace

This is namespace Types.

Topics

Name	Description
Functions (see page 4651)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4659)	The following table lists structs, records, enums in this documentation.
Types (see page 4663)	The following table lists types in this documentation.
Constants (see page 4675)	The following table lists constants in this documentation.

Constants

Name	Description
EqualsValue (see page 4675)	EqualsValue indicates that both values are equal in a comparison operation.
GreaterThanValue (see page 4675)	GreaterThanValue indicates that one value is greater than an other in a comparison operation.
LessThanValue (see page 4676)	LessThanValue indicates that one value is less than an other in a comparison operation.
RT_RCDATA (see page 4676)	This is constant Types.RT_RCDATA.

Functions

	Name	Description
	Bounds (see page 4651)	Returns the TRect (see page 4660) for a rectangle of given dimensions.
	CenteredRect (see page 4652)	This is function Types.CenteredRect.
	CenterPoint (see page 4652)	Returns the point in the center of a rectangle.
	EqualRect (see page 4652)	Indicates whether two TRect (see page 4660) values are the same.
	IntersectRect (see page 4653)	Returns the intersection of two rectangles
	IsRectEmpty (see page 4653)	Indicates wither a specified rectangle has a positive width and height.
	OffsetRect (see page 4654)	Changes the origin of a rectangle by a specified amount.
	Point (see page 4654)	Creates a TPoint (see page 4660) structure from a pair of coordinates.
	PtInRect (see page 4655)	Indicates whether a specified point lies inside a specified rectangle.
	Rect (see page 4655)	Creates a TRect (see page 4660) structure from a set of coordinates.
	RectHeight (see page 4656)	This is function Types.RectHeight.
	RectWidth (see page 4656)	This is function Types.RectWidth.
	SmallPoint (see page 4656)	Creates a TSmallPoint (see page 4661) structure from a pair of coordinates.
	SmallPoint (see page 4656)	Creates a TSmallPoint (see page 4661) structure from a pair of coordinates.
	SplitRect (see page 4657)	This is function Types.SplitRect.

	SplitRect (see page 4657)	This is function Types.SplitRect.
	UnionRect (see page 4658)	Returns the union of two rectangles

Structs, Records, Enums

	Name	Description
	tagSIZE (see page 4659)	tagSIZE is a record holding width and height values.
	TPoint (see page 4660)	TPoint type defines a pixel location onscreen.
	TRect (see page 4660)	TRect defines a rectangle.
	TSmallPoint (see page 4661)	TSmallPoint type defines a point with two 16-bit coordinates.
	TSplitRectType (see page 4662)	This is record Types.TSplitRectType.

Types

Name	Description
DWORD (see page 4663)	DWORD is equivalent to LongWord.
PByte (see page 4664)	PByte is a pointer to a `Byte` variable.
PDouble (see page 4664)	PDouble is a pointer to a `Double` variable.
PInteger (see page 4664)	PInteger is a pointer to a `Integer` variable.
PLongint (see page 4665)	PLongint is a pointer to a `LongInt` variable.
PPoint (see page 4665)	PPoint is a pointer to a TPoint (see page 4660) variable.
PRect (see page 4666)	PRect is a pointer to a TRect (see page 4660) variable.
PSize (see page 4666)	PSize is a pointer to a TSize (see page 4671) variable.
PSmallInt (see page 4666)	This is type Types.PSmallInt.
PSmallPoint (see page 4667)	This is type Types.PSmallPoint.
SIZE (see page 4667)	This is type Types.SIZE.
tagPOINT (see page 4667)	This is type Types.tagPOINT.
TBooleanDynArray (see page 4668)	TBooleanDynArray defines an array of Boolean elements.
TByteDynArray (see page 4668)	This is type Types.TByteDynArray.
TCardinalDynArray (see page 4668)	TCardinalDynArray defines an array of Cardinal elements.
TDoubleDynArray (see page 4669)	TDoubleDynArray defines an array of Double elements.
TInt64DynArray (see page 4669)	TInt64DynArray defines an array of Int64 elements.
TIntegerDynArray (see page 4670)	This is type Types.TIntegerDynArray.
TLongWordDynArray (see page 4670)	TLongWordDynArray defines an array of LongWord elements.
TShortIntDynArray (see page 4670)	TShortIntDynArray defines an array of ShortInt elements.
TSingleDynArray (see page 4671)	TSingleDynArray defines an array of Single elements.
TSize (see page 4671)	TSize represents a width and height.
TSmallIntDynArray (see page 4672)	TSmallIntDynArray defines an array of SmallInt elements.
TStringDynArray (see page 4672)	TStringDynArray defines an array of String elements.
TValueRelationship (see page 4673)	TValueRelationship defines value comparison possibilities.
TWideStringDynArray (see page 4673)	TWideStringDynArray defines an array of WideString (see page 3467) elements.
TWordDynArray (see page 4674)	TWordDynArray defines an array of Word elements.

44.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	Bounds (see page 4651)	Returns the TRect (see page 4660) for a rectangle of given dimensions.
	CenteredRect (see page 4652)	This is function Types.CenteredRect.
	CenterPoint (see page 4652)	Returns the point in the center of a rectangle.
	EqualRect (see page 4652)	Indicates whether two TRect (see page 4660) values are the same.
	IntersectRect (see page 4653)	Returns the intersection of two rectangles
	IsRectEmpty (see page 4653)	Indicates wither a specified rectangle has a positive width and height.
	OffsetRect (see page 4654)	Changes the origin of a rectangle by a specified amount.
	Point (see page 4654)	Creates a TPoint (see page 4660) structure from a pair of coordinates.
	PtInRect (see page 4655)	Indicates whether a specified point lies inside a specified rectangle.
	Rect (see page 4655)	Creates a TRect (see page 4660) structure from a set of coordinates.
	RectHeight (see page 4656)	This is function Types.RectHeight.
	RectWidth (see page 4656)	This is function Types.RectWidth.
	SmallPoint (see page 4656)	Creates a TSmallPoint (see page 4661) structure from a pair of coordinates.
	SmallPoint (see page 4656)	Creates a TSmallPoint (see page 4661) structure from a pair of coordinates.
	SplitRect (see page 4657)	This is function Types.SplitRect.
	SplitRect (see page 4657)	This is function Types.SplitRect.
	UnionRect (see page 4658)	Returns the union of two rectangles

44.1.1 Types.Bounds Function

Returns the TRect (see page 4660) for a rectangle of given dimensions.

Pascal

```
function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer): TRect;
```

C++

```
TRect Bounds(int ALeft, int ATop, int AWidth, int AHeight);
```

File

Types

Description

Call Bounds to obtain a TRect (see page 4660) for a rectangle with upper left corner at the point (ALeft, ATop) with width AWidth and height AHeight.

See Also

Rect (see page 4655)

44.1.2 Types.CenteredRect Function

Pascal

```
function CenteredRect(SourceRect: TRect; CenteredRect: TRect): TRect;
```

C++

```
TRect CenteredRect(TRect SourceRect, TRect CenteredRect);
```

File

Types

Description

This is function Types.CenteredRect.

44.1.3 Types.CenterPoint Function

Returns the point in the center of a rectangle.

Pascal

```
function CenterPoint(const Rect: TRect): TPoint;
```

C++

```
TPoint CenterPoint(const TRect Rect);
```

File

Types

Description

Call CenterPoint to obtain the point in the center of a rectangle.

Rect (see page 4655) is the rectangle whose center you want to locate.

See Also

PtInRect (see page 4655)

IsRectEmpty (see page 4653)

UnionRect (see page 4658)

IntersectRect (see page 4653)

44.1.4 Types.EqualRect Function

Indicates whether two TRect (see page 4660) values are the same.

Pascal

```
function EqualRect(const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean EqualRect(const TRect R1, const TRect R2);
```

File

Types

Description

Call EqualRect to compare two TRect (see page 4660) values. EqualRect returns true if R1 and R2 are identical (same position and size), false if they are different.

See Also

Rect (see page 4655)

Bounds (see page 4651)

IntersectRect (see page 4653)

44.1.5 Types.IntersectRect Function

Returns the intersection of two rectangles

Pascal

```
function IntersectRect(out Rect: TRect; const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean IntersectRect(TRect Rect, const TRect R1, const TRect R2);
```

File

Types

Description

Call IntersectRect to obtain the rectangle that is the intersection of R1 and R2. The intersection is returned as the Rect (see page 4655) parameter.

IntersectRect returns true if the two rectangles have a nonempty intersection. If R1 and R2 do not overlap, IntersectRect returns false, and the Rect (see page 4655) parameter is set to a rectangle at position (0,0) with 0 Width and 0 Height.

See Also

PtInRect (see page 4655)

IsRectEmpty (see page 4653)

UnionRect (see page 4658)

44.1.6 Types.IsRectEmpty Function

Indicates wither a specified rectangle has a positive width and height.

Pascal

```
function IsRectEmpty(const Rect: TRect): Boolean;
```

C++

```
Boolean IsRectEmpty(const TRect Rect);
```

File

Types

Description

Call IsRectEmpty to determine whether the rectangle specified by Rect (see page 4655) encloses a positive amount of space.

IsRectEmpty returns true if both the width and height of Rect (see page 4655) are greater than 0. If either the width or height is less than or equal to 0, IsRectEmpty returns false. Note that this means non-normalized rectangles (rectangles with negative height or width) are considered empty.

See Also

PtInRect (see page 4655)

IntersectRect (see page 4653)

44.1.7 Types.OffsetRect Function

Changes the origin of a rectangle by a specified amount.

Pascal

```
function OffsetRect(var Rect: TRect; DX: Integer; DY: Integer): Boolean;
```

C++

```
Boolean OffsetRect(TRect Rect, int DX, int DY);
```

File

Types

Description

Call OffsetRect to move the upper left corner of the rectangle specified by Rect (see page 4655) while maintaining the same width and height. The left of the rectangle moves by DX while the top moves by DY.

OffsetRect returns true if the Rect (see page 4655) parameter is not nil (Delphi) or NULL (C++), false if it can't return a new rectangle because Rect (see page 4655) is nil(Delphi) or NULL (C++).

See Also

Rect (see page 4655)

Bounds (see page 4651)

44.1.8 Types.Point Function

Creates a TPoint (see page 4660) structure from a pair of coordinates.

Pascal

```
function Point(X: Integer; Y: Integer): TPoint;
```

C++

```
TPoint Point(int X, int Y);
```

File

Types

Description

Call Point to create a TPoint (see page 4660) that represents the specified coordinates. Use Point to construct parameters for functions that require a point, rather than setting up local variables for each parameter.

See Also

SmallPoint (see page 4656)

Rect (see page 4655)

44.1.9 Types.PtInRect Function

Indicates whether a specified point lies inside a specified rectangle.

Pascal

```
function PtInRect(const Rect: TRect; const P: TPoint): Boolean;
```

C++

```
Boolean PtInRect(const TRect Rect, const TPoint P);
```

File

Types

Description

PtInRect returns true if the point, P, lies inside the rectangle, Rect (see page 4655), and false if P is not in Rect (see page 4655).

PtInRect does not consider a point inside a rectangle if the rectangle has a negative width or height. A point is considered inside a rectangle if it lies on the left or top edge, but not if it lies on the right or bottom edge.

See Also

IntersectRect (see page 4653)

IsRectEmpty (see page 4653)

CenterPoint (see page 4652)

44.1.10 Types.Rect Function

Creates a TRect (see page 4660) structure from a set of coordinates.

Pascal

```
function Rect(Left: Integer; Top: Integer; Right: Integer; Bottom: Integer): TRect;
```

C++

```
TRect Rect(int Left, int Top, int Right, int Bottom);
```

File

Types

Description

Call Rect to create a TRect (see page 4660) that represents the rectangle with the specified coordinates. Use Rect to construct parameters for functions that require TRect (see page 4660), rather than setting up local variables for each parameter.

See Also

Bounds (see page 4651)

Point (see page 4654)

44.1.11 Types.RectHeight Function

Pascal

```
function RectHeight(const Rect: TRect): Integer;
```

C++

```
int RectHeight(const TRect Rect);
```

File

Types

Description

This is function Types.RectHeight.

44.1.12 Types.RectWidth Function

Pascal

```
function RectWidth(const Rect: TRect): Integer;
```

C++

```
int RectWidth(const TRect Rect);
```

File

Types

Description

This is function Types.RectWidth.

44.1.13 Types.SmallPoint Function

Creates a TSmallPoint (see page 4661) structure from a pair of coordinates.

Pascal

```
function SmallPoint(X: Integer; Y: Integer): TSmallPoint; overload;
function SmallPoint(XY: LongWord): TSmallPoint; overload;
```

C++

```
TSmallPoint SmallPoint(int X, int Y);
TSmallPoint SmallPoint(LongWord XY);
```

File

Types

Description

Call SmallPoint to create a TSmallPoint (see page 4661) that represents the specified coordinates.

See Also

Point

44.1.14 Types.SplitRect Function

Pascal

```
function SplitRect(Rect: TRect; SplitType: TSplitRectType; Percent: Double): TRect; overload;
```

C++

```
TRect SplitRect(TRect Rect, TSplitRectType SplitType, Double Percent);
```

File

Types

Description

This is function Types.SplitRect.

44.1.15 Types.SplitRect Function

Pascal

```
function SplitRect(Rect: TRect; SplitType: TSplitRectType; Size: Integer): TRect; overload;
```

C++

```
TRect SplitRect(TRect Rect, TSplitRectType SplitType, int Size);
```

File

Types

Description

This is function Types.SplitRect.

44.1.16 Types.UnionRect Function

Returns the union of two rectangles

Pascal

```
function UnionRect(out Rect: TRect; const R1: TRect; const R2: TRect): Boolean;
```

C++

```
Boolean UnionRect(TRect Rect, const TRect R1, const TRect R2);
```

File

Types

Description

Call UnionRect to obtain the smallest rectangle that includes both R1 and R2. The union is returned as the Rect (see page 4655) parameter.

UnionRect returns true if the resulting rectangle is nonempty. If the resulting rectangle as a width or height that is less than or equal to 0, UnionRect returns false.

See Also

IntersectRect (see page 4653)

OffsetRect (see page 4654)

IsRectEmpty (see page 4653)

44.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TSplitRectType (see page 4662)	This is record Types.TSplitRectType.

Records

	Name	Description
	tagSIZE (see page 4659)	tagSIZE is a record holding width and height values.
	TPoint (see page 4660)	TPoint type defines a pixel location onscreen.
	TRect (see page 4660)	TRect defines a rectangle.
	TSmallPoint (see page 4661)	TSmallPoint type defines a point with two 16-bit coordinates.

44.2.1 Types.tagSIZE Record

tagSIZE is a record holding width and height values.

Pascal

```
tagSIZE = record
  cx: Longint;
  cy: Longint;
end;
```

C++

```
struct tagSIZE {
  Longint cx;
  Longint cy;
};
```

File

Types

Description

The SIZE (see page 4667) equates to a TSize (see page 4671) type, which in turn equates to a tagSIZE record.

tagSIZE holds the width (cx) and height (cy) of something, typically in pixels. For example, TextExtent returns the width and height, in pixels, of a string rendered in the current font.

See Also

SIZE (see page 4667)

TextExtent

TSize (see page 4671)

44.2.2 Types.TPoint Record

TPoint type defines a pixel location onscreen.

Pascal

```
TPoint = record
  X: Longint;
  Y: Longint;
end;
```

C++

```
struct TPoint {
  Longint X;
  Longint Y;
};
```

File

Types

Description

The TPoint type defines a pixel location onscreen, with the origin in the top left corner. X specifies the horizontal coordinate of the point, Y specifies the vertical coordinate.

C++ Examples:

```
/*
This example draws a polygon in the specified shape, and
fills it with the color teal.
*/
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
  TPoint points[4];
  points[0] = Point(10,10);
  points[1] = Point(30,10);
  points[2] = Point(130,30);
  points[3] = Point(240,120);
  (dynamic_cast<TPaintBox *>(Sender))->Canvas->Brush->Color = clTeal;
  (dynamic_cast<TPaintBox *>(Sender))->Canvas->Polygon(points, 3);
}
```

See Also

Point (see page 4654)

44.2.3 Types.TRect Record

TRect defines a rectangle.

Pascal

```
TRect = record
  case Integer of
    0: (Bottom: Longint;
        Left: Longint;
        Right: Longint;
        Top: Longint;);
```

```
      1: (BottomRight: TPoint;
          TopLeft: TPoint;);
  end;
```

C++

```
  struct TRect {
    Longint Bottom;
    Longint Left;
    Longint Right;
    Longint Top;
    TPoint BottomRight;
    TPoint TopLeft;
  };
```

File

Types

Description

TRect represents the dimensions of a rectangle. The coordinates are specified as either four separate integers representing the left, top, right, and bottom sides, or as two points representing the locations of the top left and bottom right corners.

Typically, TRect values represent pixel locations, where the origin of the pixel coordinate system is in the top left corner of the screen (screen coordinates) or the top left corner of a control's client area (client coordinates). When a TRect value represents a rectangle on the screen, by convention the top and left edges are considered inside the rectangle and the bottom and right edges are considered outside the rectangle. This convention allows the width of the rectangle to be Right - Left and the height to be Bottom - Top.

44.2.4 Types.TSmallPoint Record

TSmallPoint type defines a point with two 16-bit coordinates.

Pascal

```
  TSmallPoint = record
    x: SmallInt;
    y: SmallInt;
  end;
```

C++

```
  struct TSmallPoint {
    SmallInt x;
    SmallInt y;
  };
```

File

Types

Description

The TSmallPoint type defines a point where each coordinate is a 16-bit integer. x specifies the horizontal coordinate of the point, y specifies the vertical coordinate.

44.2.5 Types.TSplitRectType Enumeration

Pascal

```
TSplitRectType = (
  srLeft,
  srRight,
  srTop,
  srBottom
);
```

C++

```
enum TSplitRectType {
  srLeft,
  srRight,
  srTop,
  srBottom
};
```

File

Types

Description

This is record Types.TSplitRectType.

44.3 Types

The following table lists types in this documentation.

Types

Name	Description
DWORD (see page 4663)	DWORD is equivalent to LongWord.
PByte (see page 4664)	PByte is a pointer to a `Byte` variable.
PDouble (see page 4664)	PDouble is a pointer to a `Double` variable.
PInteger (see page 4664)	PInteger is a pointer to a `Integer` variable.
PLongint (see page 4665)	PLongint is a pointer to a `LongInt` variable.
PPoint (see page 4665)	PPoint is a pointer to a TPoint (see page 4660) variable.
PRect (see page 4666)	PRect is a pointer to a TRect (see page 4660) variable.
PSize (see page 4666)	PSize is a pointer to a TSize (see page 4671) variable.
PSmallInt (see page 4666)	This is type Types.PSmallInt.
PSmallPoint (see page 4667)	This is type Types.PSmallPoint.
SIZE (see page 4667)	This is type Types.SIZE.
tagPOINT (see page 4667)	This is type Types.tagPOINT.
TBooleanDynArray (see page 4668)	TBooleanDynArray defines an array of Boolean elements.
TByteDynArray (see page 4668)	This is type Types.TByteDynArray.
TCardinalDynArray (see page 4668)	TCardinalDynArray defines an array of Cardinal elements.
TDoubleDynArray (see page 4669)	TDoubleDynArray defines an array of Double elements.
TInt64DynArray (see page 4669)	TInt64DynArray defines an array of Int64 elements.
TIntegerDynArray (see page 4670)	This is type Types.TIntegerDynArray.
TLongWordDynArray (see page 4670)	TLongWordDynArray defines an array of LongWord elements.
TShortIntDynArray (see page 4670)	TShortIntDynArray defines an array of ShortInt elements.
TSingleDynArray (see page 4671)	TSingleDynArray defines an array of Single elements.
TSize (see page 4671)	TSize represents a width and height.
TSmallIntDynArray (see page 4672)	TSmallIntDynArray defines an array of SmallInt elements.
TStringDynArray (see page 4672)	TStringDynArray defines an array of String elements.
TValueRelationship (see page 4673)	TValueRelationship defines value comparison possibilities.
TWideStringDynArray (see page 4673)	TWideStringDynArray defines an array of WideString (see page 3467) elements.
TWordDynArray (see page 4674)	TWordDynArray defines an array of Word elements.

44.3.1 Types.DWORD Type

DWORD is equivalent to LongWord.

Pascal

```
DWORD = LongWord;
```

C++

```
LongWord DWORD;
```

File

Types

Description

The DWORD holds positive integers up to a maximum value of 4,294,967,295.

44.3.2 Types.PByte Type

PByte is a pointer to a `Byte` variable.

Pascal

```
PByte = System.PByte;
```

C++

```
System.PByte PByte;
```

File

Types

Description

PByte is the same as PByte.

See Also

PByte (see page 3758)

44.3.3 Types.PDouble Type

PDouble is a pointer to a `Double` variable.

Pascal

```
PDouble = System.PDouble;
```

C++

```
System.PDouble PDouble;
```

File

Types

Description

PDouble is the same as PDouble.

See Also

PDouble (see page 3762)

44.3.4 Types.PInteger Type

PInteger is a pointer to a `Integer` variable.

Pascal

```
PInteger = System.PInteger;
```

C++

```
System.PInteger PInteger;
```

File

Types

Description

Types.PInteger is the same as PInteger.

See Also

PInteger (see page 3765)

44.3.5 Types.PLongint Type

PLongint is a pointer to a `LongInt` variable.

Pascal

```
PLongint = System.PLongint;
```

C++

```
System.PLongint PLongint;
```

File

Types

Description

PLongint is the same as PLongint.

See Also

PLongint (see page 3767)

44.3.6 Types.PPoint Type

PPoint is a pointer to a TPoint (see page 4660) variable.

Pascal

```
PPoint = ^TPoint;
```

C++

```
^TPoint PPoint;
```

File

Types

Description

The TPoint (see page 4660) type defines a pixel location onscreen, with the origin in the top left corner.

See Also

TPoint (see page 4660)

44.3.7 Types.PRect Type

PRect is a pointer to a TRect (see page 4660) variable.

Pascal

```
PRect = ^TRect;
```

C++

```
^TRect PRect;
```

File

Types

Description

The TRect (see page 4660) type represents the dimensions of a rectangle.

See Also

TRect (see page 4660)

44.3.8 Types.PSize Type

PSize is a pointer to a TSize (see page 4671) variable.

Pascal

```
PSize = ^TSize;
```

C++

```
^TSize PSize;
```

File

Types

Description

The TSize (see page 4671) type holds the width and height of something, typically in pixels.

See Also

TSize (see page 4671)

44.3.9 Types.PSmallInt Type

Pascal

```
PSmallInt = System.PSmallInt;
```

C++

```
System.PSmallInt PSmallInt;
```

File

Types

Description

This is type Types.PSmallInt.

44.3.10 Types.PSmallPoint Type

Pascal

```
PSmallPoint = ^TSmallPoint;
```

C++

```
^TSmallPoint PSmallPoint;
```

File

Types

Description

This is type Types.PSmallPoint.

44.3.11 Types.SIZE Type

Pascal

```
SIZE = tagSIZE;
```

C++

```
tagSIZE SIZE;
```

File

Types

Description

This is type Types.SIZE.

44.3.12 Types.tagPOINT Type

Pascal

```
tagPOINT = TPoint;
```

C++

```
TPoint tagPOINT;
```

File

Types

Description

This is type Types.tagPOINT.

44.3.13 Types.TBooleanDynArray Type

TBooleanDynArray defines an array of Boolean elements.

Pascal

```
TBooleanDynArray = array of Boolean;
```

C++

```
array of Boolean TBooleanDynArray;
```

File

Types

Description

The TBooleanDynArray defines a dynamic array of Boolean elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.14 Types.TByteDynArray Type

Pascal

```
TByteDynArray = array of Byte;
```

C++

```
array of Byte TByteDynArray;
```

File

Types

Description

This is type Types.TByteDynArray.

44.3.15 Types.TCardinalDynArray Type

TCardinalDynArray defines an array of Cardinal elements.

Pascal

```
TCardinalDynArray = array of Cardinal;
```

C++

```
array of Cardinal TCardinalDynArray;
```

File

Types

Description

The TCardinalDynArray defines a dynamic array of Cardinal elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.16 Types.TDoubleDynArray Type

TDoubleDynArray defines an array of Double elements.

Pascal

```
TDoubleDynArray = array of Double;
```

C++

```
array of Double TDoubleDynArray;
```

File

Types

Description

The TDoubleDynArray defines a dynamic array of Double elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.17 Types.TInt64DynArray Type

TInt64DynArray defines an array of Int64 elements.

Pascal

```
TInt64DynArray = array of Int64;
```

C++

```
array of Int64 TInt64DynArray;
```

File

Types

Description

The TInt64DynArray defines a dynamic array of Int64 elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.18 Types.TIntegerDynArray Type

Pascal

```
TIntegerDynArray = array of Integer;
```

C++

```
array of Integer TIntegerDynArray;
```

File

Types

Description

This is type Types.TIntegerDynArray.

44.3.19 Types.TLongWordDynArray Type

TLongWordDynArray defines an array of LongWord elements.

Pascal

```
TLongWordDynArray = array of LongWord;
```

C++

```
array of LongWord TLongWordDynArray;
```

File

Types

Description

The TLongWordDynArray defines a dynamic array of LongWord elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.20 Types.TShortIntDynArray Type

TShortIntDynArray defines an array of ShortInt elements.

Pascal

```
TShortIntDynArray = array of ShortInt;
```

C++

```
array of ShortInt TShortIntDynArray;
```

File

Types

Description

The TShortIntDynArray defines a dynamic array of ShortInt elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.21 Types.TSingleDynArray Type

TSingleDynArray defines an array of Single elements.

Pascal

```
TSingleDynArray = array of Single;
```

C++

```
array of Single TSingleDynArray;
```

File

Types

Description

The TSingleDynArray defines a dynamic array of Single elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.22 Types.TSize Type

TSize represents a width and height.

Pascal

```
TSize = tagSIZE;
```

C++

```
tagSIZE TSize;
```

File

Types

Description

TSize represents the size of something. Typically, the implied units of measurement are pixels.

cx is the width.

cy is the height.

44.3.23 Types.TSmallIntDynArray Type

TSmallIntDynArray defines an array of SmallInt elements.

Pascal

```
TSmallIntDynArray = array of SmallInt;
```

C++

```
array of SmallInt TSmallIntDynArray;
```

File

Types

Description

The TSmallIntDynArray defines a dynamic array of SmallInt elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.24 Types.TStringDynArray Type

TStringDynArray defines an array of String elements.

Pascal

```
TStringDynArray = array of string;
```

C++

```
array of string TStringDynArray;
```

File

Types

Description

The TStringDynArray defines a dynamic array of String elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.25 Types.TValueRelationship Type

TValueRelationship defines value comparison possibilities.

Pascal

```
TValueRelationship = -1..1;
```

C++

```
-1..1 TValueRelationship;
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship value as follows :

Value	Associated constant	Meaning
-1	LessThanValue (see page 4676)	The first value is less than the second value.
0	EqualsValue (see page 4675)	The two values are equal.
+1	GreaterThanValue (see page 4675)	The first value is greater than the second value.

See Also

LessThanValue (see page 4676)

EqualsValue (see page 4675)

GreaterThanValue (see page 4675)

44.3.26 Types.TWideStringDynArray Type

TWideStringDynArray defines an array of WideString (see page 3467) elements.

Pascal

```
TWideStringDynArray = array of WideString;
```

C++

```
array of WideString TWideStringDynArray;
```

File

Types

Description

The TWideStringDynArray defines a dynamic array of WideString (see page 3467) elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.3.27 Types.TWordDynArray Type

TWordDynArray defines an array of Word elements.

Pascal

```
TWordDynArray = array of Word;
```

C++

```
array of Word TWordDynArray;
```

File

Types

Description

The TWordDynArray defines a dynamic array of Word elements.

Use SetLength (see page 3657) to allocate storage for such an array.

See Also

TSize (see page 4671)

44.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
EqualsValue (see page 4675)	EqualsValue indicates that both values are equal in a comparison operation.
GreaterThanValue (see page 4675)	GreaterThanValue indicates that one value is greater than an other in a comparison operation.
LessThanValue (see page 4676)	LessThanValue indicates that one value is less than an other in a comparison operation.
RT_RCDATA (see page 4676)	This is constant Types.RT_RCDATA.

44.4.1 Types.EqualsValue Constant

EqualsValue indicates that both values are equal in a comparison operation.

Pascal

```
EqualsValue = 0;
```

C++

```
EqualsValue = 0;
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 4673) value. If the first value is equal to the second, the value is EqualsValue

See Also

CompareValue

GreaterThanValue (see page 4675)

LessThanValue (see page 4676)

TValueRelationship (see page 4673)

44.4.2 Types.GreaterThanValue Constant

GreaterThanValue indicates that one value is greater than an other in a comparison operation.

Pascal

```
GreaterThanValue = High(TValueRelationship);
```

C++

```
GreaterThanValue = High(TValueRelationship);
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 4673) value. If the first value is greater than the second, the value is GreaterThanValue

See Also

EqualsValue (see page 4675)

CompareValue

LessThanValue (see page 4676)

TValueRelationship (see page 4673)

44.4.3 Types.LessThanValue Constant

LessThanValue indicates that one value is less than an other in a comparison operation.

Pascal

```
LessThanValue = Low(TValueRelationship);
```

C++

```
LessThanValue = Low(TValueRelationship);
```

File

Types

Description

Routines such as the CompareValue routine compare two values for equality. They return the result in a TValueRelationship (see page 4673) value. If the first value is less than the second, the value is LessThanValue

See Also

EqualsValue (see page 4675)

CompareValue

GreaterThanValue (see page 4675)

TValueRelationship (see page 4673)

44.4.4 Types.RT_RCDATA Constant

Pascal

```
RT_RCDATA = PChar(10);
```

C++

```
RT_RCDATA = PChar(10);
```

File

Types

Description

This is constant Types.RT_RCDATA.

45 TypeTrans Namespace

This is namespace TypeTrans.

Topics

Name	Description
Classes (see page 4680)	The following table lists classes in this documentation.
Functions (see page 4684)	The following table lists functions in this documentation.
Variables (see page 4687)	The following table lists variables in this documentation.

Classes

	Name	Description
	ETypeTransException (see page 4680)	ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.
	TTypeTranslator (see page 4680)	This is class TypeTrans.TTypeTranslator.

Functions

	Name	Description
	FloatToStrEx (see page 4684)	This is function TypeTrans.FloatToStrEx.
	GetEnumValueEx (see page 4684)	This is function TypeTrans.GetEnumValueEx.
	GetEnumValueExW (see page 4685)	This is function TypeTrans.GetEnumValueExW.
	SetEnumPropEx (see page 4685)	This is function TypeTrans.SetEnumPropEx.
	SetEnumPropExW (see page 4685)	This is function TypeTrans.SetEnumPropExW.
	StrToFloatEx (see page 4686)	This is function TypeTrans.StrToFloatEx.

Variables

Name	Description
TypeTranslator (see page 4687)	This is variable TypeTrans.TypeTranslator.

45.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ETypeTransException (see page 4680)	ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.
	TTypeTranslator (see page 4680)	This is class TypeTrans.TTypeTranslator.

45.1.1 ETypeTransException Class

ETypeTransException is the exception raised when a conversion between a SOAP type description and an Object Pascal type fails.

Class Hierarchy

Pascal

```
ETypeTransException = class(Exception);
```

C++

```
class ETypeTransException : public Exception;
```

File

TypeTrans

Description

ETypeTransException is thrown when an application attempts to convert a type in a SOAP packet to an Object Pascal type and fails for some reason.

See Also

TOPToSoapDomConvert

45.1.2 TTypeTranslator Class

Class Hierarchy

Pascal

```
TTypeTranslator = class;
```

C++

```
class TTypeTranslator;
```

File

TypeTrans

Description

This is class TypeTrans.TTypeTranslator.

45.1.2.1 TTypeTranslator Members

The following tables list the members exposed by TTypeTranslator.

Public Methods

	Name	Description
	Base64ToVar (see page 4682)	This is the overview for the Base64ToVar method overload.
	CastNativeToSoap (see page 4682)	This is CastNativeToSoap, a member of class TTypeTranslator.
	CastSoapToNative (see page 4683)	This is CastSoapToNative, a member of class TTypeTranslator.
	CastSoapToVariant (see page 4683)	This is the overview for the CastSoapToVariant method overload.
	Create (see page 4681)	This is Create, a member of class TTypeTranslator.
	Destroy (see page 4681)	This is Destroy, a member of class TTypeTranslator.

45.1.2.2 TTypeTranslator.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TTypeTranslator();
```

Description

This is Create, a member of class TTypeTranslator.

45.1.2.3 TTypeTranslator.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TTypeTranslator();
```

Description

This is Destroy, a member of class TTypeTranslator.

45.1.2.4 TTypeTranslator Methods

The methods of the TTypeTranslator class are listed here.

Public Methods

	Name	Description
	Base64ToVar (see page 4682)	This is the overview for the Base64ToVar method overload.

	CastNativeToSoap (see page 4682)	This is CastNativeToSoap, a member of class TTypeTranslator.
	CastSoapToNative (see page 4683)	This is CastSoapToNative, a member of class TTypeTranslator.
	CastSoapToVariant (see page 4683)	This is the overview for the CastSoapToVariant method overload.

45.1.2.4.1 Base64ToVar Method

This is the overview for the Base64ToVar method overload.

Overload List

	Name	Description
	TTypeTranslator.Base64ToVar (Pointer, InvString) (see page 4682)	This is Base64ToVar, a member of class TTypeTranslator.
	TTypeTranslator.Base64ToVar (Variant, InvString) (see page 4682)	This is Base64ToVar, a member of class TTypeTranslator.

45.1.2.4.1.1 TTypeTranslator.Base64ToVar Method (Pointer, InvString)

Pascal

```
procedure Base64ToVar(NatData: Pointer; const SoapData: InvString); overload;
```

C++

```
__fastcall Base64ToVar(void * NatData, const InvString SoapData);
```

Description

This is Base64ToVar, a member of class TTypeTranslator.

45.1.2.4.1.2 TTypeTranslator.Base64ToVar Method (Variant, InvString)

Pascal

```
procedure Base64ToVar(var V: Variant; const SoapData: InvString); overload;
```

C++

```
__fastcall Base64ToVar(Variant V, const InvString SoapData);
```

Description

This is Base64ToVar, a member of class TTypeTranslator.

45.1.2.4.2 TTypeTranslator.CastNativeToSoap Method

Pascal

```
procedure CastNativeToSoap(Info: PTypeInfo; var SoapData: InvString; NatData: Pointer; var
IsNull: Boolean);
```

C++

```
__fastcall CastNativeToSoap(PTypeInfo Info, InvString SoapData, void * NatData, Boolean
IsNull);
```

Description

This is CastNativeToSoap, a member of class TTypeTranslator.

45.1.2.4.3 TTypeTranslator.CastSoapToNative Method

Pascal

```
function CastSoapToNative(Info: PTypeInfo; const SoapData: InvString; NatData: Pointer;
IsNull: Boolean): Boolean;
```

C++

```
__fastcall Boolean CastSoapToNative(PTypeInfo Info, const InvString SoapData, void * NatData,
Boolean IsNull);
```

Description

This is CastSoapToNative, a member of class TTypeTranslator.

45.1.2.4.4 CastSoapToVariant Method

This is the overview for the CastSoapToVariant method overload.

Overload List

	Name	Description
	TTypeTranslator.CastSoapToVariant (PTypeInfo, InvString) (see page 4683)	This is CastSoapToVariant, a member of class TTypeTranslator.
	TTypeTranslator.CastSoapToVariant (PTypeInfo, InvString, Pointer) (see page 4683)	This is CastSoapToVariant, a member of class TTypeTranslator.

45.1.2.4.4.1 TTypeTranslator.CastSoapToVariant Method (PTypeInfo, InvString)

Pascal

```
function CastSoapToVariant(SoapInfo: PTypeInfo; const SoapData: InvString): Variant; overload;
```

C++

```
__fastcall Variant CastSoapToVariant(PTypeInfo SoapInfo, const InvString SoapData);
```

Description

This is CastSoapToVariant, a member of class TTypeTranslator.

45.1.2.4.4.2 TTypeTranslator.CastSoapToVariant Method (PTypeInfo, InvString, Pointer)

Pascal

```
procedure CastSoapToVariant(SoapInfo: PTypeInfo; const SoapData: InvString; NatData: Pointer);
overload;
```

C++

```
__fastcall CastSoapToVariant(PTypeInfo SoapInfo, const InvString SoapData, void * NatData);
```

Description

This is CastSoapToVariant, a member of class TTypeTranslator.

45.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FloatToStrEx (see page 4684)	This is function TypeTrans.FloatToStrEx.
	GetEnumValueEx (see page 4684)	This is function TypeTrans.GetEnumValueEx.
	GetEnumValueExW (see page 4685)	This is function TypeTrans.GetEnumValueExW.
	SetEnumPropEx (see page 4685)	This is function TypeTrans.SetEnumPropEx.
	SetEnumPropExW (see page 4685)	This is function TypeTrans.SetEnumPropExW.
	StrToFloatEx (see page 4686)	This is function TypeTrans.StrToFloatEx.

45.2.1 TypeTrans.FloatToStrEx Function

Pascal

```
function FloatToStrEx(Value: Extended): string;
```

C++

```
UnicodeString FloatToStrEx(Extended Value);
```

File

TypeTrans

Description

This is function TypeTrans.FloatToStrEx.

45.2.2 TypeTrans.GetEnumValueEx Function

Pascal

```
function GetEnumValueEx(TypInfo: PTypeInfo; const Name: string): Integer; deprecated;
```

C++

```
int GetEnumValueEx(PTypeInfo TypInfo, const UnicodeString Name);
```

File

TypeTrans

Description

This is function TypeTrans.GetEnumValueEx.

45.2.3 TypeTrans.GetEnumValueExW Function

Pascal

```
function GetEnumValueExW(TypInfo: PTypeInfo; const Name: InvString): Integer;
```

C++

```
int GetEnumValueExW(PTypeInfo TypInfo, const InvString Name);
```

File

TypeTrans

Description

This is function TypeTrans.GetEnumValueExW.

45.2.4 TypeTrans.SetEnumPropEx Function

Pascal

```
procedure SetEnumPropEx(Instance: TObject; PropInfo: PPropInfo; const Value: string);
deprecated;
```

C++

```
SetEnumPropEx(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypeTrans

Description

This is function TypeTrans.SetEnumPropEx.

45.2.5 TypeTrans.SetEnumPropExW Function

Pascal

```
procedure SetEnumPropExW(Instance: TObject; PropInfo: PPropInfo; const Value: InvString);
```

C++

```
SetEnumPropExW(TObject * Instance, PPropInfo PropInfo, const InvString Value);
```

File

TypeTrans

Description

This is function TypeTrans.SetEnumPropExW.

45.2.6 TypeTrans.StrToFloatEx Function

Pascal

```
function StrToFloatEx(const S: string): Extended;
```

C++

```
Extended StrToFloatEx(const UnicodeString S);
```

File

TypeTrans

Description

This is function TypeTrans.StrToFloatEx.

45.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
TypeTranslator (see page 4687)	This is variable TypeTrans.TypeTranslator.

45.3.1 TypeTrans.TypeTranslator Variable

Pascal

```
TypeTranslator: TTypeTranslator;
```

C++

```
TTypeTranslator TypeTranslator;
```

File

TypeTrans

Description

This is variable TypeTrans.TypeTranslator.

46 TypInfo Namespace

This is namespace TypInfo.

Topics

Name	Description
Classes (see page 4693)	The following table lists classes in this documentation.
Functions (see page 4701)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 4747)	The following table lists structs, records, enums in this documentation.
Types (see page 4762)	The following table lists types in this documentation.
Variables (see page 4774)	The following table lists variables in this documentation.
Constants (see page 4775)	The following table lists constants in this documentation.

Classes

	Name	Description
	EPropertyConvertError (see page 4693)	EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.
	EPropertyError (see page 4693)	EPropertyError is the exception class for unsuccessful attempts to set the value of a property.
	TPublishableVariantType (see page 4694)	TPublishableVariantType is the base class for custom Variant (see page 3421) types that surface the published properties of an object that represents the Variant (see page 3421)'s data.

Constants

Name	Description
tkAny (see page 4775)	tkAny is a set that contains all values defined by the TTypeKind (see page 4760) enumeration.
tkMethods (see page 4775)	tkMethods is a set that contains values, defined by the TTypeKind (see page 4760) enumeration, that represent a method kind.
tkProperties (see page 4776)	tkProperties is a set that contains values, defined by the TTypeKind (see page 4760) enumeration, that represent types that can be published.

Functions

	Name	Description
	FreeAndNilProperties (see page 4703)	Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).
	GetAnsiStrProp (see page 4703)	Returns the value of a component property that is of AnsiString type.
	GetAnsiStrProp (see page 4703)	Returns the value of a component property that is of AnsiString type.
	GetDynArrayProp (see page 4704)	Returns the value of a component property that is a dynamic array type.

	GetDynArrayProp (see page 4704)	Returns the value of a component property that is a dynamic array type.
	GetEnumName (see page 4705)	Returns the name of a Delphi enumerated type constant given its value.
	GetEnumProp (see page 4708)	Returns the value of a component property that is an enumerated type.
	GetEnumProp (see page 4708)	Returns the value of a component property that is an enumerated type.
	GetEnumValue (see page 4710)	Returns the value of an enumerated type constant given its string representation.
	GetFloatProp (see page 4711)	Returns the value of a component property that is a float type.
	GetFloatProp (see page 4711)	Returns the value of a component property that is a float type.
	GetInt64Prop (see page 4712)	Returns the value of a component property that is an Int64 type.
	GetInt64Prop (see page 4712)	Returns the value of a component property that is an Int64 type.
	GetInterfaceProp (see page 4712)	Returns the value of a component property that is an interface type.
	GetInterfaceProp (see page 4712)	Returns the value of a component property that is an interface type.
	GetMethodProp (see page 4713)	Returns the value of a component property that is a method type.
	GetMethodProp (see page 4713)	Returns the value of a component property that is a method type.
	GetObjectProp (see page 4714)	Returns the value of a component property that is an object type.
	GetObjectProp (see page 4714)	Returns the value of a component property that is an object type.
	GetObjectPropClass (see page 4715)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4715)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4715)	Returns the class type of a component property that is an object type.
	GetOrdProp (see page 4716)	Returns the value of a component property that is an ordinal type.
	GetOrdProp (see page 4716)	Returns the value of a component property that is an ordinal type.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfos (see page 4717)	Retrieves a list of component properties.
	GetPropList (see page 4718)	Retrieves a list of component properties.
	GetPropList (see page 4718)	Retrieves a list of component properties.
	GetPropList (see page 4718)	Retrieves a list of component properties.
	GetPropName (see page 4719)	Returns the name of a component property identified by a given PPropInfo (see page 4762).
	GetPropValue (see page 4719)	Retrieves a component's property value as a variant.
	GetPropValue (see page 4719)	Retrieves a component's property value as a variant.
	GetSetElementName (see page 4720)	Returns the string representation of a given constant.
	GetSetElementValue (see page 4721)	Returns the integer representation of a given constant identified by it's name.
	GetSetProp (see page 4722)	Returns the value of a component property that is a set type.
	GetSetProp (see page 4722)	Returns the value of a component property that is a set type.
	GetStrProp (see page 4722)	Returns the value of a component property that is a string type.
	GetStrProp (see page 4722)	Returns the value of a component property that is a string type.
	GetTypeData (see page 4723)	Returns a pointer to information about a type.
	GetTypeName (see page 4724)	Returns the name of the type from it's type information record.
	GetUnicodeStrProp (see page 4724)	Returns the value of a component property that is of UnicodeString (see page 3381) type.
	GetUnicodeStrProp (see page 4724)	Returns the value of a component property that is of UnicodeString (see page 3381) type.
	GetVariantProp (see page 4725)	Returns the value of a component property that is a variant type.
	GetVariantProp (see page 4725)	Returns the value of a component property that is a variant type.
	GetWideStrProp (see page 4726)	Returns the value of a component property that is a WideString (see page 3467) type.
	GetWideStrProp (see page 4726)	Returns the value of a component property that is a WideString (see page 3467) type.
	IsPublishedProp (see page 4727)	Determines if a specified component property is a published property.
	IsPublishedProp (see page 4727)	Determines if a specified component property is a published property.
	IsStoredProp (see page 4727)	Determines if a specified component property has the stored attribute.
	IsStoredProp (see page 4727)	Determines if a specified component property has the stored attribute.
	PropIsType (see page 4728)	Determines if a component property is of a particular data type.
	PropIsType (see page 4728)	Determines if a component property is of a particular data type.

	PropType (see page 4729)	Retrieves a component's property type.
	PropType (see page 4729)	Retrieves a component's property type.
	SamePropTypeName (see page 4729)	Checks whether the two given property names are equal.
	SetAnsiStrProp (see page 4730)	Sets the value of a component property that is of AnsiString type.
	SetAnsiStrProp (see page 4730)	Sets the value of a component property that is of AnsiString type.
	SetDynArrayProp (see page 4731)	Sets the value of a component property that is a dynamic array type.
	SetDynArrayProp (see page 4731)	Sets the value of a component property that is a dynamic array type.
	SetEnumProp (see page 4732)	Sets the value of a component property that is an enumerated type.
	SetEnumProp (see page 4732)	Sets the value of a component property that is an enumerated type.
	SetFloatProp (see page 4734)	Sets the value of a component property that is a float type.
	SetFloatProp (see page 4734)	Sets the value of a component property that is a float type.
	SetInt64Prop (see page 4735)	Sets the value of a component property that is an Int64 type.
	SetInt64Prop (see page 4735)	Sets the value of a component property that is an Int64 type.
	SetInterfaceProp (see page 4736)	Sets the value of a component property that is an Interface type.
	SetInterfaceProp (see page 4736)	Sets the value of a component property that is an Interface type.
	SetMethodProp (see page 4737)	Sets the value of a component property that is a method type.
	SetMethodProp (see page 4737)	Sets the value of a component property that is a method type.
	SetObjectProp (see page 4737)	Sets the value of a component property that is an object type.
	SetObjectProp (see page 4737)	Sets the value of a component property that is an object type.
	SetOrdProp (see page 4738)	Sets the value of a component property that is an ordinal type.
	SetOrdProp (see page 4738)	Sets the value of a component property that is an ordinal type.
	SetPropValue (see page 4739)	Sets the value of a component property using a variant value.
	SetPropValue (see page 4739)	Sets the value of a component property using a variant value.
	SetSetProp (see page 4740)	Sets the value of a component property that is a set type.
	SetSetProp (see page 4740)	Sets the value of a component property that is a set type.
	SetStrProp (see page 4741)	Sets the value of a component property that is a string type.
	SetStrProp (see page 4741)	Sets the value of a component property that is a string type.
	SetToString (see page 4742)	Returns one or more of the set values of a component's property.
	SetUnicodeStrProp (see page 4742)	Sets the value of a component property that is of UnicodeString (see page 3381) type.
	SetUnicodeStrProp (see page 4742)	Sets the value of a component property that is of UnicodeString (see page 3381) type.
	SetVariantProp (see page 4743)	Sets the value of a component property that is a variant type.
	SetVariantProp (see page 4743)	Sets the value of a component property that is a variant type.
	SetWideStrProp (see page 4744)	Sets the value of a component property that is a WideString (see page 3467) type.
	SetWideStrProp (see page 4744)	Sets the value of a component property that is a WideString (see page 3467) type.
	SortPropList (see page 4745)	Sorts a list of component properties into alphanumeric sequence.
	StringToSet (see page 4746)	Returns a bit mask of set values for a set type component property.

Structs, Records, Enums

	Name	Description
	TFloatType (see page 4747)	Enumerates floating point number types in the TTypeData (see page 4758) record.
	TIntfFlag (see page 4751)	This is record TypInfo.TIntfFlag.
	TManagedField (see page 4751)	This is record TypInfo.TManagedField.
	TMethodKind (see page 4751)	Enumerates floating point number types in the TTypeData (see page 4758) record.
	TOrdType (see page 4753)	Enumerates floating point number types in the TTypeData (see page 4758) record.
	TParamFlag (see page 4756)	TParamFlag and TParamFlags (see page 4772) indicates the attributes of a method parameter.
	TPropData (see page 4757)	This is record TypInfo.TPropData.
	TPropInfo (see page 4757)	TPropInfo is a record of component property type information.
	TTypeData (see page 4758)	TTypeData is used internally to represent type information.
	TTypeInfo (see page 4759)	Holds information for a data type as returned by TypeInfo.

	TTypeKind (see page 4760)	These types describe a type that is manipulated in the forms designer.

Types

Name	Description
PPropInfo (see page 4762)	PPropInfo points to a TPropInfo (see page 4757) record that holds component property information.
PPropList (see page 4763)	Defines a pointer to a TPropList (see page 4773) array.
PPTypeInfo (see page 4763)	Holds a pointer to a pointer to information for a data type as returned by TypeInfo.
PTypeData (see page 4764)	PTypeData points to a TTypeData (see page 4758) record that holds component property type information.
PTypeInfo (see page 4767)	Holds a pointer to information for a data type as returned by TypeInfo.
ShortStringBase (see page 4770)	Defines a fixed length string for general use in the TTypeData (see page 4758) record.
TIntfFlags (see page 4770)	Enumerates interface flag types in the TTypeData (see page 4758) record.
TIntfFlagsBase (see page 4771)	Set of interface flag types in the TTypeData (see page 4758) record.
TParamFlags (see page 4772)	This is type TypInfo.TParamFlags.
TParamFlagsBase (see page 4772)	This is type TypInfo.TParamFlagsBase.
TPropInfoProc (see page 4772)	This is type TypInfo.TPropInfoProc.
TPropList (see page 4773)	Defines an array of PPropInfo (see page 4762) values.
TTypeKinds (see page 4773)	Defines a set of TTypeKind (see page 4760) values.

Variables

Name	Description
BooleanIdents (see page 4774)	An array of string representations of true and false.
DotSep (see page 4774)	This is variable TypInfo.DotSep.

46.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EPropertyConvertError (see page 4693)	EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.
	EPropertyError (see page 4693)	EPropertyError is the exception class for unsuccessful attempts to set the value of a property.
	TPublishableVariantType (see page 4694)	TPublishableVariantType is the base class for custom Variant (see page 3421) types that surface the published properties of an object that represents the Variant (see page 3421)'s data.

46.1.1 EPropertyConvertError Class

EPropertyConvertError is the exception class for type errors when attempting to get or set the value of a property.

Class Hierarchy

Pascal

```
EPropertyConvertError = class(Exception);
```

C++

```
class EPropertyConvertError : public Exception;
```

File

TypInfo

Description

EPropertyConvertError is raised when an application tries to set a property to a value of the wrong type or to read a property that has invalid type information.

See Also

EPropertyError (see page 4693)

TPropertyEditor

46.1.2 EPropertyError Class

EPropertyError is the exception class for unsuccessful attempts to set the value of a property.

Class Hierarchy

Pascal

```
EPropertyError = class(Exception);
```

C++

```
class EPropertyError : public Exception;
```

File

TypInfo

Description

EPropertyError is raised by the SetValue (or Edit) methods of property-editor objects when they cannot set a requested property value. This usually occurs because the requested value is out of range or of an invalid data type.

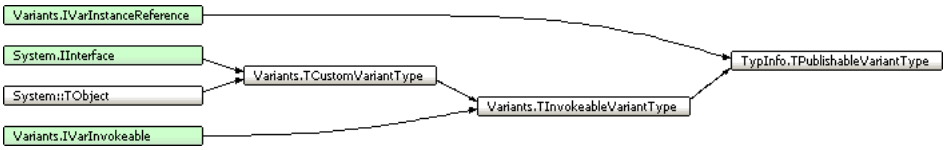
See Also

TPropertyEditor

46.1.3 TPublishableVariantType Class

TPublishableVariantType is the base class for custom Variant (see page 3421) types that surface the published properties of an object that represents the Variant (see page 3421)'s data.

Class Hierarchy



Pascal

```
TPublishableVariantType = class(TInvokeableVariantType, IVarInstanceReference);
```

C++

```
class TPublishableVariantType : public TInvokeableVariantType, public IVarInstanceReference;
```

File

TypInfo

Description

TPublishableVariantType is the base class for new data types for Variants (see page 4973) that have properties and methods. TPublishableVariantType descendants do not represent the Variants (see page 4973) themselves. Rather, they act as utility classes that the custom Variants (see page 4973) call on to perform their operations. TPublishableVariantType adds to its ancestor's ability to implement properties and methods on Variants (see page 4973) of the custom type by automatically implementing all published properties of the object it uses to store its data.

Note: To create a Variant (see page 3421) type that descends from TPublishableVariantType,

Note: 1. Map the storage of the Variant (see page 3421)'s data on to the TVarData (see page 3740) record.

Note: 2. Write (see page 3707) a method that creates instances of your custom Variant (see page 3421). This fills in the

Variant (see page 3421)'s data as defined in step 1.

Note: 3. Declare a class that descends from TPublishableVariantType.

Note: 4. Override the GetInstance (see page 4698) method to return the object that stores the custom Variant (see page 3421)'s data.

Note: 5. Implement all other required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically DoFunction, DoProcedure, BinaryOp, UnaryOp, CompareOp or Compare, Cast, CastTo, Clear, Copy (see page 3547), IsClear, LeftPromotion, and RightPromotion).

Note: 6. In the initialization section of the unit that defines your TPublishableVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the Variant (see page 3421)-handling system so that the new Variant (see page 3421) type is enabled.

Note: 7. In the finalization section of the unit that defines your TPublishableVariantType descendant, free the instance of your class. This automatically unregisters the Variant (see page 3421) type.

Note: For an example, see the VarCmplx (see page 4899) (Windows) or FMTBcd (Linux) unit, which implements a custom Variant (see page 3421) type for complex (Windows) or binary-coded decimal (Linux) numbers.

Note: Do not create your own custom Variant (see page 3421) types in C++. They are only supported in Delphi.

See Also

TInvokeableVariantType (see page 5014)

Supporting Properties and Methods in Custom Variants

46.1.3.1 TPublishableVariantType Members

The following tables list the members exposed by TPublishableVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 4993)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 4993)	Responds when the caller releases a reference to this object's interface.
V	DispInvoke (see page 4999)	Invokes a method call when the System::Variant (see page 3421) supports such operations.
V	DispInvoke (see page 5018)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5014) method.
V	FixupIdent (see page 5020)	Fixes up dispatch identifier.
A	GetInstance (see page 4698)	Returns the object instance that represents the data from a Variant (see page 3421) of the custom type.
V	LeftPromotion (see page 5000)	Indicates whether the left argument to a binary operation should be coerced to a different type.

	OlePromotion (see page 5001)	Indicates the type to which a System::Variant (see page 3421) value must be cast to make it Automation-compatible.
	QueryInterface (see page 5002)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5002)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5003)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5003)	Generates a system error indicating an invalid operation.
	RightPromotion (see page 5004)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5005)	Clears a TVarData (see page 3740) record by re-initializing it.
	SimplisticCopy (see page 5005)	Performs a simple block copy of one TVarData (see page 3740) record to another.
	VarDataCast (see page 5006)	Performs a cast from one System::Variant (see page 3421) type to another.
	VarDataCastTo (see page 5007)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5007)	Performs a cast to a System::Variant (see page 3421) type of varOleStr (see page 3869).
	VarDataClear (see page 5008)	Clears the data from a built-in System::Variant (see page 3421) type.
	VarDataCopy (see page 5008)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataCopyNoInd (see page 5009)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataFromOleStr (see page 5009)	Creates a TVarData (see page 3740) record to represent a Unicode string.
	VarDataFromStr (see page 5010)	Creates a TVarData (see page 3740) record to represent a string.
	VarDataInit (see page 5010)	Initializes a TVarData (see page 3740) record.
	VarDataIsArray (see page 5010)	Indicates whether a TVarData (see page 3740) record represents the data from a System::Variant (see page 3421) array.
	VarDataIsByRef (see page 5011)	Indicates whether a TVarData (see page 3740) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5011)	Indicates whether a TVarData (see page 3740) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5011)	Indicates whether a TVarData (see page 3740) record represents a floating-point value.
	VarDataIsNumeric (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a numeric value.
	VarDataIsOrdinal (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a ordinal value.
	VarDataIsStr (see page 5013)	Indicates whether a TVarData (see page 3740) record represents a string (normal or Unicode).
	VarDataToStr (see page 5013)	Returns a string that represents the value from a TVarData (see page 3740) record.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	BinaryOp (see page 4994)	Implements any binary operations of which the custom System::Variant (see page 3421) type is capable.
	Cast (see page 4995)	Converts the data from another System::Variant (see page 3421) type so that it represents this custom type's data format.
	CastTo (see page 4995)	Converts data in the format of this TCustomVariantType (see page 4987) to the format of another System::Variant (see page 3421) type.
	CastToOle (see page 4996)	Converts data in the format of this custom System::Variant (see page 3421) type to a format that can be used by COM Variants (see page 4973).
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.

	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 4996)	Sets the data for this custom System::Variant (see page 3421) type to a blank or unassigned value.
	Compare (see page 4997)	Implements comparisons that involve the custom System::Variant (see page 3421) type.
	CompareOp (see page 4998)	Implements any comparison operations of which the custom System::Variant (see page 3421) type is capable.
	Copy (see page 4999)	Copies the data from a System::Variant (see page 3421) of this custom type.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4990)	This is the overview for the Create constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 4991)	Frees an instance of TCustomVariantType (see page 4987).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DoFunction (see page 5024)	Implements all the methods that return values.
	DoFunction (see page 5018)	Implements all methods that return values.
	DoProcedure (see page 5025)	Implements all methods that do not return values.
	DoProcedure (see page 5019)	Implements all methods that do not return values.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetProperty (see page 4698)	Returns the value of a published property on the object that GetInstance (see page 4698) returns, so that it is available in custom Variants (see page 4973).
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	IsClear (see page 5000)	Indicates whether the data from a custom System::Variant (see page 3421) represents a blank or unassigned value.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetProperty (see page 4699)	Sets the value of a published property on the object that GetInstance (see page 4698) returns, so that it is available in custom Variants (see page 4973).
	ToString (see page 3380)	Returns a string containing the class name.
	UnaryOp (see page 5006)	Implements any unary operations of which the custom System::Variant (see page 3421) type is capable.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	VarType (see page 5014)	Indicates the System::Variant (see page 3421) type code for the custom System::Variant (see page 3421) type.

46.1.3.2 TPublishableVariantType Methods

The methods of the TPublishableVariantType class are listed here.

Protected Methods

	Name	Description
	GetInstance (see page 4698)	Returns the object instance that represents the data from a Variant (see page 3421) of the custom type.

Public Methods

	Name	Description
	GetProperty (see page 4698)	Returns the value of a published property on the object that GetInstance (see page 4698) returns, so that it is available in custom Variants (see page 4973).
	SetProperty (see page 4699)	Sets the value of a published property on the object that GetInstance (see page 4698) returns, so that it is available in custom Variants (see page 4973).

46.1.3.2.1 TPublishableVariantType.GetInstance Method

Returns the object instance that represents the data from a Variant (see page 3421) of the custom type.

Pascal

```
function GetInstance(const V: TVarData): TObject; virtual; abstract;
```

C++

```
virtual __fastcall TObject * GetInstance(const TVarData V) = 0;
```

Description

Override GetInstance to return the object that represents the data from a Variant (see page 3421) of the custom Type. TPublishableVariantType (see page 4694) implements all the published properties of that object so that they are published properties of the Variant (see page 3421).

V is the data from a Variant (see page 3421) of the custom type. This data must include a reference to an object that represents the Variant (see page 3421)'s data, and which GetInstance returns.

GetInstance is abstract or, in C++ terminology, pure virtual in TPublishableVariantType (see page 4694), which means it has no implementation. Descendant classes override this method to enable Variants (see page 4973) to publish the properties from the object that implements their data.

See Also

GetProperty (see page 4698)

SetProperty (see page 4699)

Supporting Properties and Methods in Custom Variants

46.1.3.2.2 TPublishableVariantType.GetProperty Method

Returns the value of a published property on the object that GetInstance (see page 4698) returns, so that it is available in custom Variants (see page 4973).

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
override;
```

C++

```
virtual __fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const UnicodeString
Name);
```

Description

GetProperty enables an application to read the properties of instances of the publishable Variant (see page 3421) type. It implements all the properties that are published by the object that GetInstance (see page 4698) returns.

When the application contains a line such as

```
PropValue := MyVariant.SpecialProperty;
```

(Delphi) or calls the Exec method of the Variant (see page 3421) (C++), a call is automatically generated to GetProperty. If SpecialProperty is a published property on the object that implements the Variant (see page 3421)'s data, GetProperty fills in the Dest parameter with the property value.

Dest is a TVarData (see page 3740) record (Delphi) or structure (C++) that receives the property value.

V is the TVarData (see page 3740) record or structure from the Variant (see page 3421) instance whose property is being read.

Name is the name of the property.

GetProperty returns true if it returns a property value, and raises an exception if it does not implement the specified property.

See Also

DoFunction (see page 5018)

DoProcedure (see page 5019)

GetInstance (see page 4698)

System::Variant::Exec

46.1.3.2.3 TPublishableVariantType.SetProperty Method

Sets the value of a published property on the object that GetInstance (see page 4698) returns, so that it is available in custom Variants (see page 4973).

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
override;
```

C++

```
virtual __fastcall Boolean SetProperty(const TVarData V, const UnicodeString Name, const
TVarData Value);
```

Description

SetProperty enables an application to set the properties of instances of the publishable Variant (see page 3421) type. It implements all the settable properties that are published by the object that GetInstance (see page 4698) returns.

When the application contains a line such as

```
MyVariant.SpecialProperty := NewValue;
```

(Delphi) or calls the Exec method of the Variant (see page 3421) (C++), a call is automatically generated to SetProperty. If SpecialProperty is a published settable property on the object that implements the Variant (see page 3421)'s data, SetProperty sets the property on the object included in V so that the Variant (see page 3421)'s value is changed accordingly.

V is the TVarData (see page 3740) record (Delphi) or structure (C++) from the Variant (see page 3421) instance whose

property is being set. It includes the object that the GetInstance (see page 4698) method returns.

Name is the name of the property.

Value is a TVarData (see page 3740) record or structure that contains the new property value.

GetProperty (see page 4698) returns true if it sets a property value, and raises an exception of Name does not specify a published settable property on the object that GetInstance (see page 4698) returns.

See Also

DoFunction (see page 5018)

DoProcedure (see page 5019)

GetInstance (see page 4698)

System::Variant::Exec

46.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FreeAndNilProperties (see page 4703)	Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).
	GetAnsiStrProp (see page 4703)	Returns the value of a component property that is of AnsiString type.
	GetAnsiStrProp (see page 4703)	Returns the value of a component property that is of AnsiString type.
	GetDynArrayProp (see page 4704)	Returns the value of a component property that is a dynamic array type.
	GetDynArrayProp (see page 4704)	Returns the value of a component property that is a dynamic array type.
	GetEnumName (see page 4705)	Returns the name of a Delphi enumerated type constant given its value.
	GetEnumProp (see page 4708)	Returns the value of a component property that is an enumerated type.
	GetEnumProp (see page 4708)	Returns the value of a component property that is an enumerated type.
	GetEnumValue (see page 4710)	Returns the value of an enumerated type constant given its string representation.
	GetFloatProp (see page 4711)	Returns the value of a component property that is a float type.
	GetFloatProp (see page 4711)	Returns the value of a component property that is a float type.
	GetInt64Prop (see page 4712)	Returns the value of a component property that is an Int64 type.
	GetInt64Prop (see page 4712)	Returns the value of a component property that is an Int64 type.
	GetInterfaceProp (see page 4712)	Returns the value of a component property that is an interface type.
	GetInterfaceProp (see page 4712)	Returns the value of a component property that is an interface type.
	GetMethodProp (see page 4713)	Returns the value of a component property that is a method type.
	GetMethodProp (see page 4713)	Returns the value of a component property that is a method type.
	GetObjectProp (see page 4714)	Returns the value of a component property that is an object type.
	GetObjectProp (see page 4714)	Returns the value of a component property that is an object type.
	GetObjectPropClass (see page 4715)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4715)	Returns the class type of a component property that is an object type.
	GetObjectPropClass (see page 4715)	Returns the class type of a component property that is an object type.
	GetOrdProp (see page 4716)	Returns the value of a component property that is an ordinal type.
	GetOrdProp (see page 4716)	Returns the value of a component property that is an ordinal type.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfo (see page 4717)	Retrieves information about a component's property.
	GetPropInfos (see page 4717)	Retrieves a list of component properties.
	GetPropList (see page 4718)	Retrieves a list of component properties.
	GetPropList (see page 4718)	Retrieves a list of component properties.
	GetPropList (see page 4718)	Retrieves a list of component properties.
	GetPropName (see page 4719)	Returns the name of a component property identified by a given PPropInfo (see page 4762).
	GetPropValue (see page 4719)	Retrieves a component's property value as a variant.
	GetPropValue (see page 4719)	Retrieves a component's property value as a variant.
	GetSetElementName (see page 4720)	Returns the string representation of a given constant.
	GetSetElementValue (see page 4721)	Returns the integer representation of a given constant identified by it's name.
	GetSetProp (see page 4722)	Returns the value of a component property that is a set type.
	GetSetProp (see page 4722)	Returns the value of a component property that is a set type.
	GetStrProp (see page 4722)	Returns the value of a component property that is a string type.
	GetStrProp (see page 4722)	Returns the value of a component property that is a string type.
	GetTypeData (see page 4723)	Returns a pointer to information about a type.
	GetTypeName (see page 4724)	Returns the name of the type from it's type information record.

	GetUnicodeStrProp (see page 4724)	Returns the value of a component property that is of UnicodeString (see page 3381) type.
	GetUnicodeStrProp (see page 4724)	Returns the value of a component property that is of UnicodeString (see page 3381) type.
	GetVariantProp (see page 4725)	Returns the value of a component property that is a variant type.
	GetVariantProp (see page 4725)	Returns the value of a component property that is a variant type.
	GetWideStrProp (see page 4726)	Returns the value of a component property that is a WideString (see page 3467) type.
	GetWideStrProp (see page 4726)	Returns the value of a component property that is a WideString (see page 3467) type.
	IsPublishedProp (see page 4727)	Determines if a specified component property is a published property.
	IsPublishedProp (see page 4727)	Determines if a specified component property is a published property.
	IsStoredProp (see page 4727)	Determines if a specified component property has the stored attribute.
	IsStoredProp (see page 4727)	Determines if a specified component property has the stored attribute.
	PropIsType (see page 4728)	Determines if a component property is of a particular data type.
	PropIsType (see page 4728)	Determines if a component property is of a particular data type.
	PropType (see page 4729)	Retrieves a component's property type.
	PropType (see page 4729)	Retrieves a component's property type.
	SamePropTypeName (see page 4729)	Checks whether the two given property names are equal.
	SetAnsiStrProp (see page 4730)	Sets the value of a component property that is of AnsiString type.
	SetAnsiStrProp (see page 4730)	Sets the value of a component property that is of AnsiString type.
	SetDynArrayProp (see page 4731)	Sets the value of a component property that is a dynamic array type.
	SetDynArrayProp (see page 4731)	Sets the value of a component property that is a dynamic array type.
	SetEnumProp (see page 4732)	Sets the value of a component property that is an enumerated type.
	SetEnumProp (see page 4732)	Sets the value of a component property that is an enumerated type.
	SetFloatProp (see page 4734)	Sets the value of a component property that is a float type.
	SetFloatProp (see page 4734)	Sets the value of a component property that is a float type.
	SetInt64Prop (see page 4735)	Sets the value of a component property that is an Int64 type.
	SetInt64Prop (see page 4735)	Sets the value of a component property that is an Int64 type.
	SetInterfaceProp (see page 4736)	Sets the value of a component property that is an Interface type.
	SetInterfaceProp (see page 4736)	Sets the value of a component property that is an Interface type.
	SetMethodProp (see page 4737)	Sets the value of a component property that is a method type.
	SetMethodProp (see page 4737)	Sets the value of a component property that is a method type.
	SetObjectProp (see page 4737)	Sets the value of a component property that is an object type.
	SetObjectProp (see page 4737)	Sets the value of a component property that is an object type.
	SetOrdProp (see page 4738)	Sets the value of a component property that is an ordinal type.
	SetOrdProp (see page 4738)	Sets the value of a component property that is an ordinal type.
	SetPropValue (see page 4739)	Sets the value of a component property using a variant value.
	SetPropValue (see page 4739)	Sets the value of a component property using a variant value.
	SetSetProp (see page 4740)	Sets the value of a component property that is a set type.
	SetSetProp (see page 4740)	Sets the value of a component property that is a set type.
	SetStrProp (see page 4741)	Sets the value of a component property that is a string type.
	SetStrProp (see page 4741)	Sets the value of a component property that is a string type.
	SetToString (see page 4742)	Returns one or more of the set values of a component's property.
	SetUnicodeStrProp (see page 4742)	Sets the value of a component property that is of UnicodeString (see page 3381) type.
	SetUnicodeStrProp (see page 4742)	Sets the value of a component property that is of UnicodeString (see page 3381) type.
	SetVariantProp (see page 4743)	Sets the value of a component property that is a variant type.
	SetVariantProp (see page 4743)	Sets the value of a component property that is a variant type.
	SetWideStrProp (see page 4744)	Sets the value of a component property that is a WideString (see page 3467) type.
	SetWideStrProp (see page 4744)	Sets the value of a component property that is a WideString (see page 3467) type.
	SortPropList (see page 4745)	Sorts a list of component properties into alphanumeric sequence.
	StringToSet (see page 4746)	Returns a bit mask of set values for a set type component property.

46.2.1 TypInfo.FreeAndNilProperties Function

Frees any property values that are objects and sets their value to nil (Delphi) or NULL (C++).

Pascal

```
procedure FreeAndNilProperties(AObject: TObject);
```

C++

```
FreeAndNilProperties(TObject * AObject);
```

File

TypInfo

Description

Call FreeAndNilProperties from a destructor to free all internally created objects and set the corresponding property values to nil (Delphi) or NULL (C++). FreeAndNilProperties searches the properties of AObject, identifying any that are objects. It frees those objects, and sets the corresponding property to nil (Delphi) or NULL (C++).

. By default, all persistent objects (TPersistent and descendants) are compiled with RTTI. Other TObject (see page 3354) descendants can be compiled with RTTI by using the $M+ compiler switch in Delphi or the __declspec(delphirtti) directive in C++.

Warning: Calling FreeAndNilProperties frees all object properties, which may lead to unintended results. FreeAndNilProperties can't distinguish between objects that were created by AObject (and typically stored in a class member) from objects that are created independently and which should not be freed with AObject. To avoid unintentionally freeing independently-created objects, set properties that refer to external objects to nil (Delphi) or NULL (C++) before calling FreeAndNilProperties.

See Also

FreeAndNil (see page 4201)

46.2.2 TypInfo.GetAnsiStrProp Function

Returns the value of a component property that is of AnsiString type.

Pascal

```
function GetAnsiStrProp(Instance: TObject; const PropName: string): AnsiString; overload;
function GetAnsiStrProp(Instance: TObject; PropInfo: PPropInfo): AnsiString; overload;
```

C++

```
AnsiString GetAnsiStrProp(TObject * Instance, const UnicodeString PropName);
AnsiString GetAnsiStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetAnsiStrProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is an AnsiString. An empty string is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and

GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetAnsiStrProp (see page 4730)

46.2.3 TypInfo.GetDynArrayProp Function

Returns the value of a component property that is a dynamic array type.

Pascal

```
function GetDynArrayProp(Instance: TObject; const PropName: string): Pointer; overload;
function GetDynArrayProp(Instance: TObject; PropInfo: PPropInfo): Pointer; overload;
```

C++

```
void * GetDynArrayProp(TObject * Instance, const UnicodeString PropName);
void * GetDynArrayProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetDynArrayProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is a dynamic array type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a pointer to the array.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetDynArrayProp (see page 4731)

46.2.4 TypInfo.GetEnumName Function

Returns the name of a Delphi enumerated type constant given its value.

Pascal

```
function GetEnumName(TypeInfo: PTypeInfo; Value: Integer): string;
```

C++

```
UnicodeString GetEnumName(PTypeInfo TypeInfo, int Value);
```

File

TypInfo

Description

Use GetEnumName to convert a Delphi enumerated value into the symbolic name that represents it in code.

TypeInfo is the type information record that describes the enumerated type. You can obtain the type information for a type by calling the TypeInfo function in Delphi or the __delphirtti operator in C++.

Value is the enumerated value, represented as an integer.

GetEnumName returns the string that represents the specified value.

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);
```

```
  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
This example will print useful information about any given
type including it's name, size, kind and sub-kind.
}

type
  TForm2 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
{ Defining a generic type that contains a single class (static)
  method that will print information about the type it operates on.
}
  TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
  public
    procedure PrintTypeInfo(memo: TMemo);
  end;
  TGenericString = class(TGenericClass<UnicodeString>)
  end;
  TGenericByte = class(TGenericClass<Byte>)
  end;
  TGenericDouble = class(TGenericClass<Double>)
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

{ Method defintition }
procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
var
  Info     : PTypeInfo;
  Data     : PTypeData;
  KindName : String;
  SubName  : String;
begin
  { Get type info for the "yet unknown type" }
  Info := System.TypeInfo(T);

  { There is no RTTI attached for some types like Records for example. }
  if Info <> nil then
  begin
    memo.Lines.Add('Type name: ' + Info^.Name);

    { Find out the name of an enum item from it's ordinal value }
    KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));
```

```
    memo.Lines.Add('Type kind: ' + KindName);

    Data := GetTypeData(Info);

    if Info^.Kind = tkInteger then
    begin
      { In case of integer let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
      memo.Lines.Add('Integer kind: ' + SubName);
    end;

    if Info^.Kind = tkFloat then
    begin
      { In case of float let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
      memo.Lines.Add('Float kind: ' + SubName);
    end;

    if Info^.Kind = tkDynArray then
    begin
      { Let's check out the element size  }
      memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
      memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
    end;

  end;
  memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
end;


procedure TForm2.FormCreate(Sender: TObject);
var
  GenericString : TGenericClass<UnicodeString>;
  GenericByte : TGenericClass<Byte>;
  GenericDouble : TGenericClass<Double>;
//  GenericStringArray : TGenericClass<array of string>;
begin
  { Prints type info for string type}

  GenericString := TGenericClass<UnicodeString>.Create;
  GenericString.PrintTypeInfo(Memo1);

  { Prints type info for Byte type}
  GenericByte := TGenericClass<Byte>.Create;
  GenericByte.PrintTypeInfo(Memo1);

  { Prints type info for Double type}
  GenericDouble := TGenericClass<Double>.Create;
  GenericDouble.PrintTypeInfo(Memo1);
end;

{
Expected results are:

Type name: string
Type kind: tkUString
Size of type: 4

Type name: Byte
Type kind: tkInteger
Integer kind: otUByte
Size of type: 1

Type name: Double
Type kind: tkFloat
```

```
Float kind: ftDouble
Size of type: 8

Type name: .1
Type kind: tkDynArray
Array element type name: string
Array element type size: 4
Size of type: 4
}
```

See Also

IntToIdent

GetEnumValue (see page 4710)

46.2.5 TypInfo.GetEnumProp Function

Returns the value of a component property that is an enumerated type.

Pascal

```
function GetEnumProp(Instance: TObject; const PropName: string): string; overload;
function GetEnumProp(Instance: TObject; PropInfo: PPropInfo): string; overload;
```

C++

```
UnicodeString GetEnumProp(TObject * Instance, const UnicodeString PropName);
UnicodeString GetEnumProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetEnumProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is an enumerated type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

C++ Examples:

```
/*
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
*/
```

```
void __fastcall TForm2::Button1Click(TObject *Sender)
{
    //getting the current color of the workspace
    String currentPropColor = GetEnumProp(this,"Color");
    int currentColorInt = StrToInt(currentPropColor);

    //getting the first button align enum and if different,
    //setting it to alLeft
    String currentAlignProp = GetEnumProp(Button1, "Align");
    if (currentAlignProp != "alLeft")
    {
        SetEnumProp(Button1, "Align", "alLeft");
     }

    //checking if the form background color was set.
    if(currentColorInt < 0)
    {
       currentColorInt = GetSysColor(COLOR_APPWORKSPACE);
    }

    //setting the form background color as the negative value
    //of the current background color
    SetEnumProp(this, "Color",
                IntToStr((int)(clWhite - currentColorInt)));
}

void __fastcall TForm2::Button2Click(TObject *Sender)
{
    int p[5] = {clYellow, clGreen, clRed, clBlue, clBlack};
    SetEnumProp(this, "Color", IntToStr(p[random(5)]));

    //getting the second button align enum and if different,
    //setting it to alRight
    String currentAlignProp = GetEnumProp(Button2, "Align");
    if (currentAlignProp != "alRight")
    {
        SetEnumProp(Button2, "Align", "alRight");
     }
}
```

Delphi Examples:

```
{
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
}
procedure TForm3.Button1Click(Sender: TObject);
var
  currentPropColor : String;
  currentAlignProp : String;
  currentColorInt : Integer;
begin
  //getting the current color of the workspace
  currentPropColor := GetEnumProp(Self,'Color');
  currentColorInt := StrToInt(currentPropColor);

  //getting the first button align enum and if different,
  //setting it to alLeft
  currentAlignProp := GetEnumProp(Button1, 'Align');
  if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alLeft) then
     SetEnumProp(Button1, 'Align', 'alLeft');

  //checking if the form background color was set.
    if currentColorInt < 0 then
    begin
        currentColorInt := GetSysColor(COLOR_APPWORKSPACE);
```

```
    end;

  //setting the form background color as the negative value
  // of the current background color
  currentColorInt := clWhite - currentColorInt;

  SetEnumProp(Self, 'Color', IntToStr(currentColorInt));
end;

procedure TForm3.Button2Click(Sender: TObject);
var
  p : array [0..4] of Integer;
  currentAlignProp : String;
begin
  p[0] := clYellow;
  p[1] := clRed;
  p[2] := clBlue;
  p[3] := clBlack;
  p[4] := clGreen;

    SetEnumProp(Self, 'Color', IntToStr(p[random(5)]));

  //getting the second button align enum and if different,
  //setting it to alRight
  currentAlignProp := GetEnumProp(Button2, 'Align');
  if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alRight) then
     SetEnumProp(Button2, 'Align', 'alRight');

end;
```

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetEnumProp (see page 4732)

46.2.6 TypInfo.GetEnumValue Function

Returns the value of an enumerated type constant given its string representation.

Pascal

```
function GetEnumValue(TypeInfo: PTypeInfo; const Name: string): Integer;
```

C++

```
int GetEnumValue(PTypeInfo TypeInfo, const UnicodeString Name);
```

File

TypInfo

Description

Use GetEnumValue to convert the string that represents an enumerated value into the corresponding integer that is its implemented value.

TypeInfo is the type information record that describes the enumerated type. You can obtain the type information for a type by

calling the TypeInfo function in Delphi or using the __delphirtti operator in C++.

Name is the string that is used in code to represent the enumerated value.

GetEnumValue returns the corresponding integer value.

See Also

IdentToInt

GetEnumName (see page 4705)

46.2.7 TypInfo.GetFloatProp Function

Returns the value of a component property that is a float type.

Pascal

```
function GetFloatProp(Instance: TObject; const PropName: string): Extended; overload;
function GetFloatProp(Instance: TObject; PropInfo: PPropInfo): Extended; overload;
```

C++

```
Extended GetFloatProp(TObject * Instance, const UnicodeString PropName);
Extended GetFloatProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetFloatProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is a floating point number type. 0.0 is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a float value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetFloatProp (see page 4734)

46.2.8 TypInfo.GetInt64Prop Function

Returns the value of a component property that is an Int64 type.

Pascal

```
function GetInt64Prop(Instance: TObject; const PropName: string): Int64; overload;
function GetInt64Prop(Instance: TObject; PropInfo: PPropInfo): Int64; overload;
```

C++

```
Int64 GetInt64Prop(TObject * Instance, const UnicodeString PropName);
Int64 GetInt64Prop(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetInt64Prop utilises Delphi's RTTI (Run Time see page 4297) Type Information) to return the current value of a component's property where that property is an Int64 type. 0 is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an Int64 value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetInt64Prop (see page 4735)

46.2.9 TypInfo.GetInterfaceProp Function

Returns the value of a component property that is an interface type.

Pascal

```
function GetInterfaceProp(Instance: TObject; const PropName: string): IInterface; overload;
function GetInterfaceProp(Instance: TObject; PropInfo: PPropInfo): IInterface; overload;
```

C++

```
IInterface GetInterfaceProp(TObject * Instance, const UnicodeString PropName);
IInterface GetInterfaceProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetInterfaceProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is an interface type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an interface value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetInterfaceProp (see page 4736)

46.2.10 TypInfo.GetMethodProp Function

Returns the value of a component property that is a method type.

Pascal

```
function GetMethodProp(Instance: TObject; const PropName: string): TMethod; overload;
function GetMethodProp(Instance: TObject; PropInfo: PPropInfo): TMethod; overload;
```

C++

```
TMethod GetMethodProp(TObject * Instance, const UnicodeString PropName);
TMethod GetMethodProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetMethodProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is a method type : an event routine.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a method value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetMethodProp (see page 4737)

46.2.11 TypInfo.GetObjectProp Function

Returns the value of a component property that is an object type.

Pascal

```
function GetObjectProp(Instance: TObject; const PropName: string; MinClass: TClass = nil):
TObject; overload;
function GetObjectProp(Instance: TObject; PropInfo: PPropInfo; MinClass: TClass = nil):
TObject; overload;
```

C++

```
TObject * GetObjectProp(TObject * Instance, const UnicodeString PropName, TClass MinClass =
nil);
TObject * GetObjectProp(TObject * Instance, PPropInfo PropInfo, TClass MinClass = nil);
```

File

TypInfo

Description

GetObjectProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is an object type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an object, or nil if the property is not an object type.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetObjectProp (see page 4737)

46.2.12 TypInfo.GetObjectPropClass Function

Returns the class type of a component property that is an object type.

Pascal

```
function GetObjectPropClass(Instance: TObject; const PropName: string): TClass; overload;
function GetObjectPropClass(Instance: TObject; PropInfo: PPropInfo): TClass; overload;
function GetObjectPropClass(PropInfo: PPropInfo): TClass; overload;
```

C++

```
TClass GetObjectPropClass(TObject * Instance, const UnicodeString PropName);
TClass GetObjectPropClass(TObject * Instance, PPropInfo PropInfo);
TClass GetObjectPropClass(PPropInfo PropInfo);
```

File

TypInfo

Description

GetObjectPropClass utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the class type of a component's property where that property is an object.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the class type of the property object value is returned.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

46.2.13 TypInfo.GetOrdProp Function

Returns the value of a component property that is an ordinal type.

Pascal

```
function GetOrdProp(Instance: TObject; const PropName: string): Longint; overload;
function GetOrdProp(Instance: TObject; PropInfo: PPropInfo): Longint; overload;
```

C++

```
Longint GetOrdProp(TObject * Instance, const UnicodeString PropName);
Longint GetOrdProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetOrdProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is an Integer, Char, WideChar, or Class type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as an LongInt value.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetOrdProp (see page 4738)

46.2.14 TypInfo.GetPropInfo Function

Retrieves information about a component's property.

Pascal

```
function GetPropInfo(Instance: TObject; const PropName: string; AKinds: TTypeKinds = []):
PPropInfo; overload;
function GetPropInfo(AClass: TClass; const PropName: string; AKinds: TTypeKinds = []):
PPropInfo; overload;
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string): PPropInfo; overload;
function GetPropInfo(TypeInfo: PTypeInfo; const PropName: string; AKinds: TTypeKinds):
PPropInfo; overload;
```

C++

```
PPropInfo GetPropInfo(TObject * Instance, const UnicodeString PropName, TTypeKinds AKinds =
[]);
PPropInfo GetPropInfo(TClass AClass, const UnicodeString PropName, TTypeKinds AKinds = []);
PPropInfo GetPropInfo(PTypeInfo TypeInfo, const UnicodeString PropName);
PPropInfo GetPropInfo(PTypeInfo TypeInfo, const UnicodeString PropName, TTypeKinds AKinds);
```

File

TypInfo

Description

GetPropInfo utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to retrieve a pointer to a property information record for a component.

In the first form of this function, **TypeInfo** specifies the component type whose property you want to retrieve. **PropName** specifies the property name as a string. You may optionally specify a **AKinds** set of TTypeKind (see page 4760) property value types that the property must belong to.

The second form of the function is similar to the first, with **AClass** replacing **TypeInfo** as the means of specifying the component type. **AKinds** is now a required value.

The final form of the function is like the second form, with **Instance** specifying a component instance.

In all forms of the function, the returned value is a pointer to a TPropInfo (see page 4757) value. This provides rich data about the property being investigated at run time. This enables more elegant component processing than can be achieved with hard coded component names/properties.

Note: If the specified property does not exist, a nil pointer is returned.

See Also

PTypeInfo (see page 4767)

PPropInfo (see page 4762)

TTypeKinds (see page 4773)

46.2.15 TypInfo.GetPropInfos Function

Retrieves a list of component properties.

Pascal

```
procedure GetPropInfos(TypeInfo: PTypeInfo; PropList: PPropList);
```

C++

```
GetPropInfos(PTypeInfo TypeInfo, PPropList PropList);
```

File

TypInfo

Description

GetPropInfos utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to retrieve a list of property information records for a component or component type.

The **TypeInfo** defines the component type and **PropList** defines a variable to receive the list of properties : a pointer to a TPropList (see page 4773) variable

The Delphi inbuilt TypeInfo function may be used to provide the **TypeInfo** value.

Note: If you want the count of properties returned in PropList

, use GetPropList (see page 4718) instead.

See Also

GetPropList (see page 4718)

PTypeInfo (see page 4767)

PPropList (see page 4763)

TPropList (see page 4773)

TTypeKinds (see page 4773)

46.2.16 TypInfo.GetPropList Function

Retrieves a list of component properties.

Pascal

```
function GetPropList(TypeInfo: PTypeInfo; TypeKinds: TTypeKinds; PropList: PPropList;
SortList: Boolean = True): Integer; overload;
function GetPropList(TypeInfo: PTypeInfo; out PropList: PPropList): Integer; overload;
function GetPropList(AObject: TObject; out PropList: PPropList): Integer; overload;
```

C++

```
int GetPropList(PTypeInfo TypeInfo, TTypeKinds TypeKinds, PPropList PropList, Boolean SortList
= True);
int GetPropList(PTypeInfo TypeInfo, PPropList PropList);
int GetPropList(TObject * AObject, PPropList PropList);
```

File

TypInfo

Description

GetPropList utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to retrieve a list of property information records for a component or component type.

In one form of this function, **TypeInfo** defines the component type, **PropList** defines a variable to receive the list of properties : a pointer to a TPropList (see page 4773) variable, and **TypeKinds** defines the type of properties that will be returned. Use tkAny (see page 4775) to return all types. The list may optionally be sorted by setting **SortList** true.

The Delphi inbuilt TypeInfo function may be used to provide the **TypeInfo** value.

In the other form of this function, you specify a component instance rather than component type. The **TypeKinds** and **SortList** parameters can no longer be specified.

In both forms of the function, the returned value is the number of properties in **PropList**.

Note: If you do not need the count of properties returned, use GetPropInfos (see page 4717).

See Also

GetPropInfos (see page 4717)

PTypeInfo (see page 4767)

PPropList (see page 4763)

TPropList (see page 4773)

TTypeKinds (see page 4773)

46.2.17 TypInfo.GetPropName Function

Returns the name of a component property identified by a given PPropInfo (see page 4762).

Pascal

```
function GetPropName(PropInfo: PPropInfo): string;
```

C++

```
UnicodeString GetPropName(PPropInfo PropInfo);
```

File

TypInfo

Description

Use GetPropName to obtain the qualified property name of a component's property identified by a given PPropInfo (see page 4762) parameter.

PropInfo is the type information record that describes the property. You can obtain the type information for a component property by calling the GetPropInfo (see page 4717) function.

See Also

GetPropInfo (see page 4717)

TPropInfo (see page 4757)

46.2.18 TypInfo.GetPropValue Function

Retrieves a component's property value as a variant.

Pascal

```
function GetPropValue(Instance: TObject; const PropName: string; PreferStrings: Boolean =
True): Variant; overload;
function GetPropValue(Instance: TObject; PropInfo: PPropInfo; PreferStrings: Boolean = True):
Variant; overload;
```

C++

```
Variant GetPropValue(TObject * Instance, const UnicodeString PropName, Boolean PreferStrings =
True);
Variant GetPropValue(TObject * Instance, PPropInfo PropInfo, Boolean PreferStrings = True);
```

File

TypInfo

Description

GetPropValue utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to retrieve a a component property as a variant.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the returned value the property value as a variant.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

PTypeInfo (see page 4767)

PPropInfo (see page 4762)

TTypeKinds (see page 4773)

SetPropValue (see page 4739)

46.2.19 TypInfo.GetSetElementName Function

Returns the string representation of a given constant.

Pascal

```
function GetSetElementName(TypeInfo: PTypeInfo; Value: Integer): string;
```

C++

```
UnicodeString GetSetElementName(PTypeInfo TypeInfo, int Value);
```

File

TypInfo

Description

Use GetSetElementName to convert a constant into the symbolic name that represents it in code.

TypeInfo is the type information record that describes the type for which the constant has a meaning. You can obtain the type information for a type by calling the TypeInfo function in Delphi or the __delphirtti operator in C++.

Value is the constant, represented as an integer.

GetSetElementName returns the string that represents the specified value.

If TypeInfo describes an Integer type, Value is assumed to be a simple integer and is converted to its numerical string representation. If TypeInfo describes a char type, a string containing the Pascal character notation of Value is returned. If TypeInfo describes an enumerated type, GetEnumName (see page 4705) returns the string representation of Value.

See Also

GetSetElementValue (see page 4721)

GetEnumName (see page 4705)

GetEnumValue (see page 4710)

TTypeInfo (see page 4759)

46.2.20 TypInfo.GetSetElementValue Function

Returns the integer representation of a given constant identified by it's name.

Pascal

```
function GetSetElementValue(TypeInfo: PTypeInfo; const Name: string): Integer;
```

C++

```
int GetSetElementValue(PTypeInfo TypeInfo, const UnicodeString Name);
```

File

TypInfo

Description

Use GetSetElementValue to convert a constant identified by it's symbolic name to an integer.

TypeInfo is the type information record that describes the type for which the constant has a meaning. You can obtain the type information for a type by calling the TypeInfo function in Delphi or the __delphirtti operator in C++.

Value is the constant, represented as a string.

GetSetElementValue returns the integer that represents the specified value.

If TypeInfo describes an Integer type, Value is assumed to be a string representation of an integer and is converted to an integer. If TypeInfo describes a char type, an integer containing the character code is returned. If TypeInfo describes an enumerated type, GetEnumValue (see page 4710) returns the integer representation of Value.

See Also

GetSetElementName (see page 4720)

GetEnumName (see page 4705)

GetEnumValue (see page 4710)

TTypeInfo (see page 4759)

46.2.21 TypInfo.GetSetProp Function

Returns the value of a component property that is a set type.

Pascal

```
function GetSetProp(Instance: TObject; const PropName: string; Brackets: Boolean = False):
string; overload;
function GetSetProp(Instance: TObject; PropInfo: PPropInfo; Brackets: Boolean = False):
string; overload;
```

C++

```
UnicodeString GetSetProp(TObject * Instance, const UnicodeString PropName, Boolean Brackets =
False);
UnicodeString GetSetProp(TObject * Instance, PPropInfo PropInfo, Boolean Brackets = False);
```

File

TypInfo

Description

GetSetProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is a set of values type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the list if set values is returned as a comma separated string. You may optionally set **Brackets** true so as to enclose this within square brackets, as shown in the IDE for the property.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetSetProp (see page 4740)

46.2.22 TypInfo.GetStrProp Function

Returns the value of a component property that is a string type.

Pascal

```
function GetStrProp(Instance: TObject; const PropName: string): string; overload;
function GetStrProp(Instance: TObject; PropInfo: PPropInfo): string; overload;
```

C++

```
UnicodeString GetStrProp(TObject * Instance, const UnicodeString PropName);
UnicodeString GetStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetStrProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is a String, LongString or WideString (see page 3467) type. An empty string is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetStrProp (see page 4741)

46.2.23 TypInfo.GetTypeData Function

Returns a pointer to information about a type.

Pascal

```
function GetTypeData(TypeInfo: PTypeInfo): PTypeData;
```

C++

```
PTypeData GetTypeData(PTypeInfo TypeInfo);
```

File

TypInfo

Description

GetTypeData utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a pointer to TTypeData (see page 4758) record.

The TTypeData (see page 4758) record is a variant type, but does not contain the variant as a tag. To identify which variant applies to a particular type, you should retrieve the TTypeKind (see page 4760) value using the TypeInfo routine.

The TTypeData (see page 4758) record holds meta data about a property. For example, the name of the unit a class type is stored in.

See Also

PTypeData (see page 4764)

TTypeData (see page 4758)

TTypeKind (see page 4760)

46.2.24 TypInfo.GetTypeName Function

Returns the name of the type from it's type information record.

Pascal

```
function GetTypeName(TypeInfo: PTypeInfo): string;
```

C++

```
UnicodeString GetTypeName(PTypeInfo TypeInfo);
```

File

TypInfo

Description

GetTypeName utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the name of a type as defined in the code.

TypeInfo is the type information record that describes the type for which the name must be obtained. You can obtain the type information for a type by calling the TypeInfo function in Delphi or the __delphirtti operator in C++.

See Also

TTypeInfo (see page 4759)

GetTypeData (see page 4723)

46.2.25 TypInfo.GetUnicodeStrProp Function

Returns the value of a component property that is of UnicodeString (see page 3381) type.

Pascal

```
function GetUnicodeStrProp(Instance: TObject; const PropName: string): UnicodeString; overload;
function GetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo): UnicodeString; overload;
```

C++

```
UnicodeString GetUnicodeStrProp(TObject * Instance, const UnicodeString PropName);
UnicodeString GetUnicodeStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetUnicodeStrProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is a UnicodeString (see page 3381). An empty string is returned if the property is not of this type.

In one form of this function, the property is looked up by using the data in the **PropInfo** parameter. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a string.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetUnicodeStrProp (see page 4742)

46.2.26 TypInfo.GetVariantProp Function

Returns the value of a component property that is a variant type.

Pascal

```
function GetVariantProp(Instance: TObject; const PropName: string): Variant; overload;
function GetVariantProp(Instance: TObject; PropInfo: PPropInfo): Variant; overload;
```

C++

```
Variant GetVariantProp(TObject * Instance, const UnicodeString PropName);
Variant GetVariantProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetVariantProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a

component's property that is a variant type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a variant.

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown. If the property is not a variant type, an EVariantBadVarTypeError (see page 4980) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetVariantProp (see page 4743)

46.2.27 TypInfo.GetWideStrProp Function

Returns the value of a component property that is a WideString (see page 3467) type.

Pascal

```
function GetWideStrProp(Instance: TObject; const PropName: string): WideString; overload;
function GetWideStrProp(Instance: TObject; PropInfo: PPropInfo): WideString; overload;
```

C++

```
BSTR GetWideStrProp(TObject * Instance, const UnicodeString PropName);
BSTR GetWideStrProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

GetWideStrProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the current value of a component's property where that property is a WideString (see page 3467) type. A string or LongString type is returned via a call to the GetWideStrProp routine. Otherwise, for other types, an empty string is returned if the property is not of this type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the current value of the property is returned as a WideString (see page 3467).

The intrinsic value of this function is to allow retrieval of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use GetPropValue (see page 4719) to retrieve a property value where the type is unknown - it is returned in a Variant (see page 3421) type.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropList (see page 4718)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

SetWideStrProp (see page 4744)

46.2.28 TypInfo.IsPublishedProp Function

Determines if a specified component property is a published property.

Pascal

```
function IsPublishedProp(Instance: TObject; const PropName: string): Boolean; overload;
function IsPublishedProp(AClass: TClass; const PropName: string): Boolean; overload;
```

C++

```
Boolean IsPublishedProp(TObject * Instance, const UnicodeString PropName);
Boolean IsPublishedProp(TClass AClass, const UnicodeString PropName);
```

File

TypInfo

Description

IsPublishedProp returns true if the specified property is published. Delphi's RTTI (Run Time (see page 4297) Type Information) facility requires that properties are published.

In one form of this function, the component is defined by the **TClass (see page 3785)** parameter.

In the other form of this function, an **Instance** of the component is used.

In both forms of the function, **PropName** specifies the property as a string.

See Also

GetPropInfo (see page 4717)

GetPropValue (see page 4719)

46.2.29 TypInfo.IsStoredProp Function

Determines if a specified component property has the stored attribute.

Pascal

```
function IsStoredProp(Instance: TObject; const PropName: string): Boolean; overload;
function IsStoredProp(Instance: TObject; PropInfo: PPropInfo): Boolean; overload;
```

C++

```
Boolean IsStoredProp(TObject * Instance, const UnicodeString PropName);
Boolean IsStoredProp(TObject * Instance, PPropInfo PropInfo);
```

File

TypInfo

Description

IsStoredProp returns true if the specified property is saved in the form file. The stored attribute is set true by default : only those that have the keyword stored defined as false do not.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the value of the stored attribute is returned.

See Also

GetPropInfo (see page 4717)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

46.2.30 TypInfo.PropIsType Function

Determines if a component property is of a particular data type.

Pascal

```
function PropIsType(Instance: TObject; const PropName: string; TypeKind: TTypeKind): Boolean;
overload;
function PropIsType(AClass: TClass; const PropName: string; TypeKind: TTypeKind): Boolean;
overload;
```

C++

```
Boolean PropIsType(TObject * Instance, const UnicodeString PropName, TTypeKind TypeKind);
Boolean PropIsType(TClass AClass, const UnicodeString PropName, TTypeKind TypeKind);
```

File

TypInfo

Description

PropIsType returns true if the property is of the specified data type.

In the first form of this function, **AClass** specifies the component class whose property you want to look at.

The second form of the function a class **Instance** replaces **AClass**.

In both forms of the function, **PropName** specifies the property name as a string. If the property has the **TypeKind** data type, then true is returned. Otherwise false is returned.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropInfo (see page 4717)

GetPropValue (see page 4719)

PPropInfo (see page 4762)

TTypeKinds (see page 4773)

46.2.31 TypInfo.PropType Function

Retrieves a component's property type.

Pascal

```
function PropType(Instance: TObject; const PropName: string): TTypeKind; overload;
function PropType(AClass: TClass; const PropName: string): TTypeKind; overload;
```

C++

```
TTypeKind PropType(TObject * Instance, const UnicodeString PropName);
TTypeKind PropType(TClass AClass, const UnicodeString PropName);
```

File

TypInfo

Description

PropType utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return the type of a component property in a TTypeKind (see page 4760) value.

In the first form of this function, **AClass** specifies the component class whose property you want to retrieve. **PropName** specifies the property name as a string.

The second form of the function is similar to the first, with **Instance** replacing **AClass** as the means of specifying the component.

In both forms of the function, the returned value is a TTypeKind (see page 4760) value.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

PTypeInfo (see page 4767)

PPropInfo (see page 4762)

TTypeKind (see page 4760)

46.2.32 TypInfo.SamePropTypeName Function

Checks whether the two given property names are equal.

Pascal

```
function SamePropTypeName(const Name1: ShortString; const Name2: ShortString): Boolean;
```

C++

```
Boolean SamePropTypeName(const ShortString Name1, const ShortString Name2);
```

File

TypInfo

Description

SamePropTypeName compares the two given property names represented as ShortStrings. Use this function rather than the equality operator, because the internal format in which these names are stored might be changed in the future.

See Also

GetPropName (see page 4719)

TPropInfo (see page 4757)

46.2.33 TypInfo.SetAnsiStrProp Function

Sets the value of a component property that is of AnsiString type.

Pascal

```
procedure SetAnsiStrProp(Instance: TObject; const PropName: string; const Value: AnsiString);
overload;
procedure SetAnsiStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: AnsiString);
overload;
```

C++

```
SetAnsiStrProp(TObject * Instance, const UnicodeString PropName, const AnsiString Value);
SetAnsiStrProp(TObject * Instance, PPropInfo PropInfo, const AnsiString Value);
```

File

TypInfo

Description

SetAnsiStrProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is an AnsiString.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetAnsiStrProp (see page 4703)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.34 TypInfo.SetDynArrayProp Function

Sets the value of a component property that is a dynamic array type.

Pascal

```
procedure SetDynArrayProp(Instance: TObject; const PropName: string; const Value: Pointer);
overload;
procedure SetDynArrayProp(Instance: TObject; PropInfo: PPropInfo; const Value: Pointer);
overload;
```

C++

```
SetDynArrayProp(TObject * Instance, const UnicodeString PropName, const void * Value);
SetDynArrayProp(TObject * Instance, PPropInfo PropInfo, const void * Value);
```

File

TypInfo

Description

SetDynArrayProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is a dynamic array type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetEnumProp (see page 4708)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.35 TypInfo.SetEnumProp Function

Sets the value of a component property that is an enumerated type.

Pascal

```
procedure SetEnumProp(Instance: TObject; const PropName: string; const Value: string);
overload;
procedure SetEnumProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetEnumProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
SetEnumProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

SetEnumProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is an enumerated type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

C++ Examples:

```
/*
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
*/
void __fastcall TForm2::Button1Click(TObject *Sender)
{
    //getting the current color of the workspace
    String currentPropColor = GetEnumProp(this,"Color");
    int currentColorInt = StrToInt(currentPropColor);

    //getting the first button align enum and if different,
    //setting it to alLeft
    String currentAlignProp = GetEnumProp(Button1, "Align");
    if (currentAlignProp != "alLeft")
    {
         SetEnumProp(Button1, "Align", "alLeft");
     }
```

```
    //checking if the form background color was set.
    if(currentColorInt < 0)
    {
        currentColorInt = GetSysColor(COLOR_APPWORKSPACE);
    }

    //setting the form background color as the negative value
    //of the current background color
    SetEnumProp(this, "Color",
                IntToStr((int)(clWhite - currentColorInt)));
}

void __fastcall TForm2::Button2Click(TObject *Sender)
{
    int p[5] = {clYellow, clGreen, clRed, clBlue, clBlack};
    SetEnumProp(this, "Color", IntToStr(p[random(5)]));

    //getting the second button align enum and if different,
    //setting it to alRight
    String currentAlignProp = GetEnumProp(Button2, "Align");
    if (currentAlignProp != "alRight")
    {
         SetEnumProp(Button2, "Align", "alRight");
     }
}
```

Delphi Examples:

```
{
This example shows how to use GetEnumProp and SetEnumProp in order to
check for visual components properties and to change them.
This example uses two buttons, Button1 and Button2.
}
procedure TForm3.Button1Click(Sender: TObject);
var
  currentPropColor : String;
  currentAlignProp : String;
  currentColorInt : Integer;
begin
  //getting the current color of the workspace
  currentPropColor := GetEnumProp(Self,'Color');
  currentColorInt := StrToInt(currentPropColor);

  //getting the first button align enum and if different,
  //setting it to alLeft
  currentAlignProp := GetEnumProp(Button1, 'Align');
  if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alLeft) then
     SetEnumProp(Button1, 'Align', 'alLeft');

  //checking if the form background color was set.
    if currentColorInt < 0 then
    begin
        currentColorInt := GetSysColor(COLOR_APPWORKSPACE);
    end;

  //setting the form background color as the negative value
  // of the current background color
  currentColorInt := clWhite - currentColorInt;

  SetEnumProp(Self, 'Color', IntToStr(currentColorInt));
end;

procedure TForm3.Button2Click(Sender: TObject);
var
  p : array [0..4] of Integer;
  currentAlignProp : String;
begin
```

```
    p[0] := clYellow;
    p[1] := clRed;
    p[2] := clBlue;
    p[3] := clBlack;
    p[4] := clGreen;

      SetEnumProp(Self, 'Color', IntToStr(p[random(5)]));

    //getting the second button align enum and if different,
    //setting it to alRight
    currentAlignProp := GetEnumProp(Button2, 'Align');
    if GetEnumValue(TypeInfo(TAlign), currentAlignProp) <> Ord(alRight) then
       SetEnumProp(Button2, 'Align', 'alRight');

  end;
```

See Also

GetEnumProp (see page 4708)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.36 TypInfo.SetFloatProp Function

Sets the value of a component property that is a float type.

Pascal

```
procedure SetFloatProp(Instance: TObject; const PropName: string; const Value: Extended);
overload;
procedure SetFloatProp(Instance: TObject; PropInfo: PPropInfo; const Value: Extended);
overload;
```

C++

```
SetFloatProp(TObject * Instance, const UnicodeString PropName, const Extended Value);
SetFloatProp(TObject * Instance, PPropInfo PropInfo, const Extended Value);
```

File

TypInfo

Description

SetFloatProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is a floating point number.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when

using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetFloatProp (see page 4711)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.37 TypInfo.SetInt64Prop Function

Sets the value of a component property that is an Int64 type.

Pascal

```
procedure SetInt64Prop(Instance: TObject; const PropName: string; const Value: Int64);
overload;
procedure SetInt64Prop(Instance: TObject; PropInfo: PPropInfo; const Value: Int64); overload;
```

C++

```
SetInt64Prop(TObject * Instance, const UnicodeString PropName, const Int64 Value);
SetInt64Prop(TObject * Instance, PPropInfo PropInfo, const Int64 Value);
```

File

TypInfo

Description

SetInt64Prop utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is an Int64 number.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetInt64Prop (see page 4712)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.38 TypInfo.SetInterfaceProp Function

Sets the value of a component property that is an Interface type.

Pascal

```
procedure SetInterfaceProp(Instance: TObject; const PropName: string; const Value:
IInterface); overload;
procedure SetInterfaceProp(Instance: TObject; PropInfo: PPropInfo; const Value: IInterface);
overload;
```

C++

```
SetInterfaceProp(TObject * Instance, const UnicodeString PropName, const IInterface Value);
SetInterfaceProp(TObject * Instance, PPropInfo PropInfo, const IInterface Value);
```

File

TypInfo

Description

SetInterfaceProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is an Interface.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetInterfaceProp (see page 4712)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.39 TypInfo.SetMethodProp Function

Sets the value of a component property that is a method type.

Pascal

```
procedure SetMethodProp(Instance: TObject; const PropName: string; const Value: TMethod);
overload;
procedure SetMethodProp(Instance: TObject; PropInfo: PPropInfo; const Value: TMethod);
overload;
```

C++

```
SetMethodProp(TObject * Instance, const UnicodeString PropName, const TMethod Value);
SetMethodProp(TObject * Instance, PPropInfo PropInfo, const TMethod Value);
```

File

TypInfo

Description

SetMethodProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is an Event (Method property).

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetMethodProp (see page 4713)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.40 TypInfo.SetObjectProp Function

Sets the value of a component property that is an object type.

Pascal

```
procedure SetObjectProp(Instance: TObject; const PropName: string; Value: TObject); overload;
procedure SetObjectProp(Instance: TObject; PropInfo: PPropInfo; Value: TObject; ValidateClass:
Boolean = True); overload;
```

C++

```
SetObjectProp(TObject * Instance, const UnicodeString PropName, TObject * Value);
SetObjectProp(TObject * Instance, PPropInfo PropInfo, TObject * Value, Boolean ValidateClass =
True);
```

File

TypInfo

Description

SetObjectProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is an object.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetObjectProp (see page 4714)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.41 TypInfo.SetOrdProp Function

Sets the value of a component property that is an ordinal type.

Pascal

```
procedure SetOrdProp(Instance: TObject; const PropName: string; Value: Longint); overload;
procedure SetOrdProp(Instance: TObject; PropInfo: PPropInfo; Value: Longint); overload;
```

C++

```
SetOrdProp(TObject * Instance, const UnicodeString PropName, Longint Value);
SetOrdProp(TObject * Instance, PPropInfo PropInfo, Longint Value);
```

File

TypInfo

Description

SetOrdProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is an Integer, Char, WideChar, or Class type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetOrdProp (see page 4716)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.42 TypInfo.SetPropValue Function

Sets the value of a component property using a variant value.

Pascal

```
procedure SetPropValue(Instance: TObject; const PropName: string; const Value: Variant);
overload;
procedure SetPropValue(Instance: TObject; PropInfo: PPropInfo; const Value: Variant); overload;
```

C++

```
SetPropValue(TObject * Instance, const UnicodeString PropName, const Variant Value);
SetPropValue(TObject * Instance, PPropInfo PropInfo, const Variant Value);
```

File

TypInfo

Description

SetPropValue utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property using a variant value.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**. Note that only certain property assignments are possible using variants. It is normally be used when using GetPropValue (see page 4719) from another property.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetPropValue (see page 4719)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

PPropInfo (see page 4762)

46.2.43 TypInfo.SetSetProp Function

Sets the value of a component property that is a set type.

Pascal

```
procedure SetSetProp(Instance: TObject; const PropName: string; const Value: string); overload;
procedure SetSetProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetSetProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
SetSetProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

SetSetProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is a set type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**. You specify the set of values as a comma separated string. You may optionally enclose these in square brackets. This allows the returned value from one GetSetProp (see page 4722) call to be used as the assignment value in a SetSetProp call.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetSetProp (see page 4722)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.44 TypInfo.SetStrProp Function

Sets the value of a component property that is a string type.

Pascal

```
procedure SetStrProp(Instance: TObject; const PropName: string; const Value: string); overload;
procedure SetStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: string); overload;
```

C++

```
SetStrProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
SetStrProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

SetStrProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is a String, LongString or WideString (see page 3467) type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetStrProp (see page 4722)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.45 TypInfo.SetToString Function

Returns one or more of the set values of a component's property.

Pascal

```
function SetToString(PropInfo: PPropInfo; Value: Integer; Brackets: Boolean = False): string;
```

C++

```
UnicodeString SetToString(PPropInfo PropInfo, int Value, Boolean Brackets = False);
```

File

TypInfo

Description

SetToString utilises Delphi's RTTI (Run Time see page 4297) Type Information) to retrieve one or more of the values of a component property that is of the set type.

The property to interrogate is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

Value defines a bit mask that selects the possible values of this property. Each bit corresponds to a set value. The low order bit corresponds to the first set value. A string is returned with the selected set values as a comma separated value string. Note that the current property set values are not taken into account - the set values selected by **Value** are those in the complete set.

You may optionally set **Brackets** true so as to enclose this returned string within square brackets, as shown in the IDE for the property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetSetProp (see page 4722)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

PPropInfo (see page 4762)

SetPropValue (see page 4739)

SetSetProp (see page 4740)

46.2.46 TypInfo.SetUnicodeStrProp Function

Sets the value of a component property that is of UnicodeString (see page 3381) type.

Pascal

```
procedure SetUnicodeStrProp(Instance: TObject; const PropName: string; const Value:
UnicodeString); overload;
procedure SetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo; const Value:
UnicodeString); overload;
```

C++

```
SetUnicodeStrProp(TObject * Instance, const UnicodeString PropName, const UnicodeString Value);
SetUnicodeStrProp(TObject * Instance, PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

SetUnicodeStrProp uses Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is a UnicodeString (see page 3381).

In one form of this function, the property is looked up by using the data in the **PropInfo** parameter. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetUnicodeStrProp (see page 4724)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.47 TypInfo.SetVariantProp Function

Sets the value of a component property that is a variant type.

Pascal

```
procedure SetVariantProp(Instance: TObject; const PropName: string; const Value: Variant);
overload;
procedure SetVariantProp(Instance: TObject; PropInfo: PPropInfo; const Value: Variant);
overload;
```

C++

```
SetVariantProp(TObject * Instance, const UnicodeString PropName, const Variant Value);
SetVariantProp(TObject * Instance, PPropInfo PropInfo, const Variant Value);
```

File

TypInfo

Description

SetVariantProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is a variant type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetVariantProp (see page 4725)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.48 TypInfo.SetWideStrProp Function

Sets the value of a component property that is a WideString (see page 3467) type.

Pascal

```
procedure SetWideStrProp(Instance: TObject; const PropName: string; const Value: WideString);
overload;
procedure SetWideStrProp(Instance: TObject; PropInfo: PPropInfo; const Value: WideString);
overload;
```

C++

```
SetWideStrProp(TObject * Instance, const UnicodeString PropName, const BSTR Value);
SetWideStrProp(TObject * Instance, PPropInfo PropInfo, const BSTR Value);
```

File

TypInfo

Description

SetWideStrProp utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to set the value of a component's property where that property is a WideString (see page 3467) type.

In one form of this function, the object **Instance**'s property is defined by a **PropInfo** record. GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record.

In the other form of this function, the name of the property, **PropName**, is explicitly given.

In both forms of the function, the new property value is defined by **Value**.

The intrinsic value of this function is to allow setting of component properties without the need to hard code the property name. This allows mass visual component processing at run time.

Use SetPropValue (see page 4739) to set a property value where the type is unknown. This would normally be used when using GetPropValue (see page 4719) from another property.

Note: If the specified property does not exist, or is not a published property, an EPropertyError (see page 4693) exception is thrown.

See Also

GetWideStrProp (see page 4726)

GetPropInfo (see page 4717)

GetPropList (see page 4718)

SetPropValue (see page 4739)

PPropInfo (see page 4762)

46.2.49 TypInfo.SortPropList Function

Sorts a list of component properties into alphanumeric sequence.

Pascal

```
procedure SortPropList(PropList: PPropList; PropCount: Integer);
```

C++

```
SortPropList(PPropList PropList, int PropCount);
```

File

TypInfo

Description

SortPropList is used to sort a PPropList (see page 4763) that was originally created in natural rather than sorted sequence (by GetPropList (see page 4718)).

PropList defines a pointer to the array of TPropInfo (see page 4757) pointers (normally obtained using GetPropList (see page 4718)). **Count** specifies the number of TPropList (see page 4773)

See Also

GetPropList (see page 4718)

PTypeInfo (see page 4767)

PPropList (see page 4763)

PPropInfo (see page 4762)

TPropInfo (see page 4757)

46.2.50 TypInfo.StringToSet Function

Returns a bit mask of set values for a set type component property.

Pascal

```
function StringToSet(PropInfo: PPropInfo; const Value: string): Integer;
```

C++

```
int StringToSet(PPropInfo PropInfo, const UnicodeString Value);
```

File

TypInfo

Description

StringToSet utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a bit mask of set indices for component's property where that property is a set type.

The property is defined by the **PropInfo** pointer. (GetPropInfo (see page 4717) and GetPropList (see page 4718) can be used to obtain such a record).

Value defines a comma separated string of set values the property may have.

The resulting integer value is a bit mask. The 0 bit relates to the 1st set value for the property. If that property value is included in the **Value** string, it is set to 1, otherwise 0. This continues for bits 1, 2 and so on referring to set positions 2, 3 and so on.

See Also

GetPropList (see page 4718)

SetToString (see page 4742)

PPropInfo (see page 4762)

TPropInfo (see page 4757)

46.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFloatType (see page 4747)	Enumerates floating point number types in the TTypeData (see page 4758) record.
	TIntfFlag (see page 4751)	This is record TypInfo.TIntfFlag.
	TMethodKind (see page 4751)	Enumerates floating point number types in the TTypeData (see page 4758) record.
	TOrdType (see page 4753)	Enumerates floating point number types in the TTypeData (see page 4758) record.
	TParamFlag (see page 4756)	TParamFlag and TParamFlags (see page 4772) indicates the attributes of a method parameter.
	TTypeKind (see page 4760)	These types describe a type that is manipulated in the forms designer.

Records

	Name	Description
	TManagedField (see page 4751)	This is record TypInfo.TManagedField.
	TPropData (see page 4757)	This is record TypInfo.TPropData.
	TPropInfo (see page 4757)	TPropInfo is a record of component property type information.
	TTypeData (see page 4758)	TTypeData is used internally to represent type information.
	TTypeInfo (see page 4759)	Holds information for a data type as returned by TypeInfo.

46.3.1 TypInfo.TFloatType Enumeration

Enumerates floating point number types in the TTypeData (see page 4758) record.

Pascal

```
TFloatType = (
  ftSingle,
  ftDouble,
  ftExtended,
  ftComp,
  ftCurr
);
```

C++

```
enum TFloatType {
  ftSingle,
  ftDouble,
  ftExtended,
  ftComp,
  ftCurr
};
```

File

TypInfo

Description

The GetTypeData (see page 4723) routine utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a

pointer to TTypeData (see page 4758) record that describes a component property data type.

TTypeData (see page 4758) is a variant record : the layout depends on the property TTypeKind (see page 4760) value. When it is tkFloat, the data is one of the TFloatType enumerated values.

The TFloatType value meanings are as follows :

Value	Meaning
ftSingle	Single data type. (-1.5 x 10^45 .. 3.4 x 10^38)
ftDouble	Double data type. (-5.0 x 10^324 .. 1.7 x 10^308)
ftExtended	Extended data type. (-3.6 x 10^4951 .. 1.1 x 10^4932)
ftComp	Comp data type. (-2^63+1 .. 2^63-1)
ftCurr	Currency (see page 3235) data type. (20 significant digits, 4 decimal places)

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);

  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
This example will print useful information about any given
type including it's name, size, kind and sub-kind.
}

type
  TForm2 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
```

```
    { Public declarations }
  end;
{ Defining a generic type that contains a single class (static)
  method that will print information about the type it operates on.
}
  TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
  public
    procedure PrintTypeInfo(memo: TMemo);
  end;
  TGenericString = class(TGenericClass<UnicodeString>)
  end;
  TGenericByte = class(TGenericClass<Byte>)
  end;
  TGenericDouble = class(TGenericClass<Double>)
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

{ Method defintition }
procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
var
  Info     : PTypeInfo;
  Data     : PTypeData;
  KindName : String;
  SubName  : String;
begin
  { Get type info for the "yet unknown type" }
  Info := System.TypeInfo(T);

  { There is no RTTI attached for some types like Records for example. }
  if Info <> nil then
  begin
    memo.Lines.Add('Type name: ' + Info^.Name);

    { Find out the name of an enum item from it's ordinal value }
    KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));

    memo.Lines.Add('Type kind: ' + KindName);

    Data := GetTypeData(Info);

    if Info^.Kind = tkInteger then
    begin
      { In case of integer let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
      memo.Lines.Add('Integer kind: ' + SubName);
    end;

    if Info^.Kind = tkFloat then
    begin
      { In case of float let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
      memo.Lines.Add('Float kind: ' + SubName);
    end;

    if Info^.Kind = tkDynArray then
    begin
      { Let's check out the element size  }
      memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
      memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
    end;
```

```
    end;
    memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
  end;


  procedure TForm2.FormCreate(Sender: TObject);
  var
    GenericString : TGenericClass<UnicodeString>;
    GenericByte : TGenericClass<Byte>;
    GenericDouble : TGenericClass<Double>;
  //  GenericStringArray : TGenericClass<array of string>;
  begin
    { Prints type info for string type}

    GenericString := TGenericClass<UnicodeString>.Create;
    GenericString.PrintTypeInfo(Memo1);

    { Prints type info for Byte type}
    GenericByte := TGenericClass<Byte>.Create;
    GenericByte.PrintTypeInfo(Memo1);

    { Prints type info for Double type}
    GenericDouble := TGenericClass<Double>.Create;
    GenericDouble.PrintTypeInfo(Memo1);
  end;

  {
  Expected results are:

  Type name: string
  Type kind: tkUString
  Size of type: 4

  Type name: Byte
  Type kind: tkInteger
  Integer kind: otUByte
  Size of type: 1

  Type name: Double
  Type kind: tkFloat
  Float kind: ftDouble
  Size of type: 8

  Type name: .1
  Type kind: tkDynArray
  Array element type name: string
  Array element type size: 4
  Size of type: 4
  }
```

See Also

GetTypeData (see page 4723)

TTypeData (see page 4758)

TTypeKind (see page 4760)

46.3.2 TypInfo.TIntfFlag Enumeration

Pascal

```
TIntfFlag = (
  ifHasGuid,
  ifDispInterface,
  ifDispatch
);
```

C++

```
enum TIntfFlag {
  ifHasGuid,
  ifDispInterface,
  ifDispatch
};
```

File

TypInfo

Description

This is record TypInfo.TIntfFlag.

46.3.3 TypInfo.TManagedField Record

Pascal

```
TManagedField = record
  TypeRef: PTypeInfo;
  FldOffset: Integer;
end;
```

C++

```
struct TManagedField {
  PTypeInfo TypeRef;
  int FldOffset;
};
```

File

TypInfo

Description

This is record TypInfo.TManagedField.

46.3.4 TypInfo.TMethodKind Enumeration

Enumerates floating point number types in the TTypeData (see page 4758) record.

Pascal

```
TMethodKind = (
```

```
    mkProcedure,
    mkFunction,
    mkConstructor,
    mkDestructor,
    mkClassProcedure,
    mkClassFunction,
    mkClassConstructor,
    mkOperatorOverload,
    mkSafeProcedure,
    mkSafeFunction
  );
```

C++

```
  enum TMethodKind {
    mkProcedure,
    mkFunction,
    mkConstructor,
    mkDestructor,
    mkClassProcedure,
    mkClassFunction,
    mkClassConstructor,
    mkOperatorOverload,
    mkSafeProcedure,
    mkSafeFunction
  };
```

File

TypInfo

Description

The GetTypeData (see page 4723) routine utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a pointer to TTypeData (see page 4758) record that describes a component property data type.

TTypeData (see page 4758) is a variant record : the layout depends on the property TTypeKind (see page 4760) value. When it is tkMethod, the data is one of the TMethodKind enumerated values.

The TMethodKind value meanings are as follows :

Value	Meaning
mkProcedure	Stand alone procedure.
mkFunction	Stand alone function.
mkDestructor	Destructor.
mkConstructor	Constructor.
mkClassProcedure	Class procedure.
mkClassFunction	Class function.
mkClassConstructor	Class constructor.
mkOperatorOverload	Operator overload.
mkSafeProcedure	Safe procedure.
mkSafeFunction	Safe function.

See Also

GetTypeData (see page 4723)

TTypeData (see page 4758)

TTypeKind (see page 4760)

46.3.5 TypInfo.TOrdType Enumeration

Enumerates floating point number types in the TTypeData (see page 4758) record.

Pascal

```
TOrdType = (
  otSByte,
  otUByte,
  otSWord,
  otUWord,
  otSLong,
  otULong
);
```

C++

```
enum TOrdType {
  otSByte,
  otUByte,
  otSWord,
  otUWord,
  otSLong,
  otULong
};
```

File

TypInfo

Description

The GetTypeData (see page 4723) routine utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a pointer to TTypeData (see page 4758) record that describes a component property data type.

TTypeData (see page 4758) is a variant record : the layout depends on the property TTypeKind (see page 4760) value. When it is tkWChar, the data is one of the TOrdType enumerated values.

The TOrdType value meanings are as follows :

Value	Meaning
otSByte	Signed byte.
otUByte	Unsigned byte
otSWord	Signed Word
otUWord	Unsigned Word
otSLong	Signed Long
otULong	Unsigned Long

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);

  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
This example will print useful information about any given
type including it's name, size, kind and sub-kind.
}

type
  TForm2 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
{ Defining a generic type that contains a single class (static)
  method that will print information about the type it operates on.
}
  TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
  public
    procedure PrintTypeInfo(memo: TMemo);
  end;
  TGenericString = class(TGenericClass<UnicodeString>)
  end;
  TGenericByte = class(TGenericClass<Byte>)
  end;
  TGenericDouble = class(TGenericClass<Double>)
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

{ Method defintition }
procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
var
  Info     : PTypeInfo;
  Data     : PTypeData;
  KindName : String;
  SubName  : String;
```

```
begin
  { Get type info for the "yet unknown type" }
  Info := System.TypeInfo(T);

  { There is no RTTI attached for some types like Records for example. }
  if Info <> nil then
  begin
    memo.Lines.Add('Type name: ' + Info^.Name);

    { Find out the name of an enum item from it's ordinal value }
    KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));

    memo.Lines.Add('Type kind: ' + KindName);

    Data := GetTypeData(Info);

    if Info^.Kind = tkInteger then
    begin
      { In case of integer let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
      memo.Lines.Add('Integer kind: ' + SubName);
    end;

    if Info^.Kind = tkFloat then
    begin
      { In case of float let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
      memo.Lines.Add('Float kind: ' + SubName);
    end;

    if Info^.Kind = tkDynArray then
    begin
      { Let's check out the element size  }
      memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
      memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
    end;

  end;
  memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
end;


procedure TForm2.FormCreate(Sender: TObject);
var
  GenericString : TGenericClass<UnicodeString>;
  GenericByte : TGenericClass<Byte>;
  GenericDouble : TGenericClass<Double>;
//  GenericStringArray : TGenericClass<array of string>;
begin
  { Prints type info for string type}

  GenericString := TGenericClass<UnicodeString>.Create;
  GenericString.PrintTypeInfo(Memo1);

  { Prints type info for Byte type}
  GenericByte := TGenericClass<Byte>.Create;
  GenericByte.PrintTypeInfo(Memo1);

  { Prints type info for Double type}
  GenericDouble := TGenericClass<Double>.Create;
  GenericDouble.PrintTypeInfo(Memo1);
end;

{
Expected results are:
```

```
Type name: string
Type kind: tkUString
Size of type: 4

Type name: Byte
Type kind: tkInteger
Integer kind: otUByte
Size of type: 1

Type name: Double
Type kind: tkFloat
Float kind: ftDouble
Size of type: 8

Type name: .1
Type kind: tkDynArray
Array element type name: string
Array element type size: 4
Size of type: 4
}
```

See Also

GetTypeData (see page 4723)

TTypeData (see page 4758)

TTypeKind (see page 4760)

46.3.6 TypInfo.TParamFlag Enumeration

TParamFlag and TParamFlags (see page 4772) indicates the attributes of a method parameter.

Pascal

```
TParamFlag = (
  pfVar,
  pfConst,
  pfArray,
  pfAddress,
  pfReference,
  pfOut
);
```

C++

```
enum TParamFlag {
  pfVar,
  pfConst,
  pfArray,
  pfAddress,
  pfReference,
  pfOut
};
```

File

TypInfo

Description

TParamFlags (see page 4772) indicates the attributes of a parameter on a Delphi language function or procedure. It is a set of TParamFlag values:

Value	Meaning
pfVar	The parameter is input/output. Caller must supply a variable to receive output.
pfConst	The parameter can't be changed by the function or procedure.
pfArray	The parameter represents an open array of values.
pfAddress	The parameter is a memory address (pointer).
pfReference	The parameter is passed by reference.
pfOut	The parameter is output. Caller supplies a variable to receive the output.

46.3.7 TypInfo.TPropData Record

Pascal

```
TPropData = packed record
  PropCount: Word;
  PropList: record;
end;
```

C++

```
struct TPropData {
  Word PropCount;
  record PropList;
};
```

File

TypInfo

Description

This is record TypInfo.TPropData.

46.3.8 TypInfo.TPropInfo Record

TPropInfo is a record of component property type information.

Pascal

```
TPropInfo = packed record
  PropType: PPTypeInfo;
  GetProc: Pointer;
  SetProc: Pointer;
  StoredProc: Pointer;
  Index: Integer;
  Default: Longint;
  NameIndex: SmallInt;
  Name: ShortString;
end;
```

C++

```
struct TPropInfo {
  PPTypeInfo PropType;
  void * GetProc;
  void * SetProc;
```

```
    void * StoredProc;
    int Index;
    Longint Default;
    SmallInt NameIndex;
    ShortString Name;
  };
```

File

TypInfo

Description

TPropInfo is principally used to hold the results of a call to GetPropInfo (see page 4717). It provides meta details about a component property as part of Delphi's RTTI (Run Time (see page 4297) Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values.

The TPropInfo fields have the foillowing meanings :

Field	Meaning
PropType	Points to a data structure that gives information about the property type.
GetProc	Method pointer for the method that gets the component property value.
SetProc	Method pointer for the method that sets the component property value.
Index	The index of the property. It is used as a parameter to the GetProc and SetProc methods on indexed properties.
Default	The default value of the property.
NameIndex	The offset into the Name field where the property's name starts.
Name	The qualified property name.

See Also

GetPropInfo (see page 4717)

PPropInfo (see page 4762)

46.3.9 TypInfo.TTypeData Record

TTypeData is used internally to represent type information.

Pascal

```
TTypeData = packed record
  case TTypeKind of
    tkClass: (ClassType: TClass;
              ParentInfo: PPTypeInfo;
              PropCount: SmallInt;
              UnitName: ShortStringBase;);
    tkDynArray: (elSize: Longint;
                 elType: PPTypeInfo;
                 varType: Integer;
                 elType2: PPTypeInfo;
                 DynUnitName: ShortStringBase;);
    tkEnumeration: (BaseType: PPTypeInfo;
                    NameList: ShortStringBase;
                    EnumUnitName: ShortStringBase;);
    tkFloat: (FloatType: TFloatType;);
    tkInt64: (MaxInt64Value: Int64;
```

```
                MinInt64Value: Int64;);
      tkLString: (CodePage: Word;);
      tkMethod: (MethodKind: TMethodKind;
                 ParamCount: Byte;
                 ParamList: array[0..1023] of AnsiChar );
                 tkInterface: ( IntfParent : PPTypeInfo; IntfFlags : TIntfFlagsBase; Guid :
  TGUID; IntfUnit : ShortStringBase;;);
      tkRecord: (RecSize: Integer;
                 ManagedFldCount: Integer;);
      tkSet: (CompType: PPTypeInfo;);
      tkString: (MaxLength: Byte;);
      tkWChar: (OrdType: TOrdType;
                MinValue: Longint;
                MaxValue: Longint;);
  end;
```

C++

```
  struct TTypeData {
    Word CodePage;
    TOrdType OrdType;
    Longint MinValue;
    Longint MaxValue;
    PPTypeInfo BaseType;
    ShortStringBase NameList;
    ShortStringBase EnumUnitName;
    PPTypeInfo CompType;
    TFloatType FloatType;
    Byte MaxLength;
    TClass ClassType;
    PPTypeInfo ParentInfo;
    SmallInt PropCount;
    ShortStringBase UnitName;
    TMethodKind MethodKind;
    Byte ParamCount;
    array[0..1023] of AnsiChar ) ParamList;
    ( IntfParent : PPTypeInfo; IntfFlags : TIntfFlagsBase; Guid : TGUID; IntfUnit :
  ShortStringBase; tkInterface;
    Int64 MaxInt64Value;
    Int64 MinInt64Value;
    Longint elSize;
    PPTypeInfo elType;
    int varType;
    PPTypeInfo elType2;
    ShortStringBase DynUnitName;
    int RecSize;
    int ManagedFldCount;
  };
```

File

TypInfo

Description

A TTypeData structure represents a category of properties, events, or methods in parameters passed to several routines in the TypInfo (see page 4689) unit.

46.3.10 TypInfo.TTypeInfo Record

Holds information for a data type as returned by TypeInfo.

Pascal

```
TTypeInfo = record
  Kind: TTypeKind;
  Name: ShortString;
end;
```

C++

```
struct TTypeInfo {
  TTypeKind Kind;
  ShortString Name;
};
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time (see page 4297) Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo (see page 4767) pointer to a TTypeInfo value.

The TTypeInfo holds the following fields :

Field	Meaning
Kind	The kind of type in RTTI terms. This value is required to select the appropriate variant section in the TTypeData (see page 4758) returned by the GetTypeData (see page 4723).
Name	The name of the data type.

See Also

GetTypeData (see page 4723)

PTypeInfo (see page 4767)

PPTypeInfo (see page 4763)

TTypeKind (see page 4760)

TTypeData (see page 4758)

46.3.11 TypInfo.TTypeKind Enumeration

These types describe a type that is manipulated in the forms designer.

Pascal

```
TTypeKind = (
  tkUnknown,
  tkInteger,
  tkChar,
  tkEnumeration,
  tkFloat,
  tkString,
  tkSet,
  tkClass,
  tkMethod,
  tkWChar,
  tkLString,
  tkWString,
  tkVariant,
```

```
    tkArray,
    tkRecord,
    tkInterface,
    tkInt64,
    tkDynArray,
    tkUString
  );
```

C++

```
  enum TTypeKind {
    tkUnknown,
    tkInteger,
    tkChar,
    tkEnumeration,
    tkFloat,
    tkString,
    tkSet,
    tkClass,
    tkMethod,
    tkWChar,
    tkLString,
    tkWString,
    tkVariant,
    tkArray,
    tkRecord,
    tkInterface,
    tkInt64,
    tkDynArray,
    tkUString
  };
```

File

TypInfo

Description

TTypeKind is an enumerated type that classifies a type.

TTypeInfo (see page 4759) describes a type by giving its classification and its name.

PTypeInfo (see page 4767) is a pointer to a TTypeInfo (see page 4759) value.

PPTypeInfo (see page 4763) is a pointer to a PTypeInfo (see page 4767) pointer.

46.4 Types

The following table lists types in this documentation.

Types

Name	Description
PPropInfo (see page 4762)	PPropInfo points to a TPropInfo (see page 4757) record that holds component property information.
PPropList (see page 4763)	Defines a pointer to a TPropList (see page 4773) array.
PPTypeInfo (see page 4763)	Holds a pointer to a pointer to information for a data type as returned by TypeInfo.
PTypeData (see page 4764)	PTypeData points to a TTypeData (see page 4758) record that holds component property type information.
PTypeInfo (see page 4767)	Holds a pointer to information for a data type as returned by TypeInfo.
ShortStringBase (see page 4770)	Defines a fixed length string for general use in the TTypeData (see page 4758) record.
TIntfFlags (see page 4770)	Enumerates interface flag types in the TTypeData (see page 4758) record.
TIntfFlagsBase (see page 4771)	Set of interface flag types in the TTypeData (see page 4758) record.
TParamFlags (see page 4772)	This is type TypInfo.TParamFlags.
TParamFlagsBase (see page 4772)	This is type TypInfo.TParamFlagsBase.
TPropInfoProc (see page 4772)	This is type TypInfo.TPropInfoProc.
TPropList (see page 4773)	Defines an array of PPropInfo (see page 4762) values.
TTypeKinds (see page 4773)	Defines a set of TTypeKind (see page 4760) values.

46.4.1 TypInfo.PPropInfo Type

PPropInfo points to a TPropInfo (see page 4757) record that holds component property information.

Pascal

```
PPropInfo = ^TPropInfo;
```

C++

```
^TPropInfo PPropInfo;
```

File

TypInfo

Description

TPropInfo (see page 4757) is principally used to hold the results of a call to GetPropInfo (see page 4717). It provides meta details about a component property as part of Delphi's RTTI (Run Time (see page 4297) Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values. PPropInfo is a pointer to a TPropInfo (see page 4757) record.

See Also

GetPropInfo (see page 4717)

TPropInfo (see page 4757)

46.4.2 TypInfo.PPropList Type

Defines a pointer to a TPropList (see page 4773) array.

Pascal

```
PPropList = ^TPropList;
```

C++

```
^TPropList PPropList;
```

File

TypInfo

Description

PPropList defines a pointer to an array of PPropInfo (see page 4762) values. This TPropList (see page 4773) array is principally used to hold the results of a call to GetPropList (see page 4718). This list (array) holds pointers to TPropInfo (see page 4757) records that describe the nature of the properties of the component provided to GetPropList (see page 4718).

See Also

GetPropList (see page 4718)

PPropInfo (see page 4762)

TPropInfo (see page 4757)

TPropList (see page 4773)

46.4.3 TypInfo.PPTypeInfo Type

Holds a pointer to a pointer to information for a data type as returned by TypeInfo.

Pascal

```
PPTypeInfo = ^PTypeInfo;
```

C++

```
^PTypeInfo PPTypeInfo;
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time (see page 4297) Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo (see page 4767) pointer to a TTypeInfo (see page 4759) value. PPTypeInfo is a PTypeInfo (see page 4767) pointer.

See Also

GetTypeData (see page 4723)

PTypeInfo (see page 4767)

TTypeInfo (see page 4759)

TTypeKind (see page 4760)

TTypeData (see page 4758)

46.4.4 TypInfo.PTypeData Type

PTypeData points to a TTypeData (see page 4758) record that holds component property type information.

Pascal

```
PTypeData = ^TTypeData;
```

C++

```
^TTypeData PTypeData;
```

File

TypInfo

Description

TTypeData (see page 4758) is principally used to hold the results of a call to GetTypeData (see page 4723). It provides meta details about a component property type as part of Delphi's RTTI (Run Time (see page 4297) Type Information) mechanism, giving run time component property processing. This allows for more generalised component handling that avoids excessive hard coded values. PTypeData is a pointer to a TTypeData (see page 4758) record.

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);

  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
```

```
  This example will print useful information about any given
  type including it's name, size, kind and sub-kind.
  }

  type
    TForm2 = class(TForm)
      Memo1: TMemo;
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;
  { Defining a generic type that contains a single class (static)
    method that will print information about the type it operates on.
  }
    TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
    public
      procedure PrintTypeInfo(memo: TMemo);
    end;
    TGenericString = class(TGenericClass<UnicodeString>)
    end;
    TGenericByte = class(TGenericClass<Byte>)
    end;
    TGenericDouble = class(TGenericClass<Double>)
    end;

  var
    Form2: TForm2;

  implementation

  {$R *.dfm}

  { Method defintition }
  procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
  var
    Info     : PTypeInfo;
    Data     : PTypeData;
    KindName : String;
    SubName  : String;
  begin
    { Get type info for the "yet unknown type" }
    Info := System.TypeInfo(T);

    { There is no RTTI attached for some types like Records for example. }
    if Info <> nil then
    begin
      memo.Lines.Add('Type name: ' + Info^.Name);

      { Find out the name of an enum item from it's ordinal value }
      KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));

      memo.Lines.Add('Type kind: ' + KindName);

      Data := GetTypeData(Info);

      if Info^.Kind = tkInteger then
      begin
        { In case of integer let's see the actual sub-type name }
        SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
        memo.Lines.Add('Integer kind: ' + SubName);
      end;

      if Info^.Kind = tkFloat then
      begin
```

```
        { In case of float let's see the actual sub-type name }
        SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
        memo.Lines.Add('Float kind: ' + SubName);
      end;

      if Info^.Kind = tkDynArray then
      begin
        { Let's check out the element size  }
        memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
        memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
      end;

    end;
    memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
  end;


  procedure TForm2.FormCreate(Sender: TObject);
  var
    GenericString : TGenericClass<UnicodeString>;
    GenericByte : TGenericClass<Byte>;
    GenericDouble : TGenericClass<Double>;
  //  GenericStringArray : TGenericClass<array of string>;
  begin
    { Prints type info for string type}

    GenericString := TGenericClass<UnicodeString>.Create;
    GenericString.PrintTypeInfo(Memo1);

    { Prints type info for Byte type}
    GenericByte := TGenericClass<Byte>.Create;
    GenericByte.PrintTypeInfo(Memo1);

    { Prints type info for Double type}
    GenericDouble := TGenericClass<Double>.Create;
    GenericDouble.PrintTypeInfo(Memo1);
  end;

  {
  Expected results are:

  Type name: string
  Type kind: tkUString
  Size of type: 4

  Type name: Byte
  Type kind: tkInteger
  Integer kind: otUByte
  Size of type: 1

  Type name: Double
  Type kind: tkFloat
  Float kind: ftDouble
  Size of type: 8

  Type name: .1
  Type kind: tkDynArray
  Array element type name: string
  Array element type size: 4
  Size of type: 4
  }
```

See Also

GetTypeData (see page 4723)

TTypeData (see page 4758)

46.4.5 TypInfo.PTypeInfo Type

Holds a pointer to information for a data type as returned by TypeInfo.

Pascal

```
PTypeInfo = ^TTypeInfo;
```

C++

```
^TTypeInfo PTypeInfo;
```

File

TypInfo

Description

The TypeInfo function is part of Delphi's Run Time (see page 4297) Type Information (RTTI) mechanism. It provides run time information about a specific data type. TypeInfo returns a PTypeInfo pointer to a TTypeInfo (see page 4759) value.

The TTypeInfo (see page 4759) holds the following fields :

Field	Meaning
Kind	The kind of type in RTTI terms. This value is required to select the appropriate variant section in the TTypeData (see page 4758) returned by the GetTypeData (see page 4723).
Name	The name of the data type.

C++ Examples:

```
/*
Add the Delphi source file that appears on this Help page into
a CPP Builder project that includes a CPP module containing the
following code.  An hpp file will be generated for the Delphi
code when you build the project.  Add the include line for
that hpp file at the top of the CPP module.  The FormCreates
for both forms will execute and the following generics code
will work.  Remember to give the forms different names!
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  // Prints type info for string type
  TGenericClass__1<System::UnicodeString> *GString = new
TGenericClass__1<System::UnicodeString>();
  GString->PrintTypeInfo(Memo1);

  // Prints type info for Byte type
  TGenericClass__1<Byte> *GByte = new TGenericClass__1<Byte>();
  GByte->PrintTypeInfo(Memo1);

  // Prints type info for Double type
  TGenericClass__1<Double> *GDouble = new TGenericClass__1<Double>();
  GDouble->PrintTypeInfo(Memo1);

  // Prints type info for "array of String" type
  // TGenericClass__1<array of string> *GStringArray = new TGenericClass__1<array of string>();
  // GStringArray->PrintTypeInfo(Memo1);
}
```

Delphi Examples:

```
{
This example demostrates the methods to retreive of
Run-time Type Information (RTTI) information.
This example will print useful information about any given
type including it's name, size, kind and sub-kind.
}

type
  TForm2 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
{ Defining a generic type that contains a single class (static)
  method that will print information about the type it operates on.
}
  TGenericClass<T> = class(TList<T>)  // TList is a class in Generics.Collections
  public
    procedure PrintTypeInfo(memo: TMemo);
  end;
  TGenericString = class(TGenericClass<UnicodeString>)
  end;
  TGenericByte = class(TGenericClass<Byte>)
  end;
  TGenericDouble = class(TGenericClass<Double>)
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

{ Method defintition }
procedure TGenericClass<T>.PrintTypeInfo(memo: TMemo);
var
  Info     : PTypeInfo;
  Data     : PTypeData;
  KindName : String;
  SubName  : String;
begin
  { Get type info for the "yet unknown type" }
  Info := System.TypeInfo(T);

  { There is no RTTI attached for some types like Records for example. }
  if Info <> nil then
  begin
    memo.Lines.Add('Type name: ' + Info^.Name);

    { Find out the name of an enum item from it's ordinal value }
    KindName := TypInfo.GetEnumName(System.TypeInfo(TTypeKind), Ord(Info^.Kind));

    memo.Lines.Add('Type kind: ' + KindName);

    Data := GetTypeData(Info);

    if Info^.Kind = tkInteger then
    begin
      { In case of integer let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TOrdType), Ord(Data^.OrdType));
      memo.Lines.Add('Integer kind: ' + SubName);
    end;
```

```
    if Info^.Kind = tkFloat then
    begin
      { In case of float let's see the actual sub-type name }
      SubName := TypInfo.GetEnumName(System.TypeInfo(TFloatType), Ord(Data^.FloatType));
      memo.Lines.Add('Float kind: ' + SubName);
    end;

    if Info^.Kind = tkDynArray then
    begin
      { Let's check out the element size  }
      memo.Lines.Add('Array element type name: ' + Data^.elType^^.Name);
      memo.Lines.Add('Array element type size: ' + IntToStr(Data^.elSize));
    end;

  end;
  memo.Lines.Add('Size of type: ' + IntToStr(SizeOf(T)));
end;


procedure TForm2.FormCreate(Sender: TObject);
var
  GenericString : TGenericClass<UnicodeString>;
  GenericByte : TGenericClass<Byte>;
  GenericDouble : TGenericClass<Double>;
//  GenericStringArray : TGenericClass<array of string>;
begin
  { Prints type info for string type}

  GenericString := TGenericClass<UnicodeString>.Create;
  GenericString.PrintTypeInfo(Memo1);

  { Prints type info for Byte type}
  GenericByte := TGenericClass<Byte>.Create;
  GenericByte.PrintTypeInfo(Memo1);

  { Prints type info for Double type}
  GenericDouble := TGenericClass<Double>.Create;
  GenericDouble.PrintTypeInfo(Memo1);
end;

{
Expected results are:

Type name: string
Type kind: tkUString
Size of type: 4

Type name: Byte
Type kind: tkInteger
Integer kind: otUByte
Size of type: 1

Type name: Double
Type kind: tkFloat
Float kind: ftDouble
Size of type: 8

Type name: .1
Type kind: tkDynArray
Array element type name: string
Array element type size: 4
Size of type: 4
}
```

See Also

GetTypeData (see page 4723)

PPTypeInfo (see page 4763)

TTypeInfo (see page 4759)

TTypeKind (see page 4760)

TTypeData (see page 4758)

46.4.6 TypInfo.ShortStringBase Type

Defines a fixed length string for general use in the TTypeData (see page 4758) record.

Pascal

```
ShortStringBase = string[255];
```

C++

```
string[255] ShortStringBase;
```

File

TypInfo

Description

The GetTypeData (see page 4723) routine utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a pointer to TTypeData (see page 4758) record that describes a component property data type.

A number of the variant data sections in the TTypeData (see page 4758) record use the ShortStringBase to define a 255 character string to hold Name values such as UnitName for a class data type.

See Also

GetTypeData (see page 4723)

TTypeData (see page 4758)

46.4.7 TypInfo.TIntfFlags Type

Enumerates interface flag types in the TTypeData (see page 4758) record.

Pascal

```
TIntfFlags = set of TIntfFlag;
```

C++

```
TIntfFlag TIntfFlags;
```

File

TypInfo

Description

The GetTypeData (see page 4723) routine utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a

pointer to TTypeData (see page 4758) record that describes a component property data type.

TTypeData (see page 4758) is a variant record : the layout depends on the property TTypeKind (see page 4760) value. When it is tkInterface, the data contains a TIntfFlagsBase (see page 4771) set of TIntfFlags values.

The TIntfFlags value meanings are as follows :

Value	Meaning
ifHasGuid	Interface has a GUID (Globally Unique IDentifier).
ifDispInterface	Is a dispatch interface.
ifDispatch	Can be dispatched.

See Also

GetTypeData (see page 4723)

TIntfFlagsBase (see page 4771)

TTypeData (see page 4758)

TTypeKind (see page 4760)

46.4.8 TypInfo.TIntfFlagsBase Type

Set of interface flag types in the TTypeData (see page 4758) record.

Pascal

```
TIntfFlagsBase = set of TIntfFlag;
```

C++

```
TIntfFlag TIntfFlagsBase;
```

File

TypInfo

Description

The GetTypeData (see page 4723) routine utilises Delphi's RTTI (Run Time (see page 4297) Type Information) to return a pointer to TTypeData (see page 4758) record that describes a component property data type.

TTypeData (see page 4758) is a variant record : the layout depends on the property TTypeKind (see page 4760) value. When it is tkInterface, the data contains a TIntfFlagsBase set of TIntfFlag (see page 4751) values.

The TIntfFlag (see page 4751) value meanings are as follows :

Value	Meaning
ifHasGuid	Interface has a GUID (Globally Unique IDentifier).
ifDispInterface	Is a dispatch interface.
ifDispatch	Can be dispatched.

See Also

GetTypeData (see page 4723)

TIntfFlag (see page 4751)

TTypeData (see page 4758)

TTypeKind (see page 4760)

46.4.9 TypInfo.TParamFlags Type

Pascal

```
TParamFlags = set of TParamFlag;
```

C++

```
TParamFlag TParamFlags;
```

File

TypInfo

Description

This is type TypInfo.TParamFlags.

46.4.10 TypInfo.TParamFlagsBase Type

Pascal

```
TParamFlagsBase = set of TParamFlag;
```

C++

```
TParamFlag TParamFlagsBase;
```

File

TypInfo

Description

This is type TypInfo.TParamFlagsBase.

46.4.11 TypInfo.TPropInfoProc Type

Pascal

```
TPropInfoProc = procedure (PropInfo: PPropInfo) of object;
```

C++

```
(PropInfo: PPropInfo) ( TPropInfoProc)();
```

File

TypInfo

Description

This is type TypInfo.TPropInfoProc.

46.4.12 TypInfo.TPropList Type

Defines an array of PPropInfo (see page 4762) values.

Pascal

```
TPropList = array[0..16379] of PPropInfo;
```

C++

```
array[0..16379] of PPropInfo TPropList;
```

File

TypInfo

Description

TPropList is principally used to hold the results of a call to GetPropList (see page 4718). This list (array) holds pointers to TPropInfo (see page 4757) records that describe the nature of the properties of the component provided to GetPropList (see page 4718).

See Also

GetPropList (see page 4718)

TPropInfo (see page 4757)

46.4.13 TypInfo.TTypeKinds Type

Defines a set of TTypeKind (see page 4760) values.

Pascal

```
TTypeKinds = set of TTypeKind;
```

C++

```
TTypeKind TTypeKinds;
```

File

TypInfo

Description

TTypeKinds defines a set of TTypeKind (see page 4760) values. It is used to select which component property types to use in various routines, such as GetPropInfo (see page 4717).

See Also

GetPropInfo (see page 4717)

TTypeKind (see page 4760)

46.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
BooleanIdents (see page 4774)	An array of string representations of true and false.
DotSep (see page 4774)	This is variable TypInfo.DotSep.

46.5.1 TypInfo.BooleanIdents Variable

An array of string representations of true and false.

Pascal

```
BooleanIdents: array [Boolean] of string = ('False', 'True');
```

C++

```
array [Boolean] of string BooleanIdents = ('False', 'True');
```

File

TypInfo

Description

The GetEnumName (see page 4705) routine is part of Delphi's RTTI (Run Time (see page 4297) Type Information). It returns a string representation of an enumerated type value. If that type is true or false, BooleanIdents is used internally.

See Also

GetEnumName (see page 4705)

46.5.2 TypInfo.DotSep Variable

Pascal

```
DotSep: string = '.';
```

C++

```
UnicodeString DotSep = '.';
```

File

TypInfo

Description

This is variable TypInfo.DotSep.

46.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
tkAny (see page 4775)	tkAny is a set that contains all values defined by the TTypeKind (see page 4760) enumeration.
tkMethods (see page 4775)	tkMethods is a set that contains values, defined by the TTypeKind (see page 4760) enumeration, that represent a method kind.
tkProperties (see page 4776)	tkProperties is a set that contains values, defined by the TTypeKind (see page 4760) enumeration, that represent types that can be published.

46.6.1 TypInfo.tkAny Constant

tkAny is a set that contains all values defined by the TTypeKind (see page 4760) enumeration.

Pascal

```
tkAny = [Low(TTypeKind)..High(TTypeKind)];
```

C++

```
tkAny = [Low(TTypeKind)..High(TTypeKind)];
```

File

TypInfo

Description

tkAny set is not particularly useful because it contains all the values defined by TTypeKind (see page 4760) enumeration.

It is primarily used by other more restricted sets, but can also be passed as a parameter to routines like GetPropInfo (see page 4717).

See Also

TTypeKinds (see page 4773)

tkMethods (see page 4775)

tkProperties (see page 4776)

46.6.2 TypInfo.tkMethods Constant

tkMethods is a set that contains values, defined by the TTypeKind (see page 4760) enumeration, that represent a method kind.

Pascal

```
tkMethods = [tkMethod];
```

C++

```
tkMethods = [tkMethod];
```

File

TypInfo

Description

tkMethods set can be used when you need to identify a type kind as being a method.

Use this constant in conjunction with routines such as GetPropInfo (see page 4717) that require a TTypeKind (see page 4760) set as a parameter.

See Also

TTypeKinds (see page 4773)

tkAny (see page 4775)

tkProperties (see page 4776)

46.6.3 TypInfo.tkProperties Constant

tkProperties is a set that contains values, defined by the TTypeKind (see page 4760) enumeration, that represent types that can be published.

Pascal

```
tkProperties = tkAny - tkMethods - [tkUnknown];
```

C++

```
tkProperties = tkAny - tkMethods - [tkUnknown];
```

File

TypInfo

Description

tkProperties is useful in cases when you need to select only the type kinds which represent data types that can be published in a TObject (see page 3354)-derived class.

See Also

TTypeKinds (see page 4773)

tkAny (see page 4775)

tkMethods (see page 4775)

47 ValEdit Namespace

This is namespace ValEdit.

Topics

Name	Description
Classes (see page 4778)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 4892)	The following table lists structs, records, enums in this documentation.
Types (see page 4894)	The following table lists types in this documentation.

Classes

	Name	Description
	TItemProp (see page 4778)	TItemProp configures how users can edit the item values in a value list editor.
	TValueListEditor (see page 4784)	TValueListEditor provides a simple way to edit string lists that contain name/value pairs.
	TValueListStrings (see page 4882)	This is class ValEdit.TValueListStrings.

Structs, Records, Enums

	Name	Description
	TDisplayOption (see page 4892)	TDisplayOption and TDisplayOptions (see page 4894) control the appearance of a value list editor.
	TKeyOption (see page 4893)	TKeyOption and TKeyOptions (see page 4895) indicate the response of a value list editor to certain keystrokes.

Types

Name	Description
TDisplayOptions (see page 4894)	TDisplayOptions and TDisplayOption (see page 4892) control the appearance of a value list editor.
TGetPickListEvent (see page 4895)	TGetPickListEvent is the type for event handlers that supply pick list values to a value list editor.
TItemProps (see page 4895)	This is type ValEdit.TItemProps.
TKeyOptions (see page 4895)	TKeyOptions and TKeyOption (see page 4893) indicate the response of a value list editor to certain keystrokes.
TOnValidateEvent (see page 4896)	TOnValidateEvent is the type for event handlers that validate the values a user types into a value list editor.

47.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TItemProp (see page 4778)	TItemProp configures how users can edit the item values in a value list editor.
	TValueListEditor (see page 4784)	TValueListEditor provides a simple way to edit string lists that contain name/value pairs.
	TValueListStrings (see page 4882)	This is class ValEdit.TValueListStrings.

47.1.1 TItemProp Class

TItemProp configures how users can edit the item values in a value list editor.

Class Hierarchy

Pascal

```
TItemProp = class(TPersistent);
```

C++

```
class TItemProp : public TPersistent;
```

File

ValEdit

Description

For each item that it allows users to edit, the TValueListEditor (see page 4784) component maintains a TItemProp object. This object's properties govern how users can edit the value in the value list editor.

TItemProp objects are created and owned by the value list editor that uses them to represent properties of item values (such as an edit mask or maximum length) or to supply a fixed pick list of values that users can assign to the item.

See Also

TValueListEditor (see page 4784)

47.1.1.1 TItemProp Members

The following tables list the members exposed by TItemProp.

Protected Methods

	Name	Description
	AssignTo (see page 4780)	Copies the properties of an object to a destination object.
	UpdateEdit (see page 4781)	This is UpdateEdit, a member of class TItemProp.

Public Methods

	Name	Description
	Create (see page 4779)	Creates an instance of TItemProp (see page 4778).
	Destroy (see page 4779)	Frees the memory associated with the TItemProp (see page 4778) object.
	HasPickList (see page 4781)	Indicates whether the PickList (see page 4783) property provides a static set of values for this item.

Published Properties

	Name	Description
	EditMask (see page 4781)	Restricts the value that can be entered for the associated item.
	EditStyle (see page 4782)	Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.
	KeyDesc (see page 4783)	Specifies the name of the item, which appears in the names column of the Value list editor.
	MaxLength (see page 4783)	Specifies the maximum number of characters that users can enter for the item's value.
	PickList (see page 4783)	Lists the items that appear in the drop-down pick list for the item.
	ReadOnly (see page 4784)	Indicates whether users can edit the item's value by typing in a new value.

47.1.1.2 TItemProp.Create Constructor

Creates an instance of TItemProp (see page 4778).

Pascal

```
constructor Create(AEditor: TValueListEditor);
```

C++

```
__fastcall TItemProp(TValueListEditor AEditor);
```

Description

Do not call Create directly in an application. TValueListEditor (see page 4784) creates TItemProp (see page 4778) instances as needed when you make assignments to its ItemProps property.

The AEditor parameter is the value list editor that creates the TItemProp (see page 4778) instance to represent the properties of one of its items.

See Also

ItemProps (see page 4821)

47.1.1.3 TItemProp.Destroy Destructor

Frees the memory associated with the TItemProp (see page 4778) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TItemProp();
```

Description

Do not call Destroy. The TValueListEditor (see page 4784) that owns this TItemProp (see page 4778) object destroys the object when it is no longer needed.

See Also

Create (see page 4779)

System::TObject::Free (see page 3372)

47.1.1.4 TItemProp Methods

The methods of the TItemProp class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 4780)	Copies the properties of an object to a destination object.
	UpdateEdit (see page 4781)	This is UpdateEdit, a member of class TItemProp.

Public Methods

	Name	Description
	HasPickList (see page 4781)	Indicates whether the PickList (see page 4783) property provides a static set of values for this item.

47.1.1.4.1 TItemProp.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3919)

47.1.1.4.2 TItemProp.HasPickList Method

Indicates whether the PickList (see page 4783) property provides a static set of values for this item.

Pascal

```
function HasPickList: Boolean;
```

C++

```
__fastcall Boolean HasPickList();
```

Description

Call PickList (see page 4783) to determine whether this TItemProp (see page 4778) object supplies values using the PickList (see page 4783) property. HasPickList returns true if the PickList (see page 4783) property contains a nonempty list of values.

See Also

PickList (see page 4783)

47.1.1.4.3 TItemProp.UpdateEdit Method

Pascal

```
procedure UpdateEdit;
```

C++

```
__fastcall UpdateEdit();
```

Description

This is UpdateEdit, a member of class TItemProp.

47.1.1.5 TItemProp Properties

The properties of the TItemProp class are listed here.

Published Properties

	Name	Description
	EditMask (see page 4781)	Restricts the value that can be entered for the associated item.
	EditStyle (see page 4782)	Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.
	KeyDesc (see page 4783)	Specifies the name of the item, which appears in the names column of the Value list editor.
	MaxLength (see page 4783)	Specifies the maximum number of characters that users can enter for the item's value.
	PickList (see page 4783)	Lists the items that appear in the drop-down pick list for the item.
	ReadOnly (see page 4784)	Indicates whether users can edit the item's value by typing in a new value.

47.1.1.5.1 TItemProp.EditMask Property

Restricts the value that can be entered for the associated item.

Pascal

```
property EditMask: string;
```

C++

```
__property UnicodeString EditMask;
```

Description

Use EditMask to restrict the characters a user can enter. If the user attempts to enter a character that is not valid, the character is not accepted. Validation using the EditMask property is performed on a character-by-character basis.

A mask consists of three fields with semicolons separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are included as part of the data. The third part of the mask is the character used to represent unentered characters in the mask.

Setting EditMask to an empty string removes the mask.

For more information on using edit masks, see the TEditMask type.

See Also

Values (see page 4835)

OnSetEditText

47.1.1.5.2 TItemProp.EditStyle Property

Determines whether the cell that contains the associated value has a drop-down arrow or ellipsis button.

Pascal

```
property EditStyle: TEditStyle;
```

C++

```
__property TEditStyle EditStyle;
```

Description

Set EditStyle to specify whether the cell in the value list editor that contains the associated value should allow users to drop down a pick list or to launch a dialog to edit the value.

When EditStyle is esSimple, the value cell is a simple edit box that lets users edit the associated value directly (unless the ReadOnly (see page 4784) property is true).

When EditStyle is esEllipsis, the value cell contains an ellipsis button that users can click to generate an OnEditButtonClick event. Typically, applications use this event to launch a dialog that edits the value.

When EditStyle is esPickList, the value cell contains a drop-down arrow that users can click to display a list of values from which to select. This value must be supplied by the PickList (see page 4783) property or in an OnGetPickList event handler (or supplied by the PickList (see page 4783) property and modified in an OnGetPickList event handler).

See Also

PickList (see page 4783)

OnGetPickList (see page 4860)

OnEditButtonClick (see page 4849)

ReadOnly (see page 4784)

KeyOptions (see page 4821)

DisplayOptions (see page 4812)

47.1.1.5.3 TItemProp.KeyDesc Property

Specifies the name of the item, which appears in the names column of the Value list editor.

Pascal

```
property KeyDesc: string;
```

C++

```
__property UnicodeString KeyDesc;
```

Description

Set KeyDesc to specify a name for this item. This name appears in the names column of the Value list editor.

If ItemProp is not set, then the name that appears in the key column of the Value list editor is the corresponding value of the Keys property.

See Also

Keys (see page 4822)

47.1.1.5.4 TItemProp.MaxLength Property

Specifies the maximum number of characters that users can enter for the item's value.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be typed when assigning a value to the item. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing value, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

Note: Even when MaxLength is 0, there may be limitations imposed by the operating system on the number of characters that may be entered.

See Also

Values (see page 4835)

EditMask (see page 4781)

47.1.1.5.5 TItemProp.PickList Property

Lists the items that appear in the drop-down pick list for the item.

Pascal

```
property PickList: TStrings;
```

C++

```
__property TStrings PickList;
```

Description

Use PickList to supply the values that appear in the drop-down pick list for the associated item. PickList lets you supply a static set of items that appear in the drop-down list for the associated item when EditStyle (see page 4782) is esPickList. These items can be subsequently modified by an OnGetPickList event handler.

Note: PickList is only used when the EditStyle (see page 4782) property is esPickList. If EditStyle (see page 4782) is esPickList, and PickList does list a set of values, you must supply an OnGetPickList event handler to supply the items.

See Also

EditStyle (see page 4782)

OnGetPickList (see page 4860)

DropDownRows (see page 4816)

47.1.1.5.6 TItemProp.ReadOnly Property

Indicates whether users can edit the item's value by typing in a new value.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set ReadOnly to true to prevent users from typing in a new value for the associated item. For example, to force users to select item values only from a pick list, you can set the EditStyle (see page 4782) to esPickList and set ReadOnly to true. Similarly, to force users to enter values only using a property editor, you can set EditStyle (see page 4782) to esEllipsis and set ReadOnly to true.

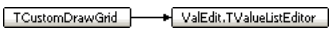
See Also

EditStyle (see page 4782)

47.1.2 TValueListEditor Class

TValueListEditor provides a simple way to edit string lists that contain name/value pairs.

Class Hierarchy



Pascal

```
TValueListEditor = class(TCustomDrawGrid);
```

C++

```
class TValueListEditor : public TCustomDrawGrid;
```

File

ValEdit

Description

Use TValueListEditor to let users edit string lists that consist of strings with the form Name=Value. By default, the names appear

in a column headed "Key", and the values in a column headed "Value", although you can customize these labels. You can control whether users can select, add, or edit the strings in the Key column. For each entry in the Value column, you can let users edit the string, supply a drop-down list from which they can select values, or provide a dialog in which they can edit values.

See Also

TStringList

TStrings

Value List Editors (VCL Only)

47.1.2.1 TValueListEditor Members

The following tables list the members exposed by TValueListEditor.

Protected Methods

	Name	Description
	AdjustColWidths (see page 4789)	This is AdjustColWidths, a member of class TValueListEditor.
	AdjustRowCount (see page 4789)	This is AdjustRowCount, a member of class TValueListEditor.
	CanEditModify (see page 4790)	Indicates whether the current cell in the grid can be edited.
	ColWidthsChanged (see page 4790)	Responds when the column widths change.
	CreateEditor (see page 4790)	Creates the inplace editor control.
	CreateWnd (see page 4791)	Creates the underlying window.
	DoExit (see page 4792)	Hides the inplace editor when selection moves to another control in a form.
	DoOnValidate (see page 4792)	This is DoOnValidate, a member of class TValueListEditor.
	DrawCell (see page 4792)	Generates an OnDrawCell event.
	EditListGetItems (see page 4793)	This is EditListGetItems, a member of class TValueListEditor.
	GetCell (see page 4794)	This is GetCell, a member of class TValueListEditor.
	GetColCount (see page 4794)	This is GetColCount, a member of class TValueListEditor.
	GetEditLimit (see page 4794)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 4794)	Generates an OnGetEditMask event.
	GetEditStyle (see page 4795)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 4795)	This is GetEditText, a member of class TValueListEditor.
	GetPickList (see page 4795)	This is GetPickList, a member of class TValueListEditor.
	GetRowCount (see page 4795)	This is GetRowCount, a member of class TValueListEditor.
	IsEmptyRow (see page 4796)	This is IsEmptyRow, a member of class TValueListEditor.
	KeyDown (see page 4797)	Provides special processing when the user presses a key.
	MouseDown (see page 4797)	Provides special processing when the user presses a mouse button.
	Resize (see page 4798)	Respond to control resize.
	RowMoved (see page 4799)	Generates an OnRowMoved event.
	SelectCell (see page 4799)	Determines whether a particular cell in the grid can be selected.
	SetCell (see page 4799)	This is SetCell, a member of class TValueListEditor.
	SetEditText (see page 4800)	This is SetEditText, a member of class TValueListEditor.
	SetStrings (see page 4800)	This is SetStrings, a member of class TValueListEditor.
	StringsChanging (see page 4800)	This is StringsChanging, a member of class TValueListEditor.
	TitleCaptionsStored (see page 4800)	This is TitleCaptionsStored, a member of class TValueListEditor.

Protected Properties

	Name	Description
	EditList (see page 4817)	This is EditList, a member of class TValueListEditor.

Public Methods

	Name	Description
	Create (see page 4788)	Creates and initializes new TValueListEditor (see page 4784) object.
	DeleteRow (see page 4791)	Deletes a row in the value list editor and a corresponding string from the associated TStrings object.
	Destroy (see page 4788)	Frees the memory associated with the TValueListEditor (see page 4784) object.
	FindRow (see page 4793)	Locates the row that corresponds to a specified key name.
	InsertRow (see page 4796)	Inserts a row in the value list editor and a corresponding string into the associated TStrings object.
	Refresh (see page 4798)	Updates the display to reflect the current value of the Strings (see page 4834) property.
	RestoreCurrentRow (see page 4798)	Undoes all edits to the current row.

Public Properties

	Name	Description
	Cells (see page 4806)	Lists the strings for each cell in the editor.
	ColCount (see page 4807)	Indicates the number of columns in the value list editor.
	ItemProps (see page 4821)	Specifies various constraints on the way users can enter a value for each item in the list.
	Keys (see page 4822)	Lists the strings for each cell in the name (key) column.
	RowCount (see page 4831)	Indicates the number of rows in the value list editor.
	Values (see page 4835)	Lists the strings for each value from the value column.
	VisibleColCount (see page 4837)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 4838)	Indicates the number of scrollable rows visible in the grid.

Published Events

	Name	Description
	OnClick (see page 4839)	Occurs when the user clicks the control.
	OnContextPopup (see page 4841)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4841)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4842)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4845)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 4848)	Occurs when a cell in the grid needs to be drawn.
	OnEditButtonClick (see page 4849)	Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.
	OnEndDock (see page 4850)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4850)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4851)	Occurs when a control receives the input focus.
	OnExit (see page 4854)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 4858)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 4859)	Occurs when the in-place editor requests the value of a cell.
	OnGetPickList (see page 4860)	Supplies a pick list for values associated with a particular key.
	OnKeyDown (see page 4861)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4864)	Occurs when a key is pressed.
	OnKeyUp (see page 4865)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4867)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4868)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4870)	Occurs when the user moves the mouse into a control.
	OnMouseLeave (see page 4871)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4871)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

	OnMouseUp (see page 4873)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 4875)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 4876)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 4876)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 4877)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 4879)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 4879)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4880)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStringsChange (see page 4881)	Occurs immediately after the names or values in the string list change.
	OnStringsChanging (see page 4881)	Occurs immediately before the names or values in the string list change.
	OnTopLeftChanged (see page 4882)	Occurs immediately after the TopRow property or the LeftCol property changes.
	OnValidate (see page 4882)	Occurs when focus shifts away from a cell in the value list editor.

Published Properties

	Name	Description
	Align (see page 4802)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4805)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4806)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4806)	Determines whether a single line border is drawn around the grid.
	Color (see page 4807)	Specifies the background color of the control.
	Constraints (see page 4808)	Specifies the size constraints for the control.
	Ctl3D (see page 4809)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 4809)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 4811)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 4811)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DisplayOptions (see page 4812)	Controls aspects of the value list editor's appearance.
	DoubleBuffered (see page 4812)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4814)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4814)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4814)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownRows (see page 4816)	Limits the length of the drop-down list that appears for values that use a pick list.
	Enabled (see page 4817)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 4817)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 4818)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	Font (see page 4818)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 4819)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	KeyOptions (see page 4821)	Controls the way the value list editor responds to user keystrokes.
	Options (see page 4822)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 4824)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4824)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4825)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 4825)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 4825)	Determines where a control looks for its font information.
	ParentShowHint (see page 4826)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4826)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 4832)	Specifies whether the grid includes horizontal and vertical scroll bars.
	ShowHint (see page 4832)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Strings (see page 4834)	Provides access to the string list that is being edited.

	TabOrder (see page 4834)	Indicates the position of the control in its parent's tab order.
	TitleCaptions (see page 4835)	Specifies the captions that label the columns of the value list editor.
	Visible (see page 4836)	Determines whether the component appears onscreen.

47.1.2.2 TValueListEditor.Create Constructor

Creates and initializes new TValueListEditor (see page 4784) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TValueListEditor(TComponent * AOwner);
```

Description

Call Create to create a value list editor component at runtime. TValueListEditor (see page 4784) components added to forms at design time are created automatically.

The AOwner parameter specifies another component (typically the form) that is responsible for freeing the TValueListEditor (see page 4784) instance.

See Also

Owner

47.1.2.3 TValueListEditor.Destroy Destructor

Frees the memory associated with the TValueListEditor (see page 4784) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TValueListEditor();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the value list editor reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

47.1.2.4 TValueListEditor Methods

The methods of the TValueListEditor class are listed here.

Protected Methods

	Name	Description
	AdjustColWidths (see page 4789)	This is AdjustColWidths, a member of class TValueListEditor.
	AdjustRowCount (see page 4789)	This is AdjustRowCount, a member of class TValueListEditor.
	CanEditModify (see page 4790)	Indicates whether the current cell in the grid can be edited.
	ColWidthsChanged (see page 4790)	Responds when the column widths change.
	CreateEditor (see page 4790)	Creates the inplace editor control.

	CreateWnd (see page 4791)	Creates the underlying window.
	DoExit (see page 4792)	Hides the inplace editor when selection moves to another control in a form.
	DoOnValidate (see page 4792)	This is DoOnValidate, a member of class TValueListEditor.
	DrawCell (see page 4792)	Generates an OnDrawCell event.
	EditListGetItems (see page 4793)	This is EditListGetItems, a member of class TValueListEditor.
	GetCell (see page 4794)	This is GetCell, a member of class TValueListEditor.
	GetColCount (see page 4794)	This is GetColCount, a member of class TValueListEditor.
	GetEditLimit (see page 4794)	Provides the interface for a method that limits the number of characters the user can type into cells.
	GetEditMask (see page 4794)	Generates an OnGetEditMask event.
	GetEditStyle (see page 4795)	Indicates whether a specified cell has an ellipsis button or drop-down arrow.
	GetEditText (see page 4795)	This is GetEditText, a member of class TValueListEditor.
	GetPickList (see page 4795)	This is GetPickList, a member of class TValueListEditor.
	GetRowCount (see page 4795)	This is GetRowCount, a member of class TValueListEditor.
	IsEmptyRow (see page 4796)	This is IsEmptyRow, a member of class TValueListEditor.
	KeyDown (see page 4797)	Provides special processing when the user presses a key.
	MouseDown (see page 4797)	Provides special processing when the user presses a mouse button.
	Resize (see page 4798)	Respond to control resize.
	RowMoved (see page 4799)	Generates an OnRowMoved event.
	SelectCell (see page 4799)	Determines whether a particular cell in the grid can be selected.
	SetCell (see page 4799)	This is SetCell, a member of class TValueListEditor.
	SetEditText (see page 4800)	This is SetEditText, a member of class TValueListEditor.
	SetStrings (see page 4800)	This is SetStrings, a member of class TValueListEditor.
	StringsChanging (see page 4800)	This is StringsChanging, a member of class TValueListEditor.
	TitleCaptionsStored (see page 4800)	This is TitleCaptionsStored, a member of class TValueListEditor.

Public Methods

	Name	Description
	DeleteRow (see page 4791)	Deletes a row in the value list editor and a corresponding string from the associated TStrings object.
	FindRow (see page 4793)	Locates the row that corresponds to a specified key name.
	InsertRow (see page 4796)	Inserts a row in the value list editor and a corresponding string into the associated TStrings object.
	Refresh (see page 4798)	Updates the display to reflect the current value of the Strings (see page 4834) property.
	RestoreCurrentRow (see page 4798)	Undoes all edits to the current row.

47.1.2.4.1 TValueListEditor.AdjustColWidths Method

Pascal

```
procedure AdjustColWidths; virtual;
```

C++

```
virtual __fastcall AdjustColWidths();
```

Description

This is AdjustColWidths, a member of class TValueListEditor.

47.1.2.4.2 TValueListEditor.AdjustRowCount Method

Pascal

```
procedure AdjustRowCount; virtual;
```

C++

```
virtual __fastcall AdjustRowCount();
```

Description

This is AdjustRowCount, a member of class TValueListEditor.

47.1.2.4.3 TValueListEditor.CanEditModify Method

Indicates whether the current cell in the grid can be edited.

Pascal

```
function CanEditModify: Boolean; override;
```

C++

```
virtual __fastcall Boolean CanEditModify();
```

Description

Grid components use CanEditModify internally to determine whether to allow the user to edit the current cell. CanEditModify returns true unless the inplace editor has disallowed editing for the cell.

See Also

CanEditShow

InplaceEditor

47.1.2.4.4 TValueListEditor.ColWidthsChanged Method

Responds when the column widths change.

Pascal

```
procedure ColWidthsChanged; override;
```

C++

```
virtual __fastcall ColWidthsChanged();
```

Description

ColWidthsChanged is called immediately after the column widths change. The change can result from setting the ColWidths property, from setting the DefaultColWidth property, from moving one of the columns, or from resizing the columns with the mouse.

ColWidthsChanged adjusts the scroll bar and the position of the inplace editor to take into account the new column size and positions of the columns.

See Also

ColWidths

DefaultColWidth

InplaceEditor

ScrollBars

47.1.2.4.5 TValueListEditor.CreateEditor Method

Creates the inplace editor control.

Pascal

```
function CreateEditor: TInplaceEdit; override;
```

C++

```
virtual __fastcall TInplaceEdit CreateEditor();
```

Description

Grid controls call CreateEditor internally the first time the grid goes into edit mode. CreateEditor is exposed as a protected method so that descendants of TCustomGrid can substitute a specialized descendant of TInplaceEdit to handle cell editing.

See Also

EditorMode

InplaceEditor

ShowEditor

47.1.2.4.6 TValueListEditor.CreateWnd Method

Creates the underlying window.

Pascal

```
procedure CreateWnd; override;
```

C++

```
virtual __fastcall CreateWnd();
```

Description

CreateWnd is called automatically when the control is first created or when the underlying screen object must be destroyed and recreated to reflect property changes. Override CreateWnd to pass additional initialization messages to the screen object when it is created.

See Also

CreateParams

CreateWindowHandle

Perform

DestroyWnd

47.1.2.4.7 TValueListEditor.DeleteRow Method

Deletes a row in the value list editor and a corresponding string from the associated TStrings object.

Pascal

```
procedure DeleteRow(ARow: Integer); override;
```

C++

```
virtual __fastcall DeleteRow(int ARow);
```

Description

Call DeleteRow to remove the row specified by the ARow parameter. Delete (see page 3552) removes the corresponding string from the TStrings object being edited and updates the display.

ARow must be the index of a row that displays a name/value pair. That is, it can't be less than 0 or greater than the number of

rows. If DisplayOptions (see page 4812) includes doColumnTitles, ARow can't be 0, because that is the title row, not a name/value pair in the associated TStrings object.

Note: DeleteRow deletes the specified row regardless of the value of KeyOptions (see page 4821). The keyDelete member of KeyOptions (see page 4821) controls only whether the user can delete a row by pressing the Delete (see page 3552) key.

See Also

Cells (see page 4806)

Strings (see page 4834)

KeyOptions (see page 4821)

DisplayOptions (see page 4812)

47.1.2.4.8 TValueListEditor.DoExit Method

Hides the inplace editor when selection moves to another control in a form.

Pascal

```
procedure DoExit; override;
```

C++

```
virtual __fastcall DoExit();
```

Description

After calling the inherited method, which generates an OnExit event, DoExit hides the inplace editor unless the Options property includes goAlwaysShowEditor.

See Also

HideEditor

Options

OnExit

47.1.2.4.9 TValueListEditor.DoOnValidate Method

Pascal

```
procedure DoOnValidate; virtual;
```

C++

```
virtual __fastcall DoOnValidate();
```

Description

This is DoOnValidate, a member of class TValueListEditor.

47.1.2.4.10 TValueListEditor.DrawCell Method

Generates an OnDrawCell event.

Pascal

```
procedure DrawCell(ACol: Longint; ARow: Longint; ARect: TRect; AState: TGridDrawState);
override;
```

C++

```
virtual __fastcall DrawCell(Longint ACol, Longint ARow, TRect ARect, TGridDrawState AState);
```

Description

DrawCell is called automatically whenever a cell in the grid is drawn. If the DefaultDrawing property is true, the background of the cell is painted before DrawCell is called, and the 3D effects of the fixed cells or the focus rectangle around the focused cell are drawn after DrawCell. Descendants of TCustomDrawGrid can override this method to draw the contents of the cell or to block the OnDrawCell event.

See Also

DefaultDrawing

FixedCols

FixedRows

OnDrawCell

47.1.2.4.11 TValueListEditor.EditListGetItems Method

Pascal

```
procedure EditListGetItems(ACol: Integer; ARow: Integer; Items: TStrings);
```

C++

```
__fastcall EditListGetItems(int ACol, int ARow, TStrings Items);
```

Description

This is EditListGetItems, a member of class TValueListEditor.

47.1.2.4.12 TValueListEditor.FindRow Method

Locates the row that corresponds to a specified key name.

Pascal

```
function FindRow(const KeyName: string; var Row: Integer): Boolean;
```

C++

```
__fastcall Boolean FindRow(const UnicodeString KeyName, int Row);
```

Description

Call FindRow to locate the row in the value list editor that has the value specified by KeyName in the first column. This corresponds to a name/value pair in the string list where the name part is KeyName.

If a row in the value list editor has the specified key name, FindRow returns true and the Row parameter returns the index of the first row found with the specified name. If the specified name does not correspond to any of the rows in the value list editor, FindRow returns false.

See Also

Keys (see page 4822)

47.1.2.4.13 TValueListEditor.GetCell Method

Pascal

```
function GetCell(ACol: Integer; ARow: Integer): string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetCell(int ACol, int ARow);
```

Description

This is GetCell, a member of class TValueListEditor.

47.1.2.4.14 TValueListEditor.GetColCount Method

Pascal

```
function GetColCount: Integer;
```

C++

```
__fastcall int GetColCount();
```

Description

This is GetColCount, a member of class TValueListEditor.

47.1.2.4.15 TValueListEditor.GetEditLimit Method

Provides the interface for a method that limits the number of characters the user can type into cells.

Pascal

```
function GetEditLimit: Integer; override;
```

C++

```
virtual __fastcall int GetEditLimit();
```

Description

The inplace editor calls GetEditLimit to determine if a limit should be placed on the number of characters a user can type into a cell. GetEditLimit sets no limits. Descendants of TCustomGrid override this method to limit the text for certain cells.

See Also

InplaceEditor

47.1.2.4.16 TValueListEditor.GetEditMask Method

Generates an OnGetEditMask event.

Pascal

```
function GetEditMask(ACol: Longint; ARow: Longint): string; override;
```

C++

```
virtual __fastcall UnicodeString GetEditMask(Longint ACol, Longint ARow);
```

Description

The inplace editor calls GetEditMask to obtain any input mask associated with the cell specified by the ACol and ARow parameters. GetEditMask returns the Value parameter set by the OnGetEditMask event handler. If there is no OnGetEditMask

event handler, GetEditMask returns an empty string, indicating that there is no edit mask.

See Also

OnGetEditMask

47.1.2.4.17 TValueListEditor.GetEditStyle Method

Indicates whether a specified cell has an ellipsis button or drop-down arrow.

Pascal

```
function GetEditStyle(ACol: Longint; ARow: Longint): TEditStyle; override;
```

C++

```
virtual __fastcall TEditStyle GetEditStyle(Longint ACol, Longint ARow);
```

Description

GetEditStyle indicates whether the specified cell can only be edited directly (esSimple), whether it has an ellipsis button that users can click to launch an editor for modifying field values (esEllipsis), or whether it has a drop-down arrow that users can click to display a pick list of possible values (esPickList).

ACol is the column that contains the cell. ARow is the row that contains the cell.

In TCustomGrid, GetEditStyle always returns esSimple, indicating that there is no ellipsis button or drop-down arrow. Descendant classes can override this methods if they provide a way to add pick lists or ellipsis buttons to cells.

47.1.2.4.18 TValueListEditor.GetEditText Method

Pascal

```
function GetEditText(ACol: Integer; ARow: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetEditText(int ACol, int ARow);
```

Description

This is GetEditText, a member of class TValueListEditor.

47.1.2.4.19 TValueListEditor.GetPickList Method

Pascal

```
function GetPickList(Values: TStrings; ClearFirst: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean GetPickList(TStrings Values, Boolean ClearFirst = True);
```

Description

This is GetPickList, a member of class TValueListEditor.

47.1.2.4.20 TValueListEditor.GetRowCount Method

Pascal

```
function GetRowCount: Integer;
```

C++

```
__fastcall int GetRowCount();
```

Description

This is GetRowCount, a member of class TValueListEditor.

47.1.2.4.21 TValueListEditor.InsertRow Method

Inserts a row in the value list editor and a corresponding string into the associated TStrings object.

Pascal

```
function InsertRow(const KeyName: string; const Value: string; Append: Boolean): Integer;
```

C++

```
__fastcall int InsertRow(const UnicodeString KeyName, const UnicodeString Value, Boolean
Append);
```

Description

Call InsertRow to add a name/value pair to the value list editor and the TStrings object it is editing.

InsertRow adds a string of the form

`KeyName=Value`

to the Strings (see page 4834) property and updates the display to reflect the change.

If the current row of the value list editor is empty (has no name or value portion), InsertRow assigns the new KeyName and Value to the current row. Otherwise, if the Append (see page 3520) parameter is true, the new string appears after the currently selected row, while if Append (see page 3520) is false, the new string appears before the currently selected row.

Note: InsertRow inserts the specified row regardless of the value of KeyOptions (see page 4821). The keyAdd member of KeyOptions (see page 4821) controls only whether the user can add new rows by pressing the Insert (see page 3601) key or navigating past the last row.

See Also

Cells (see page 4806)

Strings (see page 4834)

KeyOptions (see page 4821)

DisplayOptions (see page 4812)

47.1.2.4.22 TValueListEditor.IsEmptyRow Method

Pascal

```
function IsEmptyRow: Boolean;
```

C++

```
__fastcall Boolean IsEmptyRow();
```

Description

This is IsEmptyRow, a member of class TValueListEditor.

47.1.2.4.23 TValueListEditor.KeyDown Method

Provides special processing when the user presses a key.

Pascal

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

C++

```
virtual __fastcall KeyDown(Word Key, TShiftState Shift);
```

Description

After generating an OnKeyDown event, KeyDown calls CanGridAcceptKey to screen out problematic key combinations. KeyDown moves to the appropriate cell in response to navigational key combinations, or sets the EditorMode property to true when the user presses F2.

See Also

CanGridAcceptKey

EditorMode

KeyPress

OnKeyDown

47.1.2.4.24 TValueListEditor.MouseDown Method

Provides special processing when the user presses a mouse button.

Pascal

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);
override;
```

C++

```
virtual __fastcall MouseDown(TMouseButton Button, TShiftState Shift, int X, int Y);
```

Description

MouseDown hides the editor window so that it can move selection to the cell that receives the mouse click. If the mouse lands in the hot region for beginning a move or resize of a row or column, MouseDown starts the move or resize, and provides visual feedback by changing the cursor. If the editor window was hidden, it is brought back up in its new location. Finally, MouseDown generates an OnMouseDown event.

See Also

HideEditor

InplaceEditor

MouseMove

MouseUp

ShowEditor

OnMouseDown

47.1.2.4.25 TValueListEditor.Refresh Method

Updates the display to reflect the current value of the Strings (see page 4834) property.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh to ensure that the value list editor accurately reflects the value of the Strings (see page 4834) property. For example, call Refresh after programmatically editing the value of the Strings (see page 4834) property.

See Also

RestoreCurrentRow (see page 4798)

Repaint

Strings (see page 4834)

47.1.2.4.26 TValueListEditor.Resize Method

Respond to control resize.

Pascal

```
procedure Resize; override;
```

C++

```
virtual __fastcall Resize();
```

Description

Resize is called automatically immediately after the control's dimensions change. As implemented in TControl, Resize calls the control's OnResize event handler if it is assigned.

See Also

OnResize

47.1.2.4.27 TValueListEditor.RestoreCurrentRow Method

Undoes all edits to the current row.

Pascal

```
function RestoreCurrentRow: Boolean;
```

C++

```
__fastcall Boolean RestoreCurrentRow();
```

Description

Call RestoreCurrentRow to back out edits to the current cell in the value list editor. RestoreCurrentRow backs out edits to the current cell if it is still in edit mode, or deletes a newly inserted row.

The user can accomplish the same effect by pressing the Escape key.

See Also

Cells (see page 4806)

InsertRow (see page 4796)

47.1.2.4.28 TValueListEditor.RowMoved Method

Generates an OnRowMoved event.

Pascal

```
procedure RowMoved(FromIndex: Longint; ToIndex: Longint); override;
```

C++

```
virtual __fastcall RowMoved(Longint FromIndex, Longint ToIndex);
```

Description

RowMoved is called immediately after a row in the grid changes position. It overrides the inherited method to generate an OnRowMoved event. Descendants of TCustomDrawGrid can override this method to make internal adjustments or to block the OnRowMoved event.

See Also

OnRowMoved

47.1.2.4.29 TValueListEditor.SelectCell Method

Determines whether a particular cell in the grid can be selected.

Pascal

```
function SelectCell(ACol: Longint; ARow: Longint): Boolean; override;
```

C++

```
virtual __fastcall Boolean SelectCell(Longint ACol, Longint ARow);
```

Description

Grid controls call SelectCell internally before attempting to move selection to a the cell identified by the ACol and ARow parameters. ACol and ARow are the column and row indexes for the cell, where the first column has index 0, and the top row has index 0.

SelectCell generates an OnSelectCell event. This method then returns true, unless the OnSelectCell event handler indicates that the cell should not be selected. Descendants of TCustomDrawGrid can override this method to check the properties of the information displayed in the cell or block the OnSelectCell event.

See Also

OnSelectCell

47.1.2.4.30 TValueListEditor.SetCell Method

Pascal

```
procedure SetCell(ACol: Integer; ARow: Integer; const Value: string); virtual;
```

C++

```
virtual __fastcall SetCell(int ACol, int ARow, const UnicodeString Value);
```

Description

This is SetCell, a member of class TValueListEditor.

47.1.2.4.31 TValueListEditor.SetEditText Method

Pascal

```
procedure SetEditText(ACol: Integer; ARow: Integer; const Value: string); override;
```

C++

```
virtual __fastcall SetEditText(int ACol, int ARow, const UnicodeString Value);
```

Description

This is SetEditText, a member of class TValueListEditor.

47.1.2.4.32 TValueListEditor.SetStrings Method

Pascal

```
procedure SetStrings(const Value: TStrings); virtual;
```

C++

```
virtual __fastcall SetStrings(const TStrings Value);
```

Description

This is SetStrings, a member of class TValueListEditor.

47.1.2.4.33 TValueListEditor.StringsChanging Method

Pascal

```
procedure StringsChanging; dynamic;
```

C++

```
__fastcall StringsChanging();
```

Description

This is StringsChanging, a member of class TValueListEditor.

47.1.2.4.34 TValueListEditor.TitleCaptionsStored Method

Pascal

```
function TitleCaptionsStored: Boolean;
```

C++

```
__fastcall Boolean TitleCaptionsStored();
```

Description

This is TitleCaptionsStored, a member of class TValueListEditor.

47.1.2.5 TValueListEditor Properties

The properties of the TValueListEditor class are listed here.

Protected Properties

	Name	Description
R	EditList (see page 4817)	This is EditList, a member of class TValueListEditor.

Public Properties

	Name	Description
	Cells (see page 4806)	Lists the strings for each cell in the editor.
R	ColCount (see page 4807)	Indicates the number of columns in the value list editor.
	ItemProps (see page 4821)	Specifies various constraints on the way users can enter a value for each item in the list.
	Keys (see page 4822)	Lists the strings for each cell in the name (key) column.
R	RowCount (see page 4831)	Indicates the number of rows in the value list editor.
	Values (see page 4835)	Lists the strings for each value from the value column.
	VisibleColCount (see page 4837)	Indicates the number of scrollable columns visible in the grid.
	VisibleRowCount (see page 4838)	Indicates the number of scrollable rows visible in the grid.

Published Properties

	Name	Description
	Align (see page 4802)	Determines how the control aligns within its container (parent control).
	Anchors (see page 4805)	Specifies how the control is anchored to its parent.
	BiDiMode (see page 4806)	Specifies the bi-directional mode for the control.
	BorderStyle (see page 4806)	Determines whether a single line border is drawn around the grid.
	Color (see page 4807)	Specifies the background color of the control.
	Constraints (see page 4808)	Specifies the size constraints for the control.
	Ctl3D (see page 4809)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
	DefaultColWidth (see page 4809)	Determines the width (in pixels) of all columns that have not been explicitly resized.
	DefaultDrawing (see page 4811)	Determines whether the Cells are painted when the grid is drawn.
	DefaultRowHeight (see page 4811)	Specifies the height (in pixels) of all rows that have not been explicitly resized.
	DisplayOptions (see page 4812)	Controls aspects of the value list editor's appearance.
	DoubleBuffered (see page 4812)	Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.
	DragCursor (see page 4814)	Indicates the image used to represent the mouse pointer when the control is being dragged.
	DragKind (see page 4814)	Specifies whether the control is being dragged normally or for docking.
	DragMode (see page 4814)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
	DropDownRows (see page 4816)	Limits the length of the drop-down list that appears for values that use a pick list.
	Enabled (see page 4817)	Controls whether the control responds to mouse, keyboard, and timer events.
	FixedColor (see page 4817)	Specifies the background color of the fixed rows and columns in the grid.
	FixedCols (see page 4818)	Specifies the number of columns on the left of the grid that cannot be scrolled.
	Font (see page 4818)	Controls the attributes of text written on or in the control.
	GridLineWidth (see page 4819)	Specifies the width (in pixels) of the lines that separate the cells of the grid.
	KeyOptions (see page 4821)	Controls the way the value list editor responds to user keystrokes.
	Options (see page 4822)	Specifies various display and behavioral properties of the grid.
	ParentBiDiMode (see page 4824)	Specifies whether the control uses its parent's BiDiMode.
	ParentColor (see page 4824)	Determines where a control looks for its color information.
	ParentCtl3D (see page 4825)	Determines where a component looks to determine if it should appear three dimensional.
	ParentDoubleBuffered (see page 4825)	ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.
	ParentFont (see page 4825)	Determines where a control looks for its font information.
	ParentShowHint (see page 4826)	Determines where a control looks to find out if its Help Hint should be shown.
	PopupMenu (see page 4826)	Identifies the pop-up menu associated with the control.
	ScrollBars (see page 4832)	Specifies whether the grid includes horizontal and vertical scroll bars.

	ShowHint (see page 4832)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
	Strings (see page 4834)	Provides access to the string list that is being edited.
	TabOrder (see page 4834)	Indicates the position of the control in its parent's tab order.
	TitleCaptions (see page 4835)	Specifies the captions that label the columns of the value list editor.
	Visible (see page 4836)	Determines whether the component appears onscreen.

47.1.2.5.1 TValueListEditor.Align Property

Determines how the control aligns within its container (parent control).

Pascal

```
property Align;
```

C++

```
__property Align;
```

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

C++ Examples:

```
/*
This example shows how to use a splitter to divide a form
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
*/

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  TSplitter* ps = new TSplitter(Form1); // These objects will all be cleaned up by the owner
(Form1).
  TFileListBox* pflb = new TFileListBox(Form1); // The parent (Form1) will clean up these
objects.
  TDirectoryListBox* pdlb = new TDirectoryListBox(Form1);
```

```
  // Line up the directory list box on the left of the form.
  pdlb->Parent = Form1;
  pdlb->Align = alLeft;
  pdlb->Width = Form1->ClientWidth/3;

  // Now use the splitter to divide the directory pane from the file pane.
  ps->Parent = Form1;
  ps->Left = pdlb->Left + pdlb->Width + 1; // Move to right of directory list.
  ps->Align = pdlb->Align; // Give same alignment as directory list.

  // Each pane must be at least one quarter of the form's width.
  ps->MinSize = Form1->ClientWidth/4;

  // Finally, create the last pane - a file list box.
  pflb->Parent = Form1;
  pflb->Align = alClient;
  pdlb->FileList = pflb; // Link the file list box to the directory list box.
}
/*
Dynamically create a TProgressBar control and align it to
the bottom of the form.
*/

#include <ComCtrls.hpp>
#include <Controls.hpp>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TProgressBar *ProgressBar = new TProgressBar(this);  // The owner will clean this up.
  ProgressBar->Parent = this;
  ProgressBar->Align = alBottom;
  TButton *Button1 = new TButton(this);  // The owner will clean this up.
  Button1->Parent = this;
}
/*
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
*/
#include <Comctrls.hpp>
TPageControl* ppc;
const int MAXTABS = 3;
TTabSheet* pts[MAXTABS];
const char * ppcTabTitles[] = { "ShortString", "Orders", "Items", "Parts" };
int iTabTitles = sizeof(ppcTabTitles)/sizeof(ppcTabTitles[0]);
const TColor colorPalette[12] = {
  clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia,
  clTeal, clNavy, clMaroon, clLime, clOlive, clPurple};

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ppc = new TPageControl(this); // The owner (this) will clean this up.
    ppc->Parent = this;
    ppc->Align = alClient;
    for (int i=0;i<iTabTitles;i++)
  {
        pts[i] = new TTabSheet(this); // The owner (this) will clean this up.
        pts[i]->PageControl = ppc;
        pts[i]->Name = AnsiString("pts") + ppcTabTitles[i];
        pts[i]->Caption = ppcTabTitles[i];
        pts[i]->Brush->Color = colorPalette[i];
  }
}
```

Delphi Examples:

```
{
This example shows how to use a splitter to divide a form
```

```
into resizable panes.  All objects on the form are created
dynamically in the OnCreate event handler of the form. To
run this example, add ExtCtrls and FileCtrl to the uses
clause of the form's implementation section.
Note that when creating a splitter dynamically at runtime,
it is important to set its position to the appropriate side
of the control it will resize.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  Split: TSplitter;
  Files: TFileListBox;
  Dirs: TDirectoryListBox;
begin
  { first add a directory list box to the form }
  Dirs := TDirectoryListBox.Create(Form1);
  { line it up on the left of the form }
  Dirs.Parent := Form1;
  Dirs.Align := alLeft;
  Dirs.Width := Form1.ClientWidth div 3;
{ now add the splitter to divide the directory pane from the file pane }
  Split := TSplitter.Create(Form1);
  Split.Parent := Form1;
  { make sure the splitter is to the right of the directory list box! }
  Split.Left := Dirs.Left + Dirs.Width + 1;
  Split.Align := Dirs.Align; { give it the same alignment as the directory }
  { each pane must be at least one quarter of the form's width }
  Split.MinSize := Form1.ClientWidth div 4;
  { Finally, create the last pane - a file list box }
  Files := TFileListBox.Create(Form1);
  Files.Parent := Form1;
  Files.Align := alClient;
  Dirs.FileList := Files; { link the file list box to the directory list box }
end;
{
Dynamically create a TProgressBar control and align it to
the bottom of the form.
}

var
  I: Integer;

procedure TForm1.Button1Click(Sender: TObject);
var
  ProgressBar: TProgressBar;
begin
  ProgressBar := TProgressBar.Create(Self);
  with ProgressBar do
  begin
    Parent := Self;
    Align := alBottom;
  end;
end;
{
This example dynamically creates a Page Control, then a
series of Tab Sheets on the Page Control.
}
var
  PageControl1: TPageControl;
  pts: array[0..3] of TTabSheet;
const
  TabTitles: array[0..3] of ShortString = ('Customer', 'Orders', 'Items', 'Parts' );

procedure TForm1.FormCreate(Sender: TObject);
const
  colorpalette: Array[0..11] of TColor = (
```

```
      clRed, clGreen, clYellow, clBlue, clWhite, clFuchsia, clTeal,
      clNavy, clMaroon, clLime, clOlive, clPurple);
  var
    i: Integer;
  begin
    PageControl1 := TPageControl.Create(Self);
    PageControl1.Parent := Self;
    PageControl1.Align := alClient;
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i]:= TTabSheet.Create(PageControl1);
      with pts[i] do
      begin
        PageControl := PageControl1;
        Name := 'ts' + TabTitles[i];
        Caption := TabTitles[i];
        Brush.Color := colorPalette[i];
     end;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    i: Integer;
  begin
    for i := Low(TabTitles) to High(TabTitles) do
    begin
      pts[i].Free;
    end;
    PageControl1.Free;
  end;
```

See Also

Parent

Anchors

47.1.2.5.2 TValueListEditor.Anchors Property

Specifies how the control is anchored to its parent.

Pascal

```
property Anchors;
```

C++

```
__property Anchors;
```

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

See Also

Parent

Align

47.1.2.5.3 TValueListEditor.BiDiMode Property

Specifies the bi-directional mode for the control.

Pascal

```
property BiDiMode;
```

C++

```
__property BiDiMode;
```

Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

See Also

UseRightToLeftAlignment

UseRightToLeftReading

UseRightToLeftScrollBar

Including Bi-directional Functionality in Applications

47.1.2.5.4 TValueListEditor.BorderStyle Property

Determines whether a single line border is drawn around the grid.

Pascal

```
property BorderStyle;
```

C++

```
__property BorderStyle;
```

Description

Set BorderStyle to bsSingle to add a single line border around the grid's image. Set BorderStyle to bsNone to omit the border.

47.1.2.5.5 TValueListEditor.Cells Property

Lists the strings for each cell in the editor.

Pascal

```
property Cells [ACol, ARow: Integer]: string;
```

C++

```
__property UnicodeString Cells[int ACol, ARow];
```

Description

Use Cells to access the string in a particular cell. ACol is the column coordinate of the cell, and ARow is the row coordinate of the cell. ACol can be 0 (for the name column) or 1 (for the value column). ARow can range from 0 (for the title row) to the number of strings in the associated TStrings descendant.

To access the strings in the name column, use the Keys (see page 4822) property instead. To access the strings in the value column, use the Values (see page 4835) property instead. To access the underlying string list, use the Strings (see page 4834) property.

See Also

Keys (see page 4822)

Values (see page 4835)

Strings (see page 4834)

47.1.2.5.6 TValueListEditor.ColCount Property

Indicates the number of columns in the value list editor.

Pascal

```
property ColCount;
```

C++

```
__property ColCount;
```

Description

TValueListEditor (see page 4784) redeclares the ColCount property, which it inherits from TCustomGrid, to change it to a read-only property. The value of ColCount is always 2 plus the number of fixed columns, because the value list editor is designed to display only the name and value of the strings in the associated string list.

See Also

Cells (see page 4806)

RowCount (see page 4831)

FixedCols

47.1.2.5.7 TValueListEditor.Color Property

Specifies the background color of the control.

Pascal

```
property Color;
```

C++

```
__property Color;
```

Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

C++ Examples:

```
/*
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```

Delphi Examples:

```
{
The following code changes the color of a shape control
using the Color dialog box. The example displays the Color
dialog box when the Button1 button is clicked, allowing the
user to select a color with the dialog box. When a color
value is selected with the dialog box, it is assigned to
the shape control.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Shape1.Brush.Color := ColorDialog1.Color;
end;
```

See Also

ParentColor

47.1.2.5.8 TValueListEditor.Constraints Property

Specifies the size constraints for the control.

Pascal

```
property Constraints;
```

C++

```
__property Constraints;
```

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Note: Constraints apply to the height of the form. However, the height of the form depends on how large Windows makes the title bar. For example, Windows XP uses much larger title bars than other windowing systems. To work around this, note the ClientHeight when you design your form and set the constraints in the FormCreate event. In the following example, ClientHeight is represented by x: `Constraints.MinHeight := x + Height - ClientHeight`.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

See Also

AutoSize

Align

Anchors

OnConstrainedResize

OnCanResize

47.1.2.5.9 TValueListEditor.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Pascal

```
property Ctl3D;
```

C++

```
__property Ctl3D;
```

Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later, with the exception of Windows XP Home, where it is used on occasion.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

See Also

ParentCtl3D

BevelKind

BevelInner

BevelOuter

BorderWidth

47.1.2.5.10 TValueListEditor.DefaultColWidth Property

Determines the width (in pixels) of all columns that have not been explicitly resized.

Pascal

```
property DefaultColWidth;
```

C++

```
__property DefaultColWidth;
```

Description

Set DefaultColWidth to change the size of all columns in the grid. When DefaultColWidth is set, columns that have been resized using the mouse or by setting the ColWidths property are given the DefaultColWidth as well.

When new columns are added to the grid using the ColCount property, they are created with a width of DefaultColWidth.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
```

```
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
  }
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)

{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clBackground;
  DrawGrid1.Canvas.FillRect(Rect);
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
  end;
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
```

```
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColCount

ColWidths

DefaultRowHeight

47.1.2.5.11 TValueListEditor.DefaultDrawing Property

Determines whether the Cells are painted when the grid is drawn.

Pascal

```
property DefaultDrawing;
```

C++

```
__property DefaultDrawing;
```

Description

Set DefaultDrawing to true to have the grid automatically paint the cells in the grid, including adding the background color, the 3-D effects on the fixed cells, and the focus rectangle around the selected cell. Set DefaultDrawing to false to skip the painting of the cells when the grid is drawn.

When DefaultDrawing is false, the cells should be painted using the DrawCell method. For TCustomGrid, DrawCell is an abstract method. Descendants of TCustomGrid use DrawCell to provide their own customized drawing code, or to generate an OnDrawCell event.

See Also

DrawCell

OnDrawCell

47.1.2.5.12 TValueListEditor.DefaultRowHeight Property

Specifies the height (in pixels) of all rows that have not been explicitly resized.

Pascal

```
property DefaultRowHeight;
```

C++

```
__property DefaultRowHeight;
```

Description

Set DefaultRowHeight to change the size of all rows in the grid. When DefaultRowHeight is set, rows that have been resized using the mouse or by setting the RowHeights property are given the DefaultRowHeight as well.

When new rows are added to the grid using the RowCount property, they are created with a height of DefaultRowHeight.

See Also

DefaultColWidth

RowCount

RowHeights

47.1.2.5.13 TValueListEditor.DisplayOptions Property

Controls aspects of the value list editor's appearance.

Pascal

```
property DisplayOptions: TDisplayOptions;
```

C++

```
__property TDisplayOptions DisplayOptions;
```

Description

Use DisplayOptions to control whether titles appear at the top of the columns in the value list editor and how the control responds when it is resized.

See Also

KeyOptions (see page 4821)

Options

TitleCaptions (see page 4835)

47.1.2.5.14 TValueListEditor.DoubleBuffered Property

Determines whether the control's image is rendered directly to the window or painted to an in-memory bitmap first.

Pascal

```
property DoubleBuffered;
```

C++

```
__property DoubleBuffered;
```

Description

When DoubleBuffered is false, the windowed control paints itself directly to the window. When DoubleBuffered is true, the windowed control paints itself to an in-memory bitmap that is then used to paint the window. Double buffering reduces the amount of flicker when the control repaints, but is more memory intensive.

When a windowed control is a dock site and has an associated dock manager, it must be double-buffered.

Note: Some controls, such as TRichEdit, can't paint themselves into a bitmap. For such controls, DoubleBuffered must be set to false.

C++ Examples:

```
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
```

```
    Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
```

```
      end;
    end;
  end;
```

See Also

DockSite

DockManager

PaintWindow

47.1.2.5.15 TValueListEditor.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

Pascal

```
property DragCursor;
```

C++

```
__property DragCursor;
```

Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

Note: To make a custom cursor available for the DragCursor property, see the Cursor property.

See Also

Cursor

DragKind

DragMode

47.1.2.5.16 TValueListEditor.DragKind Property

Specifies whether the control is being dragged normally or for docking.

Pascal

```
property DragKind;
```

C++

```
__property DragKind;
```

Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

See Also

DragCursor

DragMode

47.1.2.5.17 TValueListEditor.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

Pascal

```
property DragMode;
```

C++

```
__property DragMode;
```

Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

C++ Examples:

```
/*
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
*/
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
      int X, int Y)
{
  if (Source != dynamic_cast<TObject *>(TreeView1) || TreeView1->Selected == NULL)
    return;
  THitTests HT = TreeView1->GetHitTestInfoAt(X, Y);
  TNodeAttachMode AttachMode;
  TTreeNode *Item = TreeView1->GetNodeAt(X, Y);
  if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
    AttachMode = naAddChild;
  else if (HT.Contains(htNowhere))
    AttachMode = naAdd;
  else if (HT.Contains(htOnIndent))
    AttachMode = naInsert;
  else
    return;
  TreeView1->Selected->MoveTo(Item, AttachMode);
}

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
  Accept = (Source == dynamic_cast<TObject *>(TreeView1));
}
```

Delphi Examples:

```
{
The following code uses GetNodeAt to add a dragged node as a
child of the node under the mouse when it is dropped.  This
example requires a populated TreeView.  Also, the TTreeView
DragMode property must be set to dmAutomatic, and the TTreeView
OnDragOver event handler must be implemented to accept the
drop.
}
procedure TForm1.TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  AnItem: TTreeNode;
  AttachMode: TNodeAttachMode;
  HT: THitTests;
begin
  if TreeView1.Selected = nil then Exit;
  HT := TreeView1.GetHitTestInfoAt(X, Y);
  AnItem := TreeView1.GetNodeAt(X, Y);
  if (HT - [htOnItem, htOnIcon, htNowhere, htOnIndent] <> HT) then
  begin
    if (htOnItem in HT) or (htOnIcon in HT) then
      AttachMode := naAddChild
```

```
      else if htNowhere in HT then AttachMode := naAdd
      else if htOnIndent in HT then AttachMode := naInsert;
      TreeView1.Selected.MoveTo(AnItem, AttachMode);
    end;
  end;

  procedure TForm1.TreeView1DragOver(Sender, Source: TObject; X, Y: Integer;
    State: TDragState; var Accept: Boolean);
  begin
    Accept := Source is TTreeView;
  end;
```

See Also

BeginDrag

DragCursor

EndDrag

OnDragDrop

OnDragOver

OnEndDrag

OnStartDrag

DragKind

Arranging the Order of Persistent Columns

47.1.2.5.18 TValueListEditor.DropDownRows Property

Limits the length of the drop-down list that appears for values that use a pick list.

Pascal

```
property DropDownRows: Integer;
```

C++

```
__property int DropDownRows;
```

Description

Use DropDownRows to indicate the maximum height of the drop-down list that appears when the user clicks the drop-down button on values that have a pick list. DropDownRows is an upper limit: if the pick list has fewer entries that DropDownRows, the drop-down list includes only a line for every item in the list.

DropDownRows is ignored for values that do not have a pick list. To provide a value in the value list editor with a drop-down button and pick list, you do either of the following:

For the appropriate entry in the ItemProps (see page 4821) property, set the EditStyle property to esPickList and set the PickList property to the list of items.

Write (see page 3707) an OnGetPickList (see page 4860) event handler that supplies the items in the list.

See Also

OnGetPickList (see page 4860)

Values (see page 4835)

ItemProps (see page 4821)

47.1.2.5.19 TValueListEditor.EditList Property

Pascal

```
property EditList: TInplaceEditList;
```

C++

```
__property TInplaceEditList EditList;
```

Description

This is EditList, a member of class TValueListEditor.

47.1.2.5.20 TValueListEditor.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

Pascal

```
property Enabled;
```

C++

```
__property Enabled;
```

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to False. Some disabled controls appear dimmed (for example: buttons, check boxes, labels), while others (container controls) simply lose their functionality without changing their appearance. If Enabled is set to False, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to True.

Note: This property applies to all TControl descendants.

See Also

Visible

DisableControls

EnableControls

Enabling Mouse

47.1.2.5.21 TValueListEditor.FixedColor Property

Specifies the background color of the fixed rows and columns in the grid.

Pascal

```
property FixedColor;
```

C++

```
__property FixedColor;
```

Description

Set FixedColor to specify the color for the nonscrolling cells in the grid. The scrolling region of the grid is drawn using the Color property.

See Also

Color

FixedCols

FixedRows

47.1.2.5.22 TValueListEditor.FixedCols Property

Specifies the number of columns on the left of the grid that cannot be scrolled.

Pascal

```
property FixedCols;
```

C++

```
__property FixedCols;
```

Description

Set FixedCols to create or get rid of nonscrolling columns. Nonscrolling columns appear at the left of the grid, and are always visible, even when the user scrolls the other columns in the grid. Use nonscrolling columns for displaying row titles or row numbers, or to implement a scroll lock that the user can set.

Nonscrolling columns are visually distinct from the scrollable columns in the grid. They are painted with a separate background color, and their cells have a raised, 3-D look.

Note: Grids must include at least one scrolling column. Do not set FixedCols to a value greater than ColCount - 1.

See Also

ColCount

DefaultDrawing

FixedColor

FixedRows

47.1.2.5.23 TValueListEditor.Font Property

Controls the attributes of text written on or in the control.

Pascal

```
property Font;
```

C++

```
__property Font;
```

Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

C++ Examples:

```
/*
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
```

```
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < Screen->Fonts->Count; i++)
   ComboBox1->Items->Add(Screen->Fonts->Strings[i]);
//  ComboBox1->Items = Screen->Fonts;  // Another way to do it.
}

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
  RichEdit1->Font->Name = ComboBox1->Text;
}
```

Delphi Examples:

```
{
To use this example, place a TComboBox and TRichEdit or
TMemo on a form.  During form creation, a list of font names
is loaded into the combo box. When the combo box is clicked,
the memo or Rich Edit control font is set to the
corresponding font name in the combo box.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  for i := 0 to Screen.Fonts.Count - 1 do
   ComboBox1.Items.Add(Screen.Fonts.Strings[i]);
//  ComboBox1.Items := Screen.Fonts;  // Another way to do it.
  RichEdit1.Lines.Text := 'How is it to be you?';
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Text;
end;
```

See Also

DesktopFont

ParentFont

Displaying and Editing Text in a Memo Control

47.1.2.5.24 TValueListEditor.GridLineWidth Property

Specifies the width (in pixels) of the lines that separate the cells of the grid.

Pascal

```
property GridLineWidth;
```

C++

```
__property GridLineWidth;
```

Description

Set GridLineWidth to make the lines that separate the cells in the grid heavier or lighter. When GridLineWidth is zero, no separators are drawn between the cells of the grid.

GridLineWidth will have no effect if the Options property does not include goFixedHorzLine, goFixedVertLine, goHorzLine, or goVertLine.

Note: Values greater than 3 pixels are not recommended for applications that run on Windows 9x or Windows ME because of distortions that can appear.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (DrawGrid1->DefaultColWidth > 90)
    DrawGrid1->GridLineWidth = 2;
  else
    DrawGrid1->GridLineWidth = 1;
}

void __fastcall TForm1::DrawGrid1DrawCell(TObject *Sender, int ACol, int ARow, TRect &Rect,
          TGridDrawState State)
{
  int index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  ImageList1->Draw(DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
  }
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow, bool
&CanSelect)

{
  CanSelect = True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
This example includes a draw grid on a form. When the
application runs and the form is created, the width of the
lines on the draw grid changes if the default column width of
the grid is over 90 pixels wide.  This example requires a
populated image list, or can just display an empty grid.
Set the drawgrid DefaultColWidth and GridLineWidth statically
to values the cause the button click to do something.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if DrawGrid1.DefaultColWidth > 90 then
    DrawGrid1.GridLineWidth := 2
  else
    DrawGrid1.GridLineWidth := 1;
end;

procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
```

```
    index: Integer;
  begin
    index := ARow * DrawGrid1.ColCount + ACol;
    DrawGrid1.Canvas.Brush.Color := clBackground;
    DrawGrid1.Canvas.FillRect(Rect);
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
    end;
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
```

See Also

FixedCols

FixedRows

Options

47.1.2.5.25 TValueListEditor.ItemProps Property

Specifies various constraints on the way users can enter a value for each item in the list.

Pascal

```
property ItemProps [const KeyOrIndex: Variant]: TItemProp;
```

C++

```
__property TItemProp ItemProps[Variant const KeyOrIndex];
```

Description

Use ItemProps to indicate, for each item:

- Whether the item is read only.
- The maximum length of strings the user can enter.
- An edit mask that limits the strings the user can enter as a value.
- Whether the item lets users drop down a pick list or launch a dialog to enter values.
- A fixed set of values from which users select (a pick list).

See Also

OnGetPickList (see page 4860)

Values (see page 4835)

47.1.2.5.26 TValueListEditor.KeyOptions Property

Controls the way the value list editor responds to user keystrokes.

Pascal

```
property KeyOptions: TKeyOptions;
```

C++

```
__property TKeyOptions KeyOptions;
```

Description

Use KeyOptions to control how users can edit name/value pairs using the value list editor.

See Also

Keys (see page 4822)

Strings (see page 4834)

47.1.2.5.27 TValueListEditor.Keys Property

Lists the strings for each cell in the name (key) column.

Pascal

```
property Keys [Index: Integer]: string;
```

C++

```
__property UnicodeString Keys[int Index];
```

Description

Use Keys to access the name (or key) value for a particular row. That is, Keys assumes the strings in the associated string list have the form Name=Value, and returns the Name portion.

Index identifies the Row of the value list editor. It takes a value from 0 to the number of rows. When DisplayOptions (see page 4812) includes doColumnTitles, the value of Keys is the title of the first column when Index is 0, and is the Name portion of the string when Index is greater than 0. When DisplayOptions (see page 4812) does not include doColumnTitles, the value of Keys is the Name portion of the string in the specified row.

To access the strings in the value column, use the Values (see page 4835) property instead. To access the underlying string list, use the Strings (see page 4834) property.

See Also

Cells (see page 4806)

Values (see page 4835)

Strings (see page 4834)

DisplayOptions (see page 4812)

Names

47.1.2.5.28 TValueListEditor.Options Property

Specifies various display and behavioral properties of the grid.

Pascal

```
property Options: TGridOptions;
```

C++

```
__property TGridOptions Options;
```

Description

Set Options to include the desired properties for the grid. See the type Grids.TGridOptions for definitions of these properties.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

ColWidths

EditorMode

GridLineWidth

RowHeights

Selection

TabStops

47.1.2.5.29 TValueListEditor.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

Pascal

```
property ParentBiDiMode;
```

C++

```
__property ParentBiDiMode;
```

Description

Use ParentBiDiMode to get or set whether the control uses its parent's BiDiMode. When ParentBiDiMode is true, the control's BiDiMode property is always equal to that of its parent. If ParentBiDiMode is false, the control uses its own BiDiMode property.

Set ParentBiDiMode to true for all controls in a form in order to ensure that the entire user interface adjusts to Middle Eastern locales in a uniform manner.

When the value of the control's BiDiMode property changes, ParentBiDiMode becomes false automatically.

See Also

Parent

BiDiMode

47.1.2.5.30 TValueListEditor.ParentColor Property

Determines where a control looks for its color information.

Pascal

```
property ParentColor;
```

C++

```
__property ParentColor;
```

Description

To have a control use the same color as its parent control, set ParentColor to true. If ParentColor is false, the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See Also

Color

Parent

ParentFont

ParentShowHint

47.1.2.5.31 TValueListEditor.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

Pascal

```
property ParentCtl3D;
```

C++

```
__property ParentCtl3D;
```

Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

See Also

Ctl3D

47.1.2.5.32 TValueListEditor.ParentDoubleBuffered Property

ParentDoubleBuffered defers the DoubleBuffered property of this component to the value of the parent's DoubleBuffered property.

Pascal

```
property ParentDoubleBuffered;
```

C++

```
__property ParentDoubleBuffered;
```

Description

ParentDoubleBuffered is a boolean indicating that the DoubleBuffered property of this component is overridden by the value of the parent's DoubleBuffered property.

47.1.2.5.33 TValueListEditor.ParentFont Property

Determines where a control looks for its font information.

Pascal

```
property ParentFont;
```

C++

```
__property ParentFont;
```

Description

To have a control use the same font as its parent control, set ParentFont to true. If ParentFont is false, the control uses its own

Font property. Many controls default ParentFont to true so that all the controls in a form or other container present a uniform appearance. When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See Also

Font

Parent

ParentColor

ParentShowHint

47.1.2.5.34 TValueListEditor.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

Pascal

```
property ParentShowHint;
```

C++

```
__property ParentShowHint;
```

Description

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note: Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See Also

Hint

Parent

ParentColor

ParentFont

ShowHint

47.1.2.5.35 TValueListEditor.PopupMenu Property

Identifies the pop-up menu associated with the control.

Pascal

```
property PopupMenu;
```

C++

```
__property PopupMenu;
```

Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

C++ Examples:

```
/*
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  PopupMenu1->AutoPopup = true;
  Edit1->PopupMenu = PopupMenu1;
  Edit2->PopupMenu = PopupMenu1;
}

/*
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
*/
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CopyToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->CutToClipboard();
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}

void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  TComponent *pComponent = PopupMenu1->PopupComponent;
  if (pComponent)
  {
    if (pComponent->ClassNameIs("TEdit"))
      (dynamic_cast<TEdit *>(pComponent))->PasteFromClipboard();
```

```
    else
      MessageBeep(0);
  }
  else
    MessageBeep(0);
}
/*
This example demonstrates how to add and delete menu items
to a popup menu at runtime and assign an event handler to
the OnClick event.  Place a TPopupMenu and three buttons on
the form named "AddButton", "EditButton",  and
"DestroyButton" and add OnClick events to all three buttons.
Put the TPopupMenu in the PopupMenu property of the form.
Place the PopupMenuItemsClick procedure in the TForm1 type
declaration so that it can be used as the method call for
the menu item OnClick event.
*/
void __fastcall TForm1::AddButtonClick(TObject *Sender)
{
    // The owner (PopupMenu1) will cleanup this menu item.
    TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
    int index = PopupMenu1->Items->Count;
    PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
    NewItem->Caption = "Menu Item " + IntToStr(index);
    NewItem->Tag = index;
    NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
    TNotifyEvent();
}

void __fastcall TForm1::PopupMenuItemsClick(TObject *Sender)
{
    TMenuItem *ClickedItem = dynamic_cast<TMenuItem *>(Sender);
    if (ClickedItem)
    {
        switch (ClickedItem->Tag)
        {
            case 0:
            {
                ShowMessage("first item clicked");
                break;
            }
            case 1:
            {
                ShowMessage("second item clicked");
                break;
            }
            case 2:
            {
                ShowMessage("third item clicked");
                break;
            }
            case 3:
            {
                ShowMessage("fourth item clicked");
                break;
            }
        }
    }
}

/*
To edit or destroy an item, grab its pointer via the Items
property.
*/
void __fastcall TForm1::EditButtonClick(TObject *Sender)
{
```

```
    const int index = 1;
    TMenuItem *ItemToEdit = PopupMenu->Items->Items[index];
    ItemToEdit->Caption = "Changed Caption";
}

void __fastcall TForm1::DestroyButtonClick(TObject *Sender)
{
    const int index = 2;
    TMenuItem *ItemToDelete = PopupMenu->Items->Items[index];
    delete ItemToDelete;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    const int num_items = 4;
    for (int index = 0; index < num_items; ++index)
    {
        // The owner (PopupMenu1) will cleanup this menu item.
        TMenuItem *NewItem = new TMenuItem(PopupMenu1); // create the new item
        PopupMenu1->Items->Add(NewItem);// add it to the Popupmenu
        NewItem->Caption = "Menu Item " + IntToStr(index);
        NewItem->Tag = index;
        NewItem->OnClick = PopupMenuItemsClick;// assign it an event handler
        TNotifyEvent();
    }
}
```

Delphi Examples:

```
{
This example uses two edit boxes and one pop-up menu on a form.
The pop-up menu contains Cut, Copy, and Paste commands. This code
makes the pop-up menu available to both edit boxes:
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := True;
  Edit1.PopupMenu := PopupMenu1;
  Edit2.PopupMenu := PopupMenu1;
end;

{
These are the cut, copy, and paste OnClick events for the commands
on the pop-up menu. Double click on the pop-up menu and add three
TMenuItems.  Then set the captions to "Cut", "Copy" and "Paste".
The names will be automatically set to "Cut1", "Copy1", and
"Paste1".  Then double click on the OnClick events and they will
be named "Cut1Click", "Copy1Click" and "Paste1Click" respectively.
Note: only the selected portion of the string in the edit box will
be copied, cut or replaced.
}
procedure TForm1.Copy1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
   Edit1.CopyToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CopyToClipboard
  else
    Beep;
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  if PopupMenu1.PopupComponent = Edit1 then
    Edit1.CutToClipboard
  else if PopupMenu1.PopupComponent = Edit2 then
    Edit2.CutToClipboard
```

```
    else
      Beep;
  end;

  procedure TForm1.Paste1Click(Sender: TObject);
  begin
    if PopupMenu1.PopupComponent = Edit1 then
      Edit1.PasteFromClipboard
    else if PopupMenu1.PopupComponent = Edit2 then
      Edit2.PasteFromClipboard
    else
      Beep;
  end;
  {
  This example demonstrates how to add and delete menu items
  to a popup menu at runtime and assign an event handler to
  the OnClick event.  Place a TPopupMenu and three buttons on
  the form named "AddButton", "EditButton",  and
  "DestroyButton" and add OnClick events to all three buttons.
  Put the TPopupMenu in the PopupMenu property of the form.
  Place the PopupMenuItemsClick procedure in the TForm1 type
  declaration so that it can be used as the method call for
  the menu item OnClick event.
  }

  type
    TForm1 = class(TForm)
      AddButton: TButton;
      EditButton: TButton;
      DestroyButton: TButton;
      PopupMenu1: TPopupMenu;
      procedure AddButtonClick(Sender: TObject);
      procedure EditButtonClick(Sender: TObject);
      procedure DestroyButtonClick(Sender: TObject);
      procedure FormCreate(Sender: TObject);
    private
      { Private declarations }
    public
      procedure PopupMenuItemsClick(Sender: TObject);

    end;

  var
    Form1: TForm1;

  implementation

  {$R *.dfm}

  procedure TForm1.AddButtonClick(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
      // The owner (PopupMenu1) will cleanup this menu item.
    NewItem := TMenuItem.Create(PopupMenu1); // create the new item
    index := PopupMenu1.Items.Count;
    PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
    NewItem.Caption := 'Menu Item ' + IntToStr(index);
    NewItem.Tag := index;
    NewItem.OnClick :=
      PopupMenuItemsClick; // assign it an event handler
  end;

  procedure TForm1.PopupMenuItemsClick(Sender: TObject);
  begin
```

```
    with Sender as TMenuItem do
    begin
      case Tag of
        0:  ShowMessage('first item clicked');
        1:  ShowMessage('second item clicked');
        2:  ShowMessage('third item clicked');
        3:  ShowMessage('fourth item clicked');
      end;
    end;
  end;

  {
  To edit or destroy an item, grab its pointer via the Items
  property.
  }
  procedure TForm1.EditButtonClick(Sender: TObject);
  var
    ItemToEdit: TMenuItem;
  begin
    ItemToEdit := PopupMenu.Items[1];
    ItemToEdit.Caption := 'Changed Caption';
  end;

  procedure TForm1.DestroyButtonClick(Sender: TObject);
  var
    ItemToDelete: TMenuItem;
  begin
    ItemToDelete := PopupMenu.Items[2];
    ItemToDelete.Free;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    index: Integer;
    NewItem: TMenuItem;
  begin
    for index := 0 to 3 do
    begin
        // The owner (PopupMenu1) will cleanup this menu item.
      NewItem := TMenuItem.Create(PopupMenu1); // create the new item
      PopupMenu1.Items.Add(NewItem);// add it to the Popupmenu
      NewItem.Caption := 'Menu Item ' + IntToStr(index);
      NewItem.Tag := index;
      NewItem.OnClick :=
        PopupMenuItemsClick; // assign it an event handler
    end;
  end;
```

See Also

AutoPopup

Popup

OnContextPopup

47.1.2.5.36 TValueListEditor.RowCount Property

Indicates the number of rows in the value list editor.

Pascal

```
property RowCount: Integer;
```

C++

```
__property int RowCount;
```

Description

TValueListEditor (see page 4784) redeclares the RowCount property, which it inherits from TCustomGrid, to change it to a read-only property. The value of RowCount is the number of strings in the Strings (see page 4834) property plus the number of fixed rows (typically one, for the column titles).

See Also

Strings (see page 4834)

ColCount (see page 4807)

FixedRows

47.1.2.5.37 TValueListEditor.ScrollBars Property

Specifies whether the grid includes horizontal and vertical scroll bars.

Pascal

```
property ScrollBars;
```

C++

```
__property ScrollBars;
```

Description

Use ScrollBars to give the grid horizontal or vertical scroll bars.

Note: If all the cells in the grid fit in the ClientWidth, no horizontal scroll bar appears, even if ScrollBars is ssHorizontal or ssBoth. If all the cells fit in the ClientHeight, no vertical scroll bar appears, even if ScrollBars is ssVertical or ssBoth.

See Also

ClientHeight

ClientWidth

47.1.2.5.38 TValueListEditor.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Pascal

```
property ShowHint;
```

C++

```
__property ShowHint;
```

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either

the control's own ShowHint property must be true, or

the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint

property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

C++ Examples:

```
/*
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Edit1->Hint = "Enter your name";
  Edit1->ShowHint = true;

  char string[10];
  char index[3];
  for(int i = 1; i <= 10; i++)
  {
    strcpy(string, "Item");
    itoa(i, index, 10);
    strcat(string, index);
    ListBox1->Items->Add(string);
  }
  ListBox1->Hint = "Select an item";
  ListBox1->ShowHint = true;
  Application->ShowHint = true;
}
/*
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
*/
```

Delphi Examples:

```
{
This example uses an edit box and a list box on a form.
Items are added to the list box and a Help Hint is assigned
to both controls. The last statement enables the Help Hints
for the entire application.
}
procedure TForm1.FormCreate(Sender: TObject);
var
  I: Integer;
begin
  Edit1.Hint := 'Enter your name';
  Edit1.ShowHint := True;
  with ListBox1 do
  begin
    for I := 1 to 10 do
      Items.Add('Item ' + IntToStr(I));
    Hint := 'Select an item';
    ShowHint := True;
  end;
  Application.ShowHint := True;
end;
{
Note: To see an example that displays the hints of controls
in some place other than in a Help Hint box, see the OnHint
example.
}
```

See Also

Hint

ParentShowHint

ShowHint

Displaying Fly-over Help

47.1.2.5.39 TValueListEditor.Strings Property

Provides access to the string list that is being edited.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Use Strings to access the string list that the value list editor displays. Strings should be a TStrings descendant that contains strings of the form Name=Value.

At design time, add values to the Strings property using the string list editor. At runtime, you can use the methods of TStrings to add strings to the list one by one, or assign an existing TStrings descendant to the Strings property. When you set Strings at runtime, TValueListEditor (see page 4784) copies the strings from the TStrings object you assign: it does not take ownership of the object itself.

When Strings is assigned, the value list editor displays the name portion (TStrings . ::Names) of the strings in its key column and the value portion (TStrings . ::Values (see page 4835)) in the value column.

See Also

Names

Values

KeyOptions (see page 4821)

47.1.2.5.40 TValueListEditor.TabOrder Property

Indicates the position of the control in its parent's tab order.

Pascal

```
property TabOrder;
```

C++

```
__property TabOrder;
```

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

See Also

FindNextControl

TabStop

47.1.2.5.41 TValueListEditor.TitleCaptions Property

Specifies the captions that label the columns of the value list editor.

Pascal

```
property TitleCaptions: TStrings;
```

C++

```
__property TStrings TitleCaptions;
```

Description

Use TitleCaptions to provide titles for the columns of the value list editor. TitleCaptions is a TStrings descendant that contains two strings: the first string labels the name (key) column, and the second string labels the value column. If TitleCaptions contains less than two strings, missing captions are treated as empty strings. If TitleCaptions contains more than two strings, all strings except the first two are ignored.

Note: TitleCaptions is only used when the DisplayOptions (see page 4812) property includes doColumnTitles.

See Also

DisplayOptions (see page 4812)

47.1.2.5.42 TValueListEditor.Values Property

Lists the strings for each value from the value column.

Pascal

```
property Values [const Key: string]: string;
```

C++

```
__property UnicodeString Values[UnicodeString const Key];
```

Description

Use Values to access the value associated with a specified name from the first (key) column. Key specifies a string that appears in the first column of the value list editor. Values is the corresponding string from the second (value) column.

To access the strings in the key column, use the Keys (see page 4822) property instead. To access the underlying string list, use the Strings (see page 4834) property.

Note: Setting Values with Key set to a name that is not in the list adds a new string to Strings (see page 4834) and a new row to the value list editor.

See Also

Cells (see page 4806)

Keys (see page 4822)

Strings (see page 4834)

Values

47.1.2.5.43 TValueListEditor.Visible Property

Determines whether the component appears onscreen.

Pascal

```
property Visible;
```

C++

```
__property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
```

```
  form OnCreate event handler populates each sheet with an
  edit control placed somewhere on the client area of the
  sheet.  Unchecking the checkbox will cause the client area
  (not the tab area) of the Tab Control to become invisible.
  }
  procedure TForm1.CheckBox1Click(Sender: TObject);
  begin
    PageControl1.ActivePage.Visible := CheckBox1.Checked;
  end;

  procedure TForm1.PageControl1Change(Sender: TObject);
  begin
    CheckBox1.Checked := PageControl1.ActivePage.Visible;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  var
    i: Integer;
  begin
    for i := 0 to 9 do
      with TTabSheet.Create(Self) do
      begin
        PageControl := PageControl1;
        Caption := 'TabSheet' + IntToStr(i);
        with TEdit.Create(Self) do
        begin
          Parent := PageControl1.Pages[i];
          Left := Random(PageControl1.ActivePage.ClientWidth - Width);
          Top := Random(PageControl1.ActivePage.ClientHeight - Height);
        end;
      end;
      PageControl1Change(Sender);
  end;
```

See Also

Enabled

Hide

Show

VisibleChanging

47.1.2.5.44 TValueListEditor.VisibleColCount Property

Indicates the number of scrollable columns visible in the grid.

Pascal

```
property VisibleColCount;
```

C++

```
__property VisibleColCount;
```

Description

Use VisibleColCount to determine the number of scrollable columns fully visible in the grid. VisibleColCount does not include the fixed columns counted by the FixedCols property. It does not include any partially visible columns on the right edge of the grid.

See Also

ColCount

FixedCols

VisibleRowCount

47.1.2.5.45 TValueListEditor.VisibleRowCount Property

Indicates the number of scrollable rows visible in the grid.

Pascal

```
property VisibleRowCount;
```

C++

```
__property VisibleRowCount;
```

Description

Use VisibleRowCount to determine the number of scrollable rows fully visible in the grid. VisibleRowCount does not include the fixed rows counted by the FixedRows property. It does not include any partially visible rows on the bottom of the grid.

See Also

FixedRows

RowCount

VisibleColCount

47.1.2.6 TValueListEditor Events

The events of the TValueListEditor class are listed here.

Published Events

	Name	Description
	OnClick (see page 4839)	Occurs when the user clicks the control.
	OnContextPopup (see page 4841)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
	OnDblClick (see page 4841)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
	OnDragDrop (see page 4842)	Occurs when the user drops an object being dragged.
	OnDragOver (see page 4845)	Occurs when the user drags an object over a control.
	OnDrawCell (see page 4848)	Occurs when a cell in the grid needs to be drawn.
	OnEditButtonClick (see page 4849)	Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.
	OnEndDock (see page 4850)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
	OnEndDrag (see page 4850)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
	OnEnter (see page 4851)	Occurs when a control receives the input focus.
	OnExit (see page 4854)	Occurs when the input focus shifts away from one control to another.
	OnGetEditMask (see page 4858)	Occurs when the in-place editor requests an edit mask.
	OnGetEditText (see page 4859)	Occurs when the in-place editor requests the value of a cell.
	OnGetPickList (see page 4860)	Supplies a pick list for values associated with a particular key.
	OnKeyDown (see page 4861)	Occurs when a user presses any key while the control has focus.
	OnKeyPress (see page 4864)	Occurs when a key is pressed.
	OnKeyUp (see page 4865)	Occurs when the user releases a key that has been pressed.
	OnMouseActivate (see page 4867)	Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.
	OnMouseDown (see page 4868)	Occurs when the user presses a mouse button with the mouse pointer over a control.
	OnMouseEnter (see page 4870)	Occurs when the user moves the mouse into a control.

	OnMouseLeave (see page 4871)	Occurs when the user moves the mouse outside of a control.
	OnMouseMove (see page 4871)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
	OnMouseUp (see page 4873)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
	OnMouseWheelDown (see page 4875)	Occurs when the mouse wheel is rotated downward.
	OnMouseWheelUp (see page 4876)	Occurs when the mouse wheel is rotated upward.
	OnRowMoved (see page 4876)	Occurs immediately after the position of a row changes.
	OnSelectCell (see page 4877)	Occurs before a cell in the grid is selected.
	OnSetEditText (see page 4879)	Occurs when the user edits the value of a cell.
	OnStartDock (see page 4879)	Occurs when the user begins to drag a control with a DragKind of dkDock.
	OnStartDrag (see page 4880)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
	OnStringsChange (see page 4881)	Occurs immediately after the names or values in the string list change.
	OnStringsChanging (see page 4881)	Occurs immediately before the names or values in the string list change.
	OnTopLeftChanged (see page 4882)	Occurs immediately after the TopRow property or the LeftCol property changes.
	OnValidate (see page 4882)	Occurs when focus shifts away from a cell in the value list editor.

47.1.2.6.1 TValueListEditor.OnClick Event

Occurs when the user clicks the control.

Pascal

```
property OnClick;
```

C++

```
__property OnClick;
```

Description

Use the OnClick event handler to respond when the user clicks the control. If the control has an associated action, and that action has an OnExecute method, the action's OnExecute method responds to click events unless it is superseded by an OnClick event handler.

Usually OnClick occurs when the user presses and releases the left mouse button with the mouse pointer over the control. This event can also occur when:

- The user selects an item in a grid, outline, list, or combo box by pressing an arrow key.
- The user presses Spacebar while a button or check box has focus.
- The user presses Enter when the active form has a default button (specified by the Default property).
- The user presses Esc when the active form has a cancel button (specified by the Cancel property).
- The user presses the accelerator key for a button or check box. For example, if the value of the Caption property of a check box is '&Bold', the B is underlined at runtime and the OnClick event of the check box is triggered when the user presses Alt+B. However, focus does not move to the control in these instances.
- The Checked property of a radio button is set to true.
- The value of the Checked property of a check box is changed.
- The Click method of a menu item is called.

For a form, an OnClick event occurs when the user clicks a blank area of the form or on a disabled component.

C++ Examples:

```
/*
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
```

```
(not the tab area) of the Tab Control to become invisible.
*/
void __fastcall TForm1::PageControl1Change(TObject *Sender)
{
  CheckBox1->Checked = PageControl1->ActivePage->Visible;
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
  PageControl1->ActivePage->Visible = CheckBox1->Checked;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  for (int i = 0; i < 10; i++)
  {
    TTabSheet *pPage = new TTabSheet(this); // The owner will clean this up.
    pPage->PageControl = PageControl1;
    pPage->Caption = AnsiString("Page") + IntToStr(i);
    TEdit *pEdit = new TEdit(this); // The owner will clean this up.
    pEdit->Parent = pPage;
    pEdit->Left = random(pPage->ClientWidth - pEdit->Width);
    pEdit->Top = random(pPage->ClientHeight - pEdit->Height);
  }
  PageControl1Change(Sender);
}
```

Delphi Examples:

```
{
This examples requires a TPageControl and a TCheckBox.  The
form OnCreate event handler populates each sheet with an
edit control placed somewhere on the client area of the
sheet.  Unchecking the checkbox will cause the client area
(not the tab area) of the Tab Control to become invisible.
}
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  PageControl1.ActivePage.Visible := CheckBox1.Checked;
end;

procedure TForm1.PageControl1Change(Sender: TObject);
begin
  CheckBox1.Checked := PageControl1.ActivePage.Visible;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  for i := 0 to 9 do
    with TTabSheet.Create(Self) do
    begin
      PageControl := PageControl1;
      Caption := 'TabSheet' + IntToStr(i);
      with TEdit.Create(Self) do
      begin
        Parent := PageControl1.Pages[i];
        Left := Random(PageControl1.ActivePage.ClientWidth - Width);
        Top := Random(PageControl1.ActivePage.ClientHeight - Height);
      end;
    end;
    PageControl1Change(Sender);
end;
```

See Also

OnDblClick

Cancel (see page 2045)

Default (see page 2049)

Click

OnExecute

Action

Adding Silent Video Clips to an Application

47.1.2.6.2 TValueListEditor.OnContextPopup Event

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

Pascal

```
property OnContextPopup;
```

C++

```
__property OnContextPopup;
```

Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

Note: Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

See Also

TPopupMenu

PopupMenu

AutoPopup

47.1.2.6.3 TValueListEditor.OnDblClick Event

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

Pascal

```
property OnDblClick;
```

C++

```
__property OnDblClick;
```

Description

Use the OnDblClick event to respond to mouse double-clicks.

See Also

OnClick

Responding to User Actions at Runtime

47.1.2.6.4 TValueListEditor.OnDragDrop Event

Occurs when the user drops an object being dragged.

Pascal

```
property OnDragDrop;
```

C++

```
__property OnDragDrop;
```

Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
```

```
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
```

```
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
  end;
end;
```

See Also

BeginDrag

DragDrop

Dragging

EndDrag

OnDragOver

OnEndDrag

OnStartDrag

TDragObject

47.1.2.6.5 TValueListEditor.OnDragOver Event

Occurs when the user drags an object over a control.

Pascal

```
property OnDragOver;
```

C++

```
__property OnDragOver;
```

Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

C++ Examples:

```
/*
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
*/
void __fastcall TForm1::FormDragDrop(TObject *Sender, TObject *Source, int X, int Y)

{
  reinterpret_cast<TForm1 *>(Source)->Left = X;
  reinterpret_cast<TForm1 *>(Source)->Top = Y;
}

void __fastcall TForm1::FormDragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = Source->ClassNameIs("TLabel");
}

void __fastcall TForm1::Panel1DragOver(TObject *Sender, TObject *Source, int X, int Y,
          TDragState State, bool &Accept)
{
  Accept = false;
```

```
  if ((Source->ClassNameIs("TLabel")) && (State == dsDragEnter))
    dynamic_cast<TLabel *>(Source)->EndDrag(false);
}
/*
This code comes from an application that contains a list box
and three labels, each with a different font and color.  The
DragMode property for each of the labels is dmAutomatic.
The user can select a label and drag it to a list box and
drop it. When the label is dropped, the items in the list
box assume the color and font of the dropped label.
*/

// This OnDragOver event handler permits the list box to
// accept a dropped label:
void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept)
{
   Accept = Source->ClassNameIs("TLabel");
}

/*
This OnDragDrop event handler implements the drop behavior.
*/
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source, int X, int Y)
{
  if (Sender->ClassNameIs("TListBox") && Source->ClassNameIs("TLabel"))
  {
    TListBox *DestList = dynamic_cast<TListBox *>(Sender);
    DestList->Font = (dynamic_cast<TLabel *>(Source))->Font;
    DestList->Color = (dynamic_cast<TLabel *>(Source))->Color;
    DestList->DoubleBuffered = true;
    DestList->Color = clWindow;
  }
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  ListBox1->Items->Add("Not");
  ListBox1->Items->Add("In");
  ListBox1->Items->Add("Alphabetical");
  ListBox1->Items->Add("Order");

}
```

Delphi Examples:

```
{
This code is an OnDragOver event handler that won't allow a
label control to be dropped on a panel control and stops the
dragging of the label as soon as the user drags the label
onto the panel:
Note: Set the label's DragMode property to dmAutomatic.
}
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    procedure Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragOver(Sender, Source: TObject; X, Y: Integer;
      State: TDragState; var Accept: Boolean);
    procedure FormDragDrop(Sender, Source: TObject; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  TForm1(Source).Left := X;
  TForm1(Source).Top := Y;
end;

procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

procedure TForm1.Panel1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := False;
  if (Source is TLabel) and (State = dsDragEnter) then
    (Source as TLabel).EndDrag(False);
end;
{
This code requires a list box and three labels, each with a
different font and color.  The DragMode property for each of
the labels is dmAutomatic.  The user can select a label and
drag it to a list box and drop it. When the label is dropped,
the items in the list box assume the color and font of the
dropped label.
}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add('Not');
  ListBox1.Items.Add('In');
  ListBox1.Items.Add('Alphabetical');
  ListBox1.Items.Add('Order');
end;

// This OnDragOver event handler permits the list box to
// accept a dropped label:

procedure TForm1.ListBox1DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := Source is TLabel;
end;

// This OnDragDrop event handler implements the drop behavior.

procedure TForm1.ListBox1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
  if (Sender is TListBox) and (Source is TLabel) then
  begin
    with Sender as TListBox do
    begin
      Font := (Source as TLabel).Font;
      Color := (Source as TLabel).Color;
      DoubleBuffered := true;
      Color := clWindow;
    end;
```

```
  end;
end;
```

See Also

DragCursor

OnDragDrop

OnEndDrag

OnStartDrag

TDragObject

DragKind

OnEndDock

OnDockDrop

OnStartDock

47.1.2.6.6 TValueListEditor.OnDrawCell Event

Occurs when a cell in the grid needs to be drawn.

Pascal

```
property OnDrawCell;
```

C++

```
__property OnDrawCell;
```

Description

Write an OnDrawCell event handler to draw the contents of all the cells in the grid. Draw on the cell using the methods of the Canvas property. The Rect parameter indicates the location of the cell on the canvas. The Col and Row parameters indicate the column and row indexes of the cell that should be drawn. The State parameter indicates whether the cell has input focus, whether the cell is selected, and whether the cell is a fixed (nonscrolling) cell.

If the OnDrawCell event handler is not assigned, all cells in the draw grid will appear empty. If the DefaultDrawing property is true, the draw grid paints the background color of the cell before the OnDrawCell event, and draws a focus rectangle around the selected cell after the OnDrawCell event handler finishes drawing the contents of the cell. If the DefaultDrawing property is false, the OnDrawCell event handler should paint the background of the cell and provide all visual indication of selection and focus.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
```

```
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
  if (gdFocused in State) then
  begin
    DrawGrid1.Canvas.DrawFocusRect(Rect);
    Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
  end;
  ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
end;

procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
  var CanSelect: Boolean);
begin
  CanSelect:= True;
end;
```

See Also

Canvas

DefaultDrawing

47.1.2.6.7 TValueListEditor.OnEditButtonClick Event

Occurs when the user click the ellipsis button on a cell in the values column or types Ctrl+Enter while focus is on a value cell with an ellipsis button.

Pascal

```
property OnEditButtonClick: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnEditButtonClick;
```

Description

Write (see page 3707) an OnEditButtonClick to display a dialog that lets users edit a value in the Values (see page 4835) column. This event only occurs for items for which the ItemProps (see page 4821) property has an EditStyle of esEllipsis.

In the event handler, you can use the Row property to identify the row that the user is editing. The event handler (or the dialog that the event handler displays) can use the Keys (see page 4822) property to identify the current key value, and use that to get and set the corresponding entry in the Values (see page 4835) property that is being edited.

See Also

Values (see page 4835)

ItemProps (see page 4821)

EditStyle (see page 4782)

Row

Keys (see page 4822)

47.1.2.6.8 TValueListEditor.OnEndDock Event

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

Pascal

```
property OnEndDock;
```

C++

```
__property OnEndDock;
```

Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

See Also

DragKind

OnDragOver

OnDockDrop

OnStartDock

47.1.2.6.9 TValueListEditor.OnEndDrag Event

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

Pascal

```
property OnEndDrag;
```

C++

```
__property OnEndDrag;
```

Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

C++ Examples:

```
/*
```

```
/*
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
*/
void __fastcall TForm1::Label1EndDrag(TObject *Sender, TObject *Target, int X,
      int Y)
{
  if (Sender->ClassNameIs("TLabel"))
  {
    AnsiString S = (dynamic_cast<TLabel *>(Sender))->Name + " was dropped and ";
    if (Target)
      S = S + "accepted!";
    else
      S = S + "rejected.";
    Status->Caption = S;
  }
}
```

Delphi Examples:

```
{
This code displays a message in a label named Status. The
message displayed depends on whether the dragged label
control was successfully dropped into and accepted by a
another control.
}
procedure TForm1.Label1EndDrag(Sender, Target: TObject; X, Y: Integer);
var
  S: string;
begin
  S := (Sender as TLabel).Name + ' was dropped... and ';
  if Target <> nil then
    S := S + 'accepted!'
  else
    S := S + 'rejected!';
  Status.Caption := S;
end;
```

See Also

OnDragDrop

OnDragOver

OnStartDrag

DragKind

47.1.2.6.10 TValueListEditor.OnEnter Event

Occurs when a control receives the input focus.

Pascal

```
property OnEnter;
```

C++

```
__property OnEnter;
```

Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnEnter event occurs for the container before the OnEnter event of the contained control.

Similarly, an OnExit event of the container occurs after the OnExit event of the control in a container when focus moves to another control outside the container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

C++ Examples:

```
/*
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
*/
TWinControl *myActiveControl;

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  FontDialog1->Options << fdApplyButton;
  FontDialog1->Execute();
}

void __fastcall TForm1::FontDialog1Apply(TObject *Sender, HWND Wnd)
{
  if (myActiveControl->ClassNameIs("TEdit"))
   (dynamic_cast<TEdit *>(myActiveControl))->Font->Assign(FontDialog1->Font);
  else if (myActiveControl->ClassNameIs("TRichEdit"))
    (dynamic_cast<TRichEdit *>(myActiveControl))->SelAttributes->Assign(FontDialog1->Font);
  else
    MessageBeep(0);
}

void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  myActiveControl = ActiveControl;
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
```

```
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example takes an edit control, a button, and a rich
edit control on a form.  When the user presses the button, a
Font dialog appears.  When the user clicks the Apply (not OK)
button in the Font dialog, the currently selected font is
applied to the active control. Clicking the button sets the
ActiveControl to the button.  That is why we need to save
the ActiveControl in a shared OnEnter procedure.  In the rich
edit, only the text selected converts.
}
procedure TForm1.Button1Click(Sender: TObject);
begin
   FontDialog1.Options := FontDialog1.Options + [fdApplyButton];
   FontDialog1.Execute;
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
  myActiveControl := ActiveControl;
end;

procedure TForm1.FontDialog1Apply(Sender: TObject; Wnd: HWND);
begin
  if myActiveControl is TEdit then
    with myActiveControl as TEdit do
       Font.Assign(TFontDialog(Sender).Font)
  else if myActiveControl is TRichEdit then
    with myActiveControl as TRichEdit do
      SelAttributes.Assign(TFontDialog(Sender).Font)
  else
    Beep;
end;
{
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
system color for a window.  These event handlers could also
be shared.
}
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  Edit1.Color := clYellow;
end;

procedure TForm1.Edit1Exit(Sender: TObject);
begin
  Edit1.Color := clWindow;
end;

procedure TForm1.Memo1Enter(Sender: TObject);
begin
  Memo1.Color := clYellow;
end;

procedure TForm1.Memo1Exit(Sender: TObject);
```

```
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnExit

ActiveControl

OnActivate

Using a Single Navigator for Multiple Datasets

47.1.2.6.11 TValueListEditor.OnExit Event

Occurs when the input focus shifts away from one control to another.

Pascal

```
property OnExit;
```

C++

```
__property OnExit;
```

Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

When switching between controls in separate container controls such as the TPanel and the TGroupBox controls, an OnExit event occurs for the control inside the container before the OnExit event of the container.

Similarly, an OnEnter event of the container occurs before the OnEnter event of the control in a container when focus moves to a control inside a container.

For example, consider a form with an OK button and a group box that contains three radio buttons, where focus is currently on the OK button. When the user clicks one of the radio buttons, an OnExit event of the button occurs, followed by an OnEnter event on the group box, and finally an OnEnter event on the radio button that was clicked. If the user then clicks on the OK button, an OnExit event for the radio button occurs followed by an OnExit event for the group box, and then the button's OnEnter event occurs.

Note: In some control classes the ActiveControl property updates before the OnExit event occurs.

C++ Examples:

```
/*
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255 ,
as this validation test cannot be accomplished
using the edit mask property alone.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
```

```
{
  // initialize the maskedit boxes to contain the empty string
  MaskEdit1->Text = "";
  MaskEdit2->Text = "";
  MaskEdit3->Text = "";

  // use the Courier font in the maskedit boxes
  MaskEdit1->Font->Name = "Courier";
  MaskEdit2->Font->Name = "Courier";
  MaskEdit3->Font->Name = "Courier";

  // initialize the mask of each maskedit box
  // according to IP address coding conventions
  MaskEdit1->EditMask = "!099.099.099.099;1; ";
  MaskEdit2->EditMask = "!099.099.099.099;1; ";
  MaskEdit3->EditMask = "!099.099.099.099;1; ";

  MaskEdit2->OnExit = MaskEdit2Exit;
}

void __fastcall TForm1::MaskEdit2Exit(TObject *Sender)
{
  // extract the net and host address from the IP
  String IP = MaskEdit1->Text;
  int net1 = IP.SubString(0, 3).TrimRight().ToInt();
  int net2 = IP.SubString(5, 3).TrimRight().ToInt();
  int host1 = IP.SubString(9, 3).TrimRight().ToInt();
  int host2 = IP.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (net1 < 0 || net1 > 255 ||
      net2 < 0 || net2 > 255 ||
      host1 < 0 || host1 > 255 ||
      host2 < 0 || host2 > 255)
    throw(Exception("Not a valid IP address."));

  // extract the net and host mask from the subnet mask
  String mask = MaskEdit2->Text;
  int netmask1 = mask.SubString(0, 3).TrimRight().ToInt();
  int netmask2 = mask.SubString(5, 3).TrimRight().ToInt();
  int hostmask1 = mask.SubString(9, 3).TrimRight().ToInt();
  int hostmask2 = mask.SubString(13, 3).TrimRight().ToInt();

  // a range test that you cannot validate through edit masks
  if (netmask1 < 0 || netmask1 > 255 ||
      netmask2 < 0 || netmask2 > 255 ||
      hostmask1 < 0 || hostmask1 > 255 ||
      hostmask2 < 0 || hostmask2 > 255)
    throw(Exception("Not a valid subnet mask."));

  // compute the subnet address using bit-wise AND
  int sub_net1 = net1 & netmask1;
  int sub_net2 = net2 & netmask2;
  int sub_host1 = host1 & hostmask1;
  int sub_host2 = host2 & hostmask2;

  // display the subnet address
  MaskEdit3->Text =
    String(sub_net1) + "." + String(sub_net2) + "." +
    String(sub_host1) + "." + String(sub_host2);
}
/*
This example uses an edit box and a memo control on a form.
When either the edit box or the memo is the active control,
it is colored yellow. When the active control becomes
inactive, the color of the control returns to the Windows
```

```
system color for a window.  These event handlers could also
be shared.
*/
void __fastcall TForm1::Edit1Enter(TObject *Sender)
{
  Edit1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Exit(TObject *Sender)
{
  Edit1->Color = clWindow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Enter(TObject *Sender)
{
  Memo1->Color = clYellow;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1Exit(TObject *Sender)
{
  Memo1->Color = clWindow;
}
```

Delphi Examples:

```
{
This example uses three mask edit components on a form,
where the third mask edit box is set to be read only.
When the application runs, it initializes the mask edit boxes
to contain the empty string, to use the Courier font and to
use a particular mask, corresponding to IP address coding convetions.
The mask edit boxes display the IP address, the subnet mask
and the subnet address correspondingly. When the user exits the
subnet mask edit box, the subnet address is automatically calculated
using bit-wise AND operations and displayed in the third mask edit box.
An additional test is made to assure that the values in the IP address
and in the subnet mask are in the interval from 0 to 255,
as this validation test cannot be accomplished
using the edit mask property alone.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  { initialize the maskedit boxes to contain the empty string }
  MaskEdit1.Text := '';
  MaskEdit2.Text := '';
  MaskEdit3.Text := '';

  { use the Courier font in the maskedit boxes }
  MaskEdit1.Font.Name := 'Courier';
  MaskEdit2.Font.Name := 'Courier';
  MaskEdit3.Font.Name := 'Courier';

  {
  initialize the mask of each maskedit box
  according to IP address coding conventions
  }
  MaskEdit1.EditMask := '!099.099.099.099;1; ';
  MaskEdit2.EditMask := '!099.099.099.099;1; ';
  MaskEdit3.EditMask := '!099.099.099.099;1; ';

  MaskEdit2.OnExit := MaskEdit2Exit;
end;

procedure TForm1.MaskEdit2Exit(Sender: TObject);
var
  net1, net2, host1, host2,
  netmask1, netmask2, hostmask1, hostmask2,
```

```
    sub_net1, sub_net2, sub_host1, sub_host2: Integer;

    IP, mask: String;

  begin
    // extract the net and host address from the IP
    IP := MaskEdit1.Text;
    net1 := StrToInt(TrimRight(Copy(IP, 0, 3)));
    net2 := StrToInt(TrimRight(Copy(IP, 5, 3)));
    host1 := StrToInt(TrimRight(Copy(IP, 9, 3)));
    host2 := StrToInt(TrimRight(Copy(IP, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((net1 < 0) Or (net1 > 255) Or
        (net2 < 0) Or (net2 > 255) Or
        (host1 < 0) Or (host1 > 255) Or
        (host2 < 0) Or (host2 > 255)) then
      raise EArgumentException.Create('Not a valid IP address.');

    // extract the net and host mask from the subnet mask
    mask := MaskEdit2.Text;
    netmask1 := StrToInt(TrimRight(Copy(mask, 0, 3)));
    netmask2 := StrToInt(TrimRight(Copy(mask, 5, 3)));
    hostmask1 := StrToInt(TrimRight(Copy(mask, 9, 3)));
    hostmask2 := StrToInt(TrimRight(Copy(mask, 13, 3)));

    // a range test that you cannot validate through edit masks
    if ((netmask1 < 0) Or (netmask1 > 255) Or
        (netmask2 < 0) Or (netmask2 > 255) Or
        (hostmask1 < 0) Or (hostmask1 > 255) Or
        (hostmask2 < 0) Or (hostmask2 > 255)) then
      raise EArgumentException.Create('Not a valid subnet mask.');

    // compute the subnet address using bit-wise AND
    sub_net1 := net1 And netmask1;
    sub_net2 := net2 And netmask2;
    sub_host1 := host1 And hostmask1;
    sub_host2 := host2 And hostmask2;

    // display the subnet address
    MaskEdit3.Text :=
      IntToStr(sub_net1) + '.' + IntToStr(sub_net2) + '.' +
      IntToStr(sub_host1) + '.' + IntToStr(sub_host2);
  end;
  {
  This example uses an edit box and a memo control on a form.
  When either the edit box or the memo is the active control,
  it is colored yellow. When the active control becomes
  inactive, the color of the control returns to the Windows
  system color for a window.  These event handlers could also
  be shared.
  }
  procedure TForm1.Edit1Enter(Sender: TObject);
  begin
    Edit1.Color := clYellow;
  end;

  procedure TForm1.Edit1Exit(Sender: TObject);
  begin
    Edit1.Color := clWindow;
  end;

  procedure TForm1.Memo1Enter(Sender: TObject);
  begin
    Memo1.Color := clYellow;
  end;
```

```
procedure TForm1.Memo1Exit(Sender: TObject);
begin
  Memo1.Color := clWindow;
end;
```

See Also

OnEnter

ActiveControl

47.1.2.6.12 TValueListEditor.OnGetEditMask Event

Occurs when the in-place editor requests an edit mask.

Pascal

```
property OnGetEditMask;
```

C++

```
__property OnGetEditMask;
```

Description

Write an OnGetEditMask event handler to provide the in-place editor with an edit mask that describes the valid input strings a user can type into a cell. Set the Value parameter to the string that describes the edit mask for the cell specified by the ACol and ARow parameters. If the cell does not have an edit mask, set the Value parameter to an empty string.

OnGetEditMask does not occur unless the Options property includes goEditing.

C++ Examples:

```
/*
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
*/
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
}

void __fastcall TForm1::StringGrid1GetEditMask(TObject *Sender, int ACol, int ARow,
          UnicodeString &Value)
{
  if (StringGrid1->Cells[ACol][0] == "Phone Number")
    Value =  "!\(999\)000-0000;1";
}
```

Delphi Examples:

```
{
This example specifies an edit mask in a string grid to
limit the cells that contain telephone numbers, as indicated
by a label cell in the first row.  Include goEditing in the
TStringGrid Options property.  Set the title of one of the
columns to 'Phone Number'.  Set the DefaultColWidth large
enough to see a phone number.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
  StringGrid1.Cells[2, 0]:= 'Phone Number';
```

```
end;

procedure TForm1.StringGrid1GetEditMask(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  if StringGrid1.Cells[ACol, 0] = 'Phone Number' then
    Value :=  '!\(999\)000-0000;1';
end;
```

See Also

Options

EditMask

47.1.2.6.13 TValueListEditor.OnGetEditText Event

Occurs when the in-place editor requests the value of a cell.

Pascal

```
property OnGetEditText;
```

C++

```
__property OnGetEditText;
```

Description

Write an OnGetEditText event handler to provide the in-place editor with a string representing the contents of the cell. Set the Value parameter to the string for the cell specified by the ACol and ARow parameters. When the cell enters edit mode, the contents of the cell are drawn as the Value string returned by the OnGetEditText event handler. This image need not match the appearance of the cell when it is not in edit mode, which is drawn using the OnDrawCell event handler.

OnGetEditText does not occur unless the Options property includes goEditing.

C++ Examples:

```
/*
The following code lowercases the text for a string grid
cell when the user begins to edit it.
*/
void __fastcall TForm1::StringGrid1GetEditText(TObject *Sender, int ACol, int ARow,
          UnicodeString &Value)
{
  Value = AnsiLowerCase(Value);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
  StringGrid1->Cells[2][1] = "(408)984-1234";
  StringGrid1->Cells[2][2] = "(415)434-6789";
  StringGrid1->Cells[2][3] = "(650)696-2345";
  StringGrid1->Cells[2][4] = "(831)431-2422";
  StringGrid1->Cells[2][5] = "(408)506-9876";
  StringGrid1->Cells[1][0] = "Name";
  StringGrid1->Cells[1][1] = "Clem Myers";
  StringGrid1->Cells[1][2] = "Ed Satchel";
  StringGrid1->Cells[1][3] = "Johanna Mills";
  StringGrid1->Cells[1][4] = "Stella Gables";
  StringGrid1->Cells[1][5] = "Ted Thompson";
  StringGrid1->Cells[3][0] = "POB";
  StringGrid1->Cells[3][1] = "Detroit";
  StringGrid1->Cells[3][2] = "Memphis";
  StringGrid1->Cells[3][3] = "New Orleans";
```

```
  StringGrid1->Cells[3][4] = "Lander";
  StringGrid1->Cells[3][5] = "Round Pond";
  StringGrid1->Cells[4][0] = "State";
  StringGrid1->Cells[4][1] = "Michigan";
  StringGrid1->Cells[4][2] = "Tennessee";
  StringGrid1->Cells[4][3] = "Lousiana";
  StringGrid1->Cells[4][4] = "Wyoming";
  StringGrid1->Cells[4][5] = "Maine";
  StringGrid1->Cells[5][0] = "Last Graduated";
  StringGrid1->Cells[5][1] = "Michigan State";
  StringGrid1->Cells[5][2] = "Univ-> Colorado";
  StringGrid1->Cells[5][3] = "Univ-> Washington";
  StringGrid1->Cells[5][4] = "USC";
  StringGrid1->Cells[5][5] = "Bowdoin";
}
```

Delphi Examples:

```
{
The following code lowercases the text for a string grid
cell when the user begins to edit it.
}
procedure TForm1.StringGrid1GetEditText(Sender: TObject; ACol, ARow: Longint; var Value:
string);
begin
  Value := AnsiLowerCase(Value);
end;
```

See Also

OnDrawCell

OnSetEditText

47.1.2.6.14 TValueListEditor.OnGetPickList Event

Supplies a pick list for values associated with a particular key.

Pascal

```
property OnGetPickList: TGetPickListEvent;
```

C++

```
__property TGetPickListEvent OnGetPickList;
```

Description

Write (see page 3707) an OnGetPickList event handler to provide a drop-down list of values for specific names from the first (key) column. OnGetPickList is called when the value column gets focus and the name (key) column contains the string specified by KeyName.

If the ItemProps (see page 4821) property has an EditStyle of esPickList for this item, The Values (see page 4835) parameter of the event handler is initialized to the value of its PickList property for this item. Otherwise, Values (see page 4835) is initialized to an empty list. In either case, the event handler can change the list of values that appear in the drop-down pick list.

If the OnGetPickList event handler exits with a non-empty string list as the value of the Values (see page 4835) parameter, a drop-down arrow appears in the value column. The user can click this arrow button to display the pick list you supplied as the value of Values (see page 4835). This makes the Value column behave like a drop-down combo box: the user can select items from the list or type in an entirely different value.

See Also

Keys (see page 4822)

Values (see page 4835)

DropDownRows (see page 4816)

ItemProps (see page 4821)

PickList (see page 4783)

EditStyle (see page 4782)

47.1.2.6.15 TValueListEditor.OnKeyDown Event

Occurs when a user presses any key while the control has focus.

Pascal

```
property OnKeyDown;
```

C++

```
__property OnKeyDown;
```

Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to keyboard keys, including function keys and keys combined with the `Shift`, `Alt`, and `Ctrl` keys, and pressed mouse buttons.

Note: TCheckBox does not fire OnKeyDown events for arrow keys.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
```

```
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
/*
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
*/
#include <Printers.hpp>

void __fastcall TForm1::FormCreate(TObject *Sender)
{
   KeyPreview = True;

}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key, TShiftState Shift)

{
  if (Key == VK_ESCAPE && Printer()->Printing)
  {
    Printer()->Abort();
    MessageDlg("Printing aborted", mtInformation,
               TMsgDlgButtons() << mbOK,0);
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TMemo *Memo1 = new TMemo(Form1); // The parent (Form1) will clean up this object.
  Memo1->Parent = Form1;
  Memo1->Visible = True;
  Memo1->Width = 700;
  if (OpenDialog1->Execute())
  {
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);

    Printer()->BeginDoc();
    int X = 200;
    int Y = 200;
    for (int I = 0; I <= 10; I++)
      if (!Printer()->Aborted)
      {
        Printer()->Canvas->TextOut(X, Y, Memo1->Lines->Strings[I]);
        Y = Y + 80;
        if (Y > (Printer()->PageHeight - 300))
```

```
        {
          Y = 200;
          Printer()->NewPage();
          Sleep(1000);  // to give you time to abort!
        }
      }
    if (!Printer()->Aborted) Printer()->EndDoc();
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
{
The following code aborts a print job if the user presses
Esc. Note that you should set KeyPreview to true to ensure
that the OnKeyDown event handler of Form1 is called.
}

uses Printers;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y: Integer;
```

```
    Memo1 : TMemo;
    r: TRect;
  begin
    Memo1 := TMemo.Create(Form1);
    Memo1.Parent := Form1;
    Memo1.Visible := True;
    Memo1.Width := 700;
    if (OpenDialog1.Execute) then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

      Printer.BeginDoc;
      X := 200;
      Y := 200;
      for I := 0 to 140 do
        if (not Printer.Aborted) then
        begin
          Printer.Canvas.TextOut(X, Y, Memo1.Lines[I]);
          Y := Y + 80;
          if (Y > (Printer.PageHeight - 300)) then
          begin
            Y := 200;
            Printer.NewPage;
            Sleep(1000);  // to give you time to abort!
          end;
        end;
      if (not Printer.Aborted) then Printer.EndDoc;
    end;
  end;

  procedure TForm1.FormCreate(Sender: TObject);
  begin
     KeyPreview := True;
  end;

  procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
  begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
    Printer.Abort;
    MessageDlg('Printing aborted', mtInformation, [mbOK],0);
    end;
  end;
```

See Also

KeyPreview

OnKeyPress

OnKeyUp

47.1.2.6.16 TValueListEditor.OnKeyPress Event

Occurs when a key is pressed.

Pascal

```
property OnKeyPress;
```

C++

```
__property OnKeyPress;
```

Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

An application gets Windows WM_KEYDOWN messages for all keys when the user presses a key. These messages indirectly fire the OnKeyDown event. If you set the Key to 0 there you prevent only any further processing of this message. But for keys that generate characters Windows also produces WM_CHAR. At the time your OnKeyDown event fires, the WM_CHAR message for the key will already be in the message queue. Setting Key to 0 does not stop it from being delivered, so it fires the OnKeyPress event. You must set the Key to #0 to prevent the control from intercepting the WM_CHAR message.

This method of organizing key processing has advantages. Code that only deals with characters, including control characters like #13 for carriage return, #3 for Ctrl-C and so on, should go into the OnKeyPress event. Code that deals with keys that do not generate characters should be put into the OnKeyDown event.

C++ Examples:

```
/*
This event handler displays a message dialog box specifying
which key was pressed:
*/

void __fastcall TForm1::FormKeyPress(TObject *Sender, char &Key)
{
  char keyString[25];
  keyString[0] = Key;
  strcpy(&keyString[1], " Was Pressed");
  TMsgDlgButtons() << mbOK;
  MessageDlg(AnsiString(keyString), TMsgDlgType(mtInformation), TMsgDlgButtons() << mbOK, 0);
}
```

Delphi Examples:

```
{
This event handler displays a message dialog box specifying
which key was pressed:
}
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  MessageDlg(Key + ' has been pressed', mtInformation, [mbOK], 0)
end;
```

See Also

OnKeyDown

OnKeyUp

KeyPreview

47.1.2.6.17 TValueListEditor.OnKeyUp Event

Occurs when the user releases a key that has been pressed.

Pascal

```
property OnKeyUp;
```

C++

```
__property OnKeyUp;
```

Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

C++ Examples:

```
/*
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
*/
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
  if (Key == VK_F1)
  {
    FormColor = Form1->Color;
    Form1->Color = clAqua;
    if (ComboBox1->Text == "KeyDown")
      ComboBox1->Text = "KeyDown2";
    else
      ComboBox1->Text = "KeyDown";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key, TShiftState Shift)
{
  if (Key == VK_F1)
  {
    Form1->Color = FormColor;
    if (ComboBox1->Text == "KeyUp")
      ComboBox1->Text = "KeyUp2";
    else
      ComboBox1->Text = "KeyUp";
    Form1->Refresh();
    Memo1->Lines->Add(ComboBox1->Text);
  }
}
```

Delphi Examples:

```
{
This example changes a form's color to aqua when the user
presses the F1 key, even when a control on the form has the
focus. When the user releases the key, the form returns to
its original color.  Set the KeyPreview on the form to True.
Set the OnKeyDown event to FormKeyDown and set the OnKeyUp
event to FormKeyUp.  Set the KeyPreview on the form to True.
Behavior is dependent on keyboard driver(USB, serial, etc).
}
var
  FormColor: TColor;
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  KeyPreview := True;
  FormColor := Form1.Color;
end;

procedure TForm1.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := clAqua;
    if (ComboBox1.Text = 'KeyDown') then
      ComboBox1.Text := 'KeyDown2'
    else
      ComboBox1.Text := 'KeyDown';
    WriteLn(ComboBox1.Text);
  end;
end;

procedure TForm1.FormKeyUp(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if Key = VK_F1 then
  begin
    Form1.Color := FormColor;
    if (ComboBox1.Text = 'KeyUp') then
      ComboBox1.Text := 'KeyUp2'
    else
      ComboBox1.Text := 'KeyUp';
    WriteLn(ComboBox1.Text);
  end;
end;
```

See Also

OnKeyDown

OnKeyPress

47.1.2.6.18 TValueListEditor.OnMouseActivate Event

Occurs when the user presses a mouse button with the mouse pointer over a control and the parent form is not active.

Pascal

```
property OnMouseActivate;
```

C++

```
__property OnMouseActivate;
```

Description

Use the OnMouseActivate event handler to implement any special processing that should occur as a result of pressing a mouse button on a control when the parent top-level form is not active.

The event type is:

```
TMouseActivateEvent = procedure (Sender: TObject; Button: TMouseButton; ShiftState:
TShiftState;
X, Y: Integer; HitTest: Integer; var MouseActivate: TMouseActivate) of object;
```

When you click on a control and the parent top-level form is not active, this event fires on the control that the mouse cursor is over. MouseActivate is a protected virtual function in TControl. MouseActivate can be overridden in custom control descendants to provide special built-in processing of the OnMouseActivate events. If you leave the default value of MouseActivate as maDefault, the parent control fires the OnMouseActivate event and repeats the process all the way up to the top-level form. If no

control in the chain sets the value of MouseActivate, the behavior is the same as if a control set TMouseActivate to maActivate. At any point, a control can set the value of TMouseActivate and the parent processing stops.

Some controls set focus to themselves when they get a button down message so, in some cases, setting maNoActivate appears to have no effect. For instance, a TButton control sets focus to itself in the WM_LBUTTONDOWN message regardless of whether or not TMouseActivate is set to maNoActivate. In this case, setting maNoActivateAndEat will work because the top-level form is not activated and the WM_LBUTTONDOWN message is suppressed.

The HitTest parameter is the hit test value obtained from the WM_NCHITTEST message. See Windows.pas for valid values and their meanings. In most cases this value will be HTCLIENT, which means that the user clicked in the client area of a control. However, in the case of a top-level form, this can take other values such as HTCAPTION, or HTBOTTOM. This allows the code to decide which value to set for MouseActivate, based on the HitTest code. For example, the following code in the OnMouseActivate event handler for a TForm forces the user to click into a non-client area to activate the form:

```
if HitTest = HTCLIENT then
    MouseActivate := maNoActivateAndEat;
```

Even though this event is tied to how Windows processes WM_MOUSEACTIVATE, the actual handling of the message is done at the TControl level, which is the point in the VCL hierarchy where the control has no window handle. The OnMouseActivate event is then simulated since the all TControl descendants also have OnMouseDown, OnMouseMove, and OnMouseUp events. This information is applicable to both VCL Win32 and VCL.NET.

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

47.1.2.6.19 TValueListEditor.OnMouseDown Event

Occurs when the user presses a mouse button with the mouse pointer over a control.

Pascal

```
property OnMouseDown;
```

C++

```
__property OnMouseDown;
```

Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
```

```
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
```

```
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
  StatusBar1.Panels[1].Text := '';
  StatusBar1.Panels[2].Text := '';
  StatusBar1.Panels[3].Text := '';
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if ssLeft in Shift then { make sure button is down }
  begin
    if Y > StartY then
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
      end
    else
      begin
      StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
      StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
      end;
    if X > StartX then
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
      end
    else
      begin
      StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
      StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
      end;
  end;
end;
```

See Also

OnClick

OnMouseMove

OnMouseUp

MouseCapture

47.1.2.6.20 TValueListEditor.OnMouseEnter Event

Occurs when the user moves the mouse into a control.

Pascal

```
property OnMouseEnter;
```

C++

```
__property OnMouseEnter;
```

Description

Use the OnMouseEnter event handler to implement any special processing that should occur as a result of moving the mouse into a control when the parent top-level form is not active.

47.1.2.6.21 TValueListEditor.OnMouseLeave Event

Occurs when the user moves the mouse outside of a control.

Pascal

```
property OnMouseLeave;
```

C++

```
__property OnMouseLeave;
```

Description

Use the OnMouseLeave event handler to implement any special processing that should occur as a result of moving the mouse outside of a control when the parent top-level form is not active.

47.1.2.6.22 TValueListEditor.OnMouseMove Event

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

Pascal

```
property OnMouseMove;
```

C++

```
__property OnMouseMove;
```

Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}
```

```
// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
}
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
Y: Integer);
begin
  StartX := X;
  StartY := Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
begin
  Form1.Canvas.Rectangle(StartX, StartY, X, Y);
  StatusBar1.Panels[0].Text := '';
```

```
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseUp

MouseCapture

47.1.2.6.23 TValueListEditor.OnMouseUp Event

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

Pascal

```
property OnMouseUp;
```

C++

```
__property OnMouseUp;
```

Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

C++ Examples:

```
/*
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
```

```
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
released the mouse button.  While the user drags the mouse,
the location of the top, left, bottom, and right sides of
the rectangle are displayed in the status bar.
*/
int StartX, StartY; // Declare at the top of the form's unit
// Use this code as the OnMouseDown event handler of the form:
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  StartX = X;
  StartY = Y;
}

// Use this code as the OnMouseUp event handler of the form:
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState Shift,
int X, int Y)
{
  Form1->Canvas->Rectangle(StartX, StartY, X, Y);
  StatusBar1->Panels->Items[0]->Text = "";
  StatusBar1->Panels->Items[1]->Text = "";
  StatusBar1->Panels->Items[2]->Text = "";
  StatusBar1->Panels->Items[3]->Text = "";
}

// Use this code as the OnMouseMove event handler of the form:
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
  if (Shift.Contains(ssLeft)) // make sure button is down
  {
    if (Y > StartY)
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(StartY);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(Y);
    }
    else
    {
      StatusBar1->Panels->Items[0]->Text = "Top: " + IntToStr(Y);
      StatusBar1->Panels->Items[2]->Text = "Bottom: " + IntToStr(StartY);
    }
    if (X > StartX)
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(StartX);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(X);
    }
    else
    {
      StatusBar1->Panels->Items[1]->Text = "Left: " + IntToStr(X);
      StatusBar1->Panels->Items[3]->Text = "Right: " + IntToStr(StartX);
    }
  }
}
```

Delphi Examples:

```
{
The following example requires a form with a four-paneled
status bar.  (Set the Width of the status panels to 150
before running this example).  When the user presses a mouse
button, moves the mouse, and releases the mouse button, a
rectangle is drawn on the form. When the mouse button is
released, the rectangle appears on the form's canvas. Its
top-left and bottom-right corners are defined by the
location of the mouse pointer when the user pressed and
```

```
  released the mouse button.  While the user drags the mouse,
  the location of the top, left, bottom, and right sides of
  the rectangle are displayed in the status bar.
  }
  procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X,
  Y: Integer);
  begin
    StartX := X;
    StartY := Y;
  end;

  procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
  Integer);
  begin
    Form1.Canvas.Rectangle(StartX, StartY, X, Y);
    StatusBar1.Panels[0].Text := '';
    StatusBar1.Panels[1].Text := '';
    StatusBar1.Panels[2].Text := '';
    StatusBar1.Panels[3].Text := '';
  end;

  procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
  begin
    if ssLeft in Shift then { make sure button is down }
    begin
      if Y > StartY then
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(StartY);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(Y);
        end
      else
        begin
        StatusBar1.Panels[0].Text := 'Top: ' + IntToStr(Y);
        StatusBar1.Panels[2].Text := 'Bottom: ' + IntToStr(StartY);
        end;
      if X > StartX then
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(StartX);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(X);
        end
      else
        begin
        StatusBar1.Panels[1].Text := 'Left: ' + IntToStr(X);
        StatusBar1.Panels[3].Text := 'Right: ' + IntToStr(StartX);
        end;
    end;
  end;
```

See Also

OnMouseDown

OnMouseMove

Responding to a Mouse-up Action

47.1.2.6.24 TValueListEditor.OnMouseWheelDown Event

Occurs when the mouse wheel is rotated downward.

Pascal

```
property OnMouseWheelDown;
```

C++

```
__property OnMouseWheelDown;
```

Description

Write code in the OnMouseWheelDown event handler to perform actions when the mouse wheel is rotated downward. OnMouseWheelDown occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheel

OnMouseWheelUp

47.1.2.6.25 TValueListEditor.OnMouseWheelUp Event

Occurs when the mouse wheel is rotated upward.

Pascal

```
property OnMouseWheelUp;
```

C++

```
__property OnMouseWheelUp;
```

Description

Write code in the OnMouseWheelUp event handler to perform actions when the mouse wheel is rotated upward. OnMouseWheelUp occurs only if the message is not already handled in an OnMouseWheel event handler.

See Also

OnMouseWheelDown

OnMouseWheel

47.1.2.6.26 TValueListEditor.OnRowMoved Event

Occurs immediately after the position of a row changes.

Pascal

```
property OnRowMoved;
```

C++

```
__property OnRowMoved;
```

Description

Write an OnRowMoved event handler to provide special processing whenever a row in the grid changes position. Rows can be moved programmatically or by user manipulation. OnRowMoved does not occur unless the Options property includes goRowMoving.

C++ Examples:

```
/*
The following code displays the number of rows a row was
moved in the first cell of the row.  To move rows in a
TStringGrid, goRowMoving must be included in the Options property.
*/
void __fastcall TForm1::StringGrid1RowMoved(TObject *Sender, int FromIndex, int ToIndex)

{
```

```
  int delta = (FromIndex > ToIndex) ? (FromIndex-ToIndex) : (ToIndex-FromIndex);
  StringGrid1->Cells[0][ToIndex] = IntToStr(delta);
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
  StringGrid1->Cells[2][0] = "Phone Number";
  StringGrid1->Cells[2][1] = "(408)984-1234";
  StringGrid1->Cells[2][2] = "(415)434-6789";
  StringGrid1->Cells[2][3] = "(650)696-2345";
  StringGrid1->Cells[2][4] = "(831)431-2422";
  StringGrid1->Cells[2][5] = "(408)506-9876";
  StringGrid1->Cells[1][0] = "Name";
  StringGrid1->Cells[1][1] = "Clem Myers";
  StringGrid1->Cells[1][2] = "Ed Satchel";
  StringGrid1->Cells[1][3] = "Johanna Mills";
  StringGrid1->Cells[1][4] = "Stella Gables";
  StringGrid1->Cells[1][5] = "Ted Thompson";
  StringGrid1->Cells[3][0] = "POB";
  StringGrid1->Cells[3][1] = "Detroit";
  StringGrid1->Cells[3][2] = "Memphis";
  StringGrid1->Cells[3][3] = "New Orleans";
  StringGrid1->Cells[3][4] = "Lander";
  StringGrid1->Cells[3][5] = "Round Pond";
  StringGrid1->Cells[4][0] = "State";
  StringGrid1->Cells[4][1] = "Michigan";
  StringGrid1->Cells[4][2] = "Tennessee";
  StringGrid1->Cells[4][3] = "Lousiana";
  StringGrid1->Cells[4][4] = "Wyoming";
  StringGrid1->Cells[4][5] = "Maine";
  StringGrid1->Cells[5][0] = "Last Graduated";
  StringGrid1->Cells[5][1] = "Michigan State";
  StringGrid1->Cells[5][2] = "Univ-> Colorado";
  StringGrid1->Cells[5][3] = "Univ-> Washington";
  StringGrid1->Cells[5][4] = "USC";
  StringGrid1->Cells[5][5] = "Bowdoin";
}
```

Delphi Examples:

```
{
The following code displays the number of rows a row was
moved in a label.  To move rows in a TStringGrid, goRowMoving
must be included in the Options property.
}
procedure TForm1.StringGrid1RowMoved(Sender: TObject; FromIndex, ToIndex: Longint);
begin
  Label1.Caption := IntToStr(Abs(FromIndex-ToIndex));
end;
```

See Also

OnColumnMoved

Options

47.1.2.6.27 TValueListEditor.OnSelectCell Event

Occurs before a cell in the grid is selected.

Pascal

```
property OnSelectCell;
```

C++

```
__property OnSelectCell;
```

Description

Write an OnSelectCell event handler to specify whether any particular cell in the grid can be selected. The Col and Row parameters indicate the column and row indexes of the cell that is about to be selected. Set the CanSelect parameter to false to prevent the cell being selected.

C++ Examples:

```
/*
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
*/
void __fastcall TForm1::DrawGrid1DrawCell(
  TObject *Sender, int ACol, int ARow, TRect &Rect, TGridDrawState State)
{
  long index = ARow * DrawGrid1->ColCount + ACol;
  DrawGrid1->Canvas->Brush->Color = clBackground;
  DrawGrid1->Canvas->FillRect(Rect);
  if (State.Contains(gdFocused))
  {
    DrawGrid1->Canvas->DrawFocusRect(Rect);
    Label1->Caption = "Cell " + IntToStr(int(index)) + " has the focus.";
  }
  ImageList1->Draw(
    DrawGrid1->Canvas, Rect.Left, Rect.Top, index, True);
}

void __fastcall TForm1::DrawGrid1SelectCell(TObject *Sender, int ACol, int ARow,
      bool &CanSelect)
{
  CanSelect= True;
}
/*
The following code ensures that the user cannot select a cell
unless it is empty.
*/
void __fastcall TForm1::StringGrid1SelectCell(TObject *Sender, int ACol, int ARow,
          bool &CanSelect)
{
  CanSelect = (StringGrid1->Cells[ACol][ARow]== "");
}
```

Delphi Examples:

```
{
The following code uses the bitmaps in an image list
component to draw the contents of each cell in a draw grid.
It draws a focus rectangle around the cell that has focus.
goDrawFocusSelect in the DrawGrid Options parameter must be
True to set focus on a cell.  The ImageList Draw method must
be called after DrawFocusRect. The OnSelectCell event handler
must implemented to return true.
}
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
var
  index: Integer;
begin
  index := ARow * DrawGrid1.ColCount + ACol;
  DrawGrid1.Canvas.Brush.Color := clWhite;
  DrawGrid1.Canvas.FillRect(Rect);
```

```
    if (gdFocused in State) then
    begin
      DrawGrid1.Canvas.DrawFocusRect(Rect);
      Label1.Caption:= 'Cell ' + InttoStr(index) + ' has the focus.';
    end;
    ImageList1.Draw(DrawGrid1.Canvas,Rect.Left,Rect.Top,index, True);
  end;

  procedure TForm1.DrawGrid1SelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  begin
    CanSelect:= True;
  end;
  {
  The following code ensures that the user cannot select a cell
  unless it is empty.
  }
  procedure TForm1.StringGrid1SelectCell(Sender: TObject; ACol, ARow: Longint; var CanSelect:
  Boolean);
  begin
    CanSelect := (StringGrid1.Cells[ACol, ARow]='')
  end;
```

See Also

Col

Row

TabStops

47.1.2.6.28 TValueListEditor.OnSetEditText Event

Occurs when the user edits the value of a cell.

Pascal

```
property OnSetEditText;
```

C++

```
__property OnSetEditText;
```

Description

Write an OnSetEditText event handler to perform any special processing of the text edited by the user in an in-place editor. For example, use the OnSetEditText event to retrieve and store the value of a cell so that it can be displayed in an OnDrawCell event handler. OnSetEditText occurs every time the user changes the text.

The Value parameter is the new value for the cell specified by the ACol and ARow parameters.

OnSetEditText does not occur unless the Options property includes goEditing.

See Also

OnDrawCell

OnGetEditText

Options

47.1.2.6.29 TValueListEditor.OnStartDock Event

Occurs when the user begins to drag a control with a DragKind of dkDock.

Pascal

```
property OnStartDock;
```

C++

```
__property OnStartDock;
```

Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

Note: This information applies to Win32 applications only.

See Also

OnEndDock

TDragDockObjectEx

DragKind

OnDragOver

OnDockDrop

47.1.2.6.30 TValueListEditor.OnStartDrag Event

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

Pascal

```
property OnStartDrag;
```

C++

```
__property OnStartDrag;
```

Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itself.

Note: On some controls, such as TRichEdit, the underlying Windows control handles internal drag operations. For these controls, there are no OnStartDrag or OnEndDrag events for drag operations within the control.

See Also

OnEndDrag

TDragControlObjectEx

DragKind

OnDragOver

OnDragDrop

47.1.2.6.31 TValueListEditor.OnStringsChange Event

Occurs immediately after the names or values in the string list change.

Pascal

```
property OnStringsChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStringsChange;
```

Description

Write (see page 3707) an OnStringsChange event handler to respond when the strings in the list change. For example, you can use the OnStringsChange event to update your display to reflect the new values of the strings in the value list editor.

OnStringsChange occurs after the change is complete. The properties of the value list editor reflect the new values of the strings.

See Also

OnStringsChanging (see page 4881)

Strings (see page 4834)

OnSetEditText

47.1.2.6.32 TValueListEditor.OnStringsChanging Event

Occurs immediately before the names or values in the string list change.

Pascal

```
property OnStringsChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnStringsChanging;
```

Description

Write (see page 3707) an OnStringsChanging event handler to prepare for changes in the list of strings. For example, you can use the OnStringsChanging event to disable repaints until the OnStringsChange (see page 4881) event when the list has finished changing..

OnStringsChanging occurs before the change occurs. The properties of the value list editor reflect the old values of the strings.

See Also

OnStringsChange (see page 4881)

Strings (see page 4834)

OnSetEditText

47.1.2.6.33 TValueListEditor.OnTopLeftChanged Event

Occurs immediately after the TopRow property or the LeftCol property changes.

Pascal

```
property OnTopLeftChanged;
```

C++

```
__property OnTopLeftChanged;
```

Description

Use OnTopLeftChanged to perform special processing when the non-fixed cells in the grid are scrolled.

See Also

LeftCol

TopRow

47.1.2.6.34 TValueListEditor.OnValidate Event

Occurs when focus shifts away from a cell in the value list editor.

Pascal

```
property OnValidate: TOnValidateEvent;
```

C++

```
__property TOnValidateEvent OnValidate;
```

Description

Write (see page 3707) an OnValidate event handler to validate any edits the user enters in a cell before focus leaves it. OnValidate gives applications an opportunity to provide more validation than the EditMask property of the corresponding TItemProp (see page 4778) object can supply.

OnValidate only occurs if the user edited the value of the cell that is about to lose focus. The OnValidate event handler can verify the value the user supplied, and if it is not acceptable, raise an exception.

See Also

Keys (see page 4822)

Values (see page 4835)

ItemProps (see page 4821)

EditMask (see page 4781)

47.1.3 TValueListStrings Class

Class Hierarchy

Pascal

```
TValueListStrings = class(TStringList);
```

C++

```
class TValueListStrings : public TStringList;
```

File

ValEdit

Description

This is class ValEdit.TValueListStrings.

47.1.3.1 TValueListStrings Members

The following tables list the members exposed by TValueListStrings.

Protected Methods

	Name	Description
	Changed (see page 4887)	This is Changed, a member of class TValueListStrings.
	Changing (see page 4887)	This is Changing, a member of class TValueListStrings.
	FindItemProp (see page 4889)	This is FindItemProp, a member of class TValueListStrings.
	InsertItem (see page 4889)	This is InsertItem, a member of class TValueListStrings.
	Put (see page 4890)	This is Put, a member of class TValueListStrings.

Protected Properties

	Name	Description
	UpdateCount (see page 4890)	Indicates the number of calls to BeginUpdate that have not been matched by a call to EndUpdate.

Public Methods

	Name	Description
	Assign (see page 4884)	Sets the strings in the list, and possibly associated objects, from a source object.
	Clear (see page 4887)	Deletes all the strings from the list.
	Create (see page 4883)	This is Create, a member of class TValueListStrings.
	CustomSort (see page 4888)	Sorts the strings in the list in a customized order.
	Delete (see page 4888)	Removes the string specified by the `Index` parameter.
	Destroy (see page 4884)	Destroys an instance of TStringList.
	Exchange (see page 4889)	Swaps the position of two strings in the list.
	KeyIsValid (see page 4890)	This is KeyIsValid, a member of class TValueListStrings.

Public Properties

	Name	Description
	ItemProps (see page 4890)	This is ItemProps, a member of class TValueListStrings.

47.1.3.2 TValueListStrings.Create Constructor

Pascal

```
constructor Create(AEditor: TValueListEditor); reintroduce;
```

C++

```
__fastcall TValueListStrings(TValueListEditor AEditor);
```

Description

This is Create, a member of class TValueListStrings.

47.1.3.3 TValueListStrings.Destroy Destructor

Destroys an instance of TStringList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TValueListStrings();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TStringList reference is not nil and only then calls Destroy.

Destroy frees the memory allocated to hold the list of strings and object references before calling the inherited destructor.

See Also

System::TObject::Free (see page 3372)

47.1.3.4 TValueListStrings Methods

The methods of the TValueListStrings class are listed here.

Protected Methods

	Name	Description
	Changed (see page 4887)	This is Changed, a member of class TValueListStrings.
	Changing (see page 4887)	This is Changing, a member of class TValueListStrings.
	FindItemProp (see page 4889)	This is FindItemProp, a member of class TValueListStrings.
	InsertItem (see page 4889)	This is InsertItem, a member of class TValueListStrings.
	Put (see page 4890)	This is Put, a member of class TValueListStrings.

Public Methods

	Name	Description
	Assign (see page 4884)	Sets the strings in the list, and possibly associated objects, from a source object.
	Clear (see page 4887)	Deletes all the strings from the list.
	CustomSort (see page 4888)	Sorts the strings in the list in a customized order.
	Delete (see page 4888)	Removes the string specified by the `Index` parameter.
	Exchange (see page 4889)	Swaps the position of two strings in the list.
	KeyIsValid (see page 4890)	This is KeyIsValid, a member of class TValueListStrings.

47.1.3.4.1 TValueListStrings.Assign Method

Sets the strings in the list, and possibly associated objects, from a source object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to set the value of the TStrings object from another object. If Source is of type TStrings, the list is set to the list of the

source TStrings object, and if associated objects are supported, any associated objects are copied from the Source as well.

If Source is not of type TStrings, the inherited Assign will set the value of the list from any object that supports TStrings in its AssignTo method.

C++ Examples:

```
/*
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
*/
void __fastcall TForm1::Add_a_fileClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    int index = TabControl1->Tabs->Add(OpenDialog1->FileName);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[index]);
    TabControl1Change(Sender);
  }
}

void __fastcall TForm1::Assign_filesClick(TObject *Sender)
{
  OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist << ofHideReadOnly;
  if (OpenDialog1->Execute())
  {
    TabControl1->Tabs->Assign(OpenDialog1->Files);
    Memo1->Lines->LoadFromFile(TabControl1->Tabs->Strings[TabControl1->TabIndex]);
  }
}
/*
Place the following code in the tab control's OnChange event
handler:
*/
void __fastcall TForm1::TabControl1Change(TObject *Sender)
{
    Memo1->Lines->LoadFromFile(
      TabControl1->Tabs->Strings[TabControl1->TabIndex]);
}
/*
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
*/

#include <memory>       //for STL auto_ptr class

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  std::auto_ptr<TStrings> StringList(new TStringList());
  StringList->Add("This example uses A string List. ");
  StringList->Add("It is the easiest way to add strings");
  StringList->Add("to a combobox's list of strings. ");
  StringList->Add("Always remember TStrings is abstract,");
  StringList->Add("So create a TStringList instead.");
  ComboBox1->Width = 210;
  ComboBox1->Items->Assign(StringList.get());
  ComboBox1->ItemIndex = 0;
```

```
}
```

Delphi Examples:

```
{
This example uses a tab control to display the contents of
several files. To run the example, place a tab control on a
form and add a memo control that fits into its client area.
Be sure to leave enough room for the tabs when they appear.
Then add an OpenDialog and a button to the form.  The
"Add a file" button adds a single file to the tabcontrol.
The "Assign files" button removes previous files and can be
used to assign multiple files.  To assign multiple files,
use CNTL Select or SHIFT Select to select files in the
OpenDialog.
}
procedure TForm1.Add_a_fileClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    index:= TabControl1.Tabs.Add(OpenDialog1.FileName);
    Memo1.Lines.LoadFromFile(TabControl1.Tabs[index]);
    TabControl1Change(Sender);
  end;
end;

procedure TForm1.Assign_filesClick(Sender: TObject);
var index : integer;
begin
  OpenDialog1.Options :=
    [ofAllowMultiSelect, ofFileMustExist, ofHideReadOnly];
  if OpenDialog1.Execute then
  begin
    TabControl1.Tabs.Assign(OpenDialog1.Files);
    Memo1.Lines.LoadFromFile(
      TabControl1.Tabs[TabControl1.TabIndex]);
  end;
end;

{
Place the following code in the tab control's OnChange event
handler:
}
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  with TabControl1 do
    Memo1.Lines.LoadFromFile(Tabs[TabIndex]);
end;
{
This example creates a stringlist, adds elements to it and
assigns the strings to items of a combo box.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  StringList: TStringList;
begin
  StringList := TStringList.Create;
  try
    with StringList do begin
      Add('This example uses A string List.');
      Add('It is the easiest way to add strings');
      Add('to a combobox''s list of strings.');
      Add('Always remember TStrings.Create method');
      Add('is abstract; So use TStringList.Create');
```

```
        Add('method instead.');
      end;
      with ComboBox1 do begin
        Width := 210;
        Items.Assign(StringList);
        ItemIndex := 0;
      end;
    finally
      StringList.free;
    end;
  end;
```

See Also

AddStrings

Objects

Strings

AssignTo

47.1.3.4.2 TValueListStrings.Changed Method

Pascal

```
procedure Changed; override;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TValueListStrings.

47.1.3.4.3 TValueListStrings.Changing Method

Pascal

```
procedure Changing; override;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TValueListStrings.

47.1.3.4.4 TValueListStrings.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete

Objects

Strings

47.1.3.4.5 TValueListStrings.CustomSort Method

Sorts the strings in the list in a customized order.

Pascal

```
procedure CustomSort(Compare: TStringListSortCompare); override;
```

C++

```
virtual __fastcall CustomSort(TStringListSortCompare Compare);
```

Description

Use CustomSort to sort the strings in the list, where the sort order is defined by the Compare parameter.

Supply a value for the Compare function that compares two strings in the string list. The `List` parameter provides access to the string list, while the Index1 and Index2 parameters identify the strings to be compared.

Do not pass nil (Delphi) or NULL (C++) as the value of the Compare parameter.

Note: You must explicitly call the CustomSort method. Setting the Sorted property only sorts strings using ANSI (Windows) or UTF-8 (Linux) order, as implemented in the Sort method.

See Also

Sort

Strings

Sorted

47.1.3.4.6 TValueListStrings.Delete Method

Removes the string specified by the `Index` parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add

AddObject

Clear

IndexOf

InsertObject

Objects

Strings

47.1.3.4.7 TValueListStrings.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the index of the object as well.

Warning: Do not call Exchange on a sorted list except to swap two identical strings with different associated objects. Exchange does not check whether the list is sorted, and can destroy the sort order of a sorted list.

See Also

Move

Objects

Sorted

Strings

47.1.3.4.8 TValueListStrings.FindItemProp Method

Pascal

```
function FindItemProp(const KeyOrIndex: Variant; Create: Boolean = False): TItemProp;
```

C++

```
__fastcall TItemProp FindItemProp(const Variant KeyOrIndex, Boolean Create = False);
```

Description

This is FindItemProp, a member of class TValueListStrings.

47.1.3.4.9 TValueListStrings.InsertItem Method

Pascal

```
procedure InsertItem(Index: Integer; const S: string; AObject: TObject); override;
```

C++

```
virtual __fastcall InsertItem(int Index, const UnicodeString S, TObject * AObject);
```

Description

This is InsertItem, a member of class TValueListStrings.

47.1.3.4.10 TValueListStrings.KeyIsValid Method

Pascal

```
function KeyIsValid(const Key: string; RaiseError: Boolean = True): Boolean;
```

C++

```
__fastcall Boolean KeyIsValid(const UnicodeString Key, Boolean RaiseError = True);
```

Description

This is KeyIsValid, a member of class TValueListStrings.

47.1.3.4.11 TValueListStrings.Put Method

Pascal

```
procedure Put(Index: Integer; const S: String); override;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString S);
```

Description

This is Put, a member of class TValueListStrings.

47.1.3.5 TValueListStrings Properties

The properties of the TValueListStrings class are listed here.

Protected Properties

	Name	Description
	UpdateCount (see page 4890)	Indicates the number of calls to BeginUpdate that have not been matched by a call to EndUpdate.

Public Properties

	Name	Description
	ItemProps (see page 4890)	This is ItemProps, a member of class TValueListStrings.

47.1.3.5.1 TValueListStrings.ItemProps Property

Pascal

```
property ItemProps [const KeyOrIndex: Variant]: TItemProp;
```

C++

```
__property TItemProp ItemProps[Variant const KeyOrIndex];
```

Description

This is ItemProps, a member of class TValueListStrings.

47.1.3.5.2 TValueListStrings.UpdateCount Property

Indicates the number of calls to BeginUpdate that have not been matched by a call to EndUpdate.

Pascal

```
property UpdateCount;
```

C++

```
__property UpdateCount;
```

Description

TStrings uses UpdateCount to keep track of calls to the BeginUpdate and EndUpdate methods. Every time a call is made to BeginUpdate, TStrings increments the value of UpdateCount. Every call to EndUpdate causes TStrings to decrement UpdateCount.

When UpdateCount changes from 0 to 1, TStrings calls the SetUpdateState method with a parameter of true. When UpdateCount changes from 1 to 0, TStrings calls the SetUpdateState method with a parameter of false. This allows descendant classes to perform optimizations when handling multiple updates.

See Also

BeginUpdate

EndUpdate

SetUpdateState

47.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TDisplayOption (see page 4892)	TDisplayOption and TDisplayOptions (see page 4894) control the appearance of a value list editor.
	TKeyOption (see page 4893)	TKeyOption and TKeyOptions (see page 4895) indicate the response of a value list editor to certain keystrokes.

47.2.1 ValEdit.TDisplayOption Enumeration

TDisplayOption and TDisplayOptions (see page 4894) control the appearance of a value list editor.

Pascal

```
TDisplayOption = (
  doColumnTitles,
  doAutoColResize,
  doKeyColFixed
);
```

C++

```
enum TDisplayOption {
  doColumnTitles,
  doAutoColResize,
  doKeyColFixed
};
```

File

ValEdit

Description

TDisplayOption is a set of TDisplayOptions (see page 4894) values. The possible values of TDisplayOptions (see page 4894) are

Option	Meaning
doColumnTitles	The first row of the value list editor is a fixed (nonscrolling) row that displays the captions specified by the TitleCaptions property.
doAutoColResize	When the editor is resized, the column widths automatically adjust so that they completely fill the new width of the editor. If DisplayOptions does not include doKeyColFixed, each column has a width equal to half the width of the editor. Otherwise, the value column takes up any space left over after the fixed (key) column.
doKeyColFixed	The first (key) column of the value list editor is fixed in width. When the value list editor is resized, only the second (value) column resizes. This option is only used when DisplayOptions includes doAutoColResize.

47.2.2 ValEdit.TKeyOption Enumeration

TKeyOption and TKeyOptions (see page 4895) indicate the response of a value list editor to certain keystrokes.

Pascal

```
TKeyOption = (
  keyEdit,
  keyAdd,
  keyDelete,
  keyUnique
);
```

C++

```
enum TKeyOption {
  keyEdit,
  keyAdd,
  keyDelete,
  keyUnique
};
```

File

ValEdit

Description

TKeyOption is a set of TKeyOptions (see page 4895) values. The possible values of TKeyOptions (see page 4895) are

Option	Meaning
keyEdit	The user can select and edit values in the first (key) column. That is, the user can specify the name in a name/value pair.
keyAdd	The user can add new name/value pairs. When included, the user can create new entries by pressing the Insert (see page 3601) key or using the down arrow key when positioned on the last name/value pair. keyAdd requires that keyEdit is also included.
keyDelete	The user can delete the selected name/value pair by pressing the Delete (see page 3552) key.
keyUnique	The user can't create name/value pairs where the name portion is the same as the name in another name/value pair. Attempting to add a duplicate name causes the value list editor to raise an exception. keyUnique requires that keyEdit is also included.

47.3 Types

The following table lists types in this documentation.

Types

Name	Description
TDisplayOptions (see page 4894)	TDisplayOptions and TDisplayOption (see page 4892) control the appearance of a value list editor.
TGetPickListEvent (see page 4895)	TGetPickListEvent is the type for event handlers that supply pick list values to a value list editor.
TItemProps (see page 4895)	This is type ValEdit.TItemProps.
TKeyOptions (see page 4895)	TKeyOptions and TKeyOption (see page 4893) indicate the response of a value list editor to certain keystrokes.
TOnValidateEvent (see page 4896)	TOnValidateEvent is the type for event handlers that validate the values a user types into a value list editor.

47.3.1 ValEdit.TDisplayOptions Type

TDisplayOptions and TDisplayOption (see page 4892) control the appearance of a value list editor.

Pascal

```
TDisplayOptions = set of TDisplayOption;
```

C++

```
TDisplayOption TDisplayOptions;
```

File

ValEdit

Description

TDisplayOptions is a set of TDisplayOption (see page 4892) values. The following table lists the possible values of TDisplayOption (see page 4892).

Option	Meaning
doColumnTitles	The first row of the value list editor is a fixed (nonscrolling) row that displays the captions specified by the TitleCaptions property.
doAutoColResize	When the editor is resized, the column widths automatically adjust so that they completely fill the new width of the editor. If DisplayOptions does not include doKeyColFixed, each column has a width equal to half the width of the editor. Otherwise, the value column takes up any space left over after the fixed (key) column.
doKeyColFixed	The first (key) column of the value list editor is fixed in width. When the value list editor is resized, only the second (value) column resizes. This option is only used when DisplayOptions includes doAutoColResize.

See Also

TDisplayOption (see page 4892)

TValueListEditor (see page 4784)

47.3.2 ValEdit.TGetPickListEvent Type

TGetPickListEvent is the type for event handlers that supply pick list values to a value list editor.

Pascal

```
TGetPickListEvent = procedure (Sender: TObject; const KeyName: string; Values: TStrings) of object;
```

C++

```
(Sender: TObject; const KeyName: string; Values: TStrings) ( TGetPickListEvent)();
```

File

ValEdit

Description

`Sender` is the value list editor that uses the drop-down list of values. `KeyName` is the name of a string that appears in the first column of the current row. `Values` is a string list that you can fill with candidates for the second (value) column of the value list editor.

See Also

OnGetPickList (see page 4860)

Procedural types

Creating events

47.3.3 ValEdit.TItemProps Type

Pascal

```
TItemProps = array of TItemProp;
```

C++

```
array of TItemProp TItemProps;
```

File

ValEdit

Description

This is type ValEdit.TItemProps.

47.3.4 ValEdit.TKeyOptions Type

TKeyOptions and TKeyOption (see page 4893) indicate the response of a value list editor to certain keystrokes.

Pascal

```
TKeyOptions = set of TKeyOption;
```

C++

```
TKeyOption TKeyOptions;
```

File

ValEdit

Description

TKeyOptions is a set of TKeyOption (see page 4893) values. The possible values of TKeyOption (see page 4893) are listed in the following table.

Option	Meaning
keyEdit	The user can select and edit values in the first (key) column. That is, the user can specify the name in a name/value pair.
keyAdd	The user can add new name/value pairs. When included, the user can create new entries by pressing the Insert (see page 3601) key or using the down arrow key when positioned on the last name/value pair. keyAdd requires that keyEdit is also included.
keyDelete	The user can delete the selected name/value pair by pressing the Delete (see page 3552) key.
keyUnique	The user cannot create name/value pairs where the name portion is the same as the name in another name/value pair. Attempting to add a duplicate name causes the value list editor to raise an exception. keyUnique requires that keyEdit is also included.

See Also

TKeyOption (see page 4893)

TValueListEditor (see page 4784)

47.3.5 ValEdit.TOnValidateEvent Type

TOnValidateEvent is the type for event handlers that validate the values a user types into a value list editor.

Pascal

```
TOnValidateEvent = procedure (Sender: TObject; ACol, ARow: Longint; const KeyName, KeyValue:
string) of object;
```

C++

```
(Sender: TObject; ACol, ARow: Longint; const KeyName, KeyValue: string) ( TOnValidateEvent)();
```

File

ValEdit

Description

Sender is the value list editor that contains the edited cell.

ACol and ARow identify the cell by its column and row numbers, respectively. The first row or column has an index of 0, the second has an index of 1, and so on.

KeyName is the current value of in the key column for ARow. This value includes any edits that have not yet been committed.

KeyValue is the current value of the value column for ARow. This value includes any edits that have not yet been committed.

See Also

Procedural types

Creating events

48 VarCmplx Namespace

This is namespace VarCmplx.

Topics

Name	Description
Functions (see page 4901)	The following table lists functions in this documentation.
Variables (see page 4944)	The following table lists variables in this documentation.

Functions

	Name	Description
	VarAsComplex (see page 4902)	Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a complex number.
	VarComplex (see page 4903)	Returns the Variant (see page 3421) type code for custom Variants (see page 4973) that represent complex numbers.
	VarComplexAbs (see page 4903)	Returns the absolute value of a complex number.
	VarComplexAbsSqr (see page 4904)	Returns the square of the absolute value of a complex number.
	VarComplexAngle (see page 4905)	Returns the angle made by a complex number.
	VarComplexArcCos (see page 4905)	Returns the inverse cosine of a complex number.
	VarComplexArcCosH (see page 4906)	Returns the hyperbolic inverse cosine of a complex number.
	VarComplexArcCot (see page 4907)	Returns the inverse cotangent of a complex number.
	VarComplexArcCotH (see page 4908)	Returns the hyperbolic inverse cotangent of a complex number.
	VarComplexArcCsc (see page 4909)	Returns the inverse cosecant of a complex number.
	VarComplexArcCscH (see page 4911)	Returns the hyperbolic inverse cosecant of a complex number.
	VarComplexArcSec (see page 4912)	Returns the inverse secant of a complex number.
	VarComplexArcSecH (see page 4913)	Returns the hyperbolic inverse secant of a complex number.
	VarComplexArcSin (see page 4914)	Returns the inverse sine of a complex number.
	VarComplexArcSinH (see page 4915)	Returns the hyperbolic inverse sine of a complex number.
	VarComplexArcTan (see page 4916)	Returns the inverse tangent of a complex number.
	VarComplexArcTanH (see page 4917)	Returns the hyperbolic inverse tangent of a complex number.
	VarComplexConjugate (see page 4918)	Returns the conjugate of a complex number.
	VarComplexCos (see page 4919)	Returns the cosine of a complex number.
	VarComplexCosH (see page 4920)	Returns the hyperbolic cosine of a complex number.
	VarComplexCot (see page 4921)	Returns the cotangent of a complex number.
	VarComplexCotH (see page 4922)	Returns the hyperbolic cotangent of a complex number.
	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.
	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.

	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.
	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.
	VarComplexCsc (see page 4924)	Returns the cosecant of a complex number.
	VarComplexCscH (see page 4925)	Returns the hyperbolic cosecant of a complex number.
	VarComplexExp (see page 4926)	Returns the exponential of a complex number.
	VarComplexFromPolar (see page 4927)	Converts a set of polar coordinates to a complex number.
	VarComplexInverse (see page 4927)	Returns the inverse of a complex number.
	VarComplexLn (see page 4928)	Returns the natural log of a complex number.
	VarComplexLog10 (see page 4929)	Returns the log base 10 of a complex number.
	VarComplexLog2 (see page 4929)	Returns the log base 2 of a complex number.
	VarComplexLogN (see page 4930)	Returns the log base N of a complex number.
	VarComplexPower (see page 4930)	Returns the value of a complex number raised to a specified power.
	VarComplexSec (see page 4931)	Returns the secant of a complex number.
	VarComplexSecH (see page 4932)	Returns the hyperbolic secant of a complex number.
	VarComplexSign (see page 4933)	This is function VarCmplx.VarComplexSign.
	VarComplexSimplify (see page 4934)	Reduces a Variant (see page 3421) that represents a complex number to one that represents a real value, if possible.
	VarComplexSin (see page 4934)	Returns the sine of a complex number.
	VarComplexSinH (see page 4935)	Returns the hyperbolic sine of a complex number.
	VarComplexSqr (see page 4936)	Returns the square of a complex number.
	VarComplexSqrt (see page 4937)	Returns the square root of a complex number.
	VarComplexTan (see page 4938)	Returns the tangent of a complex number.
	VarComplexTanH (see page 4939)	Returns the hyperbolic tangent of a complex number.
	VarComplexTimesImaginary (see page 4940)	Returns the value of a complex number multiplied by an imaginary number.
	VarComplexTimesNegI (see page 4940)	Returns the value of a complex number multiplied by –i.
	VarComplexTimesPosI (see page 4941)	Returns the value of a complex number multiplied by i.
	VarComplexTimesReal (see page 4941)	Returns the value of a complex number multiplied by a real number.
	VarComplexToPolar (see page 4942)	Computes the polar coordinates that correspond to a custom Variant (see page 3421) that represents a complex number.
	VarIsComplex (see page 4942)	Indicates whether a Variant (see page 3421) stores its data internally as a complex number.

Variables

Name	Description
ComplexNumberDefuzzAtZero (see page 4944)	This is variable VarCmplx.ComplexNumberDefuzzAtZero.
ComplexNumberSymbol (see page 4944)	Indicates the symbol used to represent the imaginary part of a complex number.
ComplexNumberSymbolBeforeImaginary (see page 4945)	Indicates the placement of the symbol that represents the imaginary part of a complex number.

48.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsComplex (see page 4902)	Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a complex number.
	VarComplex (see page 4903)	Returns the Variant (see page 3421) type code for custom Variants (see page 4973) that represent complex numbers.
	VarComplexAbs (see page 4903)	Returns the absolute value of a complex number.
	VarComplexAbsSqr (see page 4904)	Returns the square of the absolute value of a complex number.
	VarComplexAngle (see page 4905)	Returns the angle made by a complex number.
	VarComplexArcCos (see page 4905)	Returns the inverse cosine of a complex number.
	VarComplexArcCosH (see page 4906)	Returns the hyperbolic inverse cosine of a complex number.
	VarComplexArcCot (see page 4907)	Returns the inverse cotangent of a complex number.
	VarComplexArcCotH (see page 4908)	Returns the hyperbolic inverse cotangent of a complex number.
	VarComplexArcCsc (see page 4909)	Returns the inverse cosecant of a complex number.
	VarComplexArcCscH (see page 4911)	Returns the hyperbolic inverse cosecant of a complex number.
	VarComplexArcSec (see page 4912)	Returns the inverse secant of a complex number.
	VarComplexArcSecH (see page 4913)	Returns the hyperbolic inverse secant of a complex number.
	VarComplexArcSin (see page 4914)	Returns the inverse sine of a complex number.
	VarComplexArcSinH (see page 4915)	Returns the hyperbolic inverse sine of a complex number.
	VarComplexArcTan (see page 4916)	Returns the inverse tangent of a complex number.
	VarComplexArcTanH (see page 4917)	Returns the hyperbolic inverse tangent of a complex number.
	VarComplexConjugate (see page 4918)	Returns the conjugate of a complex number.
	VarComplexCos (see page 4919)	Returns the cosine of a complex number.
	VarComplexCosH (see page 4920)	Returns the hyperbolic cosine of a complex number.
	VarComplexCot (see page 4921)	Returns the cotangent of a complex number.
	VarComplexCotH (see page 4922)	Returns the hyperbolic cotangent of a complex number.
	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.
	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.
	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.
	VarComplexCreate (see page 4923)	Returns a custom Variant (see page 3421) that represents a complex number.
	VarComplexCsc (see page 4924)	Returns the cosecant of a complex number.
	VarComplexCscH (see page 4925)	Returns the hyperbolic cosecant of a complex number.
	VarComplexExp (see page 4926)	Returns the exponential of a complex number.
	VarComplexFromPolar (see page 4927)	Converts a set of polar coordinates to a complex number.
	VarComplexInverse (see page 4927)	Returns the inverse of a complex number.
	VarComplexLn (see page 4928)	Returns the natural log of a complex number.
	VarComplexLog10 (see page 4929)	Returns the log base 10 of a complex number.
	VarComplexLog2 (see page 4929)	Returns the log base 2 of a complex number.
	VarComplexLogN (see page 4930)	Returns the log base N of a complex number.
	VarComplexPower (see page 4930)	Returns the value of a complex number raised to a specified power.
	VarComplexSec (see page 4931)	Returns the secant of a complex number.
	VarComplexSecH (see page 4932)	Returns the hyperbolic secant of a complex number.
	VarComplexSign (see page 4933)	This is function VarCmplx.VarComplexSign.
	VarComplexSimplify (see page 4934)	Reduces a Variant (see page 3421) that represents a complex number to one that represents a real value, if possible.
	VarComplexSin (see page 4934)	Returns the sine of a complex number.
	VarComplexSinH (see page 4935)	Returns the hyperbolic sine of a complex number.

	VarComplexSqr (see page 4936)	Returns the square of a complex number.
	VarComplexSqrt (see page 4937)	Returns the square root of a complex number.
	VarComplexTan (see page 4938)	Returns the tangent of a complex number.
	VarComplexTanH (see page 4939)	Returns the hyperbolic tangent of a complex number.
	VarComplexTimesImaginary (see page 4940)	Returns the value of a complex number multiplied by an imaginary number.
	VarComplexTimesNegI (see page 4940)	Returns the value of a complex number multiplied by –i.
	VarComplexTimesPosI (see page 4941)	Returns the value of a complex number multiplied by i.
	VarComplexTimesReal (see page 4941)	Returns the value of a complex number multiplied by a real number.
	VarComplexToPolar (see page 4942)	Computes the polar coordinates that correspond to a custom Variant (see page 3421) that represents a complex number.
	VarIsComplex (see page 4942)	Indicates whether a Variant (see page 3421) stores its data internally as a complex number.

48.1.1 VarCmplx.VarAsComplex Function

Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a complex number.

Pascal

```
function VarAsComplex(const AValue: Variant): Variant;
```

C++

```
Variant VarAsComplex(const Variant AValue);
```

File

VarCmplx

Description

VarAsComplex converts the Variant (see page 3421) specified by AValue to a custom Variant (see page 3421) representing the corresponding complex number. It returns the custom Variant (see page 3421).

If VarAsComplex can't perform the conversion, it raises an EInvalidCast exception.

Complex Variants (see page 4973) are custom Variants (see page 4973) that represent complex numbers. The Variant (see page 3421) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3323) values, and boolean values. In addition, the VarCmplx (see page 4899) unit implements a number of global functions for operating on complex Variants (see page 4973).

See Also

EInvalidCast (see page 3935)

VarIsComplex (see page 4942)

VarComplexCreate (see page 4923)

VarComplexSimplify (see page 4934)

VarAsType (see page 5073)

48.1.2 VarCmplx.VarComplex Function

Returns the Variant (see page 3421) type code for custom Variants (see page 4973) that represent complex numbers.

Pascal

```
function VarComplex: TVarType;
```

C++

```
TVarType VarComplex();
```

File

VarCmplx

Description

Use VarComplex to obtain the variant type code for custom Variants (see page 4973) that represent complex numbers. The Variant (see page 3421) type codes for custom Variants (see page 4973) are dynamically allocated when the application loads the unit that implements them. VarComplex allows the application to identify the Variant (see page 3421) type code that is currently used for custom Variants (see page 4973) that represent complex numbers. This type code, however, may not match the type code used by another application that uses a different set of custom Variant (see page 3421) types.

Complex Variants (see page 4973) are custom Variants (see page 4973) that represent complex numbers. The Variant (see page 3421) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3323) values, and boolean values. In addition, the VarCmplx (see page 4899) unit implements a number of global functions for operating on complex Variants (see page 4973).

See Also

VarType (see page 5090)

48.1.3 VarCmplx.VarComplexAbs Function

Returns the absolute value of a complex number.

Pascal

```
function VarComplexAbs(const AValue: Variant): Double;
```

C++

```
Double VarComplexAbs(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAbs to take the absolute value (length) of a complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexAbs returns the absolute value of AValue.

See Also

VarComplexAbsSqr (see page 4904)

VarComplexAngle (see page 4905)

VarComplexConjugate (see page 4918)

VarComplexInverse (see page 4927)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

48.1.4 VarCmplx.VarComplexAbsSqr Function

Returns the square of the absolute value of a complex number.

Pascal

```
function VarComplexAbsSqr(const AValue: Variant): Double;
```

C++

```
Double VarComplexAbsSqr(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAbsSqr to obtain the square of a complex number's absolute value.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexAbs (see page 4903) returns the absolute value of AValue, squared.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexConjugate (see page 4918)

VarComplexInverse (see page 4927)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexLn (see page 4928)

VarComplexExp (see page 4926)

VarComplexSqr (see page 4936)

VarComplexSqrt (see page 4937)

VarComplexPower (see page 4930)

48.1.5 VarCmplx.VarComplexAngle Function

Returns the angle made by a complex number.

Pascal

```
function VarComplexAngle(const AValue: Variant): Double;
```

C++

```
Double VarComplexAngle(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexAngle to obtain the angle of a complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexAbs (see page 4903) returns the angle that corresponds to AValue.

See Also

VarComplexToPolar (see page 4942)

VarComplexAbs (see page 4903)

48.1.6 VarCmplx.VarComplexArcCos Function

Returns the inverse cosine of a complex number.

Pascal

```
function VarComplexArcCos(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCos(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCos computes the inverse cosine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcCos returns a Variant (see page 3421) whose value is the inverse cosine of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)
VarComplexArcSec (see page 4912)
VarComplexArcSecH (see page 4913)
VarComplexArcSin (see page 4914)
VarComplexArcSinH (see page 4915)
VarComplexArcTan (see page 4916)
VarComplexArcTanH (see page 4917)
VarComplexCos (see page 4919)
VarComplexCosH (see page 4920)
VarComplexCot (see page 4921)
VarComplexCotH (see page 4922)
VarComplexCsc (see page 4924)
VarComplexCscH (see page 4925)
VarComplexSec (see page 4931)
VarComplexSecH (see page 4932)
VarComplexSin (see page 4934)
VarComplexSinH (see page 4935)
VarComplexTan (see page 4938)
VarComplexTanH (see page 4939)

48.1.7 VarCmplx.VarComplexArcCosH Function

Returns the hyperbolic inverse cosine of a complex number.

Pascal

```
function VarComplexArcCosH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCosH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCosH computes the hyperbolic inverse cosine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcCosH returns a Variant (see page 3421) whose value is the hyperbolic inverse cosine of AValue.

See Also

VarComplexArcCos (see page 4905)

VarComplexArcCot (see page 4907)
VarComplexArcCotH (see page 4908)
VarComplexArcCsc (see page 4909)
VarComplexArcCscH (see page 4911)
VarComplexArcSec (see page 4912)
VarComplexArcSecH (see page 4913)
VarComplexArcSin (see page 4914)
VarComplexArcSinH (see page 4915)
VarComplexArcTan (see page 4916)
VarComplexArcTanH (see page 4917)
VarComplexCos (see page 4919)
VarComplexCosH (see page 4920)
VarComplexCot (see page 4921)
VarComplexCotH (see page 4922)
VarComplexCsc (see page 4924)
VarComplexCscH (see page 4925)
VarComplexSec (see page 4931)
VarComplexSecH (see page 4932)
VarComplexSin (see page 4934)
VarComplexSinH (see page 4935)
VarComplexTan (see page 4938)
VarComplexTanH (see page 4939)

48.1.8 VarCmplx.VarComplexArcCot Function

Returns the inverse cotangent of a complex number.

Pascal

```
function VarComplexArcCot(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCot(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCot computes the inverse cotangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcCot returns a Variant (see page 3421) whose value is the inverse cotangent of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.9 VarCmplx.VarComplexArcCotH Function

Returns the hyperbolic inverse cotangent of a complex number.

Pascal

```
function VarComplexArcCotH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCotH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCot (see page 4907) computes the hyperbolic inverse cotangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcCot (see page 4907) returns a Variant (see page 3421) whose value is the hyperbolic inverse cotangent of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCos (see page 4905)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.10 VarCmplx.VarComplexArcCsc Function

Returns the inverse cosecant of a complex number.

Pascal

```
function VarComplexArcCsc(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCsc(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCsc computes the inverse cosecant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcCsc returns a Variant (see page 3421) whose value is the inverse cosecant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCot (see page 4907)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.11 VarCmplx.VarComplexArcCscH Function

Returns the hyperbolic inverse cosecant of a complex number.

Pascal

```
function VarComplexArcCscH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcCscH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcCsc (see page 4909) computes the hyperbolic inverse cosecant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcCsc (see page 4909) returns a Variant (see page 3421) whose value is the hyperbolic inverse cosecant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCsc (see page 4909)

VarComplexArcCos (see page 4905)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.12 VarCmplx.VarComplexArcSec Function

Returns the inverse secant of a complex number.

Pascal

```
function VarComplexArcSec(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSec(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSec computes the inverse secant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcSec returns a Variant (see page 3421) whose value is the inverse secant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCot (see page 4907)

VarComplexArcCscH (see page 4911)

VarComplexArcCsc (see page 4909)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.13 VarCmplx.VarComplexArcSecH Function

Returns the hyperbolic inverse secant of a complex number.

Pascal

```
function VarComplexArcSecH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSecH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSec (see page 4912) computes the hyperbolic inverse secant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcSec (see page 4912) returns a Variant (see page 3421) whose value is the hyperbolic inverse secant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCot (see page 4907)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.14 VarCmplx.VarComplexArcSin Function

Returns the inverse sine of a complex number.

Pascal

```
function VarComplexArcSin(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSin(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSin computes the inverse sine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcSin returns a Variant (see page 3421) whose value is the inverse sine of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.15 VarCmplx.VarComplexArcSinH Function

Returns the hyperbolic inverse sine of a complex number.

Pascal

```
function VarComplexArcSinH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcSinH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcSinH computes the hyperbolic inverse sine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcSinH returns a Variant (see page 3421) whose value is the hyperbolic inverse sine of AValue.

See Also

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcCosH (see page 4906)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.16 VarCmplx.VarComplexArcTan Function

Returns the inverse tangent of a complex number.

Pascal

```
function VarComplexArcTan(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcTan(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcTan computes the inverse tangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcTan returns a Variant (see page 3421) whose value is the inverse tangent of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcTanH (see page 4917)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.17 VarCmplx.VarComplexArcTanH Function

Returns the hyperbolic inverse tangent of a complex number.

Pascal

```
function VarComplexArcTanH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexArcTanH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexArcTanH computes the hyperbolic inverse tangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexArcTanH returns a Variant (see page 3421) whose value is the hyperbolic inverse tangent of AValue.

See Also

VarComplexArcTan (see page 4916)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcCosH (see page 4906)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.18 VarCmplx.VarComplexConjugate Function

Returns the conjugate of a complex number.

Pascal

```
function VarComplexConjugate(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexConjugate(const Variant AValue);
```

File

VarCmplx

Description

VarComplexConjugate returns the conjugate of a complex number. The conjugate is the value that has the same real part and the inverse of the imaginary part.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexConjugate returns a Variant (see page 3421) whose value is the conjugate of AValue.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexAbsSqr (see page 4904)

VarComplexInverse (see page 4927)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexLn (see page 4928)

VarComplexExp (see page 4926)

VarComplexSqr (see page 4936)

VarComplexSqrt (see page 4937)

VarComplexPower (see page 4930)

48.1.19 VarCmplx.VarComplexCos Function

Returns the cosine of a complex number.

Pascal

```
function VarComplexCos(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCos(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCos computes the cosine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexCos returns a Variant (see page 3421) whose value is the cosine of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexArcCos (see page 4905)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.20 VarCmplx.VarComplexCosH Function

Returns the hyperbolic cosine of a complex number.

Pascal

```
function VarComplexCosH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCosH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCosH computes the hyperbolic cosine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexCosH returns a Variant (see page 3421) whose value is the hyperbolic cosine of AValue.

See Also

VarComplexArcCos (see page 4905)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.21 VarCmplx.VarComplexCot Function

Returns the cotangent of a complex number.

Pascal

```
function VarComplexCot(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCot(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCot computes the cotangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexCot returns a Variant (see page 3421) whose value is the cotangent of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.22 VarCmplx.VarComplexCotH Function

Returns the hyperbolic cotangent of a complex number.

Pascal

```
function VarComplexCotH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCotH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCot (see page 4921) computes the hyperbolic cotangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexCot (see page 4921) returns a Variant (see page 3421) whose value is the hyperbolic cotangent of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCos (see page 4905)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.23 VarCmplx.VarComplexCreate Function

Returns a custom Variant (see page 3421) that represents a complex number.

Pascal

```
function VarComplexCreate: Variant; overload;
function VarComplexCreate(const AReal: Double): Variant; overload;
function VarComplexCreate(const AReal: Double; const AImaginary: Double): Variant; overload;
function VarComplexCreate(const AText: string): Variant; overload;
```

C++

```
Variant VarComplexCreate();
Variant VarComplexCreate(const Double AReal);
Variant VarComplexCreate(const Double AReal, const Double AImaginary);
Variant VarComplexCreate(const UnicodeString AText);
```

File

VarCmplx

Description

Call VarComplexCreate to create a Variant (see page 3421) whose value is a complex number.

AReal is the real part of the complex number.

AImaginary is the imaginary part of the complex number.

Using the first three versions of the constructor, omitting AReal or AImaginary results in a real or imaginary part of 0.

Using the last syntax, AText is parsed using the ComplexNumberSymbol (see page 4944) and

ComplexNumberSymbolBeforeImaginary (see page 4945) global variables and, if the string represents a complex value, that becomes the value of the new Variant (see page 3421). If the string can't be parsed into a complex number, VarComplexCreate raises an EConvertError exception.

Complex Variants (see page 4973) are custom Variants (see page 4973) that represent complex numbers. The Variant (see page 3421) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3323) values, and boolean values. In addition, the VarCmplx (see page 4899) unit implements a number of global functions for operating on complex Variants (see page 4973).

See Also

VarIsComplex (see page 4942)

VarAsComplex (see page 4902)

VarComplexSimplify (see page 4934)

VarComplexFromPolar (see page 4927)

ComplexNumberSymbol (see page 4944)

ComplexNumberSymbolBeforeImaginary (see page 4945)

EConvertError (see page 3919)

48.1.24 VarCmplx.VarComplexCsc Function

Returns the cosecant of a complex number.

Pascal

```
function VarComplexCsc(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCsc(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCsc computes the cosecant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexCsc returns a Variant (see page 3421) whose value is the cosecant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCot (see page 4907)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.25 VarCmplx.VarComplexCscH Function

Returns the hyperbolic cosecant of a complex number.

Pascal

```
function VarComplexCscH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexCscH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexCsc (see page 4924) computes the hyperbolic cosecant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexCsc (see page 4924) returns a Variant (see page 3421) whose value is the hyperbolic cosecant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCsc (see page 4909)

VarComplexArcCos (see page 4905)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.26 VarCmplx.VarComplexExp Function

Returns the exponential of a complex number.

Pascal

```
function VarComplexExp(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexExp(const Variant AValue);
```

File

VarCmplx

Description

VarComplexExp returns value of e raised to the power of a specified complex number, where e is the base of the natural logarithms.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexExp returns a Variant (see page 3421) whose value is the exponential of AValue.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexAbsSqr (see page 4904)

VarComplexInverse (see page 4927)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexLn (see page 4928)

VarComplexConjugate (see page 4918)

VarComplexSqr (see page 4936)

VarComplexSqrt (see page 4937)

VarComplexPower (see page 4930)

48.1.27 VarCmplx.VarComplexFromPolar Function

Converts a set of polar coordinates to a complex number.

Pascal

```
function VarComplexFromPolar(const ARadius: Double; const ATheta: Double): Variant;
```

C++

```
Variant VarComplexFromPolar(const Double ARadius, const Double ATheta);
```

File

VarCmplx

Description

VarComplexFromPolar returns the complex number that corresponds to the radius (length) and angle specified by the ARadius and ATheta parameters, respectively. The result is returned as a Variant (see page 3421) whose value is the corresponding complex number.

See Also

VarComplexCreate (see page 4923)

VarComplexToPolar (see page 4942)

48.1.28 VarCmplx.VarComplexInverse Function

Returns the inverse of a complex number.

Pascal

```
function VarComplexInverse(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexInverse(const Variant AValue);
```

File

VarCmplx

Description

VarComplexInverse returns the inverse of a complex number. The inverse is the value that results when you divide 1 by the complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexInverse returns a Variant (see page 3421) whose value is the inverse of AValue.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexAbsSqr (see page 4904)

VarComplexConjugate (see page 4918)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexLn (see page 4928)

VarComplexExp (see page 4926)

VarComplexSqr (see page 4936)

VarComplexSqrt (see page 4937)

VarComplexPower (see page 4930)

48.1.29 VarCmplx.VarComplexLn Function

Returns the natural log of a complex number.

Pascal

```
function VarComplexLn(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLn(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLn returns the natural logarithm of a complex number. The natural logarithm is the power to which you must raise e to get the complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexLn returns a Variant (see page 3421) whose value is the natural log of AValue.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexAbsSqr (see page 4904)

VarComplexConjugate (see page 4918)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexInverse (see page 4927)

VarComplexExp (see page 4926)

VarComplexSqr (see page 4936)

VarComplexSqrt (see page 4937)

VarComplexPower (see page 4930)

48.1.30 VarCmplx.VarComplexLog10 Function

Returns the log base 10 of a complex number.

Pascal

```
function VarComplexLog10(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLog10(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLog10 returns the base 10 logarithm of a complex number. The base 10 logarithm is the power to which you must raise 10 to get the complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexLog10 returns a Variant (see page 3421) whose value is the log base 10 of AValue.

See Also

VarComplexLn (see page 4928)

VarComplexLog2 (see page 4929)

VarComplexLogN (see page 4930)

48.1.31 VarCmplx.VarComplexLog2 Function

Returns the log base 2 of a complex number.

Pascal

```
function VarComplexLog2(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexLog2(const Variant AValue);
```

File

VarCmplx

Description

VarComplexLog2 returns the base 2 logarithm of a complex number. The base 2 logarithm is the power to which you must raise 2 to get the complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexLog2 returns a Variant (see page 3421) whose value is the log base 2 of AValue.

See Also

VarComplexLn (see page 4928)

VarComplexLog10 (see page 4929)

VarComplexLogN (see page 4930)

48.1.32 VarCmplx.VarComplexLogN Function

Returns the log base N of a complex number.

Pascal

```
function VarComplexLogN(const AValue: Variant; const X: Double): Variant;
```

C++

```
Variant VarComplexLogN(const Variant AValue, const Double X);
```

File

VarCmplx

Description

VarComplexLogN returns the base N logarithm of a complex number. The base N logarithm is the power to which you must raise an exponent N to get the complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

X is the logarithm exponent.

VarComplexLogN returns a Variant (see page 3421) whose value is the log base N of AValue.

See Also

VarComplexLn (see page 4928)

VarComplexLog2 (see page 4929)

VarComplexLog10 (see page 4929)

48.1.33 VarCmplx.VarComplexPower Function

Returns the value of a complex number raised to a specified power.

Pascal

```
function VarComplexPower(const AValue: Variant; const APower: Variant): Variant;
```

C++

```
Variant VarComplexPower(const Variant AValue, const Variant APower);
```

File

VarCmplx

Description

VarComplexPower returns the result of raising AValue to the power APower.

AValue is a Variant (see page 3421) whose value represents the complex number.

APower is a Variant (see page 3421) that specifies the power to which you want to raise AValue. It can also be a complex number.

VarComplexPower returns a Variant (see page 3421) whose value is AValue raised to the APower power.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexAbsSqr (see page 4904)

VarComplexConjugate (see page 4918)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexInverse (see page 4927)

VarComplexExp (see page 4926)

VarComplexSqr (see page 4936)

VarComplexSqrt (see page 4937)

VarComplexLn (see page 4928)

48.1.34 VarCmplx.VarComplexSec Function

Returns the secant of a complex number.

Pascal

```
function VarComplexSec(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSec(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSec computes the secant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexSec returns a Variant (see page 3421) whose value is the secant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCot (see page 4907)

VarComplexArcCscH (see page 4911)

VarComplexArcCsc (see page 4909)

VarComplexArcSecH (see page 4913)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.35 VarCmplx.VarComplexSecH Function

Returns the hyperbolic secant of a complex number.

Pascal

```
function VarComplexSecH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSecH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSec (see page 4931) computes the hyperbolic secant of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexSec (see page 4931) returns a Variant (see page 3421) whose value is the hyperbolic secant of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCos (see page 4905)

VarComplexArcCotH (see page 4908)

VarComplexArcCot (see page 4907)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexSec (see page 4931)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.36 VarCmplx.VarComplexSign Function

Pascal

```
function VarComplexSign(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSign(const Variant AValue);
```

File

VarCmplx

Description

This is function VarCmplx.VarComplexSign.

48.1.37 VarCmplx.VarComplexSimplify Function

Reduces a Variant (see page 3421) that represents a complex number to one that represents a real value, if possible.

Pascal

```
function VarComplexSimplify(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSimplify(const Variant AValue);
```

File

VarCmplx

Description

Use VarComplexSimplify to convert a complex number to a real value when possible. VarComplexSimplify checks whether the Variant (see page 3421) specified by AValue represents a complex number that has no imaginary part. If so, it returns a Variant (see page 3421) whose value is simply a real number. If AValue does not represent a complex number, or if AValue represents a complex number with a nonzero imaginary part, VarComplexSimplify returns AValue.

See Also

VarIsComplex (see page 4942)

VarComplexCreate (see page 4923)

VarAsComplex (see page 4902)

VarAsType (see page 5073)

48.1.38 VarCmplx.VarComplexSin Function

Returns the sine of a complex number.

Pascal

```
function VarComplexSin(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSin(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSin computes the sine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexSin returns a Variant (see page 3421) whose value is the sine of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcSinH (see page 4915)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.39 VarCmplx.VarComplexSinH Function

Returns the hyperbolic sine of a complex number.

Pascal

```
function VarComplexSinH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSinH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSinH computes the hyperbolic sine of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexSinH returns a Variant (see page 3421) whose value is the hyperbolic sine of AValue.

See Also

VarComplexArcSinH (see page 4915)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcCosH (see page 4906)

VarComplexArcTan (see page 4916)

VarComplexArcTanH (see page 4917)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexTan (see page 4938)

VarComplexTanH (see page 4939)

48.1.40 VarCmplx.VarComplexSqr Function

Returns the square of a complex number.

Pascal

```
function VarComplexSqr(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSqr(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSqr returns the square of AValue.

AValue is a Variant (see page 3421) whose value represents the complex number.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexAbsSqr (see page 4904)

VarComplexConjugate (see page 4918)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexInverse (see page 4927)

VarComplexExp (see page 4926)

VarComplexLn (see page 4928)

VarComplexSqrt (see page 4937)

VarComplexPower (see page 4930)

48.1.41 VarCmplx.VarComplexSqrt Function

Returns the square root of a complex number.

Pascal

```
function VarComplexSqrt(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexSqrt(const Variant AValue);
```

File

VarCmplx

Description

VarComplexSqrt returns the square root of AValue.

AValue is a Variant (see page 3421) whose value represents the complex number.

See Also

VarComplexAbs (see page 4903)

VarComplexAngle (see page 4905)

VarComplexAbsSqr (see page 4904)

VarComplexConjugate (see page 4918)

VarComplexTimesPosI (see page 4941)

VarComplexTimesNegI (see page 4940)

VarComplexInverse (see page 4927)

VarComplexExp (see page 4926)

VarComplexLn (see page 4928)

VarComplexSqr (see page 4936)

VarComplexPower (see page 4930)

48.1.42 VarCmplx.VarComplexTan Function

Returns the tangent of a complex number.

Pascal

```
function VarComplexTan(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTan(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTan computes the tangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexTan returns a Variant (see page 3421) whose value is the tangent of AValue.

See Also

VarComplexArcCosH (see page 4906)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcTanH (see page 4917)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTanH (see page 4939)

48.1.43 VarCmplx.VarComplexTanH Function

Returns the hyperbolic tangent of a complex number.

Pascal

```
function VarComplexTanH(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTanH(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTanH computes the hyperbolic tangent of a complex number and returns the result.

AValue is a Variant (see page 3421) whose value represents the complex number.

VarComplexTanH returns a Variant (see page 3421) whose value is the hyperbolic tangent of AValue.

See Also

VarComplexArcTan (see page 4916)

VarComplexArcCot (see page 4907)

VarComplexArcCotH (see page 4908)

VarComplexArcCsc (see page 4909)

VarComplexArcCscH (see page 4911)

VarComplexArcSec (see page 4912)

VarComplexArcSecH (see page 4913)

VarComplexArcCos (see page 4905)

VarComplexArcCosH (see page 4906)

VarComplexArcSin (see page 4914)

VarComplexArcSinH (see page 4915)

VarComplexCos (see page 4919)

VarComplexCosH (see page 4920)

VarComplexCot (see page 4921)

VarComplexCotH (see page 4922)

VarComplexCsc (see page 4924)

VarComplexCscH (see page 4925)

VarComplexSec (see page 4931)

VarComplexSecH (see page 4932)

VarComplexSin (see page 4934)

VarComplexSinH (see page 4935)

VarComplexTan (see page 4938)

48.1.44 VarCmplx.VarComplexTimesImaginary Function

Returns the value of a complex number multiplied by an imaginary number.

Pascal

```
function VarComplexTimesImaginary(const AValue: Variant; const AFactor: Double): Variant;
```

C++

```
Variant VarComplexTimesImaginary(const Variant AValue, const Double AFactor);
```

File

VarCmplx

Description

VarComplexTimesImaginary returns a complex value times an imaginary number.

AValue is a Variant (see page 3421) whose value represents the complex number.

AFactor is a real number that represents the imaginary number. That is, the imaginary number is AFactor times i.

VarComplexTimesImaginary returns a variant whose value is AValue times AFactor times i.

See Also

VarComplexTimesNegI (see page 4940)

VarComplexTimesPosI (see page 4941)

VarComplexTimesReal (see page 4941)

48.1.45 VarCmplx.VarComplexTimesNegI Function

Returns the value of a complex number multiplied by –i.

Pascal

```
function VarComplexTimesNegI(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTimesNegI(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTimesNegI returns AValue times –i.

AValue is a Variant (see page 3421) whose value represents the complex number.

See Also

VarComplexTimesImaginary (see page 4940)

VarComplexTimesPosI (see page 4941)

VarComplexTimesReal (see page 4941)

48.1.46 VarCmplx.VarComplexTimesPosI Function

Returns the value of a complex number multiplied by i.

Pascal

```
function VarComplexTimesPosI(const AValue: Variant): Variant;
```

C++

```
Variant VarComplexTimesPosI(const Variant AValue);
```

File

VarCmplx

Description

VarComplexTimesPosI returns AValue times i.

AValue is a Variant (see page 3421) whose value represents the complex number.

See Also

VarComplexTimesNegI (see page 4940)

VarComplexTimesImaginary (see page 4940)

VarComplexTimesReal (see page 4941)

48.1.47 VarCmplx.VarComplexTimesReal Function

Returns the value of a complex number multiplied by a real number.

Pascal

```
function VarComplexTimesReal(const AValue: Variant; const AFactor: Double): Variant;
```

C++

```
Variant VarComplexTimesReal(const Variant AValue, const Double AFactor);
```

File

VarCmplx

Description

VarComplexTimesReal returns a complex value times a real number.

AValue is a Variant (see page 3421) whose value represents the complex number.

AFactor is a real number.

VarComplexTimesReal returns a variant whose value is AValue times AFactor.

See Also

VarComplexTimesNegI (see page 4940)

VarComplexTimesPosI (see page 4941)

48.1.48 VarCmplx.VarComplexToPolar Function

Computes the polar coordinates that correspond to a custom Variant (see page 3421) that represents a complex number.

Pascal

```
procedure VarComplexToPolar(const AValue: Variant; var ARadius: Double; var ATheta: Double;
AFixTheta: Boolean = True);
```

C++

```
VarComplexToPolar(const Variant AValue, Double ARadius, Double ATheta, Boolean AFixTheta =
True);
```

File

VarCmplx

Description

Call VarComplexToPolar to obtain the polar coordinates for a complex number.

AValue is a Variant (see page 3421) whose value represents the complex number.

ARadius returns the radius of the corresponding polar coordinates.

ATheta returns the angle (Theta) of the corresponding polar coordinates.

AFixTheta indicates whether VarComplexToPolar should normalize the angle (ATheta). When AFixTheta is true, VarComplexToPolar always returns an angle between – pi and pi.

See Also

VarComplexAngle (see page 4905)

VarComplexFromPolar (see page 4927)

VarComplexAbs (see page 4903)

48.1.49 VarCmplx.VarIsComplex Function

Indicates whether a Variant (see page 3421) stores its data internally as a complex number.

Pascal

```
function VarIsComplex(const AValue: Variant): Boolean;
```

C++

```
Boolean VarIsComplex(const Variant AValue);
```

File

VarCmplx

Description

VarIsComplex returns true if AValue represents a complex number (that is, if it has a VarType (see page 5090) of VarComplex (see page 4903)).

Complex Variants (see page 4973) are custom Variants (see page 4973) that represent complex numbers. The Variant (see page 3421) type supports direct manipulation using the addition, subtraction, multiplication, division (but not integer division), and negation operators. They have 5 published properties: Real, Imaginary, Radius, Theta, and FixedTheta. They can be cast to and from integer types, floating point types, string types, TDateTime (see page 3323) values, and boolean values. In addition, the VarCmplx (see page 4899) unit implements a number of global functions for operating on complex Variants (see page 4973).

See Also

VarAsComplex (see page 4902)

VarComplexCreate (see page 4923)

VarComplexSimplify (see page 4934)

VarComplex (see page 4903)

VarType (see page 5090)

48.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ComplexNumberDefuzzAtZero (see page 4944)	This is variable VarCmplx.ComplexNumberDefuzzAtZero.
ComplexNumberSymbol (see page 4944)	Indicates the symbol used to represent the imaginary part of a complex number.
ComplexNumberSymbolBeforeImaginary (see page 4945)	Indicates the placement of the symbol that represents the imaginary part of a complex number.

48.2.1 VarCmplx.ComplexNumberDefuzzAtZero Variable

Pascal

```
ComplexNumberDefuzzAtZero: Boolean = True;
```

C++

```
Boolean ComplexNumberDefuzzAtZero = True;
```

File

VarCmplx

Description

This is variable VarCmplx.ComplexNumberDefuzzAtZero.

48.2.2 VarCmplx.ComplexNumberSymbol Variable

Indicates the symbol used to represent the imaginary part of a complex number.

Pascal

```
ComplexNumberSymbol: string = 'i';
```

C++

```
UnicodeString ComplexNumberSymbol = 'i';
```

File

VarCmplx

Description

Custom Variants (see page 4973) that represent complex numbers use ComplexNumberSymbol when converting between complex values and strings. This identifies the symbol used to signal the imaginary part of a complex number. By default, this value is 'i'. This allows the custom Variant (see page 3421) for complex numbers to interpret or produce strings such as '3.4 + 2i'.

See Also

ComplexNumberSymbolBeforeImaginary (see page 4945)

48.2.3 VarCmplx.ComplexNumberSymbolBeforeImaginary Variable

Indicates the placement of the symbol that represents the imaginary part of a complex number.

Pascal

```
ComplexNumberSymbolBeforeImaginary: Boolean = False;
```

C++

```
Boolean ComplexNumberSymbolBeforeImaginary = False;
```

File

VarCmplx

Description

Custom Variants (see page 4973) that represent complex numbers use ComplexNumberSymbolBeforeImaginary when converting between complex values and strings.

When ComplexNumberSymbolBeforeImaginary is false (the default), the symbol appears after the imaginary part, as in '3.4 + 2i'.

When ComplexNumberSymbolBeforeImaginary is true, the symbol appears before the imaginary part, as in '3.4 + i2'.

See Also

ComplexNumberSymbol (see page 4944)

49 VarConv Namespace

This is namespace VarConv.

Topics

Name	Description
Functions (see page 4948)	The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsConvert (see page 4948)	Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a measurement.
	VarAsConvert (see page 4948)	Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a measurement.
	VarConvert (see page 4949)	Returns the Variant (see page 3421) type code for custom Variants (see page 4973) that represent measurements.
	VarConvertCreate (see page 4950)	Returns a custom Variant (see page 3421) that represents a measurement.
	VarConvertCreate (see page 4950)	Returns a custom Variant (see page 3421) that represents a measurement.
	VarIsConvert (see page 4951)	Indicates whether a Variant (see page 3421) is a custom Variant (see page 3421) that represents a measurement.

49.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VarAsConvert (see page 4948)	Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a measurement.
	VarAsConvert (see page 4948)	Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a measurement.
	VarConvert (see page 4949)	Returns the Variant (see page 3421) type code for custom Variants (see page 4973) that represent measurements.
	VarConvertCreate (see page 4950)	Returns a custom Variant (see page 3421) that represents a measurement.
	VarConvertCreate (see page 4950)	Returns a custom Variant (see page 3421) that represents a measurement.
	VarIsConvert (see page 4951)	Indicates whether a Variant (see page 3421) is a custom Variant (see page 3421) that represents a measurement.

49.1.1 VarConv.VarAsConvert Function

Casts an arbitrary Variant (see page 3421) so that it is a custom Variant (see page 3421) representing a measurement.

Pascal

```
function VarAsConvert(const AValue: Variant): Variant; overload;
function VarAsConvert(const AValue: Variant; const AType: TConvType): Variant; overload;
```

C++

```
Variant VarAsConvert(const Variant AValue);
Variant VarAsConvert(const Variant AValue, const TConvType AType);
```

File

VarConv

Description

VarAsConvert converts the Variant (see page 3421) specified by AValue to a VarConvert (see page 4949) custom Variant (see page 3421).

Using the first syntax, AValue is interpreted as the numeric portion (the number of units), and AType identifies the units.

Using the second syntax, AValue should be a Variant (see page 3421) that can be parsed into a VarConv (see page 4947) custom Variant (see page 3421) (for example, a variant whose value is a string such as '1.5 miles').

If VarAsConvert can't perform the conversion, it raises an EInvalidCast (see page 3935) exception.

VarConvert (see page 4949) custom Variants (see page 4973) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4947) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4947) Variant (see page 3421) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4947) Variants (see page 4973). VarConv (see page 4947) custom Variants (see page 4973) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3421) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

EInvalidCast (see page 3935)

VarIsConvert (see page 4951)

VarConvertCreate (see page 4950)

VarAsType (see page 5073)

TVarData (see page 3740)

49.1.2 VarConv.VarConvert Function

Returns the Variant (see page 3421) type code for custom Variants (see page 4973) that represent measurements.

Pascal

```
function VarConvert: TVarType;
```

C++

```
TVarType VarConvert();
```

File

VarConv

Description

Use VarConvert to obtain the variant type code for custom Variants (see page 4973) that represent measurements. The Variant (see page 3421) type codes for custom Variants (see page 4973) are dynamically allocated when the application loads the unit that implements them. VarConvert allows the application to identify the Variant (see page 3421) type code that is currently used for custom Variants (see page 4973) that represent VarConvert custom Variants (see page 4973). This type code, however, may not match the type code used by another application that uses a different set of custom Variant (see page 3421) types.

VarConvert custom Variants (see page 4973) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4947) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4947) Variant (see page 3421) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4947) Variants (see page 4973). VarConv (see page 4947) custom Variants (see page 4973) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3421) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarType (see page 5090)

49.1.3 VarConv.VarConvertCreate Function

Returns a custom Variant (see page 3421) that represents a measurement.

Pascal

```
function VarConvertCreate(const AValue: Double; const AType: TConvType): Variant; overload;
function VarConvertCreate(const AValue: string): Variant; overload;
```

C++

```
Variant VarConvertCreate(const Double AValue, const TConvType AType);
Variant VarConvertCreate(const UnicodeString AValue);
```

File

VarConv

Description

Call VarConvertCreate to create a Variant (see page 3421) whose value is a measurement.

Using the first syntax, AType indicates the measurement units, and AValue is the number of those units. AType must represent a registered conversion unit.

Using the second syntax, AValue is a string that can be parsed by the TryStrToConvUnit function. (For example '1.5 liters'.)

If the string can't be parsed into a measurement, VarConvertCreate raises an EInvalidCast (see page 3935) exception.

VarConvert (see page 4949) custom Variants (see page 4973) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4947) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4947) Variant (see page 3421) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4947) Variants (see page 4973). VarConv (see page 4947) custom Variants (see page 4973) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3421) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarIsConvert (see page 4951)

VarAsConvert (see page 4948)

EInvalidCast (see page 3935)

49.1.4 VarConv.VarIsConvert Function

Indicates whether a Variant (see page 3421) is a custom Variant (see page 3421) that represents a measurement.

Pascal

```
function VarIsConvert(const AValue: Variant): Boolean;
```

C++

```
Boolean VarIsConvert(const Variant AValue);
```

File

VarConv

Description

VarIsConvert returns true if AValue represents a measurement (that is, if it has a VarType (see page 5090) of VarConvert (see page 4949)).

VarConvert (see page 4949) custom Variants (see page 4973) represent measurements (that is, a type of measurement units and the number of those units). The units must be registered measurement units such as those used by the ConvUtils unit. You can use the negation operator on VarConv (see page 4947) variants, and use the addition, subtraction, multiplication, division, and comparison operators on a VarConv (see page 4947) Variant (see page 3421) with a real number. You can use the addition, subtraction, division, and comparison operators on two VarConv (see page 4947) Variants (see page 4973). VarConv (see page 4947) custom Variants (see page 4973) also support the following properties:

Value is the numeric portion (the number of units). (read/write)

Type is units. (read-only)

TypeName is the name of the units. (read-only).

Family is the conversion family of the units (the thing measured). (read-only).

FamilyName is the name of the conversion family. (read-only)

Depending on the Family, each Variant (see page 3421) has a set of conversion properties with names of the form As<Unit>, where <Unit> is the name of another type of unit in the same conversion family.

See Also

VarAsConvert (see page 4948)

VarConvertCreate (see page 4950)

VarConvert (see page 4949)

VarType (see page 5090)

50 VarHlpr Namespace

This is namespace VarHlpr.

Topics

Name	Description
Functions (see page 4955)	The following table lists functions in this documentation.

Functions

	Name	Description
	VariantAdd (see page 4955)	Adds two Variants (see page 4973).
	VariantAdd2 (see page 4956)	Adds two Variants (see page 4973).
	VariantAnd (see page 4956)	Performs an **and** operation between two Variant (see page 3421) values.
	VariantAnd2 (see page 4957)	Performs an **and** operation between two Variants (see page 4973).
	VariantArrayRedim (see page 4957)	Resizes a Variant (see page 3421) array.
	VariantCast (see page 4958)	Performs a type-cast on a Variant (see page 3421).
	VariantClear (see page 4958)	Empties a Variant (see page 3421), so that it is unassigned.
	VariantCpy (see page 4959)	Copies a Variant (see page 3421).
	VariantDiv (see page 4959)	Divides two Variants (see page 4973).
	VariantDiv2 (see page 4960)	Divides two Variants (see page 4973).
	VariantFromUnicodeString (see page 4960)	Assigns an Unicode string to a Variant (see page 3421).
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantMod (see page 4962)	Returns the remainder on division of two Variants (see page 4973).
	VariantMod2 (see page 4962)	Returns the remainder on division of two Variants (see page 4973).
	VariantMul (see page 4963)	Multiplies two Variants (see page 4973).
	VariantMul2 (see page 4963)	Multiplies two Variants (see page 4973).
	VariantNeg (see page 4964)	Inverts the sign of a Variant (see page 3421).
	VariantNot (see page 4964)	Performs a **not** operation on a Variant (see page 3421).
	VariantOr (see page 4964)	Performs an **or** operation between two Variants (see page 4973).
	VariantOr2 (see page 4965)	Performs an **or** operation between two Variants (see page 4973).
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.

	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantShl (see page 4966)	Performs an **shl** (shift left) operation between two Variants (see page 4973).
	VariantShl2 (see page 4967)	Performs an **shl** (shift left) operation between two Variants (see page 4973).
	VariantShr (see page 4967)	Performs an **shr** (shift right) operation between two Variants (see page 4973).
	VariantShr2 (see page 4968)	Performs an **shr** (shift right) operation between two Variants (see page 4973).
	VariantSub (see page 4968)	Subtracts two Variants (see page 4973).
	VariantSub2 (see page 4969)	Subtracts two Variants (see page 4973).
	VariantToUnicodeString (see page 4969)	Converts a Variant (see page 3421) to an Unicode string.
	VariantXor (see page 4970)	Performs a **xor** (exclusive or) operation between two Variants (see page 4973).
	VariantXor2 (see page 4970)	Performs a **xor** (exclusive or) operation between two Variants (see page 4973).

50.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	VariantAdd (see page 4955)	Adds two Variants (see page 4973).
	VariantAdd2 (see page 4956)	Adds two Variants (see page 4973).
	VariantAnd (see page 4956)	Performs an **and** operation between two Variant (see page 3421) values.
	VariantAnd2 (see page 4957)	Performs an **and** operation between two Variants (see page 4973).
	VariantArrayRedim (see page 4957)	Resizes a Variant (see page 3421) array.
	VariantCast (see page 4958)	Performs a type-cast on a Variant (see page 3421).
	VariantClear (see page 4958)	Empties a Variant (see page 3421), so that it is unassigned.
	VariantCpy (see page 4959)	Copies a Variant (see page 3421).
	VariantDiv (see page 4959)	Divides two Variants (see page 4973).
	VariantDiv2 (see page 4960)	Divides two Variants (see page 4973).
	VariantFromUnicodeString (see page 4960)	Assigns an Unicode string to a Variant (see page 3421).
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantGetElement (see page 4961)	Returns an element from a Variant (see page 3421) array.
	VariantMod (see page 4962)	Returns the remainder on division of two Variants (see page 4973).
	VariantMod2 (see page 4962)	Returns the remainder on division of two Variants (see page 4973).
	VariantMul (see page 4963)	Multiplies two Variants (see page 4973).
	VariantMul2 (see page 4963)	Multiplies two Variants (see page 4973).
	VariantNeg (see page 4964)	Inverts the sign of a Variant (see page 3421).
	VariantNot (see page 4964)	Performs a **not** operation on a Variant (see page 3421).
	VariantOr (see page 4964)	Performs an **or** operation between two Variants (see page 4973).
	VariantOr2 (see page 4965)	Performs an **or** operation between two Variants (see page 4973).
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantPutElement (see page 4966)	Sets the value of an element in a Variant (see page 3421) array.
	VariantShl (see page 4966)	Performs an **shl** (shift left) operation between two Variants (see page 4973).
	VariantShl2 (see page 4967)	Performs an **shl** (shift left) operation between two Variants (see page 4973).
	VariantShr (see page 4967)	Performs an **shr** (shift right) operation between two Variants (see page 4973).
	VariantShr2 (see page 4968)	Performs an **shr** (shift right) operation between two Variants (see page 4973).
	VariantSub (see page 4968)	Subtracts two Variants (see page 4973).
	VariantSub2 (see page 4969)	Subtracts two Variants (see page 4973).
	VariantToUnicodeString (see page 4969)	Converts a Variant (see page 3421) to an Unicode string.
	VariantXor (see page 4970)	Performs a **xor** (exclusive or) operation between two Variants (see page 4973).
	VariantXor2 (see page 4970)	Performs a **xor** (exclusive or) operation between two Variants (see page 4973).

50.1.1 VarHlpr.VariantAdd Function

Adds two Variants (see page 4973).

Pascal

```
procedure VariantAdd(const src: Variant; var dst: Variant);
```

C++

```
VariantAdd(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantAdd adds **src** to **dst** and returns the result as **dst**.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAdd2 (see page 4956)

VariantSub (see page 4968)

VariantSub2 (see page 4969)

50.1.2 VarHlpr.VariantAdd2 Function

Adds two Variants (see page 4973).

Pascal

```
function VariantAdd2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantAdd2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantAdd2 adds **V1** to **V2** and returns a new Variant (see page 3421) holding the result.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAdd (see page 4955)

VariantSub (see page 4968)

VariantSub2 (see page 4969)

50.1.3 VarHlpr.VariantAnd Function

Performs an **and** operation between two Variant (see page 3421) values.

Pascal

```
procedure VariantAnd(const src: Variant; var dst: Variant);
```

C++

```
VariantAnd(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantAnd performs a bit-wise or a logical **and** operation between **src** and **dst** and returns the result as **dst**.

The input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAnd2 (see page 4957)

VariantOr (see page 4964)

VariantOr2 (see page 4965)

50.1.4 VarHlpr.VariantAnd2 Function

Performs an **and** operation between two Variants (see page 4973).

Pascal

```
function VariantAnd2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantAnd2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantAnd2 performs a bit-wise or a logical **and** operation between **V1** and **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantAnd (see page 4956)

VariantOr (see page 4964)

VariantOr2 (see page 4965)

50.1.5 VarHlpr.VariantArrayRedim Function

Resizes a Variant (see page 3421) array.

Pascal

```
procedure VariantArrayRedim(var V: Variant; High: Integer);
```

C++

```
VariantArrayRedim(Variant V, int High);
```

File

VarHlpr

Description

VariantArrayRedim resizes the given Variant (see page 3421) array by changing the high bound of the rightmost dimension to the given value.

The input value must be a Variant (see page 3421) array; otherwise an EVariantInvalidArgError exception will be raised.

See Also

VariantGetElement (see page 4961)

VariantPutElement (see page 4966)

50.1.6 VarHlpr.VariantCast Function

Performs a type-cast on a Variant (see page 3421).

Pascal

```
procedure VariantCast(const src: Variant; var dst: Variant; vt: Integer);
```

C++

```
VariantCast(const Variant src, Variant dst, int vt);
```

File

VarHlpr

Description

VariantCast type-casts **src** to the desired type and returns the resulted Variant (see page 3421) as **dst**. **vt** is a value which identifies the type of the destination Variant (see page 3421).

An EVariantTypeCastError exception will be raised if the source cannot be type-casted to the desired type.

See Also

VariantClear (see page 4958)

VarClear (see page 3704)

50.1.7 VarHlpr.VariantClear Function

Empties a Variant (see page 3421), so that it is unassigned.

Pascal

```
procedure VariantClear(var V: Variant);
```

C++

```
VariantClear(Variant V);
```

File

VarHlpr

Description

Calling VariantClear is equivalent to assigning the Unassigned constant to the Variant (see page 3421). After calling VariantClear, the VarIsEmpty function returns true, and the VarType function returns varEmpty (see page 3865). Using an unassigned variant in an expression causes an exception to be thrown. Likewise, if you attempt to convert an unassigned Variant (see page 3421) to another type (using VariantCast (see page 4958)), an exception is thrown.

Note: Do not confuse clearing a Variant (see page 3421), which leaves it unassigned, with assigning a Null value. A Null Variant (see page 3421) is still assigned, but has the value Null. Unlike unassigned Variants (see page 4973), Null Variants (see page 4973) can be used in expressions and can be converted to other types of Variants (see page 4973).

See Also

VariantCast (see page 4958)

VarClear (see page 3704)

50.1.8 VarHlpr.VariantCpy Function

Copies a Variant (see page 3421).

Pascal

```
procedure VariantCpy(const src: Variant; var dst: Variant);
```

C++

```
VariantCpy(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantCpy copies the Variant (see page 3421) value contained in **src** to **dst**.

See Also

VariantClear (see page 4958)

VariantCast (see page 4958)

50.1.9 VarHlpr.VariantDiv Function

Divides two Variants (see page 4973).

Pascal

```
procedure VariantDiv(const src: Variant; var dst: Variant);
```

C++

```
VariantDiv(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantDiv divides **src** to **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantDiv2 (see page 4960)

VariantMul (see page 4963)

VariantMul2 (see page 4963)

50.1.10 VarHlpr.VariantDiv2 Function

Divides two Variants (see page 4973).

Pascal

```
function VariantDiv2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantDiv2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantDiv2 divides **V1** to **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantDiv (see page 4959)

VariantMul (see page 4963)

VariantMul2 (see page 4963)

50.1.11 VarHlpr.VariantFromUnicodeString Function

Assigns an Unicode string to a Variant (see page 3421).

Pascal

```
procedure VariantFromUnicodeString(var V: Variant; const Str: UnicodeString);
```

C++

```
VariantFromUnicodeString(Variant V, const UnicodeString Str);
```

File

VarHlpr

Description

VariantFromUnicodeString assigns an Unicode string to a given Variant (see page 3421) parameter. The old value, if there was one, of the Variant (see page 3421) will be lost.

See Also

VariantToUnicodeString (see page 4969)

VariantClear (see page 4958)

VariantCast (see page 4958)

50.1.12 VarHlpr.VariantGetElement Function

Returns an element from a Variant (see page 3421) array.

Pascal

```
function VariantGetElement(const V: Variant; i1: integer): Variant; overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer): Variant; overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer; i3: integer): Variant;
overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer; i3: integer; i4:
integer): Variant; overload;
function VariantGetElement(const V: Variant; i1: integer; i2: integer; i3: integer; i4:
integer; i5: integer): Variant; overload;
```

C++

```
Variant VariantGetElement(const Variant V, int i1);
Variant VariantGetElement(const Variant V, int i1, int i2);
Variant VariantGetElement(const Variant V, int i1, int i2, int i3);
Variant VariantGetElement(const Variant V, int i1, int i2, int i3, int i4);
Variant VariantGetElement(const Variant V, int i1, int i2, int i3, int i4, int i5);
```

File

VarHlpr

Description

VariantGetElement returns the Variant (see page 3421) value located at the specified indices in the Variant (see page 3421) array **V**.

This function comes in five forms. Each form accepts a Variant (see page 3421) array and a number of indices in that array, one for each dimension.

See Also

VariantPutElement (see page 4966)

VarArrayGet (see page 5063)

VarArrayPut (see page 5069)

50.1.13 VarHlpr.VariantMod Function

Returns the remainder on division of two Variants (see page 4973).

Pascal

```
procedure VariantMod(const src: Variant; var dst: Variant);
```

C++

```
VariantMod(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantMod calculates the remainder on division of **src** by **dst** and returns the result as **dst**.

The input values must be convertible to integer numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMod2 (see page 4962)

VariantDiv (see page 4959)

VariantDiv2 (see page 4960)

50.1.14 VarHlpr.VariantMod2 Function

Returns the remainder on division of two Variants (see page 4973).

Pascal

```
function VariantMod2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantMod2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantMod2 calculates the remainder on division of **V1** by **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to integer numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMod (see page 4962)

VariantDiv (see page 4959)

VariantDiv2 (see page 4960)

50.1.15 VarHlpr.VariantMul Function

Multiplies two Variants (see page 4973).

Pascal

```
procedure VariantMul(const src: Variant; var dst: Variant);
```

C++

```
VariantMul(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantMul multiplies **src** to **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMul2 (see page 4963)

VariantDiv (see page 4959)

VariantDiv2 (see page 4960)

50.1.16 VarHlpr.VariantMul2 Function

Multiplies two Variants (see page 4973).

Pascal

```
function VariantMul2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantMul2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantMul2 multiplies **V1** to **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantMul (see page 4963)

VariantDiv (see page 4959)

VariantDiv2 (see page 4960)

50.1.17 VarHlpr.VariantNeg Function

Inverts the sign of a Variant (see page 3421).

Pascal

```
function VariantNeg(const V1: Variant): Variant;
```

C++

```
Variant VariantNeg(const Variant V1);
```

File

VarHlpr

Description

VariantNeg will invert the sign of **V1** from positive to negative and vice-versa.

The input parameter must be convertible to a numerical type; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantNot (see page 4964)

50.1.18 VarHlpr.VariantNot Function

Performs a **not** operation on a Variant (see page 3421).

Pascal

```
function VariantNot(const V1: Variant): Variant;
```

C++

```
Variant VariantNot(const Variant V1);
```

File

VarHlpr

Description

VariantNot performs a bit-wise or a logical **not** operation on **V1** and returns a new Variant (see page 3421) holding the result.

The input value must be convertible to a numerical or boolean type; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantNeg (see page 4964)

50.1.19 VarHlpr.VariantOr Function

Performs an **or** operation between two Variants (see page 4973).

Pascal

```
procedure VariantOr(const src: Variant; var dst: Variant);
```

C++

```
VariantOr(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantOr performs a bit-wise or a logical **or** operation between **src** and **dst** and returns the result as **dst**.

Given input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantOr2 (see page 4965)

VariantAnd (see page 4956)

VariantAnd2 (see page 4957)

50.1.20 VarHlpr.VariantOr2 Function

Performs an **or** operation between two Variants (see page 4973).

Pascal

```
function VariantOr2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantOr2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantOr2 performs a bit-wise or a logical **or** operation between **V1** and **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to numerical or boolean types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantOr (see page 4964)

VariantAnd (see page 4956)

VariantAnd2 (see page 4957)

50.1.21 VarHlpr.VariantPutElement Function

Sets the value of an element in a Variant (see page 3421) array.

Pascal

```
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer); overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer);
overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer; i3:
integer); overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer; i3:
integer; i4: integer); overload;
procedure VariantPutElement(var V: Variant; const data: Variant; i1: integer; i2: integer; i3:
integer; i4: integer; i5: integer); overload;
```

C++

```
VariantPutElement(Variant V, const Variant data, int i1);
VariantPutElement(Variant V, const Variant data, int i1, int i2);
VariantPutElement(Variant V, const Variant data, int i1, int i2, int i3);
VariantPutElement(Variant V, const Variant data, int i1, int i2, int i3, int i4);
VariantPutElement(Variant V, const Variant data, int i1, int i2, int i3, int i4, int i5);
```

File

VarHlpr

Description

VariantPutElement sets the Variant (see page 3421) value located at the specified indices in the Variant (see page 3421) array **V**.

This function comes in five forms. Each form accepts a Variant (see page 3421) array, a Variant (see page 3421) value to assign, and a number of indices in that array, one for each dimension.

See Also

VariantGetElement (see page 4961)

VarArrayGet (see page 5063)

VarArrayPut (see page 5069)

50.1.22 VarHlpr.VariantShl Function

Performs an **shl** (shift left) operation between two Variants (see page 4973).

Pascal

```
procedure VariantShl(const src: Variant; var dst: Variant);
```

C++

```
VariantShl(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantShl performs a bit-wise **shl** (shift left) operation between **src** and **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantShr (see page 4967)

VariantShr2 (see page 4968)

50.1.23 VarHlpr.VariantShl2 Function

Performs an **shl** (shift left) operation between two Variants (see page 4973).

Pascal

```
function VariantShl2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantShl2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantShl2 performs a bit-wise **shl** (shift left) operation between **V1** and **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantShl (see page 4966)

VariantShr (see page 4967)

VariantShr2 (see page 4968)

50.1.24 VarHlpr.VariantShr Function

Performs an **shr** (shift right) operation between two Variants (see page 4973).

Pascal

```
procedure VariantShr(const src: Variant; var dst: Variant);
```

C++

```
VariantShr(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantShr performs a bit-wise **shr** (shift right) operation between **src** and **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantShr2 (see page 4968)

VariantShl (see page 4966)

VariantShl2 (see page 4967)

50.1.25 VarHlpr.VariantShr2 Function

Performs an **shr** (shift right) operation between two Variants (see page 4973).

Pascal

```
function VariantShr2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantShr2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantShr2 performs a bit-wise **shr** (shift right) operation between **V1** and **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantShr (see page 4967)

VariantShl (see page 4966)

VariantShl2 (see page 4967)

50.1.26 VarHlpr.VariantSub Function

Subtracts two Variants (see page 4973).

Pascal

```
procedure VariantSub(const src: Variant; var dst: Variant);
```

C++

```
VariantSub(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantSub subtracts **src** from **dst** and returns the result as **dst**.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantSub2 (see page 4969)

VariantAdd (see page 4955)

VariantAdd2 (see page 4956)

50.1.27 VarHlpr.VariantSub2 Function

Subtracts two Variants (see page 4973).

Pascal

```
function VariantSub2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantSub2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantSub2 subtracts **V1** from **V2** and returns a new Variant (see page 3421) holding the result.

The input values must have compatible types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantSub (see page 4968)

VariantAdd (see page 4955)

VariantAdd2 (see page 4956)

50.1.28 VarHlpr.VariantToUnicodeString Function

Converts a Variant (see page 3421) to an Unicode string.

Pascal

```
procedure VariantToUnicodeString(const V: Variant; var Str: UnicodeString);
```

C++

```
VariantToUnicodeString(const Variant V, UnicodeString Str);
```

File

VarHlpr

Description

VariantToUnicodeString converts the parameter **V** to an Unicode string and places the result into **Str (see page 3676)**.

See Also

VariantFromUnicodeString (see page 4960)

VariantClear (see page 4958)

VariantCast (see page 4958)

50.1.29 VarHlpr.VariantXor Function

Performs a **xor** (exclusive or) operation between two Variants (see page 4973).

Pascal

```
procedure VariantXor(const src: Variant; var dst: Variant);
```

C++

```
VariantXor(const Variant src, Variant dst);
```

File

VarHlpr

Description

VariantXor performs a bit-wise **xor** (exclusive or) operation between **src** and **dst** and returns the result as **dst**.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantXor2 (see page 4970)

VariantOr (see page 4964)

VariantOr2 (see page 4965)

50.1.30 VarHlpr.VariantXor2 Function

Performs a **xor** (exclusive or) operation between two Variants (see page 4973).

Pascal

```
function VariantXor2(const V1: Variant; const V2: Variant): Variant;
```

C++

```
Variant VariantXor2(const Variant V1, const Variant V2);
```

File

VarHlpr

Description

VariantXor2 performs a bit-wise **xor** (exclusive or) operation between **V1** and **V2** and returns a new Variant (see page 3421) holding the result.

The input values must be convertible to numerical types; otherwise an EVariantTypeCastError exception will be raised.

See Also

VariantXor (see page 4970)

VariantOr (see page 4964)

VariantOr2 (see page 4965)

51 Variants Namespace

This is namespace Variants.

Topics

Name	Description
Classes (see page 4978)	The following table lists classes in this documentation.
Interfaces (see page 5022)	The following table lists interfaces in this documentation.
Functions (see page 5027)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5094)	The following table lists structs, records, enums in this documentation.
Types (see page 5097)	The following table lists types in this documentation.
Variables (see page 5100)	The following table lists variables in this documentation.

Classes

	Name	Description
	EVariantArrayCreateError (see page 4978)	EVariantArrayCreateError is the exception class for failure to create or resize a variant array.
	EVariantArrayLockedError (see page 4979)	EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.
	EVariantBadIndexError (see page 4980)	EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.
	EVariantBadVarTypeError (see page 4980)	EVariantBadVarTypeError is the exception class for variant type errors.
	EVariantDispatchError (see page 4981)	EVariantDispatchError is the exception class for failed dispatch attempts.
	EVariantInvalidArgError (see page 4981)	EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.
	EVariantInvalidNullOpError (see page 4982)	EVariantInvalidNullOpError is the exception class for invalid Null (see page 5056) variant operations.
	EVariantInvalidOpError (see page 4983)	EVariantInvalidOpError is the exception class for invalid variant operations.
	EVariantNotAnArrayError (see page 4983)	EVariantNotAnArrayError is the exception class inappropriate variant array operations.
	EVariantNotImplError (see page 4984)	EVariantNotImplError is the exception class for unimplemented variant operations.
	EVariantOutOfMemoryError (see page 4984)	EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.
	EVariantOverflowError (see page 4985)	EVariantOverflowError is the exception class for overflow in a variant operation.
	EVariantRangeCheckError (see page 4986)	EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.
	EVariantTypeCastError (see page 4986)	EVariantTypeCastError is the exception class for variant type cast errors.
	EVariantUnexpectedError (see page 4987)	EVariantUnexpectedError is the exception class for unexpected variant errors.

	TCustomVariantType (see page 4987)	TCustomVariantType is the base class for Delphi custom variant types.
	TInvokeableVariantType (see page 5014)	TInvokeableVariantType is the base class for custom System::Variant (see page 3421) types that implement custom Variants with properties and methods.

Functions

	Name	Description
	_DispInvoke (see page 5029)	_DispInvoke is used internally by the Delphi compiler.
	_OleVarFromInt (see page 5030)	_OleVarFromInt is used internally by the Delphi compiler.
	_OleVarFromInt64 (see page 5030)	_OleVarFromInt64 is used internally by the Delphi compiler.
	_OleVarFromLStr (see page 5031)	_OleVarFromLStr is used internally by the Delphi compiler.
	_OleVarFromPStr (see page 5031)	_OleVarFromPStr is used internally by the Delphi compiler.
	_OleVarFromUInt64 (see page 5031)	_OleVarFromUInt64 is used internally by the Delphi compiler.
	_OleVarFromUStr (see page 5032)	_OleVarFromUStr is used internally by the Delphi compiler.
	_OleVarFromVar (see page 5032)	_OleVarFromVar is used internally by the Delphi compiler.
	_VarAdd (see page 5032)	_VarAdd is used internally by the Delphi compiler.
	_VarAddRef (see page 5033)	_VarAddRef is used internally by the Delphi compiler.
	_VarAnd (see page 5033)	_VarAnd is used internally by the Delphi compiler.
	_VarArrayGet (see page 5034)	_VarArrayGet is used internally by the Delphi compiler.
	_VarArrayPut (see page 5034)	_VarArrayPut is used internally by the Delphi compiler.
	_VarArrayRedim (see page 5034)	_VarArrayRedim is used internally by the Delphi compiler.
	_VarCast (see page 5035)	_VarCast is used internally by the Delphi compiler.
	_VarCastOle (see page 5035)	_VarCastOle is used internally by the Delphi compiler.
	_VarClear (see page 5035)	_VarClear is used internally by the Delphi compiler.
	_VarClr (see page 5036)	_VarClr is used internally by the Delphi compiler.
	_VarCmpEQ (see page 5036)	_VarCmpEQ is used internally by the Delphi compiler.
	_VarCmpGE (see page 5037)	_VarCmpGE is used internally by the Delphi compiler.
	_VarCmpGT (see page 5037)	_VarCmpGT is used internally by the Delphi compiler.
	_VarCmpLE (see page 5037)	_VarCmpLE is used internally by the Delphi compiler.
	_VarCmpLT (see page 5038)	_VarCmpLT is used internally by the Delphi compiler.
	_VarCmpNE (see page 5038)	_VarCmpNE is used internally by the Delphi compiler.
	_VarCopy (see page 5038)	_VarCopy is used internally by the Delphi compiler.
	_VarCopyNoInd (see page 5039)	_VarCopyNoInd is used internally by the Delphi compiler.
	_VarDiv (see page 5039)	_VarDiv is used internally by the Delphi compiler.
	_VarFromBool (see page 5040)	_VarFromBool is used internally by the Delphi compiler.
	_VarFromCurr (see page 5040)	_VarFromCurr is used internally by the Delphi compiler.
	_VarFromDisp (see page 5040)	_VarFromDisp is used internally by the Delphi compiler.
	_VarFromDynArray (see page 5041)	_VarFromDynArray is used internally by the Delphi compiler.
	_VarFromInt (see page 5041)	_VarFromInt is used internally by the Delphi compiler.
	_VarFromInt64 (see page 5041)	_VarFromInt64 is used internally by the Delphi compiler.
	_VarFromIntf (see page 5042)	_VarFromIntf is used internally by the Delphi compiler.
	_VarFromLStr (see page 5042)	_VarFromLStr is used internally by the Delphi compiler.
	_VarFromPStr (see page 5043)	_VarFromPStr is used internally by the Delphi compiler.
	_VarFromReal (see page 5043)	_VarFromReal is used internally by the Delphi compiler.
	_VarFromTDateTime (see page 5043)	_VarFromTDateTime is used internally by the Delphi compiler.
	_VarFromUInt64 (see page 5044)	_VarFromUInt64 is used internally by the Delphi compiler.
	_VarFromUStr (see page 5044)	_VarFromUStr is used internally by the Delphi compiler.
	_VarFromWChar (see page 5044)	_VarFromWChar is used internally by the Delphi compiler.
	_VarFromWStr (see page 5045)	_VarFromWStr is used internally by the Delphi compiler.
	_VarMod (see page 5045)	_VarMod is used internally by the Delphi compiler.
	_VarMul (see page 5046)	_VarMul is used internally by the Delphi compiler.
	_VarNeg (see page 5046)	_VarNeg is used internally by the Delphi compiler.
	_VarNot (see page 5046)	_VarNot is used internally by the Delphi compiler.
	_VarOr (see page 5047)	_VarOr is used internally by the Delphi compiler.
	_VarRDiv (see page 5047)	_VarRDiv is used internally by the Delphi compiler.
	_VarShl (see page 5047)	_VarShl is used internally by the Delphi compiler.

	_VarShr (see page 5048)	_VarShr is used internally by the Delphi compiler.
	_VarSub (see page 5048)	_VarSub is used internally by the Delphi compiler.
	_VarToBool (see page 5049)	_VarToBool is used internally by the Delphi compiler.
	_VarToCurrency (see page 5049)	_VarToCurrency is used internally by the Delphi compiler.
	_VarToDisp (see page 5049)	_VarToDisp is used internally by the Delphi compiler.
	_VarToDynArray (see page 5050)	_VarToDynArray is used internally by the Delphi compiler.
	_VarToInt64 (see page 5050)	_VarToInt64 is used internally by the Delphi compiler.
	_VarToInteger (see page 5050)	_VarToInteger is used internally by the Delphi compiler.
	_VarToIntf (see page 5051)	_VarToIntf is used internally by the Delphi compiler.
	_VarToLStr (see page 5051)	_VarToLStr is used internally by the Delphi compiler.
	_VarToPStr (see page 5052)	_VarToPStr is used internally by the Delphi compiler.
	_VarToReal (see page 5052)	_VarToReal is used internally by the Delphi compiler.
	_VarToUInt64 (see page 5052)	_VarToUInt64 is used internally by the Delphi compiler.
	_VarToUStr (see page 5053)	_VarToUStr is used internally by the Delphi compiler.
	_VarToWStr (see page 5053)	_VarToWStr is used internally by the Delphi compiler.
	_VarXor (see page 5053)	_VarXor is used internally by the Delphi compiler.
	DynArrayFromVariant (see page 5054)	Creates a dynamic array from a System::Variant (see page 3421).
	DynArrayToVariant (see page 5054)	Creates a System::Variant (see page 3421) array from a dynamic array.
	FindCustomVariantType (see page 5055)	Retrieves the object that implements a custom System::Variant (see page 3421) type.
	FindCustomVariantType (see page 5055)	Retrieves the object that implements a custom System::Variant (see page 3421) type.
	FindVarData (see page 5055)	Returns pointer to variant's data.
	HandleConversionException (see page 5056)	Handles exceptions during variant type conversions.
	Null (see page 5056)	Returns a Null variant.
	Unassigned (see page 5058)	Returns an "empty" variant.
	VarArrayAsPSafeArray (see page 5059)	This is function Variants.VarArrayAsPSafeArray.
	VarArrayCreate (see page 5059)	Creates a variant array.
	VarArrayCreateError (see page 5061)	Indicates an array creation error.
	VarArrayDimCount (see page 5061)	Returns number of dimensions of a variant array.
	VarArrayGet (see page 5063)	Returns a System::Variant (see page 3421) that represents a single value from a multi-dimensional System::Variant (see page 3421) array.
	VarArrayHighBound (see page 5065)	Returns high bound for a dimension in a variant array.
	VarArrayLock (see page 5067)	Locks a variant array and returns a pointer to the data.
	VarArrayLowBound (see page 5067)	Returns the low bound of a dimension in a variant array.
	VarArrayOf (see page 5069)	Creates and fills a one-dimensional variant array.
	VarArrayPut (see page 5069)	Sets the value of a single cell in a multi-dimensional System::Variant (see page 3421) array.
	VarArrayRef (see page 5071)	Returns a reference to the specified variant array.
	VarArrayUnlock (see page 5072)	Unlocks a variant array.
	VarAsError (see page 5072)	Converts a HRESULT (see page 3754) into an Error (see page 3560) variant.
	VarAsType (see page 5073)	Converts a variant to specified type.
	VarCastError (see page 5073)	Raises an EVariantTypeCastError (see page 4986) exception.
	VarCastError (see page 5073)	Raises an EVariantTypeCastError (see page 4986) exception.
	VarCheckEmpty (see page 5074)	Raises an exception if a specified variant's value is Unassigned (see page 5058).
	VarCompareValue (see page 5074)	Returns the relationship between Variants.
	VarCopyNoInd (see page 5075)	Copies a System::Variant (see page 3421).
	VarEnsureRange (see page 5076)	Returns a System::Variant (see page 3421) with a value in a specified range.
	VarFromDateTime (see page 5076)	Returns a variant containing a specified data and time.
	VarInRange (see page 5077)	Indicates whether a System::Variant (see page 3421)'s value is in a specified range.
	VarInvalidNullOp (see page 5077)	Indicate variant operation on null.
	VarInvalidOp (see page 5078)	Indicate invalid variant operation.
	VarIsArray (see page 5078)	Indicates whether the specified variant is an array.

	VarIsArray (see page 5078)	Indicates whether the specified variant is an array.
	VarIsByRef (see page 5079)	Indicates whether the specified variant's value is by reference.
	VarIsClear (see page 5079)	Indicates whether the specified variant has an undefined value.
	VarIsCustom (see page 5080)	Indicates whether the specified variant is a custom variant.
	VarIsEmpty (see page 5080)	Indicates whether the specified variant is unassigned.
	VarIsEmptyParam (see page 5081)	Indicates whether the specified variant represents an unassigned optional parameter.
	VarIsError (see page 5082)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3754).
	VarIsError (see page 5082)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3754).
	VarIsFloat (see page 5082)	Indicates whether the specified variant represents a floating-point value.
	VarIsNull (see page 5083)	Indicates whether the specified variant is Null (see page 5056).
	VarIsNumeric (see page 5083)	Indicates whether the specified variant represents a numeric value.
	VarIsOrdinal (see page 5084)	Indicates whether the specified variant represents an ordinal value.
	VarIsStr (see page 5084)	Indicates whether the specified variant represents a string value.
	VarIsType (see page 5085)	Indicates whether the specified variant represents a specified type.
	VarIsType (see page 5085)	Indicates whether the specified variant represents a specified type.
	VarOverflowError (see page 5085)	Raises an EVariantOverflowError (see page 4985) exception.
	VarRangeCheckError (see page 5086)	Raises an EVariantRangeCheckError (see page 4986) exception.
	VarResultCheck (see page 5086)	Raises variant exception based on HRESULT (see page 3754) value.
	VarResultCheck (see page 5086)	Raises variant exception based on HRESULT (see page 3754) value.
	VarSameValue (see page 5087)	Indicates whether two Variants have equivalent values.
	VarSupports (see page 5087)	Indicates whether the System::Variant (see page 3421)'s value is an interface that supports a specified interface.
	VarSupports (see page 5087)	Indicates whether the System::Variant (see page 3421)'s value is an interface that supports a specified interface.
	VarToDateTime (see page 5088)	Converts a specified variant to a TDateTime (see page 3323) value.
	VarToStr (see page 5088)	Converts a variant's value to a string.
	VarToStrDef (see page 5089)	Converts a variant's value to a string.
	VarToWideStr (see page 5089)	Converts a variant's value to a WideString (see page 3467).
	VarToWideStrDef (see page 5090)	Converts a variant's value to a WideString (see page 3467).
	VarType (see page 5090)	Returns the type code of a specified variant.
	VarTypeAsText (see page 5092)	Returns variant type descriptive string.
	VarTypeIsValidArrayType (see page 5092)	Indicates whether a System::Variant (see page 3421) type code represents a type that can be used for the elements in a System::Variant (see page 3421) array.
	VarTypeIsValidElementType (see page 5093)	Indicates whether a System::Variant (see page 3421) type code represents a type that can be used for the elements in a System::Variant (see page 3421) array.

Interfaces

	Name	Description
	IVarInstanceReference (see page 5022)	IVarInstanceReference is the interface for extracting the instance that represents the data from a Variant (see page 3421) of the custom type.
	IVarInvokeable (see page 5023)	IVarInvokeable is the interface for custom Variant (see page 3421) types that implement dynamic properties and methods.

Structs, Records, Enums

	Name	Description
	TBooleanToStringRule (see page 5094)	TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 5100).
	TNullCompareRule (see page 5094)	TNullCompareRule enumerates possible rules for comparing variant nulls with other values.
	TVarCompareResult (see page 5095)	TVarCompareResult describes the results of comparing two Variants.
	TVariantRelationship (see page 5096)	TVariantRelationship describes the results of comparing two Variants.

Types

Name	Description
TAnyProc (see page 5097)	TAnyProc represents a pointer to a procedure that accepts a parameter of type TVarData (see page 3740).
TCustomVariantTypeClass (see page 5097)	TCustomVariantTypeClass defines the metaclass for TCustomVariantType (see page 4987).
TVarDataArray (see page 5098)	TVarDataArray is an array of TVarData (see page 3740) records.
TVarDispProc (see page 5098)	TVarDispProc defines a pointer to a procedure used as support for IDispatch (see page 3494)-based Automation.

Variables

Name	Description
BooleanToStringRule (see page 5100)	Specifies rule fixing up Boolean values converted to strings.
ChangeAnyProc (see page 5101)	Provides the support for changing the type of a Variant (see page 3421) from varAny (see page 3861) to a real Variant (see page 3421).
ClearAnyProc (see page 5101)	Provides the support for clearing a Variant (see page 3421) of type varAny (see page 3861).
EmptyParam (see page 5102)	Contains an OleVariant (see page 3274) that represents an unused optional parameter on a dual interface.
NullAsStringValue (see page 5102)	Specifies string value of Null (see page 5056) variants.
NullEqualityRule (see page 5103)	Specifies rule for Null (see page 5056) variant equality comparison.
NullMagnitudeRule (see page 5103)	Specifies rule for Null (see page 5056) variant magnitude comparison.
NullStrictConvert (see page 5104)	Specifies whether Null (see page 5056) conversion is an error.
OleVariantInt64AsDouble (see page 5105)	Controls the mode in which `Int64` values are stored in a OleVariant (see page 3274).
PackVarCreation (see page 5105)	Specifies whether space-efficient System::Variant (see page 3421) Types (see page 4649) are preferred.
RefAnyProc (see page 5106)	Provides the support for increasing the reference count of a Variant (see page 3421) of type varAny (see page 3861).
VarDispProc (see page 5106)	Provides the support for IDispatch (see page 3494)-based Automation.

51.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EVariantArrayCreateError (see page 4978)	EVariantArrayCreateError is the exception class for failure to create or resize a variant array.
	EVariantArrayLockedError (see page 4979)	EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.
	EVariantBadIndexError (see page 4980)	EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.
	EVariantBadVarTypeError (see page 4980)	EVariantBadVarTypeError is the exception class for variant type errors.
	EVariantDispatchError (see page 4981)	EVariantDispatchError is the exception class for failed dispatch attempts.
	EVariantInvalidArgError (see page 4981)	EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.
	EVariantInvalidNullOpError (see page 4982)	EVariantInvalidNullOpError is the exception class for invalid Null (see page 5056) variant operations.
	EVariantInvalidOpError (see page 4983)	EVariantInvalidOpError is the exception class for invalid variant operations.
	EVariantNotAnArrayError (see page 4983)	EVariantNotAnArrayError is the exception class inappropriate variant array operations.
	EVariantNotImplError (see page 4984)	EVariantNotImplError is the exception class for unimplemented variant operations.
	EVariantOutOfMemoryError (see page 4984)	EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.
	EVariantOverflowError (see page 4985)	EVariantOverflowError is the exception class for overflow in a variant operation.
	EVariantRangeCheckError (see page 4986)	EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.
	EVariantTypeCastError (see page 4986)	EVariantTypeCastError is the exception class for variant type cast errors.
	EVariantUnexpectedError (see page 4987)	EVariantUnexpectedError is the exception class for unexpected variant errors.
	TCustomVariantType (see page 4987)	TCustomVariantType is the base class for Delphi custom variant types.
	TInvokeableVariantType (see page 5014)	TInvokeableVariantType is the base class for custom System::Variant (see page 3421) types that implement custom Variants (see page 4973) with properties and methods.

51.1.1 EVariantArrayCreateError Class

EVariantArrayCreateError is the exception class for failure to create or resize a variant array.

Class Hierarchy

Pascal

```
EVariantArrayCreateError = class(EVariantError);
```

C++

```
class EVariantArrayCreateError : public EVariantError;
```

File

Variants

Description

EVariantArrayCreateError is raised when an attempt to create or resize a variant array fails. To raise this exception programmatically, call VarArrayCreateError (see page 5061)

See Also

VarArrayCreateError (see page 5061)

TVarData (see page 3740)

VarArrayCreate (see page 5059)

VarArrayOf (see page 5069)

VarArrayRedim (see page 3702)

TCustomVariantType (see page 4987)

51.1.2 EVariantArrayLockedError Class

EVariantArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.

Class Hierarchy

Pascal

```
EVariantArrayLockedError = class(EVariantError);
```

C++

```
class EVariantArrayLockedError : public EVariantError;
```

File

Variants

Description

EVariantArrayLockedError is raised when an operation on a variant array fails because the array is locked or fixed.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

VarAsType (see page 5073)

VarArrayLock (see page 5067)

VarArrayUnlock (see page 5072)

VarArrayOf (see page 5069)

VarArrayRedim (see page 3702)

51.1.3 EVariantBadIndexError Class

EVariantBadIndexError is the exception class for errors that occur because a variant array index is out of bounds.

Class Hierarchy

Pascal

```
EVariantBadIndexError = class(EVariantError);
```

C++

```
class EVariantBadIndexError : public EVariantError;
```

File

Variants

Description

EVariantBadIndexError is raised when an operation on a variant array fails because an index value is out of bounds.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

VarAsType (see page 5073)

VarArrayCreate (see page 5059)

VarArrayHighBound (see page 5065)

VarArrayLowBound (see page 5067)

VarArrayOf (see page 5069)

VarArrayRedim (see page 3702)

TCustomVariantType (see page 4987)

51.1.4 EVariantBadVarTypeError Class

EVariantBadVarTypeError is the exception class for variant type errors.

Class Hierarchy

Pascal

```
EVariantBadVarTypeError = class(EVariantError);
```

C++

```
class EVariantBadVarTypeError : public EVariantError;
```

File

Variants

Description

EVariantBadVarTypeError is raised when a variant operation fails because of a value that isn't a valid variant type.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

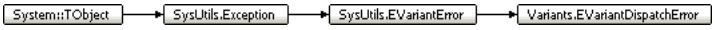
VarAsType (see page 5073)

TCustomVariantType (see page 4987)

51.1.5 EVariantDispatchError Class

EVariantDispatchError is the exception class for failed dispatch attempts.

Class Hierarchy



Pascal

```
EVariantDispatchError = class(EVariantError);
```

C++

```
class EVariantDispatchError : public EVariantError;
```

File

Variants

Description

EVariantDispatchError is raised when a variant type is unable to dispatch a method call. Do not directly instantiate EVariantDispatchError. Instead call the RaiseDispError method of TCustomVariant.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

TCustomVariantType (see page 4987)

51.1.6 EVariantInvalidArgError Class

EVariantInvalidArgError is the exception class for operations that fail due to invalid variant data.

Class Hierarchy

Pascal

```
EVariantInvalidArgError = class(EVariantError);
```

C++

```
class EVariantInvalidArgError : public EVariantError;
```

File

Variants

Description

EVariantInvalidArgError is raised when a variant operation fails because it was passed an inappropriate value.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

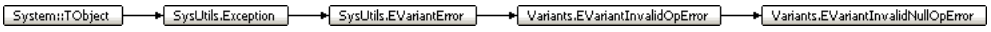
VarAsType (see page 5073)

TCustomVariantType (see page 4987)

51.1.7 EVariantInvalidNullOpError Class

EVariantInvalidNullOpError is the exception class for invalid Null (see page 5056) variant operations.

Class Hierarchy



Pascal

```
EVariantInvalidNullOpError = class(EVariantInvalidOpError);
```

C++

```
class EVariantInvalidNullOpError : public EVariantInvalidOpError;
```

File

Variants

Description

EVariantInvalidNullOpError is raised when a variant operation fails because the operation is not valid for Null (see page 5056) variants. To raise this exception programmatically, call VarInvalidNullOp (see page 5077).

Note: Do not raise EVariantInvalidNullOpError unless NullEqualityRule (see page 5103) or NullMagnitudeRule (see page 5103) is ncrError, and a corresponding operation has been attempted.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

TCustomVariantType (see page 4987)

VarInvalidNullOp (see page 5077)

Null (see page 5056)

NullEqualityRule (see page 5103)

NullMagnitudeRule (see page 5103)

51.1.8 EVariantInvalidOpError Class

EVariantInvalidOpError is the exception class for invalid variant operations.

Class Hierarchy

Pascal

```
EVariantInvalidOpError = class(EVariantError);
```

C++

```
class EVariantInvalidOpError : public EVariantError;
```

File

Variants

Description

EVariantInvalidOpError is raised when a variant operation fails because the operands are incompatible with the specific operation. To raise this exception programmatically, call VarInvalidOp (see page 5078).

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

TCustomVariantType (see page 4987)

VarInvalidOp (see page 5078)

51.1.9 EVariantNotAnArrayError Class

EVariantNotAnArrayError is the exception class inappropriate variant array operations.

Class Hierarchy

Pascal

```
EVariantNotAnArrayError = class(EVariantError);
```

C++

```
class EVariantNotAnArrayError : public EVariantError;
```

File

Variants

Description

EVariantNotAnArrayError is raised when an array operation is attempted on a variant that does not contain an array.

See Also

VarArrayCreateError (see page 5061)

TVarData (see page 3740)

VarArrayCreate (see page 5059)

VarArrayOf (see page 5069)

VarArrayRedim (see page 3702)

TCustomVariantType (see page 4987)

51.1.10 EVariantNotImplError Class

EVariantNotImplError is the exception class for unimplemented variant operations.

Class Hierarchy

Pascal

```
EVariantNotImplError = class(EVariantError);
```

C++

```
class EVariantNotImplError : public EVariantError;
```

File

Variants

Description

EVariantNotImplError is raised when an operation is attempted on a variant type that does not implement that operation.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

TCustomVariantType (see page 4987)

51.1.11 EVariantOutOfMemoryError Class

EVariantOutOfMemoryError is the exception class for variant operations that cannot allocate necessary memory.

Class Hierarchy

Pascal

```
EVariantOutOfMemoryError = class(EVariantError);
```

C++

```
class EVariantOutOfMemoryError : public EVariantError;
```

File

Variants

Description

EVariantOutOfMemoryError is raised when a variant operation fails because of a failure to allocate necessary memory.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

VarArrayCreate (see page 5059)

VarArrayOf (see page 5069)

VarArrayRedim (see page 3702)

VarToDateTime (see page 5088)

TCustomVariantType (see page 4987)

EOutOfMemory (see page 3942)

51.1.12 EVariantOverflowError Class

EVariantOverflowError is the exception class for overflow in a variant operation.

Class Hierarchy

Pascal

```
EVariantOverflowError = class(EVariantError);
```

C++

```
class EVariantOverflowError : public EVariantError;
```

File

Variants

Description

EVariantOverflowError is raised when a variant operation fails because the result was too large. To raise this exception programmatically, call VarOverflowError (see page 5085).

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

VarAsType (see page 5073)

VarToDateTime (see page 5088)

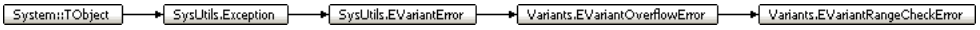
TCustomVariantType (see page 4987)

VarOverflowError (see page 5085)

51.1.13 EVariantRangeCheckError Class

EVariantRangeCheckError is the exception class for out-of-range errors in a variant operation.

Class Hierarchy



Pascal

```
EVariantRangeCheckError = class(EVariantOverflowError);
```

C++

```
class EVariantRangeCheckError : public EVariantOverflowError;
```

File

Variants

Description

EVariantRangeCheckError is raised when a variant operation fails because the result is out-of-range for the destination variant type. To raise this exception programmatically, call VarRangeCheckError (see page 5086).

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

VarAsType (see page 5073)

VarToDateTime (see page 5088)

TCustomVariantType (see page 4987)

VarRangeCheckError (see page 5086)

51.1.14 EVariantTypeCastError Class

EVariantTypeCastError is the exception class for variant type cast errors.

Class Hierarchy

Pascal

```
EVariantTypeCastError = class(EVariantError);
```

C++

```
class EVariantTypeCastError : public EVariantError;
```

File

Variants

Description

EVariantTypeCastError is raised when a variant operation fails because a value could not be cast to the destination variant type.

To raise this exception programmatically, call VarCastError (see page 5073).

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

VarAsType (see page 5073)

VarToDateTime (see page 5088)

TCustomVariantType (see page 4987)

VarOverflowError (see page 5085)

51.1.15 EVariantUnexpectedError Class

EVariantUnexpectedError is the exception class for unexpected variant errors.

Class Hierarchy

Pascal

```
EVariantUnexpectedError = class(EVariantError);
```

C++

```
class EVariantUnexpectedError : public EVariantError;
```

File

Variants

Description

EVariantUnexpectedError is raised when a variant operation generates an unexpected error.

See Also

TVarData (see page 3740)

System::TVariantManager (see page 3745)

TCustomVariantType (see page 4987)

51.1.16 TCustomVariantType Class

TCustomVariantType is the base class for Delphi custom variant types.

Class Hierarchy

Pascal

```
TCustomVariantType = class(TObject, IInterface);
```

C++

```
class TCustomVariantType : public TObject, public IInterface;
```

File

Variants

Description

Use TCustomVariantType to define new data types for Delphi Variants (see page 4973). TCustomVariantType descendants do not represent the Variants (see page 4973) themselves. Rather, they act as utility classes that the custom Variants (see page 4973) call on to perform their operations.

To create a System::Variant (see page 3421) type,

1.Map the storage of the System::Variant (see page 3421)'s data on to the TVarData (see page 3740) record.

2.Write (see page 3707) a method that creates instances of your custom System::Variant (see page 3421). This fills in the System::Variant (see page 3421)'s data as defined in step 1.

3.Declare a class that descends from TCustomVariantType.

4.Implement all required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically BinaryOp (see page 4994), UnaryOp (see page 5006), CompareOp (see page 4998) or Compare (see page 4997), Cast (see page 4995), CastTo (see page 4995), Clear (see page 4996), Copy (see page 4999), IsClear (see page 5000), LeftPromotion (see page 5000), and RightPromotion (see page 5004)).

5.In the initialization section of the unit that defines your TCustomVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the System::Variant (see page 3421)-handling system so that the new System::Variant (see page 3421) type is enabled.

6.In the finalization section of the unit that defines your TCustomVariantType descendant, free the instance of your class. This automatically unregisters the System::Variant (see page 3421) type.

Note: C++ can use Custom System::Variant (see page 3421) types, but TCustomVariantType descendants must be implemented in Delphi.

For an example, see the FMTBcd unit, which implements a custom System::Variant (see page 3421) type for binary-coded decimal.

Tip: To create a custom System::Variant (see page 3421) that allows users to call methods or procedures and to get or set property values, use TInvokeableVariantType (see page 5014) as a base class instead.

See Also

TInvokeableVariantType (see page 5014)

Creating a Class to Enable the Custom Variant Type

51.1.16.1 TCustomVariantType Members

The following tables list the members exposed by TCustomVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 4993)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 4993)	Responds when the caller releases a reference to this object's interface.
V	DispInvoke (see page 4999)	Invokes a method call when the System::Variant (see page 3421) supports such operations.

	LeftPromotion (see page 5000)	Indicates whether the left argument to a binary operation should be coerced to a different type.
	OlePromotion (see page 5001)	Indicates the type to which a System::Variant (see page 3421) value must be cast to make it Automation-compatible.
	QueryInterface (see page 5002)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5002)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5003)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5003)	Generates a system error indicating an invalid operation.
	RightPromotion (see page 5004)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5005)	Clears a TVarData (see page 3740) record by re-initializing it.
	SimplisticCopy (see page 5005)	Performs a simple block copy of one TVarData (see page 3740) record to another.
	VarDataCast (see page 5006)	Performs a cast from one System::Variant (see page 3421) type to another.
	VarDataCastTo (see page 5007)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5007)	Performs a cast to a System::Variant (see page 3421) type of varOleStr (see page 3869).
	VarDataClear (see page 5008)	Clears the data from a built-in System::Variant (see page 3421) type.
	VarDataCopy (see page 5008)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataCopyNoInd (see page 5009)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataFromOleStr (see page 5009)	Creates a TVarData (see page 3740) record to represent a Unicode string.
	VarDataFromStr (see page 5010)	Creates a TVarData (see page 3740) record to represent a string.
	VarDataInit (see page 5010)	Initializes a TVarData (see page 3740) record.
	VarDataIsArray (see page 5010)	Indicates whether a TVarData (see page 3740) record represents the data from a System::Variant (see page 3421) array.
	VarDataIsByRef (see page 5011)	Indicates whether a TVarData (see page 3740) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5011)	Indicates whether a TVarData (see page 3740) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5011)	Indicates whether a TVarData (see page 3740) record represents a floating-point value.
	VarDataIsNumeric (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a numeric value.
	VarDataIsOrdinal (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a ordinal value.
	VarDataIsStr (see page 5013)	Indicates whether a TVarData (see page 3740) record represents a string (normal or Unicode).
	VarDataToStr (see page 5013)	Returns a string that represents the value from a TVarData (see page 3740) record.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	BinaryOp (see page 4994)	Implements any binary operations of which the custom System::Variant (see page 3421) type is capable.
	Cast (see page 4995)	Converts the data from another System::Variant (see page 3421) type so that it represents this custom type's data format.
	CastTo (see page 4995)	Converts data in the format of this TCustomVariantType (see page 4987) to the format of another System::Variant (see page 3421) type.
	CastToOle (see page 4996)	Converts data in the format of this custom System::Variant (see page 3421) type to a format that can be used by COM Variants (see page 4973).
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.

	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 4996)	Sets the data for this custom System::Variant (see page 3421) type to a blank or unassigned value.
V	Compare (see page 4997)	Implements comparisons that involve the custom System::Variant (see page 3421) type.
V	CompareOp (see page 4998)	Implements any comparison operations of which the custom System::Variant (see page 3421) type is capable.
A	Copy (see page 4999)	Copies the data from a System::Variant (see page 3421) of this custom type.
	Create (see page 4990)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 4991)	Frees an instance of TCustomVariantType (see page 4987).
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
V	IsClear (see page 5000)	Indicates whether the data from a custom System::Variant (see page 3421) represents a blank or unassigned value.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
V	UnaryOp (see page 5006)	Implements any unary operations of which the custom System::Variant (see page 3421) type is capable.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	VarType (see page 5014)	Indicates the System::Variant (see page 3421) type code for the custom System::Variant (see page 3421) type.

51.1.16.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TCustomVariantType.Create () (see page 4991)	Creates an instance of TCustomVariantType (see page 4987).
	TCustomVariantType.Create (TVarType) (see page 4991)	Creates an instance of TCustomVariantType (see page 4987).

51.1.16.2.1 TCustomVariantType.Create Constructor ()

Creates an instance of TCustomVariantType (see page 4987).

Pascal

```
constructor Create; overload;
constructor Create(RequestedVarType: TVarType); overload;
```

C++

```
__fastcall TCustomVariantType();
__fastcall TCustomVariantType(TVarType RequestedVarType);
```

Description

Call Create in the initialization section of the unit that defines your TCustomVariantType (see page 4987) descendant to instantiate a single instance of the custom System::Variant (see page 3421) type. This single instance automatically registers itself with the System::Variant (see page 3421) system so that the custom System::Variant (see page 3421) type is enabled in the application.

The RequestedVarType parameter specifies a type code for the new custom System::Variant (see page 3421) type. It must be in the range from $010F to $0FFF.

When called with no parameters, the constructor generates a new type code for the custom System::Variant (see page 3421) type.

The TCustomVariantType (see page 4987) constructor raises an exception if there is a problem with the new System::Variant (see page 3421) type code. Such problems include

The RequestedVarType parameter specifies a type code that is already used by another System::Variant (see page 3421) type.

The RequestedVarType parameter specifies a type code that is not in the valid range for custom Variants (see page 4973).

There are too many custom Variants (see page 4973) already defined to allow another one.

See Also

VarType (see page 5014)

51.1.16.3 TCustomVariantType.Destroy Destructor

Frees an instance of TCustomVariantType (see page 4987).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomVariantType();
```

Description

To free a TCustomVariantType (see page 4987) instance, use the Free method instead. Free checks that the instance is not already nil before calling Destroy.

Free the single instance of a TCustomVariantType (see page 4987) descendant from the finalization section of the unit that implements the class.

See Also

Create (see page 4991)

System::TObject::Free (see page 3372)

51.1.16.4 TCustomVariantType Methods

The methods of the TCustomVariantType class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 4993)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 4993)	Responds when the caller releases a reference to this object's interface.
V	DispInvoke (see page 4999)	Invokes a method call when the System::Variant (see page 3421) supports such operations.
V	LeftPromotion (see page 5000)	Indicates whether the left argument to a binary operation should be coerced to a different type.
V	OlePromotion (see page 5001)	Indicates the type to which a System::Variant (see page 3421) value must be cast to make it Automation-compatible.
V	QueryInterface (see page 5002)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5002)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5003)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5003)	Generates a system error indicating an invalid operation.
V	RightPromotion (see page 5004)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5005)	Clears a TVarData (see page 3740) record by re-initializing it.
	SimplisticCopy (see page 5005)	Performs a simple block copy of one TVarData (see page 3740) record to another.
	VarDataCast (see page 5006)	Performs a cast from one System::Variant (see page 3421) type to another.
	VarDataCastTo (see page 5007)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5007)	Performs a cast to a System::Variant (see page 3421) type of varOleStr (see page 3869).
	VarDataClear (see page 5008)	Clears the data from a built-in System::Variant (see page 3421) type.
	VarDataCopy (see page 5008)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataCopyNoInd (see page 5009)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataFromOleStr (see page 5009)	Creates a TVarData (see page 3740) record to represent a Unicode string.
	VarDataFromStr (see page 5010)	Creates a TVarData (see page 3740) record to represent a string.
	VarDataInit (see page 5010)	Initializes a TVarData (see page 3740) record.
	VarDataIsArray (see page 5010)	Indicates whether a TVarData (see page 3740) record represents the data from a System::Variant (see page 3421) array.
	VarDataIsByRef (see page 5011)	Indicates whether a TVarData (see page 3740) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5011)	Indicates whether a TVarData (see page 3740) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5011)	Indicates whether a TVarData (see page 3740) record represents a floating-point value.
	VarDataIsNumeric (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a numeric value.
	VarDataIsOrdinal (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a ordinal value.
	VarDataIsStr (see page 5013)	Indicates whether a TVarData (see page 3740) record represents a string (normal or Unicode).
	VarDataToStr (see page 5013)	Returns a string that represents the value from a TVarData (see page 3740) record.

Public Methods

	Name	Description
V	BinaryOp (see page 4994)	Implements any binary operations of which the custom System::Variant (see page 3421) type is capable.

	Cast (see page 4995)	Converts the data from another System::Variant (see page 3421) type so that it represents this custom type's data format.
	CastTo (see page 4995)	Converts data in the format of this TCustomVariantType (see page 4987) to the format of another System::Variant (see page 3421) type.
	CastToOle (see page 4996)	Converts data in the format of this custom System::Variant (see page 3421) type to a format that can be used by COM Variants (see page 4973).
	Clear (see page 4996)	Sets the data for this custom System::Variant (see page 3421) type to a blank or unassigned value.
	Compare (see page 4997)	Implements comparisons that involve the custom System::Variant (see page 3421) type.
	CompareOp (see page 4998)	Implements any comparison operations of which the custom System::Variant (see page 3421) type is capable.
	Copy (see page 4999)	Copies the data from a System::Variant (see page 3421) of this custom type.
	IsClear (see page 5000)	Indicates whether the data from a custom System::Variant (see page 3421) represents a blank or unassigned value.
	UnaryOp (see page 5006)	Implements any unary operations of which the custom System::Variant (see page 3421) type is capable.

51.1.16.4.1 TCustomVariantType._AddRef Method

Responds when the caller acquires a reference to this object's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

When an application uses an interface that is implemented by TCustomVariantType (see page 4987) or one of its descendants, it automatically calls __AddRef. Typically, interfaced objects use the __AddRef method to increment a reference count, which is then decremented when the application releases the interface. In this typical scenario, when the reference count drops to zero, the object frees itself.

TCustomVariantType (see page 4987) implements its own version of the IInterface (see page 3502) methods __AddRef and _Release (see page 4993) that avoid this typical behavior. An application should have only a single instance of TCustomVariantType (see page 4987), and that instance should not be auto-freed when it is not in use. Thus, TCustomVariantType (see page 4987) implements __AddRef so that it always returns –1.

See Also

_Release (see page 4993)

_AddRef (see page 3503)

51.1.16.4.2 TCustomVariantType._Release Method

Responds when the caller releases a reference to this object's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

When an application releases an interface that is implemented by TCustomVariantType (see page 4987) or one of its descendants, it automatically calls __Release. Typically, interfaced objects use the __Release method to decrement a reference

count. When the reference count drops to zero, the object is automatically freed.

TCustomVariantType (see page 4987) implements its own version of the IInterface (see page 3502) methods _AddRef (see page 4993) and __Release that avoid this typical behavior. An application should have only a single instance of TCustomVariantType (see page 4987), and that instance should not be auto-freed when it is not in use. Thus, TCustomVariantType (see page 4987) implements __Release so that it always returns –1.

See Also

_AddRef (see page 4993)

_Release (see page 3503)

51.1.16.4.3 TCustomVariantType.BinaryOp Method

Implements any binary operations of which the custom System::Variant (see page 3421) type is capable.

Pascal

```
procedure BinaryOp(var Left: TVarData; const Right: TVarData; const Operator: TVarOp); virtual;
```

C++

```
virtual __fastcall BinaryOp(TVarData Left, const TVarData Right, const TVarOp Operator);
```

Description

Override BinaryOp to implement any binary operators the custom System::Variant (see page 3421) type supports.

Left is the value that appears to the left of the operator. BinaryOp changes this value to indicate the result of the operation.

Right is the value that appears to the right of the operator.

Operator identifies the operator that appears between Left and Right. It can be any of the following values:

Value	Operation
opAdd (see page 3850)	addition
opSubtract (see page 3858)	subtraction
opMultiply (see page 3855)	multiplication
opDivide (see page 3854)	floating-point division
opIntDivide (see page 3855)	integer division
opModulus (see page 3855)	remainder
opShiftLeft (see page 3857)	bitwise left shift
opShiftRight (see page 3858)	bitwise right shift
opAnd (see page 3850)	bitwise and
opOr (see page 3857)	bitwise or
opXor (see page 3859)	bitwise exclusive or

As implemented in TCustomVariantType (see page 4987), BinaryOp raises a system error indicating an invalid operation.

See Also

UnaryOp (see page 5006)

CompareOp (see page 4998)

LeftPromotion (see page 5000)

RightPromotion (see page 5004)

Implementing Binary Operations

51.1.16.4.4 TCustomVariantType.Cast Method

Converts the data from another System::Variant (see page 3421) type so that it represents this custom type's data format.

Pascal

```
procedure Cast(var Dest: TVarData; const Source: TVarData); virtual;
```

C++

```
virtual __fastcall Cast(TVarData Dest, const TVarData Source);
```

Description

Override Cast to enable other types of Variants (see page 4973) to be cast to this custom System::Variant (see page 3421) type.

Dest is a TVarData (see page 3740) record that returns the System::Variant (see page 3421)'s data stored in the format used by this custom System::Variant (see page 3421) type. The Cast method fills out this record from the information in Source.

Source is a TVarData (see page 3740) record that contains the data from another System::Variant (see page 3421) type. Check the VType field of Source to identify how it currently stores its data. Then use the other fields to fetch the data and convert it to the format used by this System::Variant (see page 3421) type.

As implemented in TCustomVariantType (see page 4987), Cast checks whether Source is another custom System::Variant (see page 3421) type. If so, it calls its CastTo (see page 4995) method, to convert it to this System::Variant (see page 3421) type. If not, it raises a system error indicating an invalid cast.

See Also

CastTo (see page 4995)

VarDataCast (see page 5006)

Enabling Casting

51.1.16.4.5 TCustomVariantType.CastTo Method

Converts data in the format of this TCustomVariantType (see page 4987) to the format of another System::Variant (see page 3421) type.

Pascal

```
procedure CastTo(var Dest: TVarData; const Source: TVarData; const AVarType: TVarType);
virtual;
```

C++

```
virtual __fastcall CastTo(TVarData Dest, const TVarData Source, const TVarType AVarType);
```

Description

Override CastTo to enable Variants (see page 4973) of this custom type to be cast to other System::Variant (see page 3421) types.

Dest is a TVarData (see page 3740) record that returns the System::Variant (see page 3421)'s data stored in the format identified by the AVarType parameter. The CastTo method fills out this record.

Source is the data from a System::Variant (see page 3421) of this custom System::Variant (see page 3421) type. CastTo

extracts the data from this record, converts it to the format expected by AVarType, and fills out Dest. If the VType field of source is not the same as the VarType (see page 5014) property, then CastTo creates a new instance of the associated System::Variant (see page 3421) data and casts it to the destination type.

AVarType is the type to which the data should be converted.

See Also

Cast (see page 4995)

VarType (see page 5014)

VarDataCastTo (see page 5007)

51.1.16.4.6 TCustomVariantType.CastToOle Method

Converts data in the format of this custom System::Variant (see page 3421) type to a format that can be used by COM Variants (see page 4973).

Pascal

```
procedure CastToOle(var Dest: TVarData; const Source: TVarData); virtual;
```

C++

```
virtual __fastcall CastToOle(TVarData Dest, const TVarData Source);
```

Description

Override CastToOle to change the way this custom type is converted for use in COM interface calls. CastToOle converts this custom System::Variant (see page 3421) type to the closest Automation-compatible System::Variant (see page 3421) type.

Dest is a TVarData (see page 3740) record that returns the System::Variant (see page 3421)'s data stored in an Automation-compatible format.

Source is a TVarData (see page 3740) record that represents the data from a System::Variant (see page 3421) of this custom type.

As implemented in TCustomVariantType (see page 4987), CastToOle calls the OlePromotion (see page 5001) method to determine the Automation-compatible type to which Source should be converted. It then calls the CastTo (see page 4995) method to perform the actual conversion.

See Also

CastTo (see page 4995)

OlePromotion (see page 5001)

VarDataCastToOleStr (see page 5007)

51.1.16.4.7 TCustomVariantType.Clear Method

Sets the data for this custom System::Variant (see page 3421) type to a blank or unassigned value.

Pascal

```
procedure Clear(var V: TVarData); virtual; abstract;
```

C++

```
virtual __fastcall Clear(TVarData V) = 0;
```

Description

Override Clear to implement the way this custom System::Variant (see page 3421) type represents a blank or unassigned

value.

V is a TVarData (see page 3740) record that represents the data from a System::Variant (see page 3421) of this custom type that needs to be cleared.

In TCustomVariantType (see page 4987), Clear is an abstract method. Descendant classes must override this method to provide an implementation. If the custom System::Variant (see page 3421) type stores its data in a manner that includes memory that needs to be freed, Clear should free that memory. Once any memory is freed, Clear can call the SimplisticClear (see page 5005) method, which re-initializes the TVarData (see page 3740) record, or set the VType field to varEmpty (see page 3865).

See Also

Unassigned (see page 5058)

IsClear (see page 5000)

SimplisticClear (see page 5005)

VarDataClear (see page 5008)

Copying and Clearing Custom Variants

51.1.16.4.8 TCustomVariantType.Compare Method

Implements comparisons that involve the custom System::Variant (see page 3421) type.

Pascal

```
procedure Compare(const Left: TVarData; const Right: TVarData; var Relationship:
TVarCompareResult); virtual;
```

C++

```
virtual __fastcall Compare(const TVarData Left, const TVarData Right, TVarCompareResult
Relationship);
```

Description

Override Compare to implement any comparison operators the custom System::Variant (see page 3421) type supports. Descendants can implement comparison operators by overriding either the Compare or the CompareOp (see page 4998) method.

Left is the value that appears to the left of the operator.

Right is the value that appears to the right of the operator.

Relationship returns a value indicating the relationship between Left and Right, as indicated in the following table:

Value	Operation
crLessThan	Left is less than Right.
crEqual	Left is equal to Right.
crGreaterThan	Left is greater than Right.

As implemented in TCustomVariantType (see page 4987), CompareOp (see page 4998) raises a system error indicating an invalid operation.

Tip: For custom Variants (see page 4973) that support a test of inequality, but not ordering, it makes more sense to override the CompareOp (see page 4998) method instead.

See Also

CompareOp (see page 4998)

CompareValue

Implementing Comparison Operations

51.1.16.4.9 TCustomVariantType.CompareOp Method

Implements any comparison operations of which the custom System::Variant (see page 3421) type is capable.

Pascal

```
function CompareOp(const Left: TVarData; const Right: TVarData; const Operator: TVarOp):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CompareOp(const TVarData Left, const TVarData Right, const TVarOp
Operator);
```

Description

Override CompareOp to implement any comparison operators the custom System::Variant (see page 3421) type supports. Descendants can implement comparison operators by overriding either the Compare (see page 4997) or the CompareOp method.

Left is the value that appears to the left of the operator.

Right is the value that appears to the right of the operator.

Operator identifies the operator that appears between Left and Right. It can be any of the following values:

Value	Operation
opCmpEQ (see page 3851)	equality comparison
opCmpNE (see page 3853)	inequality comparison
opCmpLT (see page 3853)	less than
opCmpLE (see page 3852)	less than or equal
opCmpGT (see page 3852)	greater than
opCmpGE (see page 3851)	greater than or equal

CompareOp returns true if the comparison is true as specified. It returns false if the specified comparison does not hold true.

As implemented in TCustomVariantType (see page 4987), CompareOp raises a system error indicating an invalid operation.

See Also

UnaryOp (see page 5006)

BinaryOp (see page 4994)

LeftPromotion (see page 5000)

RightPromotion (see page 5004)

Compare (see page 4997)

Implementing Comparison Operations

51.1.16.4.10 TCustomVariantType.Copy Method

Copies the data from a System::Variant (see page 3421) of this custom type.

Pascal

```
procedure Copy(var Dest: TVarData; const Source: TVarData; const Indirect: Boolean); virtual;
abstract;
```

C++

```
virtual __fastcall Copy(TVarData Dest, const TVarData Source, const Boolean Indirect) = 0;
```

Description

Override Copy to enable the custom System::Variant (see page 3421) type to be copied (assigned) to another System::Variant (see page 3421).

Dest is the TVarData (see page 3740) record from the System::Variant (see page 3421) whose value is to be a copy of the source System::Variant (see page 3421).

Source is the TVarData (see page 3740) record from a System::Variant (see page 3421) of this custom type that is to be copied to Dest.

Indirect indicates whether the copy is direct or indirect. When Indirect is true, the source TVarData (see page 3740) record contains an indirect reference to its data (the VarDataIsByRef (see page 5011) method should return true for Source). When making an indirect copy, it is not necessary to reallocate records or objects, because the System::Variant (see page 3421) does not "own" the data, it merely points to it. When making a direct copy, Copy must allocate new memory for any allocated records or objects, so that Dest can "own" its own copy of the data.

In TCustomVariantType (see page 4987), Copy is an abstract method. Descendant classes must override this method to provide an implementation.

See Also

VarDataCopy (see page 5008)

VarDataCopyNoInd (see page 5009)

SimplisticCopy (see page 5005)

VarDataIsByRef (see page 5011)

Copying and Clearing Custom Variants

51.1.16.4.11 TCustomVariantType.DispInvoke Method

Invokes a method call when the System::Variant (see page 3421) supports such operations.

Pascal

```
procedure DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); virtual;
```

C++

```
virtual __fastcall DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void *
Params);
```

Description

When the application tries to call a method, set a property, or read a property using a System::Variant (see page 3421) of the custom System::Variant (see page 3421) type, the system automatically generates a call to DispInvoke. TCustomVariantType (see page 4987) responds by generating a system error indicating an invalid dispatch attempt.

To create a custom System::Variant (see page 3421) type that allows applications to call methods or use properties, use TInvokeableVariantType (see page 5014) as a base class instead of TCustomVariantType (see page 4987). TInvokeableVariantType (see page 5014) introduces methods for executing method calls and implementing properties, and overrides the DispInvoke method so that it dispatches method calls to the appropriate methods.

Dest is a TVarData (see page 3740) record that receives the return value of any function calls.

Source is the TVarData (see page 3740) record that holds the data from the custom System::Variant (see page 3421) whose property or method was invoked.

CallDesc identifies the method that was called, including the type and number of its arguments.

Params points to an array of parameter values that were passed in the method call.

See Also

TInvokeableVariantType (see page 5014)

51.1.16.4.12 TCustomVariantType.IsClear Method

Indicates whether the data from a custom System::Variant (see page 3421) represents a blank or unassigned value.

Pascal

```
function IsClear(const V: TVarData): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsClear(const TVarData V);
```

Description

Override IsClear to identify when data from a System::Variant (see page 3421) of the custom System::Variant (see page 3421) type represents a blank or unassigned value. The global VarIsClear (see page 5079) method calls IsClear when given a custom System::Variant (see page 3421) of this type.

As Implemented in TCustomVariantType (see page 4987), IsClear always returns false. Override this method to return true for invalid values or special values set by the Clear (see page 4996) method.

See Also

Clear (see page 4996)

VarIsClear (see page 5079)

Copying and Clearing Custom Variants

51.1.16.4.13 TCustomVariantType.LeftPromotion Method

Indicates whether the left argument to a binary operation should be coerced to a different type.

Pascal

```
function LeftPromotion(const V: TVarData; const Operator: TVarOp; out RequiredVarType:
TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean LeftPromotion(const TVarData V, const TVarOp Operator, TVarType
RequiredVarType);
```

Description

Override LeftPromotion to indicate when the left argument for a binary or comparison operation should be implicitly cast to another type before performing the operation. LeftPromotion is called when the right-hand System::Variant (see page 3421) in

a binary or comparison operation is of this System::Variant (see page 3421) type and

The left-hand argument is a built-in System::Variant (see page 3421) type.

The left-hand argument is a custom System::Variant (see page 3421) type that does not indicate any problem with this System::Variant (see page 3421) type for the right-hand argument.

V is the TVarData (see page 3740) record for the left-hand argument of the operation.

Operator indicates the type of operation. It can be any of the operators in the following table:

Value	Operation
opAdd (see page 3850)	addition
opSubtract (see page 3858)	subtraction
opMultiply (see page 3855)	multiplication
opDivide (see page 3854)	floating-point division
opIntDivide (see page 3855)	integer division
opModulus (see page 3855)	remainder
opShiftLeft (see page 3857)	left shift
opShiftRight (see page 3858)	right shift
opAnd (see page 3850)	bitwise and
opOr (see page 3857)	bitwise or
opXor (see page 3859)	bitwise exclusive or
opCompare (see page 3854)	any comparison operation

RequiredVarType returns the System::Variant (see page 3421) type code for the type to which the left-hand argument should be cast before performing the operation. If the operation can proceed with the left-hand argument left as-is, RequiredVarType returns the VType field of V.

LeftPromotion returns true if the TCustomVariantType (see page 4987) descendant can perform the operation (using the BinaryOp (see page 4994), CompareOp (see page 4998), or Compare (see page 4997) method) assuming the left-hand argument can be cast to RequiredVarType. It returns false if the TCustomVariantType (see page 4987) descendant can't perform the operation.

As implemented in TCustomVariantType (see page 4987), LeftPromotion returns the VarType (see page 5014) property as RequiredVarType, and returns true.

See Also

BinaryOp (see page 4994)

CompareOp (see page 4998)

Compare (see page 4997)

VarType (see page 5014)

RightPromotion (see page 5004)

Implementing Binary Operations

51.1.16.4.14 TCustomVariantType.OlePromotion Method

Indicates the type to which a System::Variant (see page 3421) value must be cast to make it Automation-compatible.

Pascal

```
function OlePromotion(const V: TVarData; out RequiredVarType: TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean OlePromotion(const TVarData V, TVarType RequiredVarType);
```

Description

Override OlePromotion to indicate when a System::Variant (see page 3421) of this custom System::Variant (see page 3421) type must be cast to another type before it can be auto-marshaled using an Automation interface.

V is the TVarData (see page 3740) record for the custom System::Variant (see page 3421) that the application wants to marshal.

RequiredVarType returns the type to which V must be cast before it is Automation-compatible.

OlePromotion returns true if V can be cast to an Automation-compatible type, false if there is no type to which V can be cast.

As implemented in TCustomVariantType (see page 4987), OlePromotion sets RequiredVarType to varOleStr (see page 3869) and returns true if V does not already represent a Unicode string.

See Also

CastToOle (see page 4996)

51.1.16.4.15 TCustomVariantType.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the TCustomVariantType (see page 4987) object supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 4993)

_Release (see page 4993)

51.1.16.4.16 TCustomVariantType.RaiseCastError Method

Generates a system error that indicates an invalid cast.

Pascal

```
procedure RaiseCastError;
```

C++

```
__fastcall RaiseCastError();
```

Description

Call RaiseCastError from the implementation of a TCustomVariantType (see page 4987) descendant when an operation fails because a System::Variant (see page 3421) value could not be cast to a necessary type.

See Also

Cast (see page 4995)

CastTo (see page 4995)

51.1.16.4.17 TCustomVariantType.RaiseDispError Method

Generates a system error indicating an invalid dispatch attempt.

Pascal

```
procedure RaiseDispError;
```

C++

```
__fastcall RaiseDispError();
```

Description

Call RaiseDispError from the implementation of a TCustomVariantType (see page 4987) descendant when an operation fails because the variant can't execute a method call. This raises EVariantDispatchError (see page 4981).

See Also

DispInvoke (see page 4999)

EVariantDispatchError (see page 4981)

51.1.16.4.18 TCustomVariantType.RaiseInvalidOp Method

Generates a system error indicating an invalid operation.

Pascal

```
procedure RaiseInvalidOp;
```

C++

```
__fastcall RaiseInvalidOp();
```

Description

Call RaiseInvalidOp from the implementation of a TCustomVariantType (see page 4987) descendant when it can't perform an operation.

See Also

BinaryOp (see page 4994)

UnaryOp (see page 5006)

CompareOp (see page 4998)

Compare (see page 4997)

51.1.16.4.19 TCustomVariantType.RightPromotion Method

Indicates whether the right argument to a binary operation should be coerced to a different type.

Pascal

```
function RightPromotion(const V: TVarData; const Operator: TVarOp; out RequiredVarType:
TVarType): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean RightPromotion(const TVarData V, const TVarOp Operator, TVarType
RequiredVarType);
```

Description

Override RightPromotion to indicate when the right-hand argument for a binary or comparison operation should be implicitly cast to another type before performing the operation. RightPromotion is called when the right-hand System::Variant (see page 3421) in a binary or comparison operation is of this System::Variant (see page 3421) type.

Note: In operations that involve two custom Variants (see page 4973), the left-hand argument's RightPromotion method is called first. The right-hand argument's LeftPromotion (see page 5000) method is then called only if the left-hand argument's RightPromotion method returns false.

V is the TVarData (see page 3740) record for the left-hand argument of the operation.

Operator indicates the type of operation. It can be any of the operators in the following table:

Value	Operation
opAdd (see page 3850)	addition
opSubtract (see page 3858)	subtraction
opMultiply (see page 3855)	multiplication
opDivide (see page 3854)	floating-point division
opIntDivide (see page 3855)	integer division
opModulus (see page 3855)	remainder
opShiftRight (see page 3858)	right shift
opShiftRight (see page 3858)	right shift
opAnd (see page 3850)	bitwise and
opOr (see page 3857)	bitwise or
opXor (see page 3859)	bitwise exclusive or
opCompare (see page 3854)	any comparison operation

RequiredVarType returns the System::Variant (see page 3421) type code for the type to which the right-hand argument should be cast before performing the operation. If the operation can proceed with the right-hand argument left as-is, RequiredVarType returns the VType field of V.

RightPromotion returns true if the TCustomVariantType (see page 4987) descendant can perform the operation (using the BinaryOp (see page 4994), CompareOp (see page 4998), or Compare (see page 4997) method) assuming the right-hand argument can be cast to RequiredVarType. It returns false if the TCustomVariantType (see page 4987) descendant can't perform the operation.

As implemented in TCustomVariantType (see page 4987), RightPromotion returns the VarType (see page 5014) property as RequiredVarType, and returns true.

See Also

BinaryOp (see page 4994)

CompareOp (see page 4998)

Compare (see page 4997)

VarType (see page 5014)

LeftPromotion (see page 5000)

Implementing Binary Operations

51.1.16.4.20 TCustomVariantType.SimplisticClear Method

Clears a TVarData (see page 3740) record by re-initializing it.

Pascal

```
procedure SimplisticClear(var V: TVarData);
```

C++

```
__fastcall SimplisticClear(TVarData V);
```

Description

Call SimplisticClear from the implementation of a TCustomVariantType (see page 4987) descendant to clear the TVarData (see page 3740) record of a System::Variant (see page 3421) of the custom type. This method should only be called if the TVarData (see page 3740) record does not contain any members that must be freed.

SimplisticClear simply calls the VarDataInit (see page 5010) method, which reinitializes the TVarData (see page 3740) record passed as V.

See Also

Clear (see page 4996)

VarDataClear (see page 5008)

VarDataInit (see page 5010)

51.1.16.4.21 TCustomVariantType.SimplisticCopy Method

Performs a simple block copy of one TVarData (see page 3740) record to another.

Pascal

```
procedure SimplisticCopy(var Dest: TVarData; const Source: TVarData; const Indirect: Boolean =
False);
```

C++

```
__fastcall SimplisticCopy(TVarData Dest, const TVarData Source, const Boolean Indirect =
False);
```

Description

Call SimplisticCopy from the implementation of a TCustomVariantType (see page 4987) descendant to block copy one TVarData (see page 3740) record to another.

Dest is the TVarData (see page 3740) record that receives the copy.

Source is the TVarData (see page 3740) record to copy.

Indirect indicates whether Source represents the data (false) or merely a reference to the data (true).

See Also

Copy (see page 4999)

VarDataCopy (see page 5008)

VarDataCopyNoInd (see page 5009)

Copying and Clearing Custom Variants

51.1.16.4.22 TCustomVariantType.UnaryOp Method

Implements any unary operations of which the custom System::Variant (see page 3421) type is capable.

Pascal

```
procedure UnaryOp(var Right: TVarData; const Operator: TVarOp); virtual;
```

C++

```
virtual __fastcall UnaryOp(TVarData Right, const TVarOp Operator);
```

Description

Override UnaryOp to implement any unary operators the custom System::Variant (see page 3421) type supports.

Right is the value that appears to the right of the operator. UnaryOp changes this value to indicate the result of the operation.

Operator identifies the operator that appears before Right. It can be any of the following values:

Value	Operation
opNegate (see page 3856)	unary minus
opNot (see page 3856)	logical negation

As implemented in TCustomVariantType (see page 4987), UnaryOp raises a system error indicating an invalid operation.

See Also

BinaryOp (see page 4994)

CompareOp (see page 4998)

Implementing Unary Operations

51.1.16.4.23 TCustomVariantType.VarDataCast Method

Performs a cast from one System::Variant (see page 3421) type to another.

Pascal

```
procedure VarDataCast(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCast(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCast from the implementation of a TCustomVariantType (see page 4987) descendant to perform casts on the data from another System::Variant (see page 3421) type to the type specified by the VarType (see page 5014) property. Do not use VarDataCast as the entire implementation of the Cast (see page 4995) method, as this will lead to a circular reference.

Dest returns the result of the cast.

Source is the TVarData (see page 3740) record of another System::Variant (see page 3421) type that is to be converted.

See Also

Cast (see page 4995)

CastTo (see page 4995)

VarDataCastTo (see page 5007)

VarType (see page 5014)

51.1.16.4.24 VarDataCastTo Method

This is the overview for the VarDataCastTo method overload.

Overload List

	Name	Description
	TCustomVariantType.VarDataCastTo (TVarData, TVarData, TVarType) (see page 5007)	Performs a cast from one System::Variant (see page 3421) type to another.
	TCustomVariantType.VarDataCastTo (TVarData, TVarType) (see page 5007)	Performs a cast from one System::Variant (see page 3421) type to another.

51.1.16.4.24.1 TCustomVariantType.VarDataCastTo Method (TVarData, TVarData, TVarType)

Performs a cast from one System::Variant (see page 3421) type to another.

Pascal

```
procedure VarDataCastTo(var Dest: TVarData; const Source: TVarData; const AVarType: TVarType);
overload;
procedure VarDataCastTo(var Dest: TVarData; const AVarType: TVarType); overload;
```

C++

```
__fastcall VarDataCastTo(TVarData Dest, const TVarData Source, const TVarType AVarType);
__fastcall VarDataCastTo(TVarData Dest, const TVarType AVarType);
```

Description

Call VarDataCastTo from the implementation of a TCustomVariantType (see page 4987) descendant to perform casts on the data from one System::Variant (see page 3421) type to another. Do not use VarDataCastTo as the entire implementation of the CastTo (see page 4995) method, as this will lead to a circular reference.

Dest returns the result of the cast. If VarDataCastTo is called with no Source parameter, it also provides the source data to be cast.

Source is the TVarData (see page 3740) record to be cast.

VarType (see page 5014) is the type code of the System::Variant (see page 3421) type to which the data should be cast.

See Also

CastTo (see page 4995)

51.1.16.4.25 TCustomVariantType.VarDataCastToOleStr Method

Performs a cast to a System::Variant (see page 3421) type of varOleStr (see page 3869).

Pascal

```
procedure VarDataCastToOleStr(var Dest: TVarData);
```

C++

```
__fastcall VarDataCastToOleStr(TVarData Dest);
```

Description

Call VarDataCastToOleStr from the implementation of a TCustomVariantType (see page 4987) descendant to perform casts to a TVarData (see page 3740) record with System::Variant (see page 3421) type code of varOleStr (see page 3869) (Unicode string).

Dest is the TVarData (see page 3740) record of a System::Variant (see page 3421), and returns the result of the cast.

See Also

VarDataFromOleStr (see page 5009)

VarDataToStr (see page 5013)

51.1.16.4.26 TCustomVariantType.VarDataClear Method

Clears the data from a built-in System::Variant (see page 3421) type.

Pascal

```
procedure VarDataClear(var Dest: TVarData);
```

C++

```
__fastcall VarDataClear(TVarData Dest);
```

Description

Call VarDataClear from the implementation of a TCustomVariantType (see page 4987) descendant to clear the data in the TVarData (see page 3740) record of a System::Variant (see page 3421). Do not use VarDataClear as the entire implementation of the Clear (see page 4996) method, as this will lead to a circular reference.

Dest is the TVarData (see page 3740) record of a System::Variant (see page 3421) type, and returns the result after clearing the data.

See Also

SimplisticClear (see page 5005)

Clear (see page 4996)

51.1.16.4.27 TCustomVariantType.VarDataCopy Method

Copies the data from one built-in System::Variant (see page 3421) type to another.

Pascal

```
procedure VarDataCopy(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCopy(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCopy from the implementation of a TCustomVariantType (see page 4987) descendant to clear the data from a destination System::Variant (see page 3421) and then copy the TVarData (see page 3740) record from another System::Variant (see page 3421). VarDataCopy can handle data that is stored indirectly (when the VType field includes

varByRef (see page 3862)). Do not use VarDataCopy as the entire implementation of the Copy (see page 4999) method, as this will lead to a circular reference.

Dest is the TVarData (see page 3740) record that receives the copy.

Source is the TVarData (see page 3740) record to be copied.

See Also

SimplisticCopy (see page 5005)

VarDataCopyNoInd (see page 5009)

Copy (see page 4999)

51.1.16.4.28 TCustomVariantType.VarDataCopyNoInd Method

Copies the data from one built-in System::Variant (see page 3421) type to another.

Pascal

```
procedure VarDataCopyNoInd(var Dest: TVarData; const Source: TVarData);
```

C++

```
__fastcall VarDataCopyNoInd(TVarData Dest, const TVarData Source);
```

Description

Call VarDataCopyNoInd from the implementation of a TCustomVariantType (see page 4987) descendant to clear the data from a destination System::Variant (see page 3421) and then copy the TVarData (see page 3740) record from another System::Variant (see page 3421). Unlike the VarDataCopy (see page 5008) method, it can't handle data that is stored indirectly (when the VType field includes varByRef (see page 3862)). However, it is often more efficient than VarDataCopy (see page 5008).

Dest is the TVarData (see page 3740) record that receives the copy.

Source is the TVarData (see page 3740) record to be copied.

See Also

VarDataCopy (see page 5008)

Copy (see page 4999)

51.1.16.4.29 TCustomVariantType.VarDataFromOleStr Method

Creates a TVarData (see page 3740) record to represent a Unicode string.

Pascal

```
procedure VarDataFromOleStr(var V: TVarData; const Value: WideString);
```

C++

```
__fastcall VarDataFromOleStr(TVarData V, const BSTR Value);
```

Description

Call VarDataCopy (see page 5008) from the implementation of a TCustomVariantType (see page 4987) descendant to fill out a TVarData (see page 3740) record so that it represents a specified Unicode string.

V is the TVarData (see page 3740) record to be filled out.

Value is a Unicode string that becomes the value of V.

See Also

VarDataCastToOleStr (see page 5007)

VarDataFromStr (see page 5010)

51.1.16.4.30 TCustomVariantType.VarDataFromStr Method

Creates a TVarData (see page 3740) record to represent a string.

Pascal

```
procedure VarDataFromStr(var V: TVarData; const Value: string);
```

C++

```
__fastcall VarDataFromStr(TVarData V, const UnicodeString Value);
```

Description

Call VarDataCopy (see page 5008) from the implementation of a TCustomVariantType (see page 4987) descendant to fill out a TVarData (see page 3740) record so that it represents a specified string.

V is the TVarData (see page 3740) record to be filled out.

Value is a string that becomes the value of V.

See Also

VarDataFromOleStr (see page 5009)

VarDataIsStr (see page 5013)

VarDataToStr (see page 5013)

51.1.16.4.31 TCustomVariantType.VarDataInit Method

Initializes a TVarData (see page 3740) record.

Pascal

```
procedure VarDataInit(var Dest: TVarData);
```

C++

```
__fastcall VarDataInit(TVarData Dest);
```

Description

Call VarDataInit from the implementation of a TCustomVariantType (see page 4987) descendant to initialize a newly-allocated TVarData (see page 3740) record.

See Also

VarDataClear (see page 5008)

51.1.16.4.32 TCustomVariantType.VarDataIsArray Method

Indicates whether a TVarData (see page 3740) record represents the data from a System::Variant (see page 3421) array.

Pascal

```
function VarDataIsArray(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsArray(const TVarData V);
```

Description

VarDataIsArray returns true if the VType field of V contains the varArray (see page 3861) bit.

See Also

VarDataIsByRef (see page 5011)

VarType (see page 5090)

51.1.16.4.33 TCustomVariantType.VarDataIsByRef Method

Indicates whether a TVarData (see page 3740) record stores its data indirectly.

Pascal

```
function VarDataIsByRef(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsByRef(const TVarData V);
```

Description

VarDataIsArray (see page 5010) returns true if the VType field of V contains the varByRef (see page 3862) bit.

See Also

VarDataIsArray (see page 5010)

VarType (see page 5090)

51.1.16.4.34 TCustomVariantType.VarDataIsEmptyParam Method

Indicates whether a TVarData (see page 3740) record represents an unassigned optional parameter.

Pascal

```
function VarDataIsEmptyParam(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsEmptyParam(const TVarData V);
```

Description

VarDataIsEmptyParam returns true if the given TVarData (see page 3740) record represents an unassigned optional parameter.

See Also

VarDataIsArray (see page 5010)

VarType (see page 5090)

51.1.16.4.35 TCustomVariantType.VarDataIsFloat Method

Indicates whether a TVarData (see page 3740) record represents a floating-point value.

Pascal

```
function VarDataIsFloat(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsFloat(const TVarData V);
```

Description

VarDataIsFloat returns true if the given TVarData (see page 3740) record comes from a built-in System::Variant (see page 3421) type that represents a floating-point value (Single, Double, or Currency (see page 3235)). VarDataIsFloat does not return true for data from Custom System::Variant (see page 3421) types.

See Also

VarDataIsNumeric (see page 5012)

VarType (see page 5090)

VarDataIsOrdinal (see page 5012)

VarDataIsStr (see page 5013)

51.1.16.4.36 TCustomVariantType.VarDataIsNumeric Method

Indicates whether a TVarData (see page 3740) record represents a numeric value.

Pascal

```
function VarDataIsNumeric(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsNumeric(const TVarData V);
```

Description

VarDataIsNumeric returns true if the given TVarData (see page 3740) record represents a numeric value of some sort. This can be either a floating-point value or an ordinal value. VarDataIsNumeric does not return true for data from Custom System::Variant (see page 3421) types.

See Also

VarDataIsFloat (see page 5011)

VarType (see page 5090)

VarDataIsOrdinal (see page 5012)

VarDataIsStr (see page 5013)

51.1.16.4.37 TCustomVariantType.VarDataIsOrdinal Method

Indicates whether a TVarData (see page 3740) record represents a ordinal value.

Pascal

```
function VarDataIsOrdinal(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsOrdinal(const TVarData V);
```

Description

VarDataIsOrdinal returns true if the given TVarData (see page 3740) record comes from a built-in System::Variant (see page 3421) type that represents a ordinal value (such as an integer or Boolean type). VarDataIsOrdinal does not return true for data from Custom System::Variant (see page 3421) types.

See Also

VarDataIsNumeric (see page 5012)

VarType (see page 5090)

VarDataIsFloat (see page 5011)

VarDataIsStr (see page 5013)

51.1.16.4.38 TCustomVariantType.VarDataIsStr Method

Indicates whether a TVarData (see page 3740) record represents a string (normal or Unicode).

Pascal

```
function VarDataIsStr(const V: TVarData): Boolean;
```

C++

```
__fastcall Boolean VarDataIsStr(const TVarData V);
```

Description

VarDataIsStr returns true if the given TVarData (see page 3740) record has a VType field of varString (see page 3871) or varOleStr (see page 3869).

See Also

VarDataIsNumeric (see page 5012)

VarType (see page 5090)

VarDataIsFloat (see page 5011)

51.1.16.4.39 TCustomVariantType.VarDataToStr Method

Returns a string that represents the value from a TVarData (see page 3740) record.

Pascal

```
function VarDataToStr(const V: TVarData): string;
```

C++

```
__fastcall UnicodeString VarDataToStr(const TVarData V);
```

Description

Call VarDataToStr from the implementation of a TCustomVariantType (see page 4987) descendant to extract a string that represents the value of a System::Variant (see page 3421).

V is the TVarData (see page 3740) record that represents the value of the System::Variant (see page 3421).

See Also

VarDataCastToOleStr (see page 5007)

VarDataFromStr (see page 5010)

51.1.16.5 TCustomVariantType Properties

The properties of the TCustomVariantType class are listed here.

Public Properties

	Name	Description
	VarType (see page 5014)	Indicates the System::Variant (see page 3421) type code for the custom System::Variant (see page 3421) type.

51.1.16.5.1 TCustomVariantType.VarType Property

Indicates the System::Variant (see page 3421) type code for the custom System::Variant (see page 3421) type.

Pascal

```
property VarType: TVarType;
```

C++

```
__property TVarType VarType;
```

Description

The VarType property provides read-only access to the type code that identifies a custom System::Variant (see page 3421) data type.

The value of VarType is set in the constructor. This can be an automatically generated value, or it can be a value explicitly passed to the constructor. Note that if the VarType is automatically-generated, it may not remain the same for every application that uses the custom System::Variant (see page 3421) type: its value will depend on the other custom Variants (see page 4973) also used in the application.

See Also

TVarData (see page 3740)

Create (see page 4991)

VarType (see page 5090)

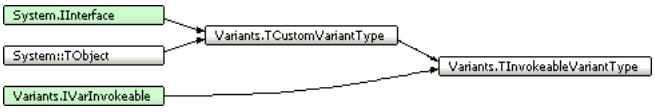
FindCustomVariantType (see page 5055)

Enabling Casting

51.1.17 TInvokeableVariantType Class

TInvokeableVariantType is the base class for custom System::Variant (see page 3421) types that implement custom Variants (see page 4973) with properties and methods.

Class Hierarchy



Pascal

```
TInvokeableVariantType = class(TCustomVariantType, IVarInvokeable);
```

C++

```
class TInvokeableVariantType : public TCustomVariantType, public IVarInvokeable;
```

File

Variants

Description

Use TInvokeableVariantType to define new data types for Delphi Variants (see page 4973) with properties and methods. TInvokeableVariantType descendants do not represent the Variants (see page 4973) themselves. Rather, they act as utility classes that the custom Variants (see page 4973) call on to perform their operations.

Note: Do not define TInvokeableVariantType descendants in C++. The logic for custom System::Variant (see page 3421) types must be defined in Delphi code.

To create a System::Variant (see page 3421) type that supports properties and methods,

1. Map the storage of the System::Variant (see page 3421)'s data on to the TVarData (see page 3740) record.
2. Write (see page 3707) a method that creates instances of your custom System::Variant (see page 3421). This fills in the System::Variant (see page 3421)'s data as defined in step 1.
3. Declare a class that descends from TInvokeableVariantType.
4. Implement all required behavior (including type conversion rules) in the new class. This involves overriding the appropriate virtual methods (typically DoFunction (see page 5018), DoProcedure (see page 5019), GetProperty (see page 5020), SetProperty (see page 5021), BinaryOp (see page 4994), UnaryOp (see page 5006), CompareOp (see page 4998) or Compare (see page 4997), Cast (see page 4995), CastTo (see page 4995), Clear (see page 4996), Copy (see page 4999), IsClear (see page 5000), LeftPromotion (see page 5000), and RightPromotion (see page 5004)).
5. In the initialization section of the unit that defines your TInvokeableVariantType descendant, create an instance of your class. When you instantiate your object, it automatically registers itself with the System::Variant (see page 3421)-handling system so that the new System::Variant (see page 3421) type is enabled.
6. In the finalization section of the unit that defines your TInvokeableVariantType descendant, free the instance of your class. This automatically unregisters the System::Variant (see page 3421) type.

For an example, see the VarConv (see page 4947) unit, which implements a custom System::Variant (see page 3421) type for measurements.

Tip: If the custom System::Variant (see page 3421) type uses an object instance to store its data, you can automatically implement the published properties of that object by deriving your custom System::Variant (see page 3421) type from TPublishableVariantType instead.

See Also

TPublishableVariantType (see page 4694)

TVarData (see page 3740)

Supporting Properties and Methods in Custom Variants

51.1.17.1 TInvokeableVariantType Members

The following tables list the members exposed by TInvokeableVariantType.

Protected Methods

	Name	Description
	_AddRef (see page 4993)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 4993)	Responds when the caller releases a reference to this object's interface.
	DispInvoke (see page 5018)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5014) method.
	FixupIdent (see page 5020)	Fixes up dispatch identifier.
	LeftPromotion (see page 5000)	Indicates whether the left argument to a binary operation should be coerced to a different type.

	OlePromotion (see page 5001)	Indicates the type to which a System::Variant (see page 3421) value must be cast to make it Automation-compatible.
	QueryInterface (see page 5002)	Returns a reference to a specified interface if the object supports that interface.
	RaiseCastError (see page 5002)	Generates a system error that indicates an invalid cast.
	RaiseDispError (see page 5003)	Generates a system error indicating an invalid dispatch attempt.
	RaiseInvalidOp (see page 5003)	Generates a system error indicating an invalid operation.
	RightPromotion (see page 5004)	Indicates whether the right argument to a binary operation should be coerced to a different type.
	SimplisticClear (see page 5005)	Clears a TVarData (see page 3740) record by re-initializing it.
	SimplisticCopy (see page 5005)	Performs a simple block copy of one TVarData (see page 3740) record to another.
	VarDataCast (see page 5006)	Performs a cast from one System::Variant (see page 3421) type to another.
	VarDataCastTo (see page 5007)	This is the overview for the VarDataCastTo method overload.
	VarDataCastToOleStr (see page 5007)	Performs a cast to a System::Variant (see page 3421) type of varOleStr (see page 3869).
	VarDataClear (see page 5008)	Clears the data from a built-in System::Variant (see page 3421) type.
	VarDataCopy (see page 5008)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataCopyNoInd (see page 5009)	Copies the data from one built-in System::Variant (see page 3421) type to another.
	VarDataFromOleStr (see page 5009)	Creates a TVarData (see page 3740) record to represent a Unicode string.
	VarDataFromStr (see page 5010)	Creates a TVarData (see page 3740) record to represent a string.
	VarDataInit (see page 5010)	Initializes a TVarData (see page 3740) record.
	VarDataIsArray (see page 5010)	Indicates whether a TVarData (see page 3740) record represents the data from a System::Variant (see page 3421) array.
	VarDataIsByRef (see page 5011)	Indicates whether a TVarData (see page 3740) record stores its data indirectly.
	VarDataIsEmptyParam (see page 5011)	Indicates whether a TVarData (see page 3740) record represents an unassigned optional parameter.
	VarDataIsFloat (see page 5011)	Indicates whether a TVarData (see page 3740) record represents a floating-point value.
	VarDataIsNumeric (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a numeric value.
	VarDataIsOrdinal (see page 5012)	Indicates whether a TVarData (see page 3740) record represents a ordinal value.
	VarDataIsStr (see page 5013)	Indicates whether a TVarData (see page 3740) record represents a string (normal or Unicode).
	VarDataToStr (see page 5013)	Returns a string that represents the value from a TVarData (see page 3740) record.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	BinaryOp (see page 4994)	Implements any binary operations of which the custom System::Variant (see page 3421) type is capable.
	Cast (see page 4995)	Converts the data from another System::Variant (see page 3421) type so that it represents this custom type's data format.
	CastTo (see page 4995)	Converts data in the format of this TCustomVariantType (see page 4987) to the format of another System::Variant (see page 3421) type.
	CastToOle (see page 4996)	Converts data in the format of this custom System::Variant (see page 3421) type to a format that can be used by COM Variants (see page 4973).
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.

	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
A	Clear (see page 4996)	Sets the data for this custom System::Variant (see page 3421) type to a blank or unassigned value.
V	Compare (see page 4997)	Implements comparisons that involve the custom System::Variant (see page 3421) type.
V	CompareOp (see page 4998)	Implements any comparison operations of which the custom System::Variant (see page 3421) type is capable.
A	Copy (see page 4999)	Copies the data from a System::Variant (see page 3421) of this custom type.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 4990)	This is the overview for the Create constructor overload.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 4991)	Frees an instance of TCustomVariantType (see page 4987).
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	DoFunction (see page 5018)	Implements all methods that return values.
V	DoProcedure (see page 5019)	Implements all methods that do not return values.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
V	GetProperty (see page 5020)	Returns the value for any property this System::Variant (see page 3421) type supports.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
V	IsClear (see page 5000)	Indicates whether the data from a custom System::Variant (see page 3421) represents a blank or unassigned value.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	SetProperty (see page 5021)	Sets the value for any property this System::Variant (see page 3421) type supports.
V	ToString (see page 3380)	Returns a string containing the class name.
V	UnaryOp (see page 5006)	Implements any unary operations of which the custom System::Variant (see page 3421) type is capable.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	VarType (see page 5014)	Indicates the System::Variant (see page 3421) type code for the custom System::Variant (see page 3421) type.

51.1.17.2 TInvokeableVariantType Methods

The methods of the TInvokeableVariantType class are listed here.

Protected Methods

	Name	Description
	DispInvoke (see page 5018)	Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5014) method.
	FixupIdent (see page 5020)	Fixes up dispatch identifier.

Public Methods

	Name	Description
	DoFunction (see page 5018)	Implements all methods that return values.
	DoProcedure (see page 5019)	Implements all methods that do not return values.
	GetProperty (see page 5020)	Returns the value for any property this System::Variant (see page 3421) type supports.
	SetProperty (see page 5021)	Sets the value for any property this System::Variant (see page 3421) type supports.

51.1.17.2.1 TInvokeableVariantType.DispInvoke Method

Invokes a method call by dispatching it to the appropriate TInvokeableVariantType (see page 5014) method.

Pascal

```
procedure DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); override;
```

C++

```
virtual __fastcall DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void *
Params);
```

Description

When the application tries to call a method, set a property, or read a property using a System::Variant (see page 3421) of the custom System::Variant (see page 3421) type, the system automatically generates a call to DispInvoke. TInvokeableVariantType (see page 5014) responds by generating a call to the GetProperty (see page 5020), SetProperty (see page 5021), DoProcedure (see page 5019), or DoFunction (see page 5018) method, as appropriate.

Dest is a TVarData (see page 3740) record that receives the return value of any function calls.

Source is the TVarData (see page 3740) record that holds the data from the custom System::Variant (see page 3421) whose property or method was invoked.

CallDesc identifies the method that was called, including the type and number of its arguments.

Params points to an array of parameter values that were passed in the method call.

See Also

GetProperty (see page 5020)

SetProperty (see page 5021)

DoProcedure (see page 5019)

DoFunction (see page 5018)

51.1.17.2.2 TInvokeableVariantType.DoFunction Method

Implements all methods that return values.

Pascal

```
function DoFunction(var Dest: TVarData; const V: TVarData; const Name: string; const
```

```
Arguments: TVarDataArray): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoFunction(TVarData Dest, const TVarData V, const UnicodeString
Name, const TVarDataArray Arguments);
```

Description

Override DoFunction to provide the implementation for method calls that return values. When the application invokes a System::Variant (see page 3421) member function (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++) a call is automatically generated to DoFunction.

Dest is a TVarData (see page 3740) record that receives the value that the function returns.

V is the TVarData (see page 3740) record from the System::Variant (see page 3421) instance that was used to call the function.

Name is the name of the function.

Arguments is a dynamic array of TVarData (see page 3740) records, where each one represents the value of an argument passed to the function.

DoFunction returns true if it executes the method call, false if it does not implement the specified function.

As implemented in TInvokeableVariant, DoFunction always returns false, because it does not implement any function calls.

See Also

GetProperty (see page 5020)

DoProcedure (see page 5019)

SetProperty (see page 5021)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.1.17.2.3 TInvokeableVariantType.DoProcedure Method

Implements all methods that do not return values.

Pascal

```
function DoProcedure(const V: TVarData; const Name: string; const Arguments: TVarDataArray):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoProcedure(const TVarData V, const UnicodeString Name, const
TVarDataArray Arguments);
```

Description

Override DoProcedure to provide the implementation for methods calls that do not return values. When the application invokes a System::Variant (see page 3421) member procedure (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++) a call is automatically generated to DoProcedure that allows the TInvokeableVariant descendant to implement the call.

V is the TVarData (see page 3740) record from the System::Variant (see page 3421) instance that was used to call the procedure.

Name is the name of the procedure.

Arguments is a dynamic array of TVarData (see page 3740) records, where each one represents the value of an argument passed to the procedure.

DoProcedure returns true if it executes the procedure call, false if it does not implement the specified procedure.

As implemented in TInvokeableVariant, DoProcedure always returns false, because it does not implement any procedure calls.

See Also

GetProperty (see page 5020)

DoFunction (see page 5018)

SetProperty (see page 5021)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.1.17.2.4 TInvokeableVariantType.FixupIdent Method

Fixes up dispatch identifier.

Pascal

```
function FixupIdent(const AText: string): string; virtual;
```

C++

```
virtual __fastcall UnicodeString FixupIdent(const UnicodeString AText);
```

Description

FixupIdent is called internally to determine the final fixup for the dispatch identifier. As defined in TInvokeableVariantType (see page 5014), this fixup consists of converting the identifier to all upper case.

51.1.17.2.5 TInvokeableVariantType.GetProperty Method

Returns the value for any property this System::Variant (see page 3421) type supports.

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const UnicodeString
Name);
```

Description

Override GetProperty to provide the read implementations for properties of instances of the System::Variant (see page 3421). When the application reads a System::Variant (see page 3421) property (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++) a call is automatically generated to GetProperty that allows the TInvokeableVariant descendant to implement the call.

Dest is a TVarData (see page 3740) record that receives the property value.

V is the TVarData (see page 3740) record from the System::Variant (see page 3421) instance whose property is being read.

Name is the name of the property.

GetProperty returns true if it returns a property value, false if it does not implement the specified property.

As implemented in TInvokeableVariant, GetProperty always returns false, because it does not implement any properties.

See Also

DoFunction (see page 5018)

DoProcedure (see page 5019)

SetProperty (see page 5021)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.1.17.2.6 TInvokeableVariantType.SetProperty Method

Sets the value for any property this System::Variant (see page 3421) type supports.

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean SetProperty(const TVarData V, const UnicodeString Name, const
TVarData Value);
```

Description

Override SetProperty to provide the write implementation for the properties of instances of the invokeable System::Variant (see page 3421) type. When the application writes to a System::Variant (see page 3421) property (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++) a call is automatically generated to SetProperty that allows the TInvokeableVariant descendant to implement the call.

Name is the name of the property.

Value is a TVarData (see page 3740) record that contains the new property value.

GetProperty (see page 5020) returns true if it sets a property value, false if it does not implement the specified property or if the specified property is read-only.

As implemented in TInvokeableVariant, GetProperty (see page 5020) always returns false, because it does not implement any properties.

See Also

DoFunction (see page 5018)

DoProcedure (see page 5019)

GetProperty (see page 5020)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.2 Interfaces

The following table lists interfaces in this documentation.

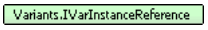
Interfaces

	Name	Description
	IVarInstanceReference (see page 5022)	IVarInstanceReference is the interface for extracting the instance that represents the data from a Variant (see page 3421) of the custom type.
	IVarInvokeable (see page 5023)	IVarInvokeable is the interface for custom Variant (see page 3421) types that implement dynamic properties and methods.

51.2.1 IVarInstanceReference Interface

IVarInstanceReference is the interface for extracting the instance that represents the data from a Variant (see page 3421) of the custom type.

Class Hierarchy



Pascal

```
IVarInstanceReference = interface;
```

C++

```
__interface IVarInstanceReference;
```

File

Variants

Description

Implement IVarInstanceReference in the proxy classes that manage a custom Variant (see page 3421) data type. IVarInstanceReference exposes a single method that must be implemented, called GetInstance (see page 5023).

See Also

TPublishableVariantType (see page 4694)

Creating a Class to Enable the Custom Variant Type

51.2.1.1 IVarInstanceReference Members

The following tables list the members exposed by IVarInstanceReference.

Public Methods

	Name	Description
	GetInstance (see page 5023)	Returns the object instance that represents the data from a Variant (see page 3421) of the custom type.

51.2.1.2 IVarInstanceReference Methods

The methods of the IVarInstanceReference class are listed here.

Public Methods

	Name	Description
	GetInstance (see page 5023)	Returns the object instance that represents the data from a Variant (see page 3421) of the custom type.

51.2.1.2.1 IVarInstanceReference.GetInstance Method

Returns the object instance that represents the data from a Variant (see page 3421) of the custom type.

Pascal

```
function GetInstance(const V: TVarData): TObject;
```

C++

```
__fastcall TObject * GetInstance(const TVarData V);
```

Description

Implement GetInstance to return the object that represents the data from a Variant (see page 3421) of the custom Type.

The V parameter is the data from a Variant (see page 3421) of the custom type. This data must include a reference to an object that represents the Variant (see page 3421)'s data and is returned by GetInstance.

See Also

TPublishableVariantType (see page 4694)

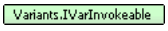
GetInstance (see page 4698)

Supporting Properties and Methods in Custom Variants

51.2.2 IVarInvokeable Interface

IVarInvokeable is the interface for custom Variant (see page 3421) types that implement dynamic properties and methods.

Class Hierarchy



Pascal

```
IVarInvokeable = interface;
```

C++

```
__interface IVarInvokeable;
```

File

Variants

Description

Implement IVarInvokeable in the proxy classes that manage a custom Variant (see page 3421) data type. IVarInvokeable exposes four functions, which need to be implemented. Each of these functions are called whenever a function, method, or a property is called on the custom Variant (see page 3421) managed by the proxy class implementing IVarInvokeable.

See Also

TInvokeableVariantType (see page 5014)

TVarData (see page 3740)

Supporting Properties and Methods in Custom Variants

51.2.2.1 IVarInvokeable Members

The following tables list the members exposed by IVarInvokeable.

Public Methods

	Name	Description
	DoFunction (see page 5024)	Implements all the methods that return values.
	DoProcedure (see page 5025)	Implements all methods that do not return values.
	GetProperty (see page 5025)	Returns the value for any property supported by a custom System::Variant (see page 3421) type.
	SetProperty (see page 5026)	Sets the value for any property supported by a custom System::Variant (see page 3421) type.

51.2.2.2 IVarInvokeable Methods

The methods of the IVarInvokeable class are listed here.

Public Methods

	Name	Description
	DoFunction (see page 5024)	Implements all the methods that return values.
	DoProcedure (see page 5025)	Implements all methods that do not return values.
	GetProperty (see page 5025)	Returns the value for any property supported by a custom System::Variant (see page 3421) type.
	SetProperty (see page 5026)	Sets the value for any property supported by a custom System::Variant (see page 3421) type.

51.2.2.2.1 IVarInvokeable.DoFunction Method

Implements all the methods that return values.

Pascal

```
function DoFunction(var Dest: TVarData; const V: TVarData; const Name: string; const
Arguments: TVarDataArray): Boolean;
```

C++

```
__fastcall Boolean DoFunction(TVarData Dest, const TVarData V, const UnicodeString Name, const
TVarDataArray Arguments);
```

Description

Implement DoFunction to provide the implementation for method calls that return values. When the application invokes a System::Variant (see page 3421) member function (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++), a call is automatically generated to DoFunction.

`Dest` is a TVarData (see page 3740) record that receives the value returned by the function. V is the TVarData (see page 3740) record from the System::Variant (see page 3421) instance that was used to call the function. `Name` is the name of the function being called. `Arguments` is a dynamic array of TVarData (see page 3740) records, where each one represents the value of an argument passed to the function.

DoFunction returns **True** if it executes the method call, or **False** if it does not implement the specified function.

See Also

DoFunction (see page 5018)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.2.2.2.2 IVarInvokeable.DoProcedure Method

Implements all methods that do not return values.

Pascal

```
function DoProcedure(const V: TVarData; const Name: string; const Arguments: TVarDataArray):
Boolean;
```

C++

```
__fastcall Boolean DoProcedure(const TVarData V, const UnicodeString Name, const TVarDataArray
Arguments);
```

Description

Implement DoProcedure to provide the implementation for method calls that do not return values. When the application invokes a System::Variant (see page 3421) member function (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++), a call is automatically generated to DoProcedure.

`V` is the TVarData (see page 3740) record from the System::Variant (see page 3421) instance that was used to call the function. `Name` is the name of the function being called. `Arguments` is a dynamic array of TVarData (see page 3740) records, where each record represents the value of an argument passed to the function.

DoProcedure returns **True** if it executes the method call, or **False** if it does not implement the specified procedure.

See Also

DoProcedure (see page 5019)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.2.2.2.3 IVarInvokeable.GetProperty Method

Returns the value for any property supported by a custom System::Variant (see page 3421) type.

Pascal

```
function GetProperty(var Dest: TVarData; const V: TVarData; const Name: string): Boolean;
```

C++

```
__fastcall Boolean GetProperty(TVarData Dest, const TVarData V, const UnicodeString Name);
```

Description

Implement GetProperty to provide the implementation for reading properties. When the application tries to read a property of a System::Variant (see page 3421) (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++), a call is automatically generated to GetProperty.

`Dest` is a TVarData (see page 3740) record that receives the value of the read property. `V` is the TVarData (see page 3740) record from the System::Variant (see page 3421) instance that was used to access the property. `Name` is the name of the property being read.

GetProperty returns **True** if it handles the property, **False** if it does not implement the specified property.

See Also

GetProperty (see page 5020)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.2.2.2.4 IVarInvokeable.SetProperty Method

Sets the value for any property supported by a custom System::Variant (see page 3421) type.

Pascal

```
function SetProperty(const V: TVarData; const Name: string; const Value: TVarData): Boolean;
```

C++

```
__fastcall Boolean SetProperty(const TVarData V, const UnicodeString Name, const TVarData
Value);
```

Description

Implement SetProperty to provide the implementation for setting properties. When the application tries to set a property of a System::Variant (see page 3421) (Delphi) or calls the System::Variant (see page 3421)'s Exec method (C++), a call is automatically generated to SetProperty.

`Value` is a TVarData (see page 3740) record that is the value assigned to the property. `V` is the TVarData (see page 3740) record from the System::Variant (see page 3421) instance that was used to access the property. `Name` is the name of the property being set.

SetProperty returns **True** if it handles the property, **False** if it does not implement the specified property.

See Also

SetProperty (see page 5021)

System::Variant::Exec

Supporting Properties and Methods in Custom Variants

51.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	_DispInvoke (see page 5029)	_DispInvoke is used internally by the Delphi compiler.
	_OleVarFromInt (see page 5030)	_OleVarFromInt is used internally by the Delphi compiler.
	_OleVarFromInt64 (see page 5030)	_OleVarFromInt64 is used internally by the Delphi compiler.
	_OleVarFromLStr (see page 5031)	_OleVarFromLStr is used internally by the Delphi compiler.
	_OleVarFromPStr (see page 5031)	_OleVarFromPStr is used internally by the Delphi compiler.
	_OleVarFromUInt64 (see page 5031)	_OleVarFromUInt64 is used internally by the Delphi compiler.
	_OleVarFromUStr (see page 5032)	_OleVarFromUStr is used internally by the Delphi compiler.
	_OleVarFromVar (see page 5032)	_OleVarFromVar is used internally by the Delphi compiler.
	_VarAdd (see page 5032)	_VarAdd is used internally by the Delphi compiler.
	_VarAddRef (see page 5033)	_VarAddRef is used internally by the Delphi compiler.
	_VarAnd (see page 5033)	_VarAnd is used internally by the Delphi compiler.
	_VarArrayGet (see page 5034)	_VarArrayGet is used internally by the Delphi compiler.
	_VarArrayPut (see page 5034)	_VarArrayPut is used internally by the Delphi compiler.
	_VarArrayRedim (see page 5034)	_VarArrayRedim is used internally by the Delphi compiler.
	_VarCast (see page 5035)	_VarCast is used internally by the Delphi compiler.
	_VarCastOle (see page 5035)	_VarCastOle is used internally by the Delphi compiler.
	_VarClear (see page 5035)	_VarClear is used internally by the Delphi compiler.
	_VarClr (see page 5036)	_VarClr is used internally by the Delphi compiler.
	_VarCmpEQ (see page 5036)	_VarCmpEQ is used internally by the Delphi compiler.
	_VarCmpGE (see page 5037)	_VarCmpGE is used internally by the Delphi compiler.
	_VarCmpGT (see page 5037)	_VarCmpGT is used internally by the Delphi compiler.
	_VarCmpLE (see page 5037)	_VarCmpLE is used internally by the Delphi compiler.
	_VarCmpLT (see page 5038)	_VarCmpLT is used internally by the Delphi compiler.
	_VarCmpNE (see page 5038)	_VarCmpNE is used internally by the Delphi compiler.
	_VarCopy (see page 5038)	_VarCopy is used internally by the Delphi compiler.
	_VarCopyNoInd (see page 5039)	_VarCopyNoInd is used internally by the Delphi compiler.
	_VarDiv (see page 5039)	_VarDiv is used internally by the Delphi compiler.
	_VarFromBool (see page 5040)	_VarFromBool is used internally by the Delphi compiler.
	_VarFromCurr (see page 5040)	_VarFromCurr is used internally by the Delphi compiler.
	_VarFromDisp (see page 5040)	_VarFromDisp is used internally by the Delphi compiler.
	_VarFromDynArray (see page 5041)	_VarFromDynArray is used internally by the Delphi compiler.
	_VarFromInt (see page 5041)	_VarFromInt is used internally by the Delphi compiler.
	_VarFromInt64 (see page 5041)	_VarFromInt64 is used internally by the Delphi compiler.
	_VarFromIntf (see page 5042)	_VarFromIntf is used internally by the Delphi compiler.
	_VarFromLStr (see page 5042)	_VarFromLStr is used internally by the Delphi compiler.
	_VarFromPStr (see page 5043)	_VarFromPStr is used internally by the Delphi compiler.
	_VarFromReal (see page 5043)	_VarFromReal is used internally by the Delphi compiler.
	_VarFromTDateTime (see page 5043)	_VarFromTDateTime is used internally by the Delphi compiler.
	_VarFromUInt64 (see page 5044)	_VarFromUInt64 is used internally by the Delphi compiler.
	_VarFromUStr (see page 5044)	_VarFromUStr is used internally by the Delphi compiler.
	_VarFromWChar (see page 5044)	_VarFromWChar is used internally by the Delphi compiler.
	_VarFromWStr (see page 5045)	_VarFromWStr is used internally by the Delphi compiler.
	_VarMod (see page 5045)	_VarMod is used internally by the Delphi compiler.
	_VarMul (see page 5046)	_VarMul is used internally by the Delphi compiler.

	_VarNeg (see page 5046)	_VarNeg is used internally by the Delphi compiler.
	_VarNot (see page 5046)	_VarNot is used internally by the Delphi compiler.
	_VarOr (see page 5047)	_VarOr is used internally by the Delphi compiler.
	_VarRDiv (see page 5047)	_VarRDiv is used internally by the Delphi compiler.
	_VarShl (see page 5047)	_VarShl is used internally by the Delphi compiler.
	_VarShr (see page 5048)	_VarShr is used internally by the Delphi compiler.
	_VarSub (see page 5048)	_VarSub is used internally by the Delphi compiler.
	_VarToBool (see page 5049)	_VarToBool is used internally by the Delphi compiler.
	_VarToCurrency (see page 5049)	_VarToCurrency is used internally by the Delphi compiler.
	_VarToDisp (see page 5049)	_VarToDisp is used internally by the Delphi compiler.
	_VarToDynArray (see page 5050)	_VarToDynArray is used internally by the Delphi compiler.
	_VarToInt64 (see page 5050)	_VarToInt64 is used internally by the Delphi compiler.
	_VarToInteger (see page 5050)	_VarToInteger is used internally by the Delphi compiler.
	_VarToIntf (see page 5051)	_VarToIntf is used internally by the Delphi compiler.
	_VarToLStr (see page 5051)	_VarToLStr is used internally by the Delphi compiler.
	_VarToPStr (see page 5052)	_VarToPStr is used internally by the Delphi compiler.
	_VarToReal (see page 5052)	_VarToReal is used internally by the Delphi compiler.
	_VarToUInt64 (see page 5052)	_VarToUInt64 is used internally by the Delphi compiler.
	_VarToUStr (see page 5053)	_VarToUStr is used internally by the Delphi compiler.
	_VarToWStr (see page 5053)	_VarToWStr is used internally by the Delphi compiler.
	_VarXor (see page 5053)	_VarXor is used internally by the Delphi compiler.
	DynArrayFromVariant (see page 5054)	Creates a dynamic array from a System::Variant (see page 3421).
	DynArrayToVariant (see page 5054)	Creates a System::Variant (see page 3421) array from a dynamic array.
	FindCustomVariantType (see page 5055)	Retrieves the object that implements a custom System::Variant (see page 3421) type.
	FindCustomVariantType (see page 5055)	Retrieves the object that implements a custom System::Variant (see page 3421) type.
	FindVarData (see page 5055)	Returns pointer to variant's data.
	HandleConversionException (see page 5056)	Handles exceptions during variant type conversions.
	Null (see page 5056)	Returns a Null variant.
	Unassigned (see page 5058)	Returns an "empty" variant.
	VarArrayAsPSafeArray (see page 5059)	This is function Variants.VarArrayAsPSafeArray.
	VarArrayCreate (see page 5059)	Creates a variant array.
	VarArrayCreateError (see page 5061)	Indicates an array creation error.
	VarArrayDimCount (see page 5061)	Returns number of dimensions of a variant array.
	VarArrayGet (see page 5063)	Returns a System::Variant (see page 3421) that represents a single value from a multi-dimensional System::Variant (see page 3421) array.
	VarArrayHighBound (see page 5065)	Returns high bound for a dimension in a variant array.
	VarArrayLock (see page 5067)	Locks a variant array and returns a pointer to the data.
	VarArrayLowBound (see page 5067)	Returns the low bound of a dimension in a variant array.
	VarArrayOf (see page 5069)	Creates and fills a one-dimensional variant array.
	VarArrayPut (see page 5069)	Sets the value of a single cell in a multi-dimensional System::Variant (see page 3421) array.
	VarArrayRef (see page 5071)	Returns a reference to the specified variant array.
	VarArrayUnlock (see page 5072)	Unlocks a variant array.
	VarAsError (see page 5072)	Converts a HRESULT (see page 3754) into an Error (see page 3560) variant.
	VarAsType (see page 5073)	Converts a variant to specified type.
	VarCastError (see page 5073)	Raises an EVariantTypeCastError (see page 4986) exception.
	VarCastError (see page 5073)	Raises an EVariantTypeCastError (see page 4986) exception.
	VarCheckEmpty (see page 5074)	Raises an exception if a specified variant's value is Unassigned (see page 5058).
	VarCompareValue (see page 5074)	Returns the relationship between Variants (see page 4973).
	VarCopyNoInd (see page 5075)	Copies a System::Variant (see page 3421).
	VarEnsureRange (see page 5076)	Returns a System::Variant (see page 3421) with a value in a specified range.
	VarFromDateTime (see page 5076)	Returns a variant containing a specified data and time.

	VarInRange (see page 5077)	Indicates whether a System::Variant (see page 3421)'s value is in a specified range.
	VarInvalidNullOp (see page 5077)	Indicate variant operation on null.
	VarInvalidOp (see page 5078)	Indicate invalid variant operation.
	VarIsArray (see page 5078)	Indicates whether the specified variant is an array.
	VarIsArray (see page 5078)	Indicates whether the specified variant is an array.
	VarIsByRef (see page 5079)	Indicates whether the specified variant's value is by reference.
	VarIsClear (see page 5079)	Indicates whether the specified variant has an undefined value.
	VarIsCustom (see page 5080)	Indicates whether the specified variant is a custom variant.
	VarIsEmpty (see page 5080)	Indicates whether the specified variant is unassigned.
	VarIsEmptyParam (see page 5081)	Indicates whether the specified variant represents an unassigned optional parameter.
	VarIsError (see page 5082)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3754).
	VarIsError (see page 5082)	Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3754).
	VarIsFloat (see page 5082)	Indicates whether the specified variant represents a floating-point value.
	VarIsNull (see page 5083)	Indicates whether the specified variant is Null (see page 5056).
	VarIsNumeric (see page 5083)	Indicates whether the specified variant represents a numeric value.
	VarIsOrdinal (see page 5084)	Indicates whether the specified variant represents an ordinal value.
	VarIsStr (see page 5084)	Indicates whether the specified variant represents a string value.
	VarIsType (see page 5085)	Indicates whether the specified variant represents a specified type.
	VarIsType (see page 5085)	Indicates whether the specified variant represents a specified type.
	VarOverflowError (see page 5085)	Raises an EVariantOverflowError (see page 4985) exception.
	VarRangeCheckError (see page 5086)	Raises an EVariantRangeCheckError (see page 4986) exception.
	VarResultCheck (see page 5086)	Raises variant exception based on HRESULT (see page 3754) value.
	VarResultCheck (see page 5086)	Raises variant exception based on HRESULT (see page 3754) value.
	VarSameValue (see page 5087)	Indicates whether two Variants (see page 4973) have equivalent values.
	VarSupports (see page 5087)	Indicates whether the System::Variant (see page 3421)'s value is an interface that supports a specified interface.
	VarSupports (see page 5087)	Indicates whether the System::Variant (see page 3421)'s value is an interface that supports a specified interface.
	VarToDateTime (see page 5088)	Converts a specified variant to a TDateTime (see page 3323) value.
	VarToStr (see page 5088)	Converts a variant's value to a string.
	VarToStrDef (see page 5089)	Converts a variant's value to a string.
	VarToWideStr (see page 5089)	Converts a variant's value to a WideString (see page 3467).
	VarToWideStrDef (see page 5090)	Converts a variant's value to a WideString (see page 3467).
	VarType (see page 5090)	Returns the type code of a specified variant.
	VarTypeAsText (see page 5092)	Returns variant type descriptive string.
	VarTypeIsValidArrayType (see page 5092)	Indicates whether a System::Variant (see page 3421) type code represents a type that can be used for the elements in a System::Variant (see page 3421) array.
	VarTypeIsValidElementType (see page 5093)	Indicates whether a System::Variant (see page 3421) type code represents a type that can be used for the elements in a System::Variant (see page 3421) array.

51.3.1 Variants._DispInvoke Function

_DispInvoke is used internally by the Delphi compiler.

Pascal

```
procedure _DispInvoke(Dest: PVarData; const Source: TVarData; CallDesc: PCallDesc; Params:
Pointer); cdecl;
```

C++

```
__cdecl _DispInvoke(PVarData Dest, const TVarData Source, PCallDesc CallDesc, void * Params);
```

File

Variants

Description

Do not call _DispInvoke directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _DispInvoke is not publicly available to Delphi applications.

51.3.2 Variants._OleVarFromInt Function

_OleVarFromInt is used internally by the Delphi compiler.

Pascal

```
procedure _OleVarFromInt(var V: TVarData; const Value: Integer; const Range: ShortInt);
```

C++

```
_OleVarFromInt(TVarData V, const int Value, const ShortInt Range);
```

File

Variants

Description

Do not call _OleVarFromInt directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _OleVarFromInt is not publicly available to Delphi applications.

51.3.3 Variants._OleVarFromInt64 Function

_OleVarFromInt64 is used internally by the Delphi compiler.

Pascal

```
procedure _OleVarFromInt64(var V: TVarData; const Value: Int64);
```

C++

```
_OleVarFromInt64(TVarData V, const Int64 Value);
```

File

Variants

Description

Do not call _OleVarFromInt64 directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _OleVarFromInt64 is not publicly available to Delphi applications.

51.3.4 Variants._OleVarFromLStr Function

_OleVarFromLStr is used internally by the Delphi compiler.

Pascal

```
procedure _OleVarFromLStr(var V: TVarData; const Value: AnsiString);
```

C++

```
_OleVarFromLStr(TVarData V, const AnsiString Value);
```

File

Variants

Description

Do not call _OleVarFromLStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _OleVarFromLStr is not publicly available to Delphi applications.

51.3.5 Variants._OleVarFromPStr Function

_OleVarFromPStr is used internally by the Delphi compiler.

Pascal

```
procedure _OleVarFromPStr(var V: TVarData; const Value: ShortString);
```

C++

```
_OleVarFromPStr(TVarData V, const ShortString Value);
```

File

Variants

Description

Do not call _OleVarFromPStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _OleVarFromPStr is not publicly available to Delphi applications.

51.3.6 Variants._OleVarFromUInt64 Function

_OleVarFromUInt64 is used internally by the Delphi compiler.

Pascal

```
procedure _OleVarFromUInt64(var V: TVarData; const Value: UInt64);
```

C++

```
_OleVarFromUInt64(TVarData V, const UInt64 Value);
```

File

Variants

Description

Do not call _OleVarFromUInt64 directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _OleVarFromUInt64 is not publicly available to Delphi applications.

51.3.7 Variants._OleVarFromUStr Function

_OleVarFromUStr is used internally by the Delphi compiler.

Pascal

```
procedure _OleVarFromUStr(var V: TVarData; const Value: UnicodeString);
```

C++

```
_OleVarFromUStr(TVarData V, const UnicodeString Value);
```

File

Variants

Description

Do not call _OleVarFromUStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _OleVarFromUStr is not publicly available to Delphi applications.

51.3.8 Variants._OleVarFromVar Function

_OleVarFromVar is used internally by the Delphi compiler.

Pascal

```
procedure _OleVarFromVar(var Dest: TVarData; const Source: TVarData);
```

C++

```
_OleVarFromVar(TVarData Dest, const TVarData Source);
```

File

Variants

Description

Do not call _OleVarFromVar directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _OleVarFromVar is not publicly available to Delphi applications.

51.3.9 Variants._VarAdd Function

_VarAdd is used internally by the Delphi compiler.

Pascal

```
procedure _VarAdd(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarAdd(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarAdd directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarAdd is not publicly available to Delphi applications.

51.3.10 Variants._VarAddRef Function

_VarAddRef is used internally by the Delphi compiler.

Pascal

```
procedure _VarAddRef(var V: TVarData);
```

C++

```
_VarAddRef(TVarData V);
```

File

Variants

Description

Do not call _VarAddRef directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarAddRef is not publicly available to Delphi applications.

51.3.11 Variants._VarAnd Function

_VarAnd is used internally by the Delphi compiler.

Pascal

```
procedure _VarAnd(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarAnd(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarAnd directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarAnd is not publicly available to Delphi applications.

51.3.12 Variants._VarArrayGet Function

_VarArrayGet is used internally by the Delphi compiler.

Pascal

```
function _VarArrayGet(var A: TVarData; IndexCount: Integer; const Indices:
TVarArrayCoorArray): TVarData; cdecl;
```

C++

```
__cdecl TVarData _VarArrayGet(TVarData A, int IndexCount, const TVarArrayCoorArray Indices);
```

File

Variants

Description

Do not call _VarArrayGet directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarArrayGet is not publicly available to Delphi applications.

51.3.13 Variants._VarArrayPut Function

_VarArrayPut is used internally by the Delphi compiler.

Pascal

```
procedure _VarArrayPut(var A: TVarData; const Value: TVarData; IndexCount: Integer; const
Indices: TVarArrayCoorArray); cdecl;
```

C++

```
__cdecl _VarArrayPut(TVarData A, const TVarData Value, int IndexCount, const
TVarArrayCoorArray Indices);
```

File

Variants

Description

Do not call _VarArrayPut directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarArrayPut is not publicly available to Delphi applications.

51.3.14 Variants._VarArrayRedim Function

_VarArrayRedim is used internally by the Delphi compiler.

Pascal

```
procedure _VarArrayRedim(var A: TVarData; HighBound: Integer);
```

C++

```
_VarArrayRedim(TVarData A, int HighBound);
```

File

Variants

Description

Do not call _VarArrayRedim directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarArrayRedim is not publicly available to Delphi applications.

51.3.15 Variants._VarCast Function

_VarCast is used internally by the Delphi compiler.

Pascal

```
procedure _VarCast(var Dest: TVarData; const Source: TVarData; AVarType: Integer);
```

C++

```
_VarCast(TVarData Dest, const TVarData Source, int AVarType);
```

File

Variants

Description

Do not call _VarCast directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCast is not publicly available to Delphi applications.

51.3.16 Variants._VarCastOle Function

_VarCastOle is used internally by the Delphi compiler.

Pascal

```
procedure _VarCastOle(var Dest: TVarData; const Source: TVarData; AVarType: Integer);
```

C++

```
_VarCastOle(TVarData Dest, const TVarData Source, int AVarType);
```

File

Variants

Description

Do not call _VarCastOle directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCastOle is not publicly available to Delphi applications.

51.3.17 Variants._VarClear Function

_VarClear is used internally by the Delphi compiler.

Pascal

```
procedure _VarClear(var V: TVarData);
```

C++

```
_VarClear(TVarData V);
```

File

Variants

Description

Do not call _VarClear directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarClear is not publicly available to Delphi applications.

51.3.18 Variants._VarClr Function

_VarClr is used internally by the Delphi compiler.

Pascal

```
procedure _VarClr(var V: TVarData);
```

C++

```
_VarClr(TVarData V);
```

File

Variants

Description

Do not call _VarClr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarClr is not publicly available to Delphi applications.

51.3.19 Variants._VarCmpEQ Function

_VarCmpEQ is used internally by the Delphi compiler.

Pascal

```
procedure _VarCmpEQ(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpEQ(const TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarCmpEQ directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCmpEQ is not publicly available to Delphi applications.

51.3.20 Variants._VarCmpGE Function

_VarCmpGE is used internally by the Delphi compiler.

Pascal

```
procedure _VarCmpGE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpGE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarCmpGE directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCmpGE is not publicly available to Delphi applications.

51.3.21 Variants._VarCmpGT Function

_VarCmpGT is used internally by the Delphi compiler.

Pascal

```
procedure _VarCmpGT(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpGT(const TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarCmpGT directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCmpGT is not publicly available to Delphi applications.

51.3.22 Variants._VarCmpLE Function

_VarCmpLE is used internally by the Delphi compiler.

Pascal

```
procedure _VarCmpLE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpLE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarCmpLE directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCmpLE is not publicly available to Delphi applications.

51.3.23 Variants._VarCmpLT Function

_VarCmpLT is used internally by the Delphi compiler.

Pascal

```
procedure _VarCmpLT(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpLT(const TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarCmpLT directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCmpLT is not publicly available to Delphi applications.

51.3.24 Variants._VarCmpNE Function

_VarCmpNE is used internally by the Delphi compiler.

Pascal

```
procedure _VarCmpNE(const Left: TVarData; const Right: TVarData);
```

C++

```
_VarCmpNE(const TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarCmpNE directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCmpNE is not publicly available to Delphi applications.

51.3.25 Variants._VarCopy Function

_VarCopy is used internally by the Delphi compiler.

Pascal

```
procedure _VarCopy(var Dest: TVarData; const Source: TVarData);
```

C++

```
_VarCopy(TVarData Dest, const TVarData Source);
```

File

Variants

Description

Do not call _VarCopy directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCopy is not publicly available to Delphi applications.

51.3.26 Variants._VarCopyNoInd Function

_VarCopyNoInd is used internally by the Delphi compiler.

Pascal

```
procedure _VarCopyNoInd(var Dest: TVarData; const Source: TVarData);
```

C++

```
_VarCopyNoInd(TVarData Dest, const TVarData Source);
```

File

Variants

Description

Do not call _VarCopyNoInd directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarCopyNoInd is not publicly available to Delphi applications.

51.3.27 Variants._VarDiv Function

_VarDiv is used internally by the Delphi compiler.

Pascal

```
procedure _VarDiv(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarDiv(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarDiv directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarDiv is not publicly available to Delphi applications.

51.3.28 Variants._VarFromBool Function

_VarFromBool is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromBool(var V: TVarData; const Value: Boolean);
```

C++

```
_VarFromBool(TVarData V, const Boolean Value);
```

File

Variants

Description

Do not call _VarFromBool directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromBool is not publicly available to Delphi applications.

51.3.29 Variants._VarFromCurr Function

_VarFromCurr is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromCurr;
```

C++

```
_VarFromCurr();
```

File

Variants

Description

Do not call _VarFromCurr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromCurr is not publicly available to Delphi applications.

51.3.30 Variants._VarFromDisp Function

_VarFromDisp is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromDisp(var V: TVarData; const Value: IDispatch);
```

C++

```
_VarFromDisp(TVarData V, const IDispatch Value);
```

File

Variants

Description

Do not call _VarFromDisp directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromDisp is not publicly available to Delphi applications.

51.3.31 Variants._VarFromDynArray Function

_VarFromDynArray is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromDynArray(var V: TVarData; const DynArray: Pointer; TypeInfo: Pointer);
```

C++

```
_VarFromDynArray(TVarData V, const void * DynArray, void * TypeInfo);
```

File

Variants

Description

Do not call _VarFromDynArray directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromDynArray is not publicly available to Delphi applications.

51.3.32 Variants._VarFromInt Function

_VarFromInt is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromInt(var V: TVarData; const Value: Integer; const Range: ShortInt);
```

C++

```
_VarFromInt(TVarData V, const int Value, const ShortInt Range);
```

File

Variants

Description

Do not call _VarFromInt directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromInt is not publicly available to Delphi applications.

51.3.33 Variants._VarFromInt64 Function

_VarFromInt64 is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromInt64(var V: TVarData; const Value: Int64);
```

C++

```
_VarFromInt64(TVarData V, const Int64 Value);
```

File

Variants

Description

Do not call _VarFromInt64 directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromInt64 is not publicly available to Delphi applications.

51.3.34 Variants._VarFromIntf Function

_VarFromIntf is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromIntf(var V: TVarData; const Value: IInterface);
```

C++

```
_VarFromIntf(TVarData V, const IInterface Value);
```

File

Variants

Description

Do not call _VarFromIntf directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromIntf is not publicly available to Delphi applications.

51.3.35 Variants._VarFromLStr Function

_VarFromLStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromLStr(var V: TVarData; const Value: AnsiString);
```

C++

```
_VarFromLStr(TVarData V, const AnsiString Value);
```

File

Variants

Description

Do not call _VarFromLStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromLStr is not publicly available to Delphi applications.

51.3.36 Variants._VarFromPStr Function

_VarFromPStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromPStr(var V: TVarData; const Value: ShortString);
```

C++

```
_VarFromPStr(TVarData V, const ShortString Value);
```

File

Variants

Description

Do not call _VarFromPStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromPStr is not publicly available to Delphi applications.

51.3.37 Variants._VarFromReal Function

_VarFromReal is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromReal;
```

C++

```
_VarFromReal();
```

File

Variants

Description

Do not call _VarFromReal directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromReal is not publicly available to Delphi applications.

51.3.38 Variants._VarFromTDateTime Function

_VarFromTDateTime is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromTDateTime;
```

C++

```
_VarFromTDateTime();
```

File

Variants

Description

Do not call _VarFromTDateTime directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromTDateTime is not publicly available to Delphi applications.

51.3.39 Variants._VarFromUInt64 Function

_VarFromUInt64 is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromUInt64(var V: TVarData; const Value: UInt64);
```

C++

```
_VarFromUInt64(TVarData V, const UInt64 Value);
```

File

Variants

Description

Do not call _VarFromUInt64 directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromUInt64 is not publicly available to Delphi applications.

51.3.40 Variants._VarFromUStr Function

_VarFromUStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromUStr(var V: TVarData; const Value: UnicodeString);
```

C++

```
_VarFromUStr(TVarData V, const UnicodeString Value);
```

File

Variants

Description

Do not call _VarFromUStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromUStr is not publicly available to Delphi applications.

51.3.41 Variants._VarFromWChar Function

_VarFromWChar is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromWChar(var V: TVarData; const Value: WideChar);
```

C++

```
_VarFromWChar(TVarData V, const WideChar Value);
```

File

Variants

Description

Do not call _VarFromWChar directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromWChar is not publicly available to Delphi applications.

51.3.42 Variants._VarFromWStr Function

_VarFromWStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarFromWStr(var V: TVarData; const Value: WideString);
```

C++

```
_VarFromWStr(TVarData V, const BSTR Value);
```

File

Variants

Description

Do not call _VarFromWStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarFromWStr is not publicly available to Delphi applications.

51.3.43 Variants._VarMod Function

_VarMod is used internally by the Delphi compiler.

Pascal

```
procedure _VarMod(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarMod(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarMod directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarMod is not publicly available to Delphi applications.

51.3.44 Variants._VarMul Function

_VarMul is used internally by the Delphi compiler.

Pascal

```
procedure _VarMul(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarMul(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarMul directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarMul is not publicly available to Delphi applications.

51.3.45 Variants._VarNeg Function

_VarNeg is used internally by the Delphi compiler.

Pascal

```
procedure _VarNeg(var Dest: TVarData);
```

C++

```
_VarNeg(TVarData Dest);
```

File

Variants

Description

Do not call _VarNeg directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarNeg is not publicly available to Delphi applications.

51.3.46 Variants._VarNot Function

_VarNot is used internally by the Delphi compiler.

Pascal

```
procedure _VarNot(var Dest: TVarData);
```

C++

```
_VarNot(TVarData Dest);
```

File

Variants

Description

Do not call _VarNot directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarNot is not publicly available to Delphi applications.

51.3.47 Variants._VarOr Function

_VarOr is used internally by the Delphi compiler.

Pascal

```
procedure _VarOr(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarOr(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarOr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarOr is not publicly available to Delphi applications.

51.3.48 Variants._VarRDiv Function

_VarRDiv is used internally by the Delphi compiler.

Pascal

```
procedure _VarRDiv(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarRDiv(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarRDiv directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarRDiv is not publicly available to Delphi applications.

51.3.49 Variants._VarShl Function

_VarShl is used internally by the Delphi compiler.

Pascal

```
procedure _VarShl(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarShl(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarShl directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarShl is not publicly available to Delphi applications.

51.3.50 Variants._VarShr Function

_VarShr is used internally by the Delphi compiler.

Pascal

```
procedure _VarShr(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarShr(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarShr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarShr is not publicly available to Delphi applications.

51.3.51 Variants._VarSub Function

_VarSub is used internally by the Delphi compiler.

Pascal

```
procedure _VarSub(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarSub(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarSub directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarSub is not publicly available to Delphi applications.

51.3.52 Variants._VarToBool Function

_VarToBool is used internally by the Delphi compiler.

Pascal

```
function _VarToBool(const V: TVarData): LongBool;
```

C++

```
LongBool _VarToBool(const TVarData V);
```

File

Variants

Description

Do not call _VarToBool directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToBool is not publicly available to Delphi applications.

51.3.53 Variants._VarToCurrency Function

_VarToCurrency is used internally by the Delphi compiler.

Pascal

```
function _VarToCurrency(const V: TVarData): Currency;
```

C++

```
Currency _VarToCurrency(const TVarData V);
```

File

Variants

Description

Do not call _VarToCurrency directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToCurrency is not publicly available to Delphi applications.

51.3.54 Variants._VarToDisp Function

_VarToDisp is used internally by the Delphi compiler.

Pascal

```
procedure _VarToDisp(var Dispatch: IDispatch; const V: TVarData);
```

C++

```
_VarToDisp(IDispatch Dispatch, const TVarData V);
```

File

Variants

Description

Do not call _VarToDisp directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToDisp is not publicly available to Delphi applications.

51.3.55 Variants._VarToDynArray Function

_VarToDynArray is used internally by the Delphi compiler.

Pascal

```
procedure _VarToDynArray(var DynArray: Pointer; const V: TVarData; TypeInfo: Pointer);
```

C++

```
_VarToDynArray(void * DynArray, const TVarData V, void * TypeInfo);
```

File

Variants

Description

Do not call _VarToDynArray directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToDynArray is not publicly available to Delphi applications.

51.3.56 Variants._VarToInt64 Function

_VarToInt64 is used internally by the Delphi compiler.

Pascal

```
function _VarToInt64(const V: TVarData): Int64;
```

C++

```
Int64 _VarToInt64(const TVarData V);
```

File

Variants

Description

Do not call _VarToInt64 directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToInt64 is not publicly available to Delphi applications.

51.3.57 Variants._VarToInteger Function

_VarToInteger is used internally by the Delphi compiler.

Pascal

```
function _VarToInteger(const V: TVarData): Integer;
```

C++

```
int _VarToInteger(const TVarData V);
```

File

Variants

Description

Do not call _VarToInteger directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToInteger is not publicly available to Delphi applications.

51.3.58 Variants._VarToIntf Function

_VarToIntf is used internally by the Delphi compiler.

Pascal

```
procedure _VarToIntf(var Intf: IInterface; const V: TVarData);
```

C++

```
_VarToIntf(IInterface Intf, const TVarData V);
```

File

Variants

Description

Do not call _VarToIntf directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToIntf is not publicly available to Delphi applications.

51.3.59 Variants._VarToLStr Function

_VarToLStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarToLStr(var S: AnsiString; const V: TVarData);
```

C++

```
_VarToLStr(AnsiString S, const TVarData V);
```

File

Variants

Description

Do not call _VarToLStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToLStr is not publicly available to Delphi applications.

51.3.60 Variants._VarToPStr Function

_VarToPStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarToPStr(var S; const V: TVarData);
```

C++

```
_VarToPStr( S, const TVarData V);
```

File

Variants

Description

Do not call _VarToPStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToPStr is not publicly available to Delphi applications.

51.3.61 Variants._VarToReal Function

_VarToReal is used internally by the Delphi compiler.

Pascal

```
function _VarToReal(const V: TVarData): Extended;
```

C++

```
Extended _VarToReal(const TVarData V);
```

File

Variants

Description

Do not call _VarToReal directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToReal is not publicly available to Delphi applications.

51.3.62 Variants._VarToUInt64 Function

_VarToUInt64 is used internally by the Delphi compiler.

Pascal

```
function _VarToUInt64(const V: TVarData): UInt64;
```

C++

```
UInt64 _VarToUInt64(const TVarData V);
```

File

Variants

Description

Do not call _VarToUInt64 directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToUInt64 is not publicly available to Delphi applications.

51.3.63 Variants._VarToUStr Function

_VarToUStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarToUStr(var S: UnicodeString; const V: TVarData);
```

C++

```
_VarToUStr(UnicodeString S, const TVarData V);
```

File

Variants

Description

Do not call _VarToUStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToUStr is not publicly available to Delphi applications.

51.3.64 Variants._VarToWStr Function

_VarToWStr is used internally by the Delphi compiler.

Pascal

```
procedure _VarToWStr(var S: WideString; const V: TVarData);
```

C++

```
_VarToWStr(BSTR S, const TVarData V);
```

File

Variants

Description

Do not call _VarToWStr directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarToWStr is not publicly available to Delphi applications.

51.3.65 Variants._VarXor Function

_VarXor is used internally by the Delphi compiler.

Pascal

```
procedure _VarXor(var Left: TVarData; const Right: TVarData);
```

C++

```
_VarXor(TVarData Left, const TVarData Right);
```

File

Variants

Description

Do not call _VarXor directly. The Delphi compiler automatically inserts calls to this function whenever appropriate.

Note: _VarXor is not publicly available to Delphi applications.

51.3.66 Variants.DynArrayFromVariant Function

Creates a dynamic array from a System::Variant (see page 3421).

Pascal

```
procedure DynArrayFromVariant(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer);
```

C++

```
DynArrayFromVariant(void * DynArray, const Variant V, void * TypeInfo);
```

File

Variants

Description

Call DynArrayFromVariant to copy the contents of a variant (V) to an existing dynamic array variable (DynArray).

DynArray is a dynamic array variable whose value is assigned from the System::Variant (see page 3421).

V is the System::Variant (see page 3421) to copy. Its value must be an array.

TypeInfo points to the runtime type information that identifies the type of array elements. You can obtain this pointer using the TypeInfo function (Delphi) or the __delphirtti function (C++).

See Also

DynArrayToVariant (see page 5054)

51.3.67 Variants.DynArrayToVariant Function

Creates a System::Variant (see page 3421) array from a dynamic array.

Pascal

```
procedure DynArrayToVariant(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer);
```

C++

```
DynArrayToVariant(Variant V, const void * DynArray, void * TypeInfo);
```

File

Variants

Description

Call DynArrayToVariant to assign the contents of a dynamic array as the value of a System::Variant (see page 3421) variable.

V is the System::Variant (see page 3421) whose value is assigned by DynArrayToVariant.

DynArray is a dynamic array to copy.

TypeInfo points to the runtime type information that identifies the type of array elements. You can obtain this pointer using the TypeInfo function (Delphi) or the __delphirtti function (C++).

See Also

DynArrayFromVariant (see page 5054)

51.3.68 Variants.FindCustomVariantType Function

Retrieves the object that implements a custom System::Variant (see page 3421) type.

Pascal

```
function FindCustomVariantType(const AVarType: TVarType; out CustomVariantType:
TCustomVariantType): Boolean; overload;
function FindCustomVariantType(const TypeName: string; out CustomVariantType:
TCustomVariantType): Boolean; overload;
```

C++

```
Boolean FindCustomVariantType(const TVarType AVarType, TCustomVariantType CustomVariantType);
Boolean FindCustomVariantType(const UnicodeString TypeName, TCustomVariantType
CustomVariantType);
```

File

Variants

Description

Use FindCustomVariantType to obtain a reference to the object that implements a custom System::Variant (see page 3421) type.

TypeName is the name of the TCustomVariantType (see page 4987) descendant class you want to retrieve.

AVarType is the type code of System::Variant (see page 3421) whose implementation class you want to retrieve. If you have an instance of a System::Variant (see page 3421) whose value is a custom System::Variant (see page 3421) Type, you can obtain a value for this parameter using the VarType (see page 5090) function.

CustomVariantType returns an instance of the TCustomVariantType (see page 4987) descendant that implements the indicated custom System::Variant (see page 3421) type.

See Also

VarType (see page 5090)

VarIsCustom (see page 5080)

51.3.69 Variants.FindVarData Function

Returns pointer to variant's data.

Pascal

```
function FindVarData(const V: Variant): PVarData;
```

C++

```
PVarData FindVarData(const Variant V);
```

File

Variants

Description

FindVarData returns a pointer to the System::Variant (see page 3421) data contained referenced by a System::Variant (see page 3421). If the System::Variant (see page 3421) contains another System::Variant (see page 3421), FindVarData follows the chain of references until it finds the actual data.

See Also

VarType (see page 5090)

51.3.70 Variants.HandleConversionException Function

Handles exceptions during variant type conversions.

Pascal

```
procedure HandleConversionException(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
HandleConversionException(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

HandleConversionException handles exceptions resulting from attempts to convert between variant types. Code that implements such conversions should call HandleConversionException when a conversion throws an exception.

ASourceType and ADestType indicate the variant types of the source and destination. HandleConversionException uses these to format an appropriate error string. If the current exception corresponds to a variant exception, HandleConversionException raises that exception. Otherwise, HandleConversionException simply re-raises the original exception.

See Also

VarResultCheck (see page 5086)

EVariantError (see page 3948)

51.3.71 Variants.Null Function

Returns a Null variant.

Pascal

```
function Null: Variant;
```

C++

```
Variant Null();
```

File

Variants

Description

Use Null to obtain a Null variant that can indicate unknown or missing data. Null Variants (see page 4973) can be assigned to variant variables in an application that must contain a null value. Assigning Null to a variant variable does not cause an error, and Null can be returned from any function with a variant return value.

Assigning Null to a variable of any type other than System::Variant (see page 3421) causes either a compile-time error or raises an EVariantTypeMismatch exception. For example, in the following code the assignment of v, the Null variant, to variant q is successful; whereas the conversion of variant v, which is now Null, to the integer return type of the Test function, raises an exception.

function Test(v: System::Variant (see page 3421)): Integer;

```
var
  q: Variant;
  msg: string;
begin
  q := v; { this is ok, since q is a variant }
  if VarIsNull(q) then
    msg := 'q is a null variant'
  else
    msg := 'q is not a null variant';
  ShowMessage(msg);

  Result := v; { this raises an exception!!!}
```

end;

procedure TForm1.Button1Click(Sender: TObject (see page 3354));

```
begin
  Test(Null);
```

end;

int Test(Variant (see page 3421) v)

```
{
  Variant q = v; // this is ok, since q is a variant
  AnsiString msg;
  if (VarIsNull(q))
    msg = "q is a null variant";
  else
    msg = "q is not a null variant";
  ShowMessage(msg);

   return v; // this throws an exception!!!
```

}

void __fastcall TForm1::Button1Click(TObject (see page 3354) *Sender)

{

```
  Test(Null());
```

}

Except for comparisons, which always result in Boolean values, expressions involving Null variants always result in Null variants.

Note: The Null variant returned by the Null function is different from the nil constant in Delphi code and the NULL macro in C++ code. Neither nil nor NULL is a variant. The nil constant is an object reference that does not refer to any actual object. The NULL macro is a pointer that does not refer to any actual object or value.

See Also

Unassigned (see page 5058)

VarIsNull (see page 5083)

VarType (see page 5090)

NullEqualityRule (see page 5103)

NullMagnitudeRule (see page 5103)

51.3.72 Variants.Unassigned Function

Returns an "empty" variant.

Pascal

```
function Unassigned: Variant;
```

C++

```
Variant Unassigned();
```

File

Variants

Description

A variant variable can be "empty", meaning it has not yet been assigned to. The Unassigned function returns an empty variant, which can be assigned to a variant variable to restore the variable to its initial state.

Use the VarIsEmpty (see page 5080) function to test whether a variant is empty. When used on an empty variant, the VarType (see page 5090) standard function returns varEmpty (see page 3865).

If an empty variant is cast to another type (for example, by assigning to a non-variant variable or calling VarAsType (see page 5073)) the following conversions occur:

Destination	Conversion result
Numeric type	Zero
String type	Empty string
Boolean	False.

Note: Unassigned is useful with variants referencing OLE Automation Objects that you want to keep "alive" until another value is assigned to the variant.

There's corresponding functionality under Linux. Should document when time permits.

See Also

Null (see page 5056)

VarIsEmpty (see page 5080)

VarType (see page 5090)

VarAsType (see page 5073)

51.3.73 Variants.VarArrayAsPSafeArray Function

Pascal

```
function VarArrayAsPSafeArray(const A: Variant): PVarArray;
```

C++

```
PVarArray VarArrayAsPSafeArray(const Variant A);
```

File

Variants

Description

This is function Variants.VarArrayAsPSafeArray.

51.3.74 Variants.VarArrayCreate Function

Creates a variant array.

Pascal

```
function VarArrayCreate(const Bounds: array of Integer; AVarType: TVarType): Variant;
```

C++

```
Variant VarArrayCreate(const array of Integer Bounds, TVarType AVarType);
```

File

Variants

Description

VarArrayCreate creates a variant array with the bounds given by Bounds (see page 4651) and the element type given by VarType (see page 5090).

The Bounds (see page 4651) parameter must contain an even number of values, where each pair of values specifies the upper and lower bounds of one dimension of the array.

Note: The Bounds_size parameter is the index of the last value in the Bounds (see page 4651) array. This is one less than the total number of values. Thus, Bounds_size must always be an odd number.

The element type of the array, given by the VarType (see page 5090) parameter, is a variant type code. This must be one of the constants defined in the System (see page 3159) unit. It cannot include the varArray (see page 3861) or varByRef (see page 3862) bits. The element type cannot be varString (see page 3871) or a custom System::Variant (see page 3421) type. To create a variant array of strings use the varOleStr (see page 3869) type code. If the element type is varVariant (see page 3873), the elements of the array are themselves variants and can in turn contain variant arrays.

If VarArrayCreate is unable to create the array, an EVariantArrayCreateError (see page 4978) is raised.

Note: :Variant (see page 3421) arrays with an element type of varByte (see page 3863) are the preferred method of passing binary data between OLE Automation controllers and servers. Such arrays are subject to no translation of their data, and can be efficiently accessed using the VarArrayLock (see page 5067) and VarArrayUnlock (see page 5072) routines.

There's corresponding functionality under Linux. Should document when time permits.

C++ Examples:

```
/*
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Variant array;
    int bounds[2] = {0, 9};

    /*
    Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer
    */
    array = VarArrayCreate(bounds, 1, varInteger);

    // Increase the length of the variant array
    VarArrayRedim(array, 49);

    MessageDlg(String("Variant array has ") + IntToStr(VarArrayDimCount(array)) +
        " dimensions", mtInformation, TMsgDlgButtons() << mbOK, 0);

    // Traverse the array from lower to higher bounds
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        VarArrayPut(array, i, &i, 0);
    }

    // Now read the elements of the array
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        if (VarArrayGet(array, &i, 0) != i)
            MessageDlg("Error! Invalid element found at current position!",
                mtError, TMsgDlgButtons() << mbOK, 0);
    }

    // Clear the variant array
    VarClear(array);
}
```

Delphi Examples:

```
{
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Arr: Variant;
  I: Integer;
begin
  { Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer }
  Arr := VarArrayCreate([0, 9], varInteger);

  { Increase the length of the variant array }
  VarArrayRedim(Arr, 49);

  MessageDlg('Variant array has ' + IntToStr(VarArrayDimCount(Arr)) + ' dimensions',
      mtInformation, [mbOK], 0);

  { Traverse the array from lower to higher bounds }
```

```
    for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
    begin
      { Put the element I at index I}
      VarArrayPut(Arr, I, [I]);
    end;

    { Now read the elements of the array }
    for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
    begin
      { Put the element I at index I}
      if VarArrayGet(Arr, [I]) <> I then
        MessageDlg('Error! Invalid element found at current position!', mtError, [mbOK], 0);
    end;

    { Clear the variant array }
    VarClear(Arr);
  end;
```

See Also

VarArrayOf (see page 5069)

VarArrayLock (see page 5067)

VarArrayUnlock (see page 5072)

EVariantArrayCreateError (see page 4978)

51.3.75 Variants.VarArrayCreateError Function

Indicates an array creation error.

Pascal

```
procedure VarArrayCreateError;
```

C++

```
VarArrayCreateError();
```

File

Variants

See Also

EVariantArrayCreateError (see page 4978)

51.3.76 Variants.VarArrayDimCount Function

Returns number of dimensions of a variant array.

Pascal

```
function VarArrayDimCount(const A: Variant): Integer;
```

C++

```
int VarArrayDimCount(const Variant A);
```

File

Variants

Description

VarArrayDimCount returns the number of dimensions of the given variant. The resulting value is zero if the given variant is not a variant array.

C++ Examples:

```
/*
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Variant array;
    int bounds[2] = {0, 9};

    /*
    Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer
    */
    array = VarArrayCreate(bounds, 1, varInteger);

    // Increase the length of the variant array
    VarArrayRedim(array, 49);

    MessageDlg(String("Variant array has ") + IntToStr(VarArrayDimCount(array)) +
        " dimensions", mtInformation, TMsgDlgButtons() << mbOK, 0);

    // Traverse the array from lower to higher bounds
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        VarArrayPut(array, i, &i, 0);
    }

    // Now read the elements of the array
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        if (VarArrayGet(array, &i, 0) != i)
            MessageDlg("Error! Invalid element found at current position!",
                mtError, TMsgDlgButtons() << mbOK, 0);
    }

    // Clear the variant array
    VarClear(array);
}
```

Delphi Examples:

```
{
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Arr: Variant;
  I: Integer;
begin
  { Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer }
  Arr := VarArrayCreate([0, 9], varInteger);
```

```
    { Increase the length of the variant array }
    VarArrayRedim(Arr, 49);

    MessageDlg('Variant array has ' + IntToStr(VarArrayDimCount(Arr)) + ' dimensions',
        mtInformation, [mbOK], 0);

    { Traverse the array from lower to higher bounds }
    for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
    begin
      { Put the element I at index I}
      VarArrayPut(Arr, I, [I]);
    end;

    { Now read the elements of the array }
    for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
    begin
      { Put the element I at index I}
      if VarArrayGet(Arr, [I]) <> I then
        MessageDlg('Error! Invalid element found at current position!', mtError, [mbOK], 0);
    end;

    { Clear the variant array }
    VarClear(Arr);
  end;
```

See Also

VarArrayHighBound (see page 5065)

VarArrayLowBound (see page 5067)

51.3.77 Variants.VarArrayGet Function

Returns a System::Variant (see page 3421) that represents a single value from a multi-dimensional System::Variant (see page 3421) array.

Pascal

```
function VarArrayGet(const A: Variant; const Indices: array of Integer): Variant;
```

C++

```
Variant VarArrayGet(const Variant A, const array of Integer Indices);
```

File

Variants

Description

Use VarArrayGet to obtain the value of a single cell from a multi-dimensional System::Variant (see page 3421) array.

A is the System::Variant (see page 3421) array that contains the desired value.

Indices is an array of index values, one for each dimension of the array.

Note: Indices_Size is the index of the last value in Indices (one less than the number of dimensions in the array).

If the values specified by Indices are out of bounds, VarArrayGet raises an EVariantBadIndexError (see page 4980).

C++ Examples:

```
/*
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
```

```
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Variant array;
    int bounds[2] = {0, 9};

    /*
    Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer
    */
    array = VarArrayCreate(bounds, 1, varInteger);

    // Increase the length of the variant array
    VarArrayRedim(array, 49);

    MessageDlg(String("Variant array has ") + IntToStr(VarArrayDimCount(array)) +
        " dimensions", mtInformation, TMsgDlgButtons() << mbOK, 0);

    // Traverse the array from lower to higher bounds
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        VarArrayPut(array, i, &i, 0);
    }

    // Now read the elements of the array
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        if (VarArrayGet(array, &i, 0) != i)
            MessageDlg("Error! Invalid element found at current position!",
                mtError, TMsgDlgButtons() << mbOK, 0);
    }

    // Clear the variant array
    VarClear(array);
}
```

Delphi Examples:

```
{
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Arr: Variant;
  I: Integer;
begin
  { Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer }
  Arr := VarArrayCreate([0, 9], varInteger);

  { Increase the length of the variant array }
  VarArrayRedim(Arr, 49);

  MessageDlg('Variant array has ' + IntToStr(VarArrayDimCount(Arr)) + ' dimensions',
      mtInformation, [mbOK], 0);

  { Traverse the array from lower to higher bounds }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    VarArrayPut(Arr, I, [I]);
  end;

  { Now read the elements of the array }
```

```
    for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
    begin
      { Put the element I at index I}
      if VarArrayGet(Arr, [I]) <> I then
        MessageDlg('Error! Invalid element found at current position!', mtError, [mbOK], 0);
    end;

    { Clear the variant array }
    VarClear(Arr);
  end;
```

See Also

VarArrayPut (see page 5069)

EVariantBadIndexError (see page 4980)

51.3.78 Variants.VarArrayHighBound Function

Returns high bound for a dimension in a variant array.

Pascal

```
function VarArrayHighBound(const A: Variant; Dim: Integer): Integer;
```

C++

```
int VarArrayHighBound(const Variant A, int Dim);
```

File

Variants

Description

VarArrayHighBound returns the high bound of the given dimension in the given variant array. The Dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An exception is raised if the variant given by A is not an array (EVariantInvalidArgError (see page 4981)), or if the dimension specified by Dim does not exist (EVariantBadIndexError (see page 4980)).

C++ Examples:

```
/*
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Variant array;
    int bounds[2] = {0, 9};

    /*
    Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer
    */
    array = VarArrayCreate(bounds, 1, varInteger);

    // Increase the length of the variant array
    VarArrayRedim(array, 49);

    MessageDlg(String("Variant array has ") + IntToStr(VarArrayDimCount(array)) +
        " dimensions", mtInformation, TMsgDlgButtons() << mbOK, 0);

    // Traverse the array from lower to higher bounds
```

```
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        VarArrayPut(array, i, &i, 0);
    }

    // Now read the elements of the array
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        if (VarArrayGet(array, &i, 0) != i)
            MessageDlg("Error! Invalid element found at current position!",
                mtError, TMsgDlgButtons() << mbOK, 0);
    }

    // Clear the variant array
    VarClear(array);
}
```

Delphi Examples:

```
{
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Arr: Variant;
  I: Integer;
begin
  { Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer }
  Arr := VarArrayCreate([0, 9], varInteger);

  { Increase the length of the variant array }
  VarArrayRedim(Arr, 49);

  MessageDlg('Variant array has ' + IntToStr(VarArrayDimCount(Arr)) + ' dimensions',
      mtInformation, [mbOK], 0);

  { Traverse the array from lower to higher bounds }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    VarArrayPut(Arr, I, [I]);
  end;

  { Now read the elements of the array }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    if VarArrayGet(Arr, [I]) <> I then
      MessageDlg('Error! Invalid element found at current position!', mtError, [mbOK], 0);
  end;

  { Clear the variant array }
  VarClear(Arr);
end;
```

See Also

VarArrayDimCount (see page 5061)

VarArrayLowBound (see page 5067)

EVariantInvalidArgError (see page 4981)

EVariantBadIndexError (see page 4980)

51.3.79 Variants.VarArrayLock Function

Locks a variant array and returns a pointer to the data.

Pascal

```
function VarArrayLock(const A: Variant): Pointer;
```

C++

```
void * VarArrayLock(const Variant A);
```

File

Variants

Description

VarArrayLock locks the given variant array and returns a pointer to the data stored in the array. While locked, a variant array can not be resized and any calls to VarArrayRedim (see page 3702) will fail. Once locked, a variant array must later be unlocked using VarArrayUnlock (see page 5072). An EVariantInvalidArgError (see page 4981) exception is raise if the variant given by A is not an array.

The pointer returned by VarArrayLock points to an array of elements in which the leftmost dimension increases first. In other words, the dimensions of the returned array pointer are reversed from the dimensions of the variant array.

Once it is verified that a variant array has the correct bounds and dimensions, VarArrayLock can be used to gain direct access to the array data, which results in improved performance.

See Also

VarArrayUnlock (see page 5072)

VarArrayRedim (see page 3702)

51.3.80 Variants.VarArrayLowBound Function

Returns the low bound of a dimension in a variant array.

Pascal

```
function VarArrayLowBound(const A: Variant; Dim: Integer): Integer;
```

C++

```
int VarArrayLowBound(const Variant A, int Dim);
```

File

Variants

Description

VarArrayLowBound returns the low bound of the given dimension in the given variant array. The Dim parameter should be 1 for the leftmost dimension, 2 for the second leftmost dimension, and so on. An exception is raised if the variant given by A is not an array (EVariantInvalidArgError (see page 4981)), or if the dimension specified by Dim does not exist (EVariantBadIndexError (see page 4980))..

C++ Examples:

```
/*
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Variant array;
    int bounds[2] = {0, 9};

    /*
    Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer
    */
    array = VarArrayCreate(bounds, 1, varInteger);

    // Increase the length of the variant array
    VarArrayRedim(array, 49);

    MessageDlg(String("Variant array has ") + IntToStr(VarArrayDimCount(array)) +
        " dimensions", mtInformation, TMsgDlgButtons() << mbOK, 0);

    // Traverse the array from lower to higher bounds
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        VarArrayPut(array, i, &i, 0);
    }

    // Now read the elements of the array
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        if (VarArrayGet(array, &i, 0) != i)
            MessageDlg("Error! Invalid element found at current position!",
                mtError, TMsgDlgButtons() << mbOK, 0);
    }

    // Clear the variant array
    VarClear(array);
}
```

Delphi Examples:

```
{
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Arr: Variant;
  I: Integer;
begin
  { Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer }
  Arr := VarArrayCreate([0, 9], varInteger);

  { Increase the length of the variant array }
  VarArrayRedim(Arr, 49);

  MessageDlg('Variant array has ' + IntToStr(VarArrayDimCount(Arr)) + ' dimensions',
      mtInformation, [mbOK], 0);

  { Traverse the array from lower to higher bounds }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    VarArrayPut(Arr, I, [I]);
```

```
    end;

    { Now read the elements of the array }
    for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
    begin
      { Put the element I at index I}
      if VarArrayGet(Arr, [I]) <> I then
        MessageDlg('Error! Invalid element found at current position!', mtError, [mbOK], 0);
    end;

    { Clear the variant array }
    VarClear(Arr);
  end;
```

See Also

VarArrayDimCount (see page 5061)

VarArrayHighBound (see page 5065)

EVariantInvalidArgError (see page 4981)

EVariantBadIndexError (see page 4980)

51.3.81 Variants.VarArrayOf Function

Creates and fills a one-dimensional variant array.

Pascal

```
function VarArrayOf(const Values: array of Variant): Variant;
```

C++

```
Variant VarArrayOf(const array of Variant Values);
```

File

Variants

Description

VarArrayOf returns a one-dimensional variant array with the elements given by the Values parameter

Values is an array of System::Variant (see page 3421) values to convert into a System::Variant (see page 3421) array.

Note: Values_Size indicates the index of the last value in the Values array (one less than the number of values).

The low bound of the returned array is zero, the high bound is the number of values given by the Values parameter less one (Values_Size), and the element type is System::Variant (see page 3421).

See Also

VarArrayCreate (see page 5059)

51.3.82 Variants.VarArrayPut Function

Sets the value of a single cell in a multi-dimensional System::Variant (see page 3421) array.

Pascal

```
procedure VarArrayPut(var A: Variant; const Value: Variant; const Indices: array of Integer);
```

C++

```
VarArrayPut(Variant A, const Variant Value, const array of Integer Indices);
```

File

Variants

Description

Use VarArrayPut to set the value of a single cell in a multi-dimensional System::Variant (see page 3421) array.

A is the System::Variant (see page 3421) array that contains the cell to set.

Value is the value to assign to the specified cell.

Indices is an array of index values, one for each dimension of the array.

Note: Indices_Size is the index of the last value in Indices (one less than the number of dimensions in the array).

If the values specified by Indices are out of bounds, VarArrayPut raises an EVariantBadIndexError (see page 4980).

C++ Examples:

```
/*
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
*/
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Variant array;
    int bounds[2] = {0, 9};

    /*
    Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer
    */
    array = VarArrayCreate(bounds, 1, varInteger);

    // Increase the length of the variant array
    VarArrayRedim(array, 49);

    MessageDlg(String("Variant array has ") + IntToStr(VarArrayDimCount(array)) +
        " dimensions", mtInformation, TMsgDlgButtons() << mbOK, 0);

    // Traverse the array from lower to higher bounds
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        VarArrayPut(array, i, &i, 0);
    }

    // Now read the elements of the array
    for (int i = VarArrayLowBound(array, 1); i <= VarArrayHighBound(array, 1); i++)
    {
        // Put the element I at index I
        if (VarArrayGet(array, &i, 0) != i)
            MessageDlg("Error! Invalid element found at current position!",
                mtError, TMsgDlgButtons() << mbOK, 0);
    }

    // Clear the variant array
    VarClear(array);
}
```

Delphi Examples:

```
{
This example demonstrates the use of variant arrays. In this
example a variant array is created, modified, populated and then freed.
}
procedure TForm1.Button1Click(Sender: TObject);
var
  Arr: Variant;
  I: Integer;
begin
  { Create a variant array of 10 elements, starting at 0
    and ending at 9. The array contains elements of type integer }
  Arr := VarArrayCreate([0, 9], varInteger);

  { Increase the length of the variant array }
  VarArrayRedim(Arr, 49);

  MessageDlg('Variant array has ' + IntToStr(VarArrayDimCount(Arr)) + ' dimensions',
      mtInformation, [mbOK], 0);

  { Traverse the array from lower to higher bounds }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    VarArrayPut(Arr, I, [I]);
  end;

  { Now read the elements of the array }
  for I := VarArrayLowBound(Arr, 1) to VarArrayHighBound(Arr, 1) do
  begin
    { Put the element I at index I}
    if VarArrayGet(Arr, [I]) <> I then
      MessageDlg('Error! Invalid element found at current position!', mtError, [mbOK], 0);
  end;

  { Clear the variant array }
  VarClear(Arr);
end;
```

See Also

VarArrayGet (see page 5063)

EVariantBadIndexError (see page 4980)

51.3.83 Variants.VarArrayRef Function

Returns a reference to the specified variant array.

Pascal

```
function VarArrayRef(const A: Variant): Variant;
```

C++

```
Variant VarArrayRef(const Variant A);
```

File

Variants

Description

Use VarArrayRef to reference a variant array when a referenced variant array is expected in an API call. VarArrayRef creates a

new variant array of the same type as A, but with an extra level of indirection if A does not include the varByRef (see page 3862) type bit. The returned variant includes the varByRef (see page 3862) type bit.

If A is not an array, VarArrayRef raises an EVariantInvalidArgError (see page 4981) exception.

See Also

VarArrayCreate (see page 5059)

TVarType (see page 3801)

VarCopyNoInd (see page 5075)

EVariantInvalidArgError (see page 4981)

51.3.84 Variants.VarArrayUnlock Function

Unlocks a variant array.

Pascal

```
procedure VarArrayUnlock(const A: Variant);
```

C++

```
VarArrayUnlock(const Variant A);
```

File

Variants

Description

VarArrayUnlock unlocks a variant array that was previously locked by VarArrayLock (see page 5067). If the variant given by A is not an array, VarArrayUnlock raises an EVariantInvalidArtgError exception.

See Also

VarArrayLock (see page 5067)

51.3.85 Variants.VarAsError Function

Converts a HRESULT (see page 3754) into an Error (see page 3560) variant.

Pascal

```
function VarAsError(AResult: HRESULT): Variant;
```

C++

```
Variant VarAsError(HRESULT AResult);
```

File

Variants

Description

VarAsError converts an HRESULT (see page 3754) result code into an equivalent variant of type varError (see page 3866).

See Also

VarType (see page 5090)

51.3.86 Variants.VarAsType Function

Converts a variant to specified type.

Pascal

```
function VarAsType(const V: Variant; AVarType: TVarType): Variant;
```

C++

```
Variant VarAsType(const Variant V, TVarType AVarType);
```

File

Variants

Description

VarAsType converts a variant to a specified type and returns a new System::Variant (see page 3421) that has the specified type.

V is the System::Variant (see page 3421) to convert.

VarType (see page 5090) is a variant type code that indicates the type to which V should be converted. This can be one of the constants defined in the System (see page 3159) unit, or it can be the type code for a custom variant type. It cannot include the varArray (see page 3861) or varByRef (see page 3862) bits.

VarAsType raises an exception if the variant cannot be converted to the given type. This will be EVariantError (see page 3948) or one of its descendants, such as EVariantOverflowError (see page 4985) or EVariantTypeCastError (see page 4986).

See Also

VarType (see page 5090)

51.3.87 Variants.VarCastError Function

Raises an EVariantTypeCastError (see page 4986) exception.

Pascal

```
procedure VarCastError; overload;
procedure VarCastError(const ASourceType: TVarType; const ADestType: TVarType); overload;
```

C++

```
VarCastError();
VarCastError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarCastError to indicate that a variant operation has failed due to cast error. VarCastError raises EVariantTypeCastError (see page 4986). If no arguments are specified, the error string is a general error message. If ASourceType and ADestType are

specified, the error string incorporates the names of the specified System::Variant (see page 3421) Types (see page 4649).

See Also

EVariantTypeCastError (see page 4986)

TVarData (see page 3740)

VarType (see page 5090)

51.3.88 Variants.VarCheckEmpty Function

Raises an exception if a specified variant's value is Unassigned (see page 5058).

Pascal

```
procedure VarCheckEmpty(const V: Variant);
```

C++

```
VarCheckEmpty(const Variant V);
```

File

Variants

Description

Call VarCheckEmpty when your code requires that a System::Variant (see page 3421) have a value, and you want to raise an exception if it doesn't. VarCheckEmpty checks whether the System::Variant (see page 3421) specified by V has a value of Unassigned (see page 5058). If so, it raises an EVariantTypeMismatchError exception.

To check whether the System::Variant (see page 3421) is Unassigned (see page 5058) without raising an exception, use the VarIsEmpty (see page 5080) function instead. To check whether the System::Variant (see page 3421)'s value is undefined, use the VarIsClear (see page 5079) function instead.

Note: Do not confuse an Unassigned (see page 5058) variant with a System::Variant (see page 3421) whose value is Null (see page 5056). A Null (see page 5056) variant is still assigned, but has the value Null (see page 5056). Unlike unassigned variants, Null (see page 5056) variants can be used in expressions and can be converted to other types of variants.

See Also

VarIsEmpty (see page 5080)

Unassigned (see page 5058)

VarType (see page 5090)

VarIsNull (see page 5083)

VarIsClear (see page 5079)

EVariantTypeCastError (see page 4986)

51.3.89 Variants.VarCompareValue Function

Returns the relationship between Variants (see page 4973).

Pascal

```
function VarCompareValue(const A: Variant; const B: Variant): TVariantRelationship;
```

C++

```
TVariantRelationship VarCompareValue(const Variant A, const Variant B);
```

File

Variants

Description

Call VarCompareValue to determine the relationship between two Variants (see page 4973).

A and B are the values to compare.

VarCompareValue returns a value that indicates whether A and B are equal (vrEqual), whether one is greater than the other (vrLessThan or vrGreaterThan), or whether the two values can't be ordered relative to each other (vrNotEqual).

See Also

VarSameValue (see page 5087)

CompareValue

51.3.90 Variants.VarCopyNoInd Function

Copies a System::Variant (see page 3421).

Pascal

```
procedure VarCopyNoInd(var Dest: Variant; const Source: Variant);
```

C++

```
VarCopyNoInd(Variant Dest, const Variant Source);
```

File

Variants

Description

VarCopyNoInd copies the variant given by Source into the variant given by Dest. Calling VarCopyNoInd in C++ or Delphi code is equivalent to a simple variant assignment in Delphi code.

Dest can be a System::Variant (see page 3421) or an OleVariant (see page 3274), and it must be possible to assign a value to it.

Source is defined as a System::Variant (see page 3421), but an OleVariant (see page 3274) is also allowed because it is a compatible type. However, Source can't have a VarType (see page 5090) that includes the varByRef (see page 3862) bit.

The following table lists the expected behavior when using System::Variant (see page 3421) and OleVariant (see page 3274) types as parameters:

Dest	Source	Behavior
Variant (see page 3421)	OleVariant (see page 3274)	Works as expected.
Variant (see page 3421)	Variant (see page 3421)	Works as expected.

OleVariant (see page 3274)	OleVariant (see page 3274)	Works as expected.
OleVariant (see page 3274)	Variant (see page 3421)	Source should be implicitly cast to an OleVariant (see page 3274).

51.3.91 Variants.VarEnsureRange Function

Returns a System::Variant (see page 3421) with a value in a specified range.

Pascal

```
function VarEnsureRange(const AValue: Variant; const AMin: Variant; const AMax: Variant): Variant;
```

C++

```
Variant VarEnsureRange(const Variant AValue, const Variant AMin, const Variant AMax);
```

File

Variants

Description

VarEnsureRange returns a System::Variant (see page 3421) whose value is guaranteed to be within a specified range.

AValue is a System::Variant (see page 3421) with the value you want as long as it is in range. If AValue is in range, VarEnsureRange returns AValue.

AMin is a System::Variant (see page 3421) with the minimum acceptable value. If AValue is less than AMin, VarEnsureRange returns AMin.

AMax is a System::Variant (see page 3421) with the maximum acceptable value. If AValue is greater than AMax, VarEnsureRange returns AMax.

See Also

VarInRange (see page 5077)

51.3.92 Variants.VarFromDateTime Function

Returns a variant containing a specified data and time.

Pascal

```
function VarFromDateTime(const DateTime: TDateTime): Variant;
```

C++

```
Variant VarFromDateTime(const TDateTime DateTime);
```

File

Variants

Description

VarFromDateTime returns a System::Variant (see page 3421) that represents the date and time of the given TDateTime (see page 3323) value, DateTime. The type code of the resulting variant is varDate (see page 3864).

See Also

VarToDateTime (see page 5088)

VarType (see page 5090)

51.3.93 Variants.VarInRange Function

Indicates whether a System::Variant (see page 3421)'s value is in a specified range.

Pascal

```
function VarInRange(const AValue: Variant; const AMin: Variant; const AMax: Variant): Boolean;
```

C++

```
Boolean VarInRange(const Variant AValue, const Variant AMin, const Variant AMax);
```

File

Variants

Description

VarInRange checks whether a System::Variant (see page 3421) has a value in a specified range.

AValue is a System::Variant (see page 3421) to check.

AMin is a System::Variant (see page 3421) with the minimum acceptable value.

AMax is a System::Variant (see page 3421) with the maximum acceptable value.

VarInRange returns true if AValue is greater than or equal to AMin and less than or equal to AMax. Otherwise, it returns false.

See Also

VarEnsureRange (see page 5076)

51.3.94 Variants.VarInvalidNullOp Function

Indicate variant operation on null.

Pascal

```
procedure VarInvalidNullOp;
```

C++

```
VarInvalidNullOp();
```

File

Variants

See Also

EVariantInvalidOpError (see page 4983)

51.3.95 Variants.VarInvalidOp Function

Indicate invalid variant operation.

Pascal

```
procedure VarInvalidOp;
```

C++

```
VarInvalidOp();
```

File

Variants

See Also

EVariantInvalidOpError (see page 4983)

51.3.96 Variants.VarIsArray Function

Indicates whether the specified variant is an array.

Pascal

```
function VarIsArray(const A: Variant): Boolean; overload;
function VarIsArray(const A: Variant; AResolveByRef: Boolean): Boolean; overload;
```

C++

```
Boolean VarIsArray(const Variant A);
Boolean VarIsArray(const Variant A, Boolean AResolveByRef);
```

File

Variants

Description

VarIsArray returns true if the given variant is an array. Otherwise, the function result is false.

A is the System::Variant (see page 3421) to check.

AResolveByRef indicates how VarIsArray should handle the case when A is a reference to a System::Variant (see page 3421) (when the VarType (see page 5090) of A is varByRef (see page 3862) or varVariant (see page 3873)).When AResolveByRef is true, VarIsArray checks whether A is an array or A points to an array. When AResolveByRef is false, VarIsArray returns false when A has a VarType (see page 5090) of varByRef (see page 3862) or varVariant (see page 3873).

See Also

VarArrayCreate (see page 5059)

VarType (see page 5090)

VarIsType (see page 5085)

51.3.97 Variants.VarIsByRef Function

Indicates whether the specified variant's value is by reference.

Pascal

```
function VarIsByRef(const V: Variant): Boolean;
```

C++

```
Boolean VarIsByRef(const Variant V);
```

File

Variants

Description

VarIsByRef returns true if the given variant's type code includes the includes the varByRef (see page 3862) bit. Otherwise, the function result is false.

See Also

VarType (see page 5090)

VarIsType (see page 5085)

51.3.98 Variants.VarIsClear Function

Indicates whether the specified variant has an undefined value.

Pascal

```
function VarIsClear(const V: Variant): Boolean;
```

C++

```
Boolean VarIsClear(const Variant V);
```

File

Variants

Description

VarIsClear returns true if the given variant's value is undefined. The value can be undefined for any of several reasons:

The System::Variant (see page 3421) may have had its value set to Unassigned (see page 5058).

The System::Variant (see page 3421)'s value may be an interface type that has been set to nil (Delphi) or NULL (C++).

The System::Variant (see page 3421) may be a custom variant that returns true from its IsClear method.

In all other cases, the function result is false.

Note: Do not confuse an unassigned variant with a Null (see page 5056) variant. A Null (see page 5056) variant is still assigned, but has the value Null (see page 5056). Unlike unassigned variants, Null (see page 5056) variants can be used in expressions and can be converted to other types of variants.

See Also

Unassigned (see page 5058)

VarType (see page 5090)

VarCheckEmpty (see page 5074)

VarIsNull (see page 5083)

VarIsEmpty (see page 5080)

VarIsType (see page 5085)

TCustomVariantType (see page 4987)

51.3.99 Variants.VarIsCustom Function

Indicates whether the specified variant is a custom variant.

Pascal

```
function VarIsCustom(const V: Variant): Boolean;
```

C++

```
Boolean VarIsCustom(const Variant V);
```

File

Variants

Description

VarIsCustom returns true if the given variant represents a custom variant type. Otherwise, the function result is false.

See Also

VarType (see page 5090)

FindCustomVariantType (see page 5055)

VarIsType (see page 5085)

51.3.100 Variants.VarIsEmpty Function

Indicates whether the specified variant is unassigned.

Pascal

```
function VarIsEmpty(const V: Variant): Boolean;
```

C++

```
Boolean VarIsEmpty(const Variant V);
```

File

Variants

Description

VarIsEmpty returns true if the given variant contains the value Unassigned (see page 5058). If the variant contains any other value, the function result is false.

Note: VarIsEmpty does not indicate when a System::Variant (see page 3421) that represents an interface has had the

interface pointer cleared or when a custom System::Variant (see page 3421) is clear. If either of these situations may exist, use the VarIsClear (see page 5079) method instead.

Note: Do not confuse an unassigned variant with a Null (see page 5056) variant. A Null (see page 5056) variant is still assigned, but has the value Null (see page 5056). Unlike unassigned variants, Null (see page 5056) variants can be used in expressions and can be converted to other types of variants.

See Also

Unassigned (see page 5058)

VarType (see page 5090)

VarCheckEmpty (see page 5074)

VarIsNull (see page 5083)

VarIsClear (see page 5079)

VarIsEmptyParam (see page 5081)

VarIsType (see page 5085)

51.3.101 Variants.VarIsEmptyParam Function

Indicates whether the specified variant represents an unassigned optional parameter.

Pascal

```
function VarIsEmptyParam(const V: Variant): Boolean;
```

C++

```
Boolean VarIsEmptyParam(const Variant V);
```

File

Variants

Description

VarIsEmptyParam returns true if the given variant represents an unassigned optional parameter. If the variant contains any other value, the function result is false.

See Also

VarIsClear (see page 5079)

VarIsEmpty (see page 5080)

VarIsNull (see page 5083)

EmptyParam (see page 5102)

VarIsType (see page 5085)

51.3.102 Variants.VarIsError Function

Indicates whether the specified variant is an error variant; optionally converts variant back to HRESULT (see page 3754).

Pascal

```
function VarIsError(const V: Variant; out AResult: HRESULT): Boolean; overload;
function VarIsError(const V: Variant): Boolean; overload;
```

C++

```
Boolean VarIsError(const Variant V, HRESULT AResult);
Boolean VarIsError(const Variant V);
```

File

Variants

Description

VarIsError returns true if the given variant is an Error (see page 3560) variant (type varError (see page 3866)). If AResult is specified, it is set to the corresponding result code.

See Also

VarType (see page 5090)

VarIsType (see page 5085)

51.3.103 Variants.VarIsFloat Function

Indicates whether the specified variant represents a floating-point value.

Pascal

```
function VarIsFloat(const V: Variant): Boolean;
```

C++

```
Boolean VarIsFloat(const Variant V);
```

File

Variants

Description

VarIsFloat returns true if the given variant represents a floating-point value (In Delphi: Single, Double, or Currency (see page 3235); In C++: float, double, or Currency (see page 3235)). If the variant contains any other type of value, the function result is false.

See Also

VarIsOrdinal (see page 5084)

VarIsNumeric (see page 5083)

VarIsStr (see page 5084)

VarType (see page 5090)

VarIsType (see page 5085)

51.3.104 Variants.VarIsNull Function

Indicates whether the specified variant is Null (see page 5056).

Pascal

```
function VarIsNull(const V: Variant): Boolean;
```

C++

```
Boolean VarIsNull(const Variant V);
```

File

Variants

Description

VarIsNull returns true if the given variant contains the value Null (see page 5056). If the variant contains any other value, the function result is false.

Note: Do not confuse a Null (see page 5056) variant with an unassigned variant. A Null (see page 5056) variant is still assigned, but has the value Null (see page 5056). Unlike unassigned variants, Null (see page 5056) variants can be used in expressions and can be converted to other types of variants.

See Also

Null (see page 5056)

VarType (see page 5090)

VarIsEmpty (see page 5080)

VarIsClear (see page 5079)

VarIsEmptyParam (see page 5081)

VarIsType (see page 5085)

51.3.105 Variants.VarIsNumeric Function

Indicates whether the specified variant represents a numeric value.

Pascal

```
function VarIsNumeric(const V: Variant): Boolean;
```

C++

```
Boolean VarIsNumeric(const Variant V);
```

File

Variants

Description

VarIsNumeric returns true if the given variant's type code indicates a numeric value of some sort. This can be either a floating-point value or an ordinal value. If the variant contains any other type of value, the function result is false.

See Also

VarIsOrdinal (see page 5084)

VarIsFloat (see page 5082)

VarIsStr (see page 5084)

VarType (see page 5090)

VarIsType (see page 5085)

51.3.106 Variants.VarIsOrdinal Function

Indicates whether the specified variant represents an ordinal value.

Pascal

```
function VarIsOrdinal(const V: Variant): Boolean;
```

C++

```
Boolean VarIsOrdinal(const Variant V);
```

File

Variants

Description

VarIsOrdinal returns true if the given variant represents an ordinal value (such as an integer or boolean type). If the variant contains any other type of value, the function result is false.

See Also

VarIsNumeric (see page 5083)

VarIsFloat (see page 5082)

VarIsStr (see page 5084)

VarType (see page 5090)

VarIsType (see page 5085)

51.3.107 Variants.VarIsStr Function

Indicates whether the specified variant represents a string value.

Pascal

```
function VarIsStr(const V: Variant): Boolean;
```

C++

```
Boolean VarIsStr(const Variant V);
```

File

Variants

Description

VarIsStr returns true if the given variant's type code is varString (see page 3871) or varOleStr (see page 3869). If the variant contains any other type of value, the function result is false.

See Also

VarIsNumeric (see page 5083)

VarIsFloat (see page 5082)

VarIsOrdinal (see page 5084)

VarType (see page 5090)

VarIsType (see page 5085)

51.3.108 Variants.VarIsType Function

Indicates whether the specified variant represents a specified type.

Pascal

```
function VarIsType(const V: Variant; AVarType: TVarType): Boolean; overload;
function VarIsType(const V: Variant; const AVarTypes: array of TVarType): Boolean; overload;
```

C++

```
Boolean VarIsType(const Variant V, TVarType AVarType);
Boolean VarIsType(const Variant V, const array of TVarType AVarTypes);
```

File

Variants

Description

VarIsType checks whether the type code for the System::Variant (see page 3421) specified by V matches the specified type code or one of the type codes in the specified array.

V is the System::Variant (see page 3421) to check.

AVarType is the type code against which to check V.

AVarTypes is an array of valid type codes against which to check V.

AVarTypes_size is the index of the last entry in AVarTypes (one less than the number of entries).

VarIsType returns true if the given variant's type code matches the supplied type code or codes, false otherwise.

See Also

VarType (see page 5090)

51.3.109 Variants.VarOverflowError Function

Raises an EVariantOverflowError (see page 4985) exception.

Pascal

```
procedure VarOverflowError(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
VarOverflowError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarOverflowError to indicate that a variant operation has failed due to numeric overflow. ASourceType and ADestType should contain the variant type codes for the source and destination variant types. VarOverflowError uses this information to compose an error string and raise EVariantOverflowError (see page 4985).

See Also

EVariantOverflowError (see page 4985)

TVarType (see page 3801)

51.3.110 Variants.VarRangeCheckError Function

Raises an EVariantRangeCheckError (see page 4986) exception.

Pascal

```
procedure VarRangeCheckError(const ASourceType: TVarType; const ADestType: TVarType);
```

C++

```
VarRangeCheckError(const TVarType ASourceType, const TVarType ADestType);
```

File

Variants

Description

Call VarRangeCheckError to indicate that a variant operation has failed because the result is not in the allowed range for the destination variant type. ASourceType and ADestType are the variant types of the source and destination.

See Also

EVariantRangeCheckError (see page 4986)

51.3.111 Variants.VarResultCheck Function

Raises variant exception based on HRESULT (see page 3754) value.

Pascal

```
procedure VarResultCheck(AResult: HRESULT); overload;
procedure VarResultCheck(AResult: HRESULT; ASourceType: TVarType; ADestType: TVarType);
overload;
```

C++

```
VarResultCheck(HRESULT AResult);
VarResultCheck(HRESULT AResult, TVarType ASourceType, TVarType ADestType);
```

File

Variants

Description

VarResultCheck examines the HRESULT (see page 3754) value returned by many variant routines. If AResult is S_OK (see page 3860), VarResultCheck does nothing. Otherwise, VarResultCheck attempts to map AResult to a variant exception and raise that exception. If VarResultCheck is unable to map AResult to a specific variant Exception (see page 3949), it raises EVariantError (see page 3948).

Specify ASourceType and ADestType when the operation involves conversion between two variant types.

See Also

EVariantError (see page 3948)

51.3.112 Variants.VarSameValue Function

Indicates whether two Variants (see page 4973) have equivalent values.

Pascal

```
function VarSameValue(const A: Variant; const B: Variant): Boolean;
```

C++

```
Boolean VarSameValue(const Variant A, const Variant B);
```

File

Variants

Description

VarSameValue returns true when A has the same value as B, false if the two values are not equivalent.

If A and B can't be compared, VarSameValue raises an exception.

See Also

VarCompareValue (see page 5074)

SameValue

51.3.113 Variants.VarSupports Function

Indicates whether the System::Variant (see page 3421)'s value is an interface that supports a specified interface.

Pascal

```
function VarSupports(const V: Variant; const IID: TGUID; out Intf): Boolean; overload;
function VarSupports(const V: Variant; const IID: TGUID): Boolean; overload;
```

C++

```
Boolean VarSupports(const Variant V, const TGUID IID,  Intf);
Boolean VarSupports(const Variant V, const TGUID IID);
```

File

Variants

Description

VarSupports returns the lower twelve bits of a variant type code. These indicate the type of the variant. For a list of possible variant type codes, see VarType (see page 5090).

Call VarSupports to determine whether the System::Variant (see page 3421) represents an interface that supports the interface identified by the IID parameter. If V supports the interface, VarSupports returns the interface as the Intf parameter (if using the first syntax) and returns true (both versions). If the interface specified by IID is not supported, VarSupports returns false.

Unlike calling the QueryInterface method of the interface, VarSupports handles the case when the System::Variant (see page 3421)'s value is nil (Delphi) or NULL (C++).

See Also

Supports (see page 4294)

51.3.114 Variants.VarToDateTime Function

Converts a specified variant to a TDateTime (see page 3323) value.

Pascal

```
function VarToDateTime(const V: Variant): TDateTime;
```

C++

```
TDateTime VarToDateTime(const Variant V);
```

File

Variants

Description

VarToDateTime converts the given variant to a TDateTime (see page 3323) value. VarToDateTime raises an exception if the conversion fails.

See Also

VarFromDateTime (see page 5076)

51.3.115 Variants.VarToStr Function

Converts a variant's value to a string.

Pascal

```
function VarToStr(const V: Variant): string;
```

C++

```
UnicodeString VarToStr(const Variant V);
```

File

Variants

Description

VarToStr converts the data in the variant V to a string and returns the result. If the variant has a null value, VarToStr returns an empty string.

See Also

VarToStrDef (see page 5089)

VarToWideStr (see page 5089)

VarToWideStrDef (see page 5090)

51.3.116 Variants.VarToStrDef Function

Converts a variant's value to a string.

Pascal

```
function VarToStrDef(const V: Variant; const ADefault: string): string;
```

C++

```
UnicodeString VarToStrDef(const Variant V, const UnicodeString ADefault);
```

File

Variants

Description

VarToStrDef converts the data in the variant V to a string and returns the result. If the variant has a null value, VarToStrDef returns ADefault.

See Also

VarToStr (see page 5088)

VarToWideStr (see page 5089)

VarToWideStrDef (see page 5090)

51.3.117 Variants.VarToWideStr Function

Converts a variant's value to a WideString (see page 3467).

Pascal

```
function VarToWideStr(const V: Variant): WideString;
```

C++

```
BSTR VarToWideStr(const Variant V);
```

File

Variants

Description

VarToWideStr converts the data in the variant V to a WideString (see page 3467) and returns the result. If the variant has a

null value, VarToWideStr returns an empty string.

See Also

VarToStrDef (see page 5089)

VarToStr (see page 5088)

VarToWideStrDef (see page 5090)

51.3.118 Variants.VarToWideStrDef Function

Converts a variant's value to a WideString (see page 3467).

Pascal

```
function VarToWideStrDef(const V: Variant; const ADefault: WideString): WideString;
```

C++

```
BSTR VarToWideStrDef(const Variant V, const BSTR ADefault);
```

File

Variants

Description

VarToWideStrDef converts the data in the variant V to a WideString (see page 3467) and returns the result. If the variant has a null value, VarToWideStrDef returns ADefault.

See Also

VarToStr (see page 5088)

VarToWideStr (see page 5089)

VarToStrDef (see page 5089)

51.3.119 Variants.VarType Function

Returns the type code of a specified variant.

Pascal

```
function VarType(const V: Variant): TVarType;
```

C++

```
TVarType VarType(const Variant V);
```

File

Variants

Description

VarType returns the type code of the given variant. The resulting value is either the type code for a custom Variant (see page 3421) type, or it is constructed from the constants declared in the System (see page 3159) unit.

The lower twelve bits of a variant type code (the bits defined by the varTypeMask (see page 3872) bit mask) define the type of the variant. The varArray (see page 3861) bit is set if the variant is an array of the given type. The varByRef (see page

3862) bit is set if the variant is a reference to a value of the given type as opposed to an actual value.

The following table describes the meaning of each of the variant type codes defined in the System (see page 3159) unit:

VarType	Contents of variant
varEmpty (see page 3865)	The variant is Unassigned (see page 5058).
varNull (see page 3868)	The variant is Null (see page 5056).
varSmallint (see page 3870)	16-bit signed integer (type Smallint in Delphi, short in C++).
varInteger (see page 3867)	32-bit signed integer (type Integer in Delphi, int in C++).
varSingle (see page 3870)	Single-precision floating-point value (type Single in Delphi, float in C++).
varDouble (see page 3865)	Double-precision floating-point value (type double).
varCurrency (see page 3863)	Currency (see page 3235) floating-point value (type Currency (see page 3235)).
varDate (see page 3864)	Date (see page 4087) and time value (type TDateTime (see page 3323)).
varOleStr (see page 3869)	Reference to a dynamically allocated UNICODE string.
varDispatch (see page 3864)	Reference to an Automation object (an IDispatch (see page 3494) interface pointer).
varError (see page 3866)	Operating system error code.
varBoolean (see page 3862)	16-bit boolean (type WordBool).
varVariant (see page 3873)	A variant.
varUnknown (see page 3873)	Reference to an unknown object (an IInterface (see page 3502) or IUnknown (see page 3756) interface pointer).
varShortInt (see page 3869)	8-bit signed integer (type ShortInt in Delphi or signed char in C++)
varByte (see page 3863)	A Byte
varWord (see page 3874)	unsigned 16-bit value (Word)
varLongWord (see page 3868)	unsigned 32-bit value (type LongWord in Delphi or unsigned long in C++)
varInt64 (see page 3867)	64-bit signed integer (Int64 in Delphi or __int64 in C++)
varStrArg (see page 3871)	COM-compatible string.
varString (see page 3871)	Reference to a dynamically allocated string (not COM compatible).

The value returned by VarType corresponds to the VType field of a TVarData (see page 3740) record.

The type of a variant can be changed using VarAsType (see page 5073).

See Also

TVarData (see page 3740)

VarAsType (see page 5073)

VarTypeToDataType

VarTypeAsText (see page 5092)

51.3.120 Variants.VarTypeAsText Function

Returns variant type descriptive string.

Pascal

```
function VarTypeAsText(const AType: TVarType): string;
```

C++

```
UnicodeString VarTypeAsText(const TVarType AType);
```

File

Variants

Description

VarTypeAsText returns a short string that names the variant type indicated by AType.

See Also

TVarData (see page 3740)

VarAsType (see page 5073)

VarTypeToDataType

VarType (see page 5090)

51.3.121 Variants.VarTypeIsValidArrayType Function

Indicates whether a System::Variant (see page 3421) type code represents a type that can be used for the elements in a System::Variant (see page 3421) array.

Pascal

```
function VarTypeIsValidArrayType(const AVarType: TVarType): Boolean;
```

C++

```
Boolean VarTypeIsValidArrayType(const TVarType AVarType);
```

File

Variants

Description

VarTypeIsValidArrayType indicates whether AVarType is the type code for a System::Variant (see page 3421) type that can

represent the elements of a System::Variant (see page 3421) array. This function is only intended for use with built-in System::Variant (see page 3421) types. It returns false for custom System::Variant (see page 3421) types, although they can be used for the elements of System::Variant (see page 3421) arrays.

See Also

VarArrayCreate (see page 5059)

VarTypeIsValidElementType (see page 5093)

51.3.122 Variants.VarTypeIsValidElementType Function

Indicates whether a System::Variant (see page 3421) type code represents a type that can be used for the elements in a System::Variant (see page 3421) array.

Pascal

```
function VarTypeIsValidElementType(const AVarType: TVarType): Boolean;
```

C++

```
Boolean VarTypeIsValidElementType(const TVarType AVarType);
```

File

Variants

Description

VarTypeIsValidElementType indicates whether AVarType is the type code for a System::Variant (see page 3421) type that can represent the elements of a System::Variant (see page 3421) array. Unlike the VarTypeIsValidArrayType (see page 5092) function, this function returns true for custom System::Variant (see page 3421) types.

See Also

VarArrayCreate (see page 5059)

VarTypeIsValidArrayType (see page 5092)

51.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TBooleanToStringRule (see page 5094)	TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 5100).
	TNullCompareRule (see page 5094)	TNullCompareRule enumerates possible rules for comparing variant nulls with other values.
	TVarCompareResult (see page 5095)	TVarCompareResult describes the results of comparing two Variants (see page 4973).
	TVariantRelationship (see page 5096)	TVariantRelationship describes the results of comparing two Variants (see page 4973).

51.4.1 Variants.TBooleanToStringRule Enumeration

TBooleanToStringRule enumerates possible rules for BooleanToStringRule (see page 5100).

Pascal

```
TBooleanToStringRule = (
  bsrAsIs,
  bsrLower,
  bsrUpper
);
```

C++

```
enum TBooleanToStringRule {
  bsrAsIs,
  bsrLower,
  bsrUpper
};
```

File

Variants

Description

TBooleanToStringRule enumerates the values that can be assigned to the global variables NullEqualityRule (see page 5103) and NullMagnitudeRule (see page 5103).

51.4.2 Variants.TNullCompareRule Enumeration

TNullCompareRule enumerates possible rules for comparing variant nulls with other values.

Pascal

```
TNullCompareRule = (
  ncrError,
  ncrStrict,
  ncrLoose
```

```
);
```

C++

```
enum TNullCompareRule {
  ncrError,
  ncrStrict,
  ncrLoose
};
```

File

Variants

Description

TNullCompareRule enumerates the values that can be assigned to the global variables NullEqualityRule (see page 5103) and NullMagnitudeRule (see page 5103).

51.4.3 Variants.TVarCompareResult Enumeration

TVarCompareResult describes the results of comparing two Variants (see page 4973).

Pascal

```
TVarCompareResult = (
  crLessThan,
  crEqual,
  crGreaterThan
);
```

C++

```
enum TVarCompareResult {
  crLessThan,
  crEqual,
  crGreaterThan
};
```

File

Variants

Description

TVarCompareResult describes the result of the comparison between two System::Variant (see page 3421) values. The following table lists the possible values and their meaning.

Value	Meaning
crEqual	A is equivalent to B. Note that two NULL Variants (see page 4973) are considered equal and two empty Variants (see page 4973) are considered equal, but a NULL System::Variant (see page 3421) does not equal an empty System::Variant (see page 3421).
crLessThan	A is less than B.
crGreaterThan	A is greater than B.

See Also

Compare (see page 4997)

51.4.4 Variants.TVariantRelationship Enumeration

TVariantRelationship describes the results of comparing two Variants (see page 4973).

Pascal

```
TVariantRelationship = (
  vrEqual,
  vrLessThan,
  vrGreaterThan,
  vrNotEqual
);
```

C++

```
enum TVariantRelationship {
  vrEqual,
  vrLessThan,
  vrGreaterThan,
  vrNotEqual
};
```

File

Variants

Description

TVariantRelationship describes the relationship between two System::Variant (see page 3421) values. The following table lists the possible values:

Value	Meaning
vrEqual	A is equivalent to B. Note that two NULL Variants (see page 4973) are considered equal, and two empty Variants (see page 4973) are considered equal, but a NULL System::Variant (see page 3421) does not equal an empty System::Variant (see page 3421).
vrLessThan	A is less than B.
vrGreaterThan	A is larger than B.
vrNotEqual	A and B are not equal, but the values are such that they can't be ordered relative to each other. For example, this describes the relationship when either A or B is empty or NULL, and the other one is not.

See Also

VarCompareValue (see page 5074)

51.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAnyProc (see page 5097)	TAnyProc represents a pointer to a procedure that accepts a parameter of type TVarData (see page 3740).
TCustomVariantTypeClass (see page 5097)	TCustomVariantTypeClass defines the metaclass for TCustomVariantType (see page 4987).
TVarDataArray (see page 5098)	TVarDataArray is an array of TVarData (see page 3740) records.
TVarDispProc (see page 5098)	TVarDispProc defines a pointer to a procedure used as support for IDispatch (see page 3494)-based Automation.

51.5.1 Variants.TAnyProc Type

TAnyProc represents a pointer to a procedure that accepts a parameter of type TVarData (see page 3740).

Pascal

```
TAnyProc = procedure (var V: TVarData);
```

C++

```
(var V: TVarData) ( TAnyProc)();
```

File

Variants

Description

Declare variables of type TAnyProc to accept any procedure that has a single parameter of type TVarData (see page 3740). TAnyProc is primarily used in the Variants (see page 4973) unit to support custom processing for Variants (see page 4973) of type varAny (see page 3861).

See Also

ChangeAnyProc (see page 5101)

ClearAnyProc (see page 5101)

RefAnyProc (see page 5106)

51.5.2 Variants.TCustomVariantTypeClass Type

TCustomVariantTypeClass defines the metaclass for TCustomVariantType (see page 4987).

Pascal

```
TCustomVariantTypeClass = class of TCustomVariantType;
```

C++

```
TCustomVariantType TCustomVariantTypeClass;
```

File

Variants

Description

TCustomVariantTypeClass is the metaclass for TCustomVariantType (see page 4987). Its value is the class reference for TCustomVariantType (see page 4987) or for one of its descendants.

See Also

TCustomVariantType (see page 4987)

Class References

51.5.3 Variants.TVarDataArray Type

TVarDataArray is an array of TVarData (see page 3740) records.

Pascal

```
TVarDataArray = array of TVarData;
```

C++

```
array of TVarData TVarDataArray;
```

File

Variants

Description

TVarDataArray represents a dynamic array of TVarData (see page 3740) records. TVarDataArray is primarily used by methods like DoProcedure and DoFunction of the TInvokeableVariantType (see page 5014) class in order to pass a list of parameters to the dynamically invoked methods in a Variant (see page 3421) variable.

See Also

TVarData (see page 3740)

DoProcedure (see page 5019)

DoFunction (see page 5018)

51.5.4 Variants.TVarDispProc Type

TVarDispProc defines a pointer to a procedure used as support for IDispatch (see page 3494)-based Automation.

Pascal

```
TVarDispProc = procedure (Dest: PVariant; const Source: Variant; CallDesc: PCallDesc; Params:
Pointer);
```

C++

```
(Dest: PVariant; const Source: Variant; CallDesc: PCallDesc; Params: Pointer) (
```

```
TVarDispProc)();
```

File

Variants

Description

In Delphi, TVarDispProc is a pointer to a routine that is called to execute methods (or access properties) of an interface that is assigned to a System::Variant (see page 3421). TVarDispProc is used in the Variants (see page 4973) unit to control how Variants (see page 4973) execute methods when a value of the Variants (see page 4973) is an interface.

See Also

VarDispProc (see page 5106)

EVariantDispatchError (see page 4981)

51.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
BooleanToStringRule (see page 5100)	Specifies rule fixing up Boolean values converted to strings.
ChangeAnyProc (see page 5101)	Provides the support for changing the type of a Variant (see page 3421) from varAny (see page 3861) to a real Variant (see page 3421).
ClearAnyProc (see page 5101)	Provides the support for clearing a Variant (see page 3421) of type varAny (see page 3861).
EmptyParam (see page 5102)	Contains an OleVariant (see page 3274) that represents an unused optional parameter on a dual interface.
NullAsStringValue (see page 5102)	Specifies string value of Null (see page 5056) variants.
NullEqualityRule (see page 5103)	Specifies rule for Null (see page 5056) variant equality comparison.
NullMagnitudeRule (see page 5103)	Specifies rule for Null (see page 5056) variant magnitude comparison.
NullStrictConvert (see page 5104)	Specifies whether Null (see page 5056) conversion is an error.
OleVariantInt64AsDouble (see page 5105)	Controls the mode in which `Int64` values are stored in a OleVariant (see page 3274).
PackVarCreation (see page 5105)	Specifies whether space-efficient System::Variant (see page 3421) Types (see page 4649) are preferred.
RefAnyProc (see page 5106)	Provides the support for increasing the reference count of a Variant (see page 3421) of type varAny (see page 3861).
VarDispProc (see page 5106)	Provides the support for IDispatch (see page 3494)-based Automation.

51.6.1 Variants.BooleanToStringRule Variable

Specifies rule fixing up Boolean values converted to strings.

Pascal

```
BooleanToStringRule: TBooleanToStringRule = bsrAsIs;
```

C++

```
TBooleanToStringRule BooleanToStringRule = bsrAsIs;
```

File

Variants

Description

BooleanToStringRule determines the rule for fixing up the string generated by a conversion from Boolean to string.

Constant	Meaning
bsrAsIs	(Default.) The value obtained from the locale is left as is.
bsrUpper	The string is converted to all upper case.
bsrLower	The string is converted to all lower case.

51.6.2 Variants.ChangeAnyProc Variable

Provides the support for changing the type of a Variant (see page 3421) from varAny (see page 3861) to a real Variant (see page 3421).

Pascal

```
ChangeAnyProc: TAnyProc;
```

C++

```
TAnyProc ChangeAnyProc;
```

File

Variants

Description

Use ChangeAnyProc to change a Variant (see page 3421) of the varAny (see page 3861) type to a real Variant (see page 3421).

Assign a custom TAnyProc (see page 5097) to ChangeAnyProc in order to control how the varAny (see page 3861) Variants (see page 4973) are being transformed to real Variants (see page 4973).

See Also

TAnyProc (see page 5097)

ClearAnyProc (see page 5101)

RefAnyProc (see page 5106)

51.6.3 Variants.ClearAnyProc Variable

Provides the support for clearing a Variant (see page 3421) of type varAny (see page 3861).

Pascal

```
ClearAnyProc: TAnyProc;
```

C++

```
TAnyProc ClearAnyProc;
```

File

Variants

Description

Use ClearAnyProc to clear a Variant (see page 3421) of type varAny (see page 3861). Clearing a varAny (see page 3861) Variant (see page 3421) usually implies clearing the real value of the Variant (see page 3421).

Assign a custom TAnyProc (see page 5097) to ClearAnyProc in order to control how the varAny (see page 3861) variants are being cleared.

See Also

TAnyProc (see page 5097)

ChangeAnyProc (see page 5101)

RefAnyProc (see page 5106)

51.6.4 Variants.EmptyParam Variable

Contains an OleVariant (see page 3274) that represents an unused optional parameter on a dual interface.

Pascal

```
EmptyParam: OleVariant;
```

C++

```
OleVariant EmptyParam;
```

File

Variants

Description

EmptyParam can be used for Variants (see page 4973) that represent optional parameters whose value is unassigned. This is useful when the code that marshals method calls requires a fixed number of parameters, even if some of them are optional.

When marshaling interface calls that include optional parameters, COM requires a value for those parameters even when they are not used. EmptyParam returns an OleVariant (see page 3274) you can pass as that value to indicate that the parameter is not used.

Marshaling interfaces also exist under Linux, but we can't spare the resources to document them right now.

See Also

VarIsEmptyParam (see page 5081)

51.6.5 Variants.NullAsStringValue Variable

Specifies string value of Null (see page 5056) variants.

Pascal

```
NullAsStringValue: string = '';
```

C++

```
UnicodeString NullAsStringValue = '';
```

File

Variants

Description

NullAsStringValue specifies the string that is the result of a conversion from a Null (see page 5056) variant to a string type. By default, NullAsStringValue is the empty string.

NullAsStringValue is only used when the NullStrictConvert (see page 5104) global variable is set to false.

See Also

NullStrictConvert (see page 5104)

Null (see page 5056)

51.6.6 Variants.NullEqualityRule Variable

Specifies rule for Null (see page 5056) variant equality comparison.

Pascal

```
NullEqualityRule: TNullCompareRule = ncrLoose;
```

C++

```
TNullCompareRule NullEqualityRule = ncrLoose;
```

File

Variants

Description

NullEqualityRule determines the rule for handling equality comparisons (equals or not-equals) when one of the values compared is a variant containing a null value. NullEqualityRule can have the following values:

Constant	Meaning
ncrLoose	(Default.) A variant containing a null is treated as a unique value that is only equal to another variant containing a null.
ncrStrict	An equality comparison of a variant containing a null is always false, regardless of the other value.
ncrError	An equality comparison raises an EVariantInvalidNullOpError (see page 4982) if either value is a variant containing a null.

See Also

Null (see page 5056)

NullMagnitudeRule (see page 5103)

NullStrictConvert (see page 5104)

EVariantInvalidNullOpError (see page 4982)

51.6.7 Variants.NullMagnitudeRule Variable

Specifies rule for Null (see page 5056) variant magnitude comparison.

Pascal

```
NullMagnitudeRule: TNullCompareRule = ncrLoose;
```

C++

```
TNullCompareRule NullMagnitudeRule = ncrLoose;
```

File

Variants

Description

NullMagnitudeRule determines the rule for handling magnitude comparisons (one value is greater than, greater than or equals,

less than, or less than or equals to another) when one of the values compared is a variant containing a null value. NullMagnitudeRule can have the following values:

Constant	Meaning
ncrLoose	(Default.) A variant containing a null is treated as a unique value that is less than any other value, but equal to another variant containing a null.
ncrStrict	A magnitude comparison of a variant containing a null is always false, regardless of the other value.
ncrError	A magnitude comparison raises an EVariantInvalidNullOpError (see page 4982) if either value is a variant containing a null.

See Also

Null (see page 5056)

NullStrictConvert (see page 5104)

EVariantInvalidNullOpError (see page 4982)

51.6.8 Variants.NullStrictConvert Variable

Specifies whether Null (see page 5056) conversion is an error.

Pascal

```
NullStrictConvert: Boolean = True;
```

C++

```
Boolean NullStrictConvert = True;
```

File

Variants

Description

NullStrictConvert determines the outcome of attempts to convert Null (see page 5056) variants to other types. If NullStrictConvert is true (default), attempting to convert a Null (see page 5056) variant raises a EVariantTypeCastError (see page 4986), unless the conversion is to a custom variant that defines a conversion from Null (see page 5056). If NullStrictConvert is false, then conversion from Null (see page 5056) follows the following rules:

Destination	Conversion result
Numeric type	Zero
String type	Value of NullAsStringValue (see page 5102) global variable. (Default: empty string.)
Boolean	False.

See Also

TVarData (see page 3740)

VarType (see page 5090)

NullAsStringValue (see page 5102)

EVariantTypeCastError (see page 4986)

51.6.9 Variants.OleVariantInt64AsDouble Variable

Controls the mode in which `Int64` values are stored in a OleVariant (see page 3274).

Pascal

```
OleVariantInt64AsDouble: Boolean = False;
```

C++

```
Boolean OleVariantInt64AsDouble = False;
```

File

Variants

Description

Use OleVariantInt64AsDouble to set or check whether the `Int64` values are stored in a OleVariant (see page 3274) as `Double` values. If the value of OleVariantInt64AsDouble is **True**, then `Int64` values are stored as `Double` values and the type of the OleVariant (see page 3274) is varDouble (see page 3865). Otherwise, the `Int64` values are stored directly and the type of the OleVariant (see page 3274) is varInt64 (see page 3867).

Note: This flag is useful only on Windows 9x/2000 platforms, on which OleVariants cannot store an `Int64` value.

See Also

BooleanToStringRule (see page 5100)

PackVarCreation (see page 5105)

51.6.10 Variants.PackVarCreation Variable

Specifies whether space-efficient System::Variant (see page 3421) Types (see page 4649) are preferred.

Pascal

```
PackVarCreation: Boolean = True;
```

C++

```
Boolean PackVarCreation = True;
```

File

Variants

Description

PackVarCreation controls the way System::Variant (see page 3421) Types (see page 4649) are chosen for newly-created Variants (see page 4973). If PackVarCreation is true (default), a new System::Variant (see page 3421) has the smallest possible System::Variant (see page 3421) Type. If PackVarCreation is false, a new System::Variant (see page 3421) has the System::Variant (see page 3421) Type that minimizes execution time.

See Also

TVarData (see page 3740)

VarType (see page 5090)

51.6.11 Variants.RefAnyProc Variable

Provides the support for increasing the reference count of a Variant (see page 3421) of type varAny (see page 3861).

Pascal

```
RefAnyProc: TAnyProc;
```

C++

```
TAnyProc RefAnyProc;
```

File

Variants

Description

Use RefAnyProc to increase the reference count of a Variant (see page 3421) of type varAny (see page 3861).

Assign a custom TAnyProc (see page 5097) to RefAnyProc in order to control the referencing of the varAny (see page 3861) Variants (see page 4973).

See Also

TAnyProc (see page 5097)

ChangeAnyProc (see page 5101)

ClearAnyProc (see page 5101)

51.6.12 Variants.VarDispProc Variable

Provides the support for IDispatch (see page 3494)-based Automation.

Pascal

```
VarDispProc: TVarDispProc;
```

C++

```
TVarDispProc VarDispProc;
```

File

Variants

Description

In Delphi, VarDispProc points to a routine that is called to execute methods (or access properties) of an interface that is assigned to a System::Variant (see page 3421). The Variants (see page 4973) unit assigns a procedure that generates error 222 or raises an EVariantDispatchError (see page 4981) exception. The ComObj unit reassigns this variable to a procedure that generates the appropriate COM IDispatch (see page 3494) calls.

To control how Variants (see page 4973) execute methods when they have a value that is an interface, assign your own procedure to VarDispProc.

See Also

EVariantDispatchError (see page 4981)

52 VarUtils Namespace

This is namespace VarUtils.

Topics

Name	Description
Classes (see page 5110)	The following table lists classes in this documentation.
Functions (see page 5115)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5132)	The following table lists structs, records, enums in this documentation.
Variables (see page 5133)	The following table lists variables in this documentation.
Constants (see page 5149)	The following table lists constants in this documentation.

Classes

	Name	Description
	ESafeArrayBoundsError (see page 5110)	ESafeArrayBoundsError is the exception class for errors that occur because a variant array index is out of bounds.
	ESafeArrayCreateError (see page 5110)	ESafeArrayCreateError is the exception class for failing to create a variant array.
	ESafeArrayError (see page 5111)	ESafeArrayError is the exception class for errors involving variant arrays.
	ESafeArrayLockedError (see page 5114)	ESafeArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.

Constants

Name	Description
ARR_DISPATCH (see page 5150)	Used to test whether the variant array contains IDispatch (see page 3494) elements.
ARR_FIXEDSIZE (see page 5150)	Used to test whether the variant array is of fixed size.
ARR_NONE (see page 5151)	Specifies that a variant array has no flags set.
ARR_OLESTR (see page 5151)	Used to test whether the variant array contains BSTR elements.
ARR_UNKNOWN (see page 5152)	Used to test whether the variant array contains IUnknown (see page 3756) elements.
ARR_VARIANT (see page 5152)	Used to test whether the variant array contains `Variant (see page 3421)` elements.
CMaxArrayVarType (see page 5153)	CMaxArrayVarType gives the upper bound of the CVarTypeToElementInfo (see page 5154) array.
CMinArrayVarType (see page 5153)	CMinArrayVarType gives the lower bound of the CVarTypeToElementInfo (see page 5154) array.
CVarTypeToElementInfo (see page 5154)	This constant is internally used by the SafeArrayCreate (see page 5118) function.

GenericOperations (see page 5155)	Specifies whether the implementation for the variant operation routines is cross-platform.
GenericSafeArrays (see page 5155)	Specifies whether the implementation for the safe array routines is cross-platform.
GenericSafeArrayUsesLibC (see page 5156)	Specifies whether the implementation for the safe array routines uses the `Libc` unit.
GenericVariants (see page 5156)	Specifies whether the implementation for the variant support routines is cross-platform.
GenericVarUtils (see page 5157)	Specifies whether the implementation for the variant-manipulating routines is cross-platform.
VAR_ARRAYISLOCKED (see page 5157)	Specifies that the given array is locked.
VAR_BADINDEX (see page 5158)	Indicates that the given index is out of range.
VAR_BADVARTYPE (see page 5158)	Specifies that the variant type of the input parameter is invalid.
VAR_CMP_EQ (see page 5158)	VAR_CMP_EQ specifies that the variants are equal, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_CMP_GT (see page 5159)	VAR_CMP_GT specifies that the first variant is greater than the second, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_CMP_LT (see page 5159)	VAR_CMP_LT specifies that the first variant is lower than the second, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_CMP_NULL (see page 5160)	VAR_CMP_NULL specifies that either one of the variants is `nil`, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_EXCEPTION (see page 5160)	Specifies that the application raised an exception.
VAR_INVALIDARG (see page 5161)	Specifies that either one of the arguments is invalid.
VAR_LOCALE_USER_DEFAULT (see page 5161)	Specifies the default locale for the user or process.
VAR_NOTIMPL (see page 5161)	Specifies that the operation is not implemented.
VAR_OK (see page 5162)	Specifies that a variant operation was successful.
VAR_OUTOFMEMORY (see page 5162)	Specifies that there is not enough memory to complete the operation.
VAR_OVERFLOW (see page 5163)	Specifies that a variant operation caused an overflow.
VAR_PARAMNOTFOUND (see page 5163)	Indicates that there is no correspondence between the parameter DISPIDs and a parameter of the method.
VAR_TYPEMISMATCH (see page 5163)	Specifies that the argument cannot be cast to the specified type.
VAR_UNEXPECTED (see page 5164)	Indicates that an unexpected error occurred.

Functions

	Name	Description
	SafeArrayAccessData (see page 5115)	Returns a pointer to the actual safe array data and locks the safe array.
	SafeArrayAllocData (see page 5116)	Allocates the actual data of the given safe array.
	SafeArrayAllocDescriptor (see page 5116)	Allocates a descriptor for the safe array.
	SafeArrayCheck (see page 5117)	Checks whether the given parameter is a safe array operation error code.
	SafeArrayCopy (see page 5117)	Copies a safe array to another.
	SafeArrayCopyData (see page 5118)	Copies the data from a safe array to another.
	SafeArrayCreate (see page 5118)	Creates a safe array.
	SafeArrayDestroy (see page 5119)	Destroys a safe array.
	SafeArrayDestroyData (see page 5120)	Destroys the actual data of the given safe array.
	SafeArrayDestroyDescriptor (see page 5120)	Destroys the descriptor of the given safe array.
	SafeArrayError (see page 5120)	Raises an event corresponding to the given safe array operation error code.
	SafeArrayGetDim (see page 5121)	Returns the number of dimension of the given safe array.
	SafeArrayGetElement (see page 5121)	Returns the element at the specified position in the given safe array.
	SafeArrayGetElemSize (see page 5122)	Returns the size in bytes of an element in the given safe array.
	SafeArrayGetLBound (see page 5122)	Returns the lower bound for the given dimension of a safe array.
	SafeArrayGetUBound (see page 5123)	Returns the upper bound for the given dimension of a safe array.
	SafeArrayLock (see page 5123)	Locks a safe array.
	SafeArrayPtrOfIndex (see page 5124)	Returns a pointer to the element at the specified position in the given safe array.
	SafeArrayPutElement (see page 5125)	Sets the element at the specified position in the given safe array.
	SafeArrayRedim (see page 5125)	Changes the bound information for the last dimension of a safe array.
	SafeArrayUnaccessData (see page 5126)	Unlocks a safe array previously locked using the SafeArrayAccessData (see page 5115) routine.
	SafeArrayUnlock (see page 5127)	Unlocks a safe array.

	VarExceptionToResult (see page 5127)	Converts an Exception (see page 3949) type argument to an HRESULT (see page 3754) type.
	VariantChangeType (see page 5128)	Converts a variant from one type to another.
	VariantClear (see page 5128)	Clears a variant.
	VariantCopy (see page 5129)	Clears the destination variant and makes a copy of the source variant.
	VariantCopyInd (see page 5130)	Clears the destination variant and makes a copy of the source variant, including the case when the source is of varByRef (see page 3862) type.
	VariantInit (see page 5130)	Initializes a variant.

Structs, Records, Enums

	Name	Description
	TVarTypeToElementInfo (see page 5132)	This is an internal record type used in declaring the CVarTypeToElementInfo (see page 5154) constant array.

Variables

Name	Description
VarAdd (see page 5133)	VarAdd is a function type variable that specifies the function making the addition operation between two variants.
VarAnd (see page 5134)	VarAnd is a function type variable that specifies the function making the bitwise **and** operation between two variants.
VarBoolFromStr (see page 5134)	VarBoolFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a boolean value.
VarBstrFromBool (see page 5135)	VarBstrFromBool is used to specify the function that makes the conversion from a boolean value to a WideString (see page 3467).
VarBstrFromCy (see page 5136)	VarBstrFromCy is used to specify the function that makes the conversion from a currency value to a WideString (see page 3467).
VarBstrFromDate (see page 5137)	VarBstrFromDate is used to specify the function that makes the conversion from a date value to a WideString (see page 3467).
VarCmp (see page 5138)	VarCmp is a variable used to specify the function that compares two variants.
VarCyFromStr (see page 5139)	VarCyFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a currency value.
VarDateFromStr (see page 5139)	VarDateFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a date value.
VarDiv (see page 5140)	VarDiv is a function type variable that specifies the function making the division operation between two variants.
VarI4FromStr (see page 5141)	VarI4FromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a four bytes integer value.
VariantChangeTypeEx (see page 5142)	VariantChangeTypeEx stores the function that converts a variant from one type to another, using locale-specific information.
VarIDiv (see page 5142)	VarIDiv is a function type variable that specifies the function making the division operation between two variants, after converting the variants to integers.
VarMod (see page 5143)	VarMod is a function type variable that specifies the function that calculates the remainder on division of two variants.
VarMul (see page 5143)	VarMul is a function type variable that specifies the function making the multiplication operation between two variants.
VarNeg (see page 5144)	VarNeg is a function type variable that specifies the function that inverts the sign of a variant.
VarNot (see page 5144)	VarNot is a function type variable that specifies the function that performs a **not** operation on a variant.
VarOr (see page 5145)	VarOr is a function type variable that specifies the function doing the bitwise **or** operation between two variants.
VarR4FromStr (see page 5145)	VarR4FromStr is used to specify the function that does the conversion from a WideString (see page 3467) to a four bytes floating-point value.
VarR8FromStr (see page 5146)	VarR8FromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a eight bytes floating-point value.
VarSub (see page 5147)	VarSub is a function type variable that specifies the function making the subtraction operation between two variants.
VarXor (see page 5147)	VarXor is a function type variable that specifies the function making the bitwise **xor** (exclusive or) operation between two variants.

52.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ESafeArrayBoundsError (see page 5110)	ESafeArrayBoundsError is the exception class for errors that occur because a variant array index is out of bounds.
	ESafeArrayCreateError (see page 5110)	ESafeArrayCreateError is the exception class for failing to create a variant array.
	ESafeArrayError (see page 5111)	ESafeArrayError is the exception class for errors involving variant arrays.
	ESafeArrayLockedError (see page 5114)	ESafeArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.

52.1.1 ESafeArrayBoundsError Class

ESafeArrayBoundsError is the exception class for errors that occur because a variant array index is out of bounds.

Class Hierarchy

Pascal

```
ESafeArrayBoundsError = class(ESafeArrayError);
```

C++

```
class ESafeArrayBoundsError : public ESafeArrayError;
```

File

VarUtils

Description

ESafeArrayBoundsError is raised when an operation on a variant array fails because an index value is out of bounds.

To raise this exception programmatically, call SafeArrayError (see page 5120) with the argument `VAR_BADINDEX` (see page 5158).

See Also

SafeArrayGetElement (see page 5121)

SafeArrayError (see page 5120)

ESafeArrayError (see page 5111)

52.1.2 ESafeArrayCreateError Class

ESafeArrayCreateError is the exception class for failing to create a variant array.

Class Hierarchy

Pascal

```
ESafeArrayCreateError = class(ESafeArrayError);
```

C++

```
class ESafeArrayCreateError : public ESafeArrayError;
```

File

VarUtils

Description

ESafeArrayCreateError is raised when an attempt to create a variant array fails.

See Also

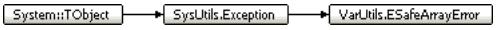
SafeArrayCreate (see page 5118)

ESafeArrayError (see page 5111)

52.1.3 ESafeArrayError Class

ESafeArrayError is the exception class for errors involving variant arrays.

Class Hierarchy



Pascal

```
ESafeArrayError = class(Exception);
```

C++

```
class ESafeArrayError : public Exception;
```

File

VarUtils

Description

ESafeArrayError is raised when a variant array operation fails.

ESafeArrayError has a number of descendant classes that provide more specific information about the error, namely ESafeArrayBoundsError (see page 5110), ESafeArrayCreateError (see page 5110), and ESafeArrayLockedError (see page 5114).

See Also

ESafeArrayBoundsError (see page 5110)

ESafeArrayCreateError (see page 5110)

ESafeArrayLockedError (see page 5114)

52.1.3.1 ESafeArrayError Members

The following tables list the members exposed by ESafeArrayError.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 3952)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateHResult (see page 5113)	This is CreateHResult, a member of class ESafeArrayError.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 3960)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
V	ToString (see page 3380)	Returns a string containing the class name.
V	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	ErrorCode (see page 5113)	This is ErrorCode, a member of class ESafeArrayError.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
R	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
R	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
R	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

52.1.3.2 ESafeArrayError.CreateHResult Constructor

Pascal

```
constructor CreateHResult(AResult: HRESULT; const AMessage: string = '');
```

C++

```
__fastcall ESafeArrayError(HRESULT AResult, const UnicodeString AMessage = '');
```

Description

This is CreateHResult, a member of class ESafeArrayError.

52.1.3.3 ESafeArrayError Properties

The properties of the ESafeArrayError class are listed here.

Public Properties

	Name	Description
	ErrorCode (see page 5113)	This is ErrorCode, a member of class ESafeArrayError.

52.1.3.3.1 ESafeArrayError.ErrorCode Property

Pascal

```
property ErrorCode: HRESULT;
```

C++

```
__property HRESULT ErrorCode;
```

Description

This is ErrorCode, a member of class ESafeArrayError.

52.1.4 ESafeArrayLockedError Class

ESafeArrayLockedError is the exception class for errors that occur because a variant array is locked or fixed.

Class Hierarchy

Pascal

```
ESafeArrayLockedError = class(ESafeArrayError);
```

C++

```
class ESafeArrayLockedError : public ESafeArrayError;
```

File

VarUtils

Description

ESafeArrayLockedError is raised when an operation on a variant array fails because the array is locked or fixed.

To raise this exception programmatically, call SafeArrayError (see page 5120) with the argument VAR_ARRAYISLOCKED (see page 5157).

See Also

SafeArrayLock (see page 5123)

SafeArrayUnlock (see page 5127)

SafeArrayError (see page 5120)

ESafeArrayError (see page 5111)

52.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	SafeArrayAccessData (see page 5115)	Returns a pointer to the actual safe array data and locks the safe array.
	SafeArrayAllocData (see page 5116)	Allocates the actual data of the given safe array.
	SafeArrayAllocDescriptor (see page 5116)	Allocates a descriptor for the safe array.
	SafeArrayCheck (see page 5117)	Checks whether the given parameter is a safe array operation error code.
	SafeArrayCopy (see page 5117)	Copies a safe array to another.
	SafeArrayCopyData (see page 5118)	Copies the data from a safe array to another.
	SafeArrayCreate (see page 5118)	Creates a safe array.
	SafeArrayDestroy (see page 5119)	Destroys a safe array.
	SafeArrayDestroyData (see page 5120)	Destroys the actual data of the given safe array.
	SafeArrayDestroyDescriptor (see page 5120)	Destroys the descriptor of the given safe array.
	SafeArrayError (see page 5120)	Raises an event corresponding to the given safe array operation error code.
	SafeArrayGetDim (see page 5121)	Returns the number of dimension of the given safe array.
	SafeArrayGetElement (see page 5121)	Returns the element at the specified position in the given safe array.
	SafeArrayGetElemSize (see page 5122)	Returns the size in bytes of an element in the given safe array.
	SafeArrayGetLBound (see page 5122)	Returns the lower bound for the given dimension of a safe array.
	SafeArrayGetUBound (see page 5123)	Returns the upper bound for the given dimension of a safe array.
	SafeArrayLock (see page 5123)	Locks a safe array.
	SafeArrayPtrOfIndex (see page 5124)	Returns a pointer to the element at the specified position in the given safe array.
	SafeArrayPutElement (see page 5125)	Sets the element at the specified position in the given safe array.
	SafeArrayRedim (see page 5125)	Changes the bound information for the last dimension of a safe array.
	SafeArrayUnaccessData (see page 5126)	Unlocks a safe array previously locked using the SafeArrayAccessData (see page 5115) routine.
	SafeArrayUnlock (see page 5127)	Unlocks a safe array.
	VarExceptionToResult (see page 5127)	Converts an Exception (see page 3949) type argument to an HRESULT (see page 3754) type.
	VariantChangeType (see page 5128)	Converts a variant from one type to another.
	VariantClear (see page 5128)	Clears a variant.
	VariantCopy (see page 5129)	Clears the destination variant and makes a copy of the source variant.
	VariantCopyInd (see page 5130)	Clears the destination variant and makes a copy of the source variant, including the case when the source is of varByRef (see page 3862) type.
	VariantInit (see page 5130)	Initializes a variant.

52.2.1 VarUtils.SafeArrayAccessData Function

Returns a pointer to the actual safe array data and locks the safe array.

Pascal

```
function SafeArrayAccessData(VarArray: PVarArray; out Data: Pointer): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayAccessData(PVarArray VarArray, void * Data);
```

File

VarUtils

Description

Use SafeArrayAccessData to obtain a pointer to the actual safe array data and also to lock the safe array.

See Also

SafeArrayUnaccessData (see page 5126)

52.2.2 VarUtils.SafeArrayAllocData Function

Allocates the actual data of the given safe array.

Pascal

```
function SafeArrayAllocData(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayAllocData(PVarArray VarArray);
```

File

VarUtils

Description

SafeArrayAllocData allocates the actual data for the given safe array. It is used internally by the SafeArrayCreate (see page 5118) and SafeArrayCopy (see page 5117) routines. We recommend you not to use it directly when writing your application.

See Also

SafeArrayCreate (see page 5118)

SafeArrayCopy (see page 5117)

52.2.3 VarUtils.SafeArrayAllocDescriptor Function

Allocates a descriptor for the safe array.

Pascal

```
function SafeArrayAllocDescriptor(DimCount: Integer; out VarArray: PVarArray): HRESULT;
stdcall;
```

C++

```
__stdcall HRESULT SafeArrayAllocDescriptor(int DimCount, PVarArray VarArray);
```

File

VarUtils

Description

SafeArrayAllocDescriptor allocates a descriptor for the given safe array. It is used internally by the SafeArrayCreate (see page 5118) and SafeArrayCopy (see page 5117) routines. We recommend you not to use it directly when writing your application.

See Also

SafeArrayCreate (see page 5118)

SafeArrayCopy (see page 5117)

52.2.4 VarUtils.SafeArrayCheck Function

Checks whether the given parameter is a safe array operation error code.

Pascal

```
procedure SafeArrayCheck(AResult: HRESULT);
```

C++

```
SafeArrayCheck(HRESULT AResult);
```

File

VarUtils

Description

Use SafeArrayCheck to check whether the parameter given through `AResult` is one of the possible safe array operation error codes.

See Also

SafeArrayError (see page 5120)

52.2.5 VarUtils.SafeArrayCopy Function

Copies a safe array to another.

Pascal

```
function SafeArrayCopy(SourceArray: PVarArray; out TargetArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayCopy(PVarArray SourceArray, PVarArray TargetArray);
```

File

VarUtils

Description

Use SafeArrayCopy to create a copy of the safe array given through `SourceArray` to the safe array stored in `TargetArray`.

In order to copy the safe array, SafeArrayCopy first checks if the source array is valid. SafeArrayCopy then locks the source array, allocates memory for the descriptor and for the actual data of the target array, and copies over the data from the source array to the target array. Finally, the source array is unlocked.

If SafeArrayCopy managed to copy the source array, then `TargetArray` contains the copied array. If SafeArrayCopy did not manage to copy over the safe array, it simply sets the `TargetArray` parameter to **nil**.

See Also

SafeArrayCreate (see page 5118)

SafeArrayCopyData (see page 5118)

52.2.6 VarUtils.SafeArrayCopyData Function

Copies the data from a safe array to another.

Pascal

```
function SafeArrayCopyData(SourceArray: PVarArray; TargetArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayCopyData(PVarArray SourceArray, PVarArray TargetArray);
```

File

VarUtils

Description

Use SafeArrayCopyData to copy over the data from the safe array given through `SourceArray` to the safe array stored in `TargetArray`.

In order to copy the data from the safe array, SafeArrayCopyData first checks if the source array is valid and also establishes that the source and target arrays are compatible. SafeArrayCopyData then locks the source array, clears up the data in the target array, and copies over the data from the source array. Finally, the source array is unlocked.

If SafeArrayCopyData managed to copy over the data from the source array, then `TargetArray` contains the copied data. If the two arrays are incompatible, SafeArrayCopyData returns a VAR_INVALIDARG (see page 5161) value. Otherwise, it returns VAR_OK (see page 5162)

See Also

SafeArrayCreate (see page 5118)

SafeArrayCopy (see page 5117)

52.2.7 VarUtils.SafeArrayCreate Function

Creates a safe array.

Pascal

```
function SafeArrayCreate(VarType: Integer; DimCount: Integer; const Bounds:
TVarArrayBoundArray): PVarArray; stdcall;
```

C++

```
__stdcall PVarArray SafeArrayCreate(int VarType, int DimCount, const TVarArrayBoundArray
Bounds);
```

File

VarUtils

Description

Use SafeArrayCreate to create a safe array with the given parameters.

The `VarType` parameter gives the type of each element in the safe array. Use the `var...` constants (varInteger (see page

3867), varBoolean (see page 3862), varDate (see page 3864), varCurrency (see page 3863) and so on) in the `System` (see page 3159) unit to specify the elements' variant type.

The `DimCount` parameter gives the safe array's number of dimensions. `DimCount` also represents the number of elements in the `Bounds` (see page 4651) constant array.

`Bounds` (see page 4651) is a constant TVarArrayBoundArray (see page 3798) type giving the bound information of the safe array, in all its dimensions.

In order to create the safe array, SafeArrayCreate first checks whether the element type is valid, then allocates memory for the safe array descriptor using the SafeArrayAllocDescriptor (see page 5116) function. Finally, it attempts to allocate memory for the actual data of the safe array, using SafeArrayAllocData (see page 5116). If this is not successful, then it destroys the previously allocated descriptor using the SafeArrayDestroyDescriptor (see page 5120) routine.

If SafeArrayCreate managed to create the safe array, then it returns a pointer to the array as a PVarArray (see page 3779) type variable. In the case when SafeArrayCreate did not manage to create the safe array, it simply returns **nil**.

To destroy a safe array, use the SafeArrayDestroy (see page 5119) function.

See Also

SafeArrayDestroy (see page 5119)

SafeArrayAllocData (see page 5116)

SafeArrayAllocDescriptor (see page 5116)

SafeArrayDestroyDescriptor (see page 5120)

TVarArrayBoundArray (see page 3798)

52.2.8 VarUtils.SafeArrayDestroy Function

Destroys a safe array.

Pascal

```
function SafeArrayDestroy(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayDestroy(PVarArray VarArray);
```

File

VarUtils

Description

Use SafeArrayDestroy to destroy safe array given through `VarArray`.

SafeArrayDestroy first checks whether the given safe array is locked or **nil**. If this is the case, then SafeArrayDestroy returns VAR_ARRAYISLOCKED (see page 5157) or VAR_INVALIDARG (see page 5161), correspondingly.

If the safe array is valid, then SafeArrayDestroy removes the actual data from memory using a call to SafeArrayDestroyData (see page 5120) and also destroys the associated safe array descriptor with a call to SafeArrayDestroyDescriptor (see page 5120). If SafeArrayDestroy manages to destroy the safe array, it returns a VAR_OK (see page 5162) result.

See Also

SafeArrayCreate (see page 5118)

SafeArrayDestroyData (see page 5120)

SafeArrayDestroyDescriptor (see page 5120)

52.2.9 VarUtils.SafeArrayDestroyData Function

Destroys the actual data of the given safe array.

Pascal

```
function SafeArrayDestroyData(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayDestroyData(PVarArray VarArray);
```

File

VarUtils

Description

SafeArrayDestroyData destroys the actual data of the given safe array. It is used internally by the SafeDestroy routine. We recommend you not to use it directly when writing your application.

See Also

SafeArrayDestroy (see page 5119)

52.2.10 VarUtils.SafeArrayDestroyDescriptor Function

Destroys the descriptor of the given safe array.

Pascal

```
function SafeArrayDestroyDescriptor(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayDestroyDescriptor(PVarArray VarArray);
```

File

VarUtils

Description

SafeArrayDestroyDescriptor destroys the descriptor of the given safe array. It is used internally by the SafeDestroy routine. We recommend you not to use it directly when writing your application.

See Also

SafeArrayDestroy (see page 5119)

52.2.11 VarUtils.SafeArrayError Function

Raises an event corresponding to the given safe array operation error code.

Pascal

```
procedure SafeArrayError(AResult: HRESULT);
```

C++

```
SafeArrayError(HRESULT AResult);
```

File

VarUtils

Description

Use SafeArrayError to programatically raise an event corresponding to the safe array operation error code given through the `AResult` parameter.

See Also

SafeArrayCheck (see page 5117)

52.2.12 VarUtils.SafeArrayGetDim Function

Returns the number of dimension of the given safe array.

Pascal

```
function SafeArrayGetDim(VarArray: PVarArray): Integer; stdcall;
```

C++

```
__stdcall int SafeArrayGetDim(PVarArray VarArray);
```

File

VarUtils

Description

Use SafeArrayGetDim to obtain the number of dimensions of the safe array referenced through the `VarArray` parameter.

See Also

SafeArrayGetLBound (see page 5122)

SafeArrayGetUBound (see page 5123)

SafeArrayGetElemSize (see page 5122)

52.2.13 VarUtils.SafeArrayGetElement Function

Returns the element at the specified position in the given safe array.

Pascal

```
function SafeArrayGetElement(VarArray: PVarArray; Indices: PVarArrayCoorArray; Data: Pointer):
HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayGetElement(PVarArray VarArray, PVarArrayCoorArray Indices, void *
Data);
```

File

VarUtils

Description

Use SafeArrayGetElement to obtain the element found at the position specified through the `Indices` parameter, in the safe array given through `VarArray`.

If retrieving the element is successful, SafeArrayGetElement stores the element in the `Data` parameter.

Note: SafeArrayGetElement also locks and unlocks the safe array before and after retrieving the element, using the SafeArrayLock (see page 5123) and SafeArrayUnlock (see page 5127) routines.

See Also

SafeArrayPutElement (see page 5125)

SafeArrayLock (see page 5123)

SafeArrayUnlock (see page 5127)

52.2.14 VarUtils.SafeArrayGetElemSize Function

Returns the size in bytes of an element in the given safe array.

Pascal

```
function SafeArrayGetElemSize(VarArray: PVarArray): LongWord; stdcall;
```

C++

```
__stdcall LongWord SafeArrayGetElemSize(PVarArray VarArray);
```

File

VarUtils

Description

Use SafeArrayGetElemSize to obtain the size in bytes of an element in the safe array referenced through the `VarArray` parameter.

See Also

SafeArrayGetLBound (see page 5122)

SafeArrayGetUBound (see page 5123)

SafeArrayGetDim (see page 5121)

52.2.15 VarUtils.SafeArrayGetLBound Function

Returns the lower bound for the given dimension of a safe array.

Pascal

```
function SafeArrayGetLBound(VarArray: PVarArray; Dim: Integer; out LBound: Integer): HRESULT;
stdcall;
```

C++

```
__stdcall HRESULT SafeArrayGetLBound(PVarArray VarArray, int Dim, int LBound);
```

File

VarUtils

Description

Use SafeArrayGetLBound to obtain the lower bound for the dimension given through `Dim` of the safe array referenced by `VarArray`.

See Also

SafeArrayGetUBound (see page 5123)

SafeArrayGetDim (see page 5121)

SafeArrayGetElemSize (see page 5122)

52.2.16 VarUtils.SafeArrayGetUBound Function

Returns the upper bound for the given dimension of a safe array.

Pascal

```
function SafeArrayGetUBound(VarArray: PVarArray; Dim: Integer; out UBound: Integer): HRESULT;
stdcall;
```

C++

```
__stdcall HRESULT SafeArrayGetUBound(PVarArray VarArray, int Dim, int UBound);
```

File

VarUtils

Description

Use SafeArrayGetUBound to obtain the upper bound for the dimension given through `Dim` of the safe array referenced by `VarArray`.

See Also

SafeArrayGetLBound (see page 5122)

SafeArrayGetDim (see page 5121)

SafeArrayGetElemSize (see page 5122)

52.2.17 VarUtils.SafeArrayLock Function

Locks a safe array.

Pascal

```
function SafeArrayLock(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayLock(PVarArray VarArray);
```

File

VarUtils

Description

SafeArrayLock locks the safe array given through the `VarArray` parameter. `VarArray` is a pointer to the safe array.

The possible return values of SafeArrayLock are listed in the following table, together with their meaning.

Value	Meaning
VAR_OK (see page 5162)	The operation was completed successfully.
VAR_INVALIDARG (see page 5161)	The given argument is `nil`.
VAR_ARRAYISLOCKED (see page 5157)	The array given through `VarArray` is locked.

While locked, a safe array cannot be resized and any calls to SafeArrayRedim (see page 5125) fail. Once locked, a safe array must later be unlocked using SafeArrayUnlock (see page 5127).

If the locking operation succeeded, `VarArray` points to an array of elements in which the leftmost dimension increases first. In other words, the dimensions of the returned array pointer are reversed from the dimensions of the safe array.

Tip: Once SafeArrayLock verifies that the safe array has the correct bounds and dimensions, SafeArrayLock can be used to gain direct access to the array data, which results in improved performance.

See Also

SafeArrayUnlock (see page 5127)

SafeArrayRedim (see page 5125)

52.2.18 VarUtils.SafeArrayPtrOfIndex Function

Returns a pointer to the element at the specified position in the given safe array.

Pascal

```
function SafeArrayPtrOfIndex(VarArray: PVarArray; Indices: PVarArrayCoorArray; var Address:
Pointer): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayPtrOfIndex(PVarArray VarArray, PVarArrayCoorArray Indices, void *
Address);
```

File

VarUtils

Description

Use SafeArrayPtrOfIndex to obtain a pointer to the element found at the position specified through the `Indices` parameter, in the safe array given through `VarArray`.

If retrieving the pointer to the element is successful, SafeArrayPtrOfIndex stores the element in the `Address` parameter.

See Also

SafeArrayGetElement (see page 5121)

SafeArrayPutElement (see page 5125)

52.2.19 VarUtils.SafeArrayPutElement Function

Sets the element at the specified position in the given safe array.

Pascal

```
function SafeArrayPutElement(VarArray: PVarArray; Indices: PVarArrayCoorArray; const Data:
Pointer): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayPutElement(PVarArray VarArray, PVarArrayCoorArray Indices, const
void * Data);
```

File

VarUtils

Description

Use SafeArrayPutElement to change the value of the element found at the position specified through the `Indices` parameter, in the safe array given through `VarArray`. The new value is stored in the `Data` parameter.

Note: SafeArrayPutElement also locks and unlocks the safe array before and after setting the element, using the SafeArrayLock (see page 5123) and SafeArrayUnlock (see page 5127) routines.

See Also

SafeArrayGetElement (see page 5121)

SafeArrayLock (see page 5123)

SafeArrayUnlock (see page 5127)

52.2.20 VarUtils.SafeArrayRedim Function

Changes the bound information for the last dimension of a safe array.

Pascal

```
function SafeArrayRedim(VarArray: PVarArray; const NewBound: TVarArrayBound): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayRedim(PVarArray VarArray, const TVarArrayBound NewBound);
```

File

VarUtils

Description

SafeArrayRedim resizes the safe array given through the `VarArray` parameter, by changing the bound information for the last dimension of the array, according to the `NewBound` parameter. `VarArray` is a pointer to the safe array.

The possible return values of SafeArrayRedim are listed in the following table, together with their meaning.

Value	Meaning
VAR_OK (see page 5162)	The operation was completed successfully.
VAR_INVALIDARG (see page 5161)	The given argument is `nil`.
VAR_ARRAYISLOCKED (see page 5157)	The array given through `VarArray` is locked.
VAR_EXCEPTION (see page 5160)	The application raised an exception.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the operation.

After the resizing operation took place, the existing elements of the array are preserved and the new elements are set to zero or empty.

Note: A safe array cannot be resized while it is locked.

See Also

SafeArrayLock (see page 5123)

SafeArrayUnlock (see page 5127)

52.2.21 VarUtils.SafeArrayUnaccessData Function

Unlocks a safe array previously locked using the SafeArrayAccessData (see page 5115) routine.

Pascal

```
function SafeArrayUnaccessData(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayUnaccessData(PVarArray VarArray);
```

File

VarUtils

Description

Use SafeArrayUnaccessData to unlock a safe array that was previously locked with a call to SafeArrayAccessData (see page 5115).

Calling SafeArrayUnaccessData is equivalent to making a call to SafeArrayUnlock (see page 5127) with the same safe array as parameter.

See Also

SafeArrayAccessData (see page 5115)

SafeArrayUnlock (see page 5127)

52.2.22 VarUtils.SafeArrayUnlock Function

Unlocks a safe array.

Pascal

```
function SafeArrayUnlock(VarArray: PVarArray): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT SafeArrayUnlock(PVarArray VarArray);
```

File

VarUtils

Description

SafeArrayUnlock unlocks the safe array given through the `VarArray` parameter. `VarArray` is a pointer to the safe array.

The possible return values of SafeArrayUnlock are listed in the table below, together with their meaning.

Value	Meaning
VAR_OK (see page 5162)	The operation was completed successfully.
VAR_INVALIDARG (see page 5161)	The given argument is `nil`.
VAR_ARRAYISLOCKED (see page 5157)	The array given through `VarArray` is locked.

If the unlocking operation succeeded, `VarArray` points to the unlocked safe array.

The safe array referenced by `VarArray` is assumed to have been locked using the SafeArrayLock (see page 5123) function.

See Also

SafeArrayLock (see page 5123)

52.2.23 VarUtils.VarExceptionToResult Function

Converts an Exception (see page 3949) type argument to an HRESULT (see page 3754) type.

Pascal

```
function VarExceptionToResult(const E: Exception): HRESULT;
```

C++

```
HRESULT VarExceptionToResult(const Exception E);
```

File

VarUtils

Description

VarExceptionToResult is an internal routine that converts the value given through E from an Exception (see page 3949) type to an HRESULT (see page 3754) type.

52.2.24 VarUtils.VariantChangeType Function

Converts a variant from one type to another.

Pascal

```
function VariantChangeType(var Dest: TVarData; const Source: TVarData; wFlags: Word; VarType:
Word): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantChangeType(TVarData Dest, const TVarData Source, Word wFlags, Word
VarType);
```

File

VarUtils

Description

Use VariantChangeType to convert the variant given in `Source` to the variant stored in `Dest` of type `VarType`, using the parameter `wFlags`, which specifies the conversion flags.

VariantChangeType returns the result of the conversion operation, which can be one of the following.

Value	Meaning
`VAR_OK` (see page 5162)	The conversion was successful.
`VAR_BADVARTYPE` (see page 5158)	The variant type of the input parameter is not valid.
`VAR_OVERFLOW` (see page 5163)	The data stored in the output parameter does not fit in the destination type.
`VAR_TYPEMISMATCH` (see page 5163)	The argument cannot be cast to the specified type.
`VAR_INVALIDARG` (see page 5161)	Either one of the arguments is invalid.
`VAR_OUTOFMEMORY` (see page 5162)	There is not enough memory to complete the conversion.

Note: To do the conversion using locale-specific information, use the function referenced by the VariantChangeTypeEx (see page 5142) variable.

See Also

VariantChangeTypeEx (see page 5142)

52.2.25 VarUtils.VariantClear Function

Clears a variant.

Pascal

```
function VariantClear(var V: TVarData): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantClear(TVarData V);
```

File

VarUtils

Description

VariantClear first clears the variant given through V and then it reinitializes it, using a call to VariantInit (see page 5130).

The possible return values of VariantClear are given in the following table, together with their meaning.

Value	Meaning
VAR_OK (see page 5162)	The operation was completed successfully.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_UNEXPECTED (see page 5164)	An unexpected error occurred.

See Also

VariantInit (see page 5130)

52.2.26 VarUtils.VariantCopy Function

Clears the destination variant and makes a copy of the source variant.

Pascal

```
function VariantCopy(var Dest: TVarData; const Source: TVarData): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantCopy(TVarData Dest, const TVarData Source);
```

File

VarUtils

Description

Use VariantCopy to copy the source variant given through `Source` to the destination variant denoted by `Dest`, after clearing up `Dest` using a call to VariantClear (see page 5128).

The possible return values of VariantCopy are given in the following table, together with their meaning.

Value	Meaning
VAR_OK (see page 5162)	The operation was completed successfully.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the operation.
VAR_UNEXPECTED (see page 5164)	An unexpected error occurred.
VAR_EXCEPTION (see page 5160)	The application raised an exception.

See Also

VariantCopyInd (see page 5130)

52.2.27 VarUtils.VariantCopyInd Function

Clears the destination variant and makes a copy of the source variant, including the case when the source is of varByRef (see page 3862) type.

Pascal

```
function VariantCopyInd(var Dest: TVarData; const Source: TVarData): HRESULT; stdcall;
```

C++

```
__stdcall HRESULT VariantCopyInd(TVarData Dest, const TVarData Source);
```

File

VarUtils

Description

Use VariantCopyInd to copy the source variant given through `Source` to the destination variant denoted by `Dest`, after clearing up `Dest` using a call to VariantClear (see page 5128).

In the case when the source is of varByRef (see page 3862) type, VariantCopyInd performs the indirection and copies the correct information.

The possible return values of VariantCopyInd are given in the following table, together with their meaning.

Value	Meaning
VAR_OK (see page 5162)	The operation was completed successfully.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the operation.
VAR_UNEXPECTED (see page 5164)	An unexpected error occurred.
VAR_EXCEPTION (see page 5160)	The application raised an exception.

See Also

VariantCopy (see page 5129)

52.2.28 VarUtils.VariantInit Function

Initializes a variant.

Pascal

```
procedure VariantInit(var V: TVarData); stdcall;
```

C++

```
__stdcall VariantInit(TVarData V);
```

File

VarUtils

Description

Use the VariantInit procedure to initialize the variant given through the V parameter.

VariantInit sets the VType field of V to varEmpty (see page 3865) and sets all its bytes to zero.

See Also

VariantClear (see page 5128)

52.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TVarTypeToElementInfo (see page 5132)	This is an internal record type used in declaring the CVarTypeToElementInfo (see page 5154) constant array.

52.3.1 VarUtils.TVarTypeToElementInfo Record

This is an internal record type used in declaring the CVarTypeToElementInfo (see page 5154) constant array.

Pascal

```
TVarTypeToElementInfo = record
  ValidBase: Boolean;
  ValidElement: Boolean;
  Size: Integer;
  Flags: Word;
end;
```

C++

```
struct TVarTypeToElementInfo {
  Boolean ValidBase;
  Boolean ValidElement;
  int Size;
  Word Flags;
};
```

File

VarUtils

Description

TVarTypeToElementInfo is an internal record type that specifies the elements' type in the CVarTypeToElementInfo (see page 5154) constant array. We recommend you not to use it directly when writing your application.

See Also

CVarTypeToElementInfo (see page 5154)

52.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
VarAdd (see page 5133)	VarAdd is a function type variable that specifies the function making the addition operation between two variants.
VarAnd (see page 5134)	VarAnd is a function type variable that specifies the function making the bitwise **and** operation between two variants.
VarBoolFromStr (see page 5134)	VarBoolFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a boolean value.
VarBstrFromBool (see page 5135)	VarBstrFromBool is used to specify the function that makes the conversion from a boolean value to a WideString (see page 3467).
VarBstrFromCy (see page 5136)	VarBstrFromCy is used to specify the function that makes the conversion from a currency value to a WideString (see page 3467).
VarBstrFromDate (see page 5137)	VarBstrFromDate is used to specify the function that makes the conversion from a date value to a WideString (see page 3467).
VarCmp (see page 5138)	VarCmp is a variable used to specify the function that compares two variants.
VarCyFromStr (see page 5139)	VarCyFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a currency value.
VarDateFromStr (see page 5139)	VarDateFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a date value.
VarDiv (see page 5140)	VarDiv is a function type variable that specifies the function making the division operation between two variants.
VarI4FromStr (see page 5141)	VarI4FromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a four bytes integer value.
VariantChangeTypeEx (see page 5142)	VariantChangeTypeEx stores the function that converts a variant from one type to another, using locale-specific information.
VarIDiv (see page 5142)	VarIDiv is a function type variable that specifies the function making the division operation between two variants, after converting the variants to integers.
VarMod (see page 5143)	VarMod is a function type variable that specifies the function that calculates the remainder on division of two variants.
VarMul (see page 5143)	VarMul is a function type variable that specifies the function making the multiplication operation between two variants.
VarNeg (see page 5144)	VarNeg is a function type variable that specifies the function that inverts the sign of a variant.
VarNot (see page 5144)	VarNot is a function type variable that specifies the function that performs a **not** operation on a variant.
VarOr (see page 5145)	VarOr is a function type variable that specifies the function doing the bitwise **or** operation between two variants.
VarR4FromStr (see page 5145)	VarR4FromStr is used to specify the function that does the conversion from a WideString (see page 3467) to a four bytes floating-point value.
VarR8FromStr (see page 5146)	VarR8FromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a eight bytes floating-point value.
VarSub (see page 5147)	VarSub is a function type variable that specifies the function making the subtraction operation between two variants.
VarXor (see page 5147)	VarXor is a function type variable that specifies the function making the bitwise **xor** (exclusive or) operation between two variants.

52.4.1 VarUtils.VarAdd Variable

VarAdd is a function type variable that specifies the function making the addition operation between two variants.

Pascal

```
VarAdd: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarAdd;
```

File

VarUtils

Description

Use VarAdd to specify the function that makes the addition operation between two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantAdd (see page 4955)

52.4.2 VarUtils.VarAnd Variable

VarAnd is a function type variable that specifies the function making the bitwise **and** operation between two variants.

Pascal

```
VarAnd: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarAnd;
```

File

VarUtils

Description

Use VarAnd to specify the function that does the bitwise **and** operation between two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantAnd (see page 4956)

52.4.3 VarUtils.VarBoolFromStr Variable

VarBoolFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a boolean value.

Pascal

```
VarBoolFromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out
boolOut: WordBool): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out boolOut: WordBool):
HRESULT VarBoolFromStr;
```

File

VarUtils

Description

Use VarBoolFromStr to specify the function that makes the conversion from the WideString (see page 3467) given through `strIn` to the boolean value stored in `boolOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarBoolFromStr returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

To make the inverse conversion, from a boolean value to a WideString (see page 3467), use the function referenced by VarBstrFromBool (see page 5135).

See Also

VarBstrFromBool (see page 5135)

52.4.4 VarUtils.VarBstrFromBool Variable

VarBstrFromBool is used to specify the function that makes the conversion from a boolean value to a WideString (see page 3467).

Pascal

```
VarBstrFromBool: function(boolIn: WordBool; LCID: Integer; dwFlags: Longint; out bstrOut:
WideString): HRESULT;
```

C++

```
function(boolIn: WordBool; LCID: Integer; dwFlags: Longint; out bstrOut: WideString): HRESULT
VarBstrFromBool;
```

File

VarUtils

Description

Use VarBstrFromBool to specify the function that makes the conversion from the boolean value given through `boolIn` to the WideString (see page 3467) stored in `bstrOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarBstrFromBool returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

To make the inverse conversion, from a WideString (see page 3467) to a boolean value, use the function referenced by VarBoolFromStr (see page 5134).

See Also

VarBoolFromStr (see page 5134)

52.4.5 VarUtils.VarBstrFromCy Variable

VarBstrFromCy is used to specify the function that makes the conversion from a currency value to a WideString (see page 3467).

Pascal

```
VarBstrFromCy: function(cyIn: Currency; LCID: Integer; dwFlags: Longint; out bstrOut:
WideString): HRESULT;
```

C++

```
function(cyIn: Currency; LCID: Integer; dwFlags: Longint; out bstrOut: WideString): HRESULT
VarBstrFromCy;
```

File

VarUtils

Description

Use VarBstrFromCy to specify the function that makes the conversion from the currency value given through `cyIn` to the WideString (see page 3467) stored in `bstrOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarBstrFromCy returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

To make the inverse conversion, from a WideString (see page 3467) to a currency value, use the function referenced by VarCyFromStr (see page 5139).

See Also

VarCyFromStr (see page 5139)

52.4.6 VarUtils.VarBstrFromDate Variable

VarBstrFromDate is used to specify the function that makes the conversion from a date value to a WideString (see page 3467).

Pascal

```
VarBstrFromDate: function(dateIn: TDateTime; LCID: Integer; dwFlags: Longint; out bstrOut:
WideString): HRESULT;
```

C++

```
function(dateIn: TDateTime; LCID: Integer; dwFlags: Longint; out bstrOut: WideString): HRESULT
VarBstrFromDate;
```

File

VarUtils

Description

Use VarBstrFromDate to specify the function that makes the conversion from the date value given through `dateIn` to the WideString (see page 3467) stored in `bstrOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarBstrFromDate returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.

VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

To make the inverse conversion, from a WideString (see page 3467) to a date value, use the function referenced by VarDateFromStr (see page 5139).

See Also

VarDateFromStr (see page 5139)

52.4.7 VarUtils.VarCmp Variable

VarCmp is a variable used to specify the function that compares two variants.

Pascal

```
VarCmp: function(const Left, Right: TVarData; LCID: Integer; Flags: LongWord): HRESULT;
```

C++

```
function(const Left, Right: TVarData; LCID: Integer; Flags: LongWord): HRESULT VarCmp;
```

File

VarUtils

Description

Use VarCmp to specify the function that compares two variants, given through the `Left` and `Right` parameters. The `LCID` parameter is a locale identifier and `Flags` specifies the flags used in the comparison.

VarCmp returns the result of the comparison, which can be one of the following.

Result	Meaning
`VAR_CMP_LT` (see page 5159)	`Left` is less than `Right`
`VAR_CMP_EQ` (see page 5158)	`Left` is equal to `Right`
`VAR_CMP_GT` (see page 5159)	`Left` is greater than `Right`
`VAR_CMP_NULL` (see page 5160)	Either one of `Left` or `Right` is `nil`.

See Also

VAR_CMP_LT (see page 5159)

VAR_CMP_EQ (see page 5158)

VAR_CMP_GT (see page 5159)

VAR_CMP_NULL (see page 5160)

52.4.8 VarUtils.VarCyFromStr Variable

VarCyFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a currency value.

Pascal

```
VarCyFromStr: function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out cyOut:
Currency): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out cyOut: Currency): HRESULT
VarCyFromStr;
```

File

VarUtils

Description

Use VarCyFromStr to specify the function that makes the conversion from the WideString (see page 3467) given through `strIn` to the currency value stored in `cyOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarCyFromStr returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

To make the inverse conversion, from a currency value to a WideString (see page 3467), use the function referenced by VarBstrFromCy (see page 5136).

See Also

VarBstrFromCy (see page 5136)

52.4.9 VarUtils.VarDateFromStr Variable

VarDateFromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a date value.

Pascal

```
VarDateFromStr: function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out dateOut:
TDateTime): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: DWORD; dwFlags: Longint; out dateOut: TDateTime):
HRESULT VarDateFromStr;
```

File

VarUtils

Description

Use VarDateFromStr to specify the function that makes the conversion from the WideString (see page 3467) given through `strIn` to the date value stored in `dateOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarDateFromStr returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

To make the inverse conversion, from a date value to a WideString (see page 3467), use the function referenced by VarBstrFromDate (see page 5137).

See Also

VarBstrFromDate (see page 5137)

52.4.10 VarUtils.VarDiv Variable

VarDiv is a function type variable that specifies the function making the division operation between two variants.

Pascal

```
VarDiv: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarDiv;
```

File

VarUtils

Description

Use VarDiv to specify the function that makes the division operation between two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils (see page 5107)` unit.

See Also

VariantDiv (see page 4959)

52.4.11 VarUtils.VarI4FromStr Variable

VarI4FromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a four bytes integer value.

Pascal

```
VarI4FromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out lOut:
Longint): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out lOut: Longint): HRESULT
VarI4FromStr;
```

File

VarUtils

Description

Use VarI4FromStr to specify the function that makes the conversion from the WideString (see page 3467) given through `strIn` to the four bytes integer value stored in `lOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarI4FromStr returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

See Also

VarR4FromStr (see page 5145)

VarR8FromStr (see page 5146)

52.4.12 VarUtils.VariantChangeTypeEx Variable

VariantChangeTypeEx stores the function that converts a variant from one type to another, using locale-specific information.

Pascal

```
VariantChangeTypeEx: function(var Dest: TVarData; const Source: TVarData; LCID: Integer;
wFlags: Word; VarType: Word): HRESULT;
```

C++

```
function(var Dest: TVarData; const Source: TVarData; LCID: Integer; wFlags: Word; VarType:
Word): HRESULT VariantChangeTypeEx;
```

File

VarUtils

Description

Use VariantChangeTypeEx to specify the function that converts the variant given in `Source` to the variant stored in `Dest` of type `VarType`, using the locale-specific information given through `LCID` and the parameter `wFlags` specifying the conversion flags.

The function stored in VariantChangeTypeEx returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

Note: To make the conversion without using locale-specific information, use the VariantChangeType (see page 5128) function.

See Also

VariantChangeType (see page 5128)

52.4.13 VarUtils.VarIDiv Variable

VarIDiv is a function type variable that specifies the function making the division operation between two variants, after converting the variants to integers.

Pascal

```
VarIDiv: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarIDiv;
```

File

VarUtils

Description

Use VarIDiv to specify the function that makes the division operation between the two variants given through the `Left` and `Right` parameters, after converting the variants to integers. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VarDiv (see page 5140)

VariantDiv (see page 4959)

52.4.14 VarUtils.VarMod Variable

VarMod is a function type variable that specifies the function that calculates the remainder on division of two variants.

Pascal

```
VarMod: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarMod;
```

File

VarUtils

Description

Use VarMod to specify the function that calculates the remainder on the division of the two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantMod (see page 4962)

52.4.15 VarUtils.VarMul Variable

VarMul is a function type variable that specifies the function making the multiplication operation between two variants.

Pascal

```
VarMul: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarMul;
```

File

VarUtils

Description

Use VarMul to specify the function that makes the multiplication operation between two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantMul (see page 4963)

52.4.16 VarUtils.VarNeg Variable

VarNeg is a function type variable that specifies the function that inverts the sign of a variant.

Pascal

```
VarNeg: function(const Source: TVarData; var Dest: TVarData): HRESULT;
```

C++

```
function(const Source: TVarData; var Dest: TVarData): HRESULT VarNeg;
```

File

VarUtils

Description

Use VarNeg to specify the function that inverts the sign of the variant given through the `Source` parameter. The result is stored in the `Dest` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantNeg (see page 4964)

52.4.17 VarUtils.VarNot Variable

VarNot is a function type variable that specifies the function that performs a **not** operation on a variant.

Pascal

```
VarNot: function(const Source: TVarData; var Dest: TVarData): HRESULT;
```

C++

```
function(const Source: TVarData; var Dest: TVarData): HRESULT VarNot;
```

File

VarUtils

Description

Use VarNot to specify the function that performs a **not** operation on the variant given through the `Source` parameter. The result is stored in the `Dest` parameter.

The function specified through VarNot returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantNeg (see page 4964)

52.4.18 VarUtils.VarOr Variable

VarOr is a function type variable that specifies the function doing the bitwise **or** operation between two variants.

Pascal

```
VarOr: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarOr;
```

File

VarUtils

Description

Use VarOr to specify the function that does the bitwise **or** operation between two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantOr (see page 4964)

52.4.19 VarUtils.VarR4FromStr Variable

VarR4FromStr is used to specify the function that does the conversion from a WideString (see page 3467) to a four bytes floating-point value.

Pascal

```
VarR4FromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out fltOut:
Single): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out fltOut: Single):
HRESULT VarR4FromStr;
```

File

VarUtils

Description

Use VarR4FromStr to specify the function that makes the conversion from the WideString (see page 3467) given through `strIn` to the four bytes floating-point value stored in `fltOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarR4FromStr returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

See Also

VarR8FromStr (see page 5146)

52.4.20 VarUtils.VarR8FromStr Variable

VarR8FromStr is used to specify the function that makes the conversion from a WideString (see page 3467) to a eight bytes floating-point value.

Pascal

```
VarR8FromStr: function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out dblOut:
Double): HRESULT;
```

C++

```
function(const strIn: WideString; LCID: Integer; dwFlags: Longint; out dblOut: Double):
HRESULT VarR8FromStr;
```

File

VarUtils

Description

Use VarR8FromStr to specify the function that makes the conversion from the WideString (see page 3467) given through `strIn` to the eight bytes floating-point value stored in `dblOut`. The `LCID` parameter is a locale identifier and `dwFlags` specifies the flags used in the conversion.

The function stored in VarR8FromStr returns the result of the conversion operation, which can be one of the following.

Value	Meaning
VAR_OK (see page 5162)	The conversion was successful.
VAR_BADVARTYPE (see page 5158)	The variant type of the input parameter is not valid.
VAR_OVERFLOW (see page 5163)	The data stored in the output parameter does not fit in the destination type.
VAR_TYPEMISMATCH (see page 5163)	The argument cannot be cast to the specified type.
VAR_INVALIDARG (see page 5161)	Either one of the arguments is invalid.
VAR_OUTOFMEMORY (see page 5162)	There is not enough memory to complete the conversion.

See Also

VarR4FromStr (see page 5145)

52.4.21 VarUtils.VarSub Variable

VarSub is a function type variable that specifies the function making the subtraction operation between two variants.

Pascal

```
VarSub: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarSub;
```

File

VarUtils

Description

Use VarSub to specify the function that makes the subtraction operation between two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantSub (see page 4968)

52.4.22 VarUtils.VarXor Variable

VarXor is a function type variable that specifies the function making the bitwise **xor** (exclusive or) operation between two variants.

Pascal

```
VarXor: function(const Left, Right: TVarData; var AResult: TVarData): HRESULT;
```

C++

```
function(const Left, Right: TVarData; var AResult: TVarData): HRESULT VarXor;
```

File

VarUtils

Description

Use VarXor to specify the function that makes the bitswise **xor** (exclusive or) operation between two variants given through the `Left` and `Right` parameters. The result is stored in the `AResult` parameter.

The function specified through VarNot (see page 5144) returns a value of type HRESULT (see page 3754). For a list of possible return values, see the VAR_ constants in the `VarUtils` (see page 5107) unit.

See Also

VariantXor (see page 4970)

52.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ARR_DISPATCH (see page 5150)	Used to test whether the variant array contains IDispatch (see page 3494) elements.
ARR_FIXEDSIZE (see page 5150)	Used to test whether the variant array is of fixed size.
ARR_NONE (see page 5151)	Specifies that a variant array has no flags set.
ARR_OLESTR (see page 5151)	Used to test whether the variant array contains BSTR elements.
ARR_UNKNOWN (see page 5152)	Used to test whether the variant array contains IUnknown (see page 3756) elements.
ARR_VARIANT (see page 5152)	Used to test whether the variant array contains `Variant (see page 3421)` elements.
CMaxArrayVarType (see page 5153)	CMaxArrayVarType gives the upper bound of the CVarTypeToElementInfo (see page 5154) array.
CMinArrayVarType (see page 5153)	CMinArrayVarType gives the lower bound of the CVarTypeToElementInfo (see page 5154) array.
CVarTypeToElementInfo (see page 5154)	This constant is internally used by the SafeArrayCreate (see page 5118) function.
GenericOperations (see page 5155)	Specifies whether the implementation for the variant operation routines is cross-platform.
GenericSafeArrays (see page 5155)	Specifies whether the implementation for the safe array routines is cross-platform.
GenericSafeArrayUsesLibC (see page 5156)	Specifies whether the implementation for the safe array routines uses the `Libc` unit.
GenericVariants (see page 5156)	Specifies whether the implementation for the variant support routines is cross-platform.
GenericVarUtils (see page 5157)	Specifies whether the implementation for the variant-manipulating routines is cross-platform.
VAR_ARRAYISLOCKED (see page 5157)	Specifies that the given array is locked.
VAR_BADINDEX (see page 5158)	Indicates that the given index is out of range.
VAR_BADVARTYPE (see page 5158)	Specifies that the variant type of the input parameter is invalid.
VAR_CMP_EQ (see page 5158)	VAR_CMP_EQ specifies that the variants are equal, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_CMP_GT (see page 5159)	VAR_CMP_GT specifies that the first variant is greater than the second, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_CMP_LT (see page 5159)	VAR_CMP_LT specifies that the first variant is lower than the second, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_CMP_NULL (see page 5160)	VAR_CMP_NULL specifies that either one of the variants is `nil`, after a call to the function referenced by the VarCmp (see page 5138) variable.
VAR_EXCEPTION (see page 5160)	Specifies that the application raised an exception.
VAR_INVALIDARG (see page 5161)	Specifies that either one of the arguments is invalid.
VAR_LOCALE_USER_DEFAULT (see page 5161)	Specifies the default locale for the user or process.
VAR_NOTIMPL (see page 5161)	Specifies that the operation is not implemented.
VAR_OK (see page 5162)	Specifies that a variant operation was successful.
VAR_OUTOFMEMORY (see page 5162)	Specifies that there is not enough memory to complete the operation.
VAR_OVERFLOW (see page 5163)	Specifies that a variant operation caused an overflow.
VAR_PARAMNOTFOUND (see page 5163)	Indicates that there is no correspondence between the parameter DISPIDs and a parameter of the method.
VAR_TYPEMISMATCH (see page 5163)	Specifies that the argument cannot be cast to the specified type.
VAR_UNEXPECTED (see page 5164)	Indicates that an unexpected error occurred.

52.5.1 VarUtils.ARR_DISPATCH Constant

Used to test whether the variant array contains IDispatch (see page 3494) elements.

Pascal

```
ARR_DISPATCH = $0400;
```

C++

```
ARR_DISPATCH = 0x0400;
```

File

VarUtils

Description

Do a bitwise **and** operation between the `Flags` field of a TVarArray (see page 3739) variable and the ARR_DISPATCH constant to test whether the variant array contains IDispatch (see page 3494) elements. If the result is nonzero, then the elements of the variant array are of IDispatch (see page 3494) type. Otherwise, the elements are of a different type.

See Also

ARR_FIXEDSIZE (see page 5150)

ARR_NONE (see page 5151)

ARR_OLESTR (see page 5151)

ARR_UNKNOWN (see page 5152)

ARR_VARIANT (see page 5152)

52.5.2 VarUtils.ARR_FIXEDSIZE Constant

Used to test whether the variant array is of fixed size.

Pascal

```
ARR_FIXEDSIZE = $0010;
```

C++

```
ARR_FIXEDSIZE = 0x0010;
```

File

VarUtils

Description

Do a bitwise **and** operation between the `Flags` field of a TVarArray (see page 3739) variable and the ARR_FIXEDSIZE constant to test whether the variant array is of fixed size. If the result is nonzero, then the array is of fixed size. Otherwise, the array can be resized or reallocated.

See Also

ARR_DISPATCH (see page 5150)

ARR_NONE (see page 5151)

ARR_OLESTR (see page 5151)

ARR_UNKNOWN (see page 5152)

ARR_VARIANT (see page 5152)

52.5.3 VarUtils.ARR_NONE Constant

Specifies that a variant array has no flags set.

Pascal

```
ARR_NONE = $0000;
```

C++

```
ARR_NONE = 0x0000;
```

File

VarUtils

Description

Assign the ARR_NONE value to the `Flags` field of a TVarArray (see page 3739) variable to specify that the variant array variable has no flags set.

See Also

ARR_DISPATCH (see page 5150)

ARR_FIXEDSIZE (see page 5150)

ARR_OLESTR (see page 5151)

ARR_UNKNOWN (see page 5152)

ARR_VARIANT (see page 5152)

52.5.4 VarUtils.ARR_OLESTR Constant

Used to test whether the variant array contains BSTR elements.

Pascal

```
ARR_OLESTR = $0100;
```

C++

```
ARR_OLESTR = 0x0100;
```

File

VarUtils

Description

Do a bitwise **and** operation between the `Flags` field of a TVarArray (see page 3739) variable and the ARR_OLESTR constant to test whether the variant array contains BSTR elements. If the result is nonzero, then the elements of the variant array are of BSTR type. Otherwise, the elements are of a different type.

See Also

ARR_DISPATCH (see page 5150)

ARR_FIXEDSIZE (see page 5150)

ARR_NONE (see page 5151)

ARR_UNKNOWN (see page 5152)

ARR_VARIANT (see page 5152)

52.5.5 VarUtils.ARR_UNKNOWN Constant

Used to test whether the variant array contains IUnknown (see page 3756) elements.

Pascal

```
ARR_UNKNOWN = $0200;
```

C++

```
ARR_UNKNOWN = 0x0200;
```

File

VarUtils

Description

Do a bitwise **and** operation between the `Flags` field of a TVarArray (see page 3739) variable and the ARR_UNKNOWN constant to test whether the variant array contains IUnknown (see page 3756) elements. If the result is nonzero, then the elements of the variant array are of IUnknown (see page 3756) type. Otherwise, the elements are of a different type.

See Also

ARR_DISPATCH (see page 5150)

ARR_FIXEDSIZE (see page 5150)

ARR_NONE (see page 5151)

ARR_OLESTR (see page 5151)

ARR_VARIANT (see page 5152)

52.5.6 VarUtils.ARR_VARIANT Constant

Used to test whether the variant array contains `Variant (see page 3421)` elements.

Pascal

```
ARR_VARIANT = $0800;
```

C++

```
ARR_VARIANT = 0x0800;
```

File

VarUtils

Description

Do a bitwise **and** operation between the `Flags` field of a TVarArray (see page 3739) variable and the ARR_VARIANT constant to test whether the variant array contains `Variant (see page 3421)` elements. If the result is nonzero, then the elements of the variant array are of `Variant (see page 3421)` type. Otherwise, the elements are of a different type.

See Also

ARR_DISPATCH (see page 5150)

ARR_FIXEDSIZE (see page 5150)

ARR_NONE (see page 5151)

ARR_OLESTR (see page 5151)

ARR_UNKNOWN (see page 5152)

52.5.7 VarUtils.CMaxArrayVarType Constant

CMaxArrayVarType gives the upper bound of the CVarTypeToElementInfo (see page 5154) array.

Pascal

```
CMaxArrayVarType = varUInt64;
```

C++

```
CMaxArrayVarType = varUInt64;
```

File

VarUtils

Description

CMaxArrayVarType is a constant that specifies the upper bound of the CVarTypeToElementInfo (see page 5154) array.

CMaxArrayVarType is equal to the varUInt64 (see page 3872) constant.

See Also

CVarTypeToElementInfo (see page 5154)

52.5.8 VarUtils.CMinArrayVarType Constant

CMinArrayVarType gives the lower bound of the CVarTypeToElementInfo (see page 5154) array.

Pascal

```
CMinArrayVarType = varEmpty;
```

C++

```
CMinArrayVarType = varEmpty;
```

File

VarUtils

Description

CMinArrayVarType is a constant that specifies the lower bound of the CVarTypeToElementInfo (see page 5154) array.

CMinArrayVarType is equal to the varEmpty (see page 3865) constant.

See Also

CVarTypeToElementInfo (see page 5154)

52.5.9 VarUtils.CVarTypeToElementInfo Constant

This constant is internally used by the SafeArrayCreate (see page 5118) function.

Pascal

```
CVarTypeToElementInfo: array [CMinArrayVarType..CMaxArrayVarType] of TVarTypeToElementInfo = (
(ValidBase: False; ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: False;
ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: True; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 4; Flags: ARR_OLESTR), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_DISPATCH), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 16; Flags: ARR_VARIANT), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_UNKNOWN), (ValidBase: False; ValidElement: False; Size: 14; Flags: ARR_NONE),
(ValidBase: False; ValidElement: False; Size: 0; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 1; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 1;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: False; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: False; ValidElement: False; Size: 8; Flags: ARR_NONE));
```

C++

```
array [CMinArrayVarType..CMaxArrayVarType] of TVarTypeToElementInfo CVarTypeToElementInfo = (
(ValidBase: False; ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: False;
ValidElement: True; Size: 0; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: True; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 8; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 4; Flags: ARR_OLESTR), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_DISPATCH), (ValidBase: True; ValidElement: True; Size: 4; Flags: ARR_NONE),
(ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 16; Flags: ARR_VARIANT), (ValidBase: True; ValidElement: True; Size:
4; Flags: ARR_UNKNOWN), (ValidBase: False; ValidElement: False; Size: 14; Flags: ARR_NONE),
(ValidBase: False; ValidElement: False; Size: 0; Flags: ARR_NONE), (ValidBase: True;
ValidElement: True; Size: 1; Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 1;
Flags: ARR_NONE), (ValidBase: True; ValidElement: True; Size: 2; Flags: ARR_NONE), (ValidBase:
True; ValidElement: True; Size: 4; Flags: ARR_NONE), (ValidBase: False; ValidElement: True;
Size: 8; Flags: ARR_NONE), (ValidBase: False; ValidElement: False; Size: 8; Flags: ARR_NONE));
```

File

VarUtils

Description

CVarTypeToElementInfo is an constant array used internally in the implementation of the SafeArrayCreate (see page 5118) function. We recommend you not to use it directly when writing your application.

See Also

SafeArrayCreate (see page 5118)

CMinArrayVarType (see page 5153)

CMaxArrayVarType (see page 5153)

52.5.10 VarUtils.GenericOperations Constant

Specifies whether the implementation for the variant operation routines is cross-platform.

Pascal

```
GenericOperations = GenericVariants;
```

C++

```
GenericOperations = GenericVariants;
```

File

VarUtils

Description

GenericOperations is a constant that specifies whether the implementation for the variant operation routines is cross-platform. The operation routines refer to the binary and unary operations on variants, as well as the conversion routines.

GenericOperations has the same value with GenericVariants (see page 5156).

See Also

GenericVarUtils (see page 5157)

GenericVariants (see page 5156)

GenericSafeArrays (see page 5155)

GenericSafeArrayUsesLibC (see page 5156)

52.5.11 VarUtils.GenericSafeArrays Constant

Specifies whether the implementation for the safe array routines is cross-platform.

Pascal

```
GenericSafeArrays = GenericVarUtils;
```

C++

```
GenericSafeArrays = GenericVarUtils;
```

File

VarUtils

Description

GenericSafeArrays is a constant that specifies whether the implementation for the safe array routines is cross-platform.

GenericSafeArrays has the same value with GenericVarUtils (see page 5157).

See Also

GenericVarUtils (see page 5157)

GenericVariants (see page 5156)

GenericOperations (see page 5155)

GenericSafeArrayUsesLibC (see page 5156)

52.5.12 VarUtils.GenericSafeArrayUsesLibC Constant

Specifies whether the implementation for the safe array routines uses the `Libc` unit.

Pascal

```
GenericSafeArrayUsesLibC = FALSE;
```

C++

```
GenericSafeArrayUsesLibC = FALSE;
```

File

VarUtils

Description

GenericSafeArrayUsesLibC is a constant that specifies whether the implementation for the safe array routines uses the `Libc` unit.

See Also

GenericVarUtils (see page 5157)

GenericVariants (see page 5156)

GenericSafeArrays (see page 5155)

GenericOperations (see page 5155)

52.5.13 VarUtils.GenericVariants Constant

Specifies whether the implementation for the variant support routines is cross-platform.

Pascal

```
GenericVariants = GenericVarUtils;
```

C++

```
GenericVariants = GenericVarUtils;
```

File

VarUtils

Description

GenericVariants is a constant that specifies whether the implementation for the variant support routines is cross-platform. Support routines refer to initializing, copying, changing the type of a variant, and other similar routines.

GenericVariants has the same value with GenericVarUtils (see page 5157).

See Also

GenericVarUtils (see page 5157)

GenericSafeArrays (see page 5155)

GenericOperations (see page 5155)

GenericSafeArrayUsesLibC (see page 5156)

52.5.14 VarUtils.GenericVarUtils Constant

Specifies whether the implementation for the variant-manipulating routines is cross-platform.

Pascal

```
GenericVarUtils = False;
```

C++

```
GenericVarUtils = False;
```

File

VarUtils

Description

GenericVarUtils is a constant that specifies whether the implementation for the variant-manipulating routines is cross-platform.

See Also

GenericVariants (see page 5156)

GenericSafeArrays (see page 5155)

GenericOperations (see page 5155)

GenericSafeArrayUsesLibC (see page 5156)

52.5.15 VarUtils.VAR_ARRAYISLOCKED Constant

Specifies that the given array is locked.

Pascal

```
VAR_ARRAYISLOCKED = HRESULT($8002000D);
```

C++

```
VAR_ARRAYISLOCKED = HRESULT($8002000D);
```

File

VarUtils

Description

VAR_ARRAYISLOCKED is a possible return value of the variant-manipulating functions, specifying that the given variant array is locked.

52.5.16 VarUtils.VAR_BADINDEX Constant

Indicates that the given index is out of range.

Pascal

```
VAR_BADINDEX = HRESULT($8002000B);
```

C++

```
VAR_BADINDEX = HRESULT($8002000B);
```

File

VarUtils

Description

VAR_BADINDEX is a possible return value of the variant-manipulating functions, specifying that the given index is out of range.

52.5.17 VarUtils.VAR_BADVARTYPE Constant

Specifies that the variant type of the input parameter is invalid.

Pascal

```
VAR_BADVARTYPE = HRESULT($80020008);
```

C++

```
VAR_BADVARTYPE = HRESULT($80020008);
```

File

VarUtils

Description

VAR_BADVARTYPE is used to specify that the variant type of a routine's input parameter is invalid.

52.5.18 VarUtils.VAR_CMP_EQ Constant

VAR_CMP_EQ specifies that the variants are equal, after a call to the function referenced by the VarCmp (see page 5138) variable.

Pascal

```
VAR_CMP_EQ = 1;
```

C++

```
VAR_CMP_EQ = 1;
```

File

VarUtils

Description

VAR_CMP_EQ is one of the possible return values of the function referenced by the VarCmp (see page 5138) variable.

VAR_CMP_EQ specifies that the two variants are equal, after making the comparison.

See Also

VarCmp (see page 5138)

VAR_CMP_LT (see page 5159)

VAR_CMP_GT (see page 5159)

VAR_CMP_NULL (see page 5160)

52.5.19 VarUtils.VAR_CMP_GT Constant

VAR_CMP_GT specifies that the first variant is greater than the second, after a call to the function referenced by the VarCmp (see page 5138) variable.

Pascal

```
VAR_CMP_GT = 2;
```

C++

```
VAR_CMP_GT = 2;
```

File

VarUtils

Description

VAR_CMP_GT is one of the possible return values of the function referenced by the VarCmp (see page 5138) variable.

VAR_CMP_GT specifies that the first variant is greater than the second, after making the comparison.

See Also

VarCmp (see page 5138)

VAR_CMP_EQ (see page 5158)

VAR_CMP_LT (see page 5159)

VAR_CMP_NULL (see page 5160)

52.5.20 VarUtils.VAR_CMP_LT Constant

VAR_CMP_LT specifies that the first variant is lower than the second, after a call to the function referenced by the VarCmp (see page 5138) variable.

Pascal

```
VAR_CMP_LT = 0;
```

C++

```
VAR_CMP_LT = 0;
```

File

VarUtils

Description

VAR_CMP_LT is one of the possible return values of the function referenced by the VarCmp (see page 5138) variable.

VAR_CMP_LT specifies that the first variant is lower than the second, after making the comparison.

See Also

VarCmp (see page 5138)

VAR_CMP_EQ (see page 5158)

VAR_CMP_GT (see page 5159)

VAR_CMP_NULL (see page 5160)

52.5.21 VarUtils.VAR_CMP_NULL Constant

VAR_CMP_NULL specifies that either one of the variants is `nil`, after a call to the function referenced by the VarCmp (see page 5138) variable.

Pascal

```
VAR_CMP_NULL = 3;
```

C++

```
VAR_CMP_NULL = 3;
```

File

VarUtils

Description

VAR_CMP_NULL is one of the possible return values of the function referenced by the VarCmp (see page 5138) variable.

VAR_CMP_NULL specifies that either one of the variants is `nil`, after making the comparison.

See Also

VarCmp (see page 5138)

VAR_CMP_LT (see page 5159)

VAR_CMP_EQ (see page 5158)

VAR_CMP_GT (see page 5159)

52.5.22 VarUtils.VAR_EXCEPTION Constant

Specifies that the application raised an exception.

Pascal

```
VAR_EXCEPTION = HRESULT($80020009);
```

C++

```
VAR_EXCEPTION = HRESULT($80020009);
```

File

VarUtils

Description

VAR_EXCEPTION is a possible return value of the variant-manipulating functions, specifying that the application raised an exception during the execution of the function.

52.5.23 VarUtils.VAR_INVALIDARG Constant

Specifies that either one of the arguments is invalid.

Pascal

```
VAR_INVALIDARG = HRESULT($80070057);
```

C++

```
VAR_INVALIDARG = HRESULT($80070057);
```

File

VarUtils

Description

VAR_INVALIDARG is a possible return value of the variant-manipulating functions, specifying that either one of the given arguments is invalid.

52.5.24 VarUtils.VAR_LOCALE_USER_DEFAULT Constant

Specifies the default locale for the user or process.

Pascal

```
VAR_LOCALE_USER_DEFAULT = $400;
```

C++

```
VAR_LOCALE_USER_DEFAULT = 0x400;
```

File

VarUtils

Description

VAR_LOCALE_USER_DEFAULT gives the default locale for the user or process.

52.5.25 VarUtils.VAR_NOTIMPL Constant

Specifies that the operation is not implemented.

Pascal

```
VAR_NOTIMPL = HRESULT($80004001);
```

C++

```
VAR_NOTIMPL = HRESULT($80004001);
```

File

VarUtils

Description

VAR_NOTIMPL is a possible return value of the variant-manipulating functions, specifying that the operation is not implemented.

52.5.26 VarUtils.VAR_OK Constant

Specifies that a variant operation was successful.

Pascal

```
VAR_OK = HRESULT($00000000);
```

C++

```
VAR_OK = HRESULT($00000000);
```

File

VarUtils

Description

VAR_OK is a possible return value of the variant-manipulating functions, specifying that the operation was successfully completed.

52.5.27 VarUtils.VAR_OUTOFMEMORY Constant

Specifies that there is not enough memory to complete the operation.

Pascal

```
VAR_OUTOFMEMORY = HRESULT($8007000E);
```

C++

```
VAR_OUTOFMEMORY = HRESULT($8007000E);
```

File

VarUtils

Description

VAR_OUTOFMEMORY is a possible return value of the variant-manipulating functions, specifying that there is not enough memory to complete the operation.

52.5.28 VarUtils.VAR_OVERFLOW Constant

Specifies that a variant operation caused an overflow.

Pascal

```
VAR_OVERFLOW = HRESULT($8002000A);
```

C++

```
VAR_OVERFLOW = HRESULT($8002000A);
```

File

VarUtils

Description

VAR_OVERFLOW is a possible return value of the variant-manipulating functions, specifying that the data stored in the output parameter does not fit in the destination type.

52.5.29 VarUtils.VAR_PARAMNOTFOUND Constant

Indicates that there is no correspondence between the parameter DISPIDs and a parameter of the method.

Pascal

```
VAR_PARAMNOTFOUND = HRESULT($80020004);
```

C++

```
VAR_PARAMNOTFOUND = HRESULT($80020004);
```

File

VarUtils

Description

VAR_PARAMNOTFOUND indicates that there is no correspondence between the parameter DISPIDs and a parameter of the method.

52.5.30 VarUtils.VAR_TYPEMISMATCH Constant

Specifies that the argument cannot be cast to the specified type.

Pascal

```
VAR_TYPEMISMATCH = HRESULT($80020005);
```

C++

```
VAR_TYPEMISMATCH = HRESULT($80020005);
```

File

VarUtils

Description

VAR_TYPEMISMATCH is a possible return value of the variant-manipulating functions, specifying that the given argument cannot be cast to the specified type.

52.5.31 VarUtils.VAR_UNEXPECTED Constant

Indicates that an unexpected error occurred.

Pascal

```
VAR_UNEXPECTED = HRESULT($8000FFFF);
```

C++

```
VAR_UNEXPECTED = HRESULT($8000FFFF);
```

File

VarUtils

Description

VAR_ARRAYISLOCKED (see page 5157) is a possible return value of the variant-manipulating functions, specifying that an unexpected error occurred during the execution of the operation.

53 VCLCom Namespace

This is namespace VCLCom.

Topics

Name	Description
Classes (see page 5166)	The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentFactory (see page 5166)	TComponentFactory is the class factory for creating instances of components that support interfaces.

53.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TComponentFactory (see page 5166)	TComponentFactory is the class factory for creating instances of components that support interfaces.

53.1.1 TComponentFactory Class

TComponentFactory is the class factory for creating instances of components that support interfaces.

Class Hierarchy

Pascal

```
TComponentFactory = class(TAutoObjectFactory, IClassFactory);
```

C++

```
class TComponentFactory : public TAutoObjectFactory, public IClassFactory;
```

File

VCLCom

Description

TComponentFactory creates instances of component classes that implement interfaces. TComponentFactory supports the IClassFactory interface and inherits support for the IClassFactory2 interface.

Note: These components are actually created as contained objects of an automation object (aggregate).

53.1.1.1 TComponentFactory Members

The following tables list the members exposed by TComponentFactory.

Protected Methods

	Name	Description
	CreateInstance (see page 5168)	This is CreateInstance, a member of class TComponentFactory.

Public Methods

	Name	Description
	Create (see page 5167)	Instantiates an instance of a TComponentFactory (see page 5166) object.
	CreateComObject (see page 5167)	Instantiates a TComObject object.
	UpdateRegistry (see page 5168)	Registers the typed COM object class.

53.1.1.2 TComponentFactory.Create Constructor

Instantiates an instance of a TComponentFactory (see page 5166) object.

Pascal

```
constructor Create(ComServer: TComServerObject; ComponentClass: TComponentClass; const
ClassID: TGUID; Instancing: TClassInstancing; ThreadingModel: TThreadingModel = tmSingle);
```

C++

```
__fastcall TComponentFactory(TComServerObject ComServer, TComponentClass ComponentClass, const
TGUID ClassID, TClassInstancing Instancing, TThreadingModel ThreadingModel = tmSingle);
```

Description

Create allocates memory for a TComponentFactory (see page 5166) object, and calls the inherited Create, passing its parameters in the same order. The ComponentClass parameter is passed cast as a TAutoClass and passed to the AutoClass parameter.

53.1.1.3 TComponentFactory Methods

The methods of the TComponentFactory class are listed here.

Protected Methods

	Name	Description
	CreateInstance (see page 5168)	This is CreateInstance, a member of class TComponentFactory.

Public Methods

	Name	Description
	CreateComObject (see page 5167)	Instantiates a TComObject object.
	UpdateRegistry (see page 5168)	Registers the typed COM object class.

53.1.1.3.1 TComponentFactory.CreateComObject Method

Instantiates a TComObject object.

Pascal

```
function CreateComObject(const Controller: IUnknown): TComObject; override;
```

C++

```
virtual __fastcall TComObject CreateComObject(const IUnknown Controller);
```

Description

CreateComObject is the method that instantiates the class type for which TComObjectFactory is the class factory. CreateComObject calls the CreateFromFactory constructor of the TComObject class to create a COM object. CreateComObject passes itself as the Factory parameter to CreateFromFactory.

CreateComClass creates a COM object based upon the value of the ComClass property. ComClass is the class identifier (CLSID) used by other applications to identify the COM class for which they request an object instance.

See Also

ComClass

CreateFromFactory

TComObject

53.1.1.3.2 TComponentFactory.CreateInstance Method

Pascal

```
function CreateInstance(const UnkOuter: IUnknown; const IID: TGUID; out Obj): HResult; stdcall;
```

C++

```
__fastcall __stdcall HRESULT CreateInstance(const IUnknown UnkOuter, const TGUID IID,  Obj);
```

Description

This is CreateInstance, a member of class TComponentFactory.

53.1.1.3.3 TComponentFactory.UpdateRegistry Method

Registers the typed COM object class.

Pascal

```
procedure UpdateRegistry(Register: Boolean); override;
```

C++

```
virtual __fastcall UpdateRegistry(Boolean Register);
```

Description

UpdateRegistry is called for EXEs when the EXE is run, and for DLLs or EXEs when the routines are called to specifically register or unregister the server.

See Also

DllRegisterServer

DllUnregisterServer

RegisterClassObject

54 WebAdapt Namespace

This is namespace WebAdapt.

Topics

Name	Description
Classes (see page 5175)	The following table lists classes in this documentation.
Interfaces (see page 5874)	The following table lists interfaces in this documentation.
Functions (see page 5897)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 5901)	The following table lists structs, records, enums in this documentation.
Types (see page 5903)	The following table lists types in this documentation.
Variables (see page 5930)	The following table lists variables in this documentation.
Constants (see page 5931)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAdapterException (see page 5177)	EAdapterException is raised by WebSnap adapter objects.
	EAdapterFieldException (see page 5178)	EAdapterFieldException is an exception class for errors by adapters using fields.
	TAbstractFieldValuesAdapter (see page 5180)	TAbstractFieldValuesAdapter is the base class for adapters that have a single default field.
	TAdapter (see page 5190)	TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.
	TAdapterAction (see page 5201)	TAdapterAction is a generic adapter action that relies on event handlers.
	TAdapterActions (see page 5209)	TAdapterActions manages the list of actions that belong to an adapter.
	TAdapterApplicationField (see page 5214)	TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 5439) component.
	TAdapterApplicationTitleField (see page 5218)	TAdapterApplicationTitleField is an adapter field used to access the application title.
	TAdapterBooleanField (see page 5223)	TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.
	TAdapterDefaultActions (see page 5231)	TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.
	TAdapterDefaultFields (see page 5237)	TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.
	TAdapterEndUserDisplayNameField (see page 5244)	TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.
	TAdapterEndUserField (see page 5248)	TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.
	TAdapterEndUserLoggedInField (see page 5252)	TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.

	TAdapterError (see page 5256)	TAdapterError is the implementation of IAdapterError (see page 992).
	TAdapterErrorsList (see page 5259)	TAdapterErrorsList keeps track of errors that an adapter encounters.
	TAdapterField (see page 5267)	TAdapterField is an event handler based adapter field that provides a single value.
	TAdapterFields (see page 5276)	TAdapterFields manages the list of fields that belong to an adapter.
	TAdapterFileField (see page 5281)	TAdapterFileField is an event handler based adapter field that can accept uploaded files.
	TAdapterGotoPageAction (see page 5291)	TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5747).
	TAdapterHiddenField (see page 5299)	This is class WebAdapt.TAdapterHiddenField.
	TAdapterHiddenFieldsList (see page 5301)	TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.
	TAdapterImageField (see page 5308)	TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.
	TAdapterLoginField (see page 5320)	TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 5824) component.
	TAdapterMemoField (see page 5323)	TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.
	TAdapterMultiValueField (see page 5332)	TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.
	TAdapterNamedDisplayField (see page 5343)	The AdapterNamedDisplayField is a base class for adapter fields.
	TAdapterNamedField (see page 5351)	TAdapterNamedField is a based class for adapter field components.
	TAdapterNextPageAction (see page 5370)	TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 5747).
	TAdapterNextPageField (see page 5379)	The TAdapterNextPageField navigates to the next page of data.
	TAdapterObjectErrorsList (see page 5387)	This is class WebAdapt.TAdapterObjectErrorsList.
	TAdapterPasswordField (see page 5391)	TAdapterPasswordField is an adapter field used to access the password entered by the end-user.
	TAdapterPrevPageAction (see page 5396)	TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 5747).
	TAdapterSubComponent (see page 5406)	TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.
	TAdapterUpdateField (see page 5419)	TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.
	TAdapterUserNameField (see page 5433)	TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.
	TApplicationAdapter (see page 5439)	TApplicationAdapter is a container for field and action components that are available through the Application script variable.
	TBaseAdapterAction (see page 5447)	TBaseAdapterAction is a based class for adapter action components.
	TBaseAdapterErrorsList (see page 5479)	TBaseAdapterErrorsList is the base class for adapter error object managers.
	TBaseAdapterMultiValueField (see page 5488)	TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.
	TBaseEndUserAdapter (see page 5497)	TBaseEndUserAdapter contains user information.
	TBaseValuesListAdapter (see page 5517)	TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.
	TBaseValuesListField (see page 5527)	TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.
	TCustomAdapter (see page 5531)	TCustomAdapter is a base class for adapter components.
	TCustomAdapterAction (see page 5579)	TCustomAdapterAction is a base class for adapter actions.
	TCustomAdapterBooleanField (see page 5589)	TCustomAdapterBooleanField is a base class for adapter fields that provide a single Boolean value.
	TCustomAdapterField (see page 5597)	TCustomAdapterField is a base class for adapter fields that provide a single value.
	TCustomAdapterFileField (see page 5604)	TCustomAdapterFileField is a base class for adapter fields that can have associated files.
	TCustomAdapterGotoPageAction (see page 5614)	TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5747).
	TCustomAdapterImageField (see page 5623)	TCustomAdapterImageField is a base class for adapter fields that have an associated image file.
	TCustomAdapterMemoField (see page 5638)	TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.

	TCustomAdapterMultiValueField (see page 5646)	TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.
	TCustomAdapterPageAction (see page 5656)	TCustomPagedAdapterAction (see page 5722) is a base class for adapter actions that execute the next or previous page commands.
	TCustomApplicationAdapter (see page 5663)	TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.
	TCustomApplicationAdapterAction (see page 5671)	TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 5663).
	TCustomEndUserAdapter (see page 5676)	TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomEndUserAdapterAction (see page 5687)	TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 5676).
	TCustomEndUserSessionAdapter (see page 5692)	TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomLoginFormAdapter (see page 5701)	TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.
	TCustomLoginFormAdapterAction (see page 5717)	TCustomEndUserAdapterAction (see page 5687) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 5701).
	TCustomPagedAdapter (see page 5721)	TCustomPagedAdapter is a base class for adapters that support paging.
	TCustomPagedAdapterAction (see page 5722)	TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 5747).
	TCustomStringsValuesList (see page 5728)	TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.
	TDefaultFieldsAdapter (see page 5738)	TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.
	TDefaultFieldsPagedAdapter (see page 5747)	TDefaultFieldsPagedAdapter is a base class for adapters that support paging.
	TEndUserAdapter (see page 5764)	TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TEndUserAdapterLoginFormAction (see page 5776)	TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.
	TEndUserAdapterLogoutAction (see page 5784)	TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.
	TEndUserSessionAdapter (see page 5792)	TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TFieldValuesAdapter (see page 5803)	TFieldValuesAdapter is an adapter that includes a single field value.
	TFieldValuesValueField (see page 5813)	TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 5803).
	TImplementedAdapterAction (see page 5818)	TImplementedAdapterAction is a base class for adapter actions.
	TLogicalPageList (see page 5819)	This is class WebAdapt.TLogicalPageList.
	TLoginFormAdapter (see page 5824)	TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.
	TLoginFormAdapterLoginAction (see page 5835)	TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 5824).
	TPagedAdapter (see page 5844)	TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.
	TStringsValuesList (see page 5857)	TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.
	TUpdateFileList (see page 5864)	TUpdateFileList is a container for uploaded files.
	TValuesListNameField (see page 5866)	TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.
	TValuesListValueField (see page 5869)	TValuesListValueField is an adapter field used to access the Value (see page 5369) of the current list item of a values list adapter.

Constants

Name	Description
sActionRequestLong (see page 5931)	This is constant WebAdapt.sActionRequestLong.
sCurrentPage (see page 5931)	This is constant WebAdapt.sCurrentPage.
sGotoPageAction (see page 5932)	This is constant WebAdapt.sGotoPageAction.
sImageRequest (see page 5932)	This is constant WebAdapt.sImageRequest.
sLoginPassword (see page 5932)	This is constant WebAdapt.sLoginPassword.
sLoginUserName (see page 5933)	This is constant WebAdapt.sLoginUserName.

sNextPageAction (see page 5933)	This is constant WebAdapt.sNextPageAction.
sPrevPageAction (see page 5933)	This is constant WebAdapt.sPrevPageAction.

Functions

	Name	Description
	AddDefaultActions (see page 5897)	This is function WebAdapt.AddDefaultActions.
	AddDefaultFields (see page 5897)	This is function WebAdapt.AddDefaultFields.
	FindAdapterAction (see page 5898)	This is function WebAdapt.FindAdapterAction.
	FindAdapterField (see page 5898)	This is function WebAdapt.FindAdapterField.
	FindAdapterInParent (see page 5898)	This is function WebAdapt.FindAdapterInParent.
	MakeAdapterRequestIdentifier (see page 5899)	This is function WebAdapt.MakeAdapterRequestIdentifier.
	MakeValidAdapterFieldName (see page 5899)	This is function WebAdapt.MakeValidAdapterFieldName.
	RaiseFileUploadNotSupported (see page 5899)	This is function WebAdapt.RaiseFileUploadNotSupported.
	RaiseMultipleFilesException (see page 5900)	This is function WebAdapt.RaiseMultipleFilesException.
	RaiseMultipleValuesException (see page 5900)	This is function WebAdapt.RaiseMultipleValuesException.

Interfaces

	Name	Description
	IAdapterEditor (see page 5874)	This is class WebAdapt.IAdapterEditor.
	IAdapterNotifyAdapterChange (see page 5875)	This is class WebAdapt.IAdapterNotifyAdapterChange.
	ICheckOrigValue (see page 5876)	This is class WebAdapt.ICheckOrigValue.
	ICheckValueChange (see page 5877)	This is class WebAdapt.ICheckValueChange.
	IClearAdapterValues (see page 5878)	This is class WebAdapt.IClearAdapterValues.
	ICreateActionRequestContext (see page 5879)	This is class WebAdapt.ICreateActionRequestContext.
	IEchoActionFieldValue (see page 5880)	This is class WebAdapt.IEchoActionFieldValue.
	IEchoAdapterFieldValues (see page 5882)	This is class WebAdapt.IEchoAdapterFieldValues.
	IExecuteValue (see page 5883)	IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.
	IFullyQualifyInputNames (see page 5885)	This is class WebAdapt.IFullyQualifyInputNames.
	IGetAdapterErrorsList (see page 5886)	This is class WebAdapt.IGetAdapterErrorsList.
	IGetAdapterModeNames (see page 5887)	This is class WebAdapt.IGetAdapterModeNames.
	IGetRecordKeys (see page 5888)	This is class WebAdapt.IGetRecordKeys.
	IIdentifyAdapter (see page 5889)	This is class WebAdapt.IIdentifyAdapter.
	ISetSubComponents (see page 5889)	This is class WebAdapt.ISetSubComponents.
	ISetWebFieldValue (see page 5890)	This is class WebAdapt.ISetWebFieldValue.
	IUpdateValue (see page 5891)	This is class WebAdapt.IUpdateValue.
	IValidateValue (see page 5892)	This is class WebAdapt.IValidateValue.
	IWebActionsSubcomponent (see page 5893)	This is class WebAdapt.IWebActionsSubcomponent.
	IWebFieldsSubcomponent (see page 5893)	This is class WebAdapt.IWebFieldsSubcomponent.
	IWebGetFieldPrevValue (see page 5894)	This is class WebAdapt.IWebGetFieldPrevValue.

Structs, Records, Enums

	Name	Description
	TAdapterHiddenFieldOption (see page 5901)	TAdapterHiddenFieldOptions (see page 5919) identifies the types of hidden fields an adapter adds to generated HTML forms.
	TIteratorMethod (see page 5902)	TIteratorMethod identifies a record of data to which an adapter should navigate.

Types

Name	Description
IActionFieldValue (see page 5904)	This is type WebAdapt.IActionFieldValue.
IActionRequest (see page 5905)	This is type WebAdapt.IActionRequest.
TAbstractWebRequestFile (see page 5905)	This is type WebAdapt.TAbstractWebRequestFile.
TActionExecuteEvent (see page 5905)	TActionExecuteEvent is the type for event handlers on adapter action objects that are called when the action executes.
TActionExecuteHandledEvent (see page 5906)	TActionExecuteHandledEvent is the type for event handlers on adapter action objects that can handle the execution of the action.

TActionGetEnabledEvent (see page 5906)	TActionGetEnabledEvent is the type for event handlers on adapter action objects that are called when the action executes.
TActionGetParamsEvent (see page 5907)	TActionGetParamsEvent is the type for event handlers that supply the parameters for an adapter action.
TActionRequestFieldsEvent (see page 5907)	TActionRequestFieldsEvent is the type for event handlers on adapter action objects that are called after the adapter performs some operation on its fields in response to an HTTP request.
TActionRequestFieldsHandledEvent (see page 5908)	TActionRequestFieldsHandledEvent is the type for event handlers on adapter action objects that are called when the adapter performs some operation on its fields in response to an HTTP request.
TAdapterActionExecuteEvent (see page 5909)	TAdapterActionExecuteEvent is the type for event handlers on adapter objects that are called after one of its actions executes.
TAdapterActionExecuteHandledEvent (see page 5909)	TAdapterActionExecuteHandledEvent is the type for event handlers on adapter objects that are called when one of its actions executes.
TAdapterActionGetParamsEvent (see page 5910)	TAdapterActionGetParamsEvent is the type for event handlers that supply the parameters needed to execute an adapter action.
TAdapterActionsClass (see page 5911)	TAdapterActionsClass defines the metaclass for TAdapterActions (see page 5209).
TAdapterFieldBooleanValueEvent (see page 5911)	TAdapterFieldBooleanValueEvent is the type for event handlers that respond when a boolean value is assigned to an adapter field.
TAdapterFieldFileEvent (see page 5912)	TAdapterFieldFileEvent is the type for event handlers that perform some action on an uploaded file.
TAdapterFieldFileListEvent (see page 5912)	TAdapterFieldFileListEvent is the type for event handlers that perform some action on a set of uploaded files.
TAdapterFieldGetBooleanEvent (see page 5913)	TAdapterFieldGetBooleanEvent is the type for event handlers that provide a boolean value associated with an adapter field.
TAdapterFieldGetMultiValueEvent (see page 5913)	TAdapterFieldGetMultiValueEvent is the type for event handlers that supply a set of values for an adapter field.
TAdapterFieldGetMultiValueStringEvent (see page 5914)	TAdapterFieldGetMultiValueStringEvent is the type for event handlers that supply a set of string values for an adapter field.
TAdapterFieldGetStringEvent (see page 5914)	TAdapterFieldGetStringEvent is the type for event handlers that provide a string associated with an adapter field.
TAdapterFieldGetValueCountEvent (see page 5915)	TAdapterFieldGetValueCountEvent is the type for event handlers that indicate the number of values associated with an adapter field.
TAdapterFieldGetValueEvent (see page 5915)	TAdapterFieldGetValueEvent is the type for event handlers that provide the value associated with an adapter field.
TAdapterFieldHasValueEvent (see page 5916)	TAdapterFieldHasValueEvent is the type for event handlers that indicate whether a value is assigned to an adapter field.
TAdapterFieldsClass (see page 5916)	This is type WebAdapt.TAdapterFieldsClass.
TAdapterFieldStringEvent (see page 5917)	TAdapterFieldStringEvent is the type for event handlers that respond when a string value is assigned to an adapter field.
TAdapterFieldStringsEvent (see page 5917)	TAdapterFieldStringsEvent is the type for event handlers that respond when a set of string values is assigned to an adapter field.
TAdapterFieldValidateValueEvent (see page 5918)	TAdapterFieldValidateValueEvent is the type for event handlers that validate a value for an adapter field.
TAdapterFieldValueEvent (see page 5918)	TAdapterFieldValueEvent is the type for event handlers that respond when an adapter field's value changes.
TAdapterHiddenFieldOptions (see page 5919)	This is type WebAdapt.TAdapterHiddenFieldOptions.
TEndUserGetNameEvent (see page 5919)	TEndUserGetNameEvent is the type for event handlers that supply the name of the current end user.
TEndUserGetUserIDEvent (see page 5920)	TEndUserGetUserIDEvent is the type for event handlers that supply the user ID of the current end user.
TEndUserHandledEvent (see page 5920)	TEndUserHandledEvent is the type for event handlers that can optionally handle a task for an end user adapter.
TEndUserIsLoggedInEvent (see page 5921)	TEndUserIsLoggedInEvent is the type for event handlers that indicate whether the current end user has logged in.
TEndUserRightsEvent (see page 5921)	TEndUserRightsEvent is the type for event handlers that check whether a user has rights to perform a specific operation.
TEndUserUserIDEvent (see page 5922)	TEndUserUserIDEvent is the type for event handlers that log in a user given the user ID.
TGetEOFEvent (see page 5922)	TGetEOFEvent is the type for event handlers on adapters that indicate when to stop iterating over values.
TGetRecordCountEvent (see page 5923)	TGetRecordCountEvent is the type for event handlers that indicate how many values an adapter has.

TGetRecordEvent (see page 5923)	TGetRecordEvent is the type for event handlers on adapters that position the adapter on one of multiple values.
TGetRecordIndexEvent (see page 5924)	TGetRecordIndexEvent is the type for event handlers that indicate which of multiple values an adapter is positioned on.
THandledEvent (see page 5924)	This is type WebAdapt.THandledEvent.
TImageFieldGetHREFEvent (see page 5925)	TImageFieldGetHREFEvent is the type for event handlers that supply an URL that can be used to access an image field associated with an adapter field..
TImageFieldGetImageEvent (see page 5925)	TImageFieldGetImageEvent is the type for event handlers that supply a stream from which to read an image.
TImageFieldGetParamsEvent (see page 5926)	TImageFieldGetParamsEvent is the type for event handlers that supply a set of parameters to identify the value of an image field.
TIterateRecordsEvent (see page 5926)	TIterateRecordsEvent is the type for event handlers that navigate the records of an adapter.
TLoginUserIDEvent (see page 5927)	TLoginUserIDEvent is the type for event handlers that respond after a user logs in.
TObjectList (see page 5928)	This is type WebAdapt.TObjectList.
TValidateAdapterFieldEvent (see page 5928)	TValidateAdapterFieldEvent is the type for event handlers that validate the values supplied for adapter fields.
TWebFormGetPageNameEvent (see page 5928)	TWebFormGetPageNameEvent is the type for event handlers that supply labels for the controls that navigate to Web pages.

Variables

Name	Description
NotifyPossibleHTMLChange (see page 5930)	This is variable WebAdapt.NotifyPossibleHTMLChange.

54.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAdapterException (see page 5177)	EAdapterException is raised by WebSnap adapter objects.
	EAdapterFieldException (see page 5178)	EAdapterFieldException is an exception class for errors by adapters using fields.
	TAbstractFieldValuesAdapter (see page 5180)	TAbstractFieldValuesAdapter is the base class for adapters that have a single default field.
	TAdapter (see page 5190)	TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.
	TAdapterAction (see page 5201)	TAdapterAction is a generic adapter action that relies on event handlers.
	TAdapterActions (see page 5209)	TAdapterActions manages the list of actions that belong to an adapter.
	TAdapterApplicationField (see page 5214)	TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 5439) component.
	TAdapterApplicationTitleField (see page 5218)	TAdapterApplicationTitleField is an adapter field used to access the application title.
	TAdapterBooleanField (see page 5223)	TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.
	TAdapterDefaultActions (see page 5231)	TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.
	TAdapterDefaultFields (see page 5237)	TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.
	TAdapterEndUserDisplayNameField (see page 5244)	TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.
	TAdapterEndUserField (see page 5248)	TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.
	TAdapterEndUserLoggedInField (see page 5252)	TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.
	TAdapterError (see page 5256)	TAdapterError is the implementation of IAdapterError (see page 992).
	TAdapterErrorsList (see page 5259)	TAdapterErrorsList keeps track of errors that an adapter encounters.
	TAdapterField (see page 5267)	TAdapterField is an event handler based adapter field that provides a single value.
	TAdapterFields (see page 5276)	TAdapterFields manages the list of fields that belong to an adapter.
	TAdapterFileField (see page 5281)	TAdapterFileField is an event handler based adapter field that can accept uploaded files.
	TAdapterGotoPageAction (see page 5291)	TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5747).
	TAdapterHiddenField (see page 5299)	This is class WebAdapt.TAdapterHiddenField.
	TAdapterHiddenFieldsList (see page 5301)	TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.
	TAdapterImageField (see page 5308)	TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.
	TAdapterLoginField (see page 5320)	TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 5824) component.
	TAdapterMemoField (see page 5323)	TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.
	TAdapterMultiValueField (see page 5332)	TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.
	TAdapterNamedDisplayField (see page 5343)	The AdapterNamedDisplayField is a base class for adapter fields.
	TAdapterNamedField (see page 5351)	TAdapterNamedField is a based class for adapter field components.
	TAdapterNextPageAction (see page 5370)	TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 5747).
	TAdapterNextPageField (see page 5379)	The TAdapterNextPageField navigates to the next page of data.
	TAdapterObjectErrorsList (see page 5387)	This is class WebAdapt.TAdapterObjectErrorsList.

	TAdapterPasswordField (see page 5391)	TAdapterPasswordField is an adapter field used to access the password entered by the end-user.
	TAdapterPrevPageAction (see page 5396)	TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 5747).
	TAdapterSubComponent (see page 5406)	TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.
	TAdapterUpdateField (see page 5419)	TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.
	TAdapterUserNameField (see page 5433)	TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.
	TApplicationAdapter (see page 5439)	TApplicationAdapter is a container for field and action components that are available through the Application script variable.
	TBaseAdapterAction (see page 5447)	TBaseAdapterAction is a based class for adapter action components.
	TBaseAdapterErrorsList (see page 5479)	TBaseAdapterErrorsList is the base class for adapter error object managers.
	TBaseAdapterMultiValueField (see page 5488)	TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.
	TBaseEndUserAdapter (see page 5497)	TBaseEndUserAdapter contains user information.
	TBaseValuesListAdapter (see page 5517)	TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.
	TBaseValuesListField (see page 5527)	TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.
	TCustomAdapter (see page 5531)	TCustomAdapter is a base class for adapter components.
	TCustomAdapterAction (see page 5579)	TCustomAdapterAction is a base class for adapter actions.
	TCustomAdapterBooleanField (see page 5589)	TCustomAdapterBooleanField is a base class for adapter fields that provide a single Boolean value.
	TCustomAdapterField (see page 5597)	TCustomAdapterField is a base class for adapter fields that provide a single value.
	TCustomAdapterFileField (see page 5604)	TCustomAdapterFileField is a base class for adapter fields that can have associated files.
	TCustomAdapterGotoPageAction (see page 5614)	TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5747).
	TCustomAdapterImageField (see page 5623)	TCustomAdapterImageField is a base class for adapter fields that have an associated image file.
	TCustomAdapterMemoField (see page 5638)	TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.
	TCustomAdapterMultiValueField (see page 5646)	TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.
	TCustomAdapterPageAction (see page 5656)	TCustomPagedAdapterAction (see page 5722) is a base class for adapter actions that execute the next or previous page commands.
	TCustomApplicationAdapter (see page 5663)	TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.
	TCustomApplicationAdapterAction (see page 5671)	TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 5663).
	TCustomEndUserAdapter (see page 5676)	TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomEndUserAdapterAction (see page 5687)	TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 5676).
	TCustomEndUserSessionAdapter (see page 5692)	TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TCustomLoginFormAdapter (see page 5701)	TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.
	TCustomLoginFormAdapterAction (see page 5717)	TCustomEndUserAdapterAction (see page 5687) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 5701).
	TCustomPagedAdapter (see page 5721)	TCustomPagedAdapter is a base class for adapters that support paging.
	TCustomPagedAdapterAction (see page 5722)	TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 5747).
	TCustomStringsValuesList (see page 5728)	TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.
	TDefaultFieldsAdapter (see page 5738)	TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.
	TDefaultFieldsPagedAdapter (see page 5747)	TDefaultFieldsPagedAdapter is a base class for adapters that support paging.

	TEndUserAdapter (see page 5764)	TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TEndUserAdapterLoginFormAction (see page 5776)	TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.
	TEndUserAdapterLogoutAction (see page 5784)	TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.
	TEndUserSessionAdapter (see page 5792)	TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.
	TFieldValuesAdapter (see page 5803)	TFieldValuesAdapter is an adapter that includes a single field value.
	TFieldValuesValueField (see page 5813)	TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 5803).
	TImplementedAdapterAction (see page 5818)	TImplementedAdapterAction is a base class for adapter actions.
	TLogicalPageList (see page 5819)	This is class WebAdapt.TLogicalPageList.
	TLoginFormAdapter (see page 5824)	TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.
	TLoginFormAdapterLoginAction (see page 5835)	TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 5824).
	TPagedAdapter (see page 5844)	TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.
	TStringsValuesList (see page 5857)	TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.
	TUpdateFileList (see page 5864)	TUpdateFileList is a container for uploaded files.
	TValuesListNameField (see page 5866)	TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.
	TValuesListValueField (see page 5869)	TValuesListValueField is an adapter field used to access the Value (see page 5369) of the current list item of a values list adapter.

54.1.1 EAdapterException Class

EAdapterException is raised by WebSnap adapter objects.

Class Hierarchy

Pascal

```
EAdapterException = class(EWebBrokerException);
```

C++

```
class EAdapterException : public EWebBrokerException;
```

File

WebAdapt

Description

EAdapterException is raised when a WebSnap adapter object encounters an error. EAdapterException provides an ID (see page 5178) property to record application-defined errors.

54.1.1.1 EAdapterException Members

The following tables list the members exposed by EAdapterException.

Public Methods

	Name	Description
	CreateID (see page 5178)	Create exception object with error ID (see page 5178).

Public Properties

	Name	Description
	ID (see page 5178)	Application error ID.

54.1.1.2 EAdapterException.CreateID Constructor

Create exception object with error ID (see page 5178).

Pascal

```
constructor CreateID(const AMsg: string; AID: Integer; Reserved: Integer = 0);
```

C++

```
__fastcall EAdapterException(const UnicodeString AMsg, int AID, int Reserved = 0);
```

Description

CreateID creates an EAdapterException (see page 5177) object in the usual way, and also assigns AID to the ID (see page 5178) property.

Reserved is not implemented.

54.1.1.3 EAdapterException Properties

The properties of the EAdapterException class are listed here.

Public Properties

	Name	Description
	ID (see page 5178)	Application error ID.

54.1.1.3.1 EAdapterException.ID Property

Application error ID.

Pascal

```
property ID: Integer;
```

C++

```
__property int ID;
```

Description

The ID property returns an application-defined value relating to the error. To specify an ID value, create an EAdapterException (see page 5177) object using the CreateID (see page 5178) constructor (Delphi) or the constructor syntax that includes an ID as a parameter (C++). For exceptions raised by built-in WebSnap adapters, ID is always 0.

See Also

CreateID (see page 5178)

54.1.2 EAdapterFieldException Class

EAdapterFieldException is an exception class for errors by adapters using fields.

Class Hierarchy

Pascal

```
EAdapterFieldException = class(EAdapterException);
```

C++

```
class EAdapterFieldException : public EAdapterException;
```

File

WebAdapt

Description

EAdapterFieldException is raised when an adapter encounters an error using a field.

54.1.2.1 EAdapterFieldException Members

The following tables list the members exposed by EAdapterFieldException.

Public Methods

	Name	Description
	Create (see page 5179)	Create an EAdapterFieldException (see page 5178) instance.
	CreateID (see page 5178)	Create exception object with error ID (see page 5178).

Public Properties

	Name	Description
	FieldName (see page 5180)	FieldName contains the name of the field associated with the error.
	ID (see page 5178)	Application error ID.

54.1.2.2 EAdapterFieldException.Create Constructor

Create an EAdapterFieldException (see page 5178) instance.

Pascal

```
constructor Create(const AMsg: string; const AFieldName: string; AID: Integer = 0);
```

C++

```
__fastcall EAdapterFieldException(const UnicodeString AMsg, const UnicodeString AFieldName,
int AID = 0);
```

Description

Create instantiates an instance of EAdapterFieldException (see page 5178).

AMsg is the string containing the runtime error message to display in the exception dialog box. AMsg can be a hard-coded string, or can be a function call that returns a string.

AFieldName and AID are assigned to the FieldName (see page 5180) and ID (see page 5178) properties.

See Also

FieldName (see page 5180)

ID (see page 5178)

54.1.2.3 EAdapterFieldException Properties

The properties of the EAdapterFieldException class are listed here.

Public Properties

	Name	Description
	FieldName (see page 5180)	FieldName contains the name of the field associated with the error.

54.1.2.3.1 EAdapterFieldException.FieldName Property

FieldName contains the name of the field associated with the error.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

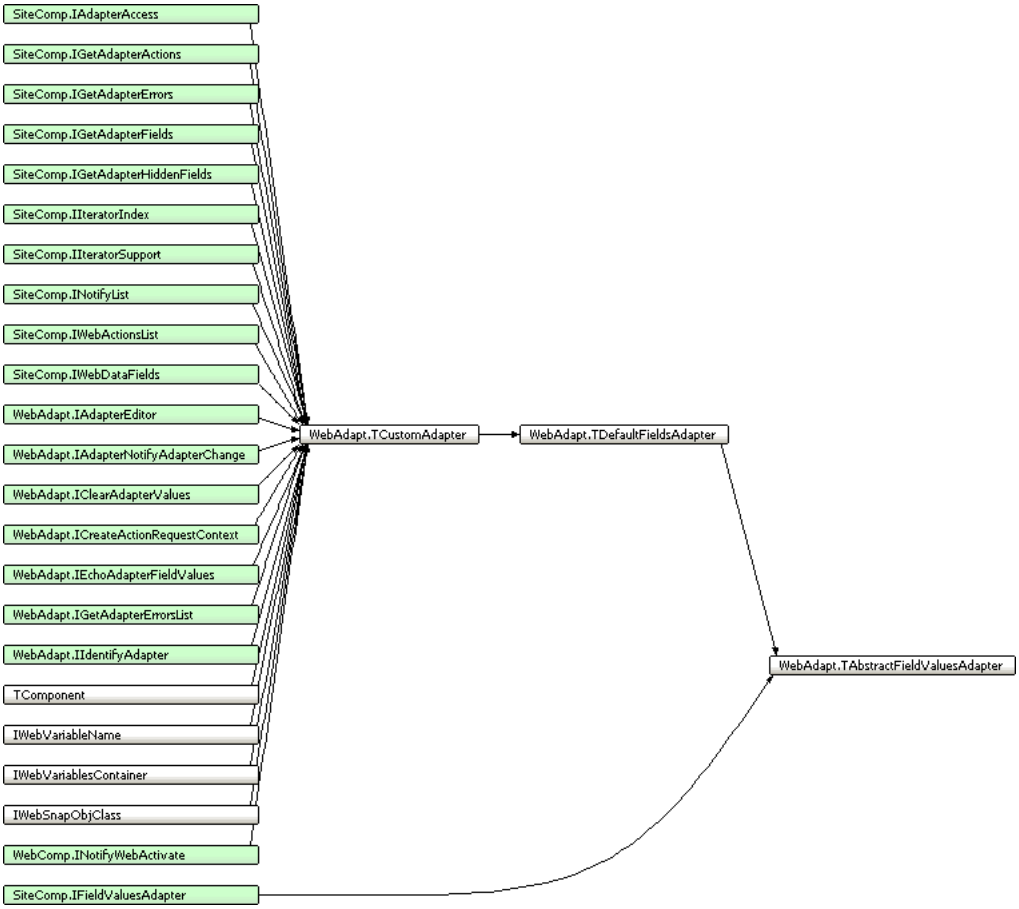
```
__property AnsiString FieldName = {read=FFieldName, write=FFieldName};
property FieldName: string;
```

FieldName contains the name of the field associated with the error.

54.1.3 TAbstractFieldValuesAdapter Class

TAbstractFieldValuesAdapter is the base class for adapters that have a single default field.

Class Hierarchy



Pascal

```
TAbstractFieldValuesAdapter = class(TDefaultFieldsAdapter, IFieldValuesAdapter);
```

C++

```
class TAbstractFieldValuesAdapter : public TDefaultFieldsAdapter, public IFieldValuesAdapter;
```

File

WebAdapt

Description

Derive from TAbstractFieldValuesAdapter to define adapter components that automatically include a single adapter field called ‘ValueField'.

Do not use TAbstractFieldValuesAdpter directly. It includes many abstract or, in C++ parlance, pure virtual methods, which must be implemented by a descendant class such as TFieldValuesAdapter (see page 5803).

See Also

TFieldValuesAdapter (see page 5803)

54.1.3.1 TAbstractFieldValuesAdapter Members

The following tables list the members exposed by TAbstractFieldValuesAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetField (see page 5186)	Returns the object that represents the single field of this adapter.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.

A	GetValueFieldClass (see page 5186)	Returns the class of the single adapter field this adapter includes.
	GetValueIndex (see page 5187)	Returns an index that identifies the current adapter field value.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5187)	Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5188)	Provides the underlying implementation for the EndIterator (see page 5542) method.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
A	ImplGetField (see page 5188)	Provides the underlying implementation for the GetField (see page 5186) method.
V	ImplGetFieldsList (see page 5188)	Provides the underlying implementation for the GetFieldsList (see page 5549) method.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
A	ImplGetValueIndex (see page 5189)	Provides the underlying implementation for the GetValueIndex (see page 5187) method.
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
V	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5189)	Provides the underlying implementation for the NextIteration (see page 5564) method.
V	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
V	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
V	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
V	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
V	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
V	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5190)	Provides the underlying implementation for the StartIterator (see page 5567) method.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.

	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
V	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.

	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.3.2 TAbstractFieldValuesAdapter Methods

The methods of the TAbstractFieldValuesAdapter class are listed here.

Protected Methods

	Name	Description
	GetField (see page 5186)	Returns the object that represents the single field of this adapter.
A	GetValueFieldClass (see page 5186)	Returns the class of the single adapter field this adapter includes.
	GetValueIndex (see page 5187)	Returns an index that identifies the current adapter field value.
V	ImplCanAddFieldClass (see page 5187)	Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.

	ImplEndIterator (see page 5188)	Provides the underlying implementation for the EndIterator (see page 5542) method.
	ImplGetField (see page 5188)	Provides the underlying implementation for the GetField (see page 5186) method.
	ImplGetFieldsList (see page 5188)	Provides the underlying implementation for the GetFieldsList (see page 5549) method.
	ImplGetValueIndex (see page 5189)	Provides the underlying implementation for the GetValueIndex (see page 5187) method.
	ImplNextIteration (see page 5189)	Provides the underlying implementation for the NextIteration (see page 5564) method.
	ImplStartIterator (see page 5190)	Provides the underlying implementation for the StartIterator (see page 5567) method.

54.1.3.2.1 TAbstractFieldValuesAdapter.GetField Method

Returns the object that represents the single field of this adapter.

Pascal

```
function GetField: TObject;
```

C++

```
__fastcall TObject * GetField();
```

Description

Call GetField to access the single field for this adapter. GetField returns an adapter field that is an instance of the class that the GetValueFieldClass (see page 5186) method returns.

GetField calls the abstract or, in C++ terminology, pure virtual ImplGetField (see page 5188) method to obtain the adapter field instance it returns. Descendant classes override ImplGetField (see page 5188) to provide the underlying implementation for GetField.

See Also

ImplGetField (see page 5188)

GetValueFieldClass (see page 5186)

54.1.3.2.2 TAbstractFieldValuesAdapter.GetValueFieldClass Method

Returns the class of the single adapter field this adapter includes.

Pascal

```
function GetValueFieldClass: TComponentClass; virtual; abstract;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass() = 0;
```

Description

GetValueFieldClass returns the class reference for the type of the single adapter field that the GetField (see page 5186) method returns. The return value must be a descendant of TComponent and is typically a descendant of TAdapterNamedField (see page 5351).

In TAbstractFieldValuesAdapter (see page 5180), GetValueFieldClass is abstract or, in C++ terminology, pure virtual. Descendant classes must override this method to indicate the type of field this adapter includes.

See Also

GetField (see page 5186)

TAdapterNamedField (see page 5351)

54.1.3.2.3 TAbstractFieldValuesAdapter.GetValueIndex Method

Returns an index that identifies the current adapter field value.

Pascal

```
function GetValueIndex: Integer;
```

C++

```
__fastcall int GetValueIndex();
```

Description

Although this adapter only includes a single adapter field, that field may have several values (for example, if it represents a column in a table). When the field has several values, GetValueIndex is the index of the current value. This index may be the same as the value that the GetIteratorIndex (see page 5550) method returns if the application is using the adapter's iteration support. However, TAbstractFieldValuesAdapter (see page 5180) introduces GetValueIndex as a separate method for descendants that support multiple values that change using some other mechanism.

GetValueIndex calls the abstract or, in C++ terminology, pure virtual ImplGetValueIndex (see page 5189) method to obtain the index of the current value. Descendant classes override ImplGetValueIndex (see page 5189) to provide the underlying implementation for GetValueIndex.

See Also

ImplGetValueIndex (see page 5189)

GetIteratorIndex (see page 5550)

54.1.3.2.4 TAbstractFieldValuesAdapter.ImplCanAddFieldClass Method

Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddFieldClass (see page 5540) method calls ImplCanAddFieldClass to determine whether this adapter can include a field that is an instance of a specified class. This allows the adapter editor to indicate the available fields at design time.

AParent is an instance of this adapter class to which the field is to be added.

AClass is the class of the object that represents the adapter field.

ImplCanAddFieldClass returns true if AClass is the same as (or a descendant of) the class that the GetValueFieldClass (see page 5186) method returns, indicating that this is the only child field this adapter supports. It returns false for any other class.

See Also

CanAddFieldClass (see page 5540)

GetValueFieldClass (see page 5186)

54.1.3.2.5 TAbstractFieldValuesAdapter.ImplEndIterator Method

Provides the underlying implementation for the EndIterator (see page 5542) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

The EndIterator (see page 5542) method calls ImplEndIterator to stop the process of iterating through the adapter's field's possible values.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way.

ImplEndIterator raises an exception, because this method should never be called at the TAbstractFieldValuesAdapter (see page 5180) level: The TAbstractFieldValuesAdapter (see page 5180) ImplStartIterator (see page 5190) method returns false, indicating that this adapter does not support iteration.

Descendant classes that implement support for iterating over field values override this method to end that iteration process.

See Also

EndIterator (see page 5542)

ImplStartIterator (see page 5190)

ImplNextIteration (see page 5189)

54.1.3.2.6 TAbstractFieldValuesAdapter.ImplGetField Method

Provides the underlying implementation for the GetField (see page 5186) method.

Pascal

```
function ImplGetField: TObject; virtual; abstract;
```

C++

```
virtual __fastcall TObject * ImplGetField() = 0;
```

Description

The GetField (see page 5186) method calls ImplGetField to obtain the object that represents this adapter's single field. Unlike the GetField (see page 5186) method, ImplGetField is virtual, so that descendant classes can override it to provide a class-specific mechanism for obtaining the adapter field object.

In TAbstractFieldValuesAdapter (see page 5180), ImplGetField is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the GetField (see page 5186) method.

See Also

GetField (see page 5186)

54.1.3.2.7 TAbstractFieldValuesAdapter.ImplGetFieldsList Method

Provides the underlying implementation for the GetFieldsList (see page 5549) method.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

The GetFieldsList (see page 5549) method calls ImplGetFieldsList to fill a string list with a list of all fields that the adapter includes. Each entry in the list is the name of one of the adapter's field, and has an associated object that is the class of the object that represents that adapter field. GetFieldsList (see page 5549) is part of the design-time support for editing adapters.

AList is an existing string list to fill with entries for all of the fields this adapter can include.

In TAbstractFieldValuesAdapter (see page 5180), ImplGetFieldsList clears AList and inserts a single entry for a field named 'ValueField'. It associates the class that the GetValueFieldClass (see page 5186) method returns with this field name.

See Also

GetValueFieldClass (see page 5186)

GetFieldsList (see page 5549)

54.1.3.2.8 TAbstractFieldValuesAdapter.ImplGetValueIndex Method

Provides the underlying implementation for the GetValueIndex (see page 5187) method.

Pascal

```
function ImplGetValueIndex: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int ImplGetValueIndex() = 0;
```

Description

The GetValueIndex (see page 5187) method calls ImplGetValueIndex to obtain the index of current value of this adapter's single field. Unlike the GetValueIndex (see page 5187) method, ImplGetValueIndex is virtual, so that descendant classes can override it to provide a class-specific mechanism for obtaining this index.

In TAbstractFieldValuesAdapter (see page 5180), ImplGetValueIndex is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the GetValueIndex (see page 5187) method.

See Also

GetValueIndex (see page 5187)

54.1.3.2.9 TAbstractFieldValuesAdapter.ImplNextIteration Method

Provides the underlying implementation for the NextIteration (see page 5564) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

The NextIteration (see page 5564) method calls ImplNextIteration when iterating over field values to determine whether there

is another field value after the current one and if so, to change the current value to the next one.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. On entry, this value is typically the same as the value returned by the last call to NextIteration (see page 5564) or to StartIterator (see page 5567).

In TAbstractFieldValuesAdapter (see page 5180), ImplNextIteration always returns false, because TAbstractFieldValuesAdapter (see page 5180) does not implement support for iteration. Descendant classes that support iterating over field values override this method to check whether their is a next value and if so, move to it.

See Also

NextIteration (see page 5564)

ImplStartIterator (see page 5190)

ImplEndIterator (see page 5188)

54.1.3.2.10 TAbstractFieldValuesAdapter.ImplStartIterator Method

Provides the underlying implementation for the StartIterator (see page 5567) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

The StartIterator (see page 5567) method calls ImplStartIterator to start the process of iterating through the adapter's field's possible values. The return value indicates whether the adapter's field has any values over which to iterate.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way. Depending on the adapter, it may be supplied by the caller, or created by the ImplStartIterator method.

In TAbstractFieldValuesAdapter (see page 5180), ImplNextIteration (see page 5189) always returns false, because TAbstractFieldValuesAdapter (see page 5180) does not implement support for obtaining any field values. Descendant classes that override this method to check whether they can move to the first value in the field's set of values. If so, they move to the first value and return true. Otherwise they return false.

See Also

StartIterator (see page 5567)

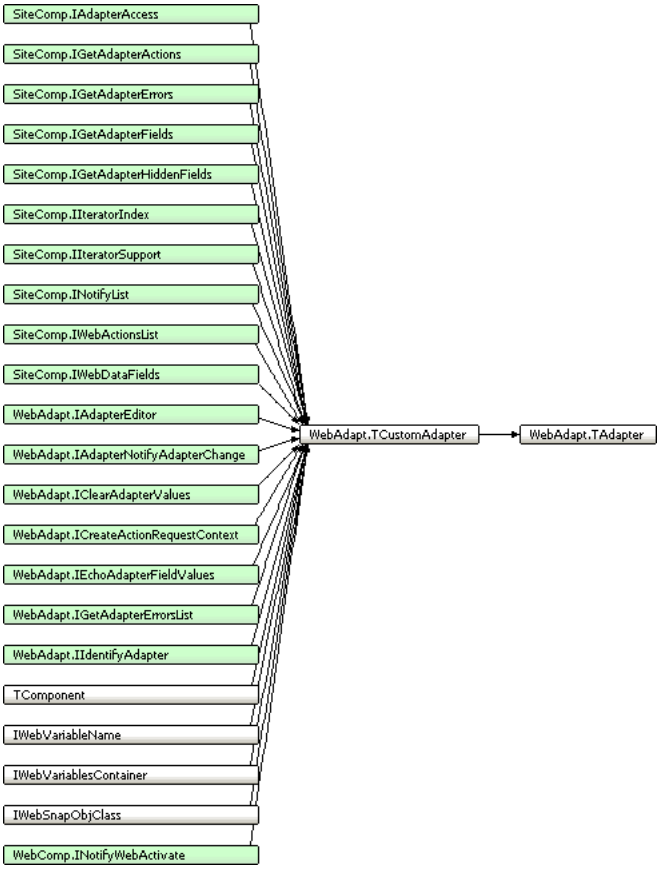
ImplEndIterator (see page 5188)

ImplNextIteration (see page 5189)

54.1.4 TAdapter Class

TAdapter defines a scriptable interface to event handlers that return data items or execute commands in your application.

Class Hierarchy



Pascal

```
TAdapter = class(TCustomAdapter);
```

C++

```
class TAdapter : public TCustomAdapter;
```

File

WebAdapt

Description

Use TAdapter to define a custom interface to application logic. Adapters contain field components and action components that represent data items and commands, respectively. Server-side script statements access the value of Adapter fields and the parameters of Adapter actions in order to build HTML forms and tables. Server-side script can be written by hand or automatically generated by the TAdapterPageProducer component. When an HTML form containing Adapter commands is submitted, the TAdapterDispatcher may automatically call an Adapter action component to execute the command. When an HTTP request for an adapter image field is received the TAdapterDispatcher may automatically call an Adapter field to retrieve the image.

For each field component added to TAdapter, you must implement an event handler to retrieve the data. For each action component added to TAdapter, you must implement an event handler to execute the command. Other adapter components, such as TDataSetAdapter and TLoginFormAdapter (see page 5824), have built in fields and actions that do not require events.

54.1.4.1 TAdapter Members

The following tables list the members exposed by TAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.

	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
V	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
V	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
V	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
V	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
V	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.

	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5197)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5197)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5198)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5198)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5198)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5199)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5199)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5200)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5200)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5200)	Occurs when the adapter needs to iterate over its data records.

Published Properties

	Name	Description
	Actions (see page 5195)	Actions is the container for adapter actions components.
	Data (see page 5196)	Data is the container for adapter field components.
	ModifyAccess (see page 5196)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5196)	Identifies the rights required to view this adapter's fields.

54.1.4.2 TAdapter Properties

The properties of the TAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5195)	Actions is the container for adapter actions components.
	Data (see page 5196)	Data is the container for adapter field components.
	ModifyAccess (see page 5196)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5196)	Identifies the rights required to view this adapter's fields.

54.1.4.2.1 TAdapter.Actions Property

Actions is the container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion,

deletion, and lookup of adapter actions by name.

See Also

TAdapterActions (see page 5209)

54.1.4.2.2 TAdapter.Data Property

Data is the container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterFields (see page 5276)

54.1.4.2.3 TAdapter.ModifyAccess Property

Identifies the rights required to update this adapter's fields.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this fields of this adapter. When server-side script accesses the CanModify property, this ModifyAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

ImplHasModifyAccess (see page 5558)

54.1.4.2.4 TAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses

the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

ImplHasViewAccess (see page 5559)

54.1.4.3 TAdapter Events

The events of the TAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5197)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5197)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5198)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5198)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5198)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5199)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5199)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5200)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5200)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5200)	Occurs when the adapter needs to iterate over its data records.

54.1.4.3.1 TAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.4.3.2 TAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.4.3.3 TAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 5568)

OnBeforeUpdateFields (see page 5578)

Errors (see page 5573)

54.1.4.3.4 TAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 5569)

OnBeforeValidateFields (see page 5578)

Errors (see page 5573)

54.1.4.3.5 TAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5478)

OnAfterExecuteAction (see page 5576)

54.1.4.3.6 TAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5478)

OnAfterGetActionResponse (see page 5576)

54.1.4.3.7 TAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 5568)

OnAfterUpdateFields (see page 5576)

Errors (see page 5573)

54.1.4.3.8 TAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 5569)

OnAfterValidateFields (see page 5577)

Errors (see page 5573)

54.1.4.3.9 TAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5478)

54.1.4.3.10 TAdapter.OnIterateRecords Event

Occurs when the adapter needs to iterate over its data records.

Pascal

```
property OnIterateRecords: TIterateRecordsEvent;
```

C++

```
__property TIterateRecordsEvent OnIterateRecords;
```

Description

Implement this event when the adapter supports multiple data records. This event handler is called when server-side script iterates over the records in adapter, such as when generating an HTML table displaying all records. The event handler must update an iteration pointer or cursor that adapter fields are also aware of. Adapter fields must use this iteration pointer when

called to retrieve their value.

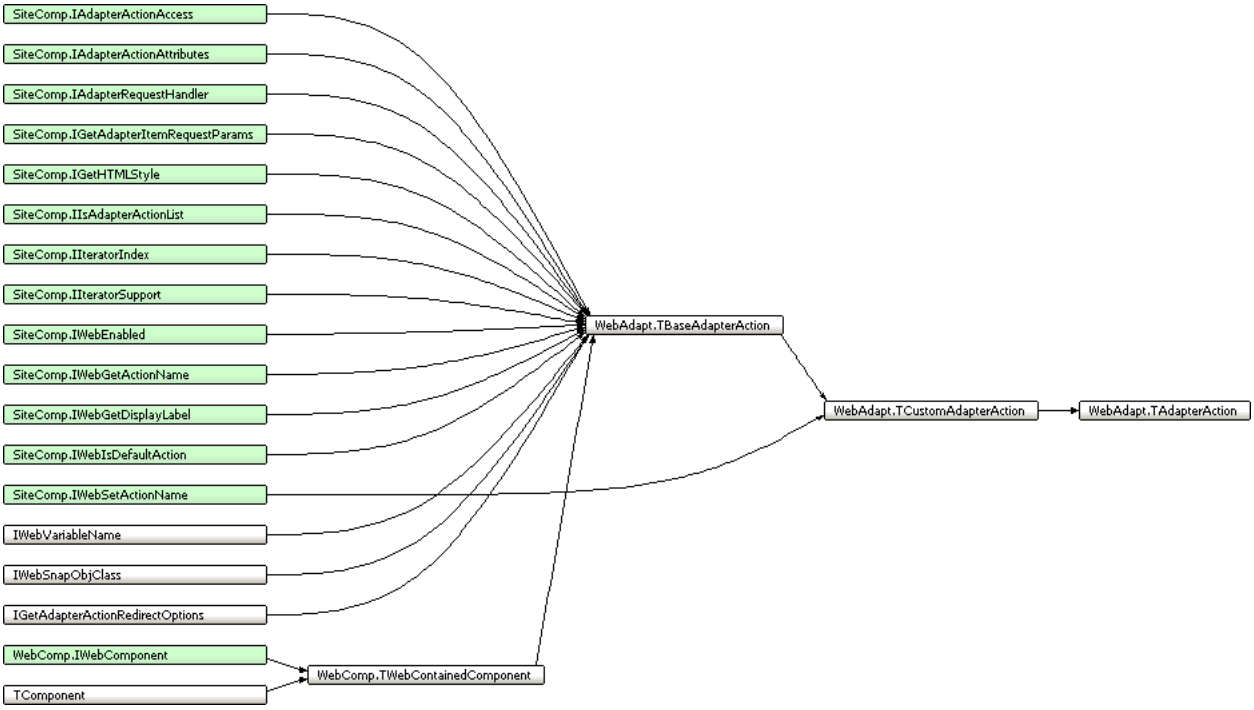
See Also

IIteratorSupport (see page 1054)

54.1.5 TAdapterAction Class

TAdapterAction is a generic adapter action that relies on event handlers.

Class Hierarchy



Pascal

```
TAdapterAction = class(TCustomAdapterAction);
```

C++

```
class TAdapterAction : public TCustomAdapterAction;
```

File

WebAdapt

Description

TAdapterAction publishes the properties in TCustomAdapterAction (see page 5579). TCustomAdapterAction (see page 5579) calls event handlers to retrieve the adapter action's parameters, get the state of the action, and execute the adapter action. Add TAdapterAction to TAdapter (see page 5190) or other adapter components types when you need an adapter action that will be controlled by event handlers. Other adapter action components declare events but also have built-in logic to work with a particular type of adapter. For example, TAdapterNextPageAction (see page 5370) has built-in logic to take the end user to the next page provided by TPagedAdapter (see page 5844). TAdapterAction, on the other hand, does not have this sort of built-in logic.

See Also

TAdapterActions (see page 5209)

TAdapter (see page 5190)

TAdapterDispatcher (see page 6148)

54.1.5.1 TAdapterAction Members

The following tables list the members exposed by TAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.

	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.

	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5207)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5207)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5208)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5208)	Called before the adapter action response is generated.

	OnExecute (see page 5208)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5209)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5209)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	ActionName (see page 5205)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 5205)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5206)	Defines the rights required to execute this action.
	RedirectOptions (see page 5206)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.5.2 TAdapterAction Properties

The properties of the TAdapterAction class are listed here.

Published Properties

	Name	Description
	ActionName (see page 5205)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 5205)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5206)	Defines the rights required to execute this action.
	RedirectOptions (see page 5206)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.5.2.1 TAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;__property UnicodeString ActionName;__property
UnicodeString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

ImplGetActionName (see page 5584)

ImplSetActionName (see page 5585)

54.1.5.2.2 TAdapterAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5465)

SetDisplayLabel (see page 5472)

54.1.5.2.3 TAdapterAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6796)

HasExecuteAccess (see page 5552)

54.1.5.2.4 TAdapterAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

54.1.5.3 TAdapterAction Events

The events of the TAdapterAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5207)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5207)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5208)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5208)	Called before the adapter action response is generated.
	OnExecute (see page 5208)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5209)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5209)	Specifies name/value pairs needed to execute this adapter action.

54.1.5.3.1 TAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.5.3.2 TAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.5.3.3 TAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.5.3.4 TAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.5.3.5 TAdapterAction.OnExecute Event

Called to execute this action using parameters passed in the HTTP request.

Pascal

```
property OnExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnExecute;
```

Description

Implement this event handler to execute this adapter action. Params contains the name/value pairs that were passed in the HTTP request. OnExecute is called by ImplExecuteActionRequest.

See Also

ImplExecuteActionRequest (see page 5584)

OnGetParams (see page 5478)

54.1.5.3.6 TAdapterAction.OnGetEnabled Event

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
property OnGetEnabled: TActionGetEnabledEvent;
```

C++

```
__property TActionGetEnabledEvent OnGetEnabled;
```

Description

ImplWebEnabled calls the OnGetEnabled event handler. Set the Enabled parameter to false to indicate that this adapter action should not be enabled.

See Also

ImplWebEnabled (see page 5585)

54.1.5.3.7 TAdapterAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

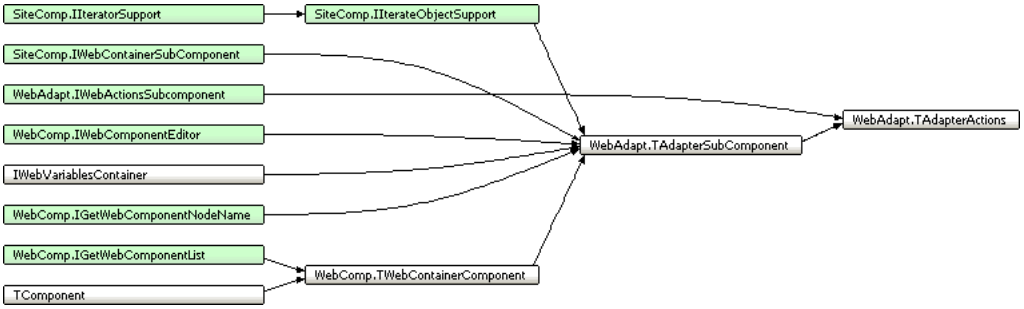
See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.6 TAdapterActions Class

TAdapterActions manages the list of actions that belong to an adapter.

Class Hierarchy



Pascal

```
TAdapterActions = class(TAdapterSubComponent, IWebActionsSubcomponent);
```

C++

```
class TAdapterActions : public TAdapterSubComponent, public IWebActionsSubcomponent;
```

File

WebAdapt

Description

TAdapterActions is the type of the Actions property on an adapter component. It manages a list that contains all the actions that belong to the adapter. The action objects (TBaseAdapterAction (see page 5447) descendants) are available using the WebActions (see page 5214) property. TAdapterActions also includes support for associating the actions with variable names so that server-side script can execute the actions.

TAdapterActions represents the list of explicitly assigned actions. Adapters that include a set of default actions use a TAdapterDefaultActions (see page 5231) object instead.

See Also

TCustomAdapter (see page 5531)

TAdapterFields (see page 5276)

TAdapterDefaultActions (see page 5231)

TAdapterDefaultFields (see page 5237)

TBaseAdapterAction (see page 5447)

54.1.6.1 TAdapterActions Members

The following tables list the members exposed by TAdapterActions.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5212)	Indicates whether the associated adapter supports a specified class of action component.
	CanAddClass (see page 5408)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5409)	Stops the TAdapterSubComponent (see page 5406) object from iterating over the Web items it lists.
	FindNamedVariable (see page 5409)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5410)	Returns the item from this list with a specified variable name.

	FindVariableContainer (see page 5410)	Returns the Web module that contains a specified item from this list.
V	GetChildren (see page 6065)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5411)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5411)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
V	GetDefaultWebComponents (see page 6065)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5412)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5413)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5413)	Returns the value of the VariablesInUse (see page 5419) property.
	GetWebComponentNodeName (see page 5413)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6065)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5212)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
V	ImplGetDefaultComponentList (see page 6066)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5213)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	NextIteration (see page 5415)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5416)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6066)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 5416)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetDefaultWebComponents (see page 6067)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetWebParent (see page 5213)	Associates this component with an adapter.
	StartIterator (see page 5418)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
R	Adapter (see page 5418)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 5419)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 6067)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.
V	Create (see page 6064)	Creates and initializes an instance of TWebContainerComponent (see page 6063).
V	Destroy (see page 6064)	Frees an instance of TWebContainerComponent (see page 6063).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
V	GetParentComponent (see page 5412)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6083)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
V	HasParent (see page 5414)	Indicates whether this component has an associated adapter.
	NextIteration (see page 1056)	Advances to the objects next item.
V	ReadState (see page 5416)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 5417)	Sets the Adapter (see page 5418) property to be the same as this component's parent.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	WebActions (see page 5214)	Lists the action components that this object manages.

54.1.6.2 TAdapterActions Methods

The methods of the TAdapterActions class are listed here.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5212)	Indicates whether the associated adapter supports a specified class of action component.
	ImplCanAddClass (see page 5212)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
	ImplGetWebComponentNodeName (see page 5213)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	SetWebParent (see page 5213)	Associates this component with an adapter.

54.1.6.2.1 TAdapterActions.AdapterCanAddClass Method

Indicates whether the associated adapter supports a specified class of action component.

Pascal

```
function AdapterCanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean AdapterCanAddClass(TComponent * AParent, TClass AClass);
```

Description

TAdapterActions (see page 5209) uses this method internally as part of its implementation for the CanAddClass (see page 5408) method. This allows the TAdapterActions (see page 5209) object to indicate, at design time, which types of action components are valid as components in the WebActions (see page 5214) property list.

AParent is the instance of TAdapterActions (see page 5209) to which the child actions are to be added.

AClass describes the class of the child component to be added to AParent.

AdapterCanAddClass returns true if the adapter recognizes and supports AClass as a child action, false otherwise.

AdapterCanAddClass calls the CanAddActionClass method of its associated adapter to determine whether it accepts the specified action class.

See Also

WebActions (see page 5214)

Adapter (see page 5418)

ImplCanAddClass (see page 5212)

CanAddActionClass (see page 5539)

54.1.6.2.2 TAdapterActions.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5408) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 5408) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterActions (see page 5209) object to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterActions (see page 5209) object can include a child of type AClass, false otherwise.

ImplCanAddClass returns true if one of the following two conditions is met:

AClass is an adapter action class that the associated adapter recognizes and accepts.

AClass is a custom adapter action (a descendant of TCustomAdapterAction (see page 5579) that is not a descendant of TImplementedAdapterAction (see page 5818)).

See Also

CanAddClass (see page 5408)

AdapterCanAddClass (see page 5212)

TCustomAdapterAction (see page 5579)

TImplementedAdapterAction (see page 5818)

54.1.6.2.3 TAdapterActions.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.

Pascal

```
function ImplGetWebComponentNodeName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 5413) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the string 'Actions'.

See Also

GetWebComponentNodeName (see page 5413)

54.1.6.2.4 TAdapterActions.SetWebParent Method

Associates this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); override;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

SetWebParent sets up the association between this component and its parent adapter.

Value is the adapter component that acts as a parent to this collection.

SetWebParent assigns Value to the protected Adapter (see page 5418) property and replaces that adapter's Actions property with this TAdapterActions (see page 5209) instance.

See Also

Actions (see page 5572)

Adapter (see page 5418)

54.1.6.3 TAdapterActions Properties

The properties of the TAdapterActions class are listed here.

Public Properties

	Name	Description
	WebActions (see page 5214)	Lists the action components that this object manages.

54.1.6.3.1 TAdapterActions.WebActions Property

Lists the action components that this object manages.

Pascal

```
property WebActions: TWebComponentList;
```

C++

```
__property TWebComponentList WebActions;
```

Description

Read (see page 3634) WebActions to access the list of action components. Each component in this list is a descendant of TBaseAdapterAction (see page 5447).

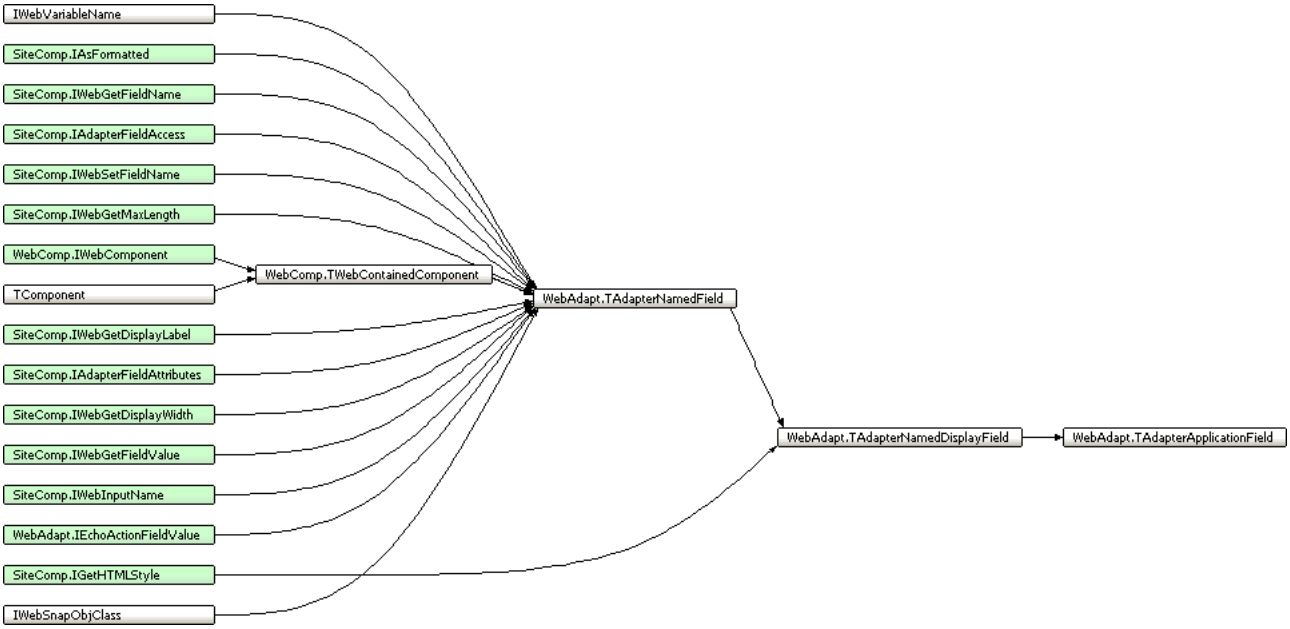
See Also

TBaseAdapterAction (see page 5447)

54.1.7 TAdapterApplicationField Class

TAdapterApplicationField is a base class for adapter field components that are children of a TApplicationAdapter (see page 5439) component.

Class Hierarchy



Pascal

```
TAdapterApplicationField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterApplicationField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterApplicationField implements functionality useful to an adapter field that can be a child of TApplicationAdapter (see page 5439).

See Also

TApplicationAdapter (see page 5439)

TAdapterNamedDisplayField (see page 5343)

TAdapterApplicationTitleField (see page 5218)

54.1.7.1 TAdapterApplicationField Members

The following tables list the members exposed by TAdapterApplicationField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.

	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.

	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
R	Adapter (see page 5218)	Returns the parent adapter of this adapter field.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.

	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.7.2 TAdapterApplicationField Properties

The properties of the TAdapterApplicationField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5218)	Returns the parent adapter of this adapter field.

54.1.7.2.1 TAdapterApplicationField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TCustomApplicationAdapter;
```

C++

```
__property TCustomApplicationAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter.

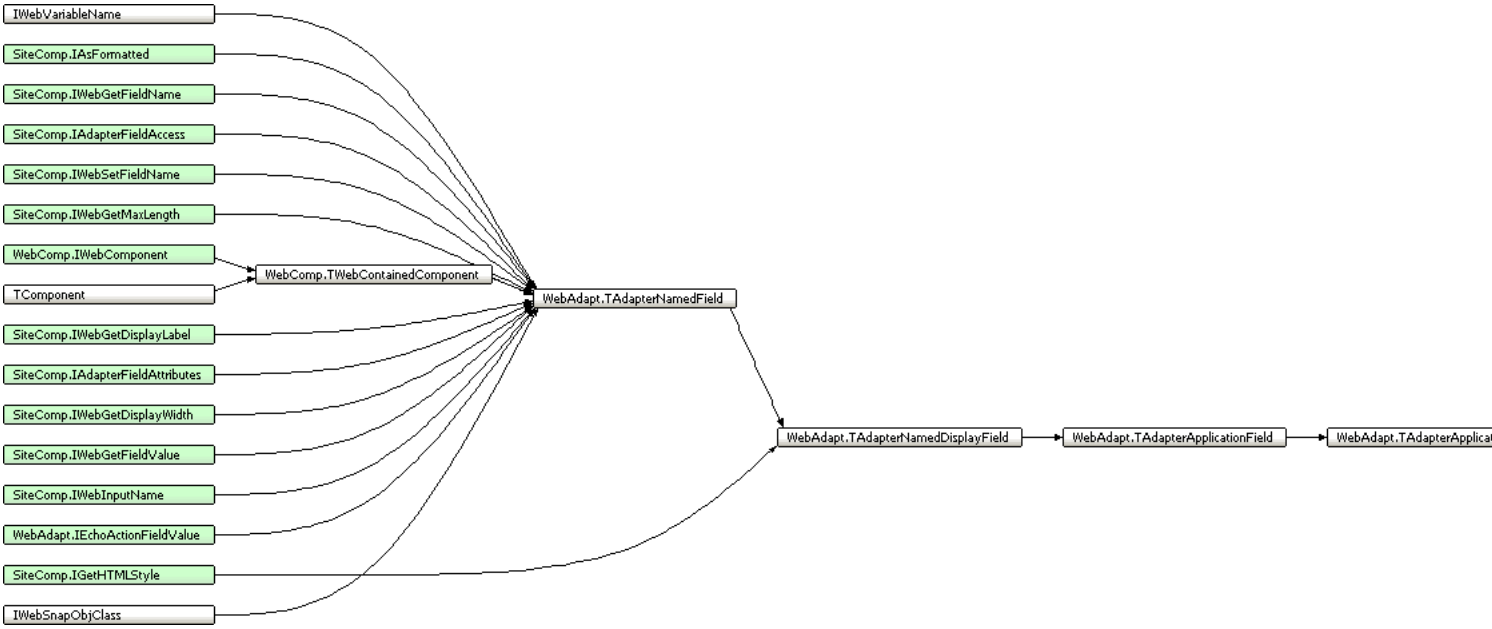
See Also

TCustomApplicationAdapter (see page 5663)

54.1.8 TAdapterApplicationTitleField Class

TAdapterApplicationTitleField is an adapter field used to access the application title.

Class Hierarchy



Pascal

```
TAdapterApplicationTitleField = class(TAdapterApplicationField);
```

C++

```
class TAdapterApplicationTitleField : public TAdapterApplicationField;
```

File

WebAdapt

Description

TAdapterApplicationTitleField is added to TApplicationAdapter (see page 5439).

See Also

TAdapterApplicationField (see page 5214)

TApplicationAdapter (see page 5439)

TCustomApplicationAdapter (see page 5663)

54.1.8.1 TAdapterApplicationTitleField Members

The following tables list the members exposed by TAdapterApplicationTitleField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.

	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5222)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5222)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5218)	Returns the parent adapter of this adapter field.
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.

	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.8.2 TAdapterApplicationTitleField Methods

The methods of the TAdapterApplicationTitleField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 5222)	Returns the name used to access this adapter field with server-side script.
	ImplGetValue (see page 5222)	Returns the value associated with this adapter field.

54.1.8.2.1 TAdapterApplicationTitleField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'Title'. This allows server-side script to access this adapter field using the parent adapter's Title property.

54.1.8.2.2 TAdapterApplicationTitleField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the value of the parent adapter's Title property. Server-side script accesses the result of this function using the DisplayText property of this adapter field. Server-side script may use this value to display the application title on a Web page.

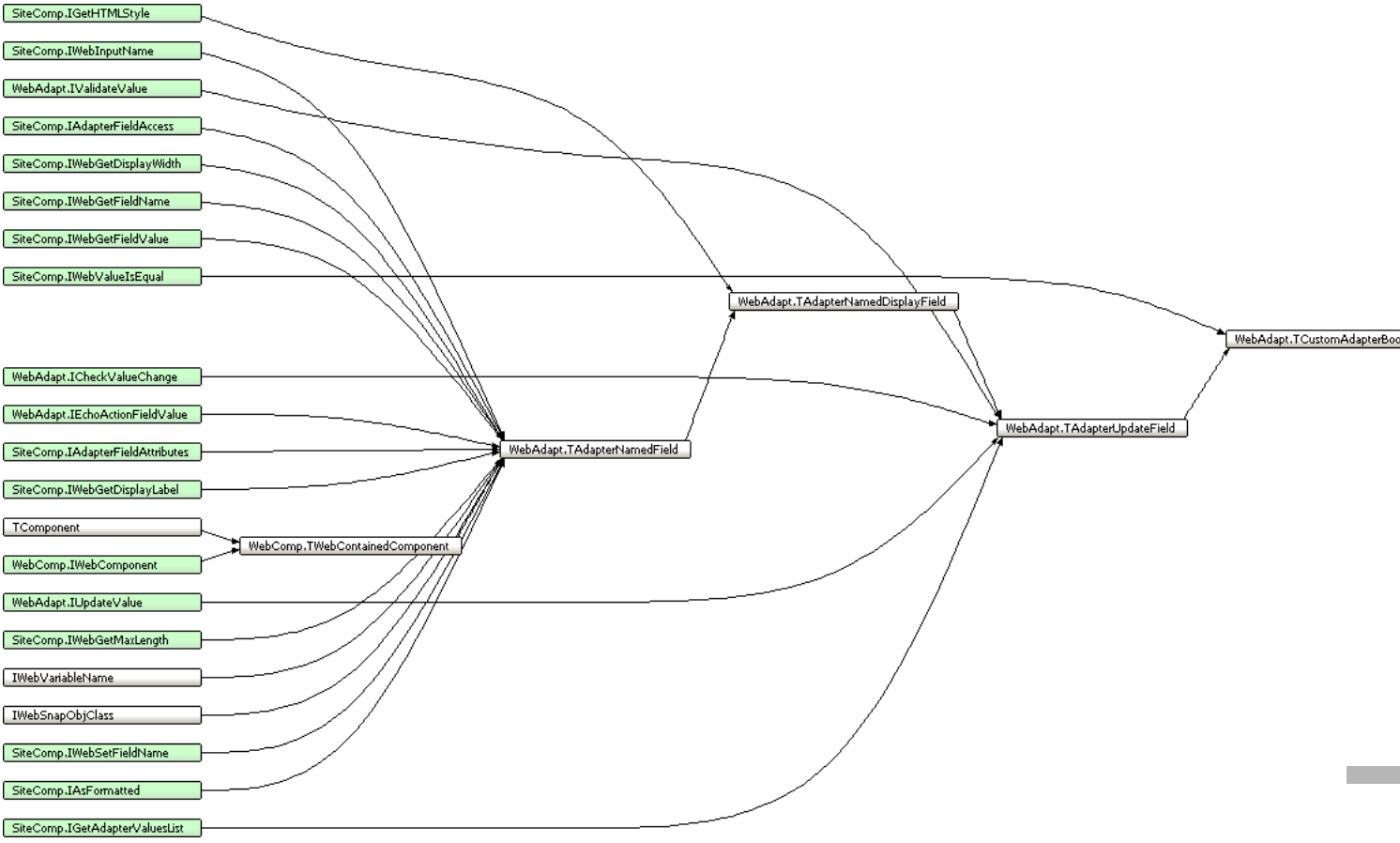
See Also

ApplicationTitle (see page 5671)

54.1.9 TAdapterBooleanField Class

TAdapterBooleanField is an event handler based adapter field that works with a Boolean value.

Class Hierarchy



Pascal

```
TAdapterBooleanField = class(TCustomAdapterBooleanField);
```

C++

```
class TAdapterBooleanField : public TCustomAdapterBooleanField;
```

File

WebAdapt

Description

TAdapterBooleanField publishes the properties in TCustomAdapterBooleanField (see page 5589). TCustomAdapterBooleanField (see page 5589) calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

Add TAdapterBooleanField to TAdapter (see page 5190) or other adapter components types when you need an adapter field that works with a Boolean value. There are other adapter field components that use event handlers. They differ in the type and

number of values that are supported. See TAdapterField (see page 5267), TAdapterMultiValueField (see page 5332), TAdapterMemoField (see page 5323), TAdapterFileField (see page 5281) and TAdapterImageField (see page 5308).

See Also

TAdapter (see page 5190)

TAdapterField (see page 5267)

TAdapterMemoField (see page 5323)

TAdapterMultiValueField (see page 5332)

TAdapterFileField (see page 5281)

TAdapterImageField (see page 5308)

54.1.9.1 TAdapterBooleanField Members

The following tables list the members exposed by TAdapterBooleanField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValue (see page 5594)	Retrieves a Boolean value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5594)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5595)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).

	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetValue (see page 5595)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplIsEqual (see page 5596)	Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5596)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsEqual (see page 5596)	Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.

	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	IsEqual (see page 1119)	This is IsEqual, a member of class IWebValueIsEqual.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValue (see page 5230)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5230)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5231)	Event handler called to validate value(s) entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5228)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5228)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5228)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5229)	Identifies the rights required to update this field.
	ReadOnly (see page 5229)	Indicates whether this adapter field is read-only.
	ValuesList (see page 5229)	Associates a list of values with the adapter field.
	ViewAccess (see page 5230)	Identifies the rights required to view this field.

54.1.9.2 TAdapterBooleanField Properties

The properties of the TAdapterBooleanField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5228)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5228)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5228)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5229)	Identifies the rights required to update this field.
	ReadOnly (see page 5229)	Indicates whether this adapter field is read-only.
	ValuesList (see page 5229)	Associates a list of values with the adapter field.
	ViewAccess (see page 5230)	Identifies the rights required to view this field.

54.1.9.2.1 TAdapterBooleanField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.9.2.2 TAdapterBooleanField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.9.2.3 TAdapterBooleanField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.9.2.4 TAdapterBooleanField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

54.1.9.2.5 TAdapterBooleanField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.9.2.6 TAdapterBooleanField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1072)

TStringsValuesList (see page 5857)

TDataSetValuesList

54.1.9.2.7 TAdapterBooleanField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

54.1.9.3 TAdapterBooleanField Events

The events of the TAdapterBooleanField class are listed here.

Published Events

	Name	Description
	OnGetValue (see page 5230)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5230)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5231)	Event handler called to validate value(s) entered by the end-user.

54.1.9.3.1 TAdapterBooleanField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetBooleanEvent;
```

C++

```
__property TAdapterFieldGetBooleanEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5595)

54.1.9.3.2 TAdapterBooleanField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldBooleanValueEvent;
```

C++

```
__property TAdapterFieldBooleanValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 5596)

54.1.9.3.3 TAdapterBooleanField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

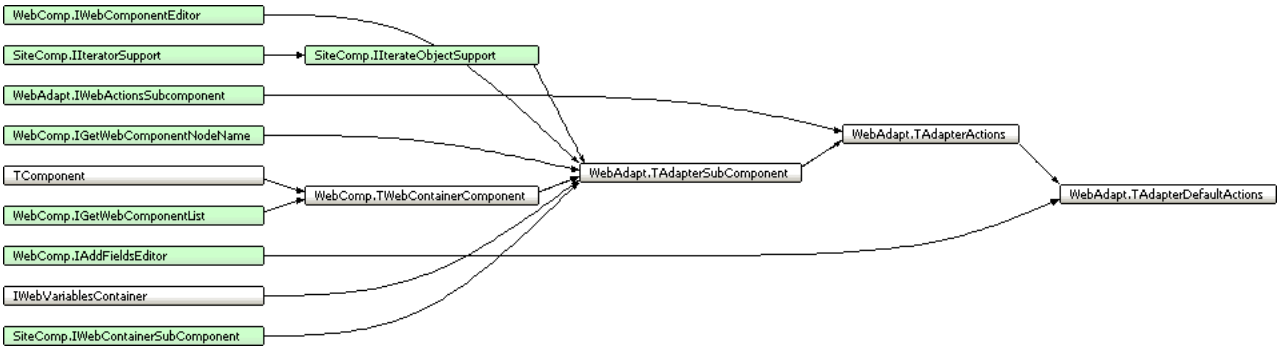
DoOnValidateValue (see page 5426)

IActionFieldValue

54.1.10 TAdapterDefaultActions Class

TAdapterDefaultActions manages the list of actions that belong to an adapter that supports a default set of actions.

Class Hierarchy



Pascal

```
TAdapterDefaultActions = class(TAdapterActions, IAddFieldsEditor);
```

C++

```
class TAdapterDefaultActions : public TAdapterActions, public IAddFieldsEditor;
```

File

WebAdapt

Description

TAdapterDefaultActions is the type of the Actions property on an adapter component. It manages two lists of actions (TBaseAdapterAction (see page 5447) descendants) that belong to the adapter: the list of actions that have been explicitly added to the adapter, and the list of actions that the adapter includes by default. The explicitly added action objects are available using the WebActions (see page 5214) property. The default actions are available using the DefaultWebComponents (see page 5237) property.

See Also

TDefaultFieldsAdapter (see page 5738)

TAdapterFields (see page 5276)

TAdapterActions (see page 5209)

TAdapterDefaultFields (see page 5237)

TBaseAdapterAction (see page 5447)

54.1.10.1 TAdapterDefaultActions Members

The following tables list the members exposed by TAdapterDefaultActions.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5212)	Indicates whether the associated adapter supports a specified class of action component.
	CanAddClass (see page 5408)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5409)	Stops the TAdapterSubComponent (see page 5406) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 5409)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5410)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5410)	Returns the Web module that contains a specified item from this list.
	GetAddAllFieldsItem (see page 5235)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.
	GetAddFieldsDlgCaption (see page 5235)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.
	GetAddFieldsItem (see page 5235)	This is GetAddFieldsItem, a member of class TAdapterDefaultActions.
V	GetChildren (see page 6065)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5411)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5235)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5236)	This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.
	GetVariable (see page 5412)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5413)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5236)	Returns the value of the VariablesInUse property.
	GetWebComponentNodeName (see page 5413)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6065)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5212)	Provides the underlying implementation for the CanAddClass (see page 5408) method.

	ImplCanAddClass (see page 5414)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
	ImplGetDefaultComponentList (see page 6066)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	ImplGetWebComponentNodeName (see page 5213)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	ImplGetWebComponentNodeName (see page 5415)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	NextIteration (see page 5415)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5416)	Notifies the associated adapter when the list of items changes.
	SetChildOrder (see page 6066)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 5236)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetWebParent (see page 5213)	Associates this component with an adapter.
	SetWebParent (see page 5417)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5418)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
	Adapter (see page 5418)	Indicates the adapter that uses this object to manage its fields or actions.
	VariablesInUse (see page 5419)	Lists those fields or actions in the collection for which variables are defined.
	WebContainerComponentHelper (see page 6067)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.
	ClearDefaultComponents (see page 5234)	Empties the list of default actions that is the value of the DefaultWebComponents (see page 5237) property.
	Create (see page 6064)	Creates and initializes an instance of TWebContainerComponent (see page 6063).
	Destroy (see page 5233)	Frees an instance of TAdapterDefaultActions (see page 5231).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetParentComponent (see page 5412)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6083)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 5414)	Indicates whether this component has an associated adapter.
	NextIteration (see page 1056)	Advances to the objects next item.
	ReadState (see page 5416)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 5417)	Sets the Adapter (see page 5418) property to be the same as this component's parent.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	DefaultWebComponents (see page 5237)	Lists the action components that the associated adapter includes by default.
	WebActions (see page 5214)	Lists the action components that this object manages.

54.1.10.2 TAdapterDefaultActions.Destroy Destructor

Frees an instance of TAdapterDefaultActions (see page 5231).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDefaultActions();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3372)

54.1.10.3 TAdapterDefaultActions Methods

The methods of the TAdapterDefaultActions class are listed here.

Protected Methods

	Name	Description
	GetAddAllFieldsItem (see page 5235)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.
	GetAddFieldsDlgCaption (see page 5235)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.
	GetAddFieldsItem (see page 5235)	This is GetAddFieldsItem, a member of class TAdapterDefaultActions.
	GetDefaultWebComponents (see page 5235)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5236)	This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.
	GetVariablesInUse (see page 5236)	Returns the value of the VariablesInUse property.
	SetDefaultWebComponents (see page 5236)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Public Methods

	Name	Description
	ClearDefaultComponents (see page 5234)	Empties the list of default actions that is the value of the DefaultWebComponents (see page 5237) property.

54.1.10.3.1 TAdapterDefaultActions.ClearDefaultComponents Method

Empties the list of default actions that is the value of the DefaultWebComponents (see page 5237) property.

Pascal

```
procedure ClearDefaultComponents; virtual;
```

C++

```
virtual __fastcall ClearDefaultComponents();
```

Description

The associated adapter calls ClearDefaultComponents when it changes its set of default actions. After a call to ClearDefaultComponents, the adapter generates a new set of default components the next time the application reads the default actions.

See Also

DefaultWebComponents (see page 5237)

54.1.10.3.2 TAdapterDefaultActions.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TAdapterDefaultActions.

54.1.10.3.3 TAdapterDefaultActions.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultActions.

54.1.10.3.4 TAdapterDefaultActions.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TAdapterDefaultActions.

54.1.10.3.5 TAdapterDefaultActions.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 6065)

TAdapterDefaultActions (see page 5231)

TAdapterDefaultFields (see page 5237)

54.1.10.3.6 TAdapterDefaultActions.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TAdapterDefaultActions.

54.1.10.3.7 TAdapterDefaultActions.GetVariablesInUse Method

Returns the value of the VariablesInUse property.

Pascal

```
function GetVariablesInUse: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse property.

In TAdapterSubComponent, GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 5419)

GetWebComponents (see page 6065)

54.1.10.3.8 TAdapterDefaultActions.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent, SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and

TAdapterDefaultFields) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 5411)

IsDefaultField (see page 6042)

54.1.10.4 TAdapterDefaultActions Properties

The properties of the TAdapterDefaultActions class are listed here.

Public Properties

	Name	Description
	DefaultWebComponents (see page 5237)	Lists the action components that the associated adapter includes by default.

54.1.10.4.1 TAdapterDefaultActions.DefaultWebComponents Property

Lists the action components that the associated adapter includes by default.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Read (see page 3634) DefaultWebComponents to access the list of action components that the adapter uses when you do not explicitly add any actions to this object. Each default action is a descendant of TAdapterNamedAction.

If you explicitly add action components to this object, the adapter uses the actions in the WebActions (see page 5214) property instead.

See Also

TBaseAdapterAction (see page 5447)

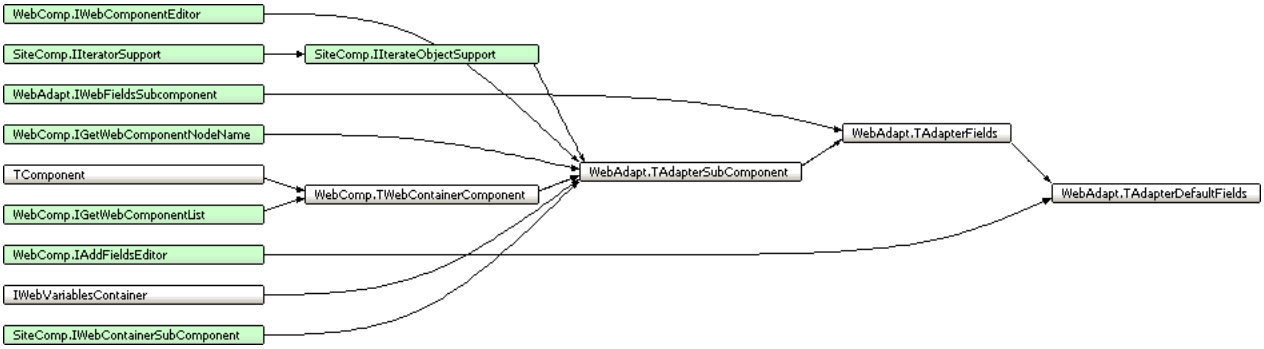
WebActions (see page 5214)

Adapter (see page 5418)

54.1.11 TAdapterDefaultFields Class

TAdapterDefaultFields manages the list of fields that belong to an adapter that supports a default set of fields.

Class Hierarchy



Pascal

```
TAdapterDefaultFields = class(TAdapterFields, IAddFieldsEditor);
```

C++

```
class TAdapterDefaultFields : public TAdapterFields, public IAddFieldsEditor;
```

File

WebAdapt

Description

TAdapterDefaultFields is the type of the Data property on an adapter component. It manages two lists of fields (TAdapterNamedField (see page 5351) descendants) that belong to the adapter: the list of fields that have been explicitly added to the adapter, and the list of fields that the adapter includes by default. The explicitly added field objects are available using the WebFields (see page 5281) property. The default fields are available using the DefaultWebComponents (see page 5243) property.

See Also

TDefaultFieldsAdapter (see page 5738)

TAdapterActions (see page 5209)

TAdapterDefaultActions (see page 5231)

TAdapterFields (see page 5276)

TAdapterNamedField (see page 5351)

54.1.11.1 TAdapterDefaultFields Members

The following tables list the members exposed by TAdapterDefaultFields.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5278)	Indicates whether the associated adapter supports a specified class of field component.
	CanAddClass (see page 5408)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5409)	Stops the TAdapterSubComponent (see page 5406) object from iterating over the Web items it lists.
	FindNamedVariable (see page 5409)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5410)	Returns the item from this list with a specified variable name.

	FindVariableContainer (see page 5410)	Returns the Web module that contains a specified item from this list.
	GetAddAllFieldsItem (see page 5241)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.
	GetAddFieldsDlgCaption (see page 5241)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.
	GetAddFieldsItem (see page 5241)	This is GetAddFieldsItem, a member of class TAdapterDefaultFields.
V	GetChildren (see page 6065)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5411)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5241)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5242)	This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.
	GetVariable (see page 5412)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5413)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5242)	Returns the value of the VariablesInUse property.
	GetWebComponentNodeName (see page 5413)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6065)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5279)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
V	ImplCanAddClass (see page 5414)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
V	ImplGetDefaultComponentList (see page 6066)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5280)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
V	ImplGetWebComponentNodeName (see page 5415)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	NextIteration (see page 5415)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5416)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6066)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 5242)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetWebParent (see page 5280)	Associates this component with an adapter.
V	SetWebParent (see page 5417)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5418)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
R	Adapter (see page 5418)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 5419)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 6067)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.
V	ClearDefaultComponents (see page 5240)	Empties the list of default fields that is the value of the DefaultWebComponents (see page 5243) property.
V	Create (see page 6064)	Creates and initializes an instance of TWebContainerComponent (see page 6063).
V	Destroy (see page 5240)	Frees an instance of TAdapterDefaultFields (see page 5237).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
V	GetParentComponent (see page 5412)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6083)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
V	HasParent (see page 5414)	Indicates whether this component has an associated adapter.

	NextIteration (see page 1056)	Advances to the objects next item.
V	ReadState (see page 5416)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 5417)	Sets the Adapter (see page 5418) property to be the same as this component's parent.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
R	DefaultWebComponents (see page 5243)	Lists the field components that the associated adapter includes by default.
R	WebFields (see page 5281)	Lists the field components that this object manages.

54.1.11.2 TAdapterDefaultFields.Destroy Destructor

Frees an instance of TAdapterDefaultFields (see page 5237).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDefaultFields();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for a nil reference before calling Destroy.

See Also

System::TObject::Free (see page 3372)

54.1.11.3 TAdapterDefaultFields Methods

The methods of the TAdapterDefaultFields class are listed here.

Protected Methods

	Name	Description
	GetAddAllFieldsItem (see page 5241)	This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.
	GetAddFieldsDlgCaption (see page 5241)	This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.
	GetAddFieldsItem (see page 5241)	This is GetAddFieldsItem, a member of class TAdapterDefaultFields.
V	GetDefaultWebComponents (see page 5241)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetNewFieldPrefix (see page 5242)	This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.
V	GetVariablesInUse (see page 5242)	Returns the value of the VariablesInUse property.
V	SetDefaultWebComponents (see page 5242)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Public Methods

	Name	Description
V	ClearDefaultComponents (see page 5240)	Empties the list of default fields that is the value of the DefaultWebComponents (see page 5243) property.

54.1.11.3.1 TAdapterDefaultFields.ClearDefaultComponents Method

Empties the list of default fields that is the value of the DefaultWebComponents (see page 5243) property.

Pascal

```
procedure ClearDefaultComponents; virtual;
```

C++

```
virtual __fastcall ClearDefaultComponents();
```

Description

The associated adapter calls ClearDefaultComponents when it changes its set of default fields. After a call to ClearDefaultComponents, the adapter generates a new set of default components the next time the application reads the default fields.

See Also

DefaultWebComponents (see page 5243)

54.1.11.3.2 TAdapterDefaultFields.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TAdapterDefaultFields.

54.1.11.3.3 TAdapterDefaultFields.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TAdapterDefaultFields.

54.1.11.3.4 TAdapterDefaultFields.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TAdapterDefaultFields.

54.1.11.3.5 TAdapterDefaultFields.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 6065)

TAdapterDefaultActions (see page 5231)

TAdapterDefaultFields (see page 5237)

54.1.11.3.6 TAdapterDefaultFields.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TAdapterDefaultFields.

54.1.11.3.7 TAdapterDefaultFields.GetVariablesInUse Method

Returns the value of the VariablesInUse property.

Pascal

```
function GetVariablesInUse: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse property.

In TAdapterSubComponent, GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 5419)

GetWebComponents (see page 6065)

54.1.11.3.8 TAdapterDefaultFields.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent, GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent, SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions and TAdapterDefaultFields) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 5411)

IsDefaultField (see page 6042)

54.1.11.4 TAdapterDefaultFields Properties

The properties of the TAdapterDefaultFields class are listed here.

Public Properties

	Name	Description
	DefaultWebComponents (see page 5243)	Lists the field components that the associated adapter includes by default.

54.1.11.4.1 TAdapterDefaultFields.DefaultWebComponents Property

Lists the field components that the associated adapter includes by default.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Read (see page 3634) DefaultWebComponents to access the list of field components that the adapter uses when you do not explicitly add any field components to this object. Each default field component is a descendant of TAdapterNamedField (see page 5351).

If you explicitly add field components to this object, the adapter uses the fields in the WebFields (see page 5281) property instead.

See Also

TAdapterNamedField (see page 5351)

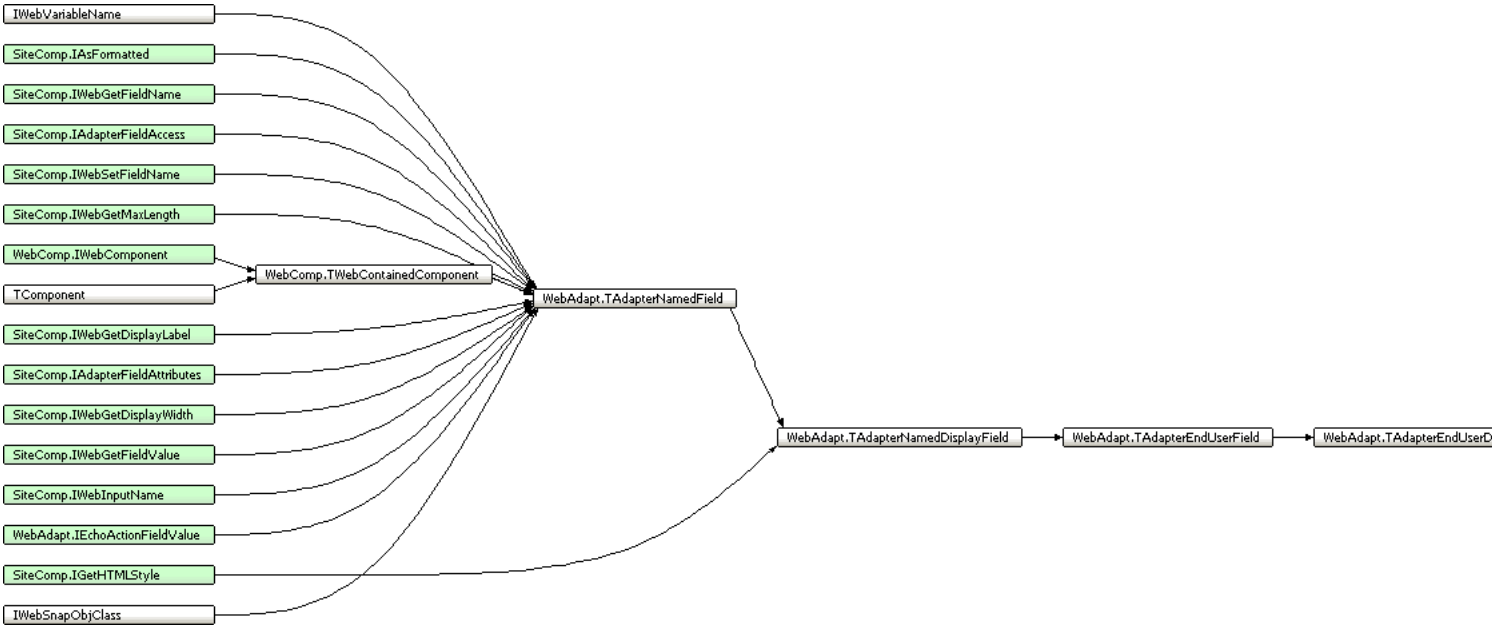
WebFields (see page 5281)

Adapter (see page 5418)

54.1.12 TAdapterEndUserDisplayNameField Class

TAdapterEndUserDisplayNameField is an adapter field used to access the end-user's name.

Class Hierarchy



Pascal

```
TAdapterEndUserDisplayNameField = class(TAdapterEndUserField);
```

C++

```
class TAdapterEndUserDisplayNameField : public TAdapterEndUserField;
```

File

WebAdapt

Description

TAdapterEndUserDisplayNameField is added to TApplicationAdapter (see page 5439).

See Also

TAdapterEndUserField (see page 5248)

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

54.1.12.1 TAdapterEndUserDisplayNameField Members

The following tables list the members exposed by TAdapterEndUserDisplayNameField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5247)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5247)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.

	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5251)	Returns the parent adapter of this adapter field.
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.

	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.12.2 TAdapterEndUserDisplayNameField Methods

The methods of the TAdapterEndUserDisplayNameField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 5247)	Returns the name used to access this adapter field with server-side script.
V	ImplGetValue (see page 5247)	Returns the value associated with this adapter field.

54.1.12.2.1 TAdapterEndUserDisplayNameField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'DisplayName'. This allows server-side script to access this adapter field using the parent adapter's DisplayName property.

54.1.12.2.2 TAdapterEndUserDisplayNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetDisplayNameValue. Server-side script accesses the result of this function using the

DisplayText property of this adapter field. Server-side script may use this value to display the end-user's name on a Web page.

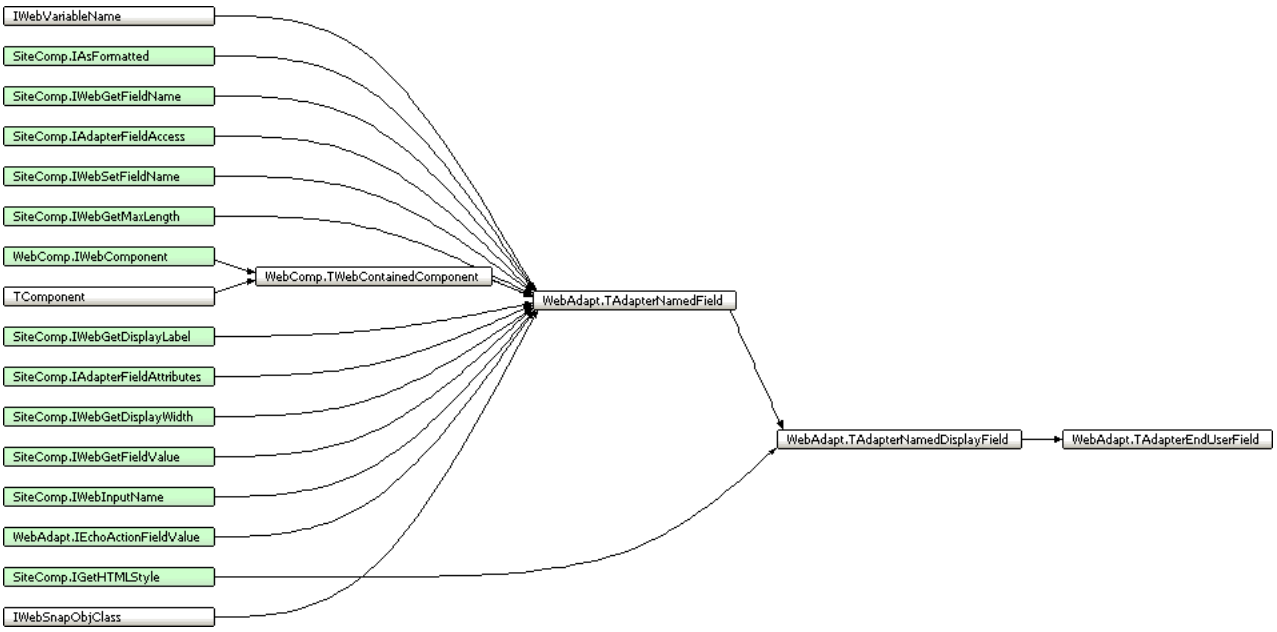
See Also

GetDisplayNameValue (see page 5507)

54.1.13 TAdapterEndUserField Class

TAdapterEndUserField is a base class for adapter field components that are children of an end-user adapter component.

Class Hierarchy



Pascal

```
TAdapterEndUserField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterEndUserField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterEndUserField implements functionality useful to an adapter field that can be a child of TEndUserAdapter (see page 5764) or TEndUserSessionAdapter (see page 5792).

See Also

TAdapterNamedDisplayField (see page 5343)

TBaseEndUserAdapter (see page 5497)

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

TAdapterEndUserLoggedInField (see page 5252)

TAdapterEndUserDisplayNameField (see page 5244)

54.1.13.1 TAdapterEndUserField Members

The following tables list the members exposed by TAdapterEndUserField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.

	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5251)	Returns the parent adapter of this adapter field.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.13.2 TAdapterEndUserField Properties

The properties of the TAdapterEndUserField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5251)	Returns the parent adapter of this adapter field.

54.1.13.2.1 TAdapterEndUserField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TBaseEndUserAdapter;
```

C++

```
__property TBaseEndUserAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter.

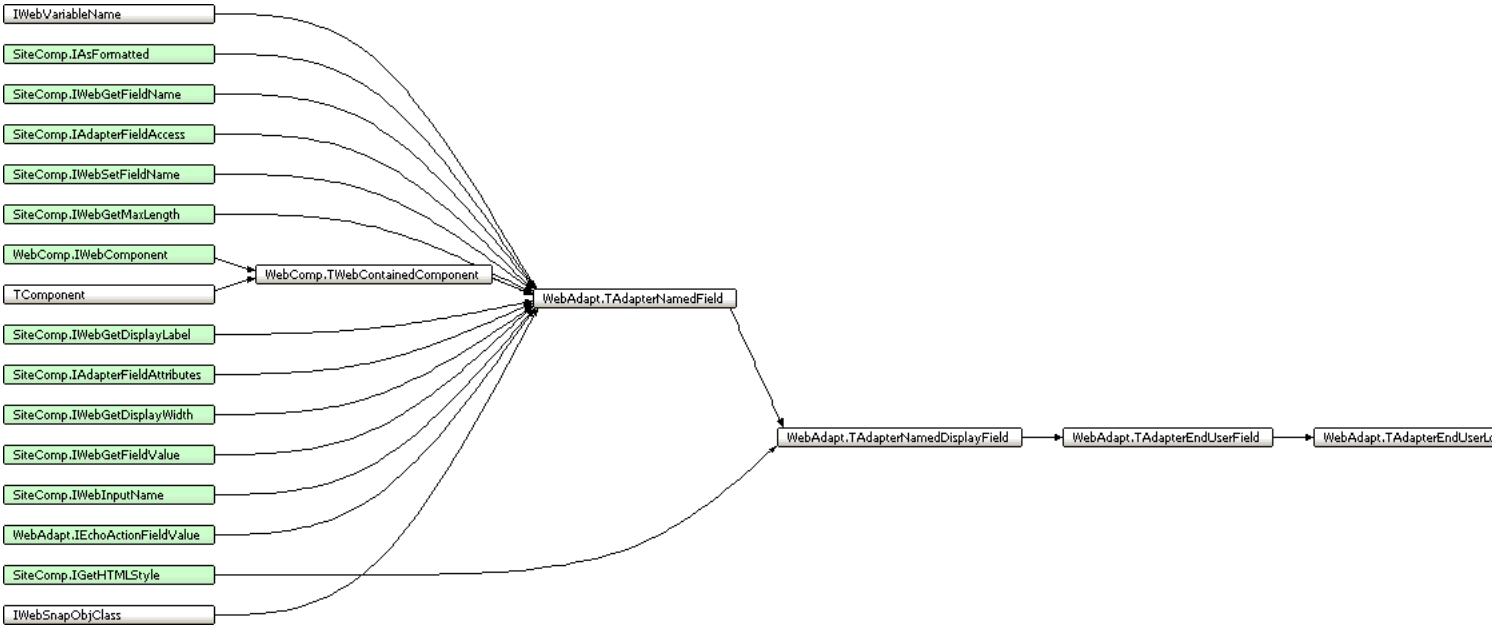
See Also

TBaseEndUserAdapter (see page 5497)

54.1.14 TAdapterEndUserLoggedInField Class

TAdapterEndUserLoggedInField is an adapter field used to access the end-user logged in flag.

Class Hierarchy



Pascal

```
TAdapterEndUserLoggedInField = class(TAdapterEndUserField);
```

C++

```
class TAdapterEndUserLoggedInField : public TAdapterEndUserField;
```

File

WebAdapt

Description

TAdapterEndUserLoggedInField is added to TApplicationAdapter (see page 5439). The logged-in flag indicates whether the user is logged-in. This value may be used by server-side script to conditionally generate HTML. For example, when the end-user is logged in, a logout hyperlink is displayed. When the end-user is not logged in, a logout hyperlink is not displayed.

See Also

TApplicationAdapter (see page 5439)

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

54.1.14.1 TAdapterEndUserLoggedInField Members

The following tables list the members exposed by TAdapterEndUserLoggedInField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5255)	Returns the name used to access this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5255)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.

	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5251)	Returns the parent adapter of this adapter field.
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.

	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.14.2 TAdapterEndUserLoggedInField Methods

The methods of the TAdapterEndUserLoggedInField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 5255)	Returns the name used to access this adapter field with server-side script.
V	ImplGetValue (see page 5255)	Returns the value associated with this adapter field.

54.1.14.2.1 TAdapterEndUserLoggedInField.GetDefaultFieldName Method

Returns the name used to access this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'LoggedIn'. This allows server-side script to access this adapter field using the parent adapter's LoggedIn property.

54.1.14.2.2 TAdapterEndUserLoggedInField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetLoggedInValue. Server-side script accesses the result of this function using the Value (

see page 5369) property of this adapter field. This value may be used by server-side script to conditionally generate HTML. For example, when the end-user is logged in, a logout hyperlink is displayed. When the end-user is not logged in, a logout hyperlink is not displayed.

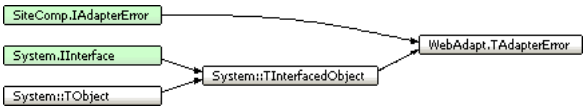
See Also

GetLoggedInValue (see page 5508)

54.1.15 TAdapterError Class

TAdapterError is the implementation of IAdapterError (see page 992).

Class Hierarchy



Pascal

```
TAdapterError = class(TInterfacedObject, IAdapterError);
```

C++

```
class TAdapterError : public TInterfacedObject, public IAdapterError;
```

File

WebAdapt

Description

TAdapterError is the implementation class for IAdapterError (see page 992).

See Also

IAdapterError (see page 992)

54.1.15.1 TAdapterError Members

The following tables list the members exposed by TAdapterError.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetErrorText (see page 5258)	Returns the value of the ErrorText property.
	GetField (see page 5258)	Returns the value of the Field property.
	GetID (see page 5259)	Returns the value of the ID property.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5258)	Create instantiates an instance of TAdapterError (see page 5256). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ErrorText (see page 994)	Returns text describing the error.
R	Field (see page 994)	Returns text describing the error.
R	ID (see page 994)	Returns error ID.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

54.1.15.2 TAdapterError.Create Constructor

Create instantiates an instance of TAdapterError (see page 5256). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.

Pascal

```
constructor Create(AErrors: TAdapterErrorsList; AIndex: Integer);
```

C++

```
__fastcall TAdapterError(TAdapterErrorsList AErrors, int AIndex);
```

Description

```
constructor Create(AErrors: TAdapterErrorsListTAdapterErrorsList; AIndex: Integer); public;
```

Create instantiates an instance of TAdapterError (see page 5256). Do not call this constructor in application code. The adapter errors list object instantiates adapter error objects as needed.

AErrors is the adapter errors list object that creates this TAdapterError (see page 5256) object.

AIndex is the index of this adapter error in the list that AErrors maintains.

54.1.15.3 TAdapterError Methods

The methods of the TAdapterError class are listed here.

Protected Methods

	Name	Description
	GetErrorText (see page 5258)	Returns the value of the ErrorText property.
	GetField (see page 5258)	Returns the value of the Field property.
	GetID (see page 5259)	Returns the value of the ID property.

54.1.15.3.1 TAdapterError.GetErrorText Method

Returns the value of the ErrorText property.

Pascal

```
function GetErrorText: string;
```

C++

```
__fastcall UnicodeString GetErrorText();
```

Description

GetErrorText is the read implementation of the ErrorText property.

See Also

ErrorText (see page 994)

54.1.15.3.2 TAdapterError.GetField Method

Returns the value of the Field property.

Pascal

```
function GetField: TComponent;
```

C++

```
__fastcall TComponent * GetField();
```

Description

GetField is the read implementation of the Field property.

See Also

Field (see page 994)

54.1.15.3.3 TAdapterError.GetID Method

Returns the value of the ID property.

Pascal

```
function GetID: integer;
```

C++

```
__fastcall int GetID();
```

Description

GetID is the read implementation of the ID property.

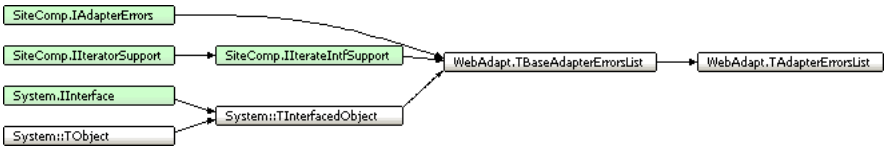
See Also

ID (see page 994)

54.1.16 TAdapterErrorsList Class

TAdapterErrorsList keeps track of errors that an adapter encounters.

Class Hierarchy



Pascal

```
TAdapterErrorsList = class(TBaseAdapterErrorsList);
```

C++

```
class TAdapterErrorsList : public TBaseAdapterErrorsList;
```

File

WebAdapt

Description

TAdapterErrorsList acts as a repository for all the errors an adapter (or any of its subcomponents) encounters while processing an HTTP request message. Adapter components create an instance of this class to implement their Errors (see page 5487) property. Server-side scripts can use the Errors (see page 5487) property to access the errors in this list. The usual purpose of such a script is to display error messages to the end user.

To add new error objects to the list, call the AddError (see page 5263) method. To access the errors in the list, read its Errors (see page 5487) property. Errors (see page 5487) lists the IAdapterError (see page 992) interfaces of adapter error objects (TAdapterError (see page 5256)) that represent each error in the list. You can also access the individual properties of those error objects using the FieldNames (see page 5266), IDs (see page 5266), and Messages (see page 5266) properties of the TAdapterErrorsList instance.

TAdapterErrorsList implements the IAdapterErrorsList interface.

Note: Do not confuse TAdapterErrorsList with TAdapterErrorList. TAdapterErrorsList stores the errors that the WebSnap application encounters when using an adapter. TAdapterErrorList is a display component that displays the errors from a TAdapterErrorsList object.

See Also

TAdapterErrorList (see page 6397)

TCustomAdapter (see page 5531)

TAdapterError (see page 5256)

IAdapterErrors (see page 995)

IAdapterError (see page 992)

54.1.16.1 TAdapterErrorsList Members

The following tables list the members exposed by TAdapterErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	EndIterator (see page 5483)	Signals the end of the iterating operation.
	GetCurrentIntf (see page 5483)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5484)	Returns a particular error.
	GetErrorCount (see page 5484)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5484)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5484)	Indicates whether the errors list contains objects associated with a particular adapter field.
	ImplClearErrors (see page 5264)	This is ImplClearErrors, a member of class TAdapterErrorsList.
	ImplDefineLabel (see page 5264)	This is ImplDefineLabel, a member of class TAdapterErrorsList.
	ImplGetError (see page 5264)	This is ImplGetError, a member of class TAdapterErrorsList.
	ImplGetErrorCount (see page 5265)	This is ImplGetErrorCount, a member of class TAdapterErrorsList.
	ImplGetObjectErrors (see page 5265)	This is ImplGetObjectErrors, a member of class TAdapterErrorsList.
	ImplHasObjectErrors (see page 5265)	This is ImplHasObjectErrors, a member of class TAdapterErrorsList.
	NextIteration (see page 5486)	Advances to the objects next item.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5486)	Positions the object on its first item and starts the iteration process.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddError (see page 5263)	This is the overview for the AddError method overload.
	AddError (see page 5481)	This is the overview for the AddError method overload.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 996)	Clears the errors list.
	ClearErrors (see page 5482)	Clear error list.
	Create (see page 5262)	Creates and initializes an instance of TAdapterErrorsList (see page 5259).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	DefineLabel (see page 996)	Associates a label with all errors associated with a particular adapter field.
	DefineLabel (see page 5483)	Define error label for field.
	Destroy (see page 5262)	Frees an instance of TAdapterErrorsList (see page 5259).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetCurrentIntf (see page 1051)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	GetError (see page 996)	Returns a particular error.
	GetErrorCount (see page 997)	Return the number of errors in the errors list.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetObjectErrors (see page 997)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 997)	Indicates whether the errors list contains objects associated with a particular adapter field.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 1056)	Advances to the objects next item.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	ErrorCount (see page 998)	Returns the number of errors in the errors list.
	ErrorCount (see page 5487)	Number of error objects contained.
	Errors (see page 998)	Returns an error from the list.
	Errors (see page 5487)	Access to error objects.
	FieldNames (see page 5266)	Lists the names of the objects that encountered the errors in the list.
	IDs (see page 5266)	Lists the error codes of the errors in the list.
	Messages (see page 5266)	Lists the error messages for the errors in the list.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

54.1.16.2 TAdapterErrorsList.Create Constructor

Creates and initializes an instance of TAdapterErrorsList (see page 5259).

Pascal

```
constructor Create(AAdapter: TObject);
```

C++

```
__fastcall TAdapterErrorsList(TObject * AAdapter);
```

Description

Typically, WebSnap applications do not create their own instances of TAdapterErrorsList (see page 5259). Adapter objects instantiate this class to implement their Errors (see page 5487) property.

AAdapter is the adapter component that uses this object to implement its Errors (see page 5487) property.

See Also

Errors (see page 5573)

54.1.16.3 TAdapterErrorsList.Destroy Destructor

Frees an instance of TAdapterErrorsList (see page 5259).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterErrorsList();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the errors list is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

54.1.16.4 TAdapterErrorsList Methods

The methods of the TAdapterErrorsList class are listed here.

Protected Methods

	Name	Description
	ImplClearErrors (see page 5264)	This is ImplClearErrors, a member of class TAdapterErrorsList.
	ImplDefineLabel (see page 5264)	This is ImplDefineLabel, a member of class TAdapterErrorsList.
	ImplGetError (see page 5264)	This is ImplGetError, a member of class TAdapterErrorsList.
	ImplGetErrorCount (see page 5265)	This is ImplGetErrorCount, a member of class TAdapterErrorsList.
	ImplGetObjectErrors (see page 5265)	This is ImplGetObjectErrors, a member of class TAdapterErrorsList.
	ImplHasObjectErrors (see page 5265)	This is ImplHasObjectErrors, a member of class TAdapterErrorsList.

Public Methods

	Name	Description
	AddError (see page 5263)	This is the overview for the AddError method overload.

54.1.16.4.1 AddError Method

This is the overview for the AddError method overload.

Overload List

	Name	Description
	TAdapterErrorsList.AddError (Exception, string) (see page 5263)	Adds an error to the list.
	TAdapterErrorsList.AddError (Exception, TObject) (see page 5263)	Adds an error to the list.
	TAdapterErrorsList.AddError (string, string, Integer) (see page 5263)	Adds an error to the list.
	TAdapterErrorsList.AddError (string, TObject, Integer) (see page 5263)	Adds an error to the list.

54.1.16.4.1.1 TAdapterErrorsList.AddError Method (Exception, TObject)

Adds an error to the list.

Pascal

```
procedure AddError(AException: Exception; const AObjectName: string = ''); override;
procedure AddError(const AMessage: string; const AObjectName: string = ''; AID: Integer = 0);
override;
procedure AddError(const AMessage: string; AObject: TObject; AID: Integer = 0); override;
procedure AddError(AException: Exception; AObject: TObject); override;
```

C++

```
virtual __fastcall AddError(Exception AException, const UnicodeString AObjectName = '');
virtual __fastcall AddError(const UnicodeString AMessage, const UnicodeString AObjectName =
'', int AID = 0);
virtual __fastcall AddError(const UnicodeString AMessage, TObject * AObject, int AID = 0);
virtual __fastcall AddError(Exception AException, TObject * AObject);
```

Description

AddError adds an error to the object's internal list. An error is described by three pieces of information:

An error message string. You can specify this string directly with the AMessage parameter, or using an Exception (see page 3949) object (with the AException parameter). If you specify an Exception (see page 3949) object, the object's Message property becomes the message string.

The name of an object (Variable) related to the error. You can specify this value as a string (with the AObjectName parameter) or as an object reference (with the AObject parameter). If you omit both these parameters, but supply an EAdapterFieldException (see page 5178) object as the AException parameter, TAdapterErrorsList (see page 5259) uses the FieldName property of that exception object. If you do not supply an AObject parameter, an AObjectName parameter, or an EAdapterFieldException (see page 5178) object as the AException parameter, TAdapterErrorsList (see page 5259) uses an empty string.

An error ID value. You can specify this value using the AID parameter. If you do not supply a value for AID but specify an EAdapterException (see page 5177) object as the AException parameter, TAdapterErrorsList (see page 5259) uses the ID property of that exception object. If you do not supply an AID parameter or an EAdapterException (see page 5177) object, TAdapterErrorsList (see page 5259) uses 0.

See Also

FieldName (see page 5180)

ID (see page 5178)

Messages (see page 5266)

FieldNames (see page 5266)

IDs (see page 5266)

54.1.16.4.2 TAdapterErrorsList.ImplClearErrors Method

Pascal

```
procedure ImplClearErrors; override;
```

C++

```
virtual __fastcall ImplClearErrors();
```

Description

This is ImplClearErrors, a member of class TAdapterErrorsList.

54.1.16.4.3 TAdapterErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); override;
```

C++

```
virtual __fastcall ImplDefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

This is ImplDefineLabel, a member of class TAdapterErrorsList.

54.1.16.4.4 TAdapterErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; override;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TAdapterErrorsList.

54.1.16.4.5 TAdapterErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

Description

This is ImplGetErrorCount, a member of class TAdapterErrorsList.

54.1.16.4.6 TAdapterErrorsList.ImplGetObjectErrors Method

Pascal

```
function ImplGetObjectErrors(const AName: string): TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetObjectErrors(const UnicodeString AName);
```

Description

This is ImplGetObjectErrors, a member of class TAdapterErrorsList.

54.1.16.4.7 TAdapterErrorsList.ImplHasObjectErrors Method

Pascal

```
function ImplHasObjectErrors(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHasObjectErrors(const UnicodeString AName);
```

Description

This is ImplHasObjectErrors, a member of class TAdapterErrorsList.

54.1.16.5 TAdapterErrorsList Properties

The properties of the TAdapterErrorsList class are listed here.

Public Properties

	Name	Description
	FieldNames (see page 5266)	Lists the names of the objects that encountered the errors in the list.
	IDs (see page 5266)	Lists the error codes of the errors in the list.
	Messages (see page 5266)	Lists the error messages for the errors in the list.

54.1.16.5.1 TAdapterErrorsList.FieldNames Property

Lists the names of the objects that encountered the errors in the list.

Pascal

```
property FieldNames [I: Integer]: string;
```

C++

```
__property UnicodeString FieldNames[int I];
```

Description

Use FieldNames to access the name of the object associated with a specified error. FieldNames values typically reflect the name of an adapter field that was the cause of the error. However, they can be any object reference for which there is a defined Variable that can be accessed from server-side script. If there is no object associated with an error, the value of FieldNames is an empty string.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of FieldNames for an error is set by the AddError (see page 5263) method when the error is added to the list.

See Also

AddError (see page 5263)

Field (see page 994)

FieldName (see page 5180)

54.1.16.5.2 TAdapterErrorsList.IDs Property

Lists the error codes of the errors in the list.

Pascal

```
property IDs [I: Integer]: integer;
```

C++

```
__property int IDs[int I];
```

Description

Use IDs to access the ID (error code) of a specified error in the list. If the error has no error code, the value of IDs is 0.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of IDs for an error is set by the AddError (see page 5263) method when the error is added to the list.

See Also

AddError (see page 5263)

ID (see page 994)

ID (see page 5178)

54.1.16.5.3 TAdapterErrorsList.Messages Property

Lists the error messages for the errors in the list.

Pascal

```
property Messages [I: Integer]: string;
```

C++

```
__property UnicodeString Messages[int I];
```

Description

Use Messages to access the error message for a specified error in the list.

I is the index of the error in the list, where 0 specifies the first error, 1 specifies the second error, and so on.

The value of Messages for an error is set by the AddError (see page 5263) method when the error is added to the list.

See Also

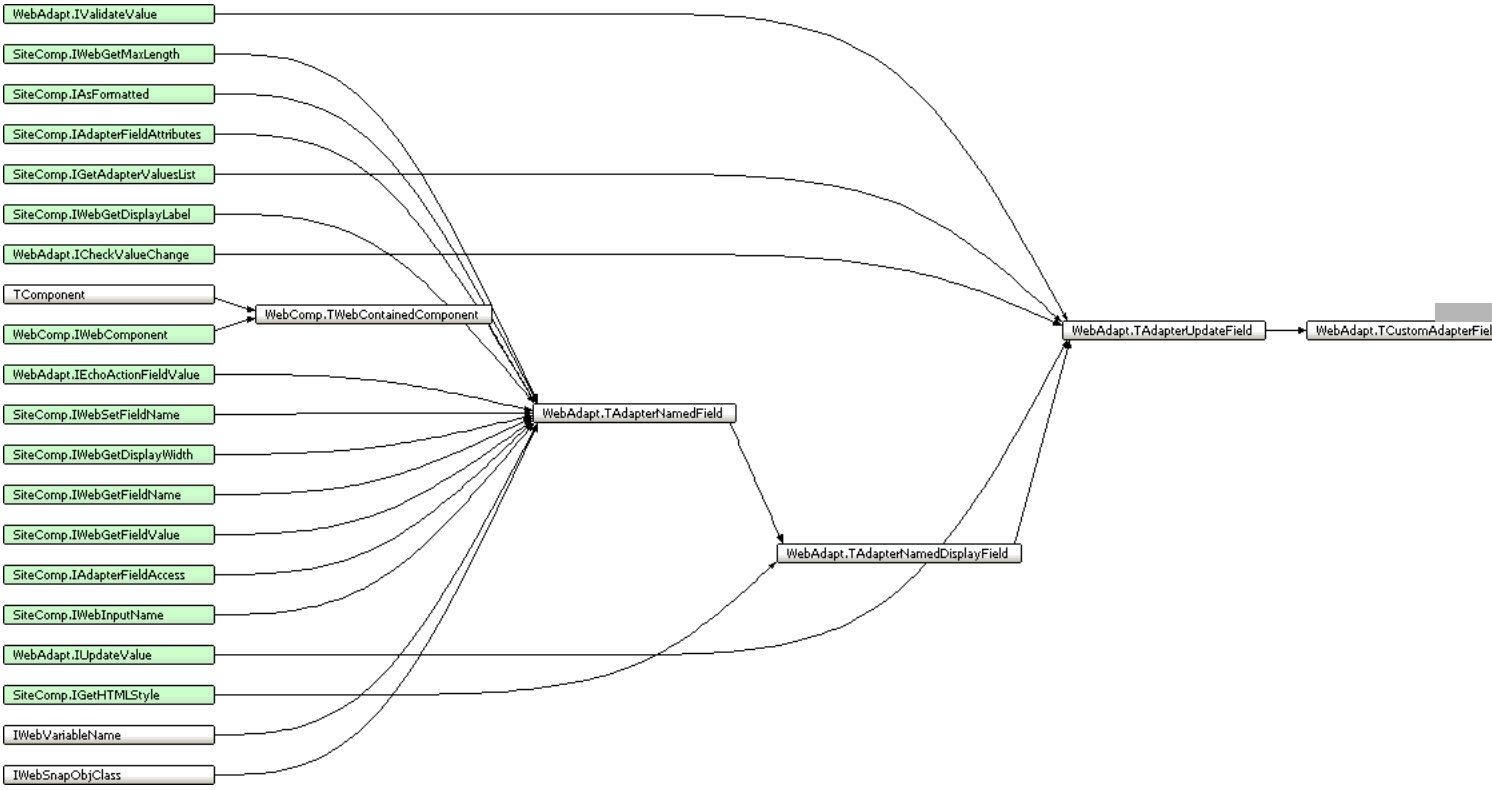
AddError (see page 5263)

ErrorText (see page 994)

54.1.17 TAdapterField Class

TAdapterField is an event handler based adapter field that provides a single value.

Class Hierarchy



Pascal

```
TAdapterField = class(TCustomAdapterField);
```

C++

```
class TAdapterField : public TCustomAdapterField;
```

File

WebAdapt

Description

TAdapterField publishes the properties in TCustomAdapterField (see page 5597). TCustomAdapterField (see page 5597) calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

Add TAdapterField to TAdapter (see page 5190) or other adapter components types when you need an adapter field that supports a single value using event handlers. Event handlers are used to provide the adapter field value(s), validate a value entered by the end-user, and update the adapter field with a value entered by the end-user.

There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. For example, TAdapterMultiValue field is used when the end-user can enter multiple values using a group of check boxes or a multiple selection select list. TAdapterFileField (see page 5281), TAdapterMemoField (see page 5323), TAdapterBooleanField (see page 5223) and TAdapterImageField (see page 5308) are other event driven adapter fields.

See Also

TAdapterUpdateField (see page 5419)

TCustomAdapterField (see page 5597)

TAdapter (see page 5190)

TAdapterMultiValueField (see page 5332)

TAdapterMemoField (see page 5323)

TAdapterFileField (see page 5281)

TAdapterBooleanField (see page 5223)

TAdapterImageField (see page 5308)

54.1.17.1 TAdapterField Members

The following tables list the members exposed by TAdapterField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5602)	Validates a single value entered by the end-user.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.

	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
V	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetValue (see page 5602)	Returns the value of this adapter field as a variant.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.

	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5603)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.

	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetDisplayText (see page 5275)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5275)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5275)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5276)	Event handler called to validate value(s) entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5272)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5272)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5272)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5273)	Identifies the rights required to update this field.
	ReadOnly (see page 5273)	Indicates whether this adapter field is read-only.
	Required (see page 5273)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5274)	Associates a list of values with the adapter field.
	ViewAccess (see page 5274)	Identifies the rights required to view this field.

54.1.17.2 TAdapterField Properties

The properties of the TAdapterField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5272)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5272)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5272)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5273)	Identifies the rights required to update this field.
	ReadOnly (see page 5273)	Indicates whether this adapter field is read-only.
	Required (see page 5273)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5274)	Associates a list of values with the adapter field.
	ViewAccess (see page 5274)	Identifies the rights required to view this field.

54.1.17.2.1 TAdapterField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.17.2.2 TAdapterField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.17.2.3 TAdapterField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.17.2.4 TAdapterField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

54.1.17.2.5 TAdapterField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.17.2.6 TAdapterField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5359)

54.1.17.2.7 TAdapterField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1072)

TStringsValuesList (see page 5857)

TDataSetValuesList

54.1.17.2.8 TAdapterField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

54.1.17.3 TAdapterField Events

The events of the TAdapterField class are listed here.

Published Events

	Name	Description
	OnGetDisplayText (see page 5275)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5275)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5275)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5276)	Event handler called to validate value(s) entered by the end-user.

54.1.17.3.1 TAdapterField.OnGetDisplayText Event

Event handler called to retrieve a string representation of the adapter field's value.

Pascal

```
property OnGetDisplayText: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

The OnGetDisplayText event handler is called to retrieve a string representation an adapter field's value appropriate for a read-only display. If OnGetDisplayText is nil (Delphi) or NULL (C++), then the value of the adapter field will be used (after conversion to a string).

See Also

ImplAsFormatted (see page 5427)

54.1.17.3.2 TAdapterField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetValueEvent;
```

C++

```
__property TAdapterFieldGetValueEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5602)

54.1.17.3.3 TAdapterField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldValueEvent;
```

C++

```
__property TAdapterFieldValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

54.1.17.3.4 TAdapterField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

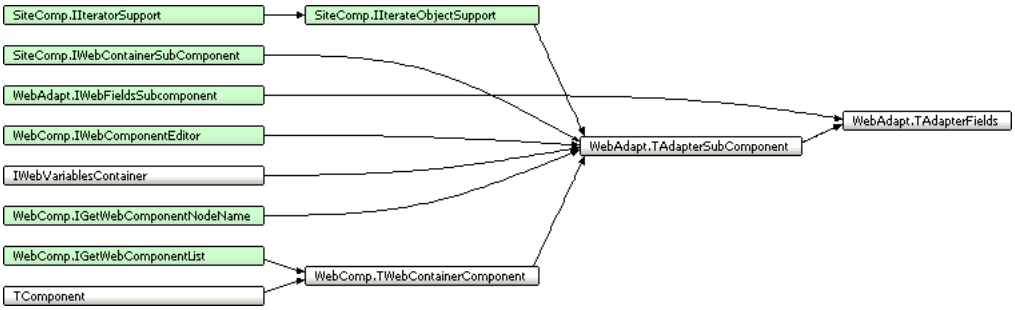
DoOnValidateValue (see page 5426)

IActionFieldValue

54.1.18 TAdapterFields Class

TAdapterFields manages the list of fields that belong to an adapter.

Class Hierarchy



Pascal

```
TAdapterFields = class(TAdapterSubComponent, IWebFieldsSubcomponent);
```

C++

```
class TAdapterFields : public TAdapterSubComponent, public IWebFieldsSubcomponent;
```

File

WebAdapt

Description

TAdapterFields is the type of the Data property on an adapter component. It manages a list that contains all the fields that belong to the adapter. The field objects (TAdapterNamedField (see page 5351) descendants) are available using the WebFields (see page 5281) property. TAdapterFields also includes support for associating the fields with variable names so that server-side script can use the field values.

TAdapterFields represents the list of explicitly assigned fields. Adapters that include a set of default fields use a TAdapterDefaultFields (see page 5237) object instead.

See Also

TCustomAdapter (see page 5531)

TAdapterActions (see page 5209)

TAdapterDefaultActions (see page 5231)

TAdapterDefaultFields (see page 5237)

TAdapterNamedField (see page 5351)

54.1.18.1 TAdapterFields Members

The following tables list the members exposed by TAdapterFields.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5278)	Indicates whether the associated adapter supports a specified class of field component.
	CanAddClass (see page 5408)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5409)	Stops the TAdapterSubComponent (see page 5406) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 5409)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5410)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5410)	Returns the Web module that contains a specified item from this list.
V	GetChildren (see page 6065)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5411)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5411)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
V	GetDefaultWebComponents (see page 6065)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5412)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5413)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5413)	Returns the value of the VariablesInUse (see page 5419) property.
	GetWebComponentNodeName (see page 5413)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6065)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5279)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
V	ImplGetDefaultComponentList (see page 6066)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5280)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	NextIteration (see page 5415)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5416)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6066)	Positions a child component in the list of child Web items.

	SetDefaultWebComponents (see page 5416)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetDefaultWebComponents (see page 6067)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetWebParent (see page 5280)	Associates this component with an adapter.
	StartIterator (see page 5418)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
	Adapter (see page 5418)	Indicates the adapter that uses this object to manage its fields or actions.
	VariablesInUse (see page 5419)	Lists those fields or actions in the collection for which variables are defined.
	WebContainerComponentHelper (see page 6067)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.
	Create (see page 6064)	Creates and initializes an instance of TWebContainerComponent (see page 6063).
	Destroy (see page 6064)	Frees an instance of TWebContainerComponent (see page 6063).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetParentComponent (see page 5412)	Returns the associated adapter.
	GetWebComponentNodeName (see page 6083)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.
	HasParent (see page 5414)	Indicates whether this component has an associated adapter.
	NextIteration (see page 1056)	Advances to the objects next item.
	ReadState (see page 5416)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 5417)	Sets the Adapter (see page 5418) property to be the same as this component's parent.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	WebFields (see page 5281)	Lists the field components that this object manages.

54.1.18.2 TAdapterFields Methods

The methods of the TAdapterFields class are listed here.

Protected Methods

	Name	Description
	AdapterCanAddClass (see page 5278)	Indicates whether the associated adapter supports a specified class of field component.
	ImplCanAddClass (see page 5279)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
	ImplGetWebComponentNodeName (see page 5280)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	SetWebParent (see page 5280)	Associates this component with an adapter.

54.1.18.2.1 TAdapterFields.AdapterCanAddClass Method

Indicates whether the associated adapter supports a specified class of field component.

Pascal

```
function AdapterCanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean AdapterCanAddClass(TComponent * AParent, TClass AClass);
```

Description

TAdapterActions (see page 5209) uses this method internally as part of its implementation for the CanAddClass (see page 5408) method. This allows the TAdapterFields (see page 5276) object to indicate, at design time, which types of components are valid as members of the WebFields (see page 5281) property list.

AParent is the instance of TAdapterFields (see page 5276) to which the child fields are to be added.

AClass describes the class of the child component to be added to AParent.

AdapterCanAddClass returns true if the adapter recognizes and supports AClass as a child field, false otherwise.

AdapterCanAddClass calls the CanAddFieldClass method of its associated adapter to determine whether it accepts the specified field class.

See Also

WebFields (see page 5281)

Adapter (see page 5418)

ImplCanAddClass (see page 5279)

CanAddFieldClass (see page 5540)

54.1.18.2.2 TAdapterFields.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5408) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 5408) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterFields (see page 5276) object to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterFields (see page 5276) object can include a child of type AClass, false otherwise.

ImplCanAddClass returns true if one of the following two conditions is met:

AClass is an adapter field class that the associated adapter recognizes and accepts.

AClass is a data entry field (a descendant of TAdapterUpdateField (see page 5419)).

See Also

CanAddClass (see page 5408)

AdapterCanAddClass (see page 5278)

TAdapterUpdateField (see page 5419)

54.1.18.2.3 TAdapterFields.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.

Pascal

```
function ImplGetWebComponentNodeName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 5413) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the string 'Fields'.

See Also

GetWebComponentNodeName (see page 5413)

54.1.18.2.4 TAdapterFields.SetWebParent Method

Associates this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); override;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

SetWebParent sets up the association between this component and its parent adapter.

Value is the adapter component that acts as a parent to this collection.

SetWebParent assigns Value to the protected Adapter (see page 5418) property and replaces that adapter's Data property with this TAdapterFields (see page 5276) instance.

See Also

Data (see page 5572)

Adapter (see page 5418)

54.1.18.3 TAdapterFields Properties

The properties of the TAdapterFields class are listed here.

Public Properties

	Name	Description
	WebFields (see page 5281)	Lists the field components that this object manages.

54.1.18.3.1 TAdapterFields.WebFields Property

Lists the field components that this object manages.

Pascal

```
property WebFields: TWebComponentList;
```

C++

```
__property TWebComponentList WebFields;
```

Description

Read (see page 3634) WebFields to access the list of field components. Each component in this list is a descendant of TAdapterNamedField (see page 5351).

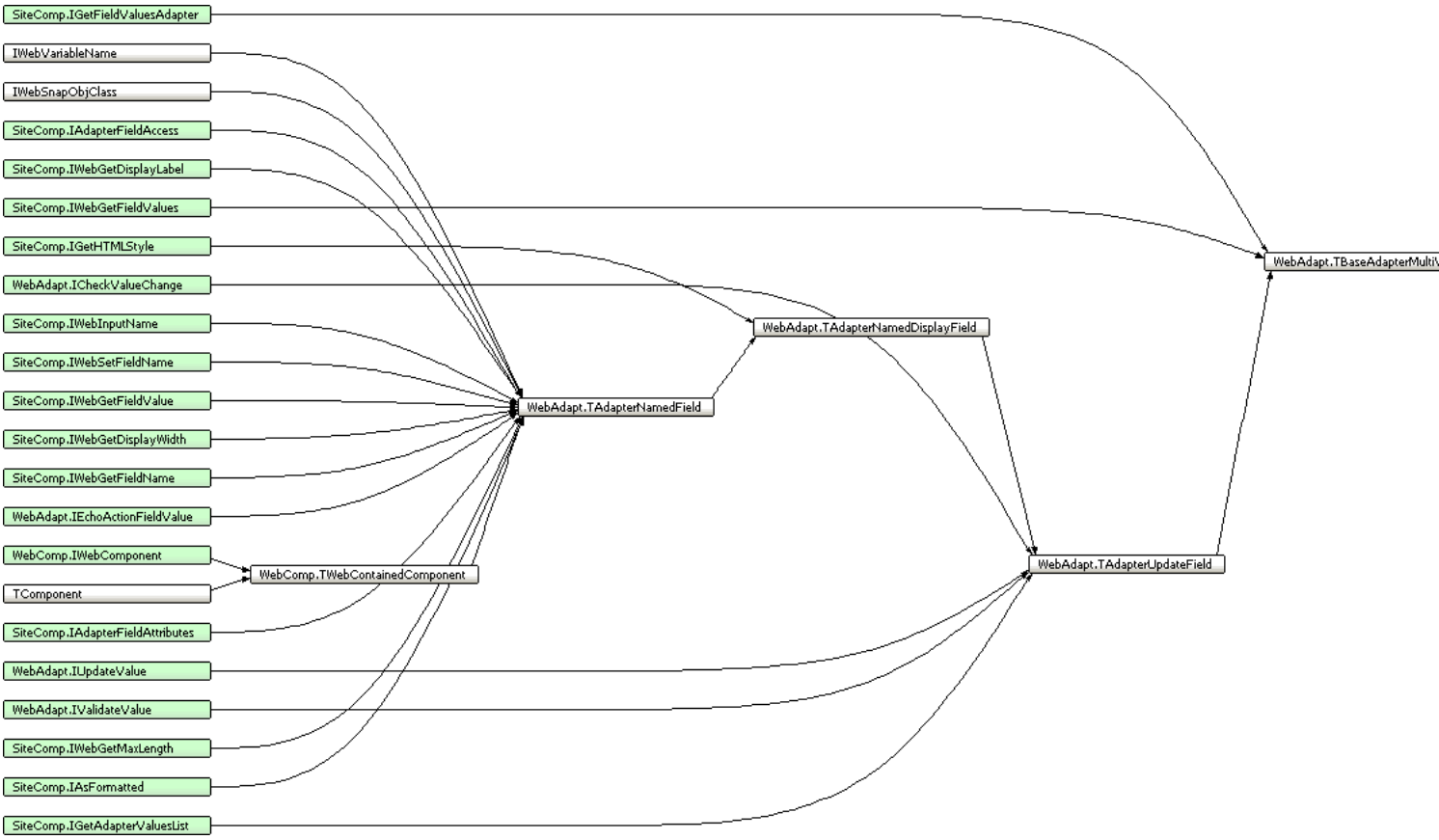
See Also

TAdapterNamedField (see page 5351)

54.1.19 TAdapterFileField Class

TAdapterFileField is an event handler based adapter field that can accept uploaded files.

Class Hierarchy



Pascal

```
TAdapterFileField = class(TCustomAdapterFileField);
```

C++

```
class TAdapterFileField : public TCustomAdapterFileField;
```

File

WebAdapt

Description

TAdapterFileField publishes the properties in TCustomAdapterFileField (see page 5604). TCustomAdapterFileField (see page 5604) calls event handlers to retrieve the adapter field's filenames, validate files uploaded by the end-user, and update the adapter field's files.

Add TAdapterFileField to TAdapter (see page 5190) or other adapter components types when you need an adapter field that supports files using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5267), TAdapterMultiValueField (see page 5332), TAdapterMemoField (see page 5323), TAdapterBooleanField (see page 5223) and TAdapterImageField (see page 5308).

See Also

TCustomAdapterFileField (see page 5604)

TAdapter (see page 5190)

TAdapterField (see page 5267)

TAdapterMemoField (see page 5323)

TAdapterMultiValueField (see page 5332)

TAdapterBooleanField (see page 5223)

TAdapterImageField (see page 5308)

54.1.19.1 TAdapterFileField Members

The following tables list the members exposed by TAdapterFileField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5609)	Validates files uploaded by the end-user.
	EndValueIterator (see page 5493)	Completes iteration.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5494)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5609)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5494)	Returns the adapter object that implements the Values property.
	GetFileNames (see page 5610)	Returns the count of files associated with this adapter field.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5610)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
V	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
V	GetValueCount (see page 5494)	Returns the number of values associated with this adapter field.

	GetValueCount (see page 5610)	Returns the number of values associated with this adapter field.
	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetValues (see page 5495)	Returns one of the values associated with the adapter field.
	GetValues (see page 5611)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' HasValue method.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplCheckValueChange (see page 5611)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplGetCurrentValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue (see page 5494) method.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 5496)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 5496)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5611)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5496)	Implements IWebGetFieldValues (see page 1092)' NextValueIteration method.

	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartValueIterator (see page 5497)	Implements IWebGetFieldValues (see page 1092)' StartValueIterator method.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 5492)	Constructor for TBaseAdapterMultiValueField (see page 5488).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 5492)	Destructor for TBaseAdapterMultiValueField (see page 5488).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndValueIterator (see page 1093)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetCurrentValue (see page 1094)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 1031)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.

	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1094)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1094)	Position the iteration pointer on the next value.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartValueIterator (see page 1095)	Reset (see page 3643) the iteration pointer to the first item.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FileCount (see page 5612)	Returns one of the filenames associated with this adapter field.
R	FileNames (see page 5612)	Returns a file associated with this adapter field.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetFileCount (see page 5289)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5290)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5290)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5290)	Validates values entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5287)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5287)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5288)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5288)	Identifies the rights required to update this field.
	ReadOnly (see page 5288)	Indicates whether this adapter field is read-only.
	Required (see page 5289)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5289)	Identifies the rights required to view this field.

54.1.19.2 TAdapterFileField Properties

The properties of the TAdapterFileField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5287)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5287)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5288)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5288)	Identifies the rights required to update this field.
	ReadOnly (see page 5288)	Indicates whether this adapter field is read-only.
	Required (see page 5289)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5289)	Identifies the rights required to view this field.

54.1.19.2.1 TAdapterFileField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.19.2.2 TAdapterFileField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.19.2.3 TAdapterFileField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.19.2.4 TAdapterFileField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

54.1.19.2.5 TAdapterFileField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.19.2.6 TAdapterFileField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5359)

54.1.19.2.7 TAdapterFileField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

54.1.19.3 TAdapterFileField Events

The events of the TAdapterFileField class are listed here.

Published Events

	Name	Description
	OnGetFileCount (see page 5289)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5290)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5290)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5290)	Validates values entered by the end-user.

54.1.19.3.1 TAdapterFileField.OnGetFileCount Event

Returns the number of values associated with this adapter field.

Pascal

```
property OnGetFileCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetFileCount;
```

Description

Write an OnGetFileCount event handler to indicate the number of filenames associated with this adapter field.

See Also

GetValueCount (see page 5610)

54.1.19.3.2 TAdapterFileField.OnGetFileNames Event

Returns one of the filenames associated with the adapter field.

Pascal

```
property OnGetFileNames: TAdapterFieldGetMultiValueStringEvent;
```

C++

```
__property TAdapterFieldGetMultiValueStringEvent OnGetFileNames;
```

Description

Write an OnGetFileNames event handler to provide the names of files associated with this adapter field.

See Also

GetFileNames (see page 5610)

54.1.19.3.3 TAdapterFileField.OnUploadFiles Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUploadFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnUploadFiles;
```

Description

OnUploadFiles is called by ImplUpdateValue. Write an OnUploadFiles event handler to accept files uploaded by the end-user. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5611)

TUpdateFileList (see page 5864)

54.1.19.3.4 TAdapterFileField.OnValidateFiles Event

Validates values entered by the end-user.

Pascal

```
property OnValidateFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnValidateFiles;
```

Description

OnValidateFiles is called by DoOnValidateValue. Implement this event handler to validate files uploaded by the end-user. Raise an exception to indicate a validation error.

See Also

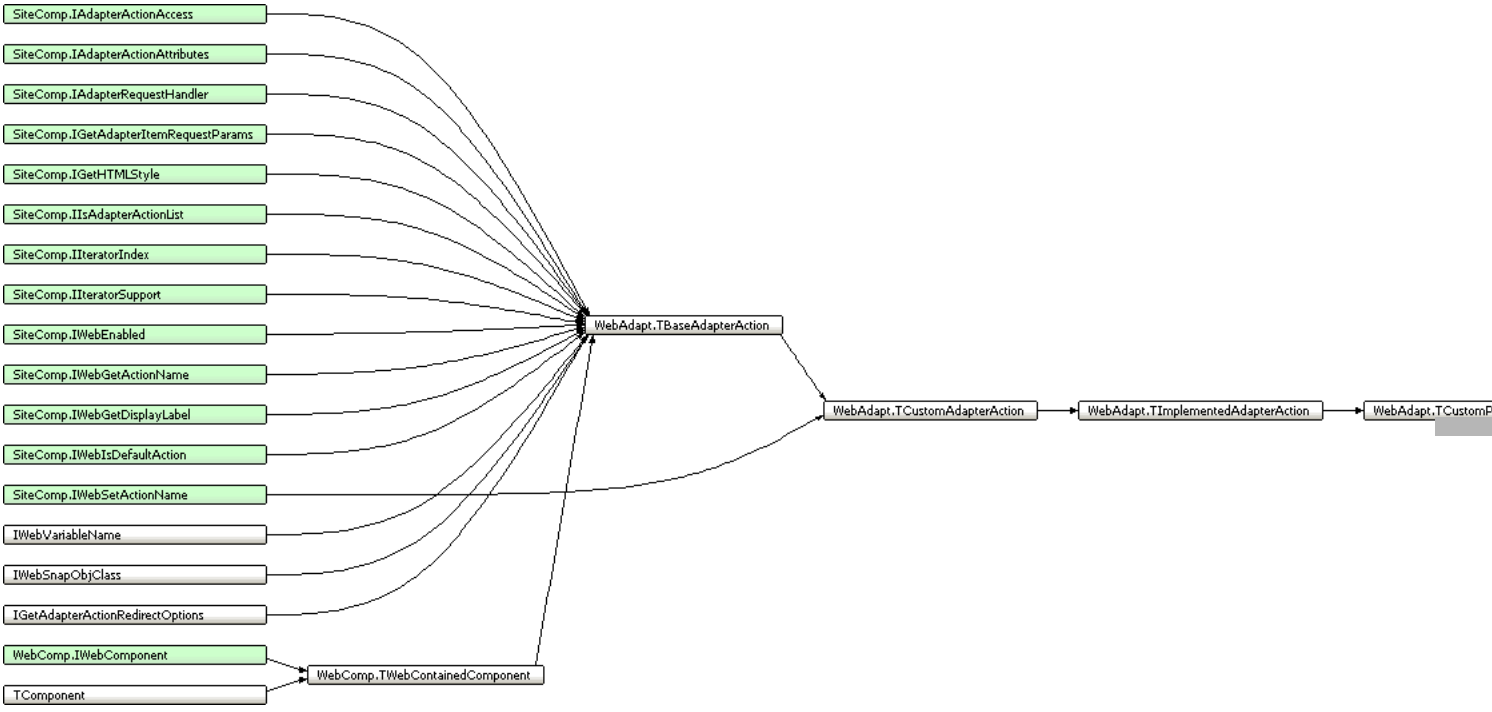
DoOnValidateValue (see page 5609)

TUpdateFileList (see page 5864)

54.1.20 TAdapterGotoPageAction Class

TAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5747).

Class Hierarchy



Pascal

```
TAdapterGotoPageAction = class(TCustomAdapterGotoPageAction);
```

C++

```
class TAdapterGotoPageAction : public TCustomAdapterGotoPageAction;
```

File

WebAdapt

Description

TAdapterGotoPageAction is an adapter action that has a list of commands. Each command has a parameter that identifies a page in the parent adapter. When executed, TAdapterGotoPageAction changes the current page in the parent adapter. TAdapterGotoPageAction publishes the properties of TCustomAdapterGotoPageAction (see page 5614). Add TAdapterGotoPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 5747) such as TPagedAdapter (see page 5844).

See Also

TCustomAdapterGotoPageAction (see page 5614)

TDefaultFieldsPagedAdapter (see page 5747)

TPagedAdapter (see page 5844)

TAdapterNextPageAction (see page 5370)

TAdapterPrevPageAction (see page 5396)

54.1.20.1 TAdapterGotoPageAction Members

The following tables list the members exposed by TAdapterGotoPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5727)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultActionName (see page 5619)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 5619)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
V	GetItemCount (see page 5620)	Returns the number of commands defined by this adapter action.

	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5620)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5620)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
V	ImplGetDisplayLabel (see page 5621)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIsAdapterActionList (see page 5621)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
V	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
V	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	ImplIsDefaultAction (see page 5622)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
V	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	ImplWebEnabled (see page 5622)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.

	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
V	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5474)	Provides access to the parent adapter component.
R	Adapter (see page 5727)	Returns the parent adapter of this adapter action.
R	Errors (see page 5475)	Returns the parent adapter's Errors list.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.

V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5297)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5297)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5297)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5298)	Called before the adapter action response is generated.
	OnGetPageName (see page 5298)	Provides the display labels for each goto page command.
	OnGetParams (see page 5298)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	ExecuteAccess (see page 5296)	Defines the rights required to execute this action.
	RedirectOptions (see page 5296)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.20.2 TAdapterGotoPageAction Properties

The properties of the TAdapterGotoPageAction class are listed here.

Published Properties

	Name	Description
	ExecuteAccess (see page 5296)	Defines the rights required to execute this action.
	RedirectOptions (see page 5296)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.20.2.1 TAdapterGotoPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6796)

HasExecuteAccess (see page 5552)

54.1.20.2.2 TAdapterGotoPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

54.1.20.3 TAdapterGotoPageAction Events

The events of the TAdapterGotoPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5297)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5297)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5297)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5298)	Called before the adapter action response is generated.
	OnGetPageName (see page 5298)	Provides the display labels for each goto page command.
	OnGetParams (see page 5298)	Specifies name/value pairs needed to execute this adapter action.

54.1.20.3.1 TAdapterGotoPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.20.3.2 TAdapterGotoPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.20.3.3 TAdapterGotoPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action

parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.20.3.4 TAdapterGotoPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.20.3.5 TAdapterGotoPageAction.OnGetPageName Event

Provides the display labels for each goto page command.

Pascal

```
property OnGetPageName: TWebFormGetPageNameEvent;
```

C++

```
__property TWebFormGetPageNameEvent OnGetPageName;
```

Description

Implement this event handler to specify display labels for the commands that goto pages in the parent adapter. APageIndex is the zero-based index of the page to goto.

See Also

ImplGetDisplayLabel (see page 5621)

54.1.20.3.6 TAdapterGotoPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

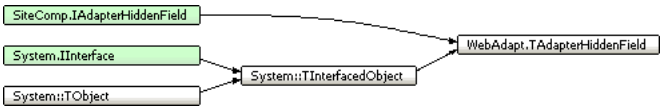
Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.21 TAdapterHiddenField Class

Class Hierarchy



Pascal

```
TAdapterHiddenField = class(TInterfacedObject, IAdapterHiddenField);
```

C++

```
class TAdapterHiddenField : public TInterfacedObject, public IAdapterHiddenField;
```

File

WebAdapt

Description

This is class WebAdapt.TAdapterHiddenField.

54.1.21.1 TAdapterHiddenField Members

The following tables list the members exposed by TAdapterHiddenField.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetName (see page 5301)	This is GetName, a member of class TAdapterHiddenField.
	GetValue (see page 5301)	This is GetValue, a member of class TAdapterHiddenField.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5301)	This is Create, a member of class TAdapterHiddenField.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Name (see page 1003)	This is Name, a member of class IAdapterHiddenField.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
R	Value (see page 1003)	This is Value, a member of class IAdapterHiddenField.

54.1.21.2 TAdapterHiddenField.Create Constructor

Pascal

```
constructor Create(AHiddenFields: TAdapterHiddenFieldsList; AIndex: Integer);
```

C++

```
__fastcall TAdapterHiddenField(TAdapterHiddenFieldsList AHiddenFields, int AIndex);
```

Description

This is Create, a member of class TAdapterHiddenField.

54.1.21.3 TAdapterHiddenField Methods

The methods of the TAdapterHiddenField class are listed here.

Protected Methods

	Name	Description
	GetName (see page 5301)	This is GetName, a member of class TAdapterHiddenField.
	GetValue (see page 5301)	This is GetValue, a member of class TAdapterHiddenField.

54.1.21.3.1 TAdapterHiddenField.GetName Method

Pascal

```
function GetName: string;
```

C++

```
__fastcall UnicodeString GetName();
```

Description

This is GetName, a member of class TAdapterHiddenField.

54.1.21.3.2 TAdapterHiddenField.GetValue Method

Pascal

```
function GetValue: string;
```

C++

```
__fastcall UnicodeString GetValue();
```

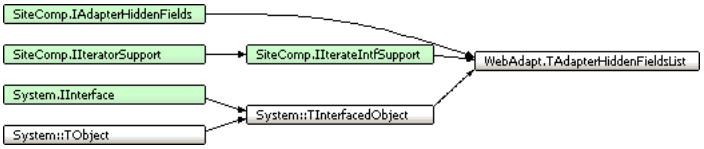
Description

This is GetValue, a member of class TAdapterHiddenField.

54.1.22 TAdapterHiddenFieldsList Class

TAdapterHiddenFieldsList maintains a collection of name/value pairs that an adapter includes as hidden fields in an HTML form.

Class Hierarchy



Pascal

```
TAdapterHiddenFieldsList = class(TInterfacedObject, IAdapterHiddenFields, IIterateIntfSupport);
```

C++

```
class TAdapterHiddenFieldsList : public TInterfacedObject, public IAdapterHiddenFields, public
IIterateIntfSupport;
```

File

WebAdapt

Description

Adapter components create instances of TAdapterHiddenFieldsList to store the name/value pairs they include as hidden fields in HTML forms. Each entry in the list has a field name and a value for that field. The adapter uses the methods of TAdapterHiddenFieldsList to add name/value pairs to the list, based on its hidden field options. When an adapter page producer generates a form using the adapter, it includes hidden fields for every name/value pair in the list. Those hidden fields are then included in any request that originates from the resulting HTML form, so that the adapter and its subcomponents can access those values to respond to the request.

For example, dataset adapters that have a Mode property that indicates whether the form is used for read-only browsing, inserting a new record, editing an existing record, or generating a query. The adapter includes the current mode as a hidden field. When the WebSnap application receives a message from the HTML form with that hidden field, the adapter can reestablish the correct mode for processing the request.

See Also

TCustomAdapter (see page 5531)

TAdapterPageProducer (see page 1136)

TCustomDataSetAdapter

54.1.22.1 TAdapterHiddenFieldsList Members

The following tables list the members exposed by TAdapterHiddenFieldsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	EndIterator (see page 5305)	Signals the end of the iterating operation.
	GetCurrentIntf (see page 5306)	This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.

	GetField (see page 5306)	This is GetField, a member of class TAdapterHiddenFieldsList.
	GetFieldCount (see page 5306)	This is GetFieldCount, a member of class TAdapterHiddenFieldsList.
	NextIteration (see page 5307)	Advances to the objects next item.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5307)	Positions the object on its first item and starts the iteration process.

Protected Properties

	Name	Description
R	FieldCount (see page 5308)	This is FieldCount, a member of class TAdapterHiddenFieldsList.
R	Fields (see page 5308)	This is Fields, a member of class TAdapterHiddenFieldsList.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	Add (see page 5305)	Adds a name/value pair to the list.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 5305)	Clears the list of all the name/value pairs it currently holds.
	Create (see page 5304)	Creates and initializes a TAdapterHiddenFieldsList (see page 5301) object.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 5304)	Frees an instance of TAdapterHiddenFieldsList (see page 5301).
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.

	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

54.1.22.2 TAdapterHiddenFieldsList.Create Constructor

Creates and initializes a TAdapterHiddenFieldsList (see page 5301) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAdapterHiddenFieldsList();
```

Description

Typically, applications have no need to create TAdapterHiddenFieldsList (see page 5301) instances. The adapter component creates and maintains its own internal TAdapterHiddenFieldsList (see page 5301) objects, which it uses to store the hidden fields it needs to add to HTML forms.

See Also

ListHiddenFields (see page 5563)

ListHiddenRecordFields (see page 5564)

54.1.22.3 TAdapterHiddenFieldsList.Destroy Destructor

Frees an instance of TAdapterHiddenFieldsList (see page 5301).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterHiddenFieldsList();
```

Description

Do not use the Destroy method directly to free a TAdapterHiddenFieldsList (see page 5301) instance. Instead, use the Free method, which checks that the TAdapterHiddenFieldsList (see page 5301) reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

54.1.22.4 TAdapterHiddenFieldsList Methods

The methods of the TAdapterHiddenFieldsList class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 5305)	Signals the end of the iterating operation.
	GetCurrentIntf (see page 5306)	This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.

	GetField (see page 5306)	This is GetField, a member of class TAdapterHiddenFieldsList.
	GetFieldCount (see page 5306)	This is GetFieldCount, a member of class TAdapterHiddenFieldsList.
	NextIteration (see page 5307)	Advances to the objects next item.
	StartIterator (see page 5307)	Positions the object on its first item and starts the iteration process.

Public Methods

	Name	Description
	Add (see page 5305)	Adds a name/value pair to the list.
	Clear (see page 5305)	Clears the list of all the name/value pairs it currently holds.

54.1.22.4.1 TAdapterHiddenFieldsList.Add Method

Adds a name/value pair to the list.

Pascal

```
procedure Add(const AName: string; const AValue: string);
```

C++

```
__fastcall Add(const UnicodeString AName, const UnicodeString AValue);
```

Description

Call Add to add a new name/value pair to the list of hidden fields that the adapter maintains.

AName is the name of the field.

AValue is the value of the field.

See Also

Clear (see page 5305)

54.1.22.4.2 TAdapterHiddenFieldsList.Clear Method

Clears the list of all the name/value pairs it currently holds.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the TAdapterHiddenFieldsList (see page 5301) object. Clear discards any name/value pairs that have been added previously by a call to the Add (see page 5305) method.

See Also

Add (see page 5305)

54.1.22.4.3 TAdapterHiddenFieldsList.EndIterator Method

Signals the end of the iterating operation.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

Server-side script calls EndIterator when it stops iterating over the object's items. The object uses this method to perform any clean-up to reflect the fact that it is no longer iterating over its items.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. It is passed to each NextIteration call (which can modify the information), and finally supplied to the EndIterator method, which can free the memory used to store the information.

See Also

StartIterator (see page 1056)

NextIteration (see page 1056)

54.1.22.4.4 TAdapterHiddenFieldsList.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class TAdapterHiddenFieldsList.

54.1.22.4.5 TAdapterHiddenFieldsList.GetField Method

Pascal

```
function GetField(I: Integer): IAdapterHiddenField;
```

C++

```
__fastcall IAdapterHiddenField GetField(int I);
```

Description

This is GetField, a member of class TAdapterHiddenFieldsList.

54.1.22.4.6 TAdapterHiddenFieldsList.GetFieldCount Method

Pascal

```
function GetFieldCount: Integer;
```

C++

```
__fastcall int GetFieldCount();
```

Description

This is GetFieldCount, a member of class TAdapterHiddenFieldsList.

54.1.22.4.7 TAdapterHiddenFieldsList.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

StartIterator (see page 1056)

EndIterator (see page 1056)

54.1.22.4.8 TAdapterHiddenFieldsList.StartIterator Method

Positions the object on its first item and starts the iteration process.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

Server-side script calls StartIterator to begin iterating over the object's items.

OwnerData returns a pointer to information that the object can use to influence the iteration process. StartIterator initializes this information to reflect what ever information the object needs to communicate to itself as it advances over its items. This pointer is then passed to the next call to the NextIteration or EndIterator method.

StartIterator returns true if it successfully moves to the first item, false if there are no items to iterate.

See Also

NextIteration (see page 1056)

EndIterator (see page 1056)

54.1.22.5 TAdapterHiddenFieldsList Properties

The properties of the TAdapterHiddenFieldsList class are listed here.

Protected Properties

	Name	Description
	FieldCount (see page 5308)	This is FieldCount, a member of class TAdapterHiddenFieldsList.
	Fields (see page 5308)	This is Fields, a member of class TAdapterHiddenFieldsList.

54.1.22.5.1 TAdapterHiddenFieldsList.FieldCount Property

Pascal

```
property FieldCount: Integer;
```

C++

```
__property int FieldCount;
```

Description

This is FieldCount, a member of class TAdapterHiddenFieldsList.

54.1.22.5.2 TAdapterHiddenFieldsList.Fields Property

Pascal

```
property Fields [I: Integer]: IAdapterHiddenField;
```

C++

```
__property IAdapterHiddenField Fields[int I];
```

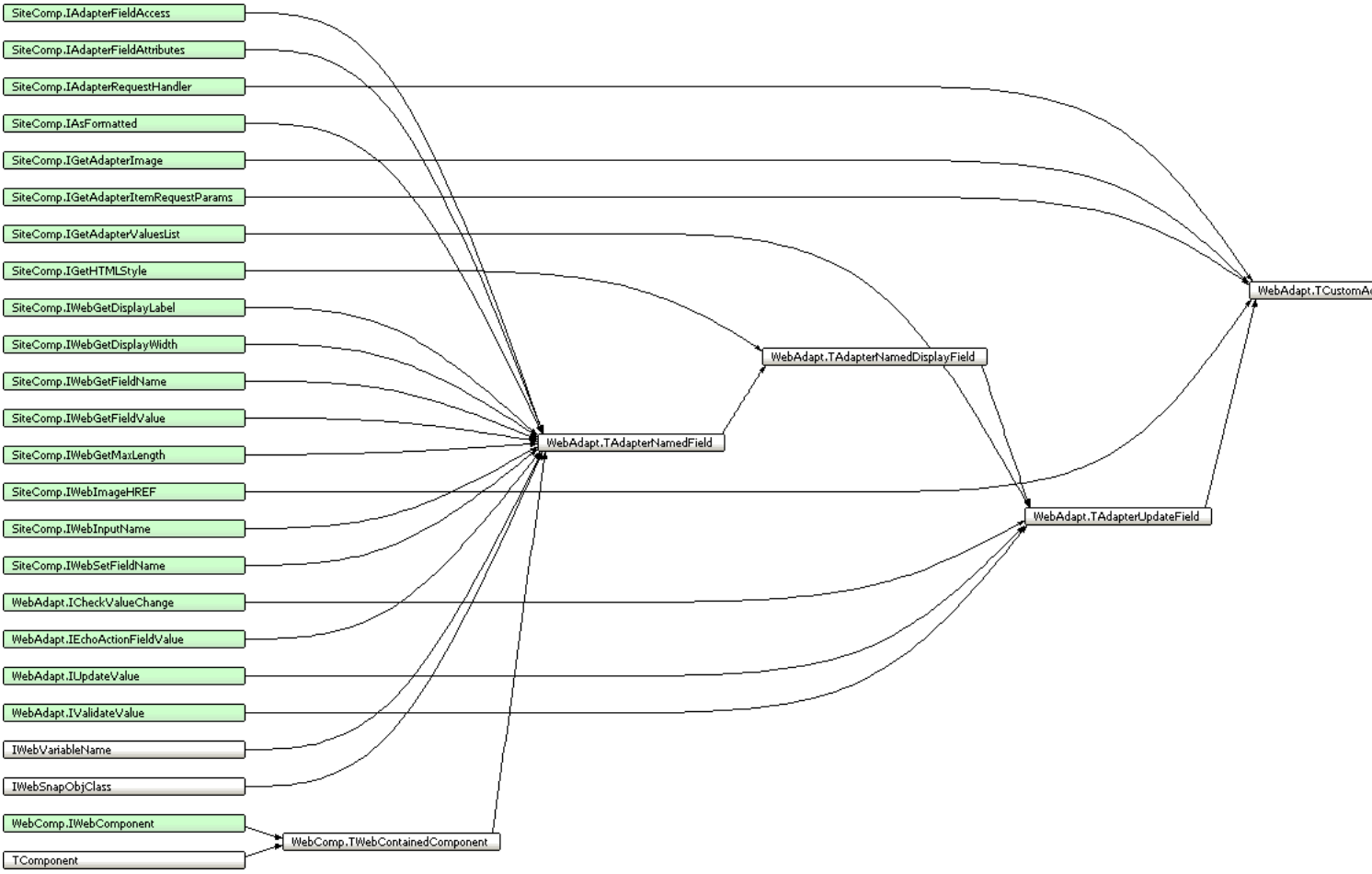
Description

This is Fields, a member of class TAdapterHiddenFieldsList.

54.1.23 TAdapterImageField Class

TAdapterImageField is an event handler based adapter field that can provide the URL of an image file and also accept an uploaded image file.

Class Hierarchy



Pascal

```
TAdapterImageField = class(TCustomAdapterImageField);
```

C++

```
class TAdapterImageField : public TCustomAdapterImageField;
```

File

WebAdapt

Description

TAdapterImageField publishes the properties in TCustomAdapterImageField (see page 5623). TCustomAdapterImageField (see page 5623) calls event handlers to retrieve the adapter field's image file name, image file, and image file URL. It also defines event handler that are called to validate files uploaded by the end-user, and update the adapter field's image file.

Add TAdapterFileField (see page 5281) to TAdapter (see page 5190) or other adapter components types when you need an adapter field that supports an image using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5267), TAdapterMultiValueField (see page 5332), TAdapterMemoField (see page 5323), TAdapterBooleanField (see page 5223) and TAdapterFileField (see page 5281).

See Also

TCustomAdapterImageField (see page 5623)

TAdapter (see page 5190)

TAdapterField (see page 5267)

TAdapterMemoField (see page 5323)

TAdapterMultiValueField (see page 5332)

TAdapterBooleanField (see page 5223)

TAdapterFileField (see page 5281)

54.1.23.1 TAdapterImageField Members

The following tables list the members exposed by TAdapterImageField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
	CreateRequestContext (see page 5628)	Creates AdapterRequest and AdapterResponse objects.
	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
	DoOnValidateValue (see page 5628)	Validates files uploaded by the end-user.
	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 5629)	Provides an object that can be used to access the image properties of this adapter field.
	GetAdapterItemRequestParams (see page 5629)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayStyleType (see page 5630)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.

	GetInputStyleType (see page 5630)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HandleRequest (see page 5630)	Handles a request for the image file associated with this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplCheckValueChange (see page 5631)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 5631)	Creates AdapterRequest and AdapterResponse objects.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetValue (see page 5632)	Returns the name of the image file associated with this adapter field.
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHandleRequest (see page 5632)	Handles a request for the image file associated with this adapter field.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5632)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	ImplWebImageHREF (see page 5633)	Returns a URL that can be used to access the image file associated with this adapter field.

	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	RenderAdapterImage (see page 5633)	Responds to a request for an image file by providing a stream containing the image.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.
	WebImageHREF (see page 5634)	Returns a URL that can be used to access the image file associated with this adapter field.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterImage (see page 1023)	This is GetAdapterImage, a member of class IGetAdapterImage.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.

	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.
	WebImageHREF (see page 1098)	Retrieves a URL for an external image file.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetHREF (see page 5317)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 5318)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5318)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 5318)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5319)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5319)	Used to validate files uploaded by the end user.

Published Properties

	Name	Description
	DisplayLabel (see page 5314)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5314)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5315)	Identifies the name of this field when using server-side script.
	HREF (see page 5315)	Defines a URL that references the image associated with the adapter field.
	ModifyAccess (see page 5315)	Identifies the rights required to update this field.
	ReadOnly (see page 5316)	Indicates whether this adapter field is read-only.
	Required (see page 5316)	Indicates whether the end-user must enter a value for this adapter field.

	ValuesList (see page 5316)	Associates a list of values with the adapter field.
	ViewAccess (see page 5317)	Identifies the rights required to view this field.

54.1.23.2 TAdapterImageField Properties

The properties of the TAdapterImageField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5314)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5314)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5315)	Identifies the name of this field when using server-side script.
	HREF (see page 5315)	Defines a URL that references the image associated with the adapter field.
	ModifyAccess (see page 5315)	Identifies the rights required to update this field.
	ReadOnly (see page 5316)	Indicates whether this adapter field is read-only.
	Required (see page 5316)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5316)	Associates a list of values with the adapter field.
	ViewAccess (see page 5317)	Identifies the rights required to view this field.

54.1.23.2.1 TAdapterImageField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.23.2.2 TAdapterImageField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.23.2.3 TAdapterImageField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.23.2.4 TAdapterImageField.HREF Property

Defines a URL that references the image associated with the adapter field.

Pascal

```
property HREF: string;
```

C++

```
__property UnicodeString HREF;
```

Description

Use the HREF property when this adapter field is associated with an external image file URL. Use the OnGetHREF event to dynamically define an external image file URL. If the image file will be dynamically generated, use the OnGetImage and OnGetImageParams event handlers instead of the HREF property or the OnGetHREF event handler.

See Also

ImplWebImageHREF (see page 5633)

OnGetHREF (see page 5635)

OnGetImage (see page 5636)

OnGetImageName (see page 5636)

54.1.23.2.5 TAdapterImageField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user

attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

54.1.23.2.6 TAdapterImageField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.23.2.7 TAdapterImageField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5359)

54.1.23.2.8 TAdapterImageField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to

enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1072)

TStringsValuesList (see page 5857)

TDataSetValuesList

54.1.23.2.9 TAdapterImageField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

54.1.23.3 TAdapterImageField Events

The events of the TAdapterImageField class are listed here.

Published Events

	Name	Description
	OnGetHREF (see page 5317)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 5318)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5318)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 5318)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5319)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5319)	Used to validate files uploaded by the end user.

54.1.23.3.1 TAdapterImageField.OnGetHREF Event

Provides the URL of an external image file associated with this adapter field.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Implement OnGetHREF to dynamically identify the URL of an external image file associated with this adapter field. When

OnGetHREF is called, the HREF parameter will contain the value of the HREF property. To dynamically generate an image file for this adapter file, implement the OnGetImage event handler instead.

See Also

ImplWebImageHREF (see page 5633)

HREF (see page 5634)

OnGetImage (see page 5636)

54.1.23.3.2 TAdapterImageField.OnGetImage Event

Retrieves a stream containing the image file associated with this adapter field.

Pascal

```
property OnGetImage: TImageFieldGetImageEvent;
```

C++

```
__property TImageFieldGetImageEvent OnGetImage;
```

Description

Implement OnGetImage to dynamically generate an image file associated with this adapter field. Don't implement OnGetImage if the image file is external. Instead implement OnGetHREF event handler or use the HREF property to specify the URL of an external image.

See Also

RenderAdapterImage (see page 5633)

OnGetParams (see page 5636)

OnGetHREF (see page 5635)

HREF (see page 5634)

54.1.23.3.3 TAdapterImageField.OnGetImageName Event

Returns the name of the image file associated with this adapter field.

Pascal

```
property OnGetImageName: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetImageName;
```

Description

Implement this event handler to identify the name of the image file associated with this adapter field.

See Also

ImplGetValue (see page 5632)

54.1.23.3.4 TAdapterImageField.OnGetParams Event

Provides name/value pairs that identify this adapter field's image.

Pascal

```
property OnGetParams: TImageFieldGetParamsEvent;
```

C++

```
__property TImageFieldGetParamsEvent OnGetParams;
```

Description

Implement this event handler to provide name/value pairs that are needed to identify this adapter field's image file. OnGetParams is called while generating an HTML page with a URL reference to this adapter field's image. The parameters returned by OnGetParams will be passed to OnGetImage when the image file is requested.

See Also

GetAdapterItemRequestParams (see page 5629)

OnGetImage (see page 5636)

54.1.23.3.5 TAdapterImageField.OnUploadFile Event

Updates this adapter field with an image file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

This event handler is called by ImplUpdateValue. This event handler should store the uploaded image file. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5632)

TAbstractWebRequestFile

54.1.23.3.6 TAdapterImageField.OnValidateFile Event

Used to validate files uploaded by the end user.

Pascal

```
property OnValidateFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnValidateFile;
```

Description

Validates the file uploaded by the end-user. Raise an exception to indicate an error. For example, it raises an exception if the Mime type of the file is not supported.

See Also

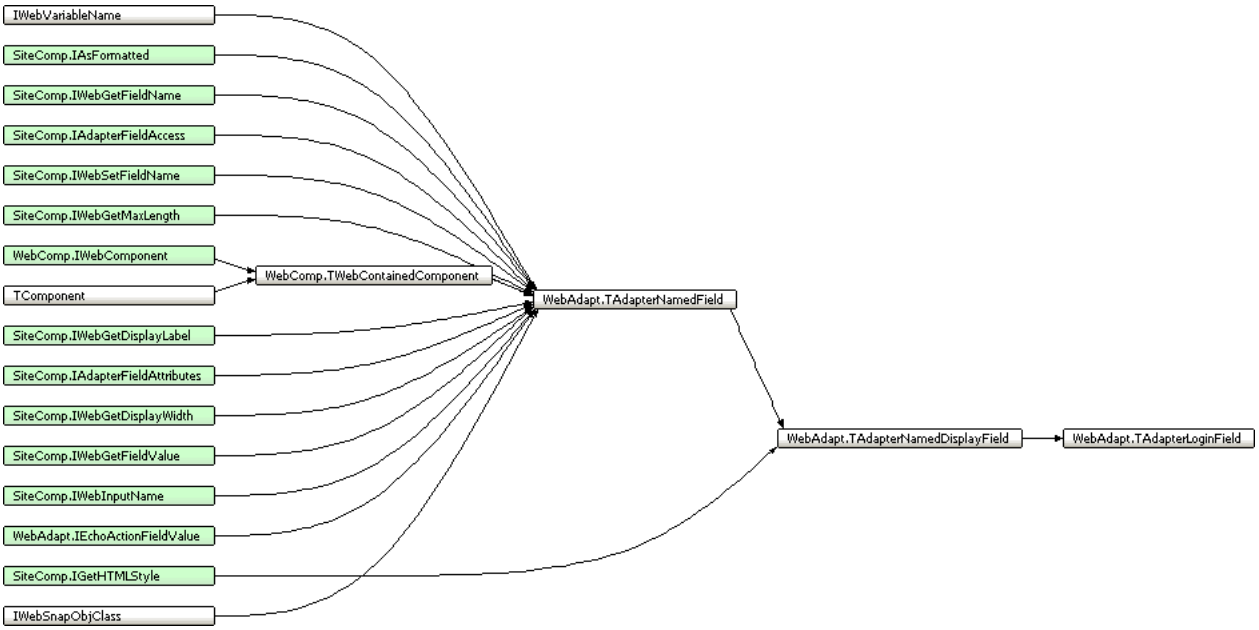
DoOnValidateValue (see page 5628)

TAbstractWebRequestFile

54.1.24 TAdapterLoginField Class

TAdaperLoginField is a base class for adapter field components that are children of a TLoginFormAdapter (see page 5824) component.

Class Hierarchy



Pascal

```
TAdapterLoginField = class(TAdapterNamedDisplayField);
```

C++

```
class TAdapterLoginField : public TAdapterNamedDisplayField;
```

File

WebAdapt

Description

TAdapterLoginField implements functionality that is used by an adapter field that is a child of TLoginFormAdapter (see page 5824).

See Also

TLoginFormAdapter (see page 5824)

TAdapterNamedDisplayField (see page 5343)

TAdapterUserNameField (see page 5433)

TAdapterPasswordField (see page 5391)

54.1.24.1 TAdapterLoginField Members

The following tables list the members exposed by TAdapterLoginField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.

	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
	Adapter (see page 5323)	Returns the parent adapter of this adapter field.

W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.24.2 TAdapterLoginField Properties

The properties of the TAdapterLoginField class are listed here.

Public Properties

	Name	Description
R	Adapter (see page 5323)	Returns the parent adapter of this adapter field.

54.1.24.2.1 TAdapterLoginField.Adapter Property

Returns the parent adapter of this adapter field.

Pascal

```
property Adapter: TCustomLoginFormAdapter;
```

C++

```
__property TCustomLoginFormAdapter Adapter;
```

Description

Descendants use this property to access the parent adapter. GetAdapter (see page 5356) implements this property.

See Also

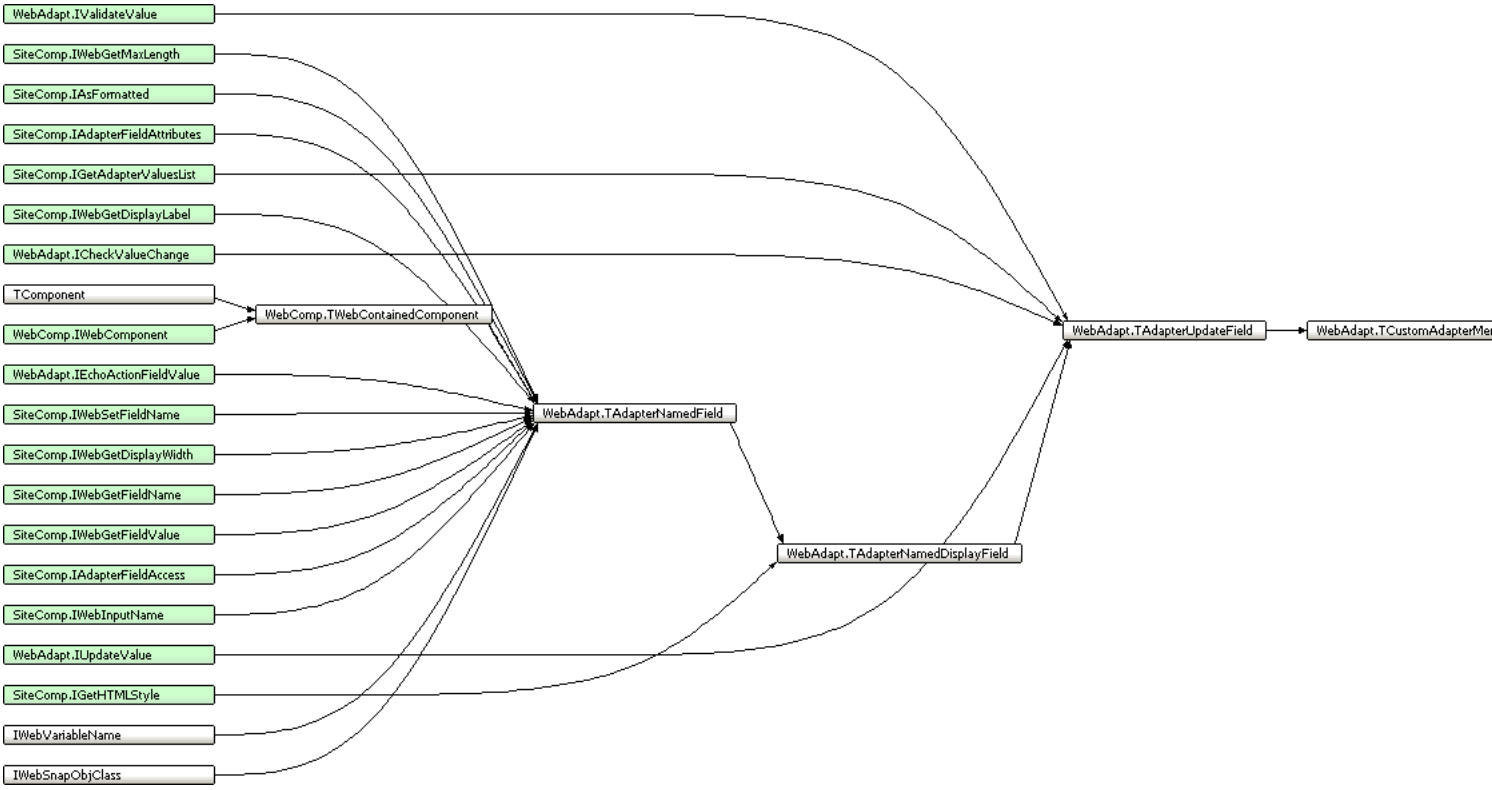
TCustomLoginFormAdapter (see page 5701)

GetAdapter (see page 5356)

54.1.25 TAdapterMemoField Class

TAdapterMemoField is an event handler based adapter field that supports a multi-line text value.

Class Hierarchy



Pascal

```
TAdapterMemoField = class(TCustomAdapterMemoField);
```

C++

```
class TAdapterMemoField : public TCustomAdapterMemoField;
```

File

WebAdapt

Description

TAdapterMemoField publishes the properties in TCustomAdapterMemoField (see page 5638). TCustomAdapterMemoField (see page 5638) calls event handlers to retrieve the adapter field's text value, validate a value entered end-user, and update the adapter field's value.

Add TAdapterMemoField to TAdapter (see page 5190) or other adapter components types when you need an adapter field that supports a multi-line text value. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5267), TAdapterMultiValueField (see page 5332), TAdapterFileField (see page 5281), TAdapterBooleanField (see page 5223) and TAdapterImageField (see page 5308).

See Also

TCustomAdapterMemoField (see page 5638)

TAdapter (see page 5190)

TAdapterField (see page 5267)

TAdapterFileField (see page 5281)

TAdapterMultiValueField (see page 5332)

TAdapterBooleanField (see page 5223)

TAdapterImageField (see page 5308)

54.1.25.1 TAdapterMemoField Members

The following tables list the members exposed by TAdapterMemoField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5642)	Validates a value entered by the end user.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5642)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5643)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
V	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.

	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplCheckValueChange (see page 5643)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetValue (see page 5644)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5644)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
R	Adapter (see page 5368)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
V	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.

	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValue (see page 5331)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5331)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5332)	Called to update the adapter field's value with a file uploaded by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5328)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5329)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5329)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5329)	Identifies the rights required to update this field.
	ReadOnly (see page 5330)	Indicates whether this adapter field is read-only.
	Required (see page 5330)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5330)	Identifies the rights required to view this field.

54.1.25.2 TAdapterMemoField Properties

The properties of the TAdapterMemoField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5328)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5329)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5329)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5329)	Identifies the rights required to update this field.
	ReadOnly (see page 5330)	Indicates whether this adapter field is read-only.
	Required (see page 5330)	Indicates whether the end-user must enter a value for this adapter field.
	ViewAccess (see page 5330)	Identifies the rights required to view this field.

54.1.25.2.1 TAdapterMemoField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.25.2.2 TAdapterMemoField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.25.2.3 TAdapterMemoField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.25.2.4 TAdapterMemoField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user

attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

54.1.25.2.5 TAdapterMemoField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.25.2.6 TAdapterMemoField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5359)

54.1.25.2.7 TAdapterMemoField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView

property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

54.1.25.3 TAdapterMemoField Events

The events of the TAdapterMemoField class are listed here.

Published Events

	Name	Description
	OnGetValue (see page 5331)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5331)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5332)	Called to update the adapter field's value with a file uploaded by the end-user.

54.1.25.3.1 TAdapterMemoField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5644)

54.1.25.3.2 TAdapterMemoField.OnUpdateValue Event

Called to update the adapter field's value with a text value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldStringEvent;
```

C++

```
__property TAdapterFieldStringEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 5644)

54.1.25.3.3 TAdapterMemoField.OnUploadFile Event

Called to update the adapter field's value with a file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

OnUploadFile is called when an action request contains an uploaded file for this adapter field. If this event handler is not implemented, the file uploaded by the end-user will be ignored.

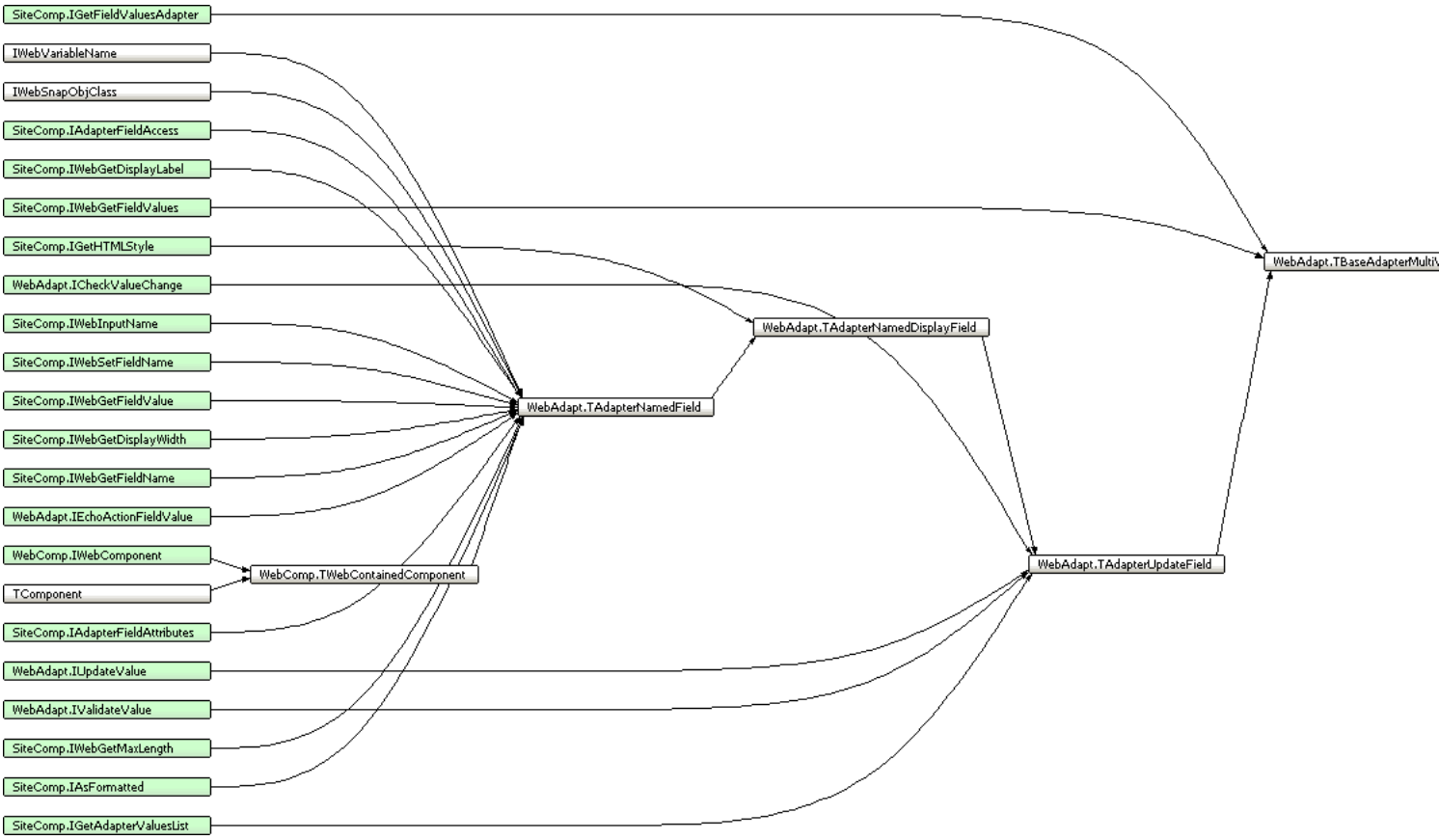
See Also

ImplUpdateValue (see page 5644)

54.1.26 TAdapterMultiValueField Class

TAdapterMultiValueField is an event handler based adapter field that provides zero or more values.

Class Hierarchy



Pascal

```
TAdapterMultiValueField = class(TCustomAdapterMultiValueField);
```

C++

```
class TAdapterMultiValueField : public TCustomAdapterMultiValueField;
```

File

WebAdapt

Description

TAdapterMultiValueField publishes the properties in TCustomAdapterMultiValueField (see page 5646). TCustomAdapterMultiValueField (see page 5646) calls event handlers to retrieve the adapter field's values, validate values entered by the end-user, and update the adapter field's values.

Add TAdapterMultiValueField to TAdapter (see page 5190) or other adapter components types when you need an adapter field that supports a multiple values using event handlers. There are other adapter field components that use event handlers. They differ in the type and number of values that are supported. See TAdapterField (see page 5267), TAdapterFileField (see page 5281), TAdapterMemoField (see page 5323), TAdapterBooleanField (see page 5223) and TAdapterImageField (see page 5308).

See Also

TCustomAdapterMultiValueField (see page 5646)

TAdapter (see page 5190)

TAdapterField (see page 5267)

TAdapterMemoField (see page 5323)

TAdapterFileField (see page 5281)

TAdapterBooleanField (see page 5223)

TAdapterImageField (see page 5308)

54.1.26.1 TAdapterMultiValueField Members

The following tables list the members exposed by TAdapterMultiValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
V	DoOnValidateValue (see page 5651)	Validates values entered by the end-user.
	EndValueIterator (see page 5493)	Completes iteration.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5494)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
V	GetDisplayStyleType (see page 5651)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5494)	Returns the adapter object that implements the Values property.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetInputStyleType (see page 5651)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
V	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
V	GetValueCount (see page 5494)	Returns the number of values associated with this adapter field.
V	GetValueCount (see page 5652)	Returns the number of values associated with this adapter field.

	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetValues (see page 5495)	Returns one of the values associated with the adapter field.
	GetValues (see page 5652)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' HasValue method.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetCurrentValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue (see page 5494) method.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 5496)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 5496)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
	ImplHasValue (see page 5653)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
	ImplUpdateValue (see page 5653)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5496)	Implements IWebGetFieldValues (see page 1092)' NextValueIteration method.

	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartValueIterator (see page 5497)	Implements IWebGetFieldValues (see page 1092)' StartValueIterator method.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 5492)	Constructor for TBaseAdapterMultiValueField (see page 5488).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 5492)	Destructor for TBaseAdapterMultiValueField (see page 5488).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndValueIterator (see page 1093)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetCurrentValue (see page 1094)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 1031)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.

	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1094)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1094)	Position the iteration pointer on the next value.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartValueIterator (see page 1095)	Reset (see page 3643) the iteration pointer to the first item.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
R	ValueCount (see page 5653)	Returns the number of values associated with this adapter field.
R	Values (see page 5654)	Returns one of the values associated with the adapter field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Events

	Name	Description
	OnGetValueCount (see page 5341)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5341)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5342)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5342)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 5342)	Validates field values entered by the end-user.

Published Properties

	Name	Description
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5339)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5339)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
	Required (see page 5340)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5340)	Associates a list of values with the adapter field.
	ViewAccess (see page 5340)	Identifies the rights required to view this field.

54.1.26.2 TAdapterMultiValueField Properties

The properties of the TAdapterMultiValueField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5338)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5338)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5339)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5339)	Identifies the rights required to update this field.
	ReadOnly (see page 5339)	Indicates whether this adapter field is read-only.
	Required (see page 5340)	Indicates whether the end-user must enter a value for this adapter field.
	ValuesList (see page 5340)	Associates a list of values with the adapter field.
	ViewAccess (see page 5340)	Identifies the rights required to view this field.

54.1.26.2.1 TAdapterMultiValueField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.26.2.2 TAdapterMultiValueField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.26.2.3 TAdapterMultiValueField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.26.2.4 TAdapterMultiValueField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

54.1.26.2.5 TAdapterMultiValueField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.26.2.6 TAdapterMultiValueField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

See Also

GetRequired (see page 5359)

54.1.26.2.7 TAdapterMultiValueField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1072)

TStringsValuesList (see page 5857)

TDataSetValuesList

54.1.26.2.8 TAdapterMultiValueField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

54.1.26.3 TAdapterMultiValueField Events

The events of the TAdapterMultiValueField class are listed here.

Published Events

	Name	Description
	OnGetValueCount (see page 5341)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5341)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5342)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5342)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 5342)	Validates field values entered by the end-user.

54.1.26.3.1 TAdapterMultiValueField.OnGetValueCount Event

Responsible for returning the count of values associated with the adapter field.

Pascal

```
property OnGetValueCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetValueCount;
```

Description

Implement this event handler to provide the count of values associated with an adapter field. Also implement OnGetValues to provide the individual values.

See Also

GetValueCount (see page 5652)

OnGetValues (see page 5655)

54.1.26.3.2 TAdapterMultiValueField.OnGetValues Event

Responsible for returning one of the values associated with the adapter field.

Pascal

```
property OnGetValues: TAdapterFieldGetMultiValueEvent;
```

C++

```
__property TAdapterFieldGetMultiValueEvent OnGetValues;
```

Description

Implement this event handler to provide the values associated with an adapter field. The Index parameter is zero-based. Also implement OnGetValueCount to provide the count of values associated with this adapter field.

See Also

GetValues (see page 5652)

OnGetValueCount (see page 5654)

54.1.26.3.3 TAdapterMultiValueField.OnHasValue Event

Indicates whether a particular value is associated with this adapter field.

Pascal

```
property OnHasValue: TAdapterFieldHasValueEvent;
```

C++

```
__property TAdapterFieldHasValueEvent OnHasValue;
```

Description

ImplHasValue calls this event handler. Implement this event handler to lookup a matching value. Implementation of this event handler is optional. If OnHasValue is nil, ImplHasValue will search for a matching value the Values and ValueCount properties.

See Also

ImplHasValue (see page 5653)

ValueCount (see page 5653)

Values (see page 5654)

54.1.26.3.4 TAdapterMultiValueField.OnUpdateValues Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUpdateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnUpdateValues;
```

Description

Implement this event handler to process values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues (see page 5656)

ImplUpdateValue (see page 5653)

54.1.26.3.5 TAdapterMultiValueField.OnValidateValues Event

Validates field values entered by the end-user.

Pascal

```
property OnValidateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnValidateValues;
```

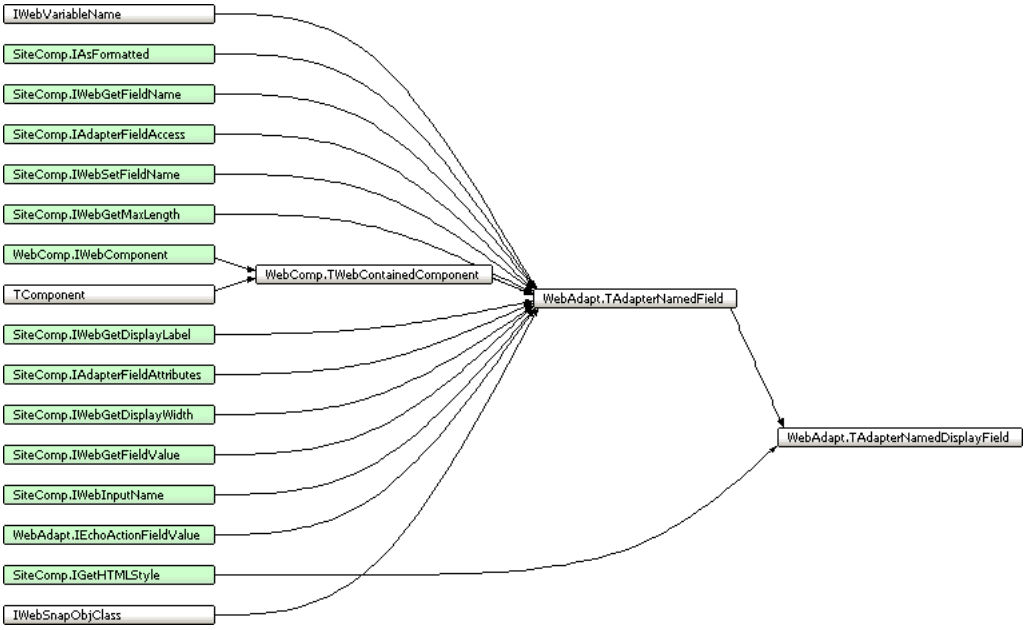
Description

Implement this event handler to validate values entered by the end-user. Raise an exception if an error occurs.

54.1.27 TAdapterNamedDisplayField Class

The AdapterNamedDisplayField is a base class for adapter fields.

Class Hierarchy



Pascal

```
TAdapterNamedDisplayField = class(TAdapterNamedField, IGetHTMLStyle);
```

C++

```
class TAdapterNamedDisplayField : public TAdapterNamedField, public IGetHTMLStyle;
```

File

WebAdapt

Description

TAdapterNameDisplayField implements the IGetHTMLStyle (see page 1031) interface used to inquire how this adapter field should be displayed in an HTML form. For example, IGetHTMLStyle (see page 1031)'s GetInputStyle (see page 5347) method may indicate that this field should be displayed using a select list. TAdapterNamedField (see page 5351) also declares properties that may be modified at design-time to control how the adapter field should be displayed. For example, the DisplayLabel (see page 5350) property is used to suggest display label for this adapter field.

See Also

TAdapterNamedField (see page 5351)

TCustomAdapter (see page 5531)

TAdapter (see page 5190)

IGetHTMLStyle (see page 1031)

54.1.27.1 TAdapterNamedDisplayField Members

The following tables list the members exposed by TAdapterNamedDisplayField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.

V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
R	Adapter (see page 5368)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.

	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.27.2 TAdapterNamedDisplayField.Create Constructor

Creates an instance of TAdapterNamedDisplayField (see page 5343).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterNamedDisplayField(TComponent * AOwner);
```

Description

Do not call Create. It will be called automatically when the TAdapterNamedDisplayField (see page 5343) object is created

54.1.27.3 TAdapterNamedDisplayField Methods

The methods of the TAdapterNamedDisplayField class are listed here.

Protected Methods

	Name	Description
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.

54.1.27.3.1 TAdapterNamedDisplayField.GetDisplayStyle Method

Returns a string value to suggest an HTML display style for this adapter field.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(const UnicodeString AAdapterMode);
```

Description

GetDisplayStyle implements IGetHTMLStyle (see page 1031)'s GetDisplayStyle by translating the result value of GetDisplayStyleType (see page 5347) into a string.

See Also

GetDisplayStyle (see page 1032)

GetDisplayStyleType (see page 5347)

54.1.27.3.2 TAdapterNamedDisplayField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
virtual;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmldText, indicating that this adapter field should be displayed by simple text. Descendant classes may override this method to suggest other HTML display types.

See Also

GetDisplayStyle (see page 5346)

54.1.27.3.3 TAdapterNamedDisplayField.GetInputStyle Method

Returns a string value to suggest an HTML input style for this adapter field.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetInputStyle(const UnicodeString AAdapterMode);
```

Description

GetInputStyle implements IGetHTMLStyle (see page 1031)'s GetInputType method by translating the result value of GetInputStyleType (see page 5347) into a string.

See Also

GetInputStyle (see page 1033)

GetInputStyleType (see page 5347)

54.1.27.3.4 TAdapterNamedDisplayField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; virtual;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType returns htmliTextInput, indicating that this adapter field should be edited by a single line input field. Descendant classes may override this method to suggest a different HTML input type.

See Also

GetInputStyle (see page 5347)

54.1.27.3.5 TAdapterNamedDisplayField.GetViewMode Method

Returns a string value to suggest a view mode for this adapter field.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetViewMode(const UnicodeString AAdapterMode);
```

Description

GetViewMode implements IGetHTMLStyle (see page 1031)'s GetViewMode method by translating the result value of GetViewModeType (see page 5348) into a string.

See Also

GetViewMode (see page 1033)

GetViewModeType (see page 5348)

54.1.27.3.6 TAdapterNamedDisplayField.GetViewModeType Method

Suggests a view mode for this adapter field.

Pascal

```
function GetViewModeType(const AAdapterMode: string): TAdapterDisplayViewModeType; virtual;
```

C++

```
virtual __fastcall TAdapterDisplayViewModeType GetViewModeType(const UnicodeString
AAdapterMode);
```

Description

GetViewModeType returns htmlvmDisplay if the ReadOnly (see page 5351) property is true. Otherwise, htmlvmInput is returned.

See Also

ReadOnly (see page 5351)

GetViewMode (see page 5348)

54.1.27.3.7 TAdapterNamedDisplayField.ImplGetDisplayLabel Method

Implements interface access to the DisplayLabel (see page 5350) property.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 5350) property.

See Also

DisplayLabel (see page 5350)

54.1.27.3.8 TAdapterNamedDisplayField.ImplGetDisplayWidth Method

Implements interface access to the DisplayWidth (see page 5350) property.

Pascal

```
function ImplGetDisplayWidth: Integer; override;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

ImplGetDisplayWidth returns the value of the DisplayWidth (see page 5350) property.

See Also

DisplayWidth (see page 5350)

54.1.27.3.9 TAdapterNamedDisplayField.ImplGetRequired Method

Implements interface access to the Required (see page 5351) property.

Pascal

```
function ImplGetRequired: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetRequired();
```

Description

ImplGetRequired returns the value of the Required (see page 5351) property.

See Also

Required (see page 5351)

54.1.27.3.10 TAdapterNamedDisplayField.SetDisplayLabel Method

SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.

Pascal

```
procedure SetDisplayLabel(const AValue: string);
```

C++

```
__fastcall SetDisplayLabel(const UnicodeString AValue);
```

Description

Use SetDisplayLabel to set the value of the DisplayLabel (see page 5350) property to AValue.

See Also

DisplayLabel (see page 5350)

54.1.27.4 TAdapterNamedDisplayField Properties

The properties of the TAdapterNamedDisplayField class are listed here.

Public Properties

	Name	Description
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.

54.1.27.4.1 TAdapterNamedDisplayField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.27.4.2 TAdapterNamedDisplayField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.27.4.3 TAdapterNamedDisplayField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode (see page 5348) which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.27.4.4 TAdapterNamedDisplayField.Required Property

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

Use this property to indicate whether a value must be entered by the end user before submitting a form. Typically, an exception is raised when a required adapter field is not found when processing an adapter action.

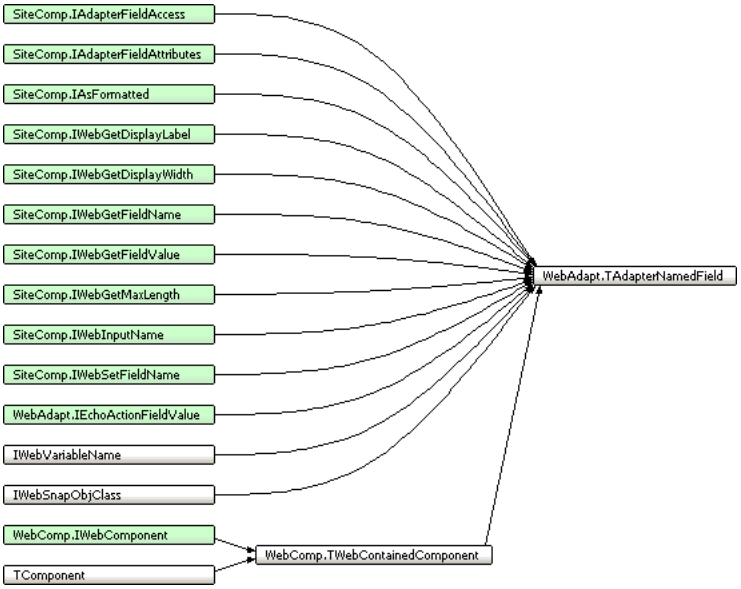
See Also

GetRequired (see page 5359)

54.1.28 TAdapterNamedField Class

TAdapterNamedField is a based class for adapter field components.

Class Hierarchy



Pascal

```
TAdapterNamedField = class(TWebContainedComponent, IWebVariableName, IWebInputName,
IWebGetFieldValue, IEchoActionFieldValue, IAsFormatted, IWebGetFieldName, IWebSetFieldName,
IWebGetDisplayWidth, IWebGetDisplayLabel, IWebGetMaxLength, IWebSnapObjClass,
IAdapterFieldAccess, IAdapterFieldAttributes);
```

C++

```
class TAdapterNamedField : public TWebContainedComponent, public IWebVariableName, public
IWebInputName, public IWebGetFieldValue, public IEchoActionFieldValue, public IAsFormatted,
public IWebGetFieldName, public IWebSetFieldName, public IWebGetDisplayWidth, public
IWebGetDisplayLabel, public IWebGetMaxLength, public IWebSnapObjClass, public
IAdapterFieldAccess, public IAdapterFieldAttributes;
```

File

WebAdapt

Description

TAdapterNamedField implements interfaces used to access an adapter field's properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize the behavior of the adapter field class.

See Also

TAdapterNamedDisplayField (see page 5343)

TAdapter (see page 5190)

54.1.28.1 TAdapterNamedField Members

The following tables list the members exposed by TAdapterNamedField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
R	Adapter (see page 5368)	Returns the parent adapter component.

R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.28.2 TAdapterNamedField Methods

The methods of the TAdapterNamedField class are listed here.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.

	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.

54.1.28.2.1 TAdapterNamedField.AsFormatted Method

Returns text used to generate a representation of the adapter field value.

Pascal

```
function AsFormatted: string;
```

C++

```
__fastcall UnicodeString AsFormatted();
```

Description

This function returns a text that is typically used when generating a read-only representation of the adapter field value. The value returned by this function can be accessed with server-side script using the DisplayText property.

See Also

ImplAsFormatted (see page 5361)

54.1.28.2.2 TAdapterNamedField.CheckModifyAccess Method

Checks end-user rights to modify the adapter field value.

Pascal

```
procedure CheckModifyAccess;
```

C++

```
__fastcall CheckModifyAccess();
```

Description

This procedure is called to verify that the current user has rights to modify this field. An exception will be raised if access is denied.

See Also

ModifyAccess (see page 5369)

HasModifyAccess (see page 5361)

54.1.28.2.3 TAdapterNamedField.CheckViewAccess Method

Checks end-user rights to view the adapter field value.

Pascal

```
procedure CheckViewAccess;
```

C++

```
__fastcall CheckViewAccess();
```

Description

This procedure is called to verify that the current user has rights to view this field. An exception will be raised if access is denied.

See Also

ViewAccess (see page 5370)

HasModifyAccess (see page 5361)

54.1.28.2.4 TAdapterNamedField.FullyQualifyInputName Method

Indicates whether the HTML input names must be qualified with the name of the adapter.

Pascal

```
function FullyQualifyInputName: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FullyQualifyInputName();
```

Description

This functions returns true if, when generating an HTML input referencing this adapter field, the name of the HTML input must be qualified with the name of the adapter component. Input (see page 3827) names must be fully qualified when a single HTML form is used to input values for multiple adapters.

See Also

MasterAdapter

54.1.28.2.5 TAdapterNamedField.GetAdapter Method

GetAdapter returns the parent adapter component.

Pascal

```
function GetAdapter: TCustomAdapter;
```

C++

```
__fastcall TCustomAdapter GetAdapter();
```

Description

To improve the efficiency of this function, a reference to the parent adapter component is cached in an instance variable.

54.1.28.2.6 TAdapterNamedField.GetDefaultDisplayLabel Method

Provides the default display label for this adapter field.

Pascal

```
function GetDefaultDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

When the display label for the field matches this value then the display label property does not need to be stored in the form file.

54.1.28.2.7 TAdapterNamedField.GetDefaultFieldName Method

Gets the default name used to reference this adapter field with server-side script.

Pascal

```
function GetDefaultFieldName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

This function returns the default field name for the field. This name will be used when a field name has not be set using ImplSetFieldName (see page 5366). GetDefaultFieldName returns the name of the adapter field component. Descendants may override this function.

See Also

ImplGetVariableName (see page 5364)

ImplSetFieldName (see page 5366)

54.1.28.2.8 TAdapterNamedField.GetDisplayLabel Method

Returns the suggested display label for this adapter field.

Pascal

```
function GetDisplayLabel: string;
```

C++

```
__fastcall UnicodeString GetDisplayLabel();
```

Description

GetDisplayLabel provides label that may be used when generating HTML to display the value of this adapter field. The value

returned by this function can be accessed with server-side script using the DisplayLabel property. GetDisplayLabel calls ImplGetDisplayLabel (see page 5361).

See Also

ImplGetDisplayLabel (see page 5361)

54.1.28.2.9 TAdapterNamedField.GetDisplayWidth Method

Returns the suggested display width for this adapter field.

Pascal

```
function GetDisplayWidth: Integer;
```

C++

```
__fastcall int GetDisplayWidth();
```

Description

This function returns the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. The value returned by this function can be accessed with server-side script using the DisplayWidth property. GetDisplayWidth calls ImplGetDisplayWidth (see page 5362).

See Also

ImplGetDisplayWidth (see page 5362)

54.1.28.2.10 TAdapterNamedField.GetEchoActionFieldValue Method

Returns the value of the echo flag.

Pascal

```
function GetEchoActionFieldValue: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValue();
```

Description

EchoActionFieldValue (see page 5368) calls ImplEchoActionFieldValue.

See Also

EchoActionFieldValue (see page 5368)

54.1.28.2.11 TAdapterNamedField.GetInputName Method

Identifies the name to use when generating an HTML input for this adapter field.

Pascal

```
function GetInputName: string;
```

C++

```
__fastcall UnicodeString GetInputName();
```

Description

This functions returns the name used for an HTML input associated with this adapter field. This value is accessed with server-side script using the InputName property. GetInputName calls ImplGetInputName (see page 5362).

See Also

ImplGetInputName (see page 5362)

54.1.28.2.12 TAdapterNamedField.GetMaxLength Method

Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.

Pascal

```
function GetMaxLength: Integer;
```

C++

```
__fastcall int GetMaxLength();
```

Description

The value returned by this function can be accessed with server-side script using the MaxLength property. Server-side script may use this value to define HTML elements that limit the number of characters that can be entered by the end-user. GetMaxLength calls ImplGetMaxLength (see page 5363).

See Also

ImplGetMaxLength (see page 5363)

54.1.28.2.13 TAdapterNamedField.GetRequired Method

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This function returns true or false to indicate whether a value must be entered by the end user before submitting a form. The value returned by this function can be accessed with server-side script using the Required property. Typically, this function is called while executing an adapter action that validates field values. The adapter action component will raise an exception if the value of the submitted field is empty. GetRequired calls ImplGetRequired (see page 5363).

See Also

ImplGetRequired (see page 5363)

54.1.28.2.14 TAdapterNamedField.GetValue Method

Gets the value of this adapter field as a Variant (see page 3421).

Pascal

```
function GetValue: Variant;
```

C++

```
__fastcall Variant GetValue();
```

Description

GetValue returns the result of ImplGetValue (see page 5363).

See Also

Value (see page 5369)

ImplGetValue (see page 5363)

54.1.28.2.15 TAdapterNamedField.GetVariableName Method

Gets the name used to reference this adapter field with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall UnicodeString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter field with server-side script. GetVariableName calls ImplGetVariableName (see page 5364).

See Also

ImplGetVariableName (see page 5364)

54.1.28.2.16 TAdapterNamedField.GetVisible Method

Indicates whether the value of this adapter field should be displayed to the end-user.

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

The value returned by this function can be accessed with server-side script using the Visible property. GetVisible calls ImplGetVisible (see page 5364).

See Also

ImplGetVisible (see page 5364)

54.1.28.2.17 TAdapterNamedField.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

54.1.28.2.18 TAdapterNamedField.HasModifyAccess Method

Checks the end-user's right to modify this adapter field.

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanModify property. HasModifyAccess calls ImplHasModifyAccess (see page 5365).

See Also

ImplHasModifyAccess (see page 5365)

54.1.28.2.19 TAdapterNamedField.HasViewAccess Method

Checks the end-user's right to view this adapter field.

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanView property. HasViewAccess calls ImplHasViewAccess (see page 5365).

See Also

ImplHasViewAccess (see page 5365)

54.1.28.2.20 TAdapterNamedField.ImplAsFormatted Method

Returns text used to generate a representation of the adapter field value.

Pascal

```
function ImplAsFormatted: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplAsFormatted();
```

Description

ImplAsFormatted is called by AsFormatted (see page 5355).

See Also

AsFormatted (see page 5355)

54.1.28.2.21 TAdapterNamedField.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter field.

Pascal

```
function ImplGetDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel is called by GetDisplayLabel (see page 5357). ImplGetDisplayLabel returns the result of GetDefaultDisplayLabel (see page 5357). Descendants may override this function.

See Also

GetDisplayLabel (see page 5357)

GetDefaultDisplayLabel (see page 5357)

54.1.28.2.22 TAdapterNamedField.ImplGetDisplayWidth Method

Returns the suggested display width for this adapter field.

Pascal

```
function ImplGetDisplayWidth: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

This function is called by GetDisplayWidth (see page 5358) to retrieve the suggested display width for this adapter field. ImplGetDisplayWidth returns –1 to indicate an undefined display width. Descendants may override this function.

See Also

GetDisplayWidth (see page 5358)

54.1.28.2.23 TAdapterNamedField.ImplGetEchoActionFieldValue Method

Returns the value of the echo flag.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

This function is called by GetEchoActionFieldValue (see page 5358).

See Also

EchoActionFieldValue (see page 5368)

GetEchoActionFieldValue (see page 5358)

54.1.28.2.24 TAdapterNamedField.ImplGetInputName Method

Identifies the name to use when generating the HTML input for this adapter field.

Pascal

```
function ImplGetInputName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetInputName();
```

Description

When an action request is received, the name/value pairs in the HTTP are searched for adapter field values. For each adapter field value found, an IActionFieldValue (see page 5904) implementation is created. In order for the HTTP request to be interpreted properly, the HTML input fields must have names that correspond to the adapter fields they represent.

See Also

GetInputName (see page 5358)

IActionFieldValue

54.1.28.2.25 TAdapterNamedField.ImplGetMaxLength Method

Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.

Pascal

```
function ImplGetMaxLength: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetMaxLength();
```

Description

ImplGetMaxLength is called by GetMaxLength (see page 5359). ImplGetMaxLength returns –1 to indicate an undefined maximum length. Descendants may override this function.

See Also

GetMaxLength (see page 5359)

54.1.28.2.26 TAdapterNamedField.ImplGetRequired Method

Indicates whether the end-user must enter a value for this adapter field.

Pascal

```
function ImplGetRequired: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetRequired();
```

Description

ImplGetRequired returns False. Descendants may override this function.

See Also

GetRequired (see page 5359)

54.1.28.2.27 TAdapterNamedField.ImplGetValue Method

Gets the value of this adapter field as a Variant (see page 3421).

Pascal

```
function ImplGetValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

This function is called by GetValue (see page 5359). ImplGetValue returns Unassigned. Descendants override this function.

See Also

Value (see page 5369)

GetValue (see page 5359)

54.1.28.2.28 TAdapterNamedField.ImplGetVariableName Method

Gets the name used to reference this adapter field with server-side script.

Pascal

```
function ImplGetVariableName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetVariableName();
```

Description

This function is called by GetVariableName (see page 5360). GetVariableName (see page 5360) returns the field name set by ImplSetFieldName (see page 5366) unless this name is blank. If the field name is blank, the result of GetDefaultFieldName (see page 5357) is returned.

See Also

GetVariableName (see page 5360)

ImplSetFieldName (see page 5366)

GetDefaultFieldName (see page 5357)

54.1.28.2.29 TAdapterNamedField.ImplGetVisible Method

Indicates whether the value of this adapter field should be displayed to the end-user.

Pascal

```
function ImplGetVisible: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

This function returns true to indicate that the field should be displayed to the end user. Descendants may override this method.

See Also

GetVisible (see page 5360)

54.1.28.2.30 TAdapterNamedField.ImplHasModifyAccess Method

Checks the end-user's right to modify this adapter field.

Pascal

```
function ImplHasModifyAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyAccess();
```

Description

This function is called by HasModifyAccess (see page 5361) to check the end-user's right to modify the adapter field. If the ModifyAccess (see page 5369) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ModifyAccess (see page 5369) property value.

See Also

HasModifyAccess (see page 5552)

ModifyAccess (see page 5369)

HasModifyAccess (see page 5361)

54.1.28.2.31 TAdapterNamedField.ImplHasViewAccess Method

Checks the end-user's right to view this adapter field.

Pascal

```
function ImplHasViewAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewAccess();
```

Description

This function is called by HasViewAccess (see page 5361) to check the end-user's right to view the adapter field. If the ViewAccess (see page 5370) property is not blank, this function calls the EndUserAdapter's HasViewRights function, passing the ViewAccess (see page 5370) property value.

See Also

HasViewAccess (see page 5553)

ViewAccess (see page 5370)

HasViewAccess (see page 5361)

54.1.28.2.32 TAdapterNamedField.ImplSetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); virtual;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplEchoActionFieldValue is called by SetEchoActionFieldValue (see page 5366). ImplEchoActionFieldValue is a placeholder. Descendants may override this method.

See Also

EchoActionFieldValue (see page 5368)

SetEchoActionFieldValue (see page 5366)

54.1.28.2.33 TAdapterNamedField.ImplSetFieldName Method

Sets the name that is used to reference this adapter field with server-side script.

Pascal

```
procedure ImplSetFieldName(const Value: string); virtual;
```

C++

```
virtual __fastcall ImplSetFieldName(const UnicodeString Value);
```

Description

ImplSetFieldName is called by SetFieldName (see page 5367). ImplSetFieldName stores the name in an instance variable. This method may be overridden by descendants.

See Also

SetFieldName (see page 5367)

54.1.28.2.34 TAdapterNamedField.ParentChanged Method

Handles movement of the component to a different parent.

Pascal

```
procedure ParentChanged; override;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called when the parent component is changed. TAdapterNamedField (see page 5351) clears its cached reference to the parent adapter.

54.1.28.2.35 TAdapterNamedField.SetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure SetEchoActionFieldValue(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValue(Boolean AValue);
```

Description

SetEchoActionFieldValue calls ImplSetActionFieldValue.

See Also

EchoActionFieldValue (see page 5368)

ImplSetEchoActionFieldValue (see page 5365)

54.1.28.2.36 TAdapterNamedField.SetFieldName Method

Sets the name that is used to reference this adapter field with server-side script.

Pascal

```
procedure SetFieldName(const AName: string);
```

C++

```
__fastcall SetFieldName(const UnicodeString AName);
```

Description

SetFieldName is typically called at design time to set the field name of a new adapter field. SetFieldName calls ImplSetFieldName (see page 5366).

See Also

ImplSetFieldName (see page 5366)

54.1.28.3 TAdapterNamedField Properties

The properties of the TAdapterNamedField class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5368)	Returns the parent adapter component.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.

54.1.28.3.1 TAdapterNamedField.ActionOrigValue Property

Retrieve the original value of this adapter field.

Pascal

```
property ActionOrigValue: IActionFieldValue;
```

C++

```
__property IActionFieldValue ActionOrigValue;
```

Description

Some adapter fields will include their original value as a hidden field in an HTML form. The original value can be compared with

the current value to verify that the current value hasn't been changed by another user between the time that the HTML form was generated and the HTML form was submitted. Use this property when executing an adapter action to retrieve the original value for this adapter field at the time that the HTML form was generated.

See Also

FieldFlags

54.1.28.3.2 TAdapterNamedField.ActionValue Property

Retrieve the value for this adapter field entered by the end-user.

Pascal

```
property ActionValue: IActionFieldValue;
```

C++

```
__property IActionFieldValue ActionValue;
```

Description

The ActionValue property retrieves the value(s) entered by the end-user and passed in the HTTP request when an HTML form is submitted. When executing an adapter action, use this property to retrieve the value(s) entered by the end-user.

54.1.28.3.3 TAdapterNamedField.Adapter Property

Returns the parent adapter component.

Pascal

```
property Adapter: TCustomAdapter;
```

C++

```
__property TCustomAdapter Adapter;
```

Description

Use this property to access the parent adapter component. The type of this class is TComponent because adapter components are not required to descend from a common adapter base class. Typically the adapter component is accessed through interfaces.

54.1.28.3.4 TAdapterNamedField.EchoActionFieldValue Property

Used to echo the value(s) entered by the end-user when generating an HTML form.

Pascal

```
property EchoActionFieldValue: Boolean;
```

C++

```
__property Boolean EchoActionFieldValue;
```

Description

The EchoActionFieldValue property is used to alter the source of the adapter fields value. When this flag is set, the adapter field retrieves it's value from the HTTP request. When this flag is clear, the adapter field retrieves it's current value. Typically, this flag is set after an error occurs while processing an adapter action. Setting this flag results in generated HTML that displays values entered by the end-user. This allows the end-user to make corrections and resubmit the form.

See Also

SetEchoActionFieldValue (see page 5366)

54.1.28.3.5 TAdapterNamedField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.28.3.6 TAdapterNamedField.ModifyAccess Property

Identifies the rights required to update this field.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this field. When server-side script accesses the CanModify property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to updates an adapter field that does not allow access.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

54.1.28.3.7 TAdapterNamedField.Value Property

Provides the value of an adapter field as a variant.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

The Value property returns the value of an adapter field as a Variant (see page 3421). Server-side script can access this value using the Value property. Note that descendant adapter fields support multiple values per field, files and images. For these fields, the Value property will not correctly represent the value of the adapter field.

See Also

GetValueCount (see page 5427)

GetValues (see page 5427)

54.1.28.3.8 TAdapterNamedField.ViewAccess Property

Identifies the rights required to view this field.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view this field. When server-side script accesses the CanView property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to view an adapter field that does not allow access.

See Also

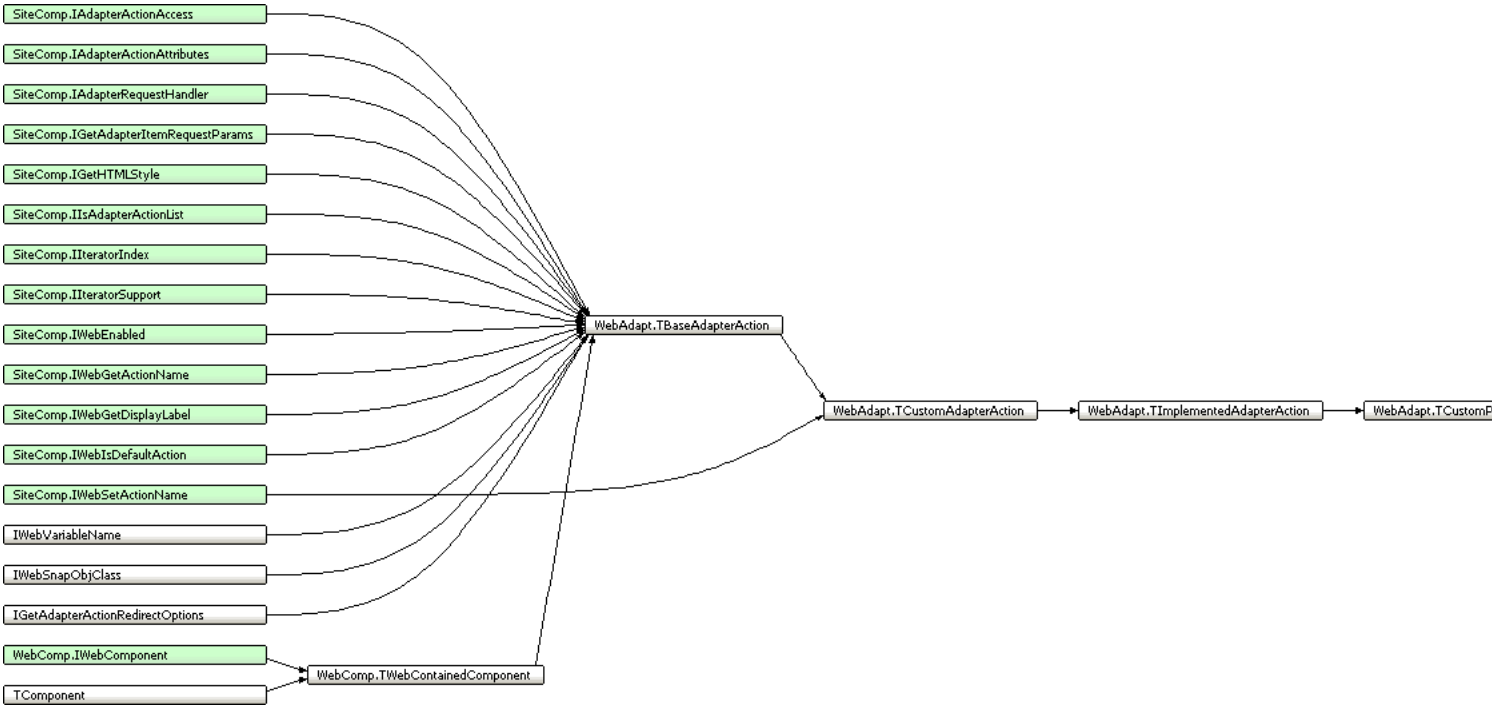
TWebUserList (see page 6796)

HasViewAccess (see page 5553)

54.1.29 TAdapterNextPageAction Class

TAdapterNextPageAction implements the next page command for TDefaultFieldsPagedAdapter (see page 5747).

Class Hierarchy



Pascal

```
TAdapterNextPageAction = class(TCustomAdapterPageAction);
```

C++

```
class TAdapterNextPageAction : public TCustomAdapterPageAction;
```

File

WebAdapt

Description

Add TAdapterNextPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 5747) such as TPagedAdapter (see page 5844).

See Also

TCustomAdapterPageAction (see page 5656)

TDefaultFieldsPagedAdapter (see page 5747)

TAdapterPrevPageAction (see page 5396)

TAdapterGotoPageAction (see page 5291)

TPagedAdapter (see page 5844)

54.1.29.1 TAdapterNextPageAction Members

The following tables list the members exposed by TAdapterNextPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5727)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5374)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5375)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 5661)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.

	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetNewPage (see page 5375)	Indicates the page to display when this action is executed.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 5662)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5662)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplIsDefaultAction (see page 5663)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5663)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.

	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Adapter (see page 5727)	Returns the parent adapter of this adapter action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.

	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5377)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5378)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5378)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5378)	Called before the adapter action response is generated.
	OnGetParams (see page 5379)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5376)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5376)	Defines the rights required to execute this action.
	RedirectOptions (see page 5377)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.29.2 TAdapterNextPageAction Methods

The methods of the TAdapterNextPageAction class are listed here.

Protected Methods

	Name	Description
V	GetDefaultActionName (see page 5374)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5375)	Provides the default display label for this adapter action.
V	GetNewPage (see page 5375)	Indicates the page to display when this action is executed.

54.1.29.2.1 TAdapterNextPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "NextPage".

See Also

GetActionName (see page 5455)

GetDefaultActionName (see page 5456)

54.1.29.2.2 TAdapterNextPageAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

GetDefaultDisplayLabel returns ">>". This displays as '>>' in the browser.

See Also

DisplayLabel (see page 5474)

54.1.29.2.3 TAdapterNextPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

GetNewPage determines whether the current page can be incremented. If so, the AValue parameter is modified and true is returned. Otherwise, false is returned.

See Also

GetNewPage (see page 5661)

RecordCount (see page 5761)

PageSize (see page 5761)

CurrentPage (see page 5760)

54.1.29.3 TAdapterNextPageAction Properties

The properties of the TAdapterNextPageAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5376)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5376)	Defines the rights required to execute this action.
	RedirectOptions (see page 5377)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.29.3.1 TAdapterNextPageAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5465)

SetDisplayLabel (see page 5472)

54.1.29.3.2 TAdapterNextPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6796)

HasExecuteAccess (see page 5552)

54.1.29.3.3 TAdapterNextPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

54.1.29.4 TAdapterNextPageAction Events

The events of the TAdapterNextPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5377)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5378)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5378)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5378)	Called before the adapter action response is generated.
	OnGetParams (see page 5379)	Specifies name/value pairs needed to execute this adapter action.

54.1.29.4.1 TAdapterNextPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.29.4.2 TAdapterNextPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.29.4.3 TAdapterNextPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.29.4.4 TAdapterNextPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains

the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.29.4.5 TAdapterNextPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

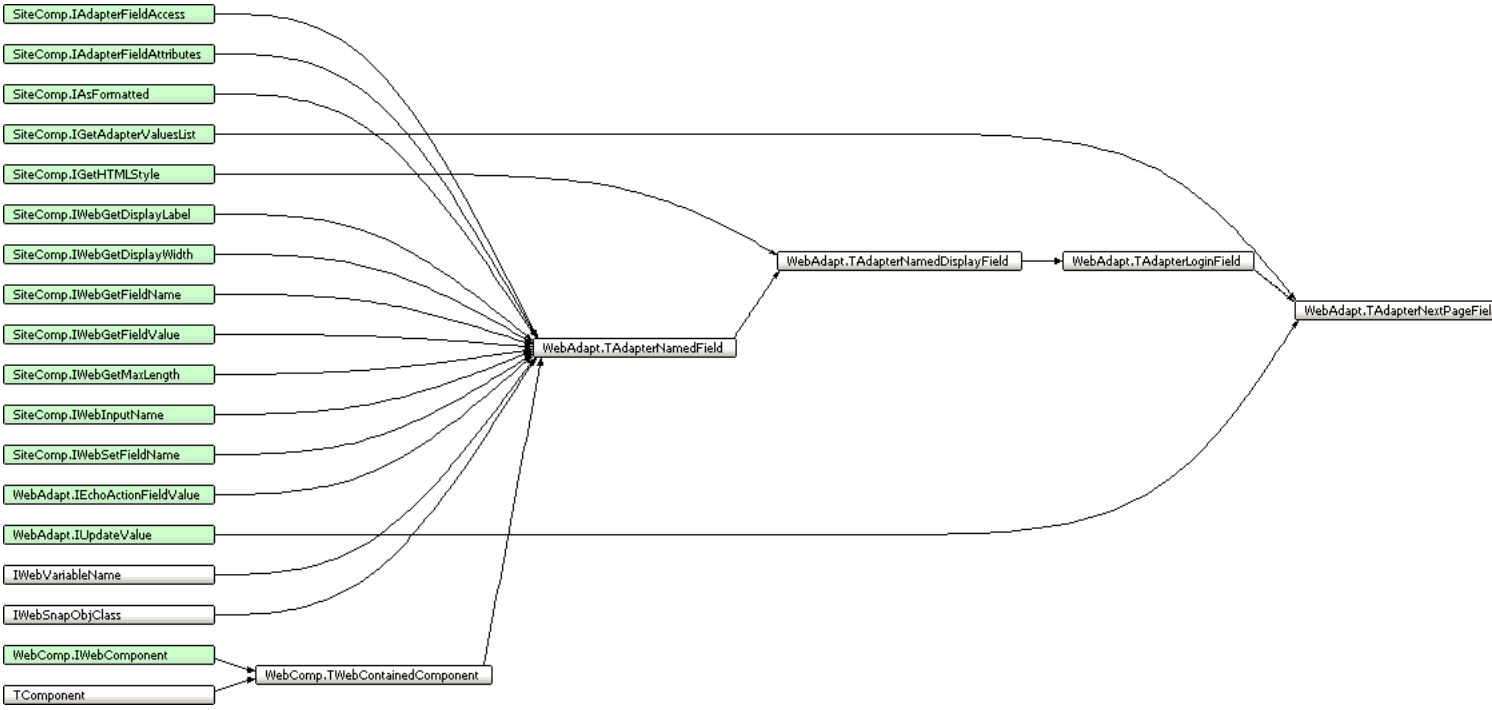
See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.30 TAdapterNextPageField Class

The TAdapterNextPageField navigates to the next page of data.

Class Hierarchy



Pascal

```
TAdapterNextPageField = class(TAdapterLoginField, IGetAdapterValuesList, IUpdateValue);
```

C++

```
class TAdapterNextPageField : public TAdapterLoginField, public IGetAdapterValuesList, public
IUpdateValue;
```

File

WebAdapt

Description

TAdapterNextPageField is one of the components that is used by TPagedAdapter (see page 5844) along with the TAdapterNextPageAction (see page 5370) to navigate to the next page of data.

The TPagedAdapter (see page 5844) PageSize property indicates the number of rows displayed on a page. When the PageSize is not 0, the TPagedAdapter (see page 5844) component automatically adds built-in actions for moving between pages. These actions are TAdapterGotoPageAction (see page 5291), TAdapterPrevPageAction (see page 5396), and TAdapterNextPageAction (see page 5370). The TAdapterPageProducer can be used to generate HTML hyperlinks or buttons for these actions.

See Also

TPagedAdapter (see page 5844)

TAdapterPrevPageAction (see page 5396)

TAdapterGotoPageAction (see page 5291)

TCustomAdapterPageAction (see page 5656)

54.1.30.1 TAdapterNextPageField Members

The following tables list the members exposed by TAdapterNextPageField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5384)	Returns a values list adapter containing the names of the published pages.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5384)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5384)	Returns the echo flag for this adapter field.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5385)	Returns the page name to be displayed after login.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5385)	Sets the echo flag for this adapter field.

	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5386)	Updates the parent adapter's next page property with the value entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 5383)	Frees an instance of TAdapterNextPageField (see page 5379).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
R	Adapter (see page 5323)	Returns the parent adapter of this adapter field.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 5386)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5386)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5387)	Identifies the name of this field when using server-side script.

54.1.30.2 TAdapterNextPageField.Destroy Destructor

Frees an instance of TAdapterNextPageField (see page 5379).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterNextPageField();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TAdapterNextPageField (see page 5379).

54.1.30.3 TAdapterNextPageField Methods

The methods of the TAdapterNextPageField class are listed here.

Protected Methods

	Name	Description
	GetAdapterValuesList (see page 5384)	Returns a values list adapter containing the names of the published pages.
V	GetInputStyleType (see page 5384)	Suggests an HTML input type for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5384)	Returns the echo flag for this adapter field.
V	ImplGetValue (see page 5385)	Returns the page name to be displayed after login.

	ImplSetEchoActionFieldValue (see page 5385)	Sets the echo flag for this adapter field.
	UpdateValue (see page 5386)	Updates the parent adapter's next page property with the value entered by the end-user.

54.1.30.3.1 TAdapterNextPageField.GetAdapterValuesList Method

Returns a values list adapter containing the names of the published pages.

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

GetAdapterValuesList returns a reference to TLogicalPageList (see page 5819). This function will be called when server-side script references the ValuesList property of this adapter field. Typically, server-side script uses the ValuesList property to populate an HTML select list with the names of published pages. The end-user may select one of these pages to go to after login.

See Also

IValuesListAdapter (see page 1072)

54.1.30.3.2 TAdapterNextPageField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType. GetInputStyleType returns htmliSelect indicating that a single selection list HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 5347)

TAdapterInputHTMLElementType (see page 1124)

54.1.30.3.3 TAdapterNextPageField.ImplGetEchoActionFieldValue Method

Returns the echo flag for this adapter field.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

ImplGetEchoActionFieldValue returns the value of an instance variable.

See Also

ImplGetEchoActionFieldValue (see page 5362)

EchoActionFieldValue (see page 5368)

ImplGetValue (see page 5385)

ImplSetEchoActionFieldValue (see page 5385)

54.1.30.3.4 TAdapterNextPageField.ImplGetValue Method

Returns the page name to be displayed after login.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

If the EchoActionFieldValue (see page 5368) property is true, then the page name selected or entered by end-user is returned. If the EchoActionFieldValue (see page 5368) property is false then the parent adapter's NextPage property is returned. The value returned by ImplGetValue is used when server-side script references the EditText property of this adapter field while generating the login form.

See Also

EchoActionFieldValue (see page 5368)

ActionValue (see page 5368)

NextPage (see page 5713)

54.1.30.3.5 TAdapterNextPageField.ImplSetEchoActionFieldValue Method

Sets the echo flag for this adapter field.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplSetEchoActionFieldValue sets an internal variable to true. The echo flag is typically set by an adapter action when an error occurs while processing an HTTP request. The purpose of the echo flag is to preserve the value entered by the end-user. When the echo flag is true, ImplGetValue (see page 5385) returns the value entered by the end-user rather than the value of the adapter field.

See Also

ImplSetEchoActionFieldValue (see page 5365)

EchoActionFieldValue (see page 5368)

ImplGetValue (see page 5385)

ImplGetEchoActionFieldValue (see page 5384)

54.1.30.3.6 TAdapterNextPageField.UpdateValue Method

Updates the parent adapter's next page property with the value entered by the end-user.

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

Typically a single selection list HTML element is displayed on the login form. The list contains all of the published pages. The HTTP request will contain the page name selected by the end-user.

See Also

NextPage (see page 5713)

IActionRequest

54.1.30.4 TAdapterNextPageField Properties

The properties of the TAdapterNextPageField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5386)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5386)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5387)	Identifies the name of this field when using server-side script.

54.1.30.4.1 TAdapterNextPageField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.30.4.2 TAdapterNextPageField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.30.4.3 TAdapterNextPageField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

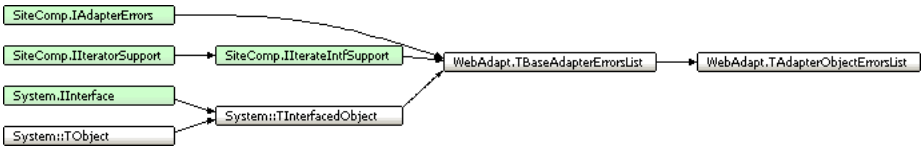
```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.31 TAdapterObjectErrorsList Class

Class Hierarchy



Pascal

```
TAdapterObjectErrorsList = class(TBaseAdapterErrorsList);
```

C++

```
class TAdapterObjectErrorsList : public TBaseAdapterErrorsList;
```

File

WebAdapt

Description

This is class WebAdapt.TAdapterObjectErrorsList.

54.1.31.1 TAdapterObjectErrorsList Members

The following tables list the members exposed by TAdapterObjectErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	EndIterator (see page 5483)	Signals the end of the iterating operation.
	GetCurrentIntf (see page 5483)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5484)	Returns a particular error.
	GetErrorCount (see page 5484)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5484)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5484)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 5485)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
V	ImplDefineLabel (see page 5390)	This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.
V	ImplGetError (see page 5390)	This is ImplGetError, a member of class TAdapterObjectErrorsList.
V	ImplGetErrorCount (see page 5390)	This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.
V	ImplGetObjectErrors (see page 5486)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
V	ImplHasObjectErrors (see page 5486)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 5486)	Advances to the objects next item.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5486)	Positions the object on its first item and starts the iteration process.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AddError (see page 5481)	This is the overview for the AddError method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 996)	Clears the errors list.
	ClearErrors (see page 5482)	Clear error list.
	Create (see page 5389)	This is Create, a member of class TAdapterObjectErrorsList.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.

	DefineLabel (see page 996)	Associates a label with all errors associated with a particular adapter field.
	DefineLabel (see page 5483)	Define error label for field.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetCurrentIntf (see page 1051)	This is GetCurrentIntf, a member of class IIterateIntfSupport.
	GetError (see page 996)	Returns a particular error.
	GetErrorCount (see page 997)	Return the number of errors in the errors list.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetObjectErrors (see page 997)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 997)	Indicates whether the errors list contains objects associated with a particular adapter field.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextIteration (see page 1056)	Advances to the objects next item.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ErrorCount (see page 998)	Returns the number of errors in the errors list.
R	ErrorCount (see page 5487)	Number of error objects contained.
R	Errors (see page 998)	Returns an error from the list.
R	Errors (see page 5487)	Access to error objects.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

54.1.31.2 TAdapterObjectErrorsList.Create Constructor

Pascal

```
constructor Create(AErrors: TAdapterErrorsList; const AName: string);
```

C++

```
__fastcall TAdapterObjectErrorsList(TAdapterErrorsList AErrors, const UnicodeString AName);
```

Description

This is Create, a member of class TAdapterObjectErrorsList.

54.1.31.3 TAdapterObjectErrorsList Methods

The methods of the TAdapterObjectErrorsList class are listed here.

Protected Methods

	Name	Description
	ImplDefineLabel (see page 5390)	This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.
	ImplGetError (see page 5390)	This is ImplGetError, a member of class TAdapterObjectErrorsList.
	ImplGetErrorCount (see page 5390)	This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.

54.1.31.3.1 TAdapterObjectErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); override;
```

C++

```
virtual __fastcall ImplDefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

This is ImplDefineLabel, a member of class TAdapterObjectErrorsList.

54.1.31.3.2 TAdapterObjectErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; override;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TAdapterObjectErrorsList.

54.1.31.3.3 TAdapterObjectErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

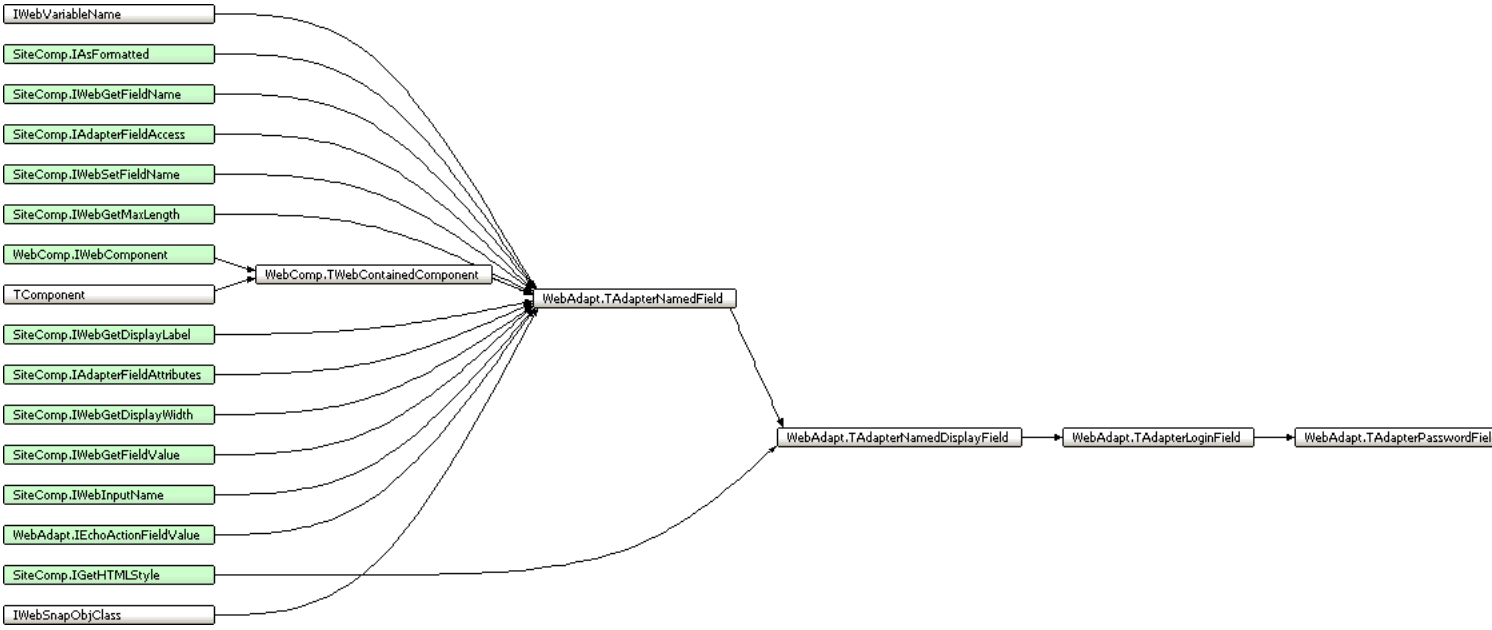
Description

This is ImplGetErrorCount, a member of class TAdapterObjectErrorsList.

54.1.32 TAdapterPasswordField Class

TAdapterPasswordField is an adapter field used to access the password entered by the end-user.

Class Hierarchy



Pascal

```
TAdapterPasswordField = class(TAdapterLoginField);
```

C++

```
class TAdapterPasswordField : public TAdapterLoginField;
```

File

WebAdapt

Description

TAdapterPasswordField is added to TLoginFormAdapter (see page 5824). TAdapterUserNameField (see page 5433), TAdapterPasswordField, TLoginFormAdapterLoginAction (see page 5835) and TAdapterNextPageField (see page 5379) are used with TLoginFormAdapter (see page 5824) to generate an HTML login form. When the HTML form is submitted, TAdapterPasswordField is used to extract the end user's password from the HTTP request.

See Also

TAdapterLoginField (see page 5320)

TLoginFormAdapter (see page 5824)

TAdapterUserNameField (see page 5433)

TAdapterNextPageField (see page 5379)

TLoginFormAdapterLoginAction (see page 5835)

54.1.32.1 TAdapterPasswordField Members

The following tables list the members exposed by TAdapterPasswordField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5394)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5395)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.

	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.

R	Adapter (see page 5323)	Returns the parent adapter of this adapter field.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 5395)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5395)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5396)	Identifies the name of this field when using server-side script.

54.1.32.2 TAdapterPasswordField Methods

The methods of the TAdapterPasswordField class are listed here.

Protected Methods

	Name	Description
V	GetInputStyleType (see page 5394)	Suggests an HTML input type for this adapter field.
V	ImplGetValue (see page 5395)	Returns the value associated with this adapter field.

54.1.32.2.1 TAdapterPasswordField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliPassword indicating that a password input HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 5347)

TAdapterInputHTMLElementType (see page 1124)

54.1.32.2.2 TAdapterPasswordField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns an empty string. Server-side script accesses the result of this function using the EditText or Value (see page 5369) property of the adapter field. These server-side script properties may be used to generate the HTML login form. ImplGetValue returns an empty string so that the generated login form's password field will be blank. When the login form is submitted, the ActionValue (see page 5368) property can be used to access the password entered by the end-user.

See Also

ActionValue (see page 5368)

54.1.32.3 TAdapterPasswordField Properties

The properties of the TAdapterPasswordField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5395)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5395)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5396)	Identifies the name of this field when using server-side script.

54.1.32.3.1 TAdapterPasswordField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.32.3.2 TAdapterPasswordField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.32.3.3 TAdapterPasswordField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

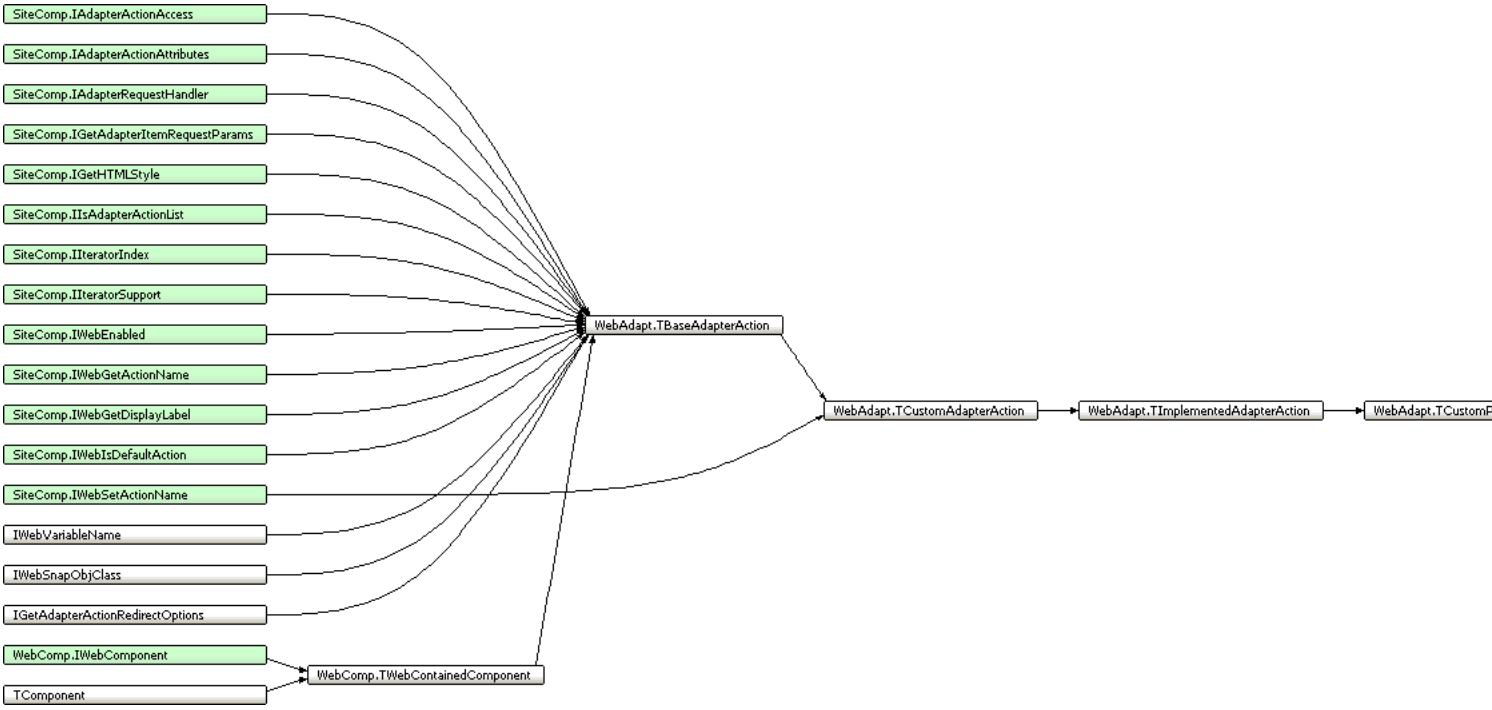
Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.33 TAdapterPrevPageAction Class

TAdapterPrevPageAction implements the previous page command for TDefaultFieldsPagedAdapter (see page 5747).

Class Hierarchy



Pascal

```
TAdapterPrevPageAction = class(TCustomAdapterPageAction);
```

C++

```
class TAdapterPrevPageAction : public TCustomAdapterPageAction;
```

File

WebAdapt

Description

Add TAdapterPrevPageAction to a descendant class of TDefaultFieldsPagedAdapter (see page 5747) such as TPagedAdapter (see page 5844).

See Also

TCustomAdapterPageAction (see page 5656)

TDefaultFieldsPagedAdapter (see page 5747)

TAdapterNextPageAction (see page 5370)

TAdapterGotoPageAction (see page 5291)

TPagedAdapter (see page 5844)

54.1.33.1 TAdapterPrevPageAction Members

The following tables list the members exposed by TAdapterPrevPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5727)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5401)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5401)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
V	GetDisplayStyleType (see page 5661)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
V	GetNewPage (see page 5402)	Indicates the page to display when this action is executed.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5662)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5662)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.

	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplIsDefaultAction (see page 5663)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5663)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Adapter (see page 5727)	Returns the parent adapter of this adapter action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5404)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5404)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5404)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5405)	Called before the adapter action response is generated.
	OnGetParams (see page 5405)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5402)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5403)	Defines the rights required to execute this action.
	RedirectOptions (see page 5403)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.33.2 TAdapterPrevPageAction Methods

The methods of the TAdapterPrevPageAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5401)	Gets the name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 5401)	Provides the default display label for this adapter action.
	GetNewPage (see page 5402)	Indicates the page to display when this action is executed.

54.1.33.2.1 TAdapterPrevPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "PrevPage".

See Also

GetActionName (see page 5455)

GetDefaultActionName (see page 5456)

54.1.33.2.2 TAdapterPrevPageAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

GetDefaultDisplayLabel returns "<<". This displays as '<<' in the browser.

See Also

DisplayLabel (see page 5474)

54.1.33.2.3 TAdapterPrevPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; override;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

GetNewPage determines whether the current page can be incremented. If so, the AValue parameter is modified and true is returned. Otherwise, false is returned.

See Also

GetNewPage (see page 5661)

RecordCount (see page 5761)

PageSize (see page 5761)

CurrentPage (see page 5760)

54.1.33.3 TAdapterPrevPageAction Properties

The properties of the TAdapterPrevPageAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5402)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5403)	Defines the rights required to execute this action.
	RedirectOptions (see page 5403)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.33.3.1 TAdapterPrevPageAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side

script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5465)

SetDisplayLabel (see page 5472)

54.1.33.3.2 TAdapterPrevPageAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6796)

HasExecuteAccess (see page 5552)

54.1.33.3.3 TAdapterPrevPageAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

54.1.33.4 TAdapterPrevPageAction Events

The events of the TAdapterPrevPageAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5404)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5404)	Called after the adapter action response is generated.

	OnBeforeExecute (see page 5404)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5405)	Called before the adapter action response is generated.
	OnGetParams (see page 5405)	Specifies name/value pairs needed to execute this adapter action.

54.1.33.4.1 TAdapterPrevPageAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.33.4.2 TAdapterPrevPageAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.33.4.3 TAdapterPrevPageAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.33.4.4 TAdapterPrevPageAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.33.4.5 TAdapterPrevPageAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

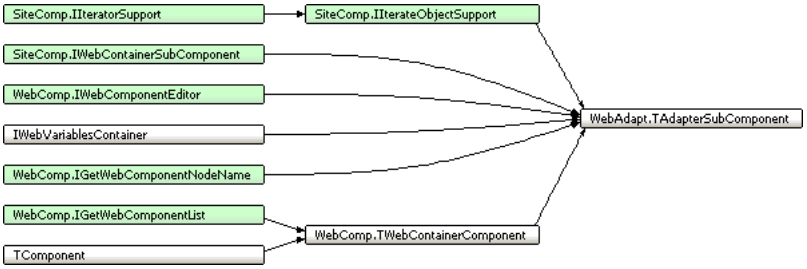
See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.34 TAdapterSubComponent Class

TAdapterSubComponent is the base class for objects that represent a collection of actions or fields that belong to an adapter.

Class Hierarchy



Pascal

```
TAdapterSubComponent = class(TWebContainerComponent, IWebContainerSubComponent,
IWebComponentEditor, IWebVariablesContainer, IGetWebComponentNodeName, IIterateObjectSupport);
```

C++

```
class TAdapterSubComponent : public TWebContainerComponent, public IWebContainerSubComponent,
public IWebComponentEditor, public IWebVariablesContainer, public IGetWebComponentNodeName,
public IIterateObjectSupport;
```

File

WebAdapt

Description

Adapter (see page 5418) components use a TAdapterSubComponent descendant to represent the list of actions they can execute or the list of fields whose values they support. Each TAdapterSubComponent is specialized for a specific type of object (actions or fields), and represents either the set of explicitly assigned actions or fields, or the set that the adapter contains by default.

TAdapterSubComponent introduces support for:

Associating the collection with an adapter component.

Assigning variable names to the child items in the collection. These are the names of variables that server-side script can use to refer to the actions or fields.

Letting server-side script use an enumerator to iterate over all the child items in the collection.

Working with the WebSnap designers at design time.

See Also

TAdapterActions (see page 5209)

TAdapterFields (see page 5276)

TAdapterDefaultActions (see page 5231)

TAdapterDefaultFields (see page 5237)

54.1.34.1 TAdapterSubComponent Members

The following tables list the members exposed by TAdapterSubComponent.

Protected Methods

	Name	Description
	CanAddClass (see page 5408)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5409)	Stops the TAdapterSubComponent (see page 5406) object from iterating over the Web items it lists.
V	FindNamedVariable (see page 5409)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5410)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5410)	Returns the Web module that contains a specified item from this list.
V	GetChildren (see page 6065)	Executes a callback for each child Web item.
	GetCurrentObject (see page 5411)	Returns the current field or action from the list when an enumerator is iterating over child items.
V	GetDefaultWebComponents (see page 5411)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5412)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5413)	Indicates the number of items in this list that can be accessed from server-side script.
V	GetVariablesInUse (see page 5413)	Returns the value of the VariablesInUse (see page 5419) property.
	GetWebComponentNodeName (see page 5413)	Returns a string that indicates the type of items in this collection.
	GetWebComponents (see page 6065)	Returns the list of Web items that this item manages.
V	ImplCanAddClass (see page 5414)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
V	ImplGetDefaultComponentList (see page 6066)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	ImplGetWebComponentNodeName (see page 5415)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	NextIteration (see page 5415)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5416)	Notifies the associated adapter when the list of items changes.
V	SetChildOrder (see page 6066)	Positions a child component in the list of child Web items.
V	SetDefaultWebComponents (see page 5416)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
V	SetWebParent (see page 5417)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5418)	Makes the first child item with a variable name the current object.

Protected Properties

	Name	Description
R	Adapter (see page 5418)	Indicates the adapter that uses this object to manage its fields or actions.
R	VariablesInUse (see page 5419)	Lists those fields or actions in the collection for which variables are defined.
R	WebContainerComponentHelper (see page 6067)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
V	Create (see page 6064)	Creates and initializes an instance of TWebContainerComponent (see page 6063).
V	Destroy (see page 6064)	Frees an instance of TWebContainerComponent (see page 6063).
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
V	GetParentComponent (see page 5412)	Returns the associated adapter.
V	HasParent (see page 5414)	Indicates whether this component has an associated adapter.
V	ReadState (see page 5416)	Reads the component's published properties, stored data, and owned components from a reader object.

	SetParentComponent (see page 5417)	Sets the Adapter (see page 5418) property to be the same as this component's parent.

54.1.34.2 TAdapterSubComponent Methods

The methods of the TAdapterSubComponent class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 5408)	Indicates whether an object of a specified type can be added to this collection.
	EndIterator (see page 5409)	Stops the TAdapterSubComponent (see page 5406) object from iterating over the Web items it lists.
	FindNamedVariable (see page 5409)	Returns the item from this list with a specified variable name.
	FindVariable (see page 5410)	Returns the item from this list with a specified variable name.
	FindVariableContainer (see page 5410)	Returns the Web module that contains a specified item from this list.
	GetCurrentObject (see page 5411)	Returns the current field or action from the list when an enumerator is iterating over child items.
	GetDefaultWebComponents (see page 5411)	Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.
	GetVariable (see page 5412)	Returns a child item from this list, given its position in the list of child items with variable names.
	GetVariableCount (see page 5413)	Indicates the number of items in this list that can be accessed from server-side script.
	GetVariablesInUse (see page 5413)	Returns the value of the VariablesInUse (see page 5419) property.
	GetWebComponentNodeName (see page 5413)	Returns a string that indicates the type of items in this collection.
	ImplCanAddClass (see page 5414)	Provides the underlying implementation for the CanAddClass (see page 5408) method.
	ImplGetWebComponentNodeName (see page 5415)	Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.
	NextIteration (see page 5415)	Changes the current object to the next item that has a variable name.
	NotifyAdapterChange (see page 5416)	Notifies the associated adapter when the list of items changes.
	SetDefaultWebComponents (see page 5416)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetWebParent (see page 5417)	Introduces a method to associate this component with an adapter.
	StartIterator (see page 5418)	Makes the first child item with a variable name the current object.

Public Methods

	Name	Description
	GetParentComponent (see page 5412)	Returns the associated adapter.
	HasParent (see page 5414)	Indicates whether this component has an associated adapter.
	ReadState (see page 5416)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 5417)	Sets the Adapter (see page 5418) property to be the same as this component's parent.

54.1.34.2.1 TAdapterSubComponent.CanAddClass Method

Indicates whether an object of a specified type can be added to this collection.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The IDE calls CanAddClass to determine what Web items can be added to this list. This method allows the IDE to limit the

actions or fields in the collection to those that are appropriate for the associated adapter.

AParent is the TAdapterSubComponent (see page 5406) descendant to which the Web item will be added.

AClass is the class of the proposed child item.

CanAddClass returns true if the TAdapterSubComponent (see page 5406) descendant can include a child of type AClass, false otherwise.

CanAddClass calls the virtual ImplCanAddClass (see page 5414) method to determine whether a class of Web item is valid.

See Also

ImplCanAddClass (see page 5414)

Adapter (see page 5418)

54.1.34.2.2 TAdapterSubComponent.EndIterator Method

Stops the TAdapterSubComponent (see page 5406) object from iterating over the Web items it lists.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

EndIterator is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called when the enumeration stops. In TAdapterSubComponent (see page 5406), EndIterator does nothing. This method exists only as a backstop for the standard call made from server-side script.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 5418) or NextIteration (see page 5415) method. TAdapterSubComponent (see page 5406) does not use the OwnerData parameter.

See Also

StartIterator (see page 5418)

NextIteration (see page 5415)

GetCurrentObject (see page 5411)

54.1.34.2.3 TAdapterSubComponent.FindNamedVariable Method

Returns the item from this list with a specified variable name.

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const UnicodeString AName);
```

Description

FindNamedVariable searches the list of child items that have variable names (the VariablesInUse (see page 5419) property) to find one with the specified name. The associated adapter component calls this method from its FindNamedField or FindNamedAction method to look up a field or action with a specified name.

AName is a variable name. Typically, this comes from server-side script that is looking for a child of an adapter with this name.

FindNamedVariable returns the item from the list with the specified variable name. If none of the items in the list have the name AName, FindNamedVariable returns nil (Delphi) or NULL (C++).

See Also

VariablesInUse (see page 5419)

FindVariable (see page 5410)

GetVariable (see page 5412)

GetVariableCount (see page 5413)

FindVariableContainer (see page 5410)

FindNamedField (see page 5544)

FindNamedAction (see page 5544)

54.1.34.2.4 TAdapterSubComponent.FindVariable Method

Returns the item from this list with a specified variable name.

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const UnicodeString AName);
```

Description

FindVariable method is called from server-side script to locate a field or action with a specified name. This method is identical to the FindNamedVariable (see page 5409) method, except that it is not virtual. Instead, FindVariable calls the virtual FindNamedVariable (see page 5409) method.

AName is a variable name.

FindNamedVariable (see page 5409) returns the item from the list with the specified variable name. If none of the items in the list have the name AName, FindNamedVariable (see page 5409) returns nil (Delphi) or NULL (C++).

See Also

VariablesInUse (see page 5419)

FindNamedVariable (see page 5409)

GetVariable (see page 5412)

GetVariableCount (see page 5413)

FindVariableContainer (see page 5410)

54.1.34.2.5 TAdapterSubComponent.FindVariableContainer Method

Returns the Web module that contains a specified item from this list.

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const UnicodeString AName);
```

Description

FindVariableContainer searches the list of child items that have variable names (the VariablesInUse (see page 5419) property) to find one with the specified name. It then returns the Web module that contains the component it located.

AName is the name of the action or field to locate.

FindVariableContainer returns the Web module that containse the specified item. If there is no item in the list with the name AName, FindVariableContainer returns nil (Delphi) or NULL (C++). If FindVariableContainer includes an item with the specified name, but is unable to determine what Web module contains that item, FindVariableContainer raises an EAdapterException (see page 5177) exception.

See Also

VariablesInUse (see page 5419)

FindNamedVariable (see page 5409)

GetVariable (see page 5412)

GetVariableCount (see page 5413)

EAdapterException (see page 5177)

54.1.34.2.6 TAdapterSubComponent.GetCurrentObject Method

Returns the current field or action from the list when an enumerator is iterating over child items.

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

Description

GetCurrentObject is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method returns the current child item.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 5418) or NextIteration (see page 5415) method. TAdapterSubComponent (see page 5406) does not use the OwnerData parameter.

See Also

StartIterator (see page 5418)

NextIteration (see page 5415)

EndIterator (see page 5409)

54.1.34.2.7 TAdapterSubComponent.GetDefaultWebComponents Method

Returns a list of actions or fields that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TAdapterSubComponent (see page 5406), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 5231) and TAdapterDefaultFields (see page 5237)) override this method to return the list of default fields or actions.

See Also

GetWebComponents (see page 6065)

TAdapterDefaultActions (see page 5231)

TAdapterDefaultFields (see page 5237)

54.1.34.2.8 TAdapterSubComponent.GetParentComponent Method

Returns the associated adapter.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TAdapterSubComponent (see page 5406) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the Adapter (see page 5418) property (the adapter that uses this component to represent the set of its fields or actions).

See Also

Adapter (see page 5418)

HasParent (see page 5414)

54.1.34.2.9 TAdapterSubComponent.GetVariable Method

Returns a child item from this list, given its position in the list of child items with variable names.

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

FindNamedVariable (see page 5409) indexes into the list of child items that have variable names (the VariablesInUse (see page 5419) property) and returns the item in the specified position.

I is the index of the child item, where 0 is the first item that can be reference from server-side script, 1 is the second item, and so on. The GetVariableCount (see page 5413) method returns an upper limit for this index.

See Also

VariablesInUse (see page 5419)

FindNamedVariable (see page 5409)

FindVariable (see page 5410)

GetVariableCount (see page 5413)

54.1.34.2.10 TAdapterSubComponent.GetVariableCount Method

Indicates the number of items in this list that can be accessed from server-side script.

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

GetVariableCount returns the number of child items that have variable names. This is the number of items in the VariablesInUse (see page 5419) proprety. Use GetVariableCount as an upper bound to the I parameter of the GetVariable (see page 5412) method.

See Also

VariablesInUse (see page 5419)

GetVariable (see page 5412)

54.1.34.2.11 TAdapterSubComponent.GetVariablesInUse Method

Returns the value of the VariablesInUse (see page 5419) property.

Pascal

```
function GetVariablesInUse: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetVariablesInUse();
```

Description

GetVariablesInUse is the protected read implementation for the VariablesInUse (see page 5419) property.

In TAdapterSubComponent (see page 5406), GetVariablesInUse calls the GetWebComponents method to return a list of all the items in the list. Descendant classes can override this method to return only those items that meet a certain criterion.

See Also

VariablesInUse (see page 5419)

GetWebComponents (see page 6065)

54.1.34.2.12 TAdapterSubComponent.GetWebComponentNodeName Method

Returns a string that indicates the type of items in this collection.

Pascal

```
function GetWebComponentNodeName: string;
```

C++

```
__fastcall UnicodeString GetWebComponentNodeName();
```

Description

GetWebComponentNodeName returns the label for the type of items in the collection. For example, if the TAdapterSubComponent (see page 5406) descendant holds the fields belonging to an adapter, GetWebComponentNodeName returns the string 'Fields'. If the TAdapterSubComponent (see page 5406) descendant holds the actions belonging to an adapter, GetWebComponentNodeName returns the string 'Actions'.

GetWebComponentNodeName delegates the task of supplying a name to the virtual ImplGetWebComponentNodeName (see page 5415) method.

See Also

ImplGetWebComponentNodeName (see page 5415)

54.1.34.2.13 TAdapterSubComponent.HasParent Method

Indicates whether this component has an associated adapter.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TAdapterSubComponent (see page 5406) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the Adapter (see page 5418) property is not nil (Delphi) or NULL (C++).

If the TAdapterSubComponent (see page 5406) object has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 5412)

Adapter (see page 5418)

54.1.34.2.14 TAdapterSubComponent.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass (see page 5408) method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass (see page 5408) method calls ImplCanAddClass to determine whether an object of a specified type can be added to this collection.

AParent is the TAdapterSubComponent (see page 5406) descendant to which the Web item will be added.

AClass is the class of the proposed child item.

ImplCanAddClass returns true if the TAdapterSubComponent (see page 5406) descendant can include a child of type AClass,

false otherwise.

In TAdapterSubComponent (see page 5406), ImplCanAddClass always returns false, disallowing any child items. Descendant classes override this method to conditionally allow those fields or actions that are appropriate to the associated adapter.

See Also

CanAddClass (see page 5408)

Adapter (see page 5418)

54.1.34.2.15 TAdapterSubComponent.ImplGetWebComponentNodeName Method

Provides the underlying implementation for the GetWebComponentNodeName (see page 5413) method.

Pascal

```
function ImplGetWebComponentNodeName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetWebComponentNodeName();
```

Description

The GetWebComponentNodeName (see page 5413) method calls ImplGetWebComponentNodeName to obtain a label for the type of items in the collection.

ImplGetWebComponentNodeName returns the name of the type of items in this list.

In TAdapterSubComponent (see page 5406), ImplGetWebComponentNodeName returns an empty string. Descendant classes created for specific types of elements override this method to return an appropriate string such as 'Fields' or 'Actions'.

See Also

GetWebComponentNodeName (see page 5413)

54.1.34.2.16 TAdapterSubComponent.NextIteration Method

Changes the current object to the next item that has a variable name.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

NextIteration is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called to move to the next item in the list. After a call to NextIteration, the GetCurrentObject (see page 5411) method returns the next item in the VariablesInUse (see page 5419) list.

OwnerData is a pointer to object-specific information returned by the last call to the StartIterator (see page 5418) or NextIteration method. TAdapterSubComponent (see page 5406) does not use the OwnerData parameter.

NextIteration returns true if it successfully moves to a new item, false if there was no addition item to which it could move.

See Also

StartIterator (see page 5418)

EndIterator (see page 5409)

GetCurrentObject (see page 5411)

VariablesInUse (see page 5419)

54.1.34.2.17 TAdapterSubComponent.NotifyAdapterChange Method

Notifies the associated adapter when the list of items changes.

Pascal

```
procedure NotifyAdapterChange;
```

C++

```
__fastcall NotifyAdapterChange();
```

Description

TAdapterSubComponent (see page 5406) descendants call NotifyAdapterChange internally when the items in this list change. NotifyAdapterChange passes this notification on to the associated adapter so that it can make any necessary adjustments.

See Also

Adapter (see page 5418)

NotifyAdapterChange (see page 5565)

54.1.34.2.18 TAdapterSubComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TAdapterSubComponent (see page 5406) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the Adapter (see page 5418) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 5412)

Adapter (see page 5418)

54.1.34.2.19 TAdapterSubComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); override;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents (see page 5411) supplies the default set of child items for this component. As implemented in TAdapterSubComponent (see page 5406), GetDefaultWebComponents (see page 5411) always returns nil (Delphi) or NULL (C++) because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 5231) and TAdapterDefaultFields (see page 5237)) override this method to return the list of default fields or actions.

SetDefaultWebComponents assigns the default set of child items for this component. As implemented in TWebContainerComponent (see page 6063), SetDefaultWebComponents does nothing because this class represents only a single list of items. Descendants that represent the default fields or actions of the associated adapter (TAdapterDefaultActions (see page 5231) and TAdapterDefaultFields (see page 5237)) override this method to store the list of default fields or actions.

See Also

GetDefaultWebComponents (see page 5411)

IsDefaultField (see page 6042)

54.1.34.2.20 TAdapterSubComponent.SetParentComponent Method

Sets the Adapter (see page 5418) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TAdapterSubComponent (see page 5406) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the Adapter (see page 5418) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 5414)

Adapter (see page 5418)

54.1.34.2.21 TAdapterSubComponent.SetWebParent Method

Introduces a method to associate this component with an adapter.

Pascal

```
procedure SetWebParent(const Value: TComponent); virtual;
```

C++

```
virtual __fastcall SetWebParent(const TComponent * Value);
```

Description

TAdapterSubComponent (see page 5406) introduces SetWebParent as a virtual method for setting the read-only Adapter (

see page 5418) property.

Value is the adapter component that acts as a parent to this collection.

In TAdapterSubComponent (see page 5406), SetWebParent does nothing. Descendant classes override this method to set the Adapter (see page 5418) property and populate the list with actions or fields appropriate to that adapter.

See Also

Adapter (see page 5418)

54.1.34.2.22 TAdapterSubComponent.StartIterator Method

Makes the first child item with a variable name the current object.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

StartIterator is part of the support that allows server-side script to use an enumerator to iterate over all the child items in the collection. This method is called to begin iterating over the items in the list. After a call to StartIterator, the GetCurrentObject (see page 5411) method returns the first item in the VariablesInUse (see page 5419) list.

OwnerData is a pointer to object-specific information returned that should be passed to the NextIteration (see page 5415) or EndIterator (see page 5409) method. TAdapterSubComponent (see page 5406) does not use the OwnerData parameter.

StartIterator returns true if it successfully moves to the first item, false if the VariablesInUse (see page 5419) property does not list any items.

See Also

NextIteration (see page 5415)

EndIterator (see page 5409)

GetCurrentObject (see page 5411)

VariablesInUse (see page 5419)

54.1.34.3 TAdapterSubComponent Properties

The properties of the TAdapterSubComponent class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5418)	Indicates the adapter that uses this object to manage its fields or actions.
	VariablesInUse (see page 5419)	Lists those fields or actions in the collection for which variables are defined.

54.1.34.3.1 TAdapterSubComponent.Adapter Property

Indicates the adapter that uses this object to manage its fields or actions.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Read (see page 3634) Adapter to access the adapter component that uses this collection. This TAdapterSubComponent (see page 5406) object is the value of the adapter's Data or Actions property.

See Also

Actions (see page 5572)

Data (see page 5572)

Actions (see page 5746)

Data (see page 5747)

54.1.34.3.2 TAdapterSubComponent.VariablesInUse Property

Lists those fields or actions in the collection for which variables are defined.

Pascal

```
property VariablesInUse: TWebComponentList;
```

C++

```
__property TWebComponentList VariablesInUse;
```

Description

Read (see page 3634) VariablesInUse to access the child items (fields or actions) that can be addressed from server-side script.

TAdapterSubComponent (see page 5406) implements VariablesInUse to include all of the fields or actions that have been added to the list. Descendant classes can override this property to provide a different set of items, such as the default fields or actions of the associated adapter.

See Also

FindNamedVariable (see page 5409)

FindVariable (see page 5410)

GetVariable (see page 5412)

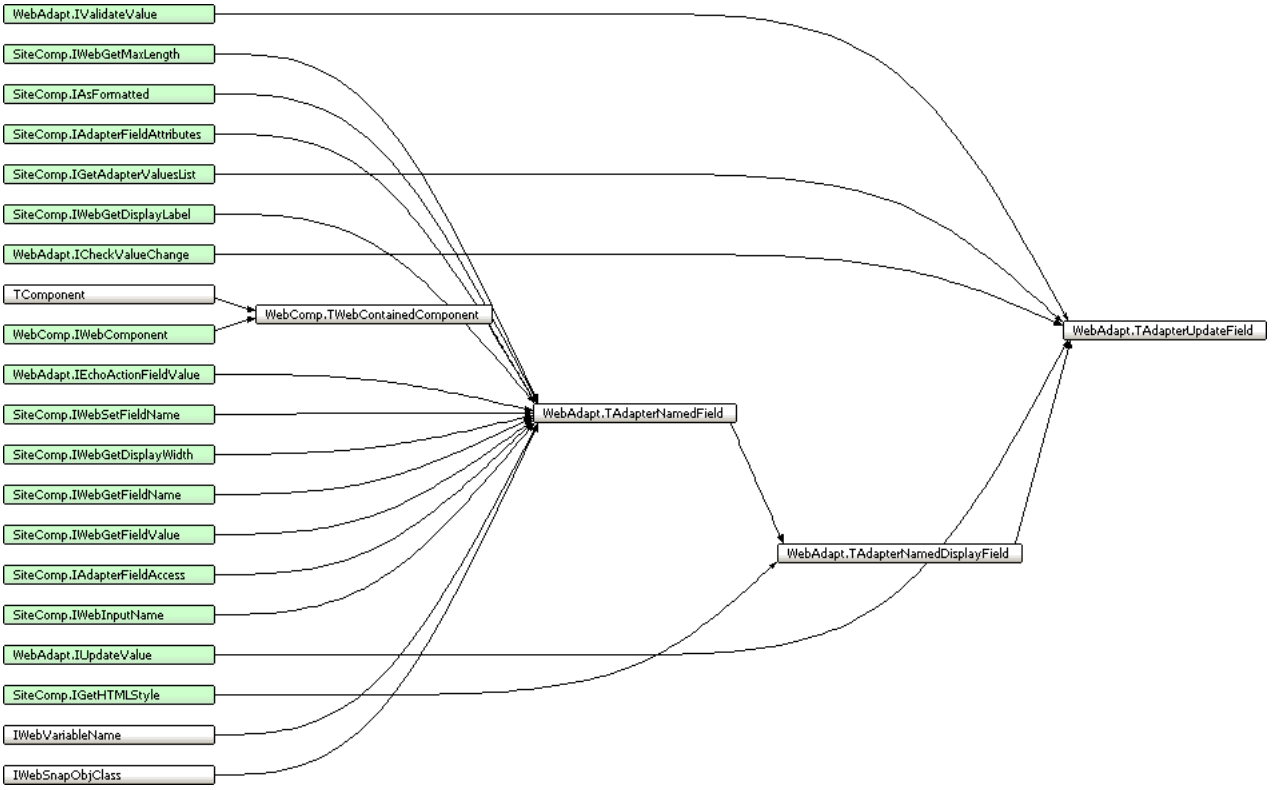
GetVariableCount (see page 5413)

FindVariableContainer (see page 5410)

54.1.35 TAdapterUpdateField Class

TAdapterUpdateField is a base class for adapter field components that support data entry into an HTML form.

Class Hierarchy



Pascal

```
TAdapterUpdateField = class(TAdapterNamedDisplayField, IGetAdapterValuesList, IUpdateValue,
IValidateValue, ICheckValueChange);
```

C++

```
class TAdapterUpdateField : public TAdapterNamedDisplayField, public IGetAdapterValuesList,
public IUpdateValue, public IValidateValue, public ICheckValueChange;
```

File

WebAdapt

Description

TAdapterUpdateField implements the interfaces necessary to support field validation and updating. Descendant classes support different types of data that can be entered by the user (for example, image upload, file upload, single value, multiple value, memo and Boolean).

See Also

TAdapterNamedDisplayField (see page 5343)

TCustomAdapterField (see page 5597)

TAdapter (see page 5190)

54.1.35.1 TAdapterUpdateField Members

The following tables list the members exposed by TAdapterUpdateField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.

	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.35.2 TAdapterUpdateField.Create Constructor

Creates an instance of TAdapterUpdateField (see page 5419).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterUpdateField(TComponent * AOwner);
```

Description

Do not call TAdapterUpdateField (see page 5419). It will be called automatically when the TAdapterUpdateField (see page 5419) object is created.

See Also

Create

54.1.35.3 TAdapterUpdateField Methods

The methods of the TAdapterUpdateField class are listed here.

Protected Methods

	Name	Description
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
V	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
V	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
V	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
V	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
V	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
V	Notification (see page 5431)	Clears references to a deleted component.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

54.1.35.3.1 TAdapterUpdateField.CheckValueChange Method

Checks the value(s) entered by the end-user against the value associated with this adapter field.

Pascal

```
function CheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CheckValueChange(IActionRequest AActionRequest, int AFieldIndex);
```

Description

CheckValueChange is called to determine whether the value(s) entered by the user is different from the value(s) provided by the adapter field. CheckValueChange calls the virtual function ImplCheckValueChange (see page 5428).

See Also

ImplCheckValueChange (see page 5428)

54.1.35.3.2 TAdapterUpdateField.ConvertValue Method

Retrieves a value entered by the end-user.

Pascal

```
function ConvertValue(AValue: IActionFieldValue): Variant; virtual;
```

C++

```
virtual __fastcall Variant ConvertValue(IActionFieldValue AValue);
```

Description

ConvertValue is called to retrieve, as a variant, the value entered by the user for this adapter field. ConvertValue using index 0 to retrieve a value from the IActionFieldValue (see page 5904) parameter. Typically, IActionFieldValue (see page 5904) will contain a string value. Descendants may override this function to caste the value into another type.

See Also

ConvertValue (see page 5594)

54.1.35.3.3 TAdapterUpdateField.ConvertValueCount Method

Retrieves the number of values entered by the end-user.

Pascal

```
function ConvertValueCount(AValue: IActionFieldValue): Integer; virtual;
```

C++

```
virtual __fastcall int ConvertValueCount(IActionFieldValue AValue);
```

Description

ConvertValueCount is called to retrieve the number of values entered by the user. ConvertValueCount returns 1. Descendants may override this method.

54.1.35.3.4 TAdapterUpdateField.ConvertValues Method

Retrieves a value entered by the end-user.

Pascal

```
function ConvertValues(AValue: IActionFieldValue; Index: Integer): Variant; virtual;
```

C++

```
virtual __fastcall Variant ConvertValues(IActionFieldValue AValue, int Index);
```

Description

ConvertValues is called to retrieve, as a variant, a particular value entered by the end-user for this adapter field.

54.1.35.3.5 TAdapterUpdateField.DoOnValidateValue Method

Validates field values entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); virtual;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue is called when the field value entered by the end-user is to be validated. DoOnValidateValue calls the OnValidateValue (see page 5433) event handler. DoOnValidateValue may be overridden by descendants.

See Also

OnValidateValue (see page 5433)

ImplValidateValue (see page 5430)

54.1.35.3.6 TAdapterUpdateField.GetAdapterValuesList Method

Provides an interface to a component that can be used to generate a list of possible values for this adapter field.

Pascal

```
function GetAdapterValuesList: IValuesListAdapter;
```

C++

```
__fastcall IValuesListAdapter GetAdapterValuesList();
```

Description

GetAdapterValuesList provides an interface to a component that can be used to generate a list of possible values for this adapter field. It does this by returning the value of the ValuesList (see page 5432) property.

See Also

IValuesListAdapter (see page 1072)

54.1.35.3.7 TAdapterUpdateField.GetInputStyleType Method

Indicates the preferred HTML element to use when editing this adapter field's value.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType returns htmliSelect if the ValuesList (see page 5432) property is not nil (Delphi) or NULL (C++). Otherwise, htmliTextInput is returned.

See Also

TAdapterInputHTMLElementType (see page 1124)

ValuesList (see page 5432)

54.1.35.3.8 TAdapterUpdateField.GetValueCount Method

Provides the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount returns the number of values provided by this adapter field. GetValueCount returns 1. Descendant classes override this method.

See Also

TAdapterMultiValueField (see page 5332)

54.1.35.3.9 TAdapterUpdateField.GetValues Method

Returns one of the values provided by this adapter field.

Pascal

```
function GetValues(I: Integer): Variant; virtual;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns one of the values provided by this adapter field. GetValues calls GetValue (see page 5359). Descendant classes override this method.

See Also

TAdapterMultiValueField (see page 5332)

54.1.35.3.10 TAdapterUpdateField.ImplAsFormatted Method

Returns a string representation of the adapter field's value that is appropriate for a read-only display.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplAsFormatted();
```

Description

ImplAsFormatted calls the OnGetDisplayText (see page 5433) event handler, if one is defined. Otherwise, ImplAsFormatted returns the adapter field's value converted to a string. Server-side script accesses the value returned by this function using the AsDisplayText property.

See Also

OnGetDisplayText (see page 5433)

GetValue (see page 5359)

54.1.35.3.11 TAdapterUpdateField.ImplCheckValueChange Method

Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange compares the number and content of values associated with the adapter field with the number and content of values entered by the end-user.

See Also

CheckValueChange (see page 5424)

54.1.35.3.12 TAdapterUpdateField.ImplGetDisplayLabel Method

Gets the preferred display label for this adapter field.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 5350) property.

See Also

DisplayLabel (see page 5350)

54.1.35.3.13 TAdapterUpdateField.ImplGetDisplayWidth Method

Returns the preferred display width for the HTML element used to edit or display this adapter field's value.

Pascal

```
function ImplGetDisplayWidth: Integer; override;
```

C++

```
virtual __fastcall int ImplGetDisplayWidth();
```

Description

ImplGetDisplayWidth returns the value of the DisplayWidth (see page 5350) property.

See Also

DisplayWidth (see page 5350)

54.1.35.3.14 TAdapterUpdateField.ImplGetEchoActionFieldValue Method

Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

Pascal

```
function ImplGetEchoActionFieldValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValue();
```

Description

Gets a flag that causes the adapter field to return the value(s) entered by the end-user.

See Also

ImplSetEchoActionFieldValue (see page 5429)

54.1.35.3.15 TAdapterUpdateField.ImplGetVariableName Method

Gets the name used to access this adapter field with server-side script.

Pascal

```
function ImplGetVariableName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetVariableName();
```

Description

ImplGetVariableName returns the value of the FieldName (see page 5369) property.

See Also

FieldName (see page 5369)

54.1.35.3.16 TAdapterUpdateField.ImplSetEchoActionFieldValue Method

Sets a flag that causes the adapter field to return the value(s) entered by the end-user.

Pascal

```
procedure ImplSetEchoActionFieldValue(AValue: Boolean); override;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValue(Boolean AValue);
```

Description

ImplSetEchoActionFieldValue is typically called when an error occurs during the execution of an adapter action. It allows the page generated after the error to display the values entered by the user rather than the action adapter field values.

54.1.35.3.17 TAdapterUpdateField.ImplSetFieldName Method

Sets the name used to access this adapter field with server-side script.

Pascal

```
procedure ImplSetFieldName(const Value: string); override;
```

C++

```
virtual __fastcall ImplSetFieldName(const UnicodeString Value);
```

Description

ImplSetVariableName sets the value of the FieldName (see page 5369) property.

See Also

FieldName (see page 5369)

54.1.35.3.18 TAdapterUpdateField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue is a placeholder. Descendants override this method.

See Also

UpdateValue (see page 5431)

54.1.35.3.19 TAdapterUpdateField.ImplValidateValue Method

Validates values(s) entered by the end-user.

Pascal

```
procedure ImplValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); virtual;
```

C++

```
virtual __fastcall ImplValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplValidateValue calls the virtual function DoOnValidateValue (see page 5426) after extracting an IActionFieldValue (see page 5904) that contains the value(s) entered by the user. The AFieldIndex parameter identifies that location of the IActionFieldValue (see page 5904) in the AActionRequest.

See Also

ValidateValue (see page 5431)

DoOnValidateValue (see page 5426)

IActionRequest

54.1.35.3.20 TAdapterUpdateField.IsMultiValueField Method

Indicates whether this adapter field can support multiple values.

Pascal

```
function IsMultiValueField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsMultiValueField();
```

Description

IsMultiValueField return true to indicate that this adapter field can provide multiple values. IsMultiValueField returns false. Descendants may override this function.

See Also

GetValueCount (see page 5427)

GetValues (see page 5427)

TAdapterMultiValueField (see page 5332)

54.1.35.3.21 TAdapterUpdateField.Notification Method

Clears references to a deleted component.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when a component is inserted or deleted. TAdapterUpdateField (see page 5419) clears the ValuesList (see page 5432) property when the associated component is deleted.

54.1.35.3.22 TAdapterUpdateField.UpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

UpdateValue calls the virtual method ImplUpdateValue (see page 5430).

See Also

ImplUpdateValue (see page 5430)

54.1.35.3.23 TAdapterUpdateField.ValidateValue Method

Validates values(s) entered by the end-user.

Pascal

```
procedure ValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ValidateValue calls the virtual function ImplValidateValue (see page 5430).

See Also

ImplValidateValue (see page 5430)

IActionRequest

54.1.35.4 TAdapterUpdateField Properties

The properties of the TAdapterUpdateField class are listed here.

Public Properties

	Name	Description
	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.

54.1.35.4.1 TAdapterUpdateField.Value Property

Retrieves the value associated with this adapter field.

Pascal

```
property Value: Variant;
```

C++

```
__property Variant Value;
```

Description

GetValue (see page 5359) calls the GetValue (see page 5359) method to retrieve the value of the adapter field as a variant.

See Also

GetValue (see page 5359)

54.1.35.4.2 TAdapterUpdateField.ValuesList Property

Associates a list of values with the adapter field.

Pascal

```
property ValuesList: IValuesListAdapter;
```

C++

```
__property IValuesListAdapter ValuesList;
```

Description

Assign a component to the ValuesList property to define possible values for the adapter field. Server-side script can be used to enumerate the name/value pairs of the ValuesList component to generate HTML select lists, check boxes, or radio buttons.

See Also

IValuesListAdapter (see page 1072)

TStringsValuesList (see page 5857)

TDataSetValuesList

54.1.35.5 TAdapterUpdateField Events

The events of the TAdapterUpdateField class are listed here.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

54.1.35.5.1 TAdapterUpdateField.OnGetDisplayText Event

Event handler called to retrieve a string representation of the adapter field's value.

Pascal

```
property OnGetDisplayText: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetDisplayText;
```

Description

The OnGetDisplayText event handler is called to retrieve a string representation an adapter field's value appropriate for a read-only display. If OnGetDisplayText is nil (Delphi) or NULL (C++), then the value of the adapter field will be used (after conversion to a string).

See Also

ImplAsFormatted (see page 5427)

54.1.35.5.2 TAdapterUpdateField.OnValidateValue Event

Event handler called to validate value(s) entered by the end-user.

Pascal

```
property OnValidateValue: TAdapterFieldValidateValueEvent;
```

C++

```
__property TAdapterFieldValidateValueEvent OnValidateValue;
```

Description

This event handler is called by DoOnValidateValue (see page 5426) to validate the value(s) entered by the end-user. The event handler must raise an exception to indicate that the value(s) are not valid.

See Also

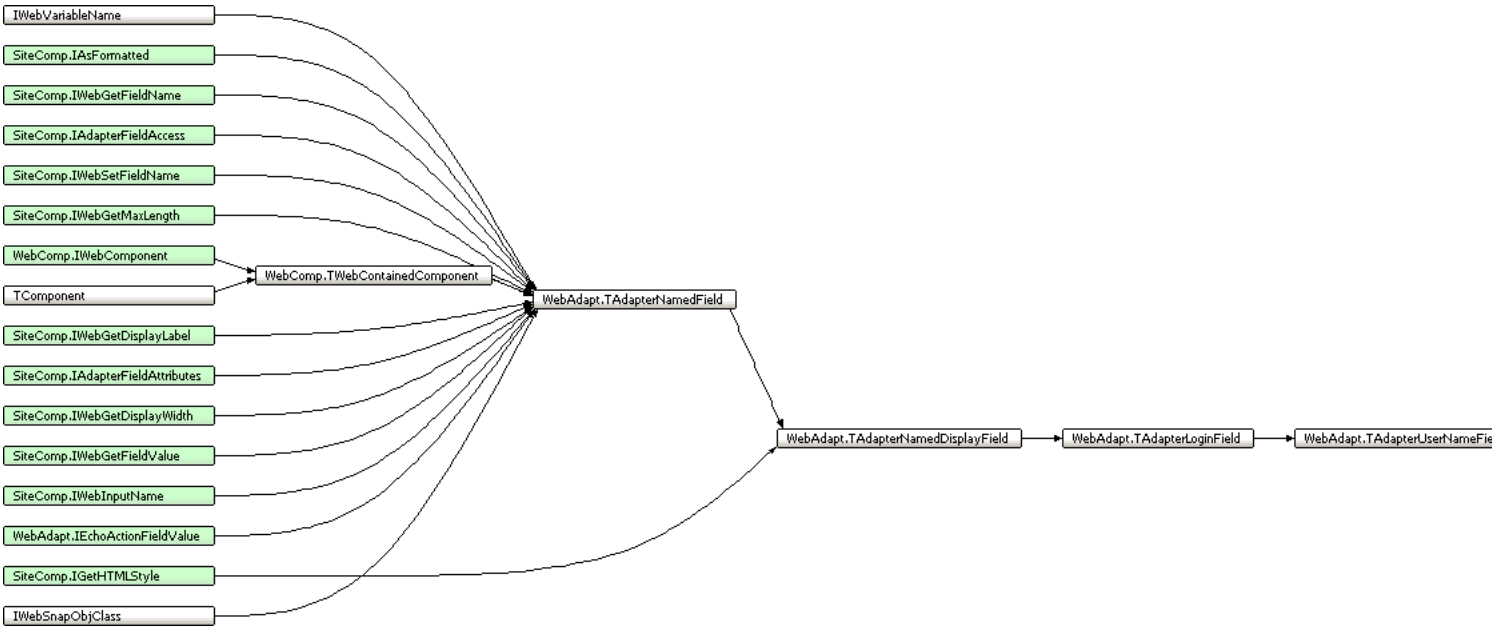
DoOnValidateValue (see page 5426)

IActionFieldValue

54.1.36 TAdapterUserNameField Class

TAdapterUserNameField is an adapter field used to access the user name entered by the end-user.

Class Hierarchy



Pascal

```
TAdapterUserNameField = class(TAdapterLoginField);
```

C++

```
class TAdapterUserNameField : public TAdapterLoginField;
```

File

WebAdapt

Description

TAdapterUserNameField is added to TLoginFormAdapter (see page 5824). TAdapterUserNameField, TAdapterPasswordField (see page 5391), TLoginFormAdapterLoginAction (see page 5835) and TAdapterNextPageField (see page 5379) are used with TLoginFormAdapter (see page 5824) to generate an HTML login form. When the HTML form is submitted, TAdapterUserNameField is used to extract the end user's name from the HTTP request.

See Also

TAdapterLoginField (see page 5320)

TLoginFormAdapter (see page 5824)

TAdapterPasswordField (see page 5391)

TAdapterNextPageField (see page 5379)

TLoginFormAdapterLoginAction (see page 5835)

54.1.36.1 TAdapterUserNameField Members

The following tables list the members exposed by TAdapterUserNameField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5437)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.

	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
	Adapter (see page 5323)	Returns the parent adapter of this adapter field.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.

	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

Published Properties

	Name	Description
	DisplayLabel (see page 5438)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5438)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5438)	Identifies the name of this field when using server-side script.

54.1.36.2 TAdapterUserNameField Methods

The methods of the TAdapterUserNameField class are listed here.

Protected Methods

	Name	Description
V	ImplGetValue (see page 5437)	Returns the value associated with this adapter field.

54.1.36.2.1 TAdapterUserNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns an empty string. Server-side script accesses the result of this function using the EditText or Value (see page 5369) property of the adapter field. These server-side script properties may be used to generate the HTML login form. ImplGetValue returns an empty string so that the generated login form's user name field will be blank. When the login form is submitted, the ActionValue (see page 5368) property can be used to access the user name entered by the end-user.

See Also

ActionValue (see page 5368)

54.1.36.3 TAdapterUserNameField Properties

The properties of the TAdapterUserNameField class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5438)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5438)	Suggests a display width, in characters, for this adapter field.
	FieldName (see page 5438)	Identifies the name of this field when using server-side script.

54.1.36.3.1 TAdapterUserNameField.DisplayLabel Property

Suggests a display label for this adapter field.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this label or not. Server-side script uses the DisplayLabel property to access this value.

See Also

GetDisplayLabel (see page 5357)

54.1.36.3.2 TAdapterUserNameField.DisplayWidth Property

Suggests a display width, in characters, for this adapter field.

Pascal

```
property DisplayWidth: integer;
```

C++

```
__property int DisplayWidth;
```

Description

Set this property to indicate the width, in characters, that is suggested when generating an HTML field to display the value of this adapter field. Typically the adapter field is displayed using server-side script, either generated or custom written. The server-side script may be written to use this width or not. Server-side script uses the DisplayWidth property to access this value.

See Also

GetDisplayWidth (see page 5358)

54.1.36.3.3 TAdapterUserNameField.FieldName Property

Identifies the name of this field when using server-side script.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

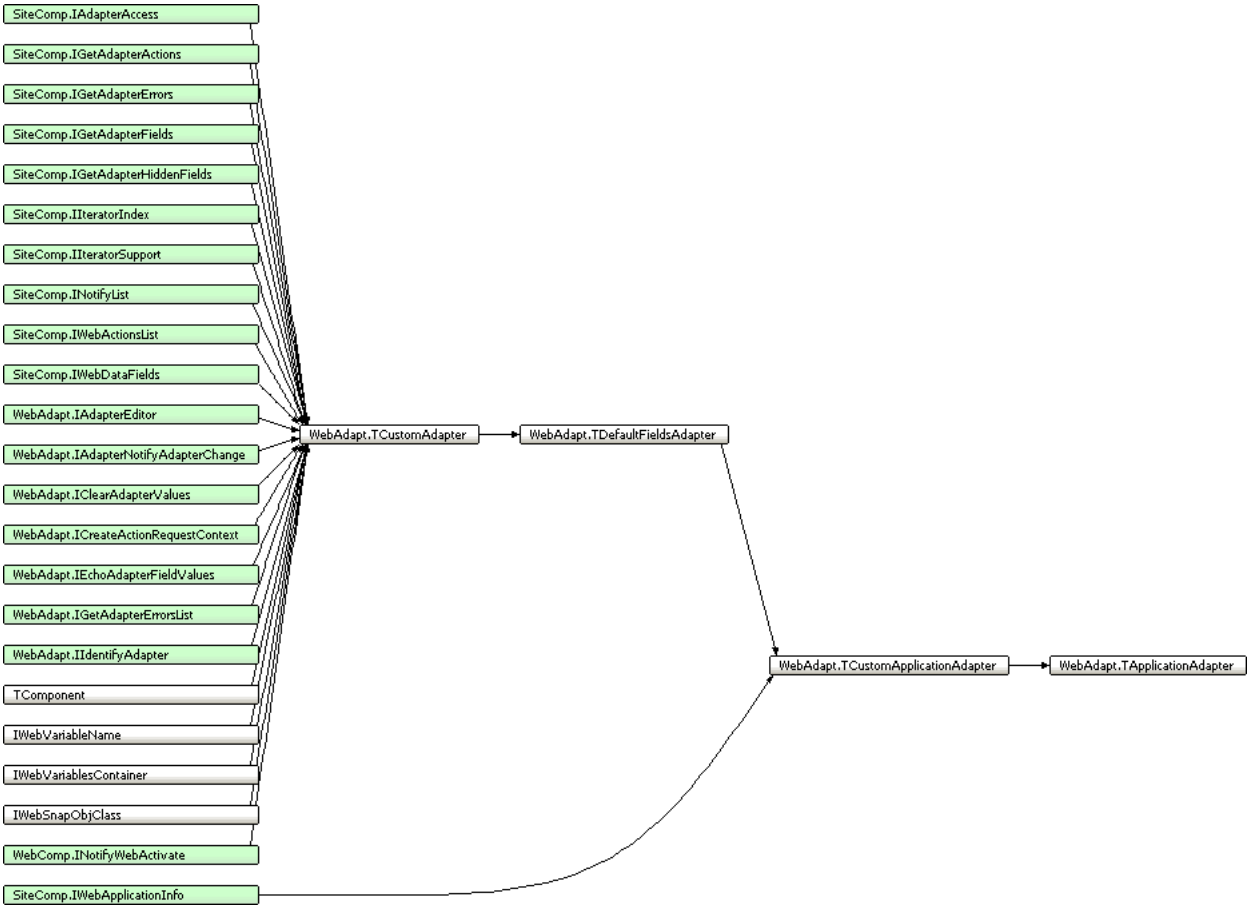
Description

The FieldName property defines the name used to access the adapter field with server-side script. Server-side script can access this value using the Name property.

54.1.37 TApplicationAdapter Class

TApplicationAdapter is a container for field and action components that are available through the Application script variable.

Class Hierarchy



Pascal

```
TApplicationAdapter = class(TCustomApplicationAdapter);
```

C++

```
class TApplicationAdapter : public TCustomApplicationAdapter;
```

File

WebAdapt

Description

By default, TApplicationAdapter contains a single field called Title that gets its value from the ApplicationTitle (see page 5444)

property. The TApplicationAdapter can be customized by adding additional field and action components. When writing server-side script, use the Application variable to access fields and actions of the ApplicationAdapter. For example, <%= Application.Title %> writes the title of the application.

See Also

TAdapter (see page 5190)

54.1.37.1 TApplicationAdapter Members

The following tables list the members exposed by TApplicationAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetApplicationAdapter (see page 5669)	Implements IWebApplicationInfo (see page 1078)'s GetApplicationAdapter method.
	GetApplicationTitle (see page 5669)	Implements IWebApplicationInfo (see page 1078)'s GetApplicationTitle method.

	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 5669)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 5669)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 5670)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 5670)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
	ImplGetFieldsList (see page 5670)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.

	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.

	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetApplicationAdapter (see page 1079)	Returns the value of the ApplicationAdapter (see page 1080) property.
	GetApplicationTitle (see page 1080)	Returns the value of the ApplicationTitle (see page 1080) property.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	ApplicationAdapter (see page 1080)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5445)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5445)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5446)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5446)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5446)	Supplies name/value pairs needed to execute an adapter action.

Published Properties

	Name	Description
	Actions (see page 5444)	Container for adapter actions components.
	ApplicationTitle (see page 5444)	Defines the title of the application.
	Data (see page 5444)	Container for adapter field components.

54.1.37.2 TApplicationAdapter Properties

The properties of the TApplicationAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5444)	Container for adapter actions components.
	ApplicationTitle (see page 5444)	Defines the title of the application.
	Data (see page 5444)	Container for adapter field components.

54.1.37.2.1 TApplicationAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5231)

54.1.37.2.2 TApplicationAdapter.ApplicationTitle Property

Defines the title of the application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property UnicodeString ApplicationTitle;__property UnicodeString ApplicationTitle;
```

Description

Set this property to identify an application title that can be accessed with server-side script as the Application.Title property.

54.1.37.2.3 TApplicationAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5237)

54.1.37.3 TApplicationAdapter Events

The events of the TApplicationAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5445)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5445)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5446)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5446)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5446)	Supplies name/value pairs needed to execute an adapter action.

54.1.37.3.1 TApplicationAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.37.3.2 TApplicationAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains

the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.37.3.3 TApplicationAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5478)

OnAfterExecuteAction (see page 5576)

54.1.37.3.4 TApplicationAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5478)

OnAfterGetActionResponse (see page 5576)

54.1.37.3.5 TApplicationAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

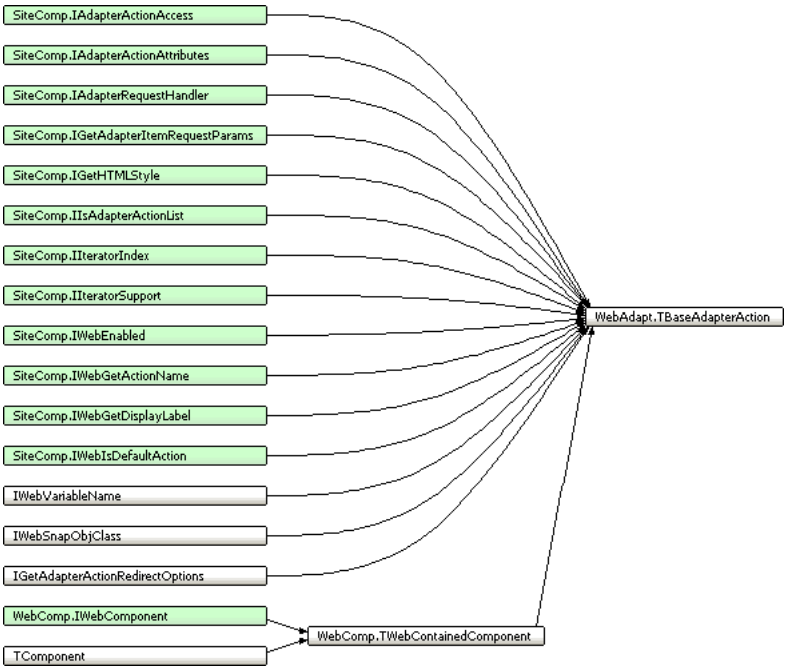
See Also

OnGetParams (see page 5478)

54.1.38 TBaseAdapterAction Class

TBaseAdapterAction is a based class for adapter action components.

Class Hierarchy



Pascal

```
TBaseAdapterAction = class(TWebContainedComponent, IWebVariableName, IWebEnabled,
IGetAdapterItemRequestParams, IWebGetActionName, IWebGetDisplayLabel, IWebSnapObjClass,
IAdapterActionAccess, IAdapterRequestHandler, IGetAdapterActionRedirectOptions, IGetHTMLStyle,
IAdapterActionAttributes, IIteratorSupport, IIteratorIndex, IIsAdapterActionList,
IWebIsDefaultAction);
```

C++

```
class TBaseAdapterAction : public TWebContainedComponent, public IWebVariableName, public
IWebEnabled, public IGetAdapterItemRequestParams, public IWebGetActionName, public
IWebGetDisplayLabel, public IWebSnapObjClass, public IAdapterActionAccess, public
IAdapterRequestHandler, public IGetAdapterActionRedirectOptions, public IGetHTMLStyle, public
IAdapterActionAttributes, public IIteratorSupport, public IIteratorIndex, public
IIsAdapterActionList, public IWebIsDefaultAction;
```

File

WebAdapt

Description

TBaseAdapterAction implements interfaces used to access an adapter action's properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize the behavior of the adapter action class.

See Also

TCustomAdapterAction (see page 5579)

TAdapter (see page 5190)

54.1.38.1 TBaseAdapterAction Members

The following tables list the members exposed by TBaseAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.38.2 TBaseAdapterAction Methods

The methods of the TBaseAdapterAction class are listed here.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.

	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.

	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
V	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

54.1.38.2.1 TBaseAdapterAction.CheckExecuteAccess Method

Checks end-user rights to execute this adapter action.

Pascal

```
procedure CheckExecuteAccess;
```

C++

```
__fastcall CheckExecuteAccess();
```

Description

This procedure is called to verify that the current user has rights to execute this action. An exception will be raised if access is denied.

See Also

ExecuteAccess (see page 5475)

HasExecuteAccess (see page 5463)

ExecuteActionRequest (see page 5455)

54.1.38.2.2 TBaseAdapterAction.CreateRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls this function before calling HandleRequest (see page 5463). The result of this function is stored in the WebContext (see page 6003) variable. During the course of executing this action, the object returned by CreateRequestContext will be used to access the action parameters and field values that were passed in the HTTP request to execute this action.

See Also

ImplCreateRequestContext (see page 5464)

AdapterRequest (see page 5971)

TAdapterDispatcher (see page 6148)

HandleRequest (see page 5463)

54.1.38.2.3 TBaseAdapterAction.DoAfterExecuteActionRequest Method

Calls event handlers after the adapter action is executed.

Pascal

```
procedure DoAfterExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoAfterExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

DoAfterExecuteActionRequest is called by ExecuteActionRequest (see page 5455). DoAfterExecuteActionRrequest packages the action request parameters into TStrings and calls two event handlers. The parent adapter's OnAfterExecuteAction event handler is called first. If it does not handle the action request then the OnAfterExecute (see page 5477) event handler is called.

See Also

OnAfterExecute (see page 5477)

OnAfterExecuteAction (see page 5576)

ExecuteActionRequest (see page 5455)

54.1.38.2.4 TBaseAdapterAction.DoAfterGetExecuteActionResponse Method

Calls event handlers after the adapter action response is formulated.

Pascal

```
procedure DoAfterGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoAfterGetExecuteActionResponse(IActionRequest AActionRequest,
IActionResponse AActionResponse);
```

Description

DoAfterGetExecuteActionResponse is called by GetExecuteActionResponse (see page 5458). The parent adapter's OnAfterGetActionResponse event handler is called first. If it does not handle the action response then the OnAfterGetResponse (see page 5477) event handler is called.

See Also

OnAfterGetResponse (see page 5477)

OnAfterGetActionResponse (see page 5576)

DoBeforeGetExecuteActionResponse (see page 5454)

GetExecuteActionResponse (see page 5458)

54.1.38.2.5 TBaseAdapterAction.DoBeforeExecuteActionRequest Method

Calls event handlers before the adapter action is executed.

Pascal

```
procedure DoBeforeExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
```

```
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoBeforeExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

DoBeforeExecuteActionRequest is called by ExecuteActionRequest (see page 5455). DoBeforeExecuteActionRequest packages the action request parameters into TStrings and calls two event handlers. The parent adapter's OnBeforeExecuteAction event handler is called first. If it does not handle the action request then the OnBeforeExecute (see page 5478) event handler is called. If either event handler handles the request then the IActionRequest (see page 5905)'s Handled property is set to true.

See Also

OnBeforeExecute (see page 5478)

OnBeforeExecuteAction (see page 5577)

DoAfterExecuteActionRequest (see page 5453)

ExecuteActionRequest (see page 5455)

54.1.38.2.6 TBaseAdapterAction.DoBeforeGetExecuteActionResponse Method

Calls event handlers before the adapter action response is formulated.

Pascal

```
procedure DoBeforeGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall DoBeforeGetExecuteActionResponse(IActionRequest AActionRequest,
IActionResponse AActionResponse);
```

Description

DoBeforeGetExecuteActionResponse is called by GetExecuteActionResponse (see page 5458). The parent adapter's OnBeforeGetActionResponse event handler is called first. If it does not handle the action response then the OnBeforeGetResponse (see page 5478) event handler is called. If either event handler handles the action response then the IActionResponse's Handled property is set to true.

See Also

OnBeforeGetResponse (see page 5478)

OnBeforeGetActionResponse (see page 5578)

DoAfterGetExecuteActionResponse (see page 5453)

GetExecuteActionResponse (see page 5458)

54.1.38.2.7 TBaseAdapterAction.EndIterator Method

Implements IIteratorSupport (see page 1054)'s EndIterator method.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

IIteratorSupport (see page 1054) is used to enumerate the list of commands supported by this adapter action. Most adapter actions have a single command. TAdapterGotoPageAction (see page 5291) is an example of an adapter action that supports multiple commands. EndIterator calls ImplNextIteration (see page 5470).

See Also

IIteratorSupport (see page 1054)

IIteratorIndex (see page 1052)

EndIterator (see page 1056)

ImplEndIterator (see page 5464)

54.1.38.2.8 TBaseAdapterAction.ExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse);
```

C++

```
__fastcall ExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ExecuteActionRequest is called by ImplHandleRequest (see page 5468) when a request to execute this adapter action is found in the HTTP request. ExecuteActionRequest first calls CheckExecuteAccess (see page 5452). CheckExecuteAccess (see page 5452) raises an exception if the end-user does not have execute rights. ExecuteActionRequest then calls, in turn, DoBeforeExecuteActionRequest (see page 5453), ImplExecuteActionRequest (see page 5464), and DoAfterExecuteActionRequest (see page 5453).

See Also

DoBeforeExecuteActionRequest (see page 5453)

DoAfterExecuteActionRequest (see page 5453)

CheckExecuteAccess (see page 5452)

ImplExecuteActionRequest (see page 5464)

ImplHandleRequest (see page 5468)

54.1.38.2.9 TBaseAdapterAction.GetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall UnicodeString GetActionName();
```

Description

GetActionName is called to retrieve the name used to access this adapter field with server-side script. GetActionName calls ImplGetActionName (see page 5465).

See Also

ActionName (see page 5474)

ImplGetActionName (see page 5465)

54.1.38.2.10 TBaseAdapterAction.GetAdapter Method

GetAdapter returns the parent adapter component.

Pascal

```
function GetAdapter: TCustomAdapter;
```

C++

```
__fastcall TCustomAdapter GetAdapter();
```

Description

To improve the efficiency of this function, a reference to the parent adapter component is cached in an instance variable.

See Also

Adapter (see page 5474)

54.1.38.2.11 TBaseAdapterAction.GetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(UnicodeString AIdentifier, IAdapterItemRequestParams
Params);
```

Description

The name/value pairs provided by this method will written to the HTTP response when the HTML page is generated and passed back in the HTTP request when this adapter action is to be executed. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted.

See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.38.2.12 TBaseAdapterAction.GetDefaultActionName Method

Gets the default name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns the name of the adapter action component. Descendants may override this function.

See Also

ImplGetActionName (see page 5465)

54.1.38.2.13 TBaseAdapterAction.GetDefaultDisplayLabel Method

Provides the default display label for this adapter action.

Pascal

```
function GetDefaultDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDefaultDisplayLabel();
```

Description

When the display label for the action matches this value then the display label property does not need to be stored in the form file.

54.1.38.2.14 TBaseAdapterAction.GetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function GetDisplayLabel: string;
```

C++

```
__fastcall UnicodeString GetDisplayLabel();
```

Description

GetDisplayLabel provides label that may be used when generating HTML to display a button or hyperlink for this adapter action. The value returned by this function can be accessed with server-side script using the DisplayLabel (see page 5474) property. GetDisplayLabel calls ImplGetDisplayLabel (see page 5465).

See Also

DisplayLabel (see page 5474)

ImplGetDisplayLabel (see page 5465)

54.1.38.2.15 TBaseAdapterAction.GetDisplayStyle Method

Returns a string value to suggest an HTML display style for this adapter action.

Pascal

```
function GetDisplayStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetDisplayStyle(const UnicodeString AAdapterMode);
```

Description

GetDisplayStyle implements IGetHTMLStyle (see page 1031)'s GetDisplayStyle method by translating the result value of GetDisplayStyleType (see page 5458) into a string.

See Also

GetDisplayStyle (see page 1032)

GetDisplayStyleType (see page 5458)

54.1.38.2.16 TBaseAdapterAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
virtual;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaButton, indicating that this adapter action should be displayed using a submit button. Descendant classes may override this method to suggest other HTML display types.

See Also

GetDisplayStyle (see page 5457)

54.1.38.2.17 TBaseAdapterAction.GetErrors Method

Returns the parent adapter's errors list.

Pascal

```
function GetErrors: TBaseAdapterErrorsList;
```

C++

```
__fastcall TBaseAdapterErrorsList GetErrors();
```

Description

GetErrors calls TCustomAdapter (see page 5531)'s GetAdapterErrorsList method.

See Also

Errors (see page 5475)

GetAdapterErrorsList (see page 5547)

54.1.38.2.18 TBaseAdapterAction.GetExecuteActionResponse Method

Generates a response after the adapter action has been executed.

Pascal

```
procedure GetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse);
```

C++

```
__fastcall GetExecuteActionResponse(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

GetExecuteActionResponse is called by ImplHandleRequest (see page 5468) after executing an adapter action specified by the HTTP request. GetExecuteActionResponse calls, in turn, DoBeforeGetExecuteActionResponse (see page 5454), ImplGetExecuteActionResponse (see page 5466), and DoAfterGetExecuteActionResponse (see page 5453).

See Also

IActionRequest

IActionResponse

DoBeforeGetExecuteActionResponse (see page 5454)

DoAfterGetExecuteActionResponse (see page 5453)

ImplGetExecuteActionResponse (see page 5466)

ImplHandleRequest (see page 5468)

54.1.38.2.19 TBaseAdapterAction.GetFailureOptions Method

Indicates whether to perform an HTTP redirect after execution of this adapter action fails.

Pascal

```
function GetFailureOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetFailureOptions();
```

Description

Set RedirectOptions (see page 5476) when an HTTP redirect should be performed after execution of this adapter action fails. GetFailureOptions returns the result of ImplGetFailureOptions (see page 5466).

See Also

ImplGetFailureOptions (see page 5466)

GetSuccessOptions (see page 5461)

RedirectOptions (see page 5476)

54.1.38.2.20 TBaseAdapterAction.GetInputStyle Method

Returns a string value to suggest an HTML display style for this adapter action.

Pascal

```
function GetInputStyle(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetInputStyle(const UnicodeString AAdapterMode);
```

Description

GetInputStyle implements IGetHTMLStyle (see page 1031)'s GetInputType method by returning the result of GetDisplayStyle (see page 5457). This method must be implemented as part of IGetHTMLStyle (see page 1031). GetInputStyle is treated as

a synonym of GetDisplayStyle (see page 5457).

See Also

GetInputStyle (see page 1033)

GetDisplayStyle (see page 5457)

54.1.38.2.21 TBaseAdapterAction.GetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function GetIsAdapterActionList: Boolean;
```

C++

```
__fastcall Boolean GetIsAdapterActionList();
```

Description

An adapter action may have a list of commands. The TAdapterGotoPageAction (see page 5291) is an example of an adapter action with a list of commands. Each command goes to a different page in the parent TPagedAdapter (see page 5844). GetIsAdapterActionList calls ImplGetIsAdapterActionList (see page 5466).

See Also

ImplGetIsAdapterActionList (see page 5466)

TAdapterGotoPageAction (see page 5291)

54.1.38.2.22 TBaseAdapterAction.GetItemCount Method

Returns the number of commands defined by this adapter action.

Pascal

```
function GetItemCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

An adapter action may have a list of commands. The TAdapterGotoPageAction (see page 5291) is an example of an adapter action with a list of commands. Each command goes to a different page in the parent TPagedAdapter (see page 5844). GetItemCount returns 1. GetItemCount may be overridden by a descendant class.

See Also

ItemCount (see page 5475)

ItemIndex (see page 5476)

TAdapterGotoPageAction (see page 5291)

GetIsAdapterActionList (see page 5460)

54.1.38.2.23 TBaseAdapterAction.GetIteratorIndex Method

Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

IIteratorIndex (see page 1052) and IIteratorSupport (see page 1054) are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. GetIteratorIndex returns the index of the current command. GetIteratorIndex calls ImplGetIteratorIndex (see page 5467).

See Also

IIteratorIndex (see page 1052)

IIteratorSupport (see page 1054)

GetIteratorIndex (see page 1053)

ImplGetIteratorIndex (see page 5467)

54.1.38.2.24 TBaseAdapterAction.GetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function GetSuccessOptions: TAdapterActionRedirectOptions;
```

C++

```
__fastcall TAdapterActionRedirectOptions GetSuccessOptions();
```

Description

Set RedirectOptions (see page 5476) when an HTTP redirect should be performed after executing an action. GetSuccessOptions returns the result of ImplGetSuccessOptions (see page 5467).

See Also

ImplGetSuccessOptions (see page 5467)

GetFailureOptions (see page 5459)

RedirectOptions (see page 5476)

54.1.38.2.25 TBaseAdapterAction.GetVariableName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall UnicodeString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter action with server-side script. GetVariableName calls ImplGetVariableName (see page 5467).

See Also

ImplGetVariableName (see page 5467)

54.1.38.2.26 TBaseAdapterAction.GetViewMode Method

Returns a string value to suggest a view mode for this adapter action.

Pascal

```
function GetViewMode(const AAdapterMode: string): string;
```

C++

```
__fastcall UnicodeString GetViewMode(const UnicodeString AAdapterMode);
```

Description

GetViewMode implements IGetHTMLStyle (see page 1031)'s GetViewMode method by translating the result value of GetViewModeType (see page 5462) into a string.

See Also

GetViewMode (see page 1033)

GetViewModeType (see page 5462)

54.1.38.2.27 TBaseAdapterAction.GetViewModeType Method

Suggests a view mode for this adapter action.

Pascal

```
function GetViewModeType(const AAdapterMode: string): TAdapterDisplayViewModeType; virtual;
```

C++

```
virtual __fastcall TAdapterDisplayViewModeType GetViewModeType(const UnicodeString
AAdapterMode);
```

Description

GetViewModeType returns htmlvmInput.

See Also

ReadOnly (see page 5351)

GetViewMode (see page 5348)

54.1.38.2.28 TBaseAdapterAction.GetVisible Method

Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

Pascal

```
function GetVisible: Boolean;
```

C++

```
__fastcall Boolean GetVisible();
```

Description

The state of the parent adapter may dictate whether a button or hyperlink for a particular adapter action should not be shown. For example, TDataSetAdapterApplyRowAction can be configured to return false when the parent TDataSetAdapter is in browse

mode. GetVisible calls ImplGetVisible (see page 5468). Server-side script can access the value returned by GetVisible using the Visible property of an adapter action object.

See Also

ImplGetVisible (see page 5468)

54.1.38.2.29 TBaseAdapterAction.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

54.1.38.2.30 TBaseAdapterAction.HandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

The adapter dispatcher calls HandleRequest to execute the adapter action and generate a response. HandleRequest calls ImplHandleRequest (see page 5468).

See Also

CreateRequestContext (see page 5452)

54.1.38.2.31 TBaseAdapterAction.HasExecuteAccess Method

Checks the end-user's right to execute this adapter action.

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanExecute property. HasExecuteAccess calls ImplHasExecuteAccess (see page 5468).

See Also

CheckExecuteAccess (see page 5452)

ImplHasExecuteAccess (see page 5468)

54.1.38.2.32 TBaseAdapterAction.ImplCreateRequestContext Method

Creates an object that describes the action parameters and field values.

Pascal

```
procedure ImplCreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall ImplCreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

ImplCreateRequestContext is called by CreateRequestContext (see page 5452). ImplCreateRequestContext returns the result of TCustomAdapter (see page 5531)'s CreateActionRequestContext method.

See Also

CreateRequestContext (see page 5452)

CreateActionRequestContext (see page 5541)

54.1.38.2.33 TBaseAdapterAction.ImplEndIterator Method

Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer);
```

C++

```
__fastcall ImplEndIterator(void * OwnerData);
```

Description

ImplEndIterator is called by EndIterator (see page 5454). ImplEndIterator resets internal variables.

See Also

IIteratorIndex (see page 1052)

IIteratorSupport (see page 1054)

EndIterator (see page 5454)

54.1.38.2.34 TBaseAdapterAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest is called by ExecuteActionRequest (see page 5455). ImplExecuteActionRequest must be

overridden by a descendant class.

See Also

IActionRequest

IActionResponse

ExecuteActionRequest (see page 5455)

54.1.38.2.35 TBaseAdapterAction.ImplGetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetActionName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetActionName();
```

Description

This function is called by GetActionName (see page 5455). ImplGetActionName returns the result of GetDefaultActionName (see page 5456). Descendants may override this function.

See Also

GetActionName (see page 5455)

GetDefaultActionName (see page 5456)

54.1.38.2.36 TBaseAdapterAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 5456). ImplGetAdapterItemRequestParams calls the OnGetActionParams event handler of the parent adapter and then calls the OnGetParams (see page 5478) event handler.

See Also

GetAdapterItemRequestParams (see page 5456)

OnGetActionParams (see page 5579)

OnGetParams (see page 5478)

54.1.38.2.37 TBaseAdapterAction.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function ImplGetDisplayLabel: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the value of the DisplayLabel (see page 5474) property.

See Also

DisplayLabel (see page 5474)

54.1.38.2.38 TBaseAdapterAction.ImplGetExecuteActionResponse Method

Generates a response after the adapter action has been executed.

Pascal

```
procedure ImplGetExecuteActionResponse(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); virtual;
```

C++

```
virtual __fastcall ImplGetExecuteActionResponse(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplGetExecuteActionResponse is called by GetExecuteActionResponse (see page 5458). ImplGetExecuteActionResponse sets properties of IActionResponse to identify the response to this action request.

See Also

GetExecuteActionResponse (see page 5458)

54.1.38.2.39 TBaseAdapterAction.ImplGetFailureOptions Method

Indicates whether to perform an HTTP redirect after execution of this adapter action fails.

Pascal

```
function ImplGetFailureOptions: TAdapterActionRedirectOptions; virtual;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetFailureOptions();
```

Description

ImplGetFailureOptions returns an empty set.

See Also

GetFailureOptions (see page 5459)

54.1.38.2.40 TBaseAdapterAction.ImplGetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function ImplGetIsAdapterActionList: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetIsAdapterActionList();
```

Description

ImplGetIsAdapterActionList is called by GetIsAdapterActionList (see page 5460). ImplGetIsAdapterActionList returns false. ImplGetIsAdapterActionList may be overridden by descendant classes.

See Also

GetIsAdapterActionList (see page 5460)

ImplGetIsAdapterActionList (see page 5621)

54.1.38.2.41 TBaseAdapterAction.ImplGetIteratorIndex Method

Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.

Pascal

```
function ImplGetIteratorIndex: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetIteratorIndex();
```

Description

ImplGetIteratorIndex is called by GetIteratorIndex (see page 5460). ImplGetIteratorIndex returns the value of an internal iteration index.

See Also

GetIteratorIndex (see page 1053)

GetIteratorIndex (see page 5460)

54.1.38.2.42 TBaseAdapterAction.ImplGetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function ImplGetSuccessOptions: TAdapterActionRedirectOptions; virtual;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetSuccessOptions();
```

Description

ImplGetSuccessOptions returns the value of the RedirectOptions (see page 5476) property.

See Also

GetSuccessOptions (see page 5461)

RedirectOptions (see page 5476)

54.1.38.2.43 TBaseAdapterAction.ImplGetVariableName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetVariableName: string;
```

C++

```
__fastcall UnicodeString ImplGetVariableName();
```

Description

This function is called by GetVariableName (see page 5461). GetVariableName (see page 5461) returns the result of GetActionName (see page 5455).

See Also

GetVariableName (see page 5461)

GetActionName (see page 5455)

54.1.38.2.44 TBaseAdapterAction.ImplGetVisible Method

Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

Pascal

```
function ImplGetVisible: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetVisible();
```

Description

ImplGetVisible is called by GetVisible (see page 5462). ImplGetVisible returns true. ImplGetVisible may be overridden by descendant components.

See Also

GetVisible (see page 5462)

54.1.38.2.45 TBaseAdapterAction.ImplHandleRequest Method

Executes the adapter action and generates a response.

Pascal

```
function ImplHandleRequest(DispatchParams: IAdapterDispatchParams): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

ImplHandleRequest is called by HandleRequest (see page 5463). ImplHandleRequest calls ExecuteActionRequest (see page 5455) followed by GetExecuteActionResponse (see page 5458).

See Also

ExecuteActionRequest (see page 5455)

GetExecuteActionResponse (see page 5458)

54.1.38.2.46 TBaseAdapterAction.ImplHasExecuteAccess Method

Checks the end-user's right to execute this adapter action.

Pascal

```
function ImplHasExecuteAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteAccess();
```

Description

This function is called by HasExecuteAccess (see page 5463) to check the end-user's right to execute the adapter action. First, the parent adapter's HasExecuteAccess (see page 5463) method is called. If the adapter allows execute access and the ExecuteAccess (see page 5475) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ExecuteAccess (see page 5475) property value.

See Also

HasExecuteRights (see page 5509)

ExecuteAccess (see page 5475)

HasExecuteAccess (see page 5463)

HasExecuteAccess (see page 5552)

54.1.38.2.47 TBaseAdapterAction.ImplInIterator Method

Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.

Pascal

```
function ImplInIterator: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplInIterator();
```

Description

ImplInIterator is called by InIterator (see page 5471). ImplInIterator returns the value of an internal flag.

See Also

IIteratorIndex (see page 1052)

IIteratorSupport (see page 1054)

InIterator (see page 1053)

54.1.38.2.48 TBaseAdapterAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 5471). ImplIsDefaultAction returns true. ImplIsDefaultAction may be overridden by a descendant class.

See Also

IsDefaultAction (see page 5471)

ImplIsDefaultAction (see page 5622)

54.1.38.2.49 TBaseAdapterAction.ImplNextIteration Method

Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration is called by NextIteration (see page 5472). ImplNextIteration increments an internal iteration index. true is returned if the iteration index is within the bounds defined by ItemCount (see page 5475).

See Also

IIteratorIndex (see page 1052)

IIteratorSupport (see page 1054)

NextIteration (see page 5472)

54.1.38.2.50 TBaseAdapterAction.ImplStartIterator Method

Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator is called by StartIterator (see page 5473). ImplStartIterator initializes the internal iteration index. ImplStartIterator returns true because it is assumed that there is at least one command associated with this adapter action.

See Also

IIteratorIndex (see page 1052)

IIteratorSupport (see page 1054)

StartIterator (see page 1056)

StartIterator (see page 5473)

54.1.38.2.51 TBaseAdapterAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled is called by WebEnabled (see page 5473). ImplWebEnabled returns true. Descendant classes may override this method.

See Also

WebEnabled (see page 5473)

54.1.38.2.52 TBaseAdapterAction.InIterator Method

Implements IIteratorIndex (see page 1052)'s InIterator method.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

IIteratorIndex (see page 1052) and IIteratorSupport (see page 1054) are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. InIterator calls ImplInIterator (see page 5469).

See Also

IIteratorIndex (see page 1052)

IIteratorSupport (see page 1054)

InIterator (see page 1053)

ImplInIterator (see page 5469)

54.1.38.2.53 TBaseAdapterAction.IsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function IsDefaultAction(ADisplay: TObject): Boolean;
```

C++

```
__fastcall Boolean IsDefaultAction(TObject * ADisplay);
```

Description

Some adapter components have built in support for specific types of adapter actions. These actions are often added to the adapter component by default unless the developer explicitly adds one or more adapter actions. IsDefaultAction is called by the adapter while adding default actions to determine whether this adapter action should be added. ADisplay identifies characteristics of the adapter component. IsDefaultAction calls ImplIsDefaultAction (see page 5469).

See Also

ImplIsDefaultAction (see page 5469)

54.1.38.2.54 TBaseAdapterAction.NextIteration Method

Implements IIteratorSupport (see page 1054)'s NextIteration method.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

IIteratorSupport (see page 1054) is used to enumerate the list of commands supported by this adapter action. Most adapter actions have a single command. TAdapterGotoPageAction (see page 5291) is an example of an adapter action that supports multiple commands. NextIterations returns the result of ImplNextIteration (see page 5470).

See Also

IIteratorSupport (see page 1054)

IIteratorIndex (see page 1052)

ImplNextIteration (see page 5470)

NextIteration (see page 1056)

54.1.38.2.55 TBaseAdapterAction.ParentChanged Method

Handles movement of the component to a different parent.

Pascal

```
procedure ParentChanged; override;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called when the parent component is changed. TBaseAdapterAction (see page 5447) clears its cached reference to the parent adapter.

54.1.38.2.56 TBaseAdapterAction.SetDisplayLabel Method

Set the value of the DisplayLabel (see page 5474) property.

Pascal

```
procedure SetDisplayLabel(const Value: string);
```

C++

```
__fastcall SetDisplayLabel(const UnicodeString Value);
```

Description

SetDisplayLabel sets the value of the DisplayLabel (see page 5474) property.

See Also

DisplayLabel (see page 5474)

54.1.38.2.57 TBaseAdapterAction.StartIterator Method

Implements IIteratorSupport (see page 1054)'s StartIterator method.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

IIteratorIndex (see page 1052) and IIteratorSupport (see page 1054) are used to iterate through the commands supported by an adapter action. Most adapter actions support a single command. TAdapterActionGotoPage is adapter action that supports multiple commands. StartIterator calls ImplImplInIterator.

See Also

IIteratorIndex (see page 1052)

IIteratorSupport (see page 1054)

StartIterator (see page 1056)

ImplStartIterator (see page 5470)

54.1.38.2.58 TBaseAdapterAction.WebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function WebEnabled: Boolean;
```

C++

```
__fastcall Boolean WebEnabled();
```

Description

The state of the parent adapter may dictate when a button or hyperlink for a particular adapter action should not be enabled. For example, TDataSetAdapterPrevRowAction returns False when the dataset is positioned at BOF. WebEnabled calls ImplWebEnabled (see page 5470). Server-side script can access the value returned by WebEnabled using the Enabled property of an adapter action object.

See Also

ImplWebEnabled (see page 5470)

54.1.38.3 TBaseAdapterAction Properties

The properties of the TBaseAdapterAction class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Errors (see page 5475)	Returns the parent adapter's Errors list.

Public Properties

	Name	Description
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.

54.1.38.3.1 TBaseAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

GetActionName (see page 5455)

54.1.38.3.2 TBaseAdapterAction.Adapter Property

Provides access to the parent adapter component.

Pascal

```
property Adapter: TCustomAdapter;
```

C++

```
__property TCustomAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter component.

54.1.38.3.3 TBaseAdapterAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5465)

SetDisplayLabel (see page 5472)

54.1.38.3.4 TBaseAdapterAction.Errors Property

Returns the parent adapter's Errors list.

Pascal

```
property Errors: TBaseAdapterErrorsList;
```

C++

```
__property TBaseAdapterErrorsList Errors;
```

Description

Descendant classes use this property to access the Errors list of the parent adapter. The Errors list is used to track Errors that occur while executing an adapter action. Server-side script can access the Errors list using the adapter object's Errors property. The server-side script Errors list is used to write the Errors to the HTML page.

See Also

GetErrors (see page 5458)

54.1.38.3.5 TBaseAdapterAction.ExecuteAccess Property

Defines the rights required to execute this action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this action. When server-side script accesses the CanExecute property, this value is checked against the end user's rights. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6796)

HasExecuteAccess (see page 5552)

54.1.38.3.6 TBaseAdapterAction.ItemCount Property

Returns the number of items defined by an adapter action.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

An adapter action may have a list of commands. The ItemCount property is implemented by GetItemCount (see page 5460). The TAdapterGotoPageAction (see page 5291) is an example of an adapter action with a list of commands. Each command of TAdapterGotoPageAction (see page 5291) displays a different page of the parent TPagedAdapter (see page 5844).

See Also

GetItemCount (see page 5460)

ItemIndex (see page 5476)

TAdapterGotoPageAction (see page 5291)

GetIsAdapterActionList (see page 5460)

54.1.38.3.7 TBaseAdapterAction.ItemIndex Property

Returns the index of the current item of an adapter action.

Pascal

```
property ItemIndex: Integer;
```

C++

```
__property int ItemIndex;
```

Description

An adapter action may have a list of commands. The ItemIndex indicates the current item in the list. The TAdapterGotoPageAction (see page 5291) is an example of an adapter action with a list of commands. Each command of TAdapterGotoPageAction (see page 5291) displays a different page of the parent TPagedAdapter (see page 5844).

See Also

ItemCount (see page 5475)

TAdapterGotoPageAction (see page 5291)

GetIsAdapterActionList (see page 5460)

54.1.38.3.8 TBaseAdapterAction.RedirectOptions Property

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
property RedirectOptions: TAdapterActionRedirectOptions;
```

C++

```
__property TAdapterActionRedirectOptions RedirectOptions;
```

Description

Set RedirectOptions to indicate that HTTP redirect should be performed after executing this adapter action. An HTTP redirect causes the browser to "forget" about the parameters passed when executing the adapter action. Without a redirect, the browser's Refresh or Reload command cause the last HTTP request to be resent, resulting in re-execution of the adapter action.

See Also

SendRedirect

54.1.38.4 TBaseAdapterAction Events

The events of the TBaseAdapterAction class are listed here.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

54.1.38.4.1 TBaseAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest (see page 5453).

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.38.4.2 TBaseAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.38.4.3 TBaseAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest (see page 5453).

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.38.4.4 TBaseAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.38.4.5 TBaseAdapterAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

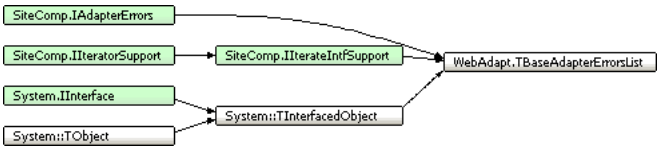
See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.39 TBaseAdapterErrorsList Class

TBaseAdapterErrorsList is the base class for adapter error object managers.

Class Hierarchy



Pascal

```
TBaseAdapterErrorsList = class(TInterfacedObject, IAdapterErrors, IIterateIntfSupport);
```

C++

```
class TBaseAdapterErrorsList : public TInterfacedObject, public IAdapterErrors, public
IIterateIntfSupport;
```

File

WebAdapt

Description

TBaseAdapterErrorsList is the base class for objects that manage lists of adapter error objects. Do not instantiate TBaseAdapterErrorsList directly. Use TAdapterErrorsList (see page 5259).

See Also

TAdapterErrorsList (see page 5259)

IAdapterError (see page 992)

54.1.39.1 TBaseAdapterErrorsList Members

The following tables list the members exposed by TBaseAdapterErrorsList.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.

	EndIterator (see page 5483)	Signals the end of the iterating operation.
	GetCurrentIntf (see page 5483)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5484)	Returns a particular error.
	GetErrorCount (see page 5484)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5484)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5484)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 5485)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
V	ImplDefineLabel (see page 5485)	This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.
V	ImplGetError (see page 5485)	This is ImplGetError, a member of class TBaseAdapterErrorsList.
V	ImplGetErrorCount (see page 5485)	This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.
V	ImplGetObjectErrors (see page 5486)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
V	ImplHasObjectErrors (see page 5486)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 5486)	Advances to the objects next item.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	StartIterator (see page 5486)	Positions the object on its first item and starts the iteration process.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AddError (see page 5481)	This is the overview for the AddError method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearErrors (see page 5482)	Clear error list.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	DefineLabel (see page 5483)	Define error label for field.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ErrorCount (see page 5487)	Number of error objects contained.
R	Errors (see page 5487)	Access to error objects.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

54.1.39.2 TBaseAdapterErrorsList Methods

The methods of the TBaseAdapterErrorsList class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 5483)	Signals the end of the iterating operation.
	GetCurrentIntf (see page 5483)	This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.
	GetError (see page 5484)	Returns a particular error.
	GetErrorCount (see page 5484)	Return the number of errors in the errors list.
	GetObjectErrors (see page 5484)	Returns an object that is used to enumerate all errors associated with an adapter field.
	HasObjectErrors (see page 5484)	Indicates whether the errors list contains objects associated with a particular adapter field.
V	ImplClearErrors (see page 5485)	This is ImplClearErrors, a member of class TBaseAdapterErrorsList.
V	ImplDefineLabel (see page 5485)	This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.
V	ImplGetError (see page 5485)	This is ImplGetError, a member of class TBaseAdapterErrorsList.
V	ImplGetErrorCount (see page 5485)	This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.
V	ImplGetObjectErrors (see page 5486)	This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.
V	ImplHasObjectErrors (see page 5486)	This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.
	NextIteration (see page 5486)	Advances to the objects next item.
	StartIterator (see page 5486)	Positions the object on its first item and starts the iteration process.

Public Methods

	Name	Description
V	AddError (see page 5481)	This is the overview for the AddError method overload.
	ClearErrors (see page 5482)	Clear error list.
	DefineLabel (see page 5483)	Define error label for field.

54.1.39.2.1 AddError Method

This is the overview for the AddError method overload.

Overload List

	Name	Description
	TBaseAdapterErrorsList.AddError (Exception, string) (see page 5482)	Add error to list.
	TBaseAdapterErrorsList.AddError (Exception, TObject) (see page 5482)	Add error to list.
	TBaseAdapterErrorsList.AddError (string, string, Integer) (see page 5482)	Add error to list.
	TBaseAdapterErrorsList.AddError (string, TObject, Integer) (see page 5482)	Add error to list.

54.1.39.2.1.1 TBaseAdapterErrorsList.AddError Method (Exception, TObject)

Add error to list.

Pascal

```
procedure AddError(AException: Exception; const AObjectName: string = ''); virtual; overload;
procedure AddError(const AMessage: string; const AObjectName: string = ''; AID: Integer = 0);
virtual; overload;
procedure AddError(const AMessage: string; AObject: TObject; AID: Integer = 0); virtual;
overload;
procedure AddError(AException: Exception; AObject: TObject); virtual; overload;
```

C++

```
virtual __fastcall AddError(Exception AException, const UnicodeString AObjectName = '');
virtual __fastcall AddError(const UnicodeString AMessage, const UnicodeString AObjectName =
'', int AID = 0);
virtual __fastcall AddError(const UnicodeString AMessage, TObject * AObject, int AID = 0);
virtual __fastcall AddError(Exception AException, TObject * AObject);
```

Description

AddError adds an error to the list maintained by the TBaseAdapterErrorsList (see page 5479) instance. As implemented in TBaseAdapterErrorsList (see page 5479), AddError does nothing. This method must be implemented by descendant classes.

See Also

AddError (see page 5263)

54.1.39.2.2 TBaseAdapterErrorsList.ClearErrors Method

Clear error list.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

Clear errors deletes all error information maintained by the TBaseAdapterErrorsList (see page 5479) instance.

See Also

AddError (see page 5263)

54.1.39.2.3 TBaseAdapterErrorsList.DefineLabel Method

Define error label for field.

Pascal

```
procedure DefineLabel(const AName: string; const ALabel: string);
```

C++

```
__fastcall DefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

DefineLabel defines a string label that represents a dataset field in error message. AName is the the field name. ALabel is the label.

See Also

AddError (see page 5263)

54.1.39.2.4 TBaseAdapterErrorsList.EndIterator Method

Signals the end of the iterating operation.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

Server-side script calls EndIterator when it stops iterating over the object's items. The object uses this method to perform any clean-up to reflect the fact that it is no longer iterating over its items.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. It is passed to each NextIteration call (which can modify the information), and finally supplied to the EndIterator method, which can free the memory used to store the information.

See Also

StartIterator (see page 1056)

NextIteration (see page 1056)

54.1.39.2.5 TBaseAdapterErrorsList.GetCurrentIntf Method

Pascal

```
function GetCurrentIntf(OwnerData: Pointer): IUnknown;
```

C++

```
__fastcall IUnknown GetCurrentIntf(void * OwnerData);
```

Description

This is GetCurrentIntf, a member of class TBaseAdapterErrorsList.

54.1.39.2.6 TBaseAdapterErrorsList.GetError Method

Returns a particular error.

Pascal

```
function GetError(I: Integer): IAdapterError;
```

C++

```
__fastcall IAdapterError GetError(int I);
```

Description

I is a zero-based index.

See Also

IAdapterError (see page 992)

54.1.39.2.7 TBaseAdapterErrorsList.GetErrorCount Method

Return the number of errors in the errors list.

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

Return the number of errors in the errors list.

54.1.39.2.8 TBaseAdapterErrorsList.GetObjectErrors Method

Returns an object that is used to enumerate all errors associated with an adapter field.

Pascal

```
function GetObjectErrors(const AName: string): TObject;
```

C++

```
__fastcall TObject * GetObjectErrors(const UnicodeString AName);
```

Description

AName identifies an adapter field. nil (Delphi) or NULL (C++) is returned if there are no errors associated with the adapter field.

See Also

HasObjectErrors (see page 997)

54.1.39.2.9 TBaseAdapterErrorsList.HasObjectErrors Method

Indicates whether the errors list contains objects associated with a particular adapter field.

Pascal

```
function HasObjectErrors(const AName: string): Boolean;
```

C++

```
__fastcall Boolean HasObjectErrors(const UnicodeString AName);
```

Description

HasObjectErrors returns true if any errors are associated with the given adapter field. AName identifies an adapter field.

See Also

GetObjectErrors (see page 997)

54.1.39.2.10 TBaseAdapterErrorsList.ImplClearErrors Method

Pascal

```
procedure ImplClearErrors; virtual;
```

C++

```
virtual __fastcall ImplClearErrors();
```

Description

This is ImplClearErrors, a member of class TBaseAdapterErrorsList.

54.1.39.2.11 TBaseAdapterErrorsList.ImplDefineLabel Method

Pascal

```
procedure ImplDefineLabel(const AName: string; const ALabel: string); virtual;
```

C++

```
virtual __fastcall ImplDefineLabel(const UnicodeString AName, const UnicodeString ALabel);
```

Description

This is ImplDefineLabel, a member of class TBaseAdapterErrorsList.

54.1.39.2.12 TBaseAdapterErrorsList.ImplGetError Method

Pascal

```
function ImplGetError(I: Integer): IAdapterError; virtual;
```

C++

```
virtual __fastcall IAdapterError ImplGetError(int I);
```

Description

This is ImplGetError, a member of class TBaseAdapterErrorsList.

54.1.39.2.13 TBaseAdapterErrorsList.ImplGetErrorCount Method

Pascal

```
function ImplGetErrorCount: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetErrorCount();
```

Description

This is ImplGetErrorCount, a member of class TBaseAdapterErrorsList.

54.1.39.2.14 TBaseAdapterErrorsList.ImplGetObjectErrors Method

Pascal

```
function ImplGetObjectErrors(const AName: string): TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetObjectErrors(const UnicodeString AName);
```

Description

This is ImplGetObjectErrors, a member of class TBaseAdapterErrorsList.

54.1.39.2.15 TBaseAdapterErrorsList.ImplHasObjectErrors Method

Pascal

```
function ImplHasObjectErrors(const AName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasObjectErrors(const UnicodeString AName);
```

Description

This is ImplHasObjectErrors, a member of class TBaseAdapterErrorsList.

54.1.39.2.16 TBaseAdapterErrorsList.NextIteration Method

Advances to the objects next item.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Server-side script calls NextIteration to move to the next item.

OwnerData is a pointer to information that the object can use to influence the iteration process. Each object stores and uses this information in its own way. This information is first supplied by the initial call to the StartIterator method. NextIteration can modify the information, after which it is supplied to the next call to the NextIteration or EndIterator method.

NextIteration returns true if it successfully moves to the next item, false if there were no more items.

See Also

StartIterator (see page 1056)

EndIterator (see page 1056)

54.1.39.2.17 TBaseAdapterErrorsList.StartIterator Method

Positions the object on its first item and starts the iteration process.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

Server-side script calls StartIterator to begin iterating over the object's items.

OwnerData returns a pointer to information that the object can use to influence the iteration process. StartIterator initializes this information to reflect what ever information the object needs to communicate to itself as it advances over its items. This pointer is then passed to the next call to the NextIteration or EndIterator method.

StartIterator returns true if it successfully moves to the first item, false if there are no items to iterate.

See Also

NextIteration (see page 1056)

EndIterator (see page 1056)

54.1.39.3 TBaseAdapterErrorsList Properties

The properties of the TBaseAdapterErrorsList class are listed here.

Public Properties

	Name	Description
R	ErrorCount (see page 5487)	Number of error objects contained.
R	Errors (see page 5487)	Access to error objects.

54.1.39.3.1 TBaseAdapterErrorsList.ErrorCount Property

Number of error objects contained.

Pascal

```
property ErrorCount: Integer;
```

C++

```
__property int ErrorCount;
```

Description

ErrorCount returns the number of error objects available.

See Also

Errors (see page 5487)

54.1.39.3.2 TBaseAdapterErrorsList.Errors Property

Access to error objects.

Pascal

```
property Errors [I: Integer]: IAdapterError;
```

C++

```
__property IAdapterError Errors[int I];
```

Description

Errors returns a specific error object.

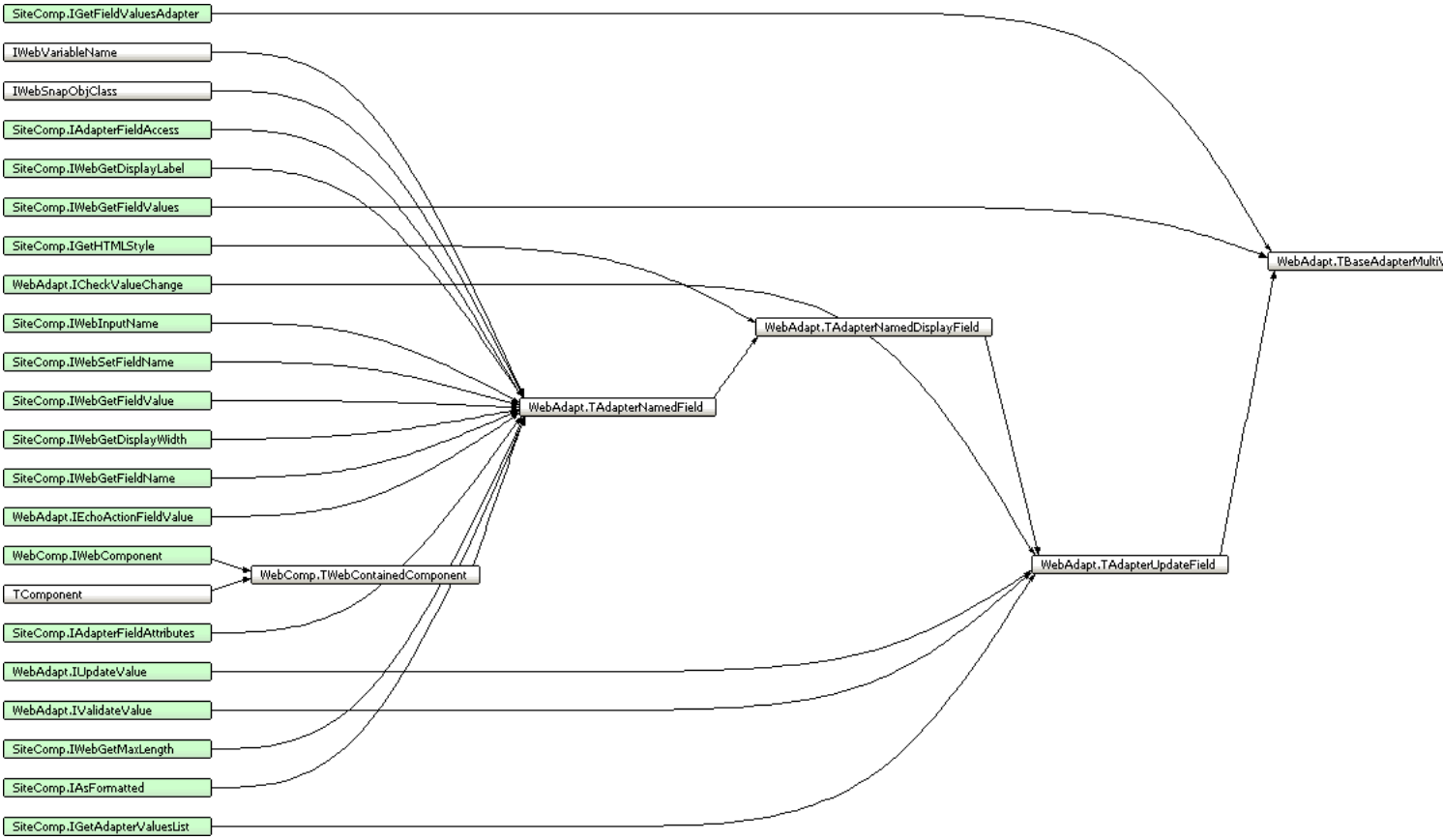
See Also

ErrorCount (see page 5487)

54.1.40 TBaseAdapterMultiValueField Class

TBaseAdapterMultiValueField is a base class for adapter fields that have zero or more values.

Class Hierarchy



Pascal

```
TBaseAdapterMultiValueField = class(TAdapterUpdateField, IGetFieldValuesAdapter,
IWebGetFieldValues);
```

C++

```
class TBaseAdapterMultiValueField : public TAdapterUpdateField, public IGetFieldValuesAdapter,
public IWebGetFieldValues;
```

File

WebAdapt

Description

TBaseAdapterMultiValueField is used to represent a data value that can have multiple values. Server-side script is used to generate a form displaying this field. Typically, the HTML form will contain group check boxes or a select list supporting multiple selection. When an Adapter (see page 5368) action is executed, TBaseAdapterMultiValueField is used to retrieve the submitted field value(s) from the HTTP request.

See Also

TCustomAdapterMultiValueField (see page 5646)

TCustomAdapterFileField (see page 5604)

54.1.40.1 TBaseAdapterMultiValueField Members

The following tables list the members exposed by TBaseAdapterMultiValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
	EndValueIterator (see page 5493)	Completes iteration.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5494)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5494)	Returns the adapter object that implements the Values property.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
V	GetValueCount (see page 5494)	Returns the number of values associated with this adapter field.
V	GetValues (see page 5495)	Returns one of the values associated with the adapter field.

	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' HasValue method.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetCurrentValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue (see page 5494) method.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetFieldValuesAdapter (see page 5496)	Returns the adapter object that implements the Values property.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasValue (see page 5496)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 5430)	Updates the adapter field with value(s) entered by the end-user.
V	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5496)	Implements IWebGetFieldValues (see page 1092)' NextValueIteration method.
V	Notification (see page 5431)	Clears references to a deleted component.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.

	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartValueIterator (see page 5497)	Implements IWebGetFieldValues (see page 1092)' StartValueIterator method.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
R	Adapter (see page 5368)	Returns the parent adapter component.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5492)	Constructor for TBaseAdapterMultiValueField (see page 5488).
V	Destroy (see page 5492)	Destructor for TBaseAdapterMultiValueField (see page 5488).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.40.2 TBaseAdapterMultiValueField.Create Constructor

Constructor for TBaseAdapterMultiValueField (see page 5488).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TBaseAdapterMultiValueField(TComponent * AOwner);
```

Description

Creates an instance of TFieldValuesAdapter (see page 5803) and calls the inherited creator method. TFieldValuesAdapter (see page 5803) is used to implement the Values property used by server-side script.

See Also

TFieldValuesAdapter (see page 5803)

ImplGetFieldValuesAdapter (see page 5496)

Destroy (see page 5492)

54.1.40.3 TBaseAdapterMultiValueField.Destroy Destructor

Destructor for TBaseAdapterMultiValueField (see page 5488).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TBaseAdapterMultiValueField();
```

Description

Destroys the instance of TFieldValuesAdapter (see page 5803) and calls the inherited destructor. TFieldValuesAdapter (see page 5803) is used to implement the Values property used by server-side script.

See Also

TFieldValuesAdapter (see page 5803)

ImplGetFieldValuesAdapter (see page 5496)

Create (see page 5492)

54.1.40.4 TBaseAdapterMultiValueField Methods

The methods of the TBaseAdapterMultiValueField class are listed here.

Protected Methods

	Name	Description
	EndValueIterator (see page 5493)	Completes iteration.
	GetCurrentValue (see page 5494)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue.
	GetFieldValuesAdapter (see page 5494)	Returns the adapter object that implements the Values property.
V	GetValueCount (see page 5494)	Returns the number of values associated with this adapter field.
V	GetValues (see page 5495)	Returns one of the values associated with the adapter field.
	HasValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' HasValue method.
V	ImplGetCurrentValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue (see page 5494) method.
V	ImplGetFieldValuesAdapter (see page 5496)	Returns the adapter object that implements the Values property.
V	ImplHasValue (see page 5496)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
	NextValueIteration (see page 5496)	Implements IWebGetFieldValues (see page 1092)' NextValueIteration method.
	StartValueIterator (see page 5497)	Implements IWebGetFieldValues (see page 1092)' StartValueIterator method.

54.1.40.4.1 TBaseAdapterMultiValueField.EndValueIterator Method

Completes iteration.

Pascal

```
procedure EndValueIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndValueIterator(void * OwnerData);
```

Description

The EndValueIterator procedure is called when iteration has completed.

See Also

NextValueIteration (see page 5496)

StartValueIterator (see page 5497)

54.1.40.4.2 TBaseAdapterMultiValueField.GetCurrentValue Method

Implements IWebGetFieldValues (see page 1092)' GetCurrentValue.

Pascal

```
function GetCurrentValue: Variant;
```

C++

```
__fastcall Variant GetCurrentValue();
```

Description

GetCurrentValue returns the result of ImplGetCurrentValue (see page 5495).

See Also

GetCurrentValue (see page 1094)

ImplGetCurrentValue (see page 5495)

54.1.40.4.3 TBaseAdapterMultiValueField.GetFieldValuesAdapter Method

Returns the adapter object that implements the Values property.

Pascal

```
function GetFieldValuesAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetFieldValuesAdapter();
```

Description

GetFieldValuesAdapter is called when server-side script references the Values property of this adapter field. GetFieldValuesAdapter calls ImplGetFieldValuesAdapter (see page 5496).

See Also

ImplGetFieldValuesAdapter (see page 5496)

54.1.40.4.4 TBaseAdapterMultiValueField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount returns 0. Descendants of TBaseAdapterMultiValueField (see page 5488) override this method. This method will be called when server-side script is used to iterate over field values using the Values property.

See Also

GetValues (see page 5495)

54.1.40.4.5 TBaseAdapterMultiValueField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns Unassigned. Descendants override this method. This method will be called when server-side script is used to iterate over field values using the Values property.

See Also

GetValueCount (see page 5494)

54.1.40.4.6 TBaseAdapterMultiValueField.HasValue Method

Implements IWebGetFieldValues (see page 1092)' HasValue method.

Pascal

```
function HasValue(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean HasValue(const Variant AValue);
```

Description

HasValue returns the result of ImplHasValue (see page 5496).

See Also

HasValue (see page 1094)

ImplHasValue (see page 5496)

54.1.40.4.7 TBaseAdapterMultiValueField.ImplGetCurrentValue Method

Implements IWebGetFieldValues (see page 1092)' GetCurrentValue (see page 5494) method.

Pascal

```
function ImplGetCurrentValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetCurrentValue();
```

Description

ImplGetCurrentValue is called by GetCurrentValue (see page 5494). ImplGetCurrentValue calls GetValues (see page 5495) with the value of the internal iterator index.

See Also

GetCurrentValue (see page 5494)

GetValues (see page 5495)

54.1.40.4.8 TBaseAdapterMultiValueField.ImplGetFieldValuesAdapter Method

Returns the adapter object that implements the Values property.

Pascal

```
function ImplGetFieldValuesAdapter: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetFieldValuesAdapter();
```

Description

ImplGetFieldValuesAdapter returns an instance of TFieldValuesAdapter (see page 5803).

See Also

TFieldValuesAdapter (see page 5803)

GetFieldValuesAdapter (see page 5494)

Create (see page 5492)

Destroy (see page 5492)

54.1.40.4.9 TBaseAdapterMultiValueField.ImplHasValue Method

Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.

Pascal

```
function ImplHasValue(const Value: Variant): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasValue(const Variant Value);
```

Description

ImplHasValue is called by HasValue (see page 5495). ImplHasValue returns False (indicating that there is no matching value). Descendants override this method.

See Also

HasValue (see page 5495)

54.1.40.4.10 TBaseAdapterMultiValueField.NextValueIteration Method

Implements IWebGetFieldValues (see page 1092)' NextValueIteration method.

Pascal

```
function NextValueIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextValueIteration(void * OwnerData);
```

Description

NextValueIteration increments an internal iterator index. false is returned if the iterator index is not less than the GetValueCount (see page 5494) .

See Also

NextValueIteration (see page 1094)

GetValueCount (see page 5494)

54.1.40.4.11 TBaseAdapterMultiValueField.StartValueIterator Method

Implements IWebGetFieldValues (see page 1092)' StartValueIterator method.

Pascal

```
function StartValueIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartValueIterator(void * OwnerData);
```

Description

StartValueIterator initializes an internal iterator index to zero. false is returned if GetValueCount (see page 5494) is zero.

See Also

StartValueIterator (see page 1095)

GetValueCount (see page 5494)

54.1.41 TBaseEndUserAdapter Class

TBaseEndUserAdapter contains user information.

Class Hierarchy

Pascal

```
TBaseEndUserAdapter = class(TDefaultFieldsAdapter, IWebEndUser);
```

C++

```
class TBaseEndUserAdapter : public TDefaultFieldsAdapter, public IWebEndUser;
```

File

WebAdapt

Description

TBaseEndUserAdapter contains information about a user such as their name, access rights, and whether they are logged in. TBaseEndUserAdapter has default fields to represent the end user's display name and login state. It has default actions for logging a user in and logging a user out. End user adapters are responsible validating end user access. For example, when the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access the page.

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

54.1.41.1 TBaseEndUserAdapter Members

The following tables list the members exposed by TBaseEndUserAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5504)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5504)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5505)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5505)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5505)	Starts the server-side login process.
V	ExecuteLoginForm (see page 5506)	Creates the HTML login page.
V	ExecuteLogout (see page 5506)	Performs server-side logout.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.

	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5506)	Retrieves the display name string.
	GetDisplayNameValue (see page 5507)	Gets the display name of the end user.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5507)	Gets the end-user adapter.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5507)	Determines if the user is logged in
	GetLoggedInValue (see page 5508)	Determines if the user is logged in.
	GetLoginFormAction (see page 5508)	Gets the action component for the login form.
	GetLogoutAction (see page 5508)	Gets the action component for the logout form.
	GetUserID (see page 5509)	Gets the UserID from this TBaseEndUserAdapter (see page 5497).
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasExecuteRights (see page 5509)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5497).
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasModifyRights (see page 5509)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5497).
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5510)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5497).
	ImplCanAddActionClass (see page 5510)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 5510)	Determines if the given action class can be added.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5511)	Lists supported end-user adapter actions.
	ImplGetDisplayName (see page 5511)	Gets the display name for this TBaseEndUserAdapter (see page 5497).
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 5511)	Returns this TBaseEndUserAdapter (see page 5497).
	ImplGetFieldsList (see page 5512)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 5512)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5512)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5497).
	ImplGetLogoutAction (see page 5513)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5497).
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.

	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5513)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5513)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplHasViewRights (see page 5514)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5514)	Processes the login for the given user.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5514)	Generate the HTML login page.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5515)	Processes the server-side login for the given user.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5515)	Generates the login page.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5515)	Gets the user list.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5516)	Determines if the user rights match the rights being queried.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.
	OnProduceLoginPage (see page 5516)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	Actions (see page 5572)	Actions is the container for adapter actions components.
	Actions (see page 5746)	Container for adapter actions components.
	Data (see page 5572)	Data is the container for adapter field components.
	Data (see page 5747)	Container for adapter field components.
	DisplayName (see page 5994)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
	EndUserAdapter (see page 5994)	Provides access to the end user adapter that implements this interface.
	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	LoggedIn (see page 5994)	Indicates whether the current user is logged in.
	LoginFormAction (see page 5995)	Indicates the action component for the login form.
	LoginPage (see page 5516)	Specifies the name of the page that acts as the login page.
	LogoutAction (see page 5996)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.41.2 TBaseEndUserAdapter Methods

The methods of the TBaseEndUserAdapter class are listed here.

Protected Methods

	Name	Description
	CanExecuteLoginForm (see page 5504)	Determines if there is a static login page.
	CanExecuteLogout (see page 5504)	Determines if the user is logged in.
	CanProduceLoginPage (see page 5505)	Determines whether there is a static login page or an automatically produced login page.
	DoOnHasRights (see page 5505)	Determines whether the user has view rights.
	ExecuteLogin (see page 5505)	Starts the server-side login process.
	ExecuteLoginForm (see page 5506)	Creates the HTML login page.
	ExecuteLogout (see page 5506)	Performs server-side logout.
	GetDisplayName (see page 5506)	Retrieves the display name string.
	GetDisplayNameValue (see page 5507)	Gets the display name of the end user.
	GetEndUserAdapter (see page 5507)	Gets the end-user adapter.
	GetLoggedIn (see page 5507)	Determines if the user is logged in
	GetLoggedInValue (see page 5508)	Determines if the user is logged in.
	GetLoginFormAction (see page 5508)	Gets the action component for the login form.
	GetLogoutAction (see page 5508)	Gets the action component for the logout form.
	GetUserID (see page 5509)	Gets the UserID from this TBaseEndUserAdapter (see page 5497).
	HasExecuteRights (see page 5509)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5497).
	HasModifyRights (see page 5509)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5497).
	HasViewRights (see page 5510)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5497).
	ImplCanAddActionClass (see page 5510)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 5510)	Determines if the given action class can be added.
	ImplGetActionsList (see page 5511)	Lists supported end-user adapter actions.

	ImplGetDisplayName (see page 5511)	Gets the display name for this TBaseEndUserAdapter (see page 5497).
	ImplGetEndUserAdapter (see page 5511)	Returns this TBaseEndUserAdapter (see page 5497).
	ImplGetFieldsList (see page 5512)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetLoggedIn (see page 5512)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5512)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5497).
	ImplGetLogoutAction (see page 5513)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5497).
	ImplHasExecuteRights (see page 5513)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyRights (see page 5513)	Checks user rights and determines if the user has modify rights.
	ImplHasViewRights (see page 5514)	Checks user rights and determines if the user has view rights.
	ImplLogIn (see page 5514)	Processes the login for the given user.
	ImplProduceLoginPage (see page 5514)	Generate the HTML login page.
	LogIn (see page 5515)	Processes the server-side login for the given user.
	ProduceLoginPage (see page 5515)	Generates the login page.
	UserList (see page 5515)	Gets the user list.

54.1.41.2.1 TBaseEndUserAdapter.CanExecuteLoginForm Method

Determines if there is a static login page.

Pascal

```
function CanExecuteLoginForm: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanExecuteLoginForm();
```

Description

The CanExecuteLoginForm calls CanProduceLoginPage (see page 5505) to determine if there is a static login page, or if one can be generated. Then if the user is not already logged in, this function returns true. If the login page cannot be produced or the user is already logged in, then this function returns false.

See Also

CanProduceLoginPage (see page 5505)

54.1.41.2.2 TBaseEndUserAdapter.CanExecuteLogout Method

Determines if the user is logged in.

Pascal

```
function CanExecuteLogout: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanExecuteLogout();
```

Description

The CanExecuteLogout function calls ImplGetLoggedIn (see page 5512) to determine if the user is logged in. If the user is logged in, this function returns true.

See Also

ImplGetLoggedIn (see page 5512)

54.1.41.2.3 TBaseEndUserAdapter.CanProduceLoginPage Method

Determines whether there is a static login page or an automatically produced login page.

Pascal

```
function CanProduceLoginPage: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean CanProduceLoginPage();
```

Description

The CanProduceLoginPage returns true if there is a static login page, or if there is an event handler defined to automatically produce the login page.

See Also

OnProduceLoginPage (see page 5516)

54.1.41.2.4 TBaseEndUserAdapter.DoOnHasRights Method

Determines whether the user has view rights.

Pascal

```
function DoOnHasRights(AEvent: TEndUserRightsEvent; const ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean DoOnHasRights(TEndUserRightsEvent AEvent, const UnicodeString
ARights);
```

Description

The DoOnHasRights function is called from ImplHasViewRights (see page 5514). It matches the given access rights to the rights in the user list and returns true if the user has view rights.

AEvent is the event handler that DoOnHasRights should call first to check access rights.

ARights is a string that represents the rights the user needs.

See Also

ImplHasViewRights (see page 5514)

HasViewRights (see page 5510)

54.1.41.2.5 TBaseEndUserAdapter.ExecuteLogin Method

Starts the server-side login process.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); virtual;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

The ExecuteLogin function is called from sever-side script to start the login process. This base class version of the function is empty. A customized overridden version of this function can be written to do specific login processing for the application.

See Also

ImplProduceLoginPage (see page 5514)

54.1.41.2.6 TBaseEndUserAdapter.ExecuteLoginForm Method

Creates the HTML login page.

Pascal

```
procedure ExecuteLoginForm; virtual;
```

C++

```
virtual __fastcall ExecuteLoginForm();
```

Description

The ExecuteLoginForm function is called from server-side script to create the HTML login page. This function calls ImplProduceLoginPage (see page 5514) to make the page producer generate the login page.

See Also

ImplProduceLoginPage (see page 5514)

54.1.41.2.7 TBaseEndUserAdapter.ExecuteLogout Method

Performs server-side logout.

Pascal

```
procedure ExecuteLogout; virtual;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

The ExecuteLogout function is called from server-side script to perform the logout process. This base class version of the function is empty. A customized overridden version of this function can be written to do specific logout processing for the application.

See Also

CanExecuteLogout (see page 5504)

54.1.41.2.8 TBaseEndUserAdapter.GetDisplayName Method

Retrieves the display name string.

Pascal

```
function GetDisplayName: string;
```

C++

```
__fastcall UnicodeString GetDisplayName();
```

Description

The GetDisplayName function is called from server-side script to get the display name string. It calls ImplGetDisplayName (see page 5511) to get the display name for this TBaseEndUserAdapter (see page 5497).

See Also

ImplGetDisplayName (see page 5511)

GetDisplayNameValue (see page 5507)

54.1.41.2.9 TBaseEndUserAdapter.GetDisplayNameValue Method

Gets the display name of the end user.

Pascal

```
function GetDisplayNameValue: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDisplayNameValue();
```

Description

The GetDisplayNameValue is called from server-side script to get the display name of the end user. This base class version of the function returns an empty string. A customized overridden version of this function could be written to return the display name for the end user in your application.

See Also

ImplGetDisplayName (see page 5511)

GetDisplayName (see page 5506)

54.1.41.2.10 TBaseEndUserAdapter.GetEndUserAdapter Method

Gets the end-user adapter.

Pascal

```
function GetEndUserAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetEndUserAdapter();
```

Description

The GetEndUserAdapter function is called from server-side script to get the end-user adapter. This function returns itself, that is, this TBaseEndUserAdapter (see page 5497).

See Also

ImplGetEndUserAdapter (see page 5511)

54.1.41.2.11 TBaseEndUserAdapter.GetLoggedIn Method

Determines if the user is logged in

Pascal

```
function GetLoggedIn: Boolean;
```

C++

```
__fastcall Boolean GetLoggedIn();
```

Description

The GetLoggedIn function is called from server-side script to determine if the user is logged in. This function calls

ImplGetLoggedIn (see page 5512) to get the result.

See Also

ImplGetLoggedIn (see page 5512)

GetLoggedInValue (see page 5508)

54.1.41.2.12 TBaseEndUserAdapter.GetLoggedInValue Method

Determines if the user is logged in.

Pascal

```
function GetLoggedInValue: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

The GetLoggedInValue function is called from server-side script to determine if the user is logged in. This base class version of the function always returns false. A customized overridden version could be written specifically for your application.

See Also

ImplGetLoggedIn (see page 5512)

GetLoggedIn (see page 5507)

54.1.41.2.13 TBaseEndUserAdapter.GetLoginFormAction Method

Gets the action component for the login form.

Pascal

```
function GetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * GetLoginFormAction();
```

Description

The GetLoginFormAction is called from server-side script to get the action component for the login form. This function calls ImplGetLoginFormAction (see page 5512) to get the action component.

See Also

ImplGetLoginFormAction (see page 5512)

54.1.41.2.14 TBaseEndUserAdapter.GetLogoutAction Method

Gets the action component for the logout form.

Pascal

```
function GetLogoutAction: TComponent;
```

C++

```
__fastcall TComponent * GetLogoutAction();
```

Description

The GetLogoutAction is called from server-side script to get the action component for the logout form. This function calls

ImplGetLogoutFormAction to get the action component.

See Also

ImplGetLogoutAction (see page 5513)

54.1.41.2.15 TBaseEndUserAdapter.GetUserID Method

Gets the UserID from this TBaseEndUserAdapter (see page 5497).

Pascal

```
function GetUserID: Variant; virtual;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

The GetUserID function is called from server-side script to get the UserID from this TBaseEndUserAdapter (see page 5497). This base class version of the function always returns Unassigned. A customized overridden version of this function could be written specifically for your application.

54.1.41.2.16 TBaseEndUserAdapter.HasExecuteRights Method

Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5497).

Pascal

```
function HasExecuteRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasExecuteRights(UnicodeString ARights);
```

Description

The HasExecuteRights function is called from server-side script to determine if the user has execute rights for the TBaseEndUserAdapter (see page 5497). This function calls ImplHasExecuteRights (see page 5513) to check the user rights.

See Also

ImplHasExecuteRights (see page 5513)

HasModifyRights (see page 5509)

HasViewRights (see page 5510)

54.1.41.2.17 TBaseEndUserAdapter.HasModifyRights Method

Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5497).

Pascal

```
function HasModifyRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasModifyRights(UnicodeString ARights);
```

Description

The HasModifyRights function is called from server-side script to determine if the user has modify rights for the TBaseEndUserAdapter (see page 5497). This function calls ImplHasModifyRights (see page 5513) to check the user rights.

See Also

ImplHasModifyRights (see page 5513)

HasViewRights (see page 5510)

54.1.41.2.18 TBaseEndUserAdapter.HasViewRights Method

Determine if the user has view rights to the TBaseEndUserAdapter (see page 5497).

Pascal

```
function HasViewRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasViewRights(UnicodeString ARights);
```

Description

The HasViewRights function is called from server-side script to determine if the user has view rights for the TBaseEndUserAdapter (see page 5497). This function calls ImplHasViewRights (see page 5514) to check the user rights.

See Also

ImplHasViewRights (see page 5514)

HasModifyRights (see page 5509)

54.1.41.2.19 TBaseEndUserAdapter.ImplCanAddActionClass Method

Determines if the given action class can be added.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddActionClass function calls the base class to determine if the given action class can be added. In addition, it allows actions components to be added if they are derived from TCustomEndUserAdapterAction (see page 5687) at some level. This function returns true if the action class is allowed to be added, or false otherwise.

See Also

ImplCanAddActionClass (see page 5553)

TCustomEndUserAdapterAction (see page 5687)

54.1.41.2.20 TBaseEndUserAdapter.ImplCanAddFieldClass Method

Determines if the given action class can be added.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddFieldClass function calls the base class to determine if the given action class can be added. In addition to that, it allows fields components to be added if they are derived from TAdapterEndUserField (see page 5248).

See Also

ImplCanAddFieldClass (see page 5553)

TAdapterEndUserField (see page 5248)

54.1.41.2.21 TBaseEndUserAdapter.ImplGetActionsList Method

Lists supported end-user adapter actions.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

The ImplGetActionsList procedure obtains the list of all supported actions for the end-user adapter.

54.1.41.2.22 TBaseEndUserAdapter.ImplGetDisplayName Method

Gets the display name for this TBaseEndUserAdapter (see page 5497).

Pascal

```
function ImplGetDisplayName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayName();
```

Description

The ImplGetDisplayName function is called from GetDisplayName (see page 5506) to get the display name for this TBaseEndUserAdapter (see page 5497). It gets the display name from the DisplayName field component.

See Also

GetDisplayName (see page 5506)

54.1.41.2.23 TBaseEndUserAdapter.ImplGetEndUserAdapter Method

Returns this TBaseEndUserAdapter (see page 5497).

Pascal

```
function ImplGetEndUserAdapter: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetEndUserAdapter();
```

Description

The ImplGetEndUserAdapter simply returns this TBaseEndUserAdapter (see page 5497).

See Also

GetEndUserAdapter (see page 5507)

54.1.41.2.24 TBaseEndUserAdapter.ImplGetFieldsList Method

Obtains the list of all of the fields for the end-user adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

The ImplGetFieldsList procedure obtains the list of all of the fields for the end-user adapter.

54.1.41.2.25 TBaseEndUserAdapter.ImplGetLoggedIn Method

Checks the value of the LoggedIn field.

Pascal

```
function ImplGetLoggedIn: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetLoggedIn();
```

Description

The ImplGetLoggedIn function is called from GetLoggedIn (see page 5507). It checks the value of the LoggedIn field component and returns the result.

See Also

GetLoggedIn (see page 5507)

54.1.41.2.26 TBaseEndUserAdapter.ImplGetLoginFormAction Method

Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5497).

Pascal

```
function ImplGetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * ImplGetLoginFormAction();
```

Description

The ImplGetLoginFormAction function is called from GetLoginFormAction (see page 5508) to get the LoginForm action component for this TBaseEndUserAdapter (see page 5497).

See Also

GetLoginFormAction (see page 5508)

54.1.41.2.27 TBaseEndUserAdapter.ImplGetLogoutAction Method

Gets the Logout action component for this TBaseEndUserAdapter (see page 5497).

Pascal

```
function ImplGetLogoutAction: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetLogoutAction();
```

Description

The ImplGetLogoutAction function is called from GetLogoutAction (see page 5508) to get the Logout action component for this TBaseEndUserAdapter (see page 5497).

See Also

GetLogoutAction (see page 5508)

54.1.41.2.28 TBaseEndUserAdapter.ImplHasExecuteRights Method

Checks user rights and determines if the user has execute rights.

Pascal

```
function ImplHasExecuteRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteRights(UnicodeString ARights);
```

Description

The ImplHasExecuteRights function is called from the HasExecuteRights (see page 5509) function. It calls the DoOnHasRights (see page 5505) function to check the user rights and determines if the user has execute rights.

See Also

OnHasRights (see page 5516)

54.1.41.2.29 TBaseEndUserAdapter.ImplHasModifyRights Method

Checks user rights and determines if the user has modify rights.

Pascal

```
function ImplHasModifyRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyRights(UnicodeString ARights);
```

Description

The ImplHasModifyRights function is called from the HasModifyRights (see page 5509) function. It calls the DoOnHasRights (see page 5505) function to check the user rights and determine if the user has modify rights.

See Also

OnHasRights (see page 5516)

54.1.41.2.30 TBaseEndUserAdapter.ImplHasViewRights Method

Checks user rights and determines if the user has view rights.

Pascal

```
function ImplHasViewRights(ARights: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewRights(UnicodeString ARights);
```

Description

The ImplHasViewRights is called from the HasViewRights (see page 5510) function. It calls the DoOnHasRights (see page 5505) function to check the user rights and determine if the user has view rights.

See Also

DoOnHasRights (see page 5505)

54.1.41.2.31 TBaseEndUserAdapter.ImplLogIn Method

Processes the login for the given user.

Pascal

```
procedure ImplLogIn(AUserID: Variant); virtual;
```

C++

```
virtual __fastcall ImplLogIn(Variant AUserID);
```

Description

The ImplLogIn procedure is called from LogIn (see page 5515). This function calls ExecuteLogin (see page 5505) to process the login for the given user.

See Also

LogIn (see page 5515)

ExecuteLogin (see page 5505)

54.1.41.2.32 TBaseEndUserAdapter.ImplProduceLoginPage Method

Generate the HTML login page.

Pascal

```
function ImplProduceLoginPage: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplProduceLoginPage();
```

Description

The ImplProduceLoginPage is called to generate the HTML login page It first calls the OnProduceLoginPage (see page 5516) event handler to allow for a customized version. If no event handler is defined, it then has default code to generate the page using the page producer.

See Also

OnProduceLoginPage (see page 5516)

54.1.41.2.33 TBaseEndUserAdapter.LogIn Method

Processes the server-side login for the given user.

Pascal

```
procedure LogIn(AUserID: Variant);
```

C++

```
__fastcall LogIn(Variant AUserID);
```

Description

The LogIn procedure is called from server-side script to process the login for the given user. This function calls ImplLogIn (see page 5514) to process the user.

See Also

ImplLogIn (see page 5514)

54.1.41.2.34 TBaseEndUserAdapter.ProduceLoginPage Method

Generates the login page.

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

The ProduceLoginPage function is called from server-side script to generate the login page. This function calls ImplProduceLoginPage (see page 5514) to generate the page.

See Also

ImplProduceLoginPage (see page 5514)

54.1.41.2.35 TBaseEndUserAdapter.UserList Method

Gets the user list.

Pascal

```
function UserList: IWebUserList;
```

C++

```
__fastcall IWebUserList UserList();
```

Description

The UserList function is called from server-side script to get the user list. If the user list is not stored in this TBaseEndUserAdapter (see page 5497), it then attempts to retrieve the user list through the application module.

54.1.41.3 TBaseEndUserAdapter Properties

The properties of the TBaseEndUserAdapter class are listed here.

Public Properties

	Name	Description
	LoginPage (see page 5516)	Specifies the name of the page that acts as the login page.

54.1.41.3.1 TBaseEndUserAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property UnicodeString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter (see page 5497). The name of the login page is the name of the Web page module that generates the login page.

54.1.41.4 TBaseEndUserAdapter Events

The events of the TBaseEndUserAdapter class are listed here.

Public Events

	Name	Description
	OnHasRights (see page 5516)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 5516)	Allows implementation of a customized login page.

54.1.41.4.1 TBaseEndUserAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights (see page 5505) to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 5505)

ImplHasExecuteRights (see page 5513)

ImplHasModifyRights (see page 5513)

ImplHasViewRights (see page 5514)

54.1.41.4.2 TBaseEndUserAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage (see page 5514). If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 5514)

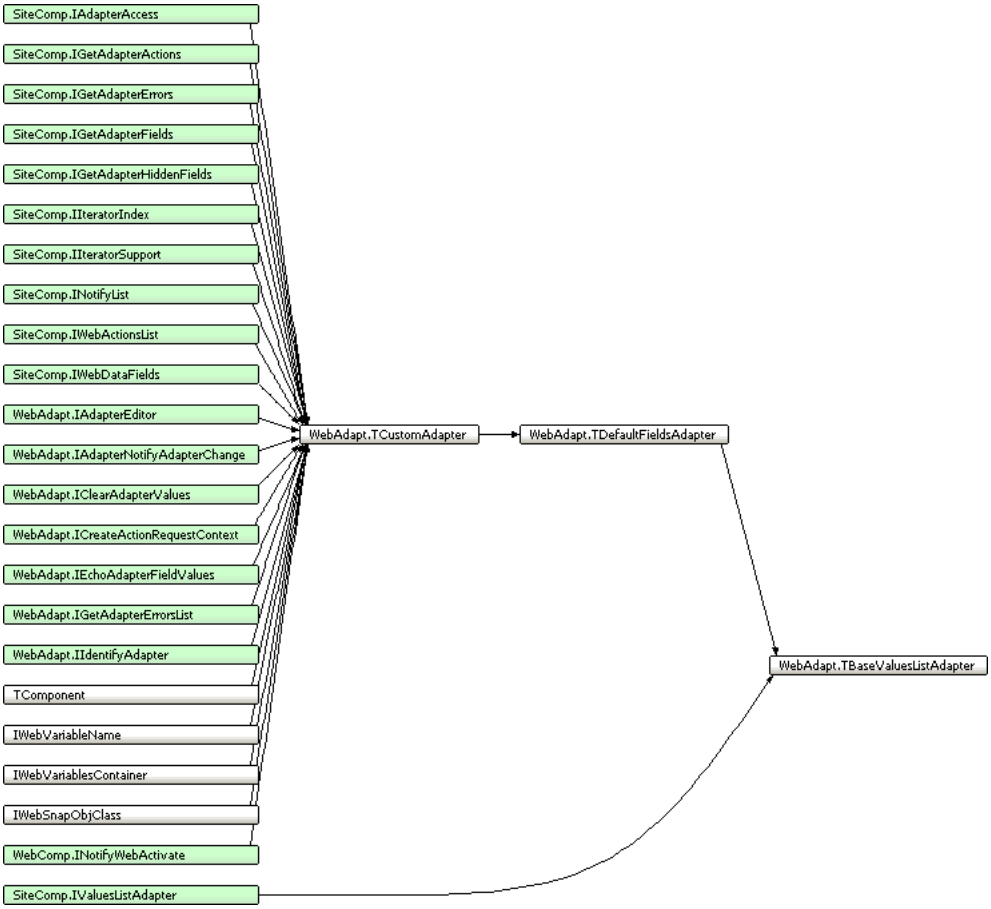
ProduceLoginPage (see page 5515)

CanProduceLoginPage (see page 5505)

54.1.42 TBaseValuesListAdapter Class

TBaseValuesListAdapter is a base class for adapters that provide a list of name/value pairs.

Class Hierarchy



Pascal

```
TBaseValuesListAdapter = class(TDefaultFieldsAdapter, IValuesListAdapter);
```

C++

```
class TBaseValuesListAdapter : public TDefaultFieldsAdapter, public IValuesListAdapter;
```

File

WebAdapt

Description

Descendants of TBaseValuesListAdapter (such as TStringsValuesList (see page 5857) and TDataSetValuesList) are used to provide name/value pairs for an adapter field. Adapter fields reference a values list adapter using the adapter field's ValuesList property. Typically, the list of name/value pairs is used by server-side scripts to populate <select>, <input type=checkbox>, or <input type=radio>.

TBaseValuesListAdapter has two default fields. One field is responsible for providing a value and the other for providing a name.

See Also

TStringsValuesList (see page 5857)

TDataSetValuesList

TAdapterUpdateField (see page 5419)

54.1.42.1 TBaseValuesListAdapter Members

The following tables list the members exposed by TBaseValuesListAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.

	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5522)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5523)	Looks up an image for a value.
	GetListName (see page 5523)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5523)	Looks up a name for a value.
	GetListValue (see page 5524)	Responsible for returning the value of the name/value pair.
	GetNameFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetValueFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5524)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5525)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5525)	Adds default field components to this adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 5525)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 5526)	Looks up an image for a value.
	ImplGetListName (see page 5526)	Responsible returning the name of the name/value pair.

	ImplGetListNameOfValue (see page 5526)	Looks up a name for a value.
	ImplGetListValue (see page 5527)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.

	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	Actions (see page 5572)	Actions is the container for adapter actions components.
	Actions (see page 5746)	Container for adapter actions components.
	Data (see page 5572)	Data is the container for adapter field components.
	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.42.2 TBaseValuesListAdapter Methods

The methods of the TBaseValuesListAdapter class are listed here.

Protected Methods

	Name	Description
	GetListImage (see page 5522)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5523)	Looks up an image for a value.
	GetListName (see page 5523)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5523)	Looks up a name for a value.
	GetListValue (see page 5524)	Responsible for returning the value of the name/value pair.
	GetNameFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetValueFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetWebSnapObjClass (see page 5524)	Returns the class type for this adapter field.
	ImplCanAddFieldClass (see page 5525)	Indicates whether a type of adapter field can be added to this adapter.
	ImplGetFieldsList (see page 5525)	Adds default field components to this adapter.
	ImplGetListImage (see page 5525)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 5526)	Looks up an image for a value.
	ImplGetListName (see page 5526)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 5526)	Looks up a name for a value.
	ImplGetListValue (see page 5527)	Responsible for returning the value of the name/value pair.

54.1.42.2.1 TBaseValuesListAdapter.GetListImage Method

Returns an image associated with the current name/value pair.

Pascal

```
function GetListImage: TComponent;
```

C++

```
__fastcall TComponent * GetListImage();
```

Description

GetListImage returns the result of ImplGetListImage (see page 5525).

See Also

ImplGetListImage (see page 5525)

54.1.42.2.2 TBaseValuesListAdapter.GetListImageOfValue Method

Looks up an image for a value.

Pascal

```
function GetListImageOfValue(Value: Variant): TComponent;
```

C++

```
__fastcall TComponent * GetListImageOfValue(Variant Value);
```

Description

GetListImageOfValue returns the result of ImplGetListImageOfValue (see page 5526).

See Also

ImplGetListImageOfValue (see page 5526)

54.1.42.2.3 TBaseValuesListAdapter.GetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function GetListName: string;
```

C++

```
__fastcall UnicodeString GetListName();
```

Description

GetListName returns the result of ImplGetListName (see page 5526).

See Also

ImplGetListValue (see page 5527)

54.1.42.2.4 TBaseValuesListAdapter.GetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function GetListNameOfValue(Value: Variant): string;
```

C++

```
__fastcall UnicodeString GetListNameOfValue(Variant Value);
```

Description

GetListNameOfValue returns the result of ImplGetListNameOfValue (see page 5526).

See Also

ImplGetListNameOfValue (see page 5526)

54.1.42.2.5 TBaseValuesListAdapter.GetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function GetListValue: Variant;
```

C++

```
__fastcall Variant GetListValue();
```

Description

GetListValue returns the result of ImplGetListValue (see page 5527).

See Also

GetListName (see page 5523)

ImplGetListValue (see page 5527)

54.1.42.2.6 TBaseValuesListAdapter.GetNameFieldClass Method

Returns the class of adapter field that is responsible for providing the value of the name/value pair.

Pascal

```
function GetNameFieldClass: TComponentClass; virtual;
```

C++

```
virtual __fastcall TComponentClass GetNameFieldClass();
```

Description

GetNameFieldClass returns nil (Delphi) or NULL (C++). Descendant classes must override this method.

See Also

GetValueFieldClass (see page 5524)

54.1.42.2.7 TBaseValuesListAdapter.GetValueFieldClass Method

Returns the class of adapter field that is responsible for providing the name of the name/value pair.

Pascal

```
function GetValueFieldClass: TComponentClass; virtual;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns nil (Delphi) or NULL (C++). Descendant classes must override this method.

See Also

GetNameFieldClass (see page 5524)

54.1.42.2.8 TBaseValuesListAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; override;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

54.1.42.2.9 TBaseValuesListAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass check the field class by calling GetNameFieldClass (see page 5524), GetValueFieldClass (see page 5524), and TCustomAdapter (see page 5531)'s ImplCanAddFieldClass method.

See Also

GetNameFieldClass (see page 5524)

GetValueFieldClass (see page 5524)

ImplCanAddFieldClass (see page 5553)

54.1.42.2.10 TBaseValuesListAdapter.ImplGetFieldsList Method

Adds default field components to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList adds a value field and a name field using the classes returned by GetNameFieldClass (see page 5524) and GetValueFieldClass (see page 5524).

See Also

GetNameFieldClass (see page 5524)

GetValueFieldClass (see page 5524)

54.1.42.2.11 TBaseValuesListAdapter.ImplGetListImage Method

Responsible returning an image associated with the current name/value pair.

Pascal

```
function ImplGetListImage: TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetListImage();
```

Description

ImplGetListImage is a placeholder. Descendant adapters that support images must override this method.

See Also

GetListImage (see page 5522)

54.1.42.2.12 TBaseValuesListAdapter.ImplGetListImageOfValue Method

Looks up an image for a value.

Pascal

```
function ImplGetListImageOfValue(Value: Variant): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetListImageOfValue(Variant Value);
```

Description

ImplGetListImageOfValue is a placeholder. Descendant classes that support image lookups must override this function.

See Also

GetListImageOfValue (see page 5523)

54.1.42.2.13 TBaseValuesListAdapter.ImplGetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function ImplGetListName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetListName();
```

Description

ImplGetListName returns an empty string. ImplGetListName must be overridden by a descendant class.

See Also

GetListName (see page 5523)

54.1.42.2.14 TBaseValuesListAdapter.ImplGetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function ImplGetListNameOfValue(Value: Variant): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetListNameOfValue(Variant Value);
```

Description

ImplGetListNameOfValue is a placeholder. Descendant classes that support name lookups must override this function.

See Also

GetListNameOfValue (see page 5523)

54.1.42.2.15 TBaseValuesListAdapter.ImplGetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function ImplGetListValue: Variant; virtual;
```

C++

```
virtual __fastcall Variant ImplGetListValue();
```

Description

ImplGetListValue is a placeholder. ImplGetListValue must be overriden by a descendant class.

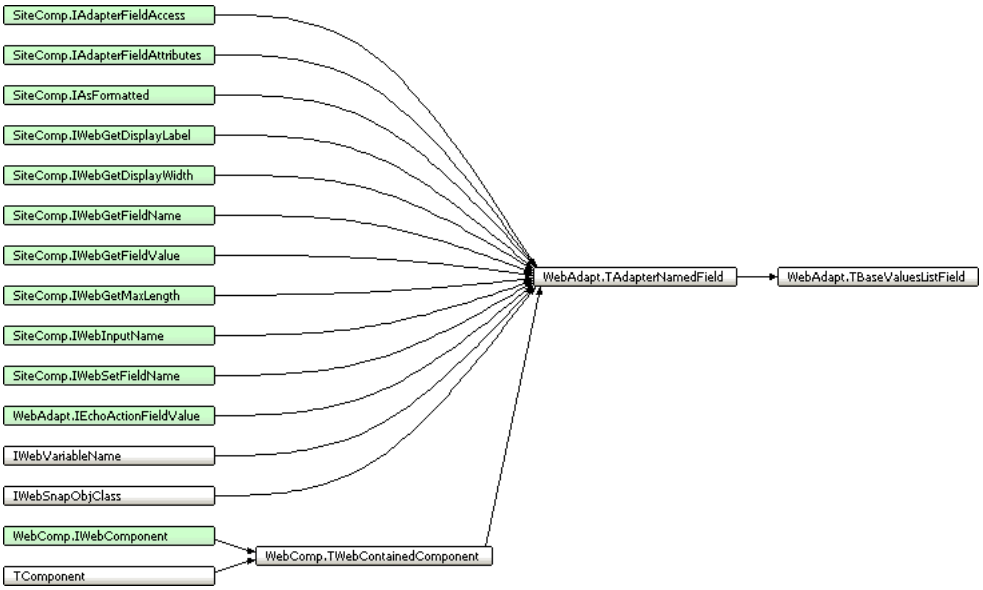
See Also

GetListValue (see page 5524)

54.1.43 TBaseValuesListField Class

TBaseValuesListField is a base class for adapter field components that are children of a values list adapter component.

Class Hierarchy



Pascal

```
TBaseValuesListField = class(TAdapterNamedField);
```

C++

```
class TBaseValuesListField : public TAdapterNamedField;
```

File

WebAdapt

Description

Derive from TBaseValuesListField to declare an adapter field that can be a child of TBaseValuesListAdapter (see page 5517).

See Also

TStringsValuesList (see page 5857)

TBaseValuesListAdapter (see page 5517)

TAdapterNamedField (see page 5351)

TValuesListNameField (see page 5866)

TValuesListValueField (see page 5869)

54.1.43.1 TBaseValuesListField Members

The following tables list the members exposed by TBaseValuesListField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5530)	Retrieves the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.

	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5531)	Retrieves the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.43.2 TBaseValuesListField Methods

The methods of the TBaseValuesListField class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 5530)	Retrieves the parent adapter component.

54.1.43.2.1 TBaseValuesListField.GetAdapter Method

Retrieves the parent adapter component.

Pascal

```
function GetAdapter: TBaseValuesListAdapter;
```

C++

```
__fastcall TBaseValuesListAdapter GetAdapter();
```

Description

GetAdapter implements the Adapter (see page 5531) property.

See Also

Adapter (see page 5531)

TBaseValuesListAdapter (see page 5517)

54.1.43.3 TBaseValuesListField Properties

The properties of the TBaseValuesListField class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5531)	Retrieves the parent adapter component.

54.1.43.3.1 TBaseValuesListField.Adapter Property

Retrieves the parent adapter component.

Pascal

```
property Adapter: TBaseValuesListAdapter;
```

C++

```
__property TBaseValuesListAdapter Adapter;
```

Description

The Adapter property is used by descendants to access the parent adapter component.

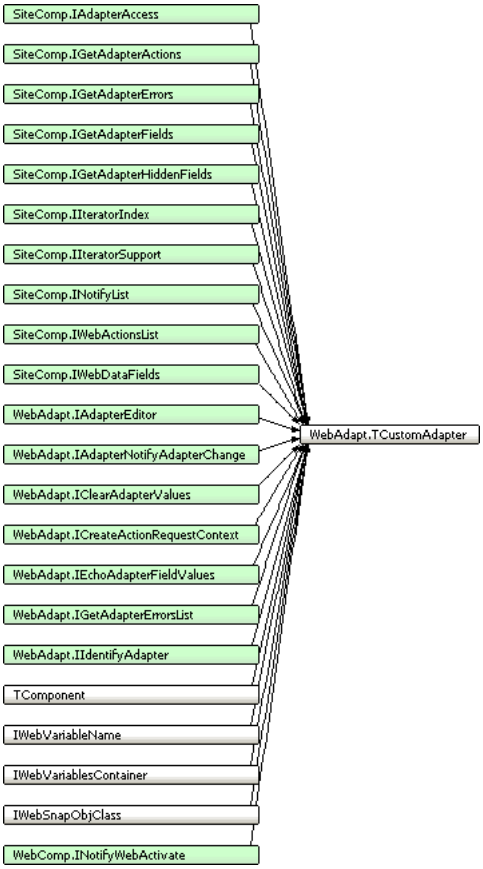
See Also

TBaseValuesListAdapter (see page 5517)

54.1.44 TCustomAdapter Class

TCustomAdapter is a base class for adapter components.

Class Hierarchy



Pascal

```
TCustomAdapter = class(TComponent, IIdentifyAdapter, IWebVariableName, IWebVariablesContainer,
INotifyList, IWebSnapObjClass, IGetAdapterErrors, IGetAdapterErrorsList, INotifyWebActivate,
IAdapterEditor, IGetAdapterFields, IGetAdapterActions, IIteratorSupport, IClearAdapterValues,
IEchoAdapterFieldValues, IAdapterAccess, IGetAdapterHiddenFields, ICreateActionRequestContext,
IWebDataFields, IWebActionsList, IAdapterNotifyAdapterChange, IIteratorIndex);
```

C++

```
class TCustomAdapter : public TComponent, public IIdentifyAdapter, public IWebVariableName,
public IWebVariablesContainer, public INotifyList, public IWebSnapObjClass, public
IGetAdapterErrors, public IGetAdapterErrorsList, public INotifyWebActivate, public
IAdapterEditor, public IGetAdapterFields, public IGetAdapterActions, public IIteratorSupport,
public IClearAdapterValues, public IEchoAdapterFieldValues, public IAdapterAccess, public
IGetAdapterHiddenFields, public ICreateActionRequestContext, public IWebDataFields, public
IWebActionsList, public IAdapterNotifyAdapterChange, public IIteratorIndex;
```

File

WebAdapt

Description

TCustomAdapter implements interfaces used to access an adapter properties using server-side script. Most of the interface implementation methods call virtual functions. When defining new derived classes, override these virtual functions to customize the behavior of the adapter class.

See Also

TAdapter (see page 5190)

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

TFieldValuesAdapter (see page 5803)

TDataSetValuesList

TStringsValuesList (see page 5857)

TApplicationAdapter (see page 5439)

TLoginFormAdapter (see page 5824)

TDataSetAdapter

TPagedAdapter (see page 5844)

54.1.44.1 TCustomAdapter Members

The following tables list the members exposed by TCustomAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>

	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
V	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
V	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
V	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
V	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.

	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
V	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.

	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.

54.1.44.2 TCustomAdapter.Create Constructor

Constructor for TCustomAdapter (see page 5531).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapter(TComponent * AOwner);
```

Description

Creates instance variables that hold interfaces, object references used by TCustomAdapter (see page 5531).

See Also

Create

54.1.44.3 TCustomAdapter.Destroy Destructor

Destructor for TCustomAdapter (see page 5531).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapter();
```

Description

Frees objects used by TCustomAdapter (see page 5531).

See Also

System::TObject::Free (see page 3372)

54.1.44.4 TCustomAdapter Methods

The methods of the TCustomAdapter class are listed here.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.

	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.

	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Public Methods

	Name	Description
	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

54.1.44.4.1 TCustomAdapter.AddAdapterModeHiddenField Method

Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddAdapterModeHiddenField(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddAdapterModeHiddenField(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Descendant adapters may support modes. For example, TDataSetAdapter supports Edit, Insert (see page 3601), and Browse modes. The hidden field defined by AddAdapterModeHiddenField is added to an HTML form using the server-side script method HiddenFields.WriteFields.

See Also

ListHiddenFields (see page 5563)

54.1.44.4.2 TCustomAdapter.AddOrigValueHiddenFields Method

Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddOrigValueHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddOrigValueHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

In order to ensure that the data associated with the adapter hasn't been changed by another user, descendant adapters may check that the current values of adapter fields are the same as the original values. For example, TDataSetAdapter raises an exception if the original values are not the same as the dataset's values. The hidden fields defined by AddOrigValueHiddenFields are added to an HTML form using the server-side script method HiddenRecordFields.WriteFields.

See Also

ListHiddenRecordFields (see page 5564)

54.1.44.4.3 TCustomAdapter.AddRecordKeysHiddenFields Method

Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.

Pascal

```
procedure AddRecordKeysHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall AddRecordKeysHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Descendant adapters may use key values to define the relationship between values in the HTTP request and adapter data values. For example, TDataSetAdapter supports updates to multiple detail records by using hidden fields to identify the relationship between values in the HTTP request with dataset rows. The hidden fields defined by AddRecordKeysHiddenFields are added to an HTML form using the server-side script method HiddenRecordFields.WriteFields.

See Also

ListHiddenRecordFields (see page 5564)

54.1.44.4.4 TCustomAdapter.CanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function CanAddActionClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

CanAddActionClass is called at design-time when editing an adapter. CanAddActionClass returns the result of ImplCanAddActionClass (see page 5553).

See Also

ImplCanAddActionClass (see page 5553)

54.1.44.4.5 TCustomAdapter.CanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function CanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

CanAddFieldClass is called at design-time when editing an adapter. CanAddFieldClass returns the result of ImplCanAddFieldClass (see page 5553).

See Also

ImplCanAddFieldClass (see page 5553)

54.1.44.4.6 TCustomAdapter.CheckModifyAccess Method

Checks end-user rights to modify the adapter's field values.

Pascal

```
procedure CheckModifyAccess;
```

C++

```
__fastcall CheckModifyAccess();
```

Description

This procedure is called to verify that the current user has rights to modify this adapter's fields. An exception will be raised if access is denied.

See Also

ModifyAccess (see page 5574)

HasModifyAccess (see page 5552)

UpdateRecords (see page 5569)

54.1.44.4.7 TCustomAdapter.ClearAdapterValues Method

Clears adapter state information.

Pascal

```
procedure ClearAdapterValues;
```

C++

```
__fastcall ClearAdapterValues();
```

Description

ClearAdapterValues calls ImplClearAdapterValues (see page 5554).

See Also

ImplNotifyDeactivate (see page 5560)

ImplClearAdapterValues (see page 5554)

54.1.44.4.8 TCustomAdapter.ClearErrors Method

Clears the content of the adapter's Errors (see page 5573) property.

Pascal

```
procedure ClearErrors;
```

C++

```
__fastcall ClearErrors();
```

Description

The errors are cleared when the web module is deactivated. This ensures that the errors encountered during the processing of an HTTP request will not affect another HTTP request.

See Also

Errors (see page 5573)

ClearErrors (see page 5482)

ImplNotifyDeactivate (see page 5560)

54.1.44.4.9 TCustomAdapter.CreateActionRequestContext Method

Creates an object that describes the adapter action parameters and adapter field values.

Pascal

```
procedure CreateActionRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateActionRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

Child adapter actions call this method.

See Also

IAdapterDispatchParams (see page 987)

CreateRequestContext (see page 5452)

WebContext (see page 6003)

54.1.44.4.10 TCustomAdapter.CreateErrorsList Method

Creates the object used to manage errors associated with this adapter.

Pascal

```
function CreateErrorsList: TBaseAdapterErrorsList; virtual;
```

C++

```
virtual __fastcall TBaseAdapterErrorsList CreateErrorsList();
```

Description

CreateErrorsList creates TAdapterErrorsList (see page 5259). Descendants may override this function to create a custom errors list.

See Also

Errors (see page 5573)

TAdapterErrorsList (see page 5259)

54.1.44.4.11 TCustomAdapter.EncodeActionParams Method

Writes name/value pairs that represent adapter state.

Pascal

```
procedure EncodeActionParams(AParams: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall EncodeActionParams(IAdapterItemRequestParams AParams);
```

Description

EncodeActionParams is called by a child adapter actions when generating the name/value pairs that must be sent in the HTTP request when executing an action. For example, TDefaultFieldsPagedAdapter (see page 5747) writes the current page number as a name/value pair. ExtractRequestParams (see page 5543) and EncodeActionParams work together. ExtractRequestParams (see page 5543) reads name/value pairs that are written by EncodeActionParams.

See Also

EncodeActionParams (see page 5754)

ExtractRequestParams (see page 5543)

54.1.44.4.12 TCustomAdapter.EndIterator Method

Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. EndIterator calls ImplEndIterator (see page 5554).

See Also

EndIterator (see page 1056)

ImplEndIterator (see page 5554)

54.1.44.4.13 TCustomAdapter.ExtractRequestParams Method

Reads name/value pairs from the HTTP request.

Pascal

```
procedure ExtractRequestParams(ARequest: IUnknown); virtual;
```

C++

```
virtual __fastcall ExtractRequestParams(IUnknown ARequest);
```

Description

ExtractRequestParams is called by a child adapter action before executing the action. This gives the adapter an opportunity to extract values from the HTTP request that are important to the behavior of the adapter. For example, TDefaultFieldsPagedAdapter (see page 5747) extracts the current page number from the request. ExtractRequestParams and EncodeActionParams (see page 5542) work together. ExtractRequestParams reads name/value pairs that are written by EncodeActionParams (see page 5542).

See Also

ExtractRequestParams (see page 5754)

EncodeActionParams (see page 5542)

54.1.44.4.14 TCustomAdapter.FindAction Method

Returns an action component with a matching action name.

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const UnicodeString AName);
```

Description

The action name is used when a component property references an adapter action by name. Actions (see page 5572) also have a variable name that is used to reference the action with server-side script. The variable name and action name are usually the same. The only time that they need to be different is when the action name is not a valid server-side script identifier.

See Also

FindField (see page 5543)

54.1.44.4.15 TCustomAdapter.FindField Method

Returns an adapter field component with a matching field name.

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const UnicodeString AName);
```

Description

The field name is used when a component property references an adapter field by name. Adapter fields also have a variable name that is used to reference the field with server-side script. The variable name and field name are usually the same. The only time that they need to be different is when the field name is not a valid server-side script identifier.

54.1.44.4.16 TCustomAdapter.FindNamedAction Method

Finds an action with a matching variable name.

Pascal

```
function FindNamedAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindNamedAction(const UnicodeString AName);
```

Description

The AName parameter contains the name used when referencing an action with server-side script. This name may not be the same as the component name. For example, server-side script accesses actions in a TDataSetAdapter with using the command name (e.g.; DeleteRow) rather than the adapter action component name.

See Also

FindNamedVariable (see page 5544)

54.1.44.4.17 TCustomAdapter.FindNamedField Method

Finds an adapter field with a matching variable name.

Pascal

```
function FindNamedField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindNamedField(const UnicodeString AName);
```

Description

The AName parameter contains the name used when referencing a field with server-side script. This name may not be the same as the component name. For example, server-side script accesses fields in a TDataSetAdapter with using the dataset field name rather than the adapter field component name.

See Also

FindNamedVariable (see page 5544)

54.1.44.4.18 TCustomAdapter.FindNamedVariable Method

Finds a field or action with a matching variable name.

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const UnicodeString AName);
```

Description

FindNamedVariable calls FindNamedField (see page 5544) and FindNamedAction (see page 5544). The AName parameter contains the name used when referencing an action or field with server-side script. This name may not be the same as the component name. For example, server-side script references fields in a TDataSetAdapter using the dataset field name rather than the adapter field component name.

See Also

FindNamedAction (see page 5544)

FindNamedField (see page 5544)

54.1.44.4.19 TCustomAdapter.FindVariable Method

Finds an adapter action or adapter field with the given variable name.

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const UnicodeString AName);
```

Description

FindVariable is called when server side script references a property of an adapter object. FindVariable calls ImplFindVariable (see page 5554).

See Also

ImplFindVariable (see page 5554)

54.1.44.4.20 TCustomAdapter.FindVariableContainer Method

Finds an adapter action or adapter field that is a container for other scriptable objects.

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const UnicodeString AName);
```

Description

FindVariableContainer is called when server side script references a property of a property. FindVariableContainer returns the result of ImplFindVariableContainer (see page 5555).

See Also

FindVariable (see page 5545)

ImplFindVariableContainer (see page 5555)

54.1.44.4.21 TCustomAdapter.GetActionsClass Method

Specifies the class to create as a container for adapter actions components.

Pascal

```
function GetActionsClass: TAdapterActionsClass; virtual;
```

C++

```
virtual __fastcall TAdapterActionsClass GetActionsClass();
```

Description

Descendants override this method to specify a different adapter action container.

See Also

Actions (see page 5572)

54.1.44.4.22 TCustomAdapter.GetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure GetActionsList(AList: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings AList);
```

Description

GetActionsList is called at design time to get a list containing names and objects. Each name identifies an action name to insert and each object identifies a class of adapter action component. When editing the adapter, these action names are displayed in the dialog box that is used to add new actions. For example, when editing a TDataSetAdapter a list of all of the dataset commands are displayed such as NextRow, PrevRow, DeleteRow, etc.

See Also

ImplGetActionsList (see page 5555)

54.1.44.4.23 TCustomAdapter.GetAdapterActions Method

Returns the adapter actions container object.

Pascal

```
function GetAdapterActions: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterActions();
```

Description

Returns the adapter actions container object.

See Also

Actions (see page 5572)

54.1.44.4.24 TCustomAdapter.GetAdapterErrors Method

Returns the adapter's errors object.

Pascal

```
function GetAdapterErrors: TObject;
```

C++

```
__fastcall TObject * GetAdapterErrors();
```

Description

Returns the adapter's errors object.

See Also

Errors (see page 5573)

54.1.44.4.25 TCustomAdapter.GetAdapterErrorsList Method

<insert text here>

Pascal

```
function GetAdapterErrorsList: TBaseAdapterErrorsList;
```

C++

```
__fastcall TBaseAdapterErrorsList GetAdapterErrorsList();
```

Description

Returns an object used to track errors.

See Also

TBaseAdapterErrorsList (see page 5479)

Errors (see page 5573)

54.1.44.4.26 TCustomAdapter.GetAdapterFields Method

Returns the adapter fields container object.

Pascal

```
function GetAdapterFields: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterFields();
```

Description

Returns the adapter fields container object.

See Also

Data (see page 5572)

54.1.44.4.27 TCustomAdapter.GetAdapterHiddenFields Method

Returns a list of hidden fields that hold adapter state information.

Pascal

```
function GetAdapterHiddenFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenFields();
```

Description

GetAdapterHiddenFields populates a list by calling ListHiddenFields (see page 5563).

See Also

ListHiddenFields (see page 5563)

54.1.44.4.28 TCustomAdapter.GetAdapterHiddenRecordFields Method

Returns a list of hidden fields that associate HTML form fields and adapter field.

Pascal

```
function GetAdapterHiddenRecordFields: TObject;
```

C++

```
__fastcall TObject * GetAdapterHiddenRecordFields();
```

Description

GetAdapterHiddenRecordsFields populates a list by calling ListHiddenRecordFields (see page 5564). If the list is empty, nil (Delphi) or NULL (C++) is returned. Otherwise, the list is returned.

See Also

ListHiddenRecordFields (see page 5564)

54.1.44.4.29 TCustomAdapter.GetChildren Method

Provides the interface for a method that adapters use to return their child components.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TCustomAdapter (see page 5531) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It calls the Proc procedure passing the Actions (see page 5572) and Data (see page 5572) properties. This allows the adapter actions and adapter fields to be loaded and saved with the TCustomAdapter (see page 5531) object.

See Also

Actions (see page 5572)

Data (see page 5572)

54.1.44.4.30 TCustomAdapter.GetDataClass Method

Specifies the class to create as a container for adapter fields components.

Pascal

```
function GetDataClass: TAdapterFieldsClass; virtual;
```

C++

```
virtual __fastcall TAdapterFieldsClass GetDataClass();
```

Description

Descendants override this method to specify a different adapter fields container.

See Also

TAdapterField (see page 5267)

Data (see page 5572)

54.1.44.4.31 TCustomAdapter.GetEchoActionFieldValues Method

Returns the value of the echo flag.

Pascal

```
function GetEchoActionFieldValues: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValues();
```

Description

EchoActionFieldValues (see page 5572) calls ImplEchoActionFieldValues.

See Also

EchoActionFieldValues (see page 5572)

ImplGetEchoActionFieldValues (see page 5555)

54.1.44.4.32 TCustomAdapter.GetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure GetFieldsList(AList: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings AList);
```

Description

GetFieldsList is called at design time to get a list containing names and objects. Each name identifies a field name to insert and each object identifies a class of adapter field. When editing the adapter, these field names are displayed in the dialog box that is used to add new adapter fields. For example, when editing a TDataSetAdapter a list of all of the dataset's fields are displayed. GetFieldsList calls ImplGetFieldsList (see page 5556).

See Also

ImplGetFieldsList (see page 5556)

54.1.44.4.33 TCustomAdapter.GetHiddenFieldOptions Method

Indicates the types of hidden fields that this adapter requires in a submitted HTML form.

Pascal

```
function GetHiddenFieldOptions: TAdapterHiddenFieldOptions; virtual;
```

C++

```
virtual __fastcall TAdapterHiddenFieldOptions GetHiddenFieldOptions();
```

Description

An adapter may use hidden fields to pass state information down to the client with the HTTP response so that the state

information is available when processing a resulting HTTP response. Descendant classes may override this method.

See Also

ListHiddenFields (see page 5563)

ListHiddenRecordFields (see page 5564)

54.1.44.4.34 TCustomAdapter.GetIteratorIndex Method

Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.

Pascal

```
function GetIteratorIndex: Integer;
```

C++

```
__fastcall int GetIteratorIndex();
```

Description

Adapters may support multiple data records. For example, TDataSetAdapter provides access to the records in a dataset. GetIteratorIndex calls ImplGetIteratorIndex (see page 5556).

See Also

GetIteratorIndex (see page 1053)

ImplGetIteratorIndex (see page 5556)

54.1.44.4.35 TCustomAdapter.GetVariable Method

Returns an action or field.

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

GetVariableCount (see page 5550) and GetVariable is used to iterate over actions and fields in an adapter. The I parameter is a zero-based index that must be less than the result of GetVariableCount (see page 5550). GetVariable calls ImplGetVariable (see page 5556).

See Also

ImplGetVariable (see page 5556)

GetVariableCount (see page 5550)

54.1.44.4.36 TCustomAdapter.GetVariableCount Method

Returns the total count of actions and fields.

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

GetVariableCount and GetVariable (see page 5550) is used to iterate over the actions and fields in an adapter. GetVariableCount calls ImplGetVariableCount (see page 5557).

See Also

GetVariable (see page 5550)

ImplGetVariableCount (see page 5557)

54.1.44.4.37 TCustomAdapter.GetVariableName Method

Gets the name used to reference this adapter with server-side script.

Pascal

```
function GetVariableName: string;
```

C++

```
__fastcall UnicodeString GetVariableName();
```

Description

GetVariableName is called to retrieve the name used to access this adapter with server-side script. GetVariableName calls ImplGetVariableName (see page 5557).

See Also

ImplGetVariableName (see page 5557)

54.1.44.4.38 TCustomAdapter.GetVisibleActions Method

Returns a list of this adapter's action components.

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

Some adapters may support default actions. For example, TLoginFormAdapter (see page 5824) has a has a default action for submitting the login form. If no actions have been explicitly added, GetVisibleActions returns a list containing all of the default actions. GetVisibleActions calls ImplGetVisibleActions (see page 5557).

See Also

TWebComponentList (see page 6025)

54.1.44.4.39 TCustomAdapter.GetVisibleFields Method

Returns a list of this adapter's field components.

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

Some adapter may support default fields. For example, TDataSetAdapter has default fields for each field in the dataset. If no adapter fields have been explicitly added, GetVisibleFields returns a list containing all default fields. GetVisibleFields calls ImplGetVisibleFields (see page 5558).

See Also

TWebComponentList (see page 6025)

54.1.44.4.40 TCustomAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

54.1.44.4.41 TCustomAdapter.HasExecuteAccess Method

Checks the end-user's right to execute this adapter's actions.

Pascal

```
function HasExecuteAccess: Boolean;
```

C++

```
__fastcall Boolean HasExecuteAccess();
```

Description

HasExecuteAccess calls ImplHasExecuteAccess (see page 5558).

See Also

ImplHasExecuteAccess (see page 5558)

54.1.44.4.42 TCustomAdapter.HasModifyAccess Method

Checks the end-user's right to modify this adapter's fields.

Pascal

```
function HasModifyAccess: Boolean;
```

C++

```
__fastcall Boolean HasModifyAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanModify property. HasModifyAccess calls ImplHasModifyAccess (see page 5558).

See Also

ImplHasModifyAccess (see page 5558)

ModifyAccess (see page 5574)

54.1.44.4.43 TCustomAdapter.HasViewAccess Method

Checks the end-user's right to view this adapter's fields.

Pascal

```
function HasViewAccess: Boolean;
```

C++

```
__fastcall Boolean HasViewAccess();
```

Description

The value returned by this function can be accessed with server-side script using the CanView property. HasViewAccess calls ImplHasViewAccess (see page 5559).

See Also

ViewAccess (see page 5574)

54.1.44.4.44 TCustomAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass returns false. This means that only TAdapterAction (see page 5201) can be added. Descendants override this method to allow additional field types.

See Also

CanAddActionClass (see page 5539)

ImplCanAddClass (see page 5212)

54.1.44.4.45 TCustomAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass returns false. This means that only TAdapterField (see page 5267) can be added. Descendants override this method to allow additional field types.

See Also

CanAddFieldClass (see page 5540)

ImplCanAddClass (see page 5279)

54.1.44.4.46 TCustomAdapter.ImplClearAdapterValues Method

Clears adapter state information.

Pascal

```
procedure ImplClearAdapterValues; virtual;
```

C++

```
virtual __fastcall ImplClearAdapterValues();
```

Description

ClearAdapterValues (see page 5540) clears the echo flag.

See Also

ImplSetEchoActionFieldValues (see page 5561)

ClearAdapterValues (see page 5540)

54.1.44.4.47 TCustomAdapter.ImplEndIterator Method

Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); virtual;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplEndIterator sets the ItemIndex to 0 and the iterator flag to false.

See Also

EndIterator (see page 1056)

EndIterator (see page 5542)

54.1.44.4.48 TCustomAdapter.ImplFindVariable Method

Finds an adapter action or adapter field child with a matching variable name.

Pascal

```
function ImplFindVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplFindVariable(const UnicodeString AName);
```

Description

ImpFindVariable returns the result of FindNamedVariable (see page 5544).

See Also

FindVariable (see page 5545)

54.1.44.4.49 TCustomAdapter.ImplFindVariableContainer Method

Finds an adapter action or adapter field that is a container for other scriptable objects.

Pascal

```
function ImplFindVariableContainer(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplFindVariableContainer(const UnicodeString AName);
```

Description

FindVariableContainer (see page 5545) is called when server side script references a property of a property. ImplFindVariableContainer tries to find an adapter field or adapter action that is a container for other objects.

See Also

FindVariableContainer (see page 5545)

54.1.44.4.50 TCustomAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); virtual;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList is called by GetActionsList (see page 5546). ImplGetActionsList provides an empty list. ImplGetActionsList may be overridden by descendant classes.

See Also

GetActionsList (see page 5546)

54.1.44.4.51 TCustomAdapter.ImplGetEchoActionFieldValues Method

Returns the value of the echo flag.

Pascal

```
function ImplGetEchoActionFieldValues: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetEchoActionFieldValues();
```

Description

ImplGetEchoActionFieldValues returns the value passed to ImplSetEchoActionFieldValues (see page 5561). If ImplSetEchoActionFieldValues (see page 5561) has not been processed during the current HTTP request then False is

returned.

See Also

EchoActionFieldValues (see page 5572)

GetEchoActionFieldValues (see page 5549)

54.1.44.4.52 TCustomAdapter.ImplGetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); virtual;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList is called by GetFieldsList (see page 5549). ImplGetFieldsList provides an empty list. ImplGetFieldsList may be overridden by descendant classes.

See Also

GetFieldsList (see page 5549)

54.1.44.4.53 TCustomAdapter.ImplGetIteratorIndex Method

Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.

Pascal

```
function ImplGetIteratorIndex: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetIteratorIndex();
```

Description

ImplGetIteratorIndex returns the value ItemIndex. ItemIndex is modified by ImplStartIterator (see page 5562), ImplNextIteration (see page 5560), and ImplEndIterator (see page 5554).

See Also

GetIteratorIndex (see page 1053)

GetIteratorIndex (see page 5550)

ImplStartIterator (see page 5562)

ImplNextIteration (see page 5560)

ImplEndIterator (see page 5554)

54.1.44.4.54 TCustomAdapter.ImplGetVariable Method

Returns an action or field.

Pascal

```
function ImplGetVariable(I: Integer): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * ImplGetVariable(int I);
```

Description

ImplGetVariable is called by GetVariable (see page 5550). ImplGetVariable uses the I parameter to locate an adapter action or adapter field.

See Also

GetVariable (see page 5550)

54.1.44.4.55 TCustomAdapter.ImplGetVariableCount Method

Returns the total count of actions and fields.

Pascal

```
function ImplGetVariableCount: Integer; virtual;
```

C++

```
virtual __fastcall int ImplGetVariableCount();
```

Description

ImplGetVariableCount is called by GetVariableCount (see page 5550).

See Also

GetVariableCount (see page 5550)

54.1.44.4.56 TCustomAdapter.ImplGetVariableName Method

Gets the name used to reference this adapter with server-side script.

Pascal

```
function ImplGetVariableName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetVariableName();
```

Description

ImplGetVariableName returns the component name.

See Also

GetVariableName (see page 5551)

54.1.44.4.57 TCustomAdapter.ImplGetVisibleActions Method

Returns a list of this adapter's action components.

Pascal

```
function ImplGetVisibleActions: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleActions();
```

Description

ImplGetVisibleActions is called by GetVisibleActions (see page 5551). ImplGetVisibleActions may be overriden by descendant classes.

See Also

GetVisibleActions (see page 5551)

54.1.44.4.58 TCustomAdapter.ImplGetVisibleFields Method

Returns a list of this adapter's field components.

Pascal

```
function ImplGetVisibleFields: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleFields();
```

Description

ImplGetVisibleFields is called by GetVisibleFields (see page 5551). ImplGetVisibleFields may be overridden by descendant classes.

See Also

GetVisibleFields (see page 5551)

54.1.44.4.59 TCustomAdapter.ImplHasExecuteAccess Method

Checks the end-user's right to execute this adapter's actions.

Pascal

```
function ImplHasExecuteAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasExecuteAccess();
```

Description

This function is called by HasExecuteAccess (see page 5552) to check the end-user's right to execute the adapter's actions. If the ExecuteAccess (see page 5573) property is not blank, this function calls the EndUserAdapter's HasExecuteRights function, passing the ExecuteAccess (see page 5573) property value.

See Also

HasExecuteAccess (see page 5552)

ExecuteAccess (see page 5573)

HasExecuteAccess (see page 5463)

54.1.44.4.60 TCustomAdapter.ImplHasModifyAccess Method

Checks the end-user's right to modify this adapter's fields.

Pascal

```
function ImplHasModifyAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasModifyAccess();
```

Description

This function is called by HasModifyAccess (see page 5552) to check the end-user's right to modify the adapter field. If the ModifyAccess (see page 5574) property is not blank, this function calls the EndUserAdapter HasModifyRights function, passing the ModifyAccess (see page 5574) property value.

See Also

HasModifyAccess (see page 5552)

ModifyAccess (see page 5574)

54.1.44.4.61 TCustomAdapter.ImplHasViewAccess Method

Checks the end-user's right to view this adapter's fields.

Pascal

```
function ImplHasViewAccess: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHasViewAccess();
```

Description

This function is called by HasViewAccess (see page 5553) to check the end-user's right to view the adapter's fields. If the ViewAccess (see page 5574) property is not blank, this function calls the EndUserAdapter's HasViewRights function, passing the ViewAccess (see page 5574) property value.

See Also

HasViewAccess (see page 5553)

ViewAccess (see page 5574)

54.1.44.4.62 TCustomAdapter.ImplInIterator Method

Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.

Pascal

```
function ImplInIterator: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplInIterator();
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplGetInIterator returns the value of the iteration flag. The iteration flag is set to true by TCustomAdapter (see page 5531)'s ImplStartIterator (see page 5562) method.

See Also

InIterator (see page 1053)

InIterator (see page 5562)

ImplStartIterator (see page 5562)

54.1.44.4.63 TCustomAdapter.ImplNextIteration Method

Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplNextIteration increments ItemIndex and returns true if the iteration index is less than ItemCount.

See Also

NextIteration (see page 1056)

NextIteration (see page 5564)

ImplStartIterator (see page 5562)

ImplEndIterator (see page 5554)

54.1.44.4.64 TCustomAdapter.ImplNotifyActivate Method

Called when the web module is activated.

Pascal

```
procedure ImplNotifyActivate; virtual;
```

C++

```
virtual __fastcall ImplNotifyActivate();
```

Description

A web module is activated when an HTTP request is received. ImplNotifyActivate is a placeholder. Descendant classes override this method to initialize the adapter before an HTTP request processed.

See Also

NotifyActivate (see page 5565)

54.1.44.4.65 TCustomAdapter.ImplNotifyDeactivate Method

Called when the web module is deactivated.

Pascal

```
procedure ImplNotifyDeactivate; virtual;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

A web module is deactivated after the HTTP request has been handled.. ImplNotifyDeactivate discards adapter state information including adapter errors, iteration flag, and iteration index.

See Also

NotifyDeactivate (see page 5566)

54.1.44.4.66 TCustomAdapter.ImplSetActions Method

Initializes the Actions (see page 5572) property.

Pascal

```
procedure ImplSetActions(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall ImplSetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 5572) property is the container for adapter action components. Descendants may override this method.

See Also

Actions (see page 5572)

SetActions (see page 5566)

54.1.44.4.67 TCustomAdapter.ImplSetData Method

Initializes the Data (see page 5572) property.

Pascal

```
procedure ImplSetData(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall ImplSetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 5572) property is the container for adapter field components. Descendants may override this method.

See Also

Data (see page 5572)

SetData (see page 5566)

54.1.44.4.68 TCustomAdapter.ImplSetEchoActionFieldValues Method

Sets the echo flag for this adapter's fields.

Pascal

```
procedure ImplSetEchoActionFieldValues(AValue: Boolean); virtual;
```

C++

```
virtual __fastcall ImplSetEchoActionFieldValues(Boolean AValue);
```

Description

ImplEchoActionFieldValues modified the EchoActionFieldValue property of the adapter's fields.

See Also

GetEchoActionFieldValues (see page 5549)

SetEchoActionFieldValues (see page 5567)

EchoActionFieldValue (see page 5368)

54.1.44.4.69 TCustomAdapter.ImplStartIterator Method

Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. ImplStartIterator sets ItemIndex to 0 and sets the iteration flag to true. true is returned.

See Also

StartIterator (see page 1056)

StartIterator (see page 5567)

ImplNextIteration (see page 5560)

ImplEndIterator (see page 5554)

54.1.44.4.70 TCustomAdapter.InIterator Method

Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.

Pascal

```
function InIterator: Boolean;
```

C++

```
__fastcall Boolean InIterator();
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. InIterator calls ImplInIterator (see page 5559).

See Also

InIterator (see page 1053)

ImplInIterator (see page 5559)

54.1.44.4.71 TCustomAdapter.IsActionInUse Method

Indicates whether a particular action has been added to the adapter.

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const UnicodeString AName);
```

Description

IsActionInUse is called at design time to determine whether an action with a particular name has been added to the adapter. When editing the adapter, the names of actions in use are not displayed in the dialog box used to add new actions.

See Also

IsFieldInUse (see page 5563)

GetFieldsList (see page 5549)

54.1.44.4.72 TCustomAdapter.IsFieldInUse Method

Indicates whether a particular field has been added to the adapter.

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const UnicodeString AName);
```

Description

IsFieldInUse is called at design time to determine whether a field with a particular name has been added to the adapter. When editing the adapter, the names of these fields if use are not displayed in the dialog box used to add new fields.

See Also

IsActionInUse (see page 5563)

GetFieldsList (see page 5549)

54.1.44.4.73 TCustomAdapter.ListHiddenFields Method

Specifies the HTML form hidden fields needed by the adapter.

Pascal

```
procedure ListHiddenFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall ListHiddenFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Some adapters have state information that must be preserved between HTTP requests. For example, TDataSetAdapter has a mode property. HTML forms that modify TDataSetAdapter datasets include the mode property value as a hidden field. When the HTML form is submitted, the mode property value will be included in the HTTP request. When executing an action TDataSetAdapter extracts the mode property value from the HTTP request and updates it's mode property. ListHiddenRecordFields (see page 5564) may call AddAdapterModeHiddenField (see page 5538) depending on the flags

returned by GetHiddenFieldOptions (see page 5549).

See Also

GetHiddenFieldOptions (see page 5549)

ListHiddenRecordFields (see page 5564)

AddAdapterModeHiddenField (see page 5538)

GetAdapterHiddenFields (see page 5547)

54.1.44.4.74 TCustomAdapter.ListHiddenRecordFields Method

Specifies the HTML form hidden fields needed by the adapter.

Pascal

```
procedure ListHiddenRecordFields(AHiddenFields: TAdapterHiddenFieldsList);
```

C++

```
__fastcall ListHiddenRecordFields(TAdapterHiddenFieldsList AHiddenFields);
```

Description

Some adapters may use hidden fields to define the relationship between input fields in the HTML form and data of the adapter. For example, an HTML form that contains an HTML table for updating 5 detail records of a TDataSetAdapter will have 5 sets of hidden fields. Each set provides state information needed by the adapter to determine the relationship between the input fields in the HTTP request and the dataset.

ListHiddenRecordFields may call AddOrigValueHiddenFields (see page 5539) and/or AddRecordKeysHiddenFields (see page 5539) depending on the flags returned by GetHiddenFieldOptions (see page 5549).

See Also

GetHiddenFieldOptions (see page 5549)

TAdapterHiddenFieldsList (see page 5301)

AddOrigValueHiddenFields (see page 5539)

AddRecordKeysHiddenFields (see page 5539)

GetAdapterHiddenRecordFields (see page 5548)

54.1.44.4.75 TCustomAdapter.NextIteration Method

Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. NextIteration calls ImplNextIteration (see page 5560).

See Also

NextIteration (see page 1056)

ImplNextIteration (see page 5560)

54.1.44.4.76 TCustomAdapter.Notification Method

Clears references to deleted components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called when a component is inserted or deleted.

54.1.44.4.77 TCustomAdapter.NotifyActivate Method

Called when the web module is activated.

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

A web module is activated when an HTTP request is received. NotifyActivate calls ImplNotifyActivate (see page 5560).

See Also

ImplNotifyActivate (see page 5560)

NotifyDeactivate (see page 5566)

54.1.44.4.78 TCustomAdapter.NotifyAdapterChange Method

Notifies components that the adapter has changed in some way.

Pascal

```
procedure NotifyAdapterChange;
```

C++

```
__fastcall NotifyAdapterChange();
```

Description

NotifyAdapterChange calls components that have registered a notification callback to let them know that the adapter has changed. For example, TAdapterFieldGroup needs to know when an adapter field is added to the adapter component that the TAdapterFieldGroup references.

See Also

NotifyList (see page 5574)

54.1.44.4.79 TCustomAdapter.NotifyDeactivate Method

Called when the web module is deactivated.

Pascal

```
procedure NotifyDeactivate;
```

C++

```
__fastcall NotifyDeactivate();
```

Description

A web module is deactivated after the HTTP request has been handled.. NotifyDeactivate calls ImplNotifyDeactivate (see page 5560).

See Also

ImplNotifyDeactivate (see page 5560)

NotifyActivate (see page 5565)

54.1.44.4.80 TCustomAdapter.SetActions Method

Initializes the Actions (see page 5572) property.

Pascal

```
procedure SetActions(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 5572) property is the container for adapter action components. SetActions calls ImplSetActions (see page 5561).

See Also

Actions (see page 5572)

ImplSetActions (see page 5561)

54.1.44.4.81 TCustomAdapter.SetData Method

Initializes the Data (see page 5572) property.

Pascal

```
procedure SetData(const Value: TAdapterSubComponent); virtual;
```

C++

```
virtual __fastcall SetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 5572) property is the container for adapter field components. SetData calls ImplSetData (see page 5561).

See Also

Data (see page 5572)

ImplSetData (see page 5561)

54.1.44.4.82 TCustomAdapter.SetEchoActionFieldValues Method

Sets the echo flag for this adapter's fields.

Pascal

```
procedure SetEchoActionFieldValues(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValues(Boolean AValue);
```

Description

SetEchoActionFieldValues calls ImplSetActionFieldValues.

See Also

EchoActionFieldValues (see page 5572)

ImplSetEchoActionFieldValues (see page 5561)

54.1.44.4.83 TCustomAdapter.StartIterator Method

Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

Adapters may support multiple records. For example, TDataSetAdapter provides access to the records in a dataset. StartIterator calls ImplStartIterator (see page 5562).

See Also

StartIterator (see page 1056)

ImplStartIterator (see page 5562)

54.1.44.4.84 TCustomAdapter.UpdateFieldsExecuteValues Method

Executes the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsExecuteValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsExecuteValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsExecuteValues loops through the adapter fields passed in the AAdapterFields parameter, calling the ExecuteValue function. Adapter fields raise an exception to indicate an error. If an exception occurs, the exception message is added to the Errors (see page 5573) object and execution continues.

When a field is executable, the field value passed in the HTTP request is used to execute a command rather than update a value. For example, an HTML form may contain a table with a delete row checkbox on each row. When the form is submitted, the values in the HTTP request indicate whether to delete one or more rows. Executable adapter fields must implement the IExecuteValue (see page 5883) interface.

See Also

Errors (see page 5573)

IActionRequest

IExecuteValue (see page 5883)

54.1.44.4.85 TCustomAdapter.UpdateFieldsGetAnyChanges Method

Determines whether the values in the HTTP request change any adapter fields.

Pascal

```
function UpdateFieldsGetAnyChanges(AActionRequest: IActionRequest; AAdapterFields:
TObjectList): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateFieldsGetAnyChanges(IActionRequest AActionRequest,
TObjectList AAdapterFields);
```

Description

TAdapterUpdateField (see page 5419)'s CheckValueChange method loops through the adapter fields passed in the AAdapterFields parameter, calling the CheckValueChange function. UpdateFieldsGetAnyChanges will return true if any adapter field returns true when its CheckValueChange method is called.

See Also

Errors (see page 5573)

ValidateValue (see page 5431)

IActionRequest

CheckValueChange (see page 5424)

54.1.44.4.86 TCustomAdapter.UpdateFieldsUpdateValues Method

Updates the adapter's fields with values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsUpdateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsUpdateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsUpdateValues loops through the adapter fields passed in the AAdapterFields parameter, calling the UpdateValue function. Adapter fields raise an exception to indicate a update error. If an exception occurs, the exception message is added to the Errors (see page 5573) object and updating continues. OnBeforeUpdateFields (see page 5578) and OnAfterUpdateFields (see page 5576) are at the beginning and end of this method.

See Also

OnBeforeUpdateFields (see page 5578)

OnAfterUpdateFields (see page 5576)

Errors (see page 5573)

UpdateValue (see page 5431)

IActionRequest

54.1.44.4.87 TCustomAdapter.UpdateFieldsValidateValues Method

Validate the adapter's field values passed in the HTTP request.

Pascal

```
procedure UpdateFieldsValidateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateFieldsValidateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsValidateValues loops through the adapter fields passed in the AAdapterFields parameter, calling the ValidateValue function. Adapter fields raise an exception to indicate a validation error. When an exception occurs, th exception messages are added to the Errors (see page 5573) object and validation continues. OnBeforeValidateFields (see page 5578) and OnAfterValidateFields (see page 5577) are at the beginning and end of this method.

See Also

OnBeforeValidateFields (see page 5578)

OnAfterValidateFields (see page 5577)

Errors (see page 5573)

ValidateValue (see page 5431)

IActionRequest

54.1.44.4.88 TCustomAdapter.UpdateRecords Method

Updates the adapter's fields with the field values passed in an HTTP request.

Pascal

```
function UpdateRecords(AActionRequest: IActionRequest = nil): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateRecords(IActionRequest AActionRequest = nil);
```

Description

Call UpdateRecords to update the adapter's fields with the field values passed in an HTTP request. Typically, UpdateRecords will be called by an adapter action that submits an HTML form. UpdatesRecords first verifies that the end-user has rights to modify the adapter by calling CheckModifyAccess (see page 5540). CheckModifyAccess (see page 5540) will raise an exception if access is denied. Next, UpdateRecordsGetAnyChanges (see page 5570) is called to determine whether the values in the HTTP request represent changes to the adapter fields. If there are changes then UpdateRecordsValidateValues (

see page 5571) is called followed by UpdateRecordsUpdateValues (see page 5570). Exceptions raised during validation or update are added to the adapter's errors object. UpdateRecords returns true if the errors object contains no errors. The AActionRequest parameter is optional.

See Also

IActionRequest

CheckModifyAccess (see page 5540)

UpdateRecordsGetAnyChanges (see page 5570)

UpdateRecordsValidateValues (see page 5571)

UpdateRecordsUpdateValues (see page 5570)

Errors (see page 5573)

54.1.44.4.89 TCustomAdapter.UpdateRecordsGetAnyChanges Method

Determines whether values passed in the HTTP request will change any adapter fields.

Pascal

```
function UpdateRecordsGetAnyChanges(AActionRequest: IActionRequest; AAdapterFields:
TObjectList): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean UpdateRecordsGetAnyChanges(IActionRequest AActionRequest,
TObjectList AAdapterFields);
```

Description

The HTTP request may contain values that are associated with multiple adapter records. For example, an HTML form that contains a grid with input fields on each row will generate an HTTP request to update multiple records. UpdateRecordsGetAnyChanges iterates all the records in the HTTP request, calling UpdateFieldsGetAnyChanges (see page 5568) for each record.

See Also

UpdateFieldsGetAnyChanges (see page 5568)

54.1.44.4.90 TCustomAdapter.UpdateRecordsUpdateValues Method

Updates adapter fields with values passed in the HTTP request.

Pascal

```
procedure UpdateRecordsUpdateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateRecordsUpdateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

The HTTP request may contain values that are associated with multiple adapter records. For example, an HTML form that contains a grid with input fields on each row will generate an HTTP request to update multiple records. UpdateRecordsUpdateValues iterates all the records in the HTTP request, calling UpdateFieldUpdateValues for each record.

See Also

UpdateFieldsUpdateValues (see page 5568)

54.1.44.4.91 TCustomAdapter.UpdateRecordsValidateValues Method

Validate the adapter's field values passed in the HTTP request.

Pascal

```
procedure UpdateRecordsValidateValues(AActionRequest: IActionRequest; AAdapterFields:
TObjectList); virtual;
```

C++

```
virtual __fastcall UpdateRecordsValidateValues(IActionRequest AActionRequest, TObjectList
AAdapterFields);
```

Description

UpdateFieldsValidateValues (see page 5569) loops through the adapter fields passed in the AAdapterFields parameter, calling the ValidateValue function. Adapter fields raise an exception to indicate a validation error. When an exception occurs, the exception message is added to the Errors (see page 5573) object and validation continues. OnBeforeValidateFields (see page 5578) and OnAfterValidateFields (see page 5577) are at the beginning and end of this method.

See Also

OnBeforeValidateFields (see page 5578)

OnAfterValidateFields (see page 5577)

Errors (see page 5573)

ValidateValue (see page 5431)

IActionRequest

54.1.44.4.92 TCustomAdapter.WebComponentsChange Method

WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Pascal

```
procedure WebComponentsChange(Sender: TObject);
```

C++

```
__fastcall WebComponentsChange(TObject * Sender);
```

Description

WebComponentsChange will be called, for example, when an adapter field is added to the adapter or when an adapter field is removed from the adapter. WebComponentsChange calls NotifyAdapterChange (see page 5565).

See Also

NotifyAdapterChange (see page 5565)

54.1.44.5 TCustomAdapter Properties

The properties of the TCustomAdapter class are listed here.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.

54.1.44.5.1 TCustomAdapter.Actions Property

Actions is the container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterActions (see page 5209)

54.1.44.5.2 TCustomAdapter.Data Property

Data is the container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterFields (see page 5276)

54.1.44.5.3 TCustomAdapter.EchoActionFieldValues Property

Modifies the adapter's echo flag.

Pascal

```
property EchoActionFieldValues: Boolean;
```

C++

```
__property Boolean EchoActionFieldValues;
```

Description

The echo flag is used to alter the value returned by the adapter's fields. When this flag is set, the adapter's fields retrieves their value from the HTTP request. When this flag is clear, the adapter's fields retrieve their current value. Typically, this flag is set after an error occurs while executing an adapter action. Setting this flag results in generated HTML that displays values entered by the end-user. This allows the end-user to make corrections and resubmit the form.

See Also

SetEchoActionFieldValues (see page 5567)

GetEchoActionFieldValues (see page 5549)

54.1.44.5.4 TCustomAdapter.Errors Property

Returns an object used to track errors on behalf of this adapter.

Pascal

```
property Errors: TBaseAdapterErrorsList;
```

C++

```
__property TBaseAdapterErrorsList Errors;
```

Description

Adapter actions typically catch exceptions raised during execution and add the exception messages to the parent adapter's errors object. Server-side script may enumerate the errors recorded by the Errors property in order to write error messages to an HTML page.

See Also

GetAdapterErrorsList (see page 5547)

TBaseAdapterErrorsList (see page 5479)

TAdapterErrorsList (see page 5259)

54.1.44.5.5 TCustomAdapter.ExecuteAccess Property

Defines the rights required to execute this adapter's action.

Pascal

```
property ExecuteAccess: string;
```

C++

```
__property UnicodeString ExecuteAccess;
```

Description

The ExecuteAccess property defines the access rights required to execute this adapter's actions. Typically, an exception is raised when an end-user attempts to execute an adapter action that does not allow access.

See Also

TWebUserList (see page 6796)

HasExecuteAccess (see page 5552)

54.1.44.5.6 TCustomAdapter.ModifyAccess Property

Identifies the rights required to update this adapter's fields.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this fields of this adapter. When server-side script accesses the CanModify property, this ModifyAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

ImplHasModifyAccess (see page 5558)

54.1.44.5.7 TCustomAdapter.NotifyList Property

Keeps track of components that are to be notified when the adapter is changed.

Pascal

```
property NotifyList: TNotifyList;
```

C++

```
__property TNotifyList NotifyList;
```

Description

Keeps track of components that are to be notified when the adapter is changed.

See Also

NotifyAdapterChange (see page 5565)

TNotifyList (see page 968)

54.1.44.5.8 TCustomAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

ImplHasViewAccess (see page 5559)

54.1.44.5.9 TCustomAdapter.VisibleActions Property

Returns a list of this adapter's action components.

Pascal

```
property VisibleActions: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleActions;
```

Description

Some adapters support default actions. For example, TLoginFormAdapter (see page 5824) has a has a defauilt action for submitting the login form. If no actions have been explicitly added, VisibleActions returns a list containing all of the default actions.

See Also

GetVisibleActions (see page 5551)

TWebComponentList (see page 6025)

54.1.44.5.10 TCustomAdapter.VisibleFields Property

Returns a list of this adapter's field components.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

Some adapter may support default fields. If no fields have been explicitly added, VisibleFields returns a list containing all of the default fields.

See Also

GetVisibleFields (see page 5551)

TWebComponentList (see page 6025)

54.1.44.6 TCustomAdapter Events

The events of the TCustomAdapter class are listed here.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.

	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

54.1.44.6.1 TCustomAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.44.6.2 TCustomAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.44.6.3 TCustomAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 5568)

OnBeforeUpdateFields (see page 5578)

Errors (see page 5573)

54.1.44.6.4 TCustomAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 5569)

OnBeforeValidateFields (see page 5578)

Errors (see page 5573)

54.1.44.6.5 TCustomAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5478)

OnAfterExecuteAction (see page 5576)

54.1.44.6.6 TCustomAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5478)

OnAfterGetActionResponse (see page 5576)

54.1.44.6.7 TCustomAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 5568)

OnAfterUpdateFields (see page 5576)

Errors (see page 5573)

54.1.44.6.8 TCustomAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the

adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 5569)

OnAfterValidateFields (see page 5577)

Errors (see page 5573)

54.1.44.6.9 TCustomAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5478)

54.1.44.6.10 TCustomAdapter.OnIterateRecords Event

Occurs when the adapter needs to iterate over its data records.

Pascal

```
property OnIterateRecords: TIterateRecordsEvent;
```

C++

```
__property TIterateRecordsEvent OnIterateRecords;
```

Description

Implement this event when the adapter supports multiple data records. This event handler is called when server-side script iterates over the records in adapter, such as when generating an HTML table displaying all records. The event handler must update an iteration pointer or cursor that adapter fields are also aware of. Adapter fields must use this iteration pointer when called to retrieve their value.

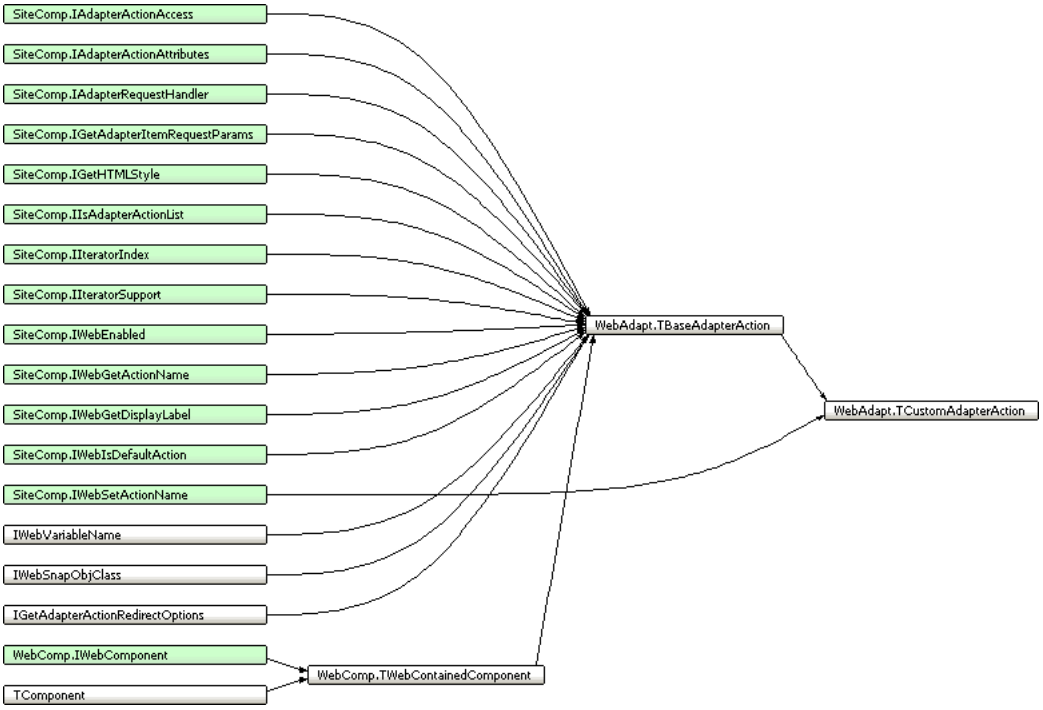
See Also

IIteratorSupport (see page 1054)

54.1.45 TCustomAdapterAction Class

TCustomAdapterAction is a base class for adapter actions.

Class Hierarchy



Pascal

```
TCustomAdapterAction = class(TBaseAdapterAction, IWebSetActionName);
```

C++

```
class TCustomAdapterAction : public TBaseAdapterAction, public IWebSetActionName;
```

File

WebAdapt

Description

TCustomAdapterAction calls event handlers to retrieve the adapter action parameters and execute the adapter action. Derive from TCustomAdapterAction to define an adapter action component that can be added to any adapter. Derive from TImplementAdapterAction to define an adapter action component that works with specific adapter component types.

See Also

TBaseAdapterAction (see page 5447)

TAdapter (see page 5190)

TAdapterAction (see page 5201)

TImplementedAdapterAction (see page 5818)

54.1.45.1 TCustomAdapterAction Members

The following tables list the members exposed by TCustomAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
V	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.

	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).

	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.45.2 TCustomAdapterAction Methods

The methods of the TCustomAdapterAction class are listed here.

Protected Methods

	Name	Description
V	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.

	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.

54.1.45.2.1 TCustomAdapterAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest create a TStrings containing the name/value pairs in AActionRequest and calls the OnExecute (see page 5588) event handler.

See Also

ImplExecuteActionRequest (see page 5464)

OnExecute (see page 5588)

54.1.45.2.2 TCustomAdapterAction.ImplGetActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function ImplGetActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetActionName();
```

Description

Gets the name used to reference this adapter action with server-side script.

See Also

ActionName (see page 5586)

ImplGetActionName (see page 5465)

54.1.45.2.3 TCustomAdapterAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

When this adapter action is executed, it provides name/value pairs which will be written when the HTML page is generated and passed back in the HTTP request. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted. When the HTML page is generated a hyperlink or button is written that identifies the name and parameters of the action. When the hyperlink or button is clicked the name of the action and parameters are passed back to the web application in the HTTP request.

See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.45.2.4 TCustomAdapterAction.ImplGetSuccessOptions Method

Indicates whether to perform an HTTP redirect after successfully executing this action.

Pascal

```
function ImplGetSuccessOptions: TAdapterActionRedirectOptions; override;
```

C++

```
virtual __fastcall TAdapterActionRedirectOptions ImplGetSuccessOptions();
```

Description

ImplGetSuccessOptions returns the value of the RedirectOptions (see page 5476) property.

See Also

RedirectOptions (see page 5476)

54.1.45.2.5 TCustomAdapterAction.ImplSetActionName Method

Sets the name that is used to reference this adapter action with server-side script.

Pascal

```
procedure ImplSetActionName(const Value: string);
```

C++

```
__fastcall ImplSetActionName(const UnicodeString Value);
```

Description

ImplSetActionName is called by SetActionName (see page 5586). ImplSetActionName stores the name in an instance variable.

See Also

SetActionName (see page 5586)

54.1.45.2.6 TCustomAdapterAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled calls the OnGetEnabled (see page 5589) event handler.

See Also

ImplWebEnabled (see page 5470)

OnGetEnabled (see page 5589)

54.1.45.2.7 TCustomAdapterAction.SetActionName Method

Sets the name that is used to reference this adapter action with server-side script.

Pascal

```
procedure SetActionName(const Value: string);
```

C++

```
__fastcall SetActionName(const UnicodeString Value);
```

Description

SetActionName is typically called at design time to set the action name of a new adapter action. SetActionName calls ImplSetActionName (see page 5585).

See Also

ImplSetActionName (see page 5585)

54.1.45.3 TCustomAdapterAction Properties

The properties of the TCustomAdapterAction class are listed here.

Public Properties

	Name	Description
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.

54.1.45.3.1 TCustomAdapterAction.ActionName Property

Identifies the name of this action when using server-side script.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

The ActionName property defines the name used to access the adapter action with server-side script.

See Also

ImplGetActionName (see page 5584)

ImplSetActionName (see page 5585)

54.1.45.4 TCustomAdapterAction Events

The events of the TCustomAdapterAction class are listed here.

Public Events

	Name	Description
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

54.1.45.4.1 TCustomAdapterAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.45.4.2 TCustomAdapterAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.45.4.3 TCustomAdapterAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.45.4.4 TCustomAdapterAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.45.4.5 TCustomAdapterAction.OnExecute Event

Called to execute this action using parameters passed in the HTTP request.

Pascal

```
property OnExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnExecute;
```

Description

Implement this event handler to execute this adapter action. Params contains the name/value pairs that were passed in the HTTP request. OnExecute is called by ImplExecuteActionRequest (see page 5584).

See Also

ImplExecuteActionRequest (see page 5584)

OnGetParams (see page 5478)

54.1.45.4.6 TCustomAdapterAction.OnGetEnabled Event

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
property OnGetEnabled: TActionGetEnabledEvent;
```

C++

```
__property TActionGetEnabledEvent OnGetEnabled;
```

Description

ImplWebEnabled (see page 5585) calls the OnGetEnabled event handler. Set the Enabled parameter to false to indicate that this adapter action should not be enabled.

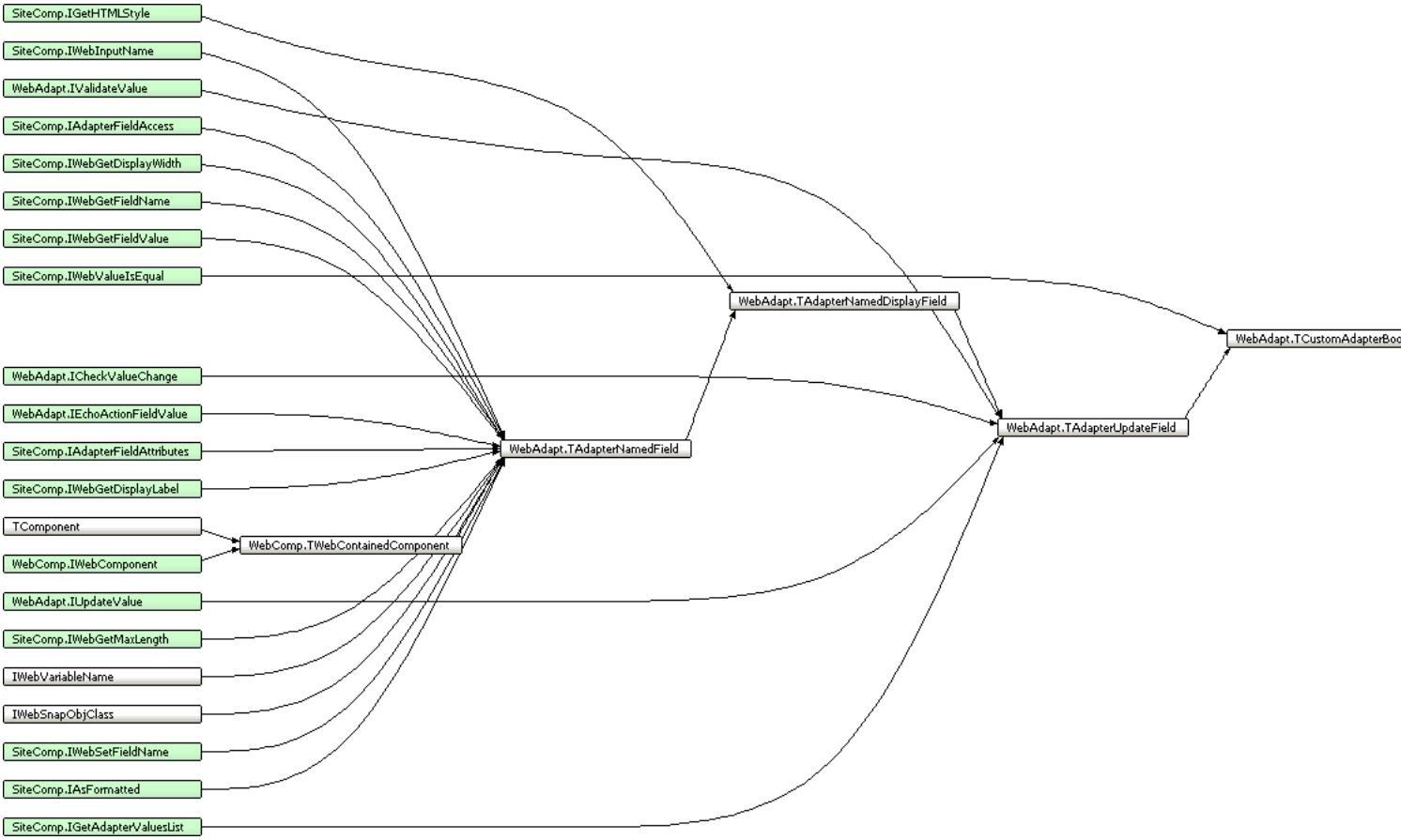
See Also

ImplWebEnabled (see page 5585)

54.1.46 TCustomAdapterBooleanField Class

TCustomAdapterBooleanField is a base class for adapter fields that provide a single Boolean value.

Class Hierarchy



Pascal

```
TCustomAdapterBooleanField = class(TAdapterUpdateField, IWebValueIsEqual);
```

C++

```
class TCustomAdapterBooleanField : public TAdapterUpdateField, public IWebValueIsEqual;
```

File

WebAdapt

Description

TCustomAdapterBooleanField calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

See Also

TAdapterUpdateField (see page 5419)

TAdapterBooleanField (see page 5223)

TAdapter (see page 5190)

54.1.46.1 TCustomAdapterBooleanField Members

The following tables list the members exposed by TCustomAdapterBooleanField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
	ConvertValue (see page 5594)	Retrieves a Boolean value entered by the end-user.
	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
	DoOnValidateValue (see page 5426)	Validates field values entered by the end-user.
	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5594)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5595)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.

	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5595)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplIsEqual (see page 5596)	Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5596)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsEqual (see page 5596)	Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5597)	Called to retrieve the adapter field's value.

	OnUpdateValue (see page 5597)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
V	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.

	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.46.2 TCustomAdapterBooleanField Methods

The methods of the TCustomAdapterBooleanField class are listed here.

Protected Methods

	Name	Description
V	ConvertValue (see page 5594)	Retrieves a Boolean value entered by the end-user.
V	GetDisplayStyleType (see page 5594)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 5595)	Suggests an HTML input type for this adapter field.
V	ImplGetValue (see page 5595)	Returns the value of this adapter field as a variant.
V	ImplIsEqual (see page 5596)	Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.
V	ImplUpdateValue (see page 5596)	Called to update the adapter field's value with a value entered by the end-user.
	IsEqual (see page 5596)	Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.

54.1.46.2.1 TCustomAdapterBooleanField.ConvertValue Method

Retrieves a Boolean value entered by the end-user.

Pascal

```
function ConvertValue(AValue: IActionFieldValue): Variant; override;
```

C++

```
virtual __fastcall Variant ConvertValue(IActionFieldValue AValue);
```

Description

ConvertValue overrides TAdapterUpdateField (see page 5419)'s ConvertValue method. Convert value attempts to cast the value entered by the end-user to a Boolean. If the cast fails, false is returned.

See Also

ConvertValue (see page 5425)

54.1.46.2.2 TCustomAdapterBooleanField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
```

```
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5343)'s GetDisplayStyleType method. GetDispayStyleType returns htmldText, indicating that this adapter field should be displayed by as an HTML text block (paragraph).

See Also

GetDisplayStyleType (see page 5347)

TAdapterDisplayHTMLElementType (see page 1123)

54.1.46.2.3 TCustomAdapterBooleanField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliCheckBox indicating that a check box should be used to input this adapter field's value.

See Also

GetInputStyleType (see page 5347)

TAdapterInputHTMLElementType (see page 1124)

54.1.46.2.4 TCustomAdapterBooleanField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5368) property is true, then the value entered by end-user is returned. If the EchoActionFieldValue (see page 5368) property is false then the OnGetValue (see page 5597) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 5597)

54.1.46.2.5 TCustomAdapterBooleanField.ImplIsEqual Method

Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.

Pascal

```
function ImplIsEqual(const AValue: Variant): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplIsEqual(const Variant AValue);
```

Description

ImplIsEqual is called by IsEqual (see page 5596). ImplIsEqual compares the parameter value with the value associated with this adapter field.

See Also

IsEqual (see page 5596)

54.1.46.2.6 TCustomAdapterBooleanField.ImplUpdateValue Method

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If the OnUpdateValue (see page 5597) event handler is not nil (Delphi) or NULL (C++), it is called to update the adapter field's value.

See Also

OnUpdateValue (see page 5597)

54.1.46.2.7 TCustomAdapterBooleanField.IsEqual Method

Indicates whether the value of this field is equal to a particular Variant (see page 3421) value.

Pascal

```
function IsEqual(const AValue: Variant): Boolean;
```

C++

```
__fastcall Boolean IsEqual(const Variant AValue);
```

Description

IsEqual is called when server-side script calls the IsEqual method of this adapter field. Server-side script may use the IsEqual function to determine whether to check an HTML checkbox that displays this adapter field's value. IsEqual returns the result of ImplIsEqual (see page 5596).

See Also

ImplIsEqual (see page 5596)

54.1.46.3 TCustomAdapterBooleanField Events

The events of the TCustomAdapterBooleanField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 5597)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5597)	Called to update the adapter field's value with a value entered by the end-user.

54.1.46.3.1 TCustomAdapterBooleanField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetBooleanEvent;
```

C++

```
__property TAdapterFieldGetBooleanEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 5595) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5595)

54.1.46.3.2 TCustomAdapterBooleanField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldBooleanValueEvent;
```

C++

```
__property TAdapterFieldBooleanValueEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

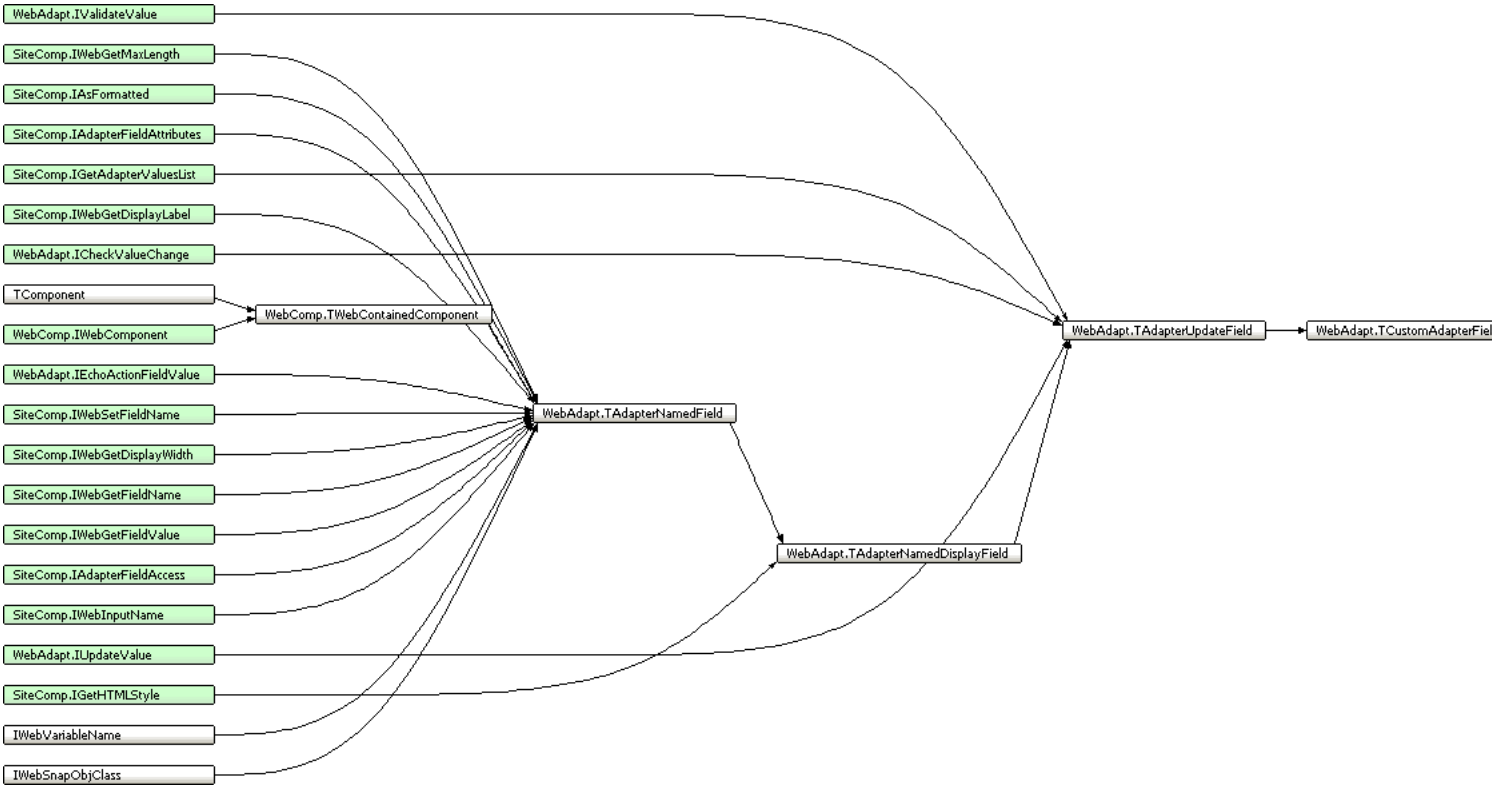
See Also

ImplUpdateValue (see page 5596)

54.1.47 TCustomAdapterField Class

TCustomAdapterField is a base class for adapter fields that provide a single value.

Class Hierarchy



Pascal

```
TCustomAdapterField = class(TAdapterUpdateField);
```

C++

```
class TCustomAdapterField : public TAdapterUpdateField;
```

File

WebAdapt

Description

TCustomAdapterField calls event handlers to retrieve the adapter field's value, validate a value entered by the end-user, and update the adapter field's value.

See Also

TAdapterUpdateField (see page 5419)

TAdapterField (see page 5267)

TAdapter (see page 5190)

54.1.47.1 TCustomAdapterField Members

The following tables list the members exposed by TCustomAdapterField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5602)	Validates a single value entered by the end-user.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5347)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5347)	Suggests an HTML input type for this adapter field.
V	GetInputStyleType (see page 5426)	Indicates the preferred HTML element to use when editing this adapter field's value.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
V	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
V	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
V	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.

	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5602)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5603)	Called to update the adapter field's value with a value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5603)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5604)	Called to update the adapter field's value with a value entered by the end-user.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.

	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.

	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.47.2 TCustomAdapterField Methods

The methods of the TCustomAdapterField class are listed here.

Protected Methods

	Name	Description
V	DoOnValidateValue (see page 5602)	Validates a single value entered by the end-user.
V	ImplGetValue (see page 5602)	Returns the value of this adapter field as a variant.
V	ImplUpdateValue (see page 5603)	Called to update the adapter field's value with a value entered by the end-user.

54.1.47.2.1 TCustomAdapterField.DoOnValidateValue Method

Validates a single value entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue calls the inherited DoOnValidateValue. If this succeeds by not raising an exception then additional checks are made. One check tests that the request does not contain more than one value for this adapter field. Another check is performed if the Required (see page 5351) property is True. In this case, the request must contain a value for this adapter field. An exception is raised if any of these checks fail.

See Also

DoOnValidateValue (see page 5426)

Required (see page 5351)

54.1.47.2.2 TCustomAdapterField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue function returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5368) property is True, then the value entered by end-user should is returned. If the EchoActionFieldValue (see page 5368) property is False then the OnGetValue (see page 5603) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 5603)

CheckViewAccess (see page 5356)

EchoActionFieldValue (see page 5368)

54.1.47.2.3 TCustomAdapterField.ImplUpdateValue Method

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value using the given action request. The AfieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated agains the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If the OnUpdateValue (see page 5604) event handler is not nil, it is called to update the adapter field's value.

See Also

OnUpdateValue (see page 5604)

54.1.47.3 TCustomAdapterField Events

The events of the TCustomAdapterField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 5603)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5604)	Called to update the adapter field's value with a value entered by the end-user.

54.1.47.3.1 TCustomAdapterField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetValueEvent;
```

C++

```
__property TAdapterFieldGetValueEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 5602) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5602)

54.1.47.3.2 TCustomAdapterField.OnUpdateValue Event

Called to update the adapter field's value with a value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldValueEvent;
```

C++

```
__property TAdapterFieldValueEvent OnUpdateValue;
```

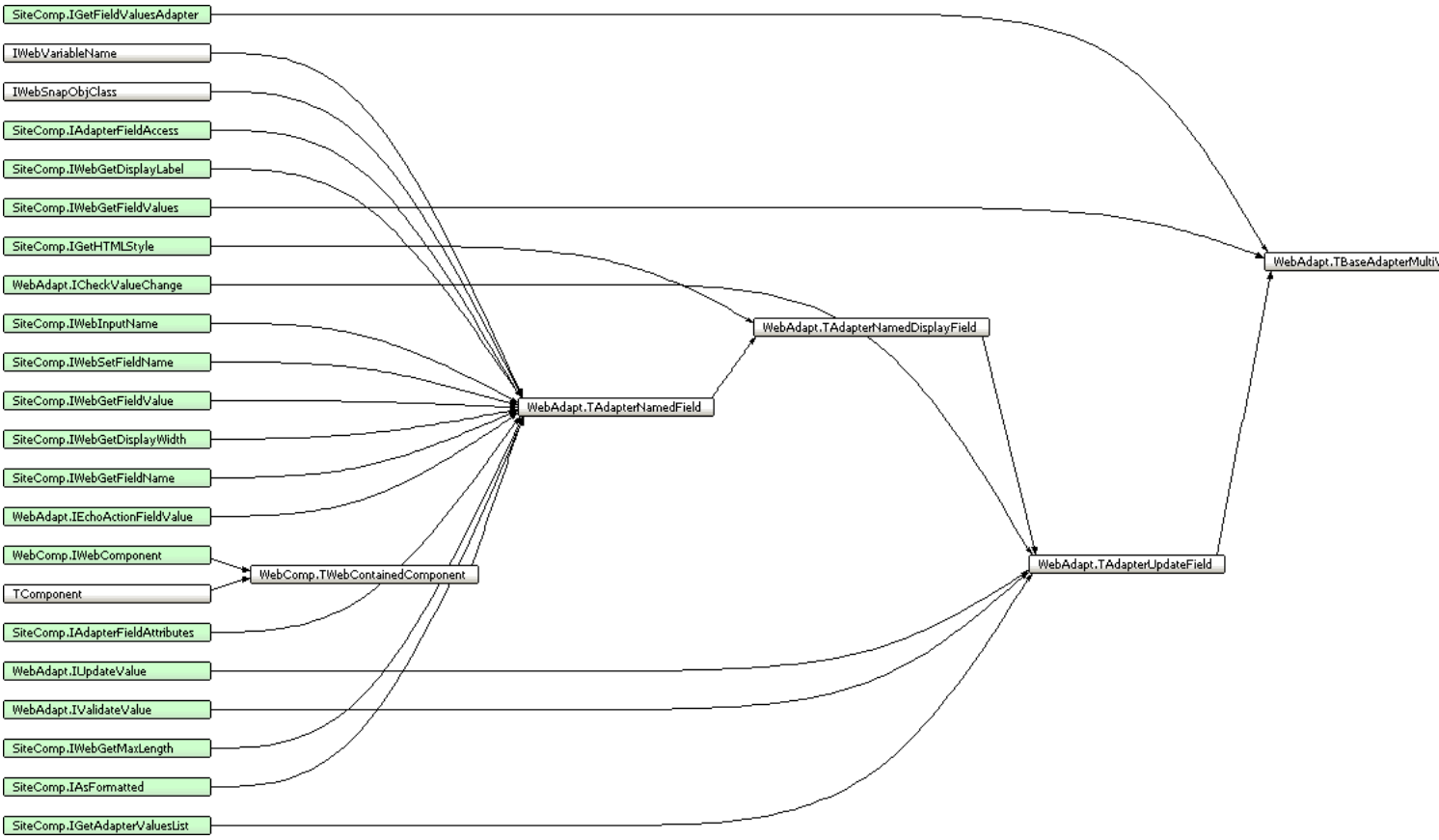
Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

54.1.48 TCustomAdapterFileField Class

TCustomAdapterFileField is a base class for adapter fields that can have associated files.

Class Hierarchy



Pascal

```
TCustomAdapterFileField = class(TBaseAdapterMultiValueField);
```

C++

```
class TCustomAdapterFileField : public TBaseAdapterMultiValueField;
```

File

WebAdapt

Description

Derive from TCustomAdapterFileField when the adapter field can have zero or more associated files. Typically, file input HTML elements are used to upload files. Multiple file input HTML elements having the same name can be used to upload multiple files in a single request.

See Also

TBaseAdapterMultiValueField (see page 5488)

TAdapterFileField (see page 5281)

TAdapterImageField (see page 5308)

54.1.48.1 TCustomAdapterFileField Members

The following tables list the members exposed by TCustomAdapterFileField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
V	DoOnValidateValue (see page 5609)	Validates files uploaded by the end-user.
	EndValueIterator (see page 5493)	Completes iteration.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5494)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5609)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5494)	Returns the adapter object that implements the Values property.
	GetFileNames (see page 5610)	Returns the count of files associated with this adapter field.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5610)	Suggests an HTML input type for this adapter field.

	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetValueCount (see page 5610)	Returns the number of values associated with this adapter field.
	GetValues (see page 5611)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' HasValue method.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5611)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplGetCurrentValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue (see page 5494) method.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetFieldValuesAdapter (see page 5496)	Returns the adapter object that implements the Values property.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasValue (see page 5496)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5611)	Updates the adapter field with value(s) entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5496)	Implements IWebGetFieldValues (see page 1092)' NextValueIteration method.
	Notification (see page 5431)	Clears references to a deleted component.

	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartValueIterator (see page 5497)	Implements IWebGetFieldValues (see page 1092)' StartValueIterator method.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetFileCount (see page 5613)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5613)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5613)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5614)	Validates values entered by the end-user.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 5492)	Constructor for TBaseAdapterMultiValueField (see page 5488).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 5492)	Destructor for TBaseAdapterMultiValueField (see page 5488).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndValueIterator (see page 1093)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetCurrentValue (see page 1094)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 1031)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.

V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1094)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1094)	Position the iteration pointer on the next value.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartValueIterator (see page 1095)	Reset (see page 3643) the iteration pointer to the first item.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
R	FileCount (see page 5612)	Returns one of the filenames associated with this adapter field.
R	FileNames (see page 5612)	Returns a file associated with this adapter field.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.48.2 TCustomAdapterFileField Methods

The methods of the TCustomAdapterFileField class are listed here.

Protected Methods

	Name	Description
	DoOnValidateValue (see page 5609)	Validates files uploaded by the end-user.
	GetDisplayStyleType (see page 5609)	Suggests an HTML display type for this adapter field.
	GetFileNames (see page 5610)	Returns the count of files associated with this adapter field.
	GetInputStyleType (see page 5610)	Suggests an HTML input type for this adapter field.
	GetValueCount (see page 5610)	Returns the number of values associated with this adapter field.
	GetValues (see page 5611)	Returns one of the values associated with the adapter field.
	ImplCheckValueChange (see page 5611)	Indicates whether the end-user may have uploaded new files for this adapter field.
	ImplUpdateValue (see page 5611)	Updates the adapter field with value(s) entered by the end-user.

54.1.48.2.1 TCustomAdapterFileField.DoOnValidateValue Method

Validates files uploaded by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5419)'s DoOnValidateValue method. If the OnValidateFiles (see page 5614) event handler is not nil (Delphi) or NULL (C++), DoOnValidateValue calls it to validate uploaded files. If the Required (see page 5351) property is true, DoOnValidateValues also verifies that 1 of more values have been entered by the end-user. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5426)

OnValidateValue (see page 5433)

Required (see page 5351)

OnValidateFiles (see page 5614)

54.1.48.2.2 TCustomAdapterFileField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5343)'s GetDisplayStyleType. GetDispayStyleTypereturns htmldList, indicating that this adapter field should be displayed by as a list of read-only values.

See Also

GetDisplayStyle (see page 5346)

TAdapterDisplayHTMLElementType (see page 1123)

54.1.48.2.3 TCustomAdapterFileField.GetFileNames Method

Returns the count of files associated with this adapter field.

Pascal

```
function GetFileNames(I: Integer): string;
```

C++

```
__fastcall UnicodeString GetFileNames(int I);
```

Description

GetFileNames implements the FileNames (see page 5612) property. If the EchoActionFieldValue (see page 5368) flag is true then the filename is extracted from the HTTP request. Otherwise, the OnGetFileNames (see page 5613) event is called. GetFileNames returns and empty string if a value can't be found because OnGetFileNames (see page 5613) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a filename for this adapter field.

See Also

FileNames (see page 5612)

OnGetFileNames (see page 5613)

54.1.48.2.4 TCustomAdapterFileField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliFile indicating that file upload HTML elements should be used to input values for this adapter field.

See Also

GetInputStyleType (see page 5347)

TAdapterInputHTMLElementType (see page 1124)

54.1.48.2.5 TCustomAdapterFileField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount implements the FileCount (see page 5612) property. If the EchoActionFieldValue (see page 5368) flag is

true then the result is the number of values in the HTTP request for this adapter field. Otherwise, the OnGetFileCount (see page 5613) event is called. GetValueCount returns zero if a value can't be found because OnGetFileCount (see page 5613) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

FileCount (see page 5612)

OnGetFileCount (see page 5613)

54.1.48.2.6 TCustomAdapterFileField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues returns the result of GetFileNames (see page 5610).

See Also

GetFileNames (see page 5610)

54.1.48.2.7 TCustomAdapterFileField.ImplCheckValueChange Method

Indicates whether the end-user may have uploaded new files for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 5419)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the end-user has uploaded any files.

See Also

IActionRequest

ImplCheckValueChange (see page 5428)

54.1.48.2.8 TCustomAdapterFileField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue overrides TAdapterUpdateField (see page 5419)'s ImplUpdateValue method. After checking that the end-user has rights to modify this adapter field, the OnUploadFiles (see page 5613) event is called.

See Also

IActionRequest

OnUploadFiles (see page 5613)

ImplUpdateValue (see page 5430)

54.1.48.3 TCustomAdapterFileField Properties

The properties of the TCustomAdapterFileField class are listed here.

Public Properties

	Name	Description
	FileCount (see page 5612)	Returns one of the filenames associated with this adapter field.
	FileNames (see page 5612)	Returns a file associated with this adapter field.

54.1.48.3.1 TCustomAdapterFileField.FileCount Property

Returns one of the filenames associated with this adapter field.

Pascal

```
property FileCount: Integer;
```

C++

```
__property int FileCount;
```

Description

```
__property int FileCount = {read=GetValueCount, nodefault};
```

FileCount can be used with the FileNames (see page 5612) property to access all the filenames associated with this adapter field. FileCount is implemented by GetValueCount (see page 5610).

See Also

FileNames (see page 5612)

GetValueCount (see page 5610)

54.1.48.3.2 TCustomAdapterFileField.FileNames Property

Returns a file associated with this adapter field.

Pascal

```
property FileNames [I: Integer]: string;
```

C++

```
__property UnicodeString FileNames[int I];
```

Description

FileNames can be used with the FileCount (see page 5612) property to access all the filenames associated with this adapter field. FileNames is implemented by GetFileNames (see page 5610).

See Also

FileCount (see page 5612)

GetFileNames (see page 5610)

54.1.48.4 TCustomAdapterFileField Events

The events of the TCustomAdapterFileField class are listed here.

Public Events

	Name	Description
	OnGetFileCount (see page 5613)	Returns the number of values associated with this adapter field.
	OnGetFileNames (see page 5613)	Returns one of the filenames associated with the adapter field.
	OnUploadFiles (see page 5613)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateFiles (see page 5614)	Validates values entered by the end-user.

54.1.48.4.1 TCustomAdapterFileField.OnGetFileCount Event

Returns the number of values associated with this adapter field.

Pascal

```
property OnGetFileCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetFileCount;
```

Description

Write (see page 3707) an OnGetFileCount event handler to indicate the number of filenames associated with this adapter field.

See Also

GetValueCount (see page 5610)

54.1.48.4.2 TCustomAdapterFileField.OnGetFileNames Event

Returns one of the filenames associated with the adapter field.

Pascal

```
property OnGetFileNames: TAdapterFieldGetMultiValueStringEvent;
```

C++

```
__property TAdapterFieldGetMultiValueStringEvent OnGetFileNames;
```

Description

Write (see page 3707) an OnGetFileNames event handler to provide the names of files associated with this adapter field.

See Also

GetFileNames (see page 5610)

54.1.48.4.3 TCustomAdapterFileField.OnUploadFiles Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUploadFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnUploadFiles;
```

Description

OnUploadFiles is called by ImplUpdateValue (see page 5611). Write (see page 3707) an OnUploadFiles event handler to accept files uploaded by the end-user. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5611)

TUpdateFileList (see page 5864)

54.1.48.4.4 TCustomAdapterFileField.OnValidateFiles Event

Validates values entered by the end-user.

Pascal

```
property OnValidateFiles: TAdapterFieldFileListEvent;
```

C++

```
__property TAdapterFieldFileListEvent OnValidateFiles;
```

Description

OnValidateFiles is called by DoOnValidateValue (see page 5609). Implement this event handler to validate files uploaded by the end-user. Raise an exception to indicate a validation error.

See Also

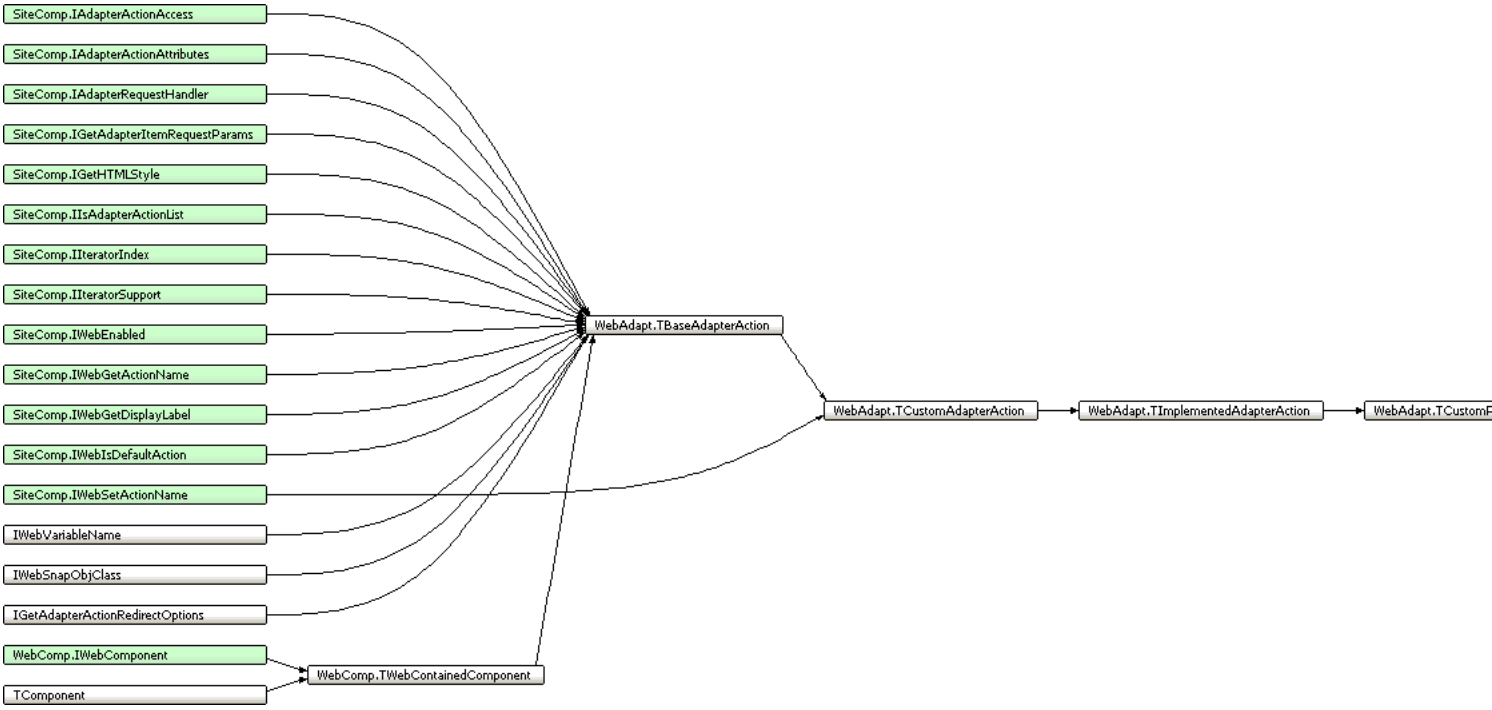
DoOnValidateValue (see page 5609)

TUpdateFileList (see page 5864)

54.1.49 TCustomAdapterGotoPageAction Class

TCustomAdapterGotoPageAction is an adapter action that supports commands to go to a page of a TDefaultFieldsPagedAdapter (see page 5747).

Class Hierarchy



Pascal

```
TCustomAdapterGotoPageAction = class(TCustomPagedAdapterAction);
```

C++

```
class TCustomAdapterGotoPageAction : public TCustomPagedAdapterAction;
```

File

WebAdapt

Description

TCustomAdapterGotoPageAction is an adapter action that has a list of commands. Each command has a parameter that identifies a page in the parent adapter. When executed, TCustomAdapterGotoPageAction examines the parameter and changes the current page in the parent adapter.

See Also

TCustomPagedAdapterAction (see page 5722)

TAdapterGotoPageAction (see page 5291)

54.1.49.1 TCustomAdapterGotoPageAction Members

The following tables list the members exposed by TCustomAdapterGotoPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.

	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5727)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5619)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5619)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5620)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5620)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5620)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5621)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5621)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
V	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5622)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5622)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Adapter (see page 5727)	Returns the parent adapter of this adapter action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetPageName (see page 5622)	Provides the display labels for each goto page command.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.49.2 TCustomAdapterGotoPageAction Methods

The methods of the TCustomAdapterGotoPageAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5619)	Gets the name used to reference this adapter action with server-side script.
	GetDisplayStyleType (see page 5619)	Suggests an HTML display type for this adapter action.
	GetItemCount (see page 5620)	Returns the number of commands defined by this adapter action.
	ImplExecuteActionRequest (see page 5620)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetAdapterItemRequestParams (see page 5620)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5621)	Returns the suggested display label for this adapter action.
	ImplGetIsAdapterActionList (see page 5621)	Indicates whether this adapter action may have multiple commands.
	ImplIsDefaultAction (see page 5622)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 5622)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

54.1.49.2.1 TCustomAdapterGotoPageAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns 'GotoPage'.

See Also

GetActionName (see page 5455)

GetDefaultActionName (see page 5456)

54.1.49.2.2 TCustomAdapterGotoPageAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaAnchor, indicating that goto page commands should be displayed using a hyperlink.

See Also

GetDisplayStyle (see page 5457)

54.1.49.2.3 TCustomAdapterGotoPageAction.GetItemCount Method

Returns the number of commands defined by this adapter action.

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

Description

TCustomAdapterGotoPageAction (see page 5614) returns the value of the parent adapter's PageCount property.

See Also

ItemCount (see page 5475)

ItemIndex (see page 5476)

GetIsAdapterActionList (see page 5460)

PageCount (see page 5761)

54.1.49.2.4 TCustomAdapterGotoPageAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest sets the parent adapter's current page using the page index passed in the AActionRequest parameter.

See Also

ImplExecuteActionRequest (see page 5464)

54.1.49.2.5 TCustomAdapterGotoPageAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 5456). ImplGetAdapterItemRequestParams adds the value of the iteration index as a parameter. Then the inherited ImplGetAdapterItemRequestParams method is called so that any event handlers that may be defined will be called.

See Also

GetAdapterItemRequestParams (see page 5456)

OnGetActionParams (see page 5579)

OnGetParams (see page 5478)

ItemIndex (see page 5476)

54.1.49.2.6 TCustomAdapterGotoPageAction.ImplGetDisplayLabel Method

Returns the suggested display label for this adapter action.

Pascal

```
function ImplGetDisplayLabel: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetDisplayLabel();
```

Description

ImplGetDisplayLabel returns the display label suggested for the goto page command that is identified by the iteration index. If the OnGetPageName (see page 5622) event handler is not nil (Delphi) or NULL (C++), it is called. Otherwise, the ItemIndex (see page 5476) property is converted to a string and returned.

See Also

DisplayLabel (see page 5474)

ItemIndex (see page 5476)

OnGetPageName (see page 5622)

54.1.49.2.7 TCustomAdapterGotoPageAction.ImplGetIsAdapterActionList Method

Indicates whether this adapter action may have multiple commands.

Pascal

```
function ImplGetIsAdapterActionList: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplGetIsAdapterActionList();
```

Description

ImplGetIsAdapterActionList is called by GetIsAdapterActionList (see page 5460). ImplGetIsAdapterActionList returns true if the parent adapters PageSize property is greater than zero.

See Also

GetIsAdapterActionList (see page 5460)

PageSize (see page 5761)

54.1.49.2.8 TCustomAdapterGotoPageAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 5471). ImplIsDefaultAction returns true if the display characteristics identified by ADisplay indicate that the goto page commands are appropriate.

See Also

IsDefaultAction (see page 5471)

54.1.49.2.9 TCustomAdapterGotoPageAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of false if the iteration index is equal to the parent adapter's current page. In other words, the command that goes to the parent adapter's current page will be disabled. Commands to goto all other pages will be enabled.

See Also

WebEnabled (see page 5473)

CurrentPage (see page 5760)

ItemIndex (see page 5476)

54.1.49.3 TCustomAdapterGotoPageAction Events

The events of the TCustomAdapterGotoPageAction class are listed here.

Public Events

	Name	Description
	OnGetPageName (see page 5622)	Provides the display labels for each goto page command.

54.1.49.3.1 TCustomAdapterGotoPageAction.OnGetPageName Event

Provides the display labels for each goto page command.

Pascal

```
property OnGetPageName: TWebFormGetPageNameEvent;
```

C++

```
__property TWebFormGetPageNameEvent OnGetPageName;
```

Description

Implement this event handler to specify display labels for the commands that goto pages in the parent adapter. APageIndex is the zero-based index of the page to goto.

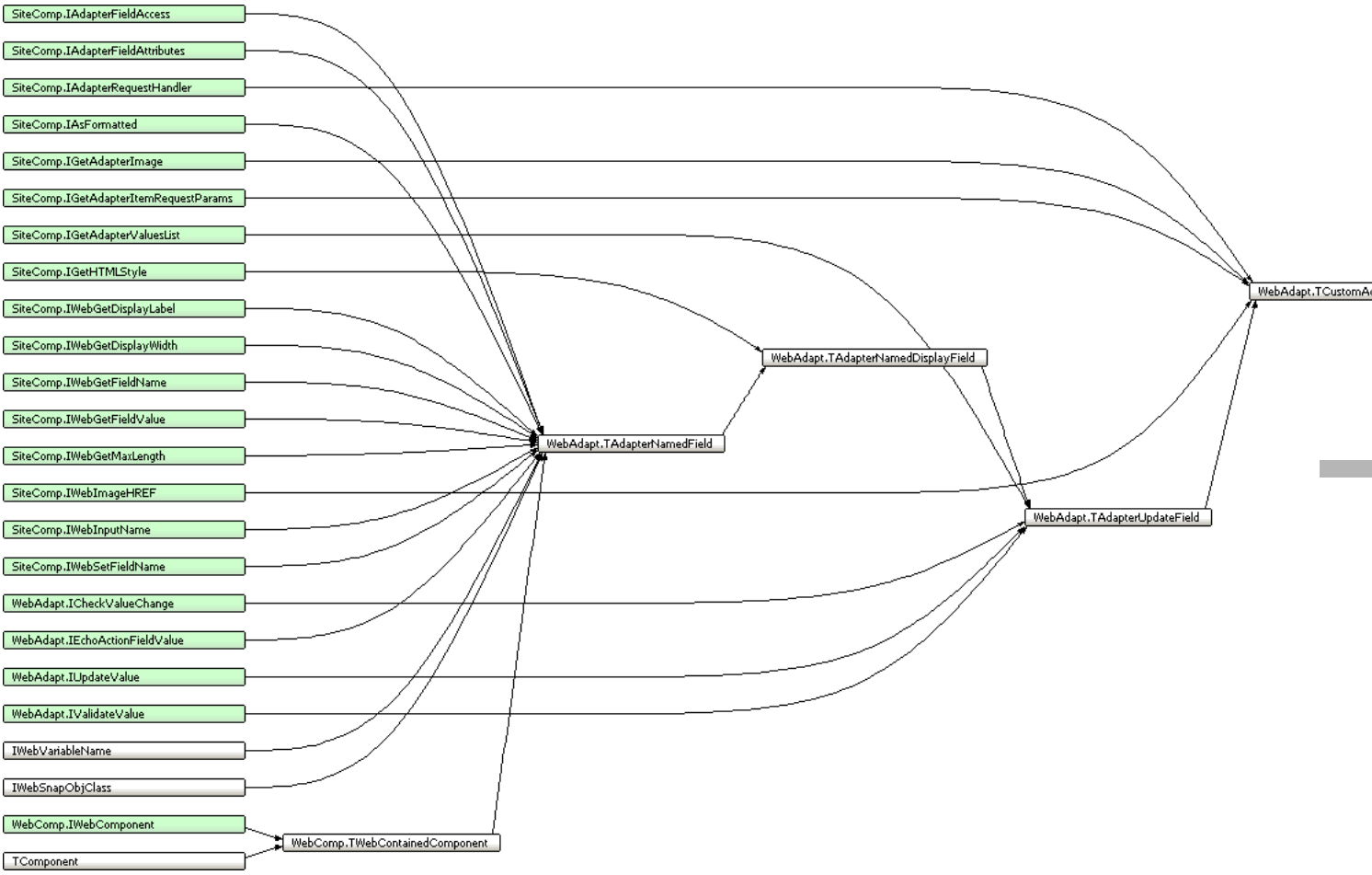
See Also

ImplGetDisplayLabel (see page 5621)

54.1.50 TCustomAdapterImageField Class

TCustomAdapterImageField is a base class for adapter fields that have an associated image file.

Class Hierarchy



Pascal

```
TCustomAdapterImageField = class(TAdapterUpdateField, IGetAdapterImage,
IGetAdapterItemRequestParams, IAdapterRequestHandler, IWebImageHREF);
```

C++

```
class TCustomAdapterImageField : public TAdapterUpdateField, public IGetAdapterImage, public
IGetAdapterItemRequestParams, public IAdapterRequestHandler, public IWebImageHREF;
```

File

WebAdapt

Description

Derived from TCustomAdapterImageField when the adapter field can an associated image file. Typically, a file input HTML element is used to upload an image file. HTML IMG elements are used to display the image.

See Also

TAdapterImageField (see page 5308)

TAdapterFileField (see page 5281)

54.1.50.1 TCustomAdapterImageField Members

The following tables list the members exposed by TCustomAdapterImageField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
V	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
V	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
	CreateRequestContext (see page 5628)	Creates AdapterRequest and AdapterResponse objects.
V	DoOnValidateValue (see page 5628)	Validates files uploaded by the end-user.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterImage (see page 5629)	Provides an object that can be used to access the image properties of this adapter field.
	GetAdapterItemRequestParams (see page 5629)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
V	GetDisplayStyleType (see page 5630)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
V	GetInputStyleType (see page 5630)	Suggests an HTML input type for this adapter field.

	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HandleRequest (see page 5630)	Handles a request for the image file associated with this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5631)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 5631)	Creates AdapterRequest and AdapterResponse objects.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5632)	Returns the name of the image file associated with this adapter field.
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHandleRequest (see page 5632)	Handles a request for the image file associated with this adapter field.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5632)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	ImplWebImageHREF (see page 5633)	Returns a URL that can be used to access the image file associated with this adapter field.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.

	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	RenderAdapterImage (see page 5633)	Responds to a request for an image file by providing a stream containing the image.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.
	WebImageHREF (see page 5634)	Returns a URL that can be used to access the image file associated with this adapter field.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetHREF (see page 5635)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 5636)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5636)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 5636)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5637)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5637)	Used to validate files uploaded by the end user.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.

	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	HREF (see page 5634)	Defines a URL that references the image associated with the adapter field.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5635)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.50.2 TCustomAdapterImageField Methods

The methods of the TCustomAdapterImageField class are listed here.

Protected Methods

	Name	Description
	CreateRequestContext (see page 5628)	Creates AdapterRequest and AdapterResponse objects.
V	DoOnValidateValue (see page 5628)	Validates files uploaded by the end-user.
	GetAdapterImage (see page 5629)	Provides an object that can be used to access the image properties of this adapter field.

	GetAdapterItemRequestParams (see page 5629)	Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.
V	GetDisplayStyleType (see page 5630)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 5630)	Suggests an HTML input type for this adapter field.
	HandleRequest (see page 5630)	Handles a request for the image file associated with this adapter field.
V	ImplCheckValueChange (see page 5631)	Indicates whether the end-user may have uploaded a new image file for this adapter field.
	ImplCreateRequestContext (see page 5631)	Creates AdapterRequest and AdapterResponse objects.
V	ImplGetValue (see page 5632)	Returns the name of the image file associated with this adapter field.
V	ImplHandleRequest (see page 5632)	Handles a request for the image file associated with this adapter field.
V	ImplUpdateValue (see page 5632)	Updates the adapter field with value entered by the end-user.
V	ImplWebImageHREF (see page 5633)	Returns a URL that can be used to access the image file associated with this adapter field.
V	RenderAdapterImage (see page 5633)	Responds to a request for an image file by providing a stream containing the image.
	WebImageHREF (see page 5634)	Returns a URL that can be used to access the image file associated with this adapter field.

54.1.50.2.1 TCustomAdapterImageField.CreateRequestContext Method

Creates AdapterRequest and AdapterResponse objects.

Pascal

```
procedure CreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

CreateRequestContext implements IAdapterRequestHandler (see page 1012)'s CreateRequestContext method. CreateRequestContext calls ImplCreateRequestContext (see page 5631).

See Also

IAdapterDispatchParams (see page 987)

IAdapterRequestHandler (see page 1012)

AdapterRequest (see page 5971)

AdapterResponse (see page 5972)

ImplCreateRequestContext (see page 5631)

54.1.50.2.2 TCustomAdapterImageField.DoOnValidateValue Method

Validates files uploaded by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5419)'s DoOnValidateValue method. If the OnValidateFile (see page 5637) event handler is not nil (Delphi) or NULL (C++), DoOnValidateValue calls it to validate an uploaded file. DoOnValidateValues also validates that not more than one image file was uploaded and, if the required property is true, that a

single file was uploaded. An exception is raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5426)

OnValidateValue (see page 5433)

Required (see page 5351)

OnValidateFile (see page 5637)

54.1.50.2.3 TCustomAdapterImageField.GetAdapterImage Method

Provides an object that can be used to access the image properties of this adapter field.

Pascal

```
function GetAdapterImage: TComponent;
```

C++

```
__fastcall TComponent * GetAdapterImage();
```

Description

GetAdapterImage returns Self. GetAdapterImage will be called when server-side script references the Image property of this adapter field.

54.1.50.2.4 TCustomAdapterImageField.GetAdapterItemRequestParams Method

Provides a unique identifier and a set of name/value pairs that will be used to create a URL and also be used later when a request is received to generate the adapter field's image file.

Pascal

```
procedure GetAdapterItemRequestParams(var AIdentifier: string; AParams:
IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(UnicodeString AIdentifier, IAdapterItemRequestParams
AParams);
```

Description

WebImageHREF (see page 5634) and GetAdapterItemRequestParams are used to identify the URL that is used to retrieve the image file associated with this adapter field. These functions are called when server-side script uses the Image.AsHREF property. If WebImageHREF (see page 5634) returns False then GetAdapterItemRequestParams is called. WebImageHREF (see page 5634) is responsible for returning a reference to an external object. GetAdapterItemRequestParams is used to generate a URL that will result in a request to the adapter field to dynamically generate the image file. GetAdapterItemRequestParams calls OnGetParams (see page 5636) to get the name/value pairs.

See Also

IAdapterItemRequestParams (see page 1005)

RenderAdapterImage (see page 5633)

OnGetParams (see page 5636)

WebImageHREF (see page 5634)

TAdapterDispatcher (see page 6148)

54.1.50.2.5 TCustomAdapterImageField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5343)'s GetDisplayStyleType method. GetDispayStyleType returns htmldImage, indicating that this adapter field should be displayed by an HTML image element.

See Also

GetDisplayStyleType (see page 5347)

54.1.50.2.6 TCustomAdapterImageField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetInputStyleType returns htmliFile indicating that a file upload HTML element should be used to input a value for this adapter field.

See Also

GetInputStyleType (see page 5347)

54.1.50.2.7 TCustomAdapterImageField.HandleRequest Method

Handles a request for the image file associated with this adapter field.

Pascal

```
function HandleRequest(DispatchParams: IAdapterDispatchParams): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

HandleRequest implements IAdapterRequestHandler (see page 1012)'s HandleRequest method. HandleRequest calls ImplHandleRequest (see page 5632).

See Also

IAdapterDispatchParams (see page 987)

ImplHandleRequest (see page 5632)

IAdapterRequestHandler (see page 1012)

54.1.50.2.8 TCustomAdapterImageField.ImplCheckValueChange Method

Indicates whether the end-user may have uploaded a new image file for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 5419)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the end-user has uploaded an image file.

See Also

IActionRequest

ImplCheckValueChange (see page 5428)

54.1.50.2.9 TCustomAdapterImageField.ImplCreateRequestContext Method

Creates AdapterRequest and AdapterResponse objects.

Pascal

```
procedure ImplCreateRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall ImplCreateRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

ImplCreateRequestContext creates a new request and response objects based on the specified parameters and stores it with the global Web context. The adapter dispatcher calls this function before calling HandleRequest (see page 5630). The WebSnap can access the resulting request and response context by reading the AdapterRequest and AdapterResponse properties of the Web context, which is available through the global WebContext (see page 6003) function.

DispatchParams is an interface that contains the parameters that were included in the incoming request. These parameters can then be used to identify an image for this field's value.

See Also

CreateRequestContext (see page 5628)

AdapterRequest (see page 5971)

AdapterResponse (see page 5972)

IAdapterDispatchParams (see page 987)

WebContext (see page 6003)

54.1.50.2.10 TCustomAdapterImageField.ImplGetValue Method

Returns the name of the image file associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the name of the image file associated with this adapter field

First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5368) property is true, then the file name entered by end-user is returned. If the EchoActionFieldValue (see page 5368) property is false then the OnGetImageName (see page 5636) event handler is called to retrieve the name of the image file. The value returned by ImplGetValue is used when server-side script references the Value (see page 5432) or DisplayText properties.

See Also

OnGetImageName (see page 5636)

EchoActionFieldValue (see page 5368)

54.1.50.2.11 TCustomAdapterImageField.ImplHandleRequest Method

Handles a request for the image file associated with this adapter field.

Pascal

```
function ImplHandleRequest(DispatchParams: IAdapterDispatchParams): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(IAdapterDispatchParams DispatchParams);
```

Description

ImplHandleRequest is called by HandleRequest (see page 5630). ImplHandleRequest extracts IImageRequest and IImageResponse from the AdapterRequest and AdapterResponse. ImplHandleRequest returns the result of RenderAdapterImage (see page 5633).

See Also

IAdapterDispatchParams (see page 987)

AdapterRequest (see page 5971)

AdapterResponse (see page 5972)

RenderAdapterImage (see page 5633)

IImageRequest

54.1.50.2.12 TCustomAdapterImageField.ImplUpdateValue Method

Updates the adapter field with value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If action request single file for this adapter field, the OnUploadFile (see page 5637) event handler is called.

See Also

OnUploadFile (see page 5637)

ImplUpdateValue (see page 5430)

54.1.50.2.13 TCustomAdapterImageField.ImplWebImageHREF Method

Returns a URL that can be used to access the image file associated with this adapter field.

Pascal

```
function ImplWebImageHREF(var AHREF: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplWebImageHREF(UnicodeString AHREF);
```

Description

ImplWebImageHREF is called by WebImageHREF (see page 5634). If the OnGetHREF (see page 5635) event handler is not nil (Delphi) or NULL (C++), ImplWebImageHREF will call it passing the value of the HREF (see page 5634) property.

See Also

WebImageHREF (see page 5634)

HREF (see page 5634)

OnGetHREF (see page 5635)

54.1.50.2.14 TCustomAdapterImageField.RenderAdapterImage Method

Responds to a request for an image file by providing a stream containing the image.

Pascal

```
procedure RenderAdapterImage(ARequest: IImageRequest; AResponse: IImageResponse); virtual;
```

C++

```
virtual __fastcall RenderAdapterImage(IImageRequest ARequest, IImageResponse AResponse);
```

Description

RenderAdapterImage is called when the TAdapterDispatcher detects a request for an image from this adapter field. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the OnGetImage (see page 5636) event handler is not nil (Delphi) or NULL (C++), it is called to retrieve the image file as a stream.

See Also

OnGetImage (see page 5636)

TAdapterDispatcher (see page 6148)

ImplHandleRequest (see page 5632)

54.1.50.2.15 TCustomAdapterImageField.WebImageHREF Method

Returns a URL that can be used to access the image file associated with this adapter field.

Pascal

```
function WebImageHREF(var AHREF: string): Boolean;
```

C++

```
__fastcall Boolean WebImageHREF(UnicodeString AHREF);
```

Description

WebImageHREF and GetAdapterItemRequestParams (see page 5629) are used to identify the URL that is used to retrieve an image file associated with this adapter field. These functions are called when server-side script uses the Image.AsHREF property. WebImageHREF returns the result of ImplWebImageHREF (see page 5633).

See Also

ImplWebImageHREF (see page 5633)

GetAdapterItemRequestParams (see page 5629)

54.1.50.3 TCustomAdapterImageField Properties

The properties of the TCustomAdapterImageField class are listed here.

Public Properties

	Name	Description
	HREF (see page 5634)	Defines a URL that references the image associated with the adapter field.
	ReadOnly (see page 5635)	Indicates whether this adapter field is read-only.

54.1.50.3.1 TCustomAdapterImageField.HREF Property

Defines a URL that references the image associated with the adapter field.

Pascal

```
property HREF: string;
```

C++

```
__property UnicodeString HREF;
```

Description

Use the HREF property when this adapter field is associated with an external image file URL. Use the OnGetHREF (see page 5635) event to dynamically define an external image file URL. If the image file will be dynamically generated, use the OnGetImage (see page 5636) and OnGetImageParams event handlers instead of the HREF property or the OnGetHREF (see page 5635) event handler.

See Also

ImplWebImageHREF (see page 5633)

OnGetHREF (see page 5635)

OnGetImage (see page 5636)

OnGetImageName (see page 5636)

54.1.50.3.2 TCustomAdapterImageField.ReadOnly Property

Indicates whether this adapter field is read-only.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

Set this property to true to indicate that this field should be represented as a read-only field in the HTML form. Set this property to false to indicate that this field should be represented as an editable field in the HTML form. The value of this property affects the value returned by GetViewMode which, in turn, may affect the server-side script generated to represent this adapter field.

See Also

GetViewMode (see page 5348)

54.1.50.4 TCustomAdapterImageField Events

The events of the TCustomAdapterImageField class are listed here.

Public Events

	Name	Description
	OnGetHREF (see page 5635)	Provides the URL of an external image file associated with this adapter field.
	OnGetImage (see page 5636)	Retrieves a stream containing the image file associated with this adapter field.
	OnGetImageName (see page 5636)	Returns the name of the image file associated with this adapter field.
	OnGetParams (see page 5636)	Provides name/value pairs that identify this adapter field's image.
	OnUploadFile (see page 5637)	Updates this adapter field with an image file uploaded by the end-user.
	OnValidateFile (see page 5637)	Used to validate files uploaded by the end user.

54.1.50.4.1 TCustomAdapterImageField.OnGetHREF Event

Provides the URL of an external image file associated with this adapter field.

Pascal

```
property OnGetHREF: TImageFieldGetHREFEvent;
```

C++

```
__property TImageFieldGetHREFEvent OnGetHREF;
```

Description

Implement OnGetHREF to dynamically identify the URL of an external image file associated with this adapter field. When OnGetHREF is called, the HREF (see page 5634) parameter will contain the value of the HREF (see page 5634) property.

To dynamically generate an image file for this adapter file, implement the OnGetImage (see page 5636) event handler instead.

See Also

ImplWebImageHREF (see page 5633)

HREF (see page 5634)

OnGetImage (see page 5636)

54.1.50.4.2 TCustomAdapterImageField.OnGetImage Event

Retrieves a stream containing the image file associated with this adapter field.

Pascal

```
property OnGetImage: TImageFieldGetImageEvent;
```

C++

```
__property TImageFieldGetImageEvent OnGetImage;
```

Description

Implement OnGetImage to dynamically generate an image file associated with this adapter field. Don't implement OnGetImage if the image file is external. Instead implement OnGetHREF (see page 5635) event handler or use the HREF (see page 5634) property to specify the URL of an external image.

See Also

RenderAdapterImage (see page 5633)

OnGetParams (see page 5636)

OnGetHREF (see page 5635)

HREF (see page 5634)

54.1.50.4.3 TCustomAdapterImageField.OnGetImageName Event

Returns the name of the image file associated with this adapter field.

Pascal

```
property OnGetImageName: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetImageName;
```

Description

Implement this event handler to identify the name of the image file associated with this adapter field.

See Also

ImplGetValue (see page 5632)

54.1.50.4.4 TCustomAdapterImageField.OnGetParams Event

Provides name/value pairs that identify this adapter field's image.

Pascal

```
property OnGetParams: TImageFieldGetParamsEvent;
```

C++

```
__property TImageFieldGetParamsEvent OnGetParams;
```

Description

Implement this event handler to provide name/value pairs that are needed to identify this adapter field's image file. OnGetParams is called while generating an HTML page with a URL reference to this adapter field's image. The parameters returned by OnGetParams will be passed to OnGetImage (see page 5636) when the image file is requested.

See Also

GetAdapterItemRequestParams (see page 5629)

OnGetImage (see page 5636)

54.1.50.4.5 TCustomAdapterImageField.OnUploadFile Event

Updates this adapter field with an image file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

This event handler is called by ImplUpdateValue (see page 5632). This event handler should store the uploaded image file. Raise an exception to indicate an error.

See Also

ImplUpdateValue (see page 5632)

TAbstractWebRequestFile

54.1.50.4.6 TCustomAdapterImageField.OnValidateFile Event

Used to validate files uploaded by the end user.

Pascal

```
property OnValidateFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnValidateFile;
```

Description

Validates the file uploaded by the end-user. Raise an exception to indicate an error. For example, it raises an exception if the Mime type of the file is not supported.

See Also

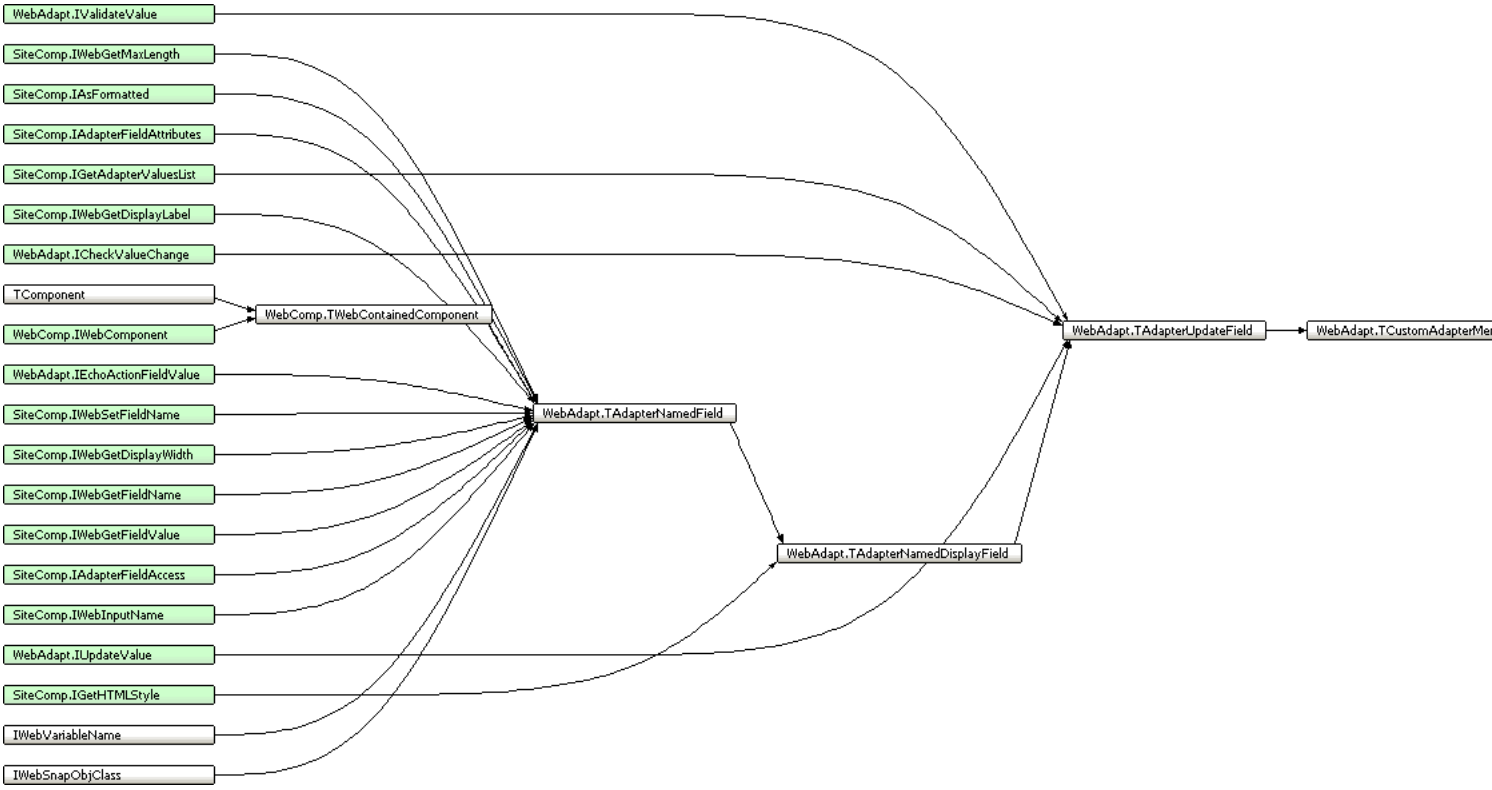
DoOnValidateValue (see page 5628)

TAbstractWebRequestFile

54.1.51 TCustomAdapterMemoField Class

TCustomAdapterMemoField is a base class for adapter fields that have a multi-line text value.

Class Hierarchy



Pascal

```
TCustomAdapterMemoField = class(TAdapterUpdateField);
```

C++

```
class TCustomAdapterMemoField : public TAdapterUpdateField;
```

File

WebAdapt

Description

Derive from TCustomAdapterMemoField when the adapter field has a multi-line text value. Typically, a text area HTML element is used to edit the value of this adapter field.

See Also

TAdapterUpdateField (see page 5419)

TAdapterMemoField (see page 5323)

54.1.51.1 TCustomAdapterMemoField Members

The following tables list the members exposed by TCustomAdapterMemoField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
	DoOnValidateValue (see page 5642)	Validates a value entered by the end user.
	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5642)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5643)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetValueCount (see page 5427)	Provides the number of values associated with this adapter field.
	GetValues (see page 5427)	Returns one of the values provided by this adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
	ImplCheckValueChange (see page 5643)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.

	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
	ImplGetValue (see page 5644)	Returns the value of this adapter field as a variant.
	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
	ImplUpdateValue (see page 5644)	Updates the adapter field with value entered by the end-user.
	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	Notification (see page 5431)	Clears references to a deleted component.
	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValue (see page 5645)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5645)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5645)	Called to update the adapter field's value with a file uploaded by the end-user.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
V	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
V	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.

	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
	Value (see page 5369)	Provides the value of an adapter field as a variant.
	Value (see page 5432)	Retrieves the value associated with this adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.51.2 TCustomAdapterMemoField Methods

The methods of the TCustomAdapterMemoField class are listed here.

Protected Methods

	Name	Description
	DoOnValidateValue (see page 5642)	Validates a value entered by the end user.
	GetDisplayStyleType (see page 5642)	Suggests an HTML display type for this adapter field.
	GetInputStyleType (see page 5643)	Suggests an HTML input type for this adapter field.
	ImplCheckValueChange (see page 5643)	Indicates whether the end-user may have entered a new value for this adapter field.
	ImplGetValue (see page 5644)	Returns the value of this adapter field as a variant.
	ImplUpdateValue (see page 5644)	Updates the adapter field with value entered by the end-user.

54.1.51.2.1 TCustomAdapterMemoField.DoOnValidateValue Method

Validates a value entered by the end user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5419)'s DoOnValidateValue method. After calling TAdapterUpdateField (see page 5419)'s DoOnValidateValue, DoOnValidate checks that only a single text file or file has been uploaded for this field. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5426)

Required (see page 5351)

IActionFieldValue

OnValidateFiles (see page 5614)

54.1.51.2.2 TCustomAdapterMemoField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType overrides TAdapterNamedDisplayField (see page 5343)'s GetDisplayStyleType method. GetDispayStyleType returns htmldText, indicating that this adapter field should be displayed by as an HTML text block.

See Also

GetDisplayStyleType (see page 5347)

TAdapterDisplayHTMLElementType (see page 1123)

54.1.51.2.3 TCustomAdapterMemoField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. GetDispayStyleTypereturns htmliTextArea, indicating that this adapter field's value should be input with an HTML multiline input field.

See Also

GetInputStyleType (see page 5347)

TAdapterInputHTMLElementType (see page 1124)

54.1.51.2.4 TCustomAdapterMemoField.ImplCheckValueChange Method

Indicates whether the end-user may have entered a new value for this adapter field.

Pascal

```
function ImplCheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
override;
```

C++

```
virtual __fastcall Boolean ImplCheckValueChange(IActionRequest AActionRequest, int
AFieldIndex);
```

Description

ImplCheckValueChange overrides TAdapterUpdateField (see page 5419)'s ImplCheckValueChange method. ImplCheckValueChange returns true if the HTTP request has a new value for this adapter field.

See Also

ImplCheckValueChange (see page 5428)

54.1.51.2.5 TCustomAdapterMemoField.ImplGetValue Method

Returns the value of this adapter field as a variant.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue function returns the value of this adapter field as a variant. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to view this field, an exception is raised. If the EchoActionFieldValue (see page 5368) property is true, then the value entered by end-user should is returned. If the EchoActionFieldValue (see page 5368) property is false then the OnGetValue (see page 5645) event handler is called to retrieve the value of the adapter field.

See Also

OnGetValue (see page 5645)

EchoActionFieldValue (see page 5368)

54.1.51.2.6 TCustomAdapterMemoField.ImplUpdateValue Method

Updates the adapter field with value entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

The ImplUpdateValue procedure updates the value of this adapter field using the given action request. The AFieldIndex parameter is used to extract the value entered by the user from the action request. First, the access rights defined for this adapter field are validated against the rights of the end-user. If the end-user does not have rights to modify this field, an exception is raised. If action request contains a string value for this adapter field, the OnUpdateValue (see page 5645) event handler is called. If the action request contains an uploaded file for this adapter field, the OnUploadFile (see page 5645) event handler is called.

See Also

OnUpdateValue (see page 5645)

OnUploadFile (see page 5645)

ImplUpdateValue (see page 5430)

54.1.51.3 TCustomAdapterMemoField Events

The events of the TCustomAdapterMemoField class are listed here.

Public Events

	Name	Description
	OnGetValue (see page 5645)	Called to retrieve the adapter field's value.
	OnUpdateValue (see page 5645)	Called to update the adapter field's value with a text value entered by the end-user.
	OnUploadFile (see page 5645)	Called to update the adapter field's value with a file uploaded by the end-user.

54.1.51.3.1 TCustomAdapterMemoField.OnGetValue Event

Called to retrieve the adapter field's value.

Pascal

```
property OnGetValue: TAdapterFieldGetStringEvent;
```

C++

```
__property TAdapterFieldGetStringEvent OnGetValue;
```

Description

OnGetValue is called by ImplGetValue (see page 5644) to retrieve the value associated with this adapter field.

See Also

ImplGetValue (see page 5644)

54.1.51.3.2 TCustomAdapterMemoField.OnUpdateValue Event

Called to update the adapter field's value with a text value entered by the end-user.

Pascal

```
property OnUpdateValue: TAdapterFieldStringEvent;
```

C++

```
__property TAdapterFieldStringEvent OnUpdateValue;
```

Description

OnUpdateValue is called when an action request contains a value for this adapter field. If this event handler is not implemented, the value entered by the end-user will be ignored.

See Also

ImplUpdateValue (see page 5644)

54.1.51.3.3 TCustomAdapterMemoField.OnUploadFile Event

Called to update the adapter field's value with a file uploaded by the end-user.

Pascal

```
property OnUploadFile: TAdapterFieldFileEvent;
```

C++

```
__property TAdapterFieldFileEvent OnUploadFile;
```

Description

OnUploadFile is called when an action request contains an uploaded file for this adapter field. If this event handler is not implemented, the file uploaded by the end-user will be ignored.

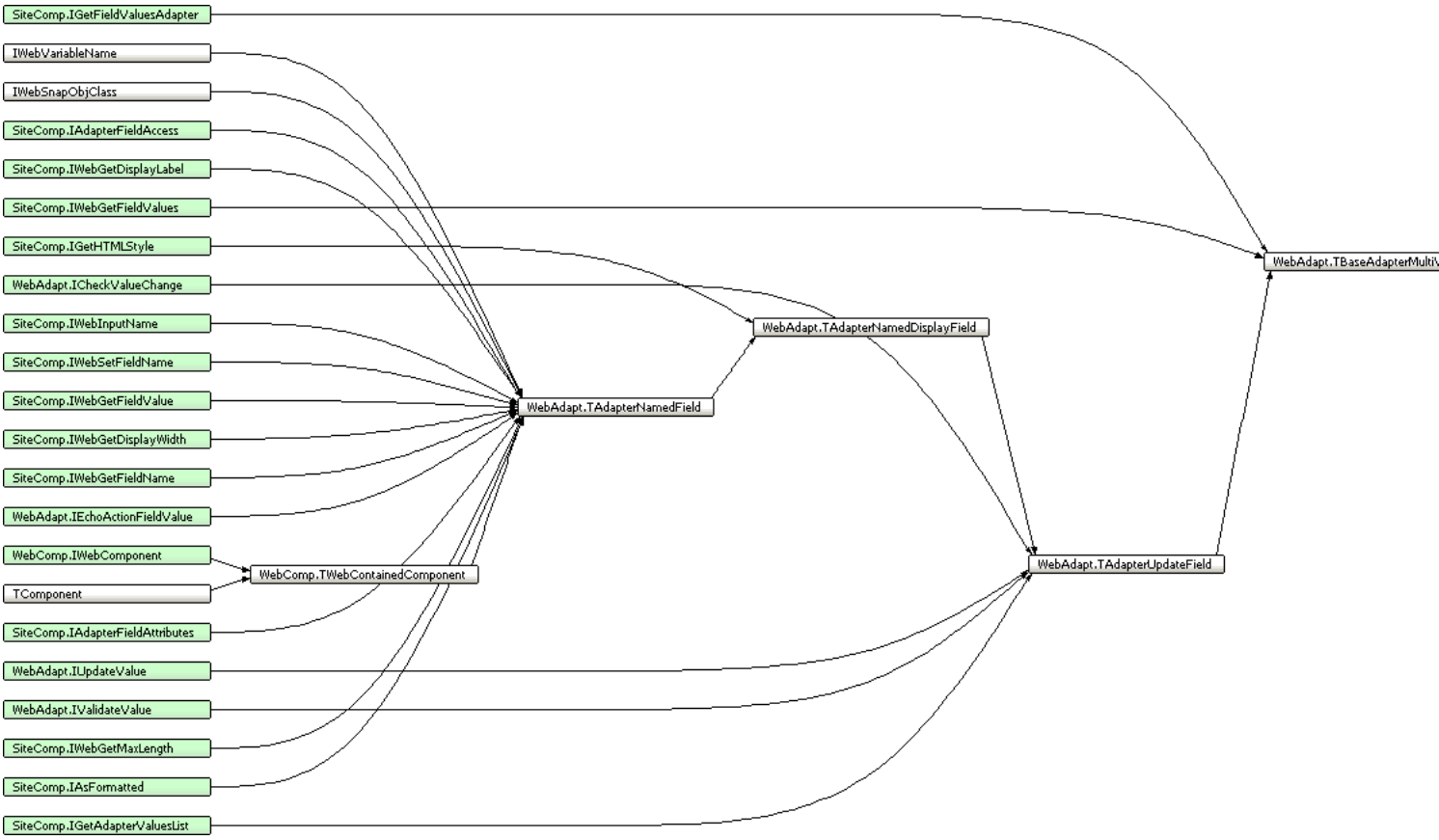
See Also

ImplUpdateValue (see page 5644)

54.1.52 TCustomAdapterMultiValueField Class

TCustomAdapterMultiValueField is a base class for adapter fields that have one or more values.

Class Hierarchy



Pascal

```
TCustomAdapterMultiValueField = class(TBaseAdapterMultiValueField);
```

C++

```
class TCustomAdapterMultiValueField : public TBaseAdapterMultiValueField;
```

File

WebAdapt

Description

Derive from TCustomAdapterMultiValueField when the adapter field can have multiple values. Typically, multi-value fields are represented in HTML with multiple selection select lists or check boxes.

See Also

TBaseAdapterMultiValueField (see page 5488)

TAdapterMultiValueField (see page 5332)

54.1.52.1 TCustomAdapterMultiValueField Members

The following tables list the members exposed by TCustomAdapterMultiValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckValueChange (see page 5424)	Checks the value(s) entered by the end-user against the value associated with this adapter field.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
	ConvertValue (see page 5425)	Retrieves a value entered by the end-user.
	ConvertValueCount (see page 5425)	Retrieves the number of values entered by the end-user.
	ConvertValues (see page 5425)	Retrieves a value entered by the end-user.
	DoOnValidateValue (see page 5651)	Validates values entered by the end-user.
	EndValueIterator (see page 5493)	Completes iteration.
	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapterValuesList (see page 5426)	Provides an interface to a component that can be used to generate a list of possible values for this adapter field.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetCurrentValue (see page 5494)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue.
	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
	GetDefaultFieldName (see page 5357)	Gets the default name used to reference this adapter field with server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayStyle (see page 5346)	Returns a string value to suggest an HTML display style for this adapter field.
	GetDisplayStyleType (see page 5651)	Suggests an HTML display type for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetFieldValuesAdapter (see page 5494)	Returns the adapter object that implements the Values property.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetInputStyle (see page 5347)	Returns a string value to suggest an HTML input style for this adapter field.
	GetInputStyleType (see page 5651)	Suggests an HTML input type for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetValueCount (see page 5652)	Returns the number of values associated with this adapter field.
	GetValues (see page 5652)	Returns one of the values associated with the adapter field.
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetViewMode (see page 5348)	Returns a string value to suggest a view mode for this adapter field.
	GetViewModeType (see page 5348)	Suggests a view mode for this adapter field.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' HasValue method.

	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplAsFormatted (see page 5427)	Returns a string representation of the adapter field's value that is appropriate for a read-only display.
V	ImplCheckValueChange (see page 5428)	Checks the value(s) entered by the end-user against the value(s) associated with this adapter field.
V	ImplGetCurrentValue (see page 5495)	Implements IWebGetFieldValues (see page 1092)' GetCurrentValue (see page 5494) method.
V	ImplGetDisplayLabel (see page 5348)	Implements interface access to the DisplayLabel (see page 5350) property.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayLabel (see page 5428)	Gets the preferred display label for this adapter field.
V	ImplGetDisplayWidth (see page 5349)	Implements interface access to the DisplayWidth (see page 5350) property.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetDisplayWidth (see page 5428)	Returns the preferred display width for the HTML element used to edit or display this adapter field's value.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetEchoActionFieldValue (see page 5429)	Gets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplGetFieldValuesAdapter (see page 5496)	Returns the adapter object that implements the Values property.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5349)	Implements interface access to the Required (see page 5351) property.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5363)	Gets the value of this adapter field as a Variant (see page 3421).
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVariableName (see page 5429)	Gets the name used to access this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasValue (see page 5653)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetEchoActionFieldValue (see page 5429)	Sets a flag that causes the adapter field to return the value(s) entered by the end-user.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	ImplSetFieldName (see page 5429)	Sets the name used to access this adapter field with server-side script.
V	ImplUpdateValue (see page 5653)	Updates the adapter field with value(s) entered by the end-user.
V	ImplValidateValue (see page 5430)	Validates values(s) entered by the end-user.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	IsMultiValueField (see page 5430)	Indicates whether this adapter field can support multiple values.
	NextValueIteration (see page 5496)	Implements IWebGetFieldValues (see page 1092)' NextValueIteration method.
V	Notification (see page 5431)	Clears references to a deleted component.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetDisplayLabel (see page 5349)	SetDisplayLabel sets the value of the DisplayLabel (see page 5350) property.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartValueIterator (see page 5497)	Implements IWebGetFieldValues (see page 1092)' StartValueIterator method.
	UpdateValue (see page 5431)	Updates the adapter field with value(s) entered by the end-user.
	ValidateValue (see page 5431)	Validates values(s) entered by the end-user.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnGetDisplayText (see page 5433)	Event handler called to retrieve a string representation of the adapter field's value.
	OnGetValueCount (see page 5654)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5655)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5655)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5656)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValue (see page 5433)	Event handler called to validate value(s) entered by the end-user.
	OnValidateValues (see page 5656)	Validates field values entered by the end-user.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.
	Create (see page 5346)	Creates an instance of TAdapterNamedDisplayField (see page 5343).
	Create (see page 5423)	Creates an instance of TAdapterUpdateField (see page 5419).
	Create (see page 5492)	Constructor for TBaseAdapterMultiValueField (see page 5488).
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 5492)	Destructor for TBaseAdapterMultiValueField (see page 5488).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndValueIterator (see page 1093)	Indicate the iteration is finished.
	GetAdapterValuesList (see page 1027)	This is GetAdapterValuesList, a member of class IGetAdapterValuesList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetCurrentValue (see page 1094)	Retrieves the current value as a variant.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetFieldValuesAdapter (see page 1031)	This is GetFieldValuesAdapter, a member of class IGetFieldValuesAdapter.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasValue (see page 1094)	Indicates whether the adapter field has a matching value.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	NextValueIteration (see page 1094)	Position the iteration pointer on the next value.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.

	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartValueIterator (see page 1095)	Reset (see page 3643) the iteration pointer to the first item.
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5350)	Suggests a display label for this adapter field.
	DisplayWidth (see page 5350)	Suggests a display width, in characters, for this adapter field.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
	ReadOnly (see page 5351)	Indicates whether this adapter field is read-only.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Required (see page 5351)	Indicates whether the end-user must enter a value for this adapter field.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
R	Value (see page 5432)	Retrieves the value associated with this adapter field.
R	ValueCount (see page 5653)	Returns the number of values associated with this adapter field.
R	Values (see page 5654)	Returns one of the values associated with the adapter field.
	ValuesList (see page 5432)	Associates a list of values with the adapter field.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.52.2 TCustomAdapterMultiValueField Methods

The methods of the TCustomAdapterMultiValueField class are listed here.

Protected Methods

	Name	Description
V	DoOnValidateValue (see page 5651)	Validates values entered by the end-user.
V	GetDisplayStyleType (see page 5651)	Suggests an HTML display type for this adapter field.
V	GetInputStyleType (see page 5651)	Suggests an HTML input type for this adapter field.
V	GetValueCount (see page 5652)	Returns the number of values associated with this adapter field.
V	GetValues (see page 5652)	Returns one of the values associated with the adapter field.
V	ImplHasValue (see page 5653)	Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.
V	ImplUpdateValue (see page 5653)	Updates the adapter field with value(s) entered by the end-user.

54.1.52.2.1 TCustomAdapterMultiValueField.DoOnValidateValue Method

Validates values entered by the end-user.

Pascal

```
procedure DoOnValidateValue(AValue: IActionFieldValue; var AHandled: Boolean); override;
```

C++

```
virtual __fastcall DoOnValidateValue(IActionFieldValue AValue, Boolean AHandled);
```

Description

DoOnValidateValue overrides TAdapterUpdateField (see page 5419)'s DoOnValidateValue method. If the OnValidateValues (see page 5656) event handler is not nil (Delphi) or NULL (C++), it is called it to validate values. If the Required (see page 5351) property is true, DoOnValidateValues also validates that 1 of more values have been entered by the end user. An exception will be raised to indicate a validation error.

See Also

DoOnValidateValue (see page 5426)

Required (see page 5351)

OnValidateValues (see page 5656)

54.1.52.2.2 TCustomAdapterMultiValueField.GetDisplayStyleType Method

Suggests an HTML display type for this adapter field.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterDisplayHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterDisplayHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns the display type for an adapter field. Here, it returns htmldList to indicate a list.

See Also

GetDisplayStyleType (see page 5347)

54.1.52.2.3 TCustomAdapterMultiValueField.GetInputStyleType Method

Suggests an HTML input type for this adapter field.

Pascal

```
function GetInputStyleType(const AAdapterMode: string): TAdapterInputHTMLElementType; override;
```

C++

```
virtual __fastcall TAdapterInputHTMLElementType GetInputStyleType(const UnicodeString
AAdapterMode);
```

Description

GetInputStyleType overrides TAdaperNamedDisplayField's GetInputStyleType method. If the ValuesList (see page 5432) property is not nil (Delphi) or NULL (C++), GetInputStyleType returns htmliCheckBox indicating that this adapter field should be

edited using a group of check boxes. If the ValuesList (see page 5432) property is nil (Delphi) or NULL (C++), then htmliTextInput is returned. htmliTextInput indicates that the adapter field value should be edited with a single line input.

See Also

GetInputStyleType (see page 5347)

ValuesList (see page 5432)

54.1.52.2.4 TCustomAdapterMultiValueField.GetValueCount Method

Returns the number of values associated with this adapter field.

Pascal

```
function GetValueCount: Integer; override;
```

C++

```
virtual __fastcall int GetValueCount();
```

Description

GetValueCount implements the ValueCount (see page 5653) property. If the EchoActionFieldValue (see page 5368) flag is true then the result is extracted from the HTTP request. Otherwise, the OnGetValueCount (see page 5654) event is called. GetValueCount returns zero if a value can't be found because OnGetValueCount (see page 5654) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

ValueCount (see page 5653)

Values (see page 5654)

EchoActionFieldValue (see page 5368)

54.1.52.2.5 TCustomAdapterMultiValueField.GetValues Method

Returns one of the values associated with the adapter field.

Pascal

```
function GetValues(I: Integer): Variant; override;
```

C++

```
virtual __fastcall Variant GetValues(int I);
```

Description

GetValues implements the Values (see page 5654) property. If the EchoActionFieldValue (see page 5368) flag is true then the result is extracted from the HTTP request. Otherwise, the OnGetValues (see page 5655) event is called. GetValues returns Unassigned if a value can't be found because OnGetValues (see page 5655) is nil (Delphi) or NULL (C++) or the HTTP request does not contain a value for this adapter field.

See Also

Values (see page 5654)

OnGetValues (see page 5655)

EchoActionFieldValue (see page 5368)

54.1.52.2.6 TCustomAdapterMultiValueField.ImplHasValue Method

Implements IWebGetFieldValues (see page 1092)' HasValue (see page 5495) method.

Pascal

```
function ImplHasValue(const Value: Variant): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHasValue(const Variant Value);
```

Description

ImplHasValue overrides TBaseAdapterMultiValueField (see page 5488)'s ImplHasValue method. If the OnHasValue (see page 5655) event is not nil (Delphi) or NULL (C++), it will be called for the result of ImplHasValue. Otherwise, ImplHasValue will loop through the values of the adapter field, looking for a matching value.

See Also

ImplHasValue (see page 5496)

HasValue (see page 1094)

54.1.52.2.7 TCustomAdapterMultiValueField.ImplUpdateValue Method

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
procedure ImplUpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer); override;
```

C++

```
virtual __fastcall ImplUpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

ImplUpdateValue overrides TAdapterUpdateField (see page 5419)'s ImplUpdateValue method. After checking that the end-user has rights to modify this adapter field, the OnUpdateValues (see page 5656) event is called.

See Also

OnUpdateValues (see page 5656)

ImplUpdateValue (see page 5430)

54.1.52.3 TCustomAdapterMultiValueField Properties

The properties of the TCustomAdapterMultiValueField class are listed here.

Public Properties

	Name	Description
	ValueCount (see page 5653)	Returns the number of values associated with this adapter field.
	Values (see page 5654)	Returns one of the values associated with the adapter field.

54.1.52.3.1 TCustomAdapterMultiValueField.ValueCount Property

Returns the number of values associated with this adapter field.

Pascal

```
property ValueCount: Integer;
```

C++

```
__property int ValueCount;
```

Description

ValueCount can be used with the Values (see page 5654) property to access all the values associated with this adapter field. ValueCount is implemented by GetValueCount (see page 5652).

See Also

GetValueCount (see page 5652)

Values (see page 5654)

54.1.52.3.2 TCustomAdapterMultiValueField.Values Property

Returns one of the values associated with the adapter field.

Pascal

```
property Values [I: Integer]: Variant;
```

C++

```
__property Variant Values[int I];
```

Description

Values can be used with the ValueCount (see page 5653) property to access all the values associated with this adapter field. Values is implemented by GetValues (see page 5652).

See Also

GetValues (see page 5652)

ValueCount (see page 5653)

54.1.52.4 TCustomAdapterMultiValueField Events

The events of the TCustomAdapterMultiValueField class are listed here.

Public Events

	Name	Description
	OnGetValueCount (see page 5654)	Responsible for returning the count of values associated with the adapter field.
	OnGetValues (see page 5655)	Responsible for returning one of the values associated with the adapter field.
	OnHasValue (see page 5655)	Indicates whether a particular value is associated with this adapter field.
	OnUpdateValues (see page 5656)	Updates the adapter field with value(s) entered by the end-user.
	OnValidateValues (see page 5656)	Validates field values entered by the end-user.

54.1.52.4.1 TCustomAdapterMultiValueField.OnGetValueCount Event

Responsible for returning the count of values associated with the adapter field.

Pascal

```
property OnGetValueCount: TAdapterFieldGetValueCountEvent;
```

C++

```
__property TAdapterFieldGetValueCountEvent OnGetValueCount;
```

Description

Implement this event handler to provide the count of values associated with an adapter field. Also implement OnGetValues (see page 5655) to provide the individual values.

See Also

GetValueCount (see page 5652)

OnGetValues (see page 5655)

54.1.52.4.2 TCustomAdapterMultiValueField.OnGetValues Event

Responsible for returning one of the values associated with the adapter field.

Pascal

```
property OnGetValues: TAdapterFieldGetMultiValueEvent;
```

C++

```
__property TAdapterFieldGetMultiValueEvent OnGetValues;
```

Description

Implement this event handler to provide the values associated with an adapter field. The Index parameter is zero-based. Also implement OnGetValueCount (see page 5654) to provide the count of values associated with this adapter field.

See Also

GetValues (see page 5652)

OnGetValueCount (see page 5654)

54.1.52.4.3 TCustomAdapterMultiValueField.OnHasValue Event

Indicates whether a particular value is associated with this adapter field.

Pascal

```
property OnHasValue: TAdapterFieldHasValueEvent;
```

C++

```
__property TAdapterFieldHasValueEvent OnHasValue;
```

Description

ImplHasValue (see page 5653) calls this event handler. Implement this event handler to lookup a matching value. Implementation of this event handler is optional. If OnHasValue is nil, ImplHasValue (see page 5653) will search for a matching value the Values (see page 5654) and ValueCount (see page 5653) properties.

See Also

ImplHasValue (see page 5653)

ValueCount (see page 5653)

Values (see page 5654)

54.1.52.4.4 TCustomAdapterMultiValueField.OnUpdateValues Event

Updates the adapter field with value(s) entered by the end-user.

Pascal

```
property OnUpdateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnUpdateValues;
```

Description

Implement this event handler to process values entered by the end-user. Raise an exception if an error occurs.

See Also

OnValidateValues (see page 5656)

ImplUpdateValue (see page 5653)

54.1.52.4.5 TCustomAdapterMultiValueField.OnValidateValues Event

Validates field values entered by the end-user.

Pascal

```
property OnValidateValues: TAdapterFieldStringsEvent;
```

C++

```
__property TAdapterFieldStringsEvent OnValidateValues;
```

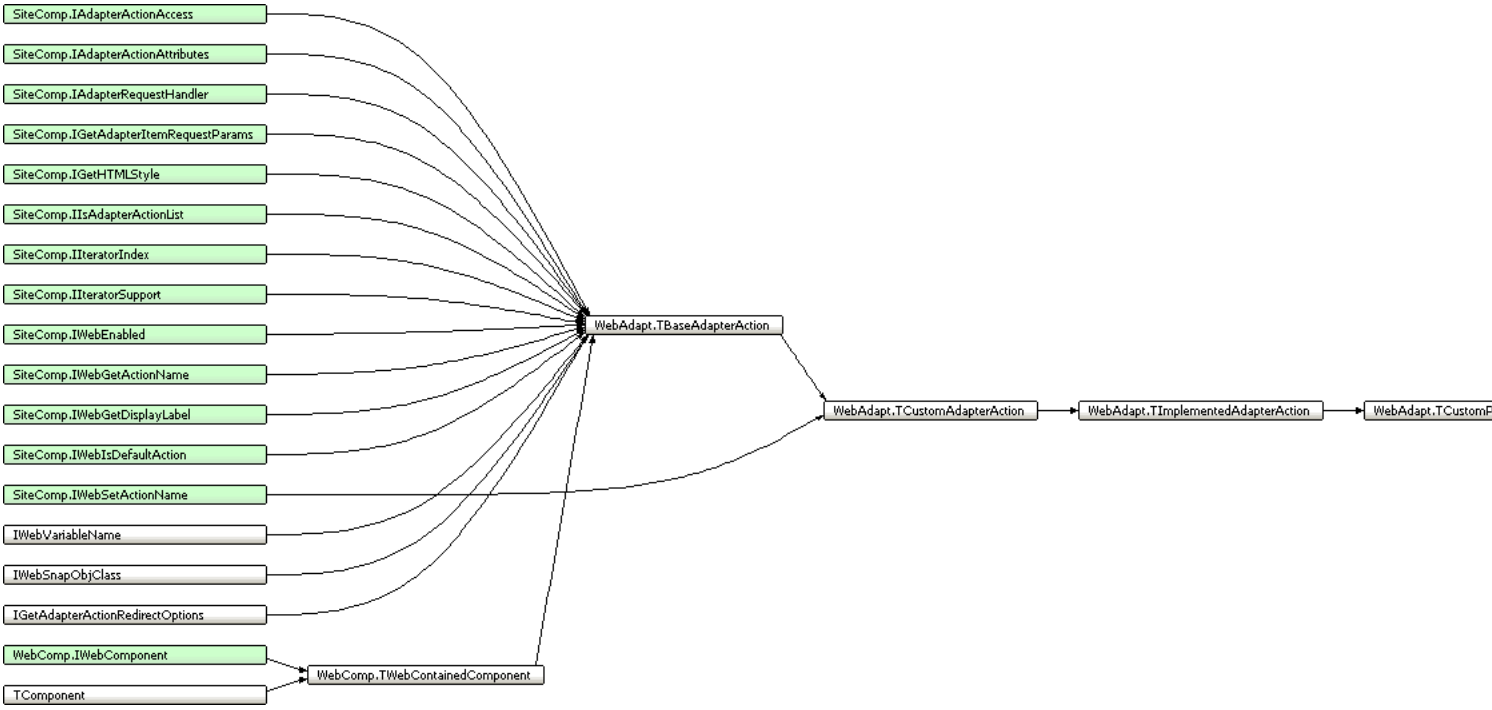
Description

Implement this event handler to validate values entered by the end-user. Raise an exception if an error occurs.

54.1.53 TCustomAdapterPageAction Class

TCustomPagedAdapterAction (see page 5722) is a base class for adapter actions that execute the next or previous page commands.

Class Hierarchy



Pascal

```
TCustomAdapterPageAction = class(TCustomPagedAdapterAction);
```

C++

```
class TCustomAdapterPageAction : public TCustomPagedAdapterAction;
```

File

WebAdapt

Description

TCustomPagedAdapterAction (see page 5722) provide methods that are used by TAdapterNextPageAction (see page 5370) and TAdapterPrevPageAction (see page 5396).

See Also

TDefaultFieldsPagedAdapter (see page 5747)

TCustomPagedAdapterAction (see page 5722)

TAdapterNextPageAction (see page 5370)

TAdapterPrevPageAction (see page 5396)

54.1.53.1 TCustomAdapterPageAction Members

The following tables list the members exposed by TCustomAdapterPageAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5727)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5661)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
V	GetNewPage (see page 5661)	Indicates the page to display when this action is executed.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5662)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5662)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.

	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5663)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5663)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5474)	Provides access to the parent adapter component.
	Adapter (see page 5727)	Returns the parent adapter of this adapter action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.

	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.

	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.53.2 TCustomAdapterPageAction Methods

The methods of the TCustomAdapterPageAction class are listed here.

Protected Methods

	Name	Description
	GetDisplayStyleType (see page 5661)	Suggests an HTML display type for this adapter action.
	GetNewPage (see page 5661)	Indicates the page to display when this action is executed.
	ImplExecuteActionRequest (see page 5662)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetAdapterItemRequestParams (see page 5662)	Provides the name/value pairs needed to execute this adapter action.
	ImplIsDefaultAction (see page 5663)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplWebEnabled (see page 5663)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

54.1.53.2.1 TCustomAdapterPageAction.GetDisplayStyleType Method

Suggests an HTML display type for this adapter action.

Pascal

```
function GetDisplayStyleType(const AAdapterMode: string): TAdapterActionHTMLElementType;
override;
```

C++

```
virtual __fastcall TAdapterActionHTMLElementType GetDisplayStyleType(const UnicodeString
AAdapterMode);
```

Description

GetDisplayStyleType returns htmlaAnchor, indicating that next page and previous page commands should be displayed using a hyperlink.

See Also

GetDisplayStyle (see page 5457)

54.1.53.2.2 TCustomAdapterPageAction.GetNewPage Method

Indicates the page to display when this action is executed.

Pascal

```
function GetNewPage(var AValue: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetNewPage(int AValue);
```

Description

This is a placeholder. Descendant classes must override this method. true is returned if the action can move to a new page. The AValue parameter identifies the new page. GetNewPage is called by ImplExecuteActionRequest (see page 5662).

See Also

GetNewPage (see page 5375)

GetNewPage (see page 5402)

ImplExecuteActionRequest (see page 5662)

54.1.53.2.3 TCustomAdapterPageAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest is called by ExecuteActionRequest (see page 5455). ImplExecuteActionRequest calls ExtractRequestParams followed by GetNewPage (see page 5661). If GetNewPage (see page 5661) returns true then the parent adapter's current page is changed.

See Also

ExecuteActionRequest (see page 5455)

ExtractRequestParams (see page 5754)

GetNewPage (see page 5661)

CurrentPage (see page 5760)

54.1.53.2.4 TCustomAdapterPageAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams is called by GetAdapterItemRequestParams (see page 5456). ImplGetAdapterItemRequestParams calls the parent adapter's EncodeActionParams method to add the current page as a parameter. Then the inherited ImplGetAdapterItemRequestParams method is called so that any event handlers that may be defined will be called.

See Also

ImplGetAdapterItemRequestParams (see page 5465)

GetAdapterItemRequestParams (see page 5456)

EncodeActionParams (see page 5754)

OnGetActionParams (see page 5579)

OnGetParams (see page 5478)

54.1.53.2.5 TCustomAdapterPageAction.ImplIsDefaultAction Method

Indicates whether this adapter action should be added, by default, to the parent adapter.

Pascal

```
function ImplIsDefaultAction(ADisplay: TObject): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplIsDefaultAction(TObject * ADisplay);
```

Description

ImplIsDefaultAction is called by IsDefaultAction (see page 5471). ImplIsDefaultAction returns true if the display characteristics identified by ADisplay indicate the a previous or next page command is appropriate.

See Also

IsDefaultAction (see page 5471)

ImplIsDefaultAction (see page 5622)

54.1.53.2.6 TCustomAdapterPageAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled is called by WebEnabled (see page 5473). ImplWebEnabled returns the result of GetNewPage (see page 5661).

See Also

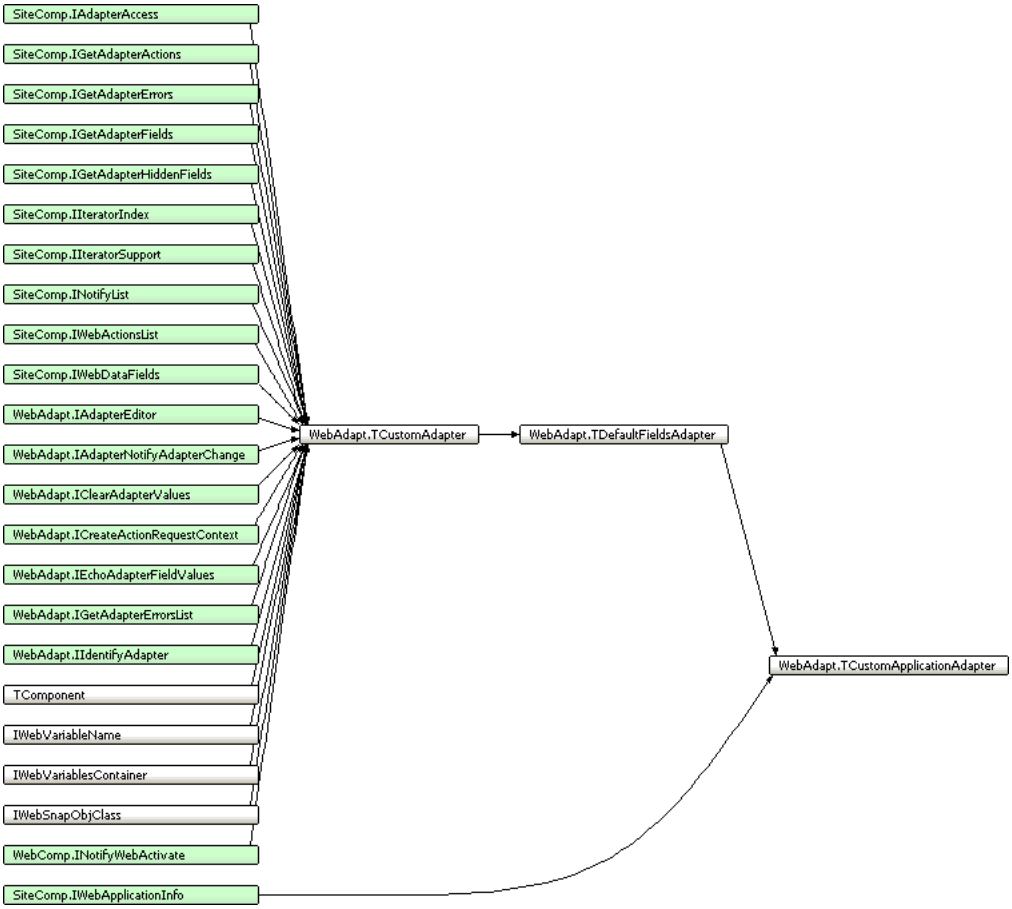
WebEnabled (see page 5473)

GetNewPage (see page 5661)

54.1.54 TCustomApplicationAdapter Class

TCustomApplicationAdapter is a base class for components that implement the server-side script Application object.

Class Hierarchy



Pascal

```
TCustomApplicationAdapter = class(TDefaultFieldsAdapter, IWebApplicationInfo);
```

C++

```
class TCustomApplicationAdapter : public TDefaultFieldsAdapter, public IWebApplicationInfo;
```

File

WebAdapt

Description

TCustomApplicationAdapter implements IWebApplicationInfo (see page 1078). This interface is used to support server-side scripting. TCustomApplicationAdapter automatically adds an adapter field that provides the application title. Additional fields and actions may be added to this adapter. They are accessed as named properties of the server-side script Application object.

54.1.54.1 TCustomApplicationAdapter Members

The following tables list the members exposed by TCustomApplicationAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetApplicationAdapter (see page 5669)	Implements IWebApplicationInfo (see page 1078)'s GetApplicationAdapter method.
	GetApplicationTitle (see page 5669)	Implements IWebApplicationInfo (see page 1078)'s GetApplicationTitle method.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.

	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5669)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5669)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5670)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5670)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5670)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
V	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
V	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
V	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
V	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
V	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
V	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
V	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
V	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.

	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.

	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
R	ApplicationAdapter (see page 1080)	Provides access to the adapter that implements the fields and actions of the Application variable in server-side scripts.
	ApplicationTitle (see page 5671)	Defines the title of the application.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.54.2 TCustomApplicationAdapter Methods

The methods of the TCustomApplicationAdapter class are listed here.

Protected Methods

	Name	Description
	GetApplicationAdapter (see page 5669)	Implements IWebApplicationInfo (see page 1078)'s GetApplicationAdapter method.
	GetApplicationTitle (see page 5669)	Implements IWebApplicationInfo (see page 1078)'s GetApplicationTitle method.
V	GetWebSnapObjClass (see page 5669)	Returns the class type for this adapter field.
V	ImplCanAddActionClass (see page 5669)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5670)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplGetActionsList (see page 5670)	Provides a list of action types that can be added to this adapter.
V	ImplGetFieldsList (see page 5670)	Provides a list of field types that can be added to this adapter.

54.1.54.2.1 TCustomApplicationAdapter.GetApplicationAdapter Method

Implements IWebApplicationInfo (see page 1078)'s GetApplicationAdapter method.

Pascal

```
function GetApplicationAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetApplicationAdapter();
```

Description

GetApplicationAdapter returns Self.

See Also

GetApplicationAdapter (see page 1079)

54.1.54.2.2 TCustomApplicationAdapter.GetApplicationTitle Method

Implements IWebApplicationInfo (see page 1078)'s GetApplicationTitle method.

Pascal

```
function GetApplicationTitle: string;
```

C++

```
__fastcall UnicodeString GetApplicationTitle();
```

Description

GetApplicationTitle returns the application title that can be accessed with server-side script as the Application.Title property.

See Also

GetApplicationTitle (see page 1080)

54.1.54.2.3 TCustomApplicationAdapter.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; override;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

54.1.54.2.4 TCustomApplicationAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass allows actions classes that derive from TCustomApplicationAdapterAction (see page 5671) in addition to those allows by TCustomAdapter (see page 5531)'s ImplCanAddActionsClass method.

See Also

ImplCanAddActionClass (see page 5553)

TCustomApplicationAdapterAction (see page 5671)

54.1.54.2.5 TCustomApplicationAdapter.ImplCanAddFieldClass Method

Indicates whether a type of adapter field can be added to this adapter.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass allows field classes that derive from TApplicationAdapterField in addition to those allowed by TCustomAdapter (see page 5531)'s ImplCanAddFieldClass method.

See Also

ImplCanAddFieldClass (see page 5553)

54.1.54.2.6 TCustomApplicationAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList clears the list. There are no default actions.

54.1.54.2.7 TCustomApplicationAdapter.ImplGetFieldsList Method

Provides a list of field types that can be added to this adapter.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetActionsList (see page 5670) lists the title field.

See Also

TAdapterApplicationTitleField (see page 5218)

54.1.54.3 TCustomApplicationAdapter Properties

The properties of the TCustomApplicationAdapter class are listed here.

Public Properties

	Name	Description
	ApplicationTitle (see page 5671)	Defines the title of the application.

54.1.54.3.1 TCustomApplicationAdapter.ApplicationTitle Property

Defines the title of the application.

Pascal

```
property ApplicationTitle: string;
```

C++

```
__property UnicodeString ApplicationTitle;
```

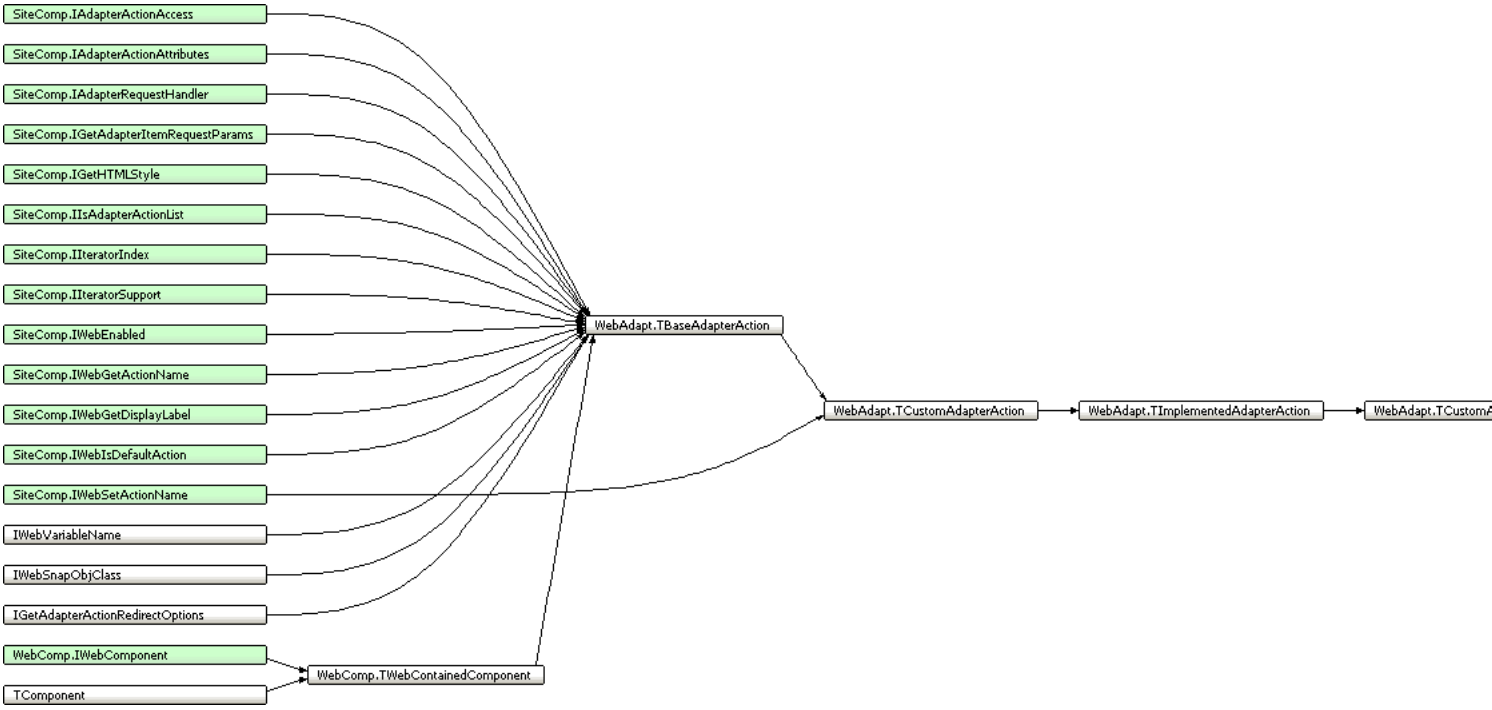
Description

Set this property to identify an application title that can be accessed with server-side script as the Application.Title property.

54.1.55 TCustomApplicationAdapterAction Class

TCustomApplicationAdapterAction is a base class for adapter actions that work with descendants of TCustomApplicationAdapter (see page 5663).

Class Hierarchy



Pascal

```
TCustomApplicationAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomApplicationAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomApplicationAdapterAction has two purposes. First, it provides a property that is used by descendants to access the parent adapter. Second, it defines a base class that TCustomApplicationAdapter (see page 5663) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 5818)

TCustomAdapterAction (see page 5579)

TAdapter (see page 5190)

TApplicationAdapter (see page 5439)

54.1.55.1 TCustomApplicationAdapterAction Members

The following tables list the members exposed by TCustomApplicationAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.

	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5676)	Returns the parent adapter of this adapter action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.

	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.

	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.55.2 TCustomApplicationAdapterAction Properties

The properties of the TCustomApplicationAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5676)	Returns the parent adapter of this adapter action.

54.1.55.2.1 TCustomApplicationAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TCustomApplicationAdapter;
```

C++

```
__property TCustomApplicationAdapter Adapter;
```

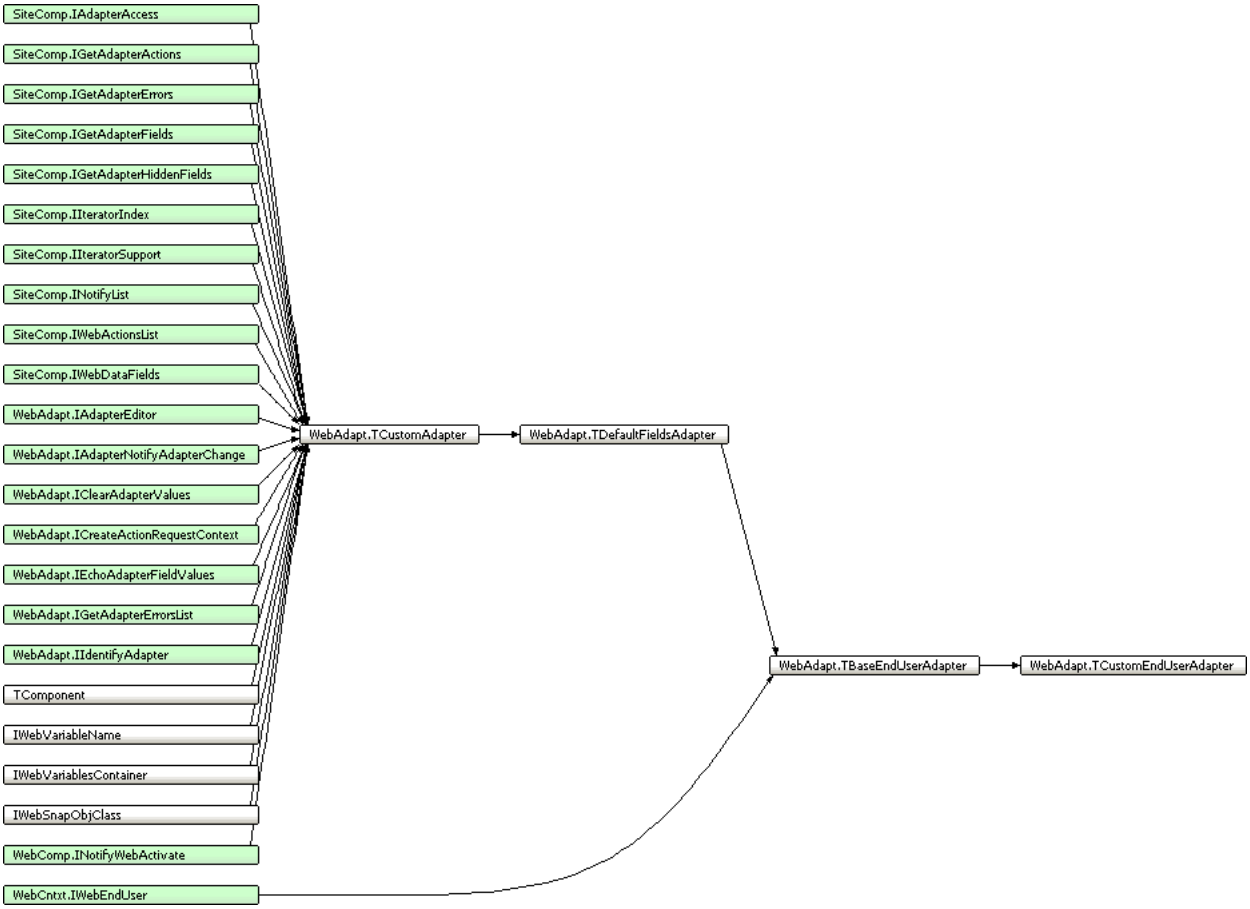
Description

Descendant classes use this property to access the parent adapter.

54.1.56 TCustomEndUserAdapter Class

TCustomEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy



Pascal

```
TCustomEndUserAdapter = class(TBaseEndUserAdapter);
```

C++

```
class TCustomEndUserAdapter : public TBaseEndUserAdapter;
```

File

WebAdapt

Description

TCustomEndUserAdapter is the base class for end user adapters that use event handlers to implement the default adapter fields and actions. It introduces events to provide values for the default end user fields (DisplayName and LoggedIn) and to implement the default end user actions (logging in and logging out).

As with any end user adapter, TCustomEndUserAdapter has a LoginPage (see page 5516) property that lets you specify the page name of the Web page that handles the login process. As an end user adapter, it also handles validation of end user access. When the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access that page.

Do not create instances of TCustomEndUserAdapter. Instead, use a descendant such as the TEndUserAdapter (see page 5764) component, which publishes the events whose handlers you must supply to make this adapter work.

See Also

TEndUserAdapter (see page 5764)

54.1.56.1 TCustomEndUserAdapter Members

The following tables list the members exposed by TCustomEndUserAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5504)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5504)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5505)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5505)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5683)	Generates an OnLogin (see page 5686) event.
V	ExecuteLoginForm (see page 5506)	Creates the HTML login page.
V	ExecuteLogout (see page 5683)	Generates an OnLogout (see page 5687) event.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.

	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5506)	Retrieves the display name string.
V	GetDisplayNameValue (see page 5683)	Generates an OnGetDisplayName (see page 5685) event.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5507)	Gets the end-user adapter.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5507)	Determines if the user is logged in
V	GetLoggedInValue (see page 5684)	Generates an OnIsLoggedIn (see page 5686) event.
	GetLoginFormAction (see page 5508)	Gets the action component for the login form.
	GetLogoutAction (see page 5508)	Gets the action component for the logout form.
V	GetUserID (see page 5684)	Generates an OnGetUserID (see page 5685) event.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasExecuteRights (see page 5509)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5497).
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasModifyRights (see page 5509)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5497).
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5510)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5497).
V	ImplCanAddActionClass (see page 5510)	Determines if the given action class can be added.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5510)	Determines if the given action class can be added.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5511)	Lists supported end-user adapter actions.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetDisplayName (see page 5511)	Gets the display name for this TBaseEndUserAdapter (see page 5497).
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetEndUserAdapter (see page 5511)	Returns this TBaseEndUserAdapter (see page 5497).
V	ImplGetFieldsList (see page 5512)	Obtains the list of all of the fields for the end-user adapter.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetLoggedIn (see page 5512)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5512)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5497).
V	ImplGetLogoutAction (see page 5513)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5497).

	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5513)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5513)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplHasViewRights (see page 5514)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5514)	Processes the login for the given user.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5514)	Generate the HTML login page.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5515)	Processes the server-side login for the given user.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5515)	Generates the login page.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5515)	Gets the user list.

	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 5685)	Occurs when the adapter needs to provide the name of the current end user.
	OnGetUserID (see page 5685)	Occurs when the adapter needs to provide the ID of the current end user.
	OnHasRights (see page 5516)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 5686)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.
	OnLogin (see page 5686)	Logs in the current end user.
	OnLogout (see page 5687)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 5516)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5989)	Returns the value of the DisplayName (see page 5994) property.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5990)	Returns the value of the EndUserAdapter (see page 5994) property.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.

	GetLoggedIn (see page 5990)	Returns the value of the LoggedIn (see page 5994) property.
	GetLoginFormAction (see page 5990)	Returns the value of the LoginFormAction (see page 5995) property.
	GetLogoutAction (see page 5991)	Returns the value of the LogoutAction (see page 5996) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5991)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5992)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5992)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5993)	Logs in a specified user.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5993)	Generates the Web page for logging in users.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
R	DisplayName (see page 5994)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5994)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	LoggedIn (see page 5994)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5995)	Indicates the action component for the login form.
	LoginPage (see page 5516)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 5996)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.56.2 TCustomEndUserAdapter Methods

The methods of the TCustomEndUserAdapter class are listed here.

Protected Methods

	Name	Description
	ExecuteLogin (see page 5683)	Generates an OnLogin (see page 5686) event.
	ExecuteLogout (see page 5683)	Generates an OnLogout (see page 5687) event.
	GetDisplayNameValue (see page 5683)	Generates an OnGetDisplayName (see page 5685) event.
	GetLoggedInValue (see page 5684)	Generates an OnIsLoggedIn (see page 5686) event.
	GetUserID (see page 5684)	Generates an OnGetUserID (see page 5685) event.

54.1.56.2.1 TCustomEndUserAdapter.ExecuteLogin Method

Generates an OnLogin (see page 5686) event.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); override;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

ExecuteLogin is the implementation of the LogIn (see page 5515) action. When a server-side script executes the login action, it generates a call to ExecuteLogin to start the login process. TCustomEndUserAdapter (see page 5676) responds by generating an OnLogin (see page 5686) event, where you can take whatever action is necessary to log in the specified user.

AUserID is the user ID of the user to log in.

See Also

OnLogin (see page 5686)

54.1.56.2.2 TCustomEndUserAdapter.ExecuteLogout Method

Generates an OnLogout (see page 5687) event.

Pascal

```
procedure ExecuteLogout; override;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

ExecuteLogout is the implementation of the Logout action. When a server-side script executes the logout action, it generates a call to ExecuteLogout. TCustomEndUserAdapter (see page 5676) responds by generating an OnLogout (see page 5687) event, where you can take whatever action is necessary to log the specified user out.

See Also

OnLogout (see page 5687)

54.1.56.2.3 TCustomEndUserAdapter.GetDisplayNameValue Method

Generates an OnGetDisplayName (see page 5685) event.

Pascal

```
function GetDisplayNameValue: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayNameValue();
```

Description

GetDisplayNameValue is the implementation of the DisplayName adapter field. When a server-side script references the display name of the current user, it generates a call to GetDisplayNameValue. TCustomEndUserAdapter (see page 5676) responds by generating an OnGetDisplayName (see page 5685) event, where you can take supply the name of the current user.

GetDisplayNameValue returns the display name that the OnGetDisplayName (see page 5685) event handler provides. If you do not supply an OnGetDisplayName (see page 5685) event handler, GetDisplayNameValue returns an empty string.

See Also

OnGetDisplayName (see page 5685)

54.1.56.2.4 TCustomEndUserAdapter.GetLoggedInValue Method

Generates an OnIsLoggedIn (see page 5686) event.

Pascal

```
function GetLoggedInValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

GetLoggedInValue is the implementation of the LoggedIn adapter field. When a server-side script references the LoggedIn field for the current user, it generates a call to GetLoggedInValue. TCustomEndUserAdapter (see page 5676) responds by generating an OnIsLoggedIn (see page 5686) event, where you can indicate whether the current user has logged in.

GetLoggedInValue returns the value that the OnGetLoggedIn event handler provides. If you do not supply an OnGetLoggedIn event handler, GetLoggedInValue returns false.

See Also

OnIsLoggedIn (see page 5686)

54.1.56.2.5 TCustomEndUserAdapter.GetUserID Method

Generates an OnGetUserID (see page 5685) event.

Pascal

```
function GetUserID: Variant; override;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

GetUserIDValue is the implementation of the UserID adapter field. When a server-side script references the user ID of the current user, it generates a call to GetUserIDValue. TCustomEndUserAdapter (see page 5676) responds by generating an OnGetUserID (see page 5685) event, where you can take supply the ID of the current user.

GetUserIDValue returns the user ID that the OnGetUserID (see page 5685) event handler provides. If you do not supply an OnGetUserID (see page 5685) event handler, GetUserIDValue returns an unassigned Variant (see page 3421).

See Also

OnGetUserID (see page 5685)

54.1.56.3 TCustomEndUserAdapter Events

The events of the TCustomEndUserAdapter class are listed here.

Public Events

	Name	Description
	OnGetDisplayName (see page 5685)	Occurs when the adapter needs to provide the name of the current end user.
	OnGetUserID (see page 5685)	Occurs when the adapter needs to provide the ID of the current end user.
	OnIsLoggedIn (see page 5686)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 5686)	Logs in the current end user.
	OnLogout (see page 5687)	Occurs when the current user logs out.

54.1.56.3.1 TCustomEndUserAdapter.OnGetDisplayName Event

Occurs when the adapter needs to provide the name of the current end user.

Pascal

```
property OnGetDisplayName: TEndUserGetNameEvent;
```

C++

```
__property TEndUserGetNameEvent OnGetDisplayName;
```

Description

Write (see page 3707) an OnGetDisplayName event handler to provide the value of the DisplayName field. When a server-side script (or a caller of the IWebEndUser (see page 5988) interface) requests the display name for the current end user, TCustomEndUserAdapter (see page 5676) generates an OnGetDisplayName event to retrieve that value.

Note: If you do not supply an OnGetDisplayName event handler, TCustomEndUserAdapter (see page 5676) always returns an empty string as the display name for the current end user.

See Also

DisplayName (see page 5994)

EndUser (see page 5973)

WebContext (see page 6003)

54.1.56.3.2 TCustomEndUserAdapter.OnGetUserID Event

Occurs when the adapter needs to provide the ID of the current end user.

Pascal

```
property OnGetUserID: TEndUserGetUserIDEvent;
```

C++

```
__property TEndUserGetUserIDEvent OnGetUserID;
```

Description

Write (see page 3707) an OnGetDisplayName (see page 5685) event handler to provide the user ID for the current user.

When a server-side script requests ID for the current end user, TCustomEndUserAdapter (see page 5676) generates an OnGetUserID event to retrieve that value.

Note: If you do not supply an OnGetUserID event handler, TCustomEndUserAdapter (see page 5676) always returns an unassigned Variant (see page 3421) as the user ID.

See Also

GetDisplayNameValue (see page 5683)

TWebUserList (see page 6796)

OnLogin (see page 5686)

54.1.56.3.3 TCustomEndUserAdapter.OnIsLoggedIn Event

Occurs when the adapter needs to indicate whether the current end user has logged in.

Pascal

```
property OnIsLoggedIn: TEndUserIsLoggedInEvent;
```

C++

```
__property TEndUserIsLoggedInEvent OnIsLoggedIn;
```

Description

Write (see page 3707) an OnIsLoggedIn event handler to provide the value of the LoggedIn field. When a server-side script (or a caller of the IWebEndUser (see page 5988) interface) inquires whether the current user has logged in, TCustomEndUserAdapter (see page 5676) generates an OnIsLoggedIn event to retrieve that information.

Note: If you do not supply an OnIsLoggedIn event handler, TCustomEndUserAdapter (see page 5676) always returns false, indicating that the current end user has not logged in.

See Also

LoggedIn (see page 5994)

EndUser (see page 5973)

WebContext (see page 6003)

OnLogin (see page 5686)

OnLogout (see page 5687)

54.1.56.3.4 TCustomEndUserAdapter.OnLogin Event

Logs in the current end user.

Pascal

```
property OnLogin: TEndUserUserIDEvent;
```

C++

```
__property TEndUserUserIDEvent OnLogin;
```

Description

Write (see page 3707) an OnLogin event handler to allow the TCustomEndUserAdapter (see page 5676) object to log users in. When a server-side script (or a caller of the IWebEndUser (see page 5988) interface) attempts to log in a user, TCustomEndUserAdapter (see page 5676) generates an OnLogin event to perform whatever steps are involved in the login process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter (see page 5676) does nothing to log users in.

See Also

LogIn (see page 5993)

EndUser (see page 5973)

WebContext (see page 6003)

OnIsLoggedIn (see page 5686)

OnLogout (see page 5687)

TWebUserList (see page 6796)

OnGetUserID (see page 5685)

54.1.56.3.5 TCustomEndUserAdapter.OnLogout Event

Occurs when the current user logs out.

Pascal

```
property OnLogout: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLogout;
```

Description

Write (see page 3707) an OnLogout event handler to allow the TCustomEndUserAdapter (see page 5676) object to log users out. When a server-side script attempts to log a user out, TCustomEndUserAdapter (see page 5676) generates an OnLogout event to perform whatever steps are involved in the logout process.

Note: If you do not supply an OnLogin (see page 5686) event handler, TCustomEndUserAdapter (see page 5676) does nothing to log users out.

See Also

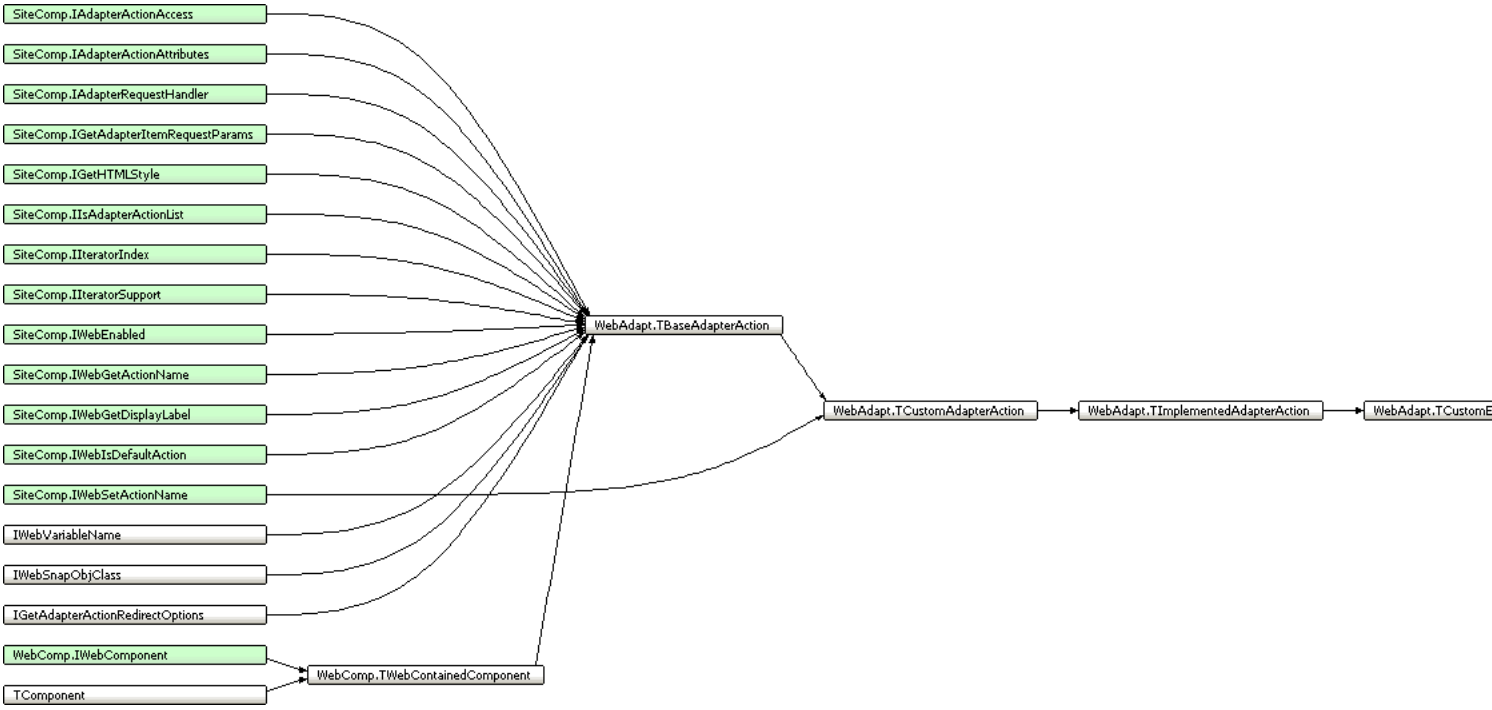
OnIsLoggedIn (see page 5686)

OnLogin (see page 5686)

54.1.57 TCustomEndUserAdapterAction Class

TCustomEndUserAdapterAction is a base class for adapter actions that work with descendants of TCustomEndUserAdapter (see page 5676).

Class Hierarchy



Pascal

```
TCustomEndUserAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomEndUserAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomEndUserAdapterAction has two purposes. First, it provides a property that descendants use to access the parent adapter. Second, it defines a base class that TCustomEndUserAdapter (see page 5676) checks for when determining whether an adapter action can be added as a child.

54.1.57.1 TCustomEndUserAdapterAction Members

The following tables list the members exposed by TCustomEndUserAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.

	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.

	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5692)	Returns the parent adapter of this adapter action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).

	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.57.2 TCustomEndUserAdapterAction Properties

The properties of the TCustomEndUserAdapterAction class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 5692)	Returns the parent adapter of this adapter action.

54.1.57.2.1 TCustomEndUserAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TBaseEndUserAdapter;
```

C++

```
__property TBaseEndUserAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

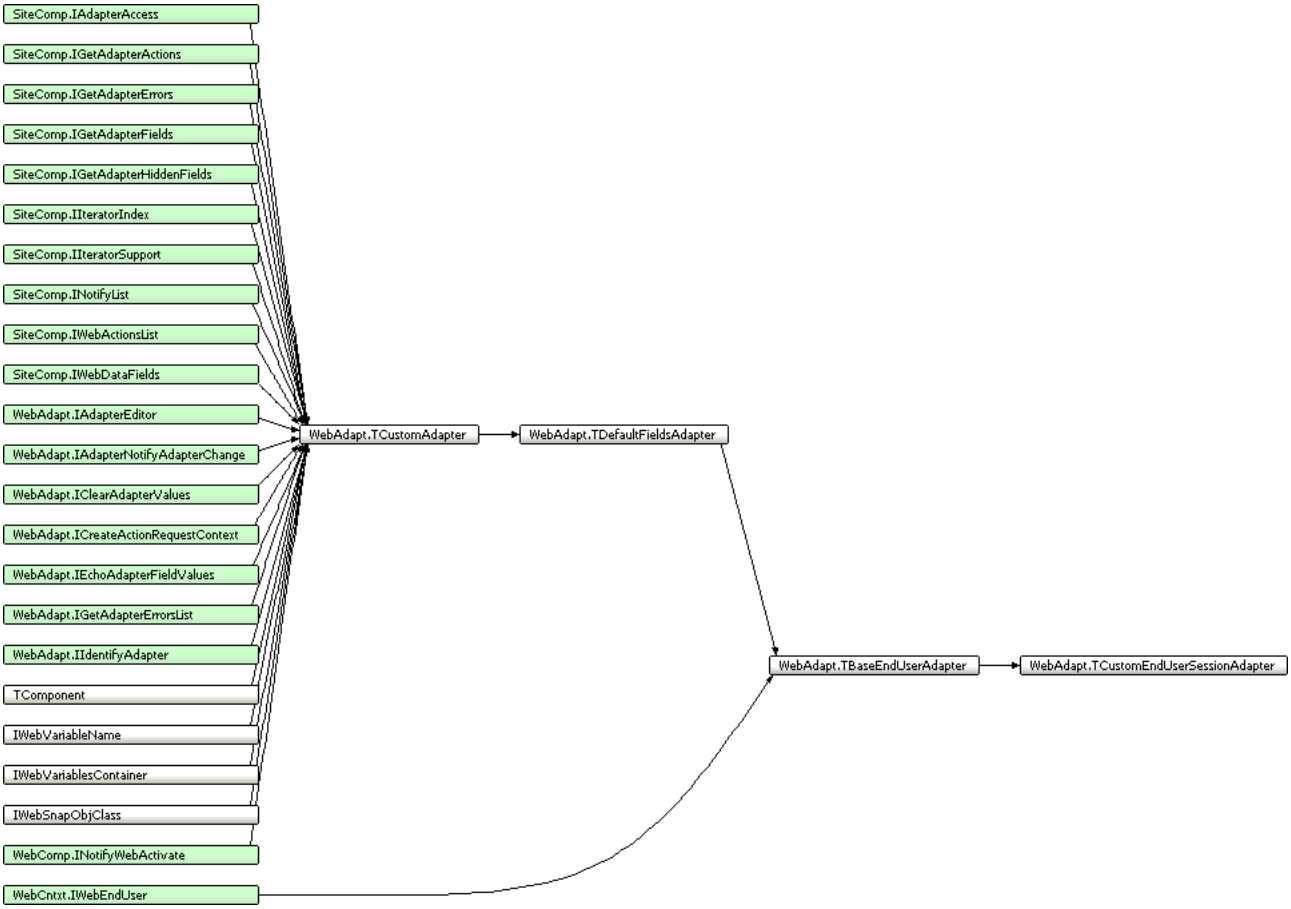
See Also

TCustomEndUserAdapter (see page 5676)

54.1.58 TCustomEndUserSessionAdapter Class

TCustomEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy



Pascal

```
TCustomEndUserSessionAdapter = class(TBaseEndUserAdapter);
```

C++

```
class TCustomEndUserSessionAdapter : public TBaseEndUserAdapter;
```

File

WebAdapt

Description

TCustomEndUserSessionAdapter is the base class for end user adapters that use a TSessionsService object to implement the default adapter fields and actions. It handles login and logout actions on behalf of the end user, and uses the TSessionsService component to store information about the current end user. The WebSnap application must contain a TSessionsService component (or equivalent) in order for the TCustomEndUserSessionAdapter to work properly.

As with any end user adapter, TCustomEndUserSessionAdapter has a LoginPage (see page 5516) property that lets you specify the page name of the Web page that handles the login process. As an end user adapter, it also handles validation of end user access. When the end user attempts to access a page, the end-user adapter is called to determine whether the end user has rights to access that page.

See Also

TEndUserSessionAdapter (see page 5792)

TSessionsService (see page 6756)

54.1.58.1 TCustomEndUserSessionAdapter Members

The following tables list the members exposed by TCustomEndUserSessionAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5504)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5504)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5505)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5505)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5699)	Logs in an end user.
V	ExecuteLoginForm (see page 5506)	Creates the HTML login page.
V	ExecuteLogout (see page 5699)	Logs out the current user.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5506)	Retrieves the display name string.
V	GetDisplayNameValue (see page 5699)	Returns the name of the current end user.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5507)	Gets the end-user adapter.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5507)	Determines if the user is logged in
V	GetLoggedInValue (see page 5700)	Indicates whether the current end user has logged in.
	GetLoginFormAction (see page 5508)	Gets the action component for the login form.
	GetLogoutAction (see page 5508)	Gets the action component for the logout form.
V	GetUserID (see page 5700)	Returns the user ID of the current user.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasExecuteRights (see page 5509)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5497).
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasModifyRights (see page 5509)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5497).
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5510)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5497).
V	ImplCanAddActionClass (see page 5510)	Determines if the given action class can be added.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5510)	Determines if the given action class can be added.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5511)	Lists supported end-user adapter actions.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetDisplayName (see page 5511)	Gets the display name for this TBaseEndUserAdapter (see page 5497).
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetEndUserAdapter (see page 5511)	Returns this TBaseEndUserAdapter (see page 5497).
V	ImplGetFieldsList (see page 5512)	Obtains the list of all of the fields for the end-user adapter.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetLoggedIn (see page 5512)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5512)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5497).

	ImplGetLogoutAction (see page 5513)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5497).
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5513)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5513)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplHasViewRights (see page 5514)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5514)	Processes the login for the given user.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5514)	Generate the HTML login page.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5515)	Processes the server-side login for the given user.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5515)	Generates the login page.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.

	UserList (see page 5515)	Gets the user list.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5516)	Determines if the user rights match the rights being queried.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.
	OnProduceLoginPage (see page 5516)	Allows implementation of a customized login page.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5989)	Returns the value of the DisplayName (see page 5994) property.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5990)	Returns the value of the EndUserAdapter (see page 5994) property.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetLoggedIn (see page 5990)	Returns the value of the LoggedIn (see page 5994) property.
	GetLoginFormAction (see page 5990)	Returns the value of the LoginFormAction (see page 5995) property.
	GetLogoutAction (see page 5991)	Returns the value of the LogoutAction (see page 5996) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.

	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5991)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5992)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5992)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5993)	Logs in a specified user.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5993)	Generates the Web page for logging in users.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
R	DisplayName (see page 5994)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5994)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	LoggedIn (see page 5994)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5995)	Indicates the action component for the login form.
	LoginPage (see page 5516)	Specifies the name of the page that acts as the login page.
R	LogoutAction (see page 5996)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	UserID (see page 5701)	Specifies the user ID of the current end user.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.58.2 TCustomEndUserSessionAdapter Methods

The methods of the TCustomEndUserSessionAdapter class are listed here.

Protected Methods

	Name	Description
V	ExecuteLogin (see page 5699)	Logs in an end user.

	ExecuteLogout (see page 5699)	Logs out the current user.
	GetDisplayNameValue (see page 5699)	Returns the name of the current end user.
	GetLoggedInValue (see page 5700)	Indicates whether the current end user has logged in.
	GetUserID (see page 5700)	Returns the user ID of the current user.

54.1.58.2.1 TCustomEndUserSessionAdapter.ExecuteLogin Method

Logs in an end user.

Pascal

```
procedure ExecuteLogin(AUserID: Variant); override;
```

C++

```
virtual __fastcall ExecuteLogin(Variant AUserID);
```

Description

ExecuteLogin is the implementation of the LogIn (see page 5515) action. When a server-side script executes the login action, it generates a call to ExecuteLogin to start the login process.

In TCustomEndUserAdapter (see page 5676), ExecuteLogin assigns the supplied user ID to the UserID (see page 5701) property, which saves the current user ID with the current session.

AUserID is the user ID of the user to log in.

54.1.58.2.2 TCustomEndUserSessionAdapter.ExecuteLogout Method

Logs out the current user.

Pascal

```
procedure ExecuteLogout; override;
```

C++

```
virtual __fastcall ExecuteLogout();
```

Description

ExecuteLogout is the implementation of the Logout action. When a server-side script executes the logout action, it generates a call to ExecuteLogout.

In TCustomEndUserAdapter (see page 5676), ExecuteLogout terminates the current Web session.

54.1.58.2.3 TCustomEndUserSessionAdapter.GetDisplayNameValue Method

Returns the name of the current end user.

Pascal

```
function GetDisplayNameValue: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDisplayNameValue();
```

Description

GetDisplayNameValue is the implementation of the DisplayName adapter field. When a server-side script references the display name of the current user, it generates a call to GetDisplayNameValue.

In TCustomEndUserSessionAdapter (see page 5692), GetDisplayNameValue returns the value of the UserID (see page

5701) property, formatted as a string.

54.1.58.2.4 TCustomEndUserSessionAdapter.GetLoggedInValue Method

Indicates whether the current end user has logged in.

Pascal

```
function GetLoggedInValue: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetLoggedInValue();
```

Description

GetLoggedInValue is the implementation of the LoggedIn adapter field. When a server-side script references the LoggedIn field for the current user, it generates a call to GetLoggedInValue.

In TCustomEndUserAdapter (see page 5676), GetLoggedInValue checks whether the UserID (see page 5701) property has been assigned. If UserID (see page 5701) is an assigned Variant (see page 3421) other than an empty string, GetLoggedInValue returns true, indicating that the user has logged in. Otherwise, GetLoggedInValue returns false.

54.1.58.2.5 TCustomEndUserSessionAdapter.GetUserID Method

Returns the user ID of the current user.

Pascal

```
function GetUserID: Variant; override;
```

C++

```
virtual __fastcall Variant GetUserID();
```

Description

GetUserIDValue is the implementation of the UserID (see page 5701) adapter field. When a server-side script references the user ID of the current user, it generates a call to GetUserIDValue.

In TCustomEndUserSessionAdapter (see page 5692), GetUserID fetches the value of the UserID (see page 5701) property from the current Web session. This value is set when the user first logs in.

See Also

GetUserID (see page 5684)

ExecuteLogin (see page 5683)

TWebSession (see page 6251)

54.1.58.3 TCustomEndUserSessionAdapter Properties

The properties of the TCustomEndUserSessionAdapter class are listed here.

Public Properties

	Name	Description
	UserID (see page 5701)	Specifies the user ID of the current end user.

54.1.58.3.1 TCustomEndUserSessionAdapter.UserID Property

Specifies the user ID of the current end user.

Pascal

```
property UserID: Variant;
```

C++

```
__property Variant UserID;
```

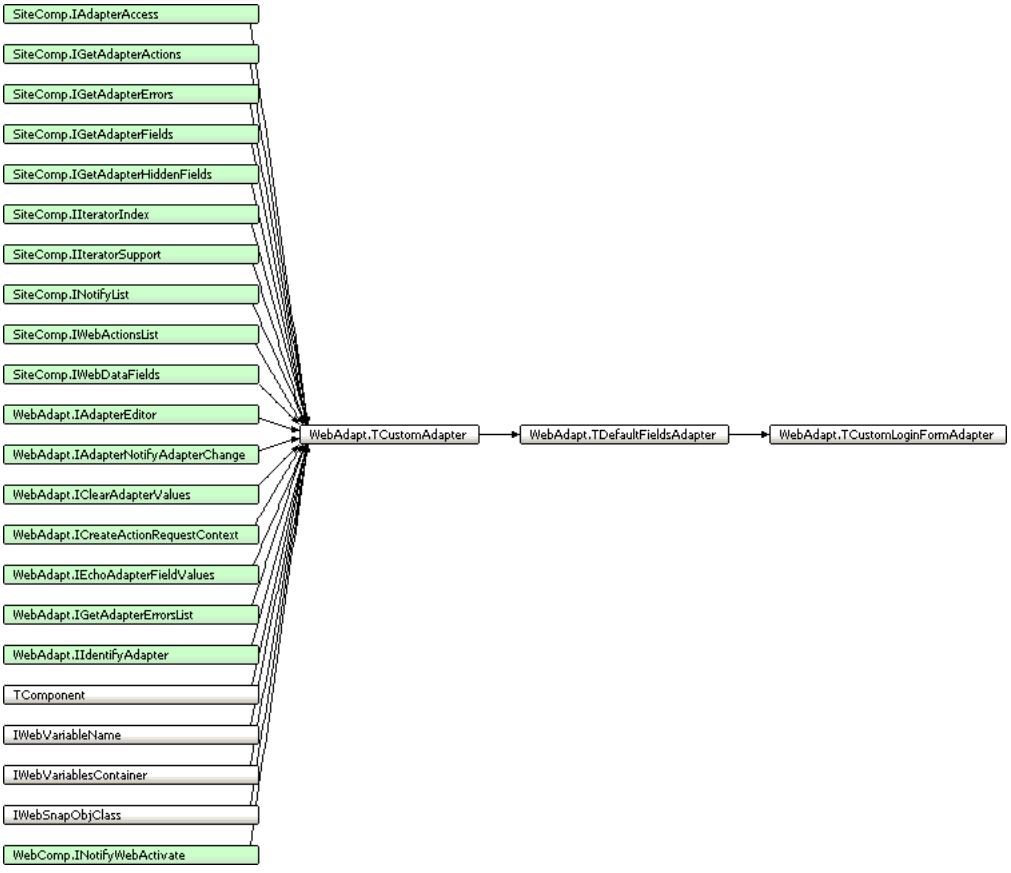
Description

Use UserID to get or set the user ID of the current user. When a user logs in (that is, when TCustomEndUserSessionAdapter (see page 5692) executes the LogIn (see page 5515) action), TCustomEndUserSessionAdapter (see page 5692) assigns the user's ID as the value of UserID. TCustomEndUserSessionAdapter (see page 5692) stores this value with the current session.

54.1.59 TCustomLoginFormAdapter Class

TCustomLoginFormAdapter is the base class for adapters that manage the fields and actions of a login form.

Class Hierarchy



Pascal

```
TCustomLoginFormAdapter = class(TDefaultFieldsAdapter);
```

C++

```
class TCustomLoginFormAdapter : public TDefaultFieldsAdapter;
```

File

WebAdapt

Description

TCustomLoginFormAdapter introduces default fields and actions that support building a login form. The adapter fields include user name (UserName (see page 5714)), password (Password (see page 5714)), and next page (NextPage (see page 5713)). The next page field lists the Web pages in the application. The properties of TCustomLoginFormAdapter reflect the values that are assigned to those fields. TCustomLoginFormAdapter implements a single default action for logging in. If the WebSnap application also contains an end user adapter such as TEndUserAdapter (see page 5764) or TEndUserSessionAdapter (see page 5792), TLoginFormAdapter (see page 5824) calls the end user adapter's LogIn method to log the user in. As an alternate approach, you can provide an OnLogin (see page 5715) event handler.

Do not create instances of TCustomLoginFormAdapter. It is intended only as a base class. Instead, use a TCustomLoginFormAdapter descendant such as TLoginFormAdapter (see page 5824).

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

TAdapterPageProducer (see page 1136)

TLoginFormAdapter (see page 5824)

54.1.59.1 TCustomLoginFormAdapter Members

The following tables list the members exposed by TCustomLoginFormAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.

	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
	EndUser (see page 5707)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 5707)	Logs in the user after the user name and password have been validated.
	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetNextPage (see page 5708)	Returns the value of the NextPage (see page 5713) property.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5708)	Provides the underlying implementation for the CanAddActionClass (see page 5539) method.
	ImplCanAddFieldClass (see page 5709)	Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5709)	Provides the underlying implementation for the GetActionsList (see page 5546) method.
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5710)	Provides the underlying implementation for the GetFieldsList (see page 5549) method.

	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5710)	Provides the underlying implementation for the NotifyDeactivate (see page 5566) method.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	RaiseBlankPassword (see page 5711)	Raises an EAdapterFieldException (see page 5178) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 5711)	Raises an EAdapterFieldException (see page 5178) with a message that the supplied user name is blank.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5711)	Returns the interface of the WebSnap application's Web user list.
	ValidatePassword (see page 5712)	Validates the password that the user supplied to the login form.
	ValidateUserName (see page 5712)	Validates the name that the user supplied to the login form.

	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.
	OnLogin (see page 5715)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5716)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5716)	Occurs when the login form adapter must validate the name that the user supplied.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.

	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	NextPage (see page 5713)	Names the Web page the user should see after successfully logging in.
R	Password (see page 5714)	Indicates the password that the user supplied to the login form.
	PasswordRequired (see page 5714)	Indicates whether the user must supply a password to log in.
R	UserName (see page 5714)	Indicates the name that the user supplied to the login form.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.59.2 TCustomLoginFormAdapter Methods

The methods of the TCustomLoginFormAdapter class are listed here.

Protected Methods

	Name	Description
	EndUser (see page 5707)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 5707)	Logs in the user after the user name and password have been validated.
V	GetNextPage (see page 5708)	Returns the value of the NextPage (see page 5713) property.
V	ImplCanAddActionClass (see page 5708)	Provides the underlying implementation for the CanAddActionClass (see page 5539) method.
V	ImplCanAddFieldClass (see page 5709)	Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.
V	ImplGetActionsList (see page 5709)	Provides the underlying implementation for the GetActionsList (see page 5546) method.
V	ImplGetFieldsList (see page 5710)	Provides the underlying implementation for the GetFieldsList (see page 5549) method.
V	ImplNotifyDeactivate (see page 5710)	Provides the underlying implementation for the NotifyDeactivate (see page 5566) method.
	RaiseBlankPassword (see page 5711)	Raises an EAdapterFieldException (see page 5178) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 5711)	Raises an EAdapterFieldException (see page 5178) with a message that the supplied user name is blank.

	UserList (see page 5711)	Returns the interface of the WebSnap application's Web user list.
	ValidatePassword (see page 5712)	Validates the password that the user supplied to the login form.
	ValidateUserName (see page 5712)	Validates the name that the user supplied to the login form.

54.1.59.2.1 TCustomLoginFormAdapter.EndUser Method

Returns the interface of the WebSnap application's end user adapter.

Pascal

```
function EndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser EndUser();
```

Description

TCustomLoginFormAdapter (see page 5701) uses EndUser internally to interact with the WebSnap application's end user adapter. For example, the login form adapter calls the LogIn method on this interface to log in an end user after validating the user name and password.

EndUser returns the interface of an end user adapter such as TEndUserAdapter (see page 5764) or TEndUserSessionAdapter (see page 5792). If the WebSnap application does not include an end user adapter, EndUser returns nil (Delphi) or NULL (C++).

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

LogIn (see page 5993)

54.1.59.2.2 TCustomLoginFormAdapter.ExecuteLogin Method

Logs in the user after the user name and password have been validated.

Pascal

```
procedure ExecuteLogin(AUserID: Variant);
```

C++

```
__fastcall ExecuteLogin(Variant AUserID);
```

Description

The LogIn action calls ExecuteLogin to log the user in after validating the user name and password. This method first generates an OnLogin (see page 5715) event, so that the OnLogin (see page 5715) event handler can perform any application-specific tasks to log the user in. Then, if the EndUser (see page 5707) property is not nil (Delphi) or NULL (C++), ExecuteLogin calls the end user adapter's LogIn method, so that the end user adapter can log the user in.

AUserID is the user ID of the end user. If the WebSnap application includes a TWebUserList component, this is the user ID that the Web user list associates with the current user name and password. Otherwise, AUserID is the value of the UserName (see page 5714) property.

See Also

EndUser (see page 5707)

OnLogin (see page 5715)

LogIn (see page 5993)

UserName (see page 5714)

TWebUserList (see page 6796)

54.1.59.2.3 TCustomLoginFormAdapter.GetNextPage Method

Returns the value of the NextPage (see page 5713) property.

Pascal

```
function GetNextPage: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetNextPage();
```

Description

GetNextPage is the getter method for the NextPage (see page 5713) property. See the NextPage (see page 5713) property for more information.

See Also

NextPage (see page 5713)

54.1.59.2.4 TCustomLoginFormAdapter.ImplCanAddActionClass Method

Provides the underlying implementation for the CanAddActionClass (see page 5539) method.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass provides the underlying implementation of the CanAddActionClass (see page 5539) method that is introduced in TCustomAdapter (see page 5531). At design time, the Actions (see page 5746) editor calls this method to determine what actions it can add to the login form adapter.

AParent is the adapter that will act as a parent to the added action.

AClass is the class of the proposed action.

ImplCanAddActionClass returns true if the login form adapter can act as a parent to an action of the specified class.

In TCustomLoginFormAdapter (see page 5701), ImplCanAddActionClass returns true if AClass is (or descends from) TLoginFormAdapterLoginAction (see page 5835) (the LogIn action).

See Also

ImplCanAddFieldClass (see page 5709)

CanAddActionClass (see page 5539)

TLoginFormAdapterLoginAction (see page 5835)

ImplGetActionsList (see page 5709)

54.1.59.2.5 TCustomLoginFormAdapter.ImplCanAddFieldClass Method

Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.

Pascal

```
function ImplCanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddFieldClass provides the underlying implementation of the CanAddFieldClass (see page 5540) method that is introduced in TCustomAdapter (see page 5531). At design time, the Fields editor calls this method to determine what fields it can add to the login form adapter.

AParent is the adapter that will act as a parent to the added field.

AClass is the class of the proposed field.

ImplCanAddFieldClass returns true if the login form adapter can act as a parent to a field of the specified class.

In TCustomLoginFormAdapter (see page 5701), ImplCanAddFieldClass returns true if AClass is (or descends from) TAdapterPasswordField (see page 5391), TAdapterUserNameField (see page 5433) or TAdapterNextPageField (see page 5379) (the Password (see page 5714), UserName (see page 5714), or NextPage (see page 5713) field).

See Also

ImplCanAddActionClass (see page 5708)

CanAddFieldClass (see page 5540)

TAdapterNextPageField (see page 5379)

TAdapterPasswordField (see page 5391)

TAdapterUserNameField (see page 5433)

ImplGetFieldsList (see page 5710)

54.1.59.2.6 TCustomLoginFormAdapter.ImplGetActionsList Method

Provides the underlying implementation for the GetActionsList (see page 5546) method.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList provides the underlying implementation of the GetActionsList (see page 5546) method that is introduced in TCustomAdapter (see page 5531). This method fills a string list with the names of all the actions an adapter can support, associating each action name with a class reference to the corresponding action class.

AList is an existing string list instance to fill with the names of the actions that this adapter supports.

In TCustomLoginFormAdapter (see page 5701), ImplGetActionsList clears the list of any strings it may contain on entry, and adds a single string, 'Login', which is the name of the login action. It associates this string with TLoginFormAdapterLoginAction (see page 5835).

See Also

ImplGetFieldsList (see page 5710)

GetActionsList (see page 5546)

TLoginFormAdapterLoginAction (see page 5835)

54.1.59.2.7 TCustomLoginFormAdapter.ImplGetFieldsList Method

Provides the underlying implementation for the GetFieldsList (see page 5549) method.

Pascal

```
procedure ImplGetFieldsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetFieldsList(TStrings AList);
```

Description

ImplGetFieldsList provides the underlying implementation of the GetFieldsList (see page 5549) method that is introduced in TCustomAdapter (see page 5531). This method fills a string list with the names of all the fields an adapter can support, associating each field name with a class reference to the corresponding field class.

AList is an existing string list instance to fill with the names of the fields that this adapter supports.

In TCustomLoginFormAdapter (see page 5701), ImplGetFieldsList clears the list of any strings it may contain on entry, and adds the strings, 'UserName (see page 5714)', 'Password (see page 5714)', and 'NextPage (see page 5713)', which it associates with TAdapterUserNameField (see page 5433), TAdapterPasswordField (see page 5391), and TAdapterNextPageField (see page 5379), respectively.

See Also

ImplGetActionsList (see page 5709)

GetFieldsList (see page 5549)

TAdapterNextPageField (see page 5379)

TAdapterPasswordField (see page 5391)

TAdapterUserNameField (see page 5433)

54.1.59.2.8 TCustomLoginFormAdapter.ImplNotifyDeactivate Method

Provides the underlying implementation for the NotifyDeactivate (see page 5566) method.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

ImplNotifyDeactivate provides the underlying implementation of the NotifyDeactivate (see page 5566) method that is introduced in TCustomAdapter (see page 5531). This method is called when the WebSnap application deactivates the Web module that contains the adapter after processing an HTTP request message.

In TCustomLoginFormAdapter (see page 5701), ImplNotifyDeactivate clears the NextPage (see page 5713) property so that

the next time a request message arrives from the login form, it can be redirected to an appropriate next page.

See Also

NotifyDeactivate (see page 5566)

NextPage (see page 5713)

54.1.59.2.9 TCustomLoginFormAdapter.RaiseBlankPassword Method

Raises an EAdapterFieldException (see page 5178) with a message that the supplied password is blank.

Pascal

```
procedure RaiseBlankPassword;
```

C++

```
__fastcall RaiseBlankPassword();
```

Description

TCustomLoginFormAdapter (see page 5701) calls RaiseBlankPassword internally when it is trying to validate a password that is an empty string and the PasswordRequired (see page 5714) property is true.

See Also

PasswordRequired (see page 5714)

EAdapterFieldException (see page 5178)

ValidatePassword (see page 5712)

54.1.59.2.10 TCustomLoginFormAdapter.RaiseBlankUserName Method

Raises an EAdapterFieldException (see page 5178) with a message that the supplied user name is blank.

Pascal

```
procedure RaiseBlankUserName;
```

C++

```
__fastcall RaiseBlankUserName();
```

Description

TCustomLoginFormAdapter (see page 5701) calls RaiseBlankUserName internally when it is trying to validate a user name that is an empty string.

See Also

EAdapterFieldException (see page 5178)

ValidateUserName (see page 5712)

54.1.59.2.11 TCustomLoginFormAdapter.UserList Method

Returns the interface of the WebSnap application's Web user list.

Pascal

```
function UserList: IWebUserList;
```

C++

```
__fastcall IWebUserList UserList();
```

Description

TCustomLoginFormAdapter (see page 5701) uses UserList internally to interact with the WebSnap application's Web user list. For example, the login form adapter calls the ValidateUser method on this interface when validating the user name and password.

UserList returns the interface of a Web user list such as TWebUserList. If the WebSnap application does not include a Web user list, UserList returns nil (Delphi) or NULL (C++).

See Also

TWebUserList (see page 6796)

ValidateUser (see page 1117)

54.1.59.2.12 TCustomLoginFormAdapter.ValidatePassword Method

Validates the password that the user supplied to the login form.

Pascal

```
procedure ValidatePassword;
```

C++

```
__fastcall ValidatePassword();
```

Description

The LogIn action associated with this adapter calls ValidatePassword to validate the password that the user supplies. ValidatePassword first generates an OnValidatePassword (see page 5716) event. If the event handler does not exist, or does not handle the validation, ValidatePassword checks whether the password is an empty string. If so, and if the PasswordRequired (see page 5714) property is true, ValidatePassword raises an EAdapterFieldException (see page 5178) exception.

See Also

OnValidatePassword (see page 5716)

PasswordRequired (see page 5714)

EAdapterFieldException (see page 5178)

Password (see page 5714)

54.1.59.2.13 TCustomLoginFormAdapter.ValidateUserName Method

Validates the name that the user supplied to the login form.

Pascal

```
procedure ValidateUserName;
```

C++

```
__fastcall ValidateUserName();
```

Description

The LogIn action associated with this adapter calls ValidateUserName to validate the name that the user supplies. ValidateUserName first generates an OnValidateUserName (see page 5716) event. If the event handler does not exist, or does not handle the validation, ValidateUserName checks whether the user name is an empty string. If so, it raises an

EAdapterFieldException (see page 5178) exception.

See Also

OnValidateUserName (see page 5716)

EAdapterFieldException (see page 5178)

UserName (see page 5714)

54.1.59.3 TCustomLoginFormAdapter Properties

The properties of the TCustomLoginFormAdapter class are listed here.

Public Properties

	Name	Description
	NextPage (see page 5713)	Names the Web page the user should see after successfully logging in.
R	Password (see page 5714)	Indicates the password that the user supplied to the login form.
	PasswordRequired (see page 5714)	Indicates whether the user must supply a password to log in.
R	UserName (see page 5714)	Indicates the name that the user supplied to the login form.

54.1.59.3.1 TCustomLoginFormAdapter.NextPage Property

Names the Web page the user should see after successfully logging in.

Pascal

```
property NextPage: string;
```

C++

```
__property UnicodeString NextPage;
```

Description

NextPage specifies the page that the WebSnap application presents to end users after they successfully log in.

The NextPage adapter field supplies a value for the NextPage property. This is a page that the user selects using the login form. If NextPage is unassigned (that is, the if the login form does not contain a NextPage field or the user does not select a next page), TCustomLoginFormAdapter (see page 5701) checks whether the request parameters include a "success page", and if so, uses that. If there is no "success page" parameter, TCustomLoginFormAdapter (see page 5701) checks the Web context to see if there is a currently requested page the user was trying to reach before being diverted to the login page. If there is no "success page" parameter and no currently requested page, TCustomLoginFormAdapter (see page 5701) uses the application's default page.

See Also

DefaultPage (see page 6195)

RequestedPageName (see page 5974)

TAdapterPageProducer (see page 1136)

TLoginFormAdapter (see page 5824)

SuccessPage (see page 1009)

GetNextPage (see page 5708)

54.1.59.3.2 TCustomLoginFormAdapter.Password Property

Indicates the password that the user supplied to the login form.

Pascal

```
property Password: Variant;
```

C++

```
__property Variant Password;
```

Description

Read (see page 3634) Password to obtain the password that the user supplied to the login form. The login form adapter reads this value from the current request message.

See Also

PasswordRequired (see page 5714)

OnValidatePassword (see page 5716)

54.1.59.3.3 TCustomLoginFormAdapter.PasswordRequired Property

Indicates whether the user must supply a password to log in.

Pascal

```
property PasswordRequired: Boolean;
```

C++

```
__property Boolean PasswordRequired;
```

Description

Set PasswordRequired to indicate whether users must supply a password to log in. If PasswordRequired is true and the OnValidatePassword (see page 5716) event handler does not handle the validation of the password, then the login form adapter checks that the value of Password (see page 5714) is not an empty string. If Password (see page 5714) is an empty string, the login form adapter raises an EAdapterFieldException (see page 5178) exception when it tries to validate the password. If the OnValidatePassword (see page 5716) event handler takes care of validating the password, PasswordRequired is ignored.

Note: The login form adapter validates the password (and hence checks PasswordRequired) when the WebSnap application executes this adapter's LogIn action.

See Also

Password (see page 5714)

OnValidatePassword (see page 5716)

TLoginFormAdapterLoginAction (see page 5835)

EAdapterFieldException (see page 5178)

54.1.59.3.4 TCustomLoginFormAdapter.UserName Property

Indicates the name that the user supplied to the login form.

Pascal

```
property UserName: Variant;
```

C++

```
__property Variant UserName;
```

Description

Read (see page 3634) UserName to obtain the name that the user supplied when logging in. The login form adapter reads this value from the current request message.

See Also

Password (see page 5714)

OnValidateUserName (see page 5716)

54.1.59.4 TCustomLoginFormAdapter Events

The events of the TCustomLoginFormAdapter class are listed here.

Public Events

	Name	Description
	OnLogin (see page 5715)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5716)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5716)	Occurs when the login form adapter must validate the name that the user supplied.

54.1.59.4.1 TCustomLoginFormAdapter.OnLogin Event

Occurs after the password and user ID are validated.

Pascal

```
property OnLogin: TLoginUserIDEvent;
```

C++

```
__property TLoginUserIDEvent OnLogin;
```

Description

Write (see page 3707) an OnLogin event handler to respond when a user has successfully logged in. This event allows your application to record the user login in an application-specific manner.

When the login form adapter's LogIn action executes, the adapter first receives an OnValidatePassword (see page 5716) and an OnValidateUserName (see page 5716) event. After validating the password and user name, the login form adapter receives an OnLogin event so that it can log the user in. After the OnLogin event, if the WebSnap application includes an end user adapter, the login form adapter calls the end user adapter's LogIn method. Thus, it may not be necessary to write an OnLogin event handler if the application includes an end user adapter.

See Also

OnValidatePassword (see page 5716)

OnValidateUserName (see page 5716)

LogIn (see page 5993)

TWebUserList (see page 6796)

UserName (see page 5714)

54.1.59.4.2 TCustomLoginFormAdapter.OnValidatePassword Event

Occurs when the login form adapter must validate the password that the user supplied.

Pascal

```
property OnValidatePassword: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidatePassword;
```

Description

Write (see page 3707) an OnValidatePassword event handler to validate the password that the user supplies to the login form. This event is especially important if the WebSnap application does not include a TWebUserList component for checking passwords against user names. The supplied Password (see page 5714) is the value of the Value parameter on the event handler. You can check this against the user's name by reading the UserName (see page 5714) property.

When the login form adapter's LogIn action executes, the adapter first receives an OnValidatePassword event to check the password. If the event handler sets the Handled parameter to true, indicating that the password is valid, the adapter goes on to validate the user name using an OnValidateUserName (see page 5716) event, and then to log the user in using an OnLogin (see page 5715) event (and, if the application includes an end user adapter, by calling the end user adapter's LogIn method). If the event handler finds the password to be invalid, it should raise an exception. If the event handler does not validate the supplied password (that is, if it sets the Handled parameter to false), the login form adapter validates the password by checking whether it is non-blank if the PasswordRequired (see page 5714) property is true.

See Also

OnLogin (see page 5715)

OnValidateUserName (see page 5716)

Password (see page 5714)

UserName (see page 5714)

PasswordRequired (see page 5714)

54.1.59.4.3 TCustomLoginFormAdapter.OnValidateUserName Event

Occurs when the login form adapter must validate the name that the user supplied.

Pascal

```
property OnValidateUserName: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidateUserName;
```

Description

Write (see page 3707) an OnValidateUserName event handler to validate the name that the user supplies when logging in. This event occurs after the OnValidatePassword (see page 5716) event and before the OnLogin (see page 5715) event.

If the event handler sets the Handled parameter to true, the login form adapter assumes that the user name (which is the value of the Value parameter and of the UserName (see page 5714) property) is valid. In this case, the adapter checks whether the WebSnap application includes a TWebUserList component, and if so, checks the user name against the password. If the event handler sets the Handled parameter to false, the adapter validates the user name by checking whether it is nonempty. In this case, the adapter raises an EAdapterFieldException (see page 5178) exception if the value of UserName (see page 5714) is an empty string. If the OnValidateUserName event handler finds the user name to be invalid, it should raise an exception.

See Also

OnLogin (see page 5715)

OnValidatePassword (see page 5716)

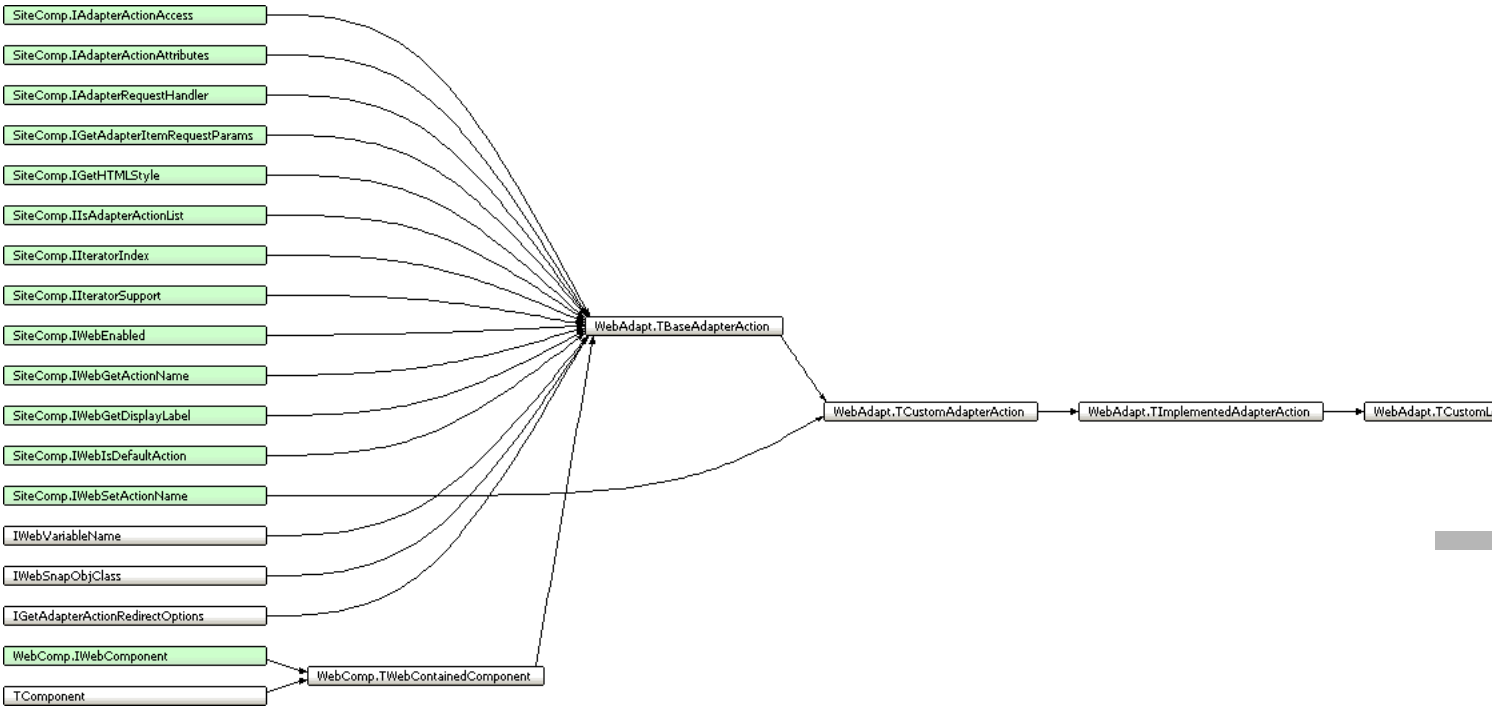
UserName (see page 5714)

EAdapterFieldException (see page 5178)

54.1.60 TCustomLoginFormAdapterAction Class

TCustomEndUserAdapterAction (see page 5687) is a base class for adapter actions that work with descendants of TCustomLoginFormAdapter (see page 5701).

Class Hierarchy



Pascal

```
TCustomLoginFormAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomLoginFormAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomLoginFormAdapterAction has two purposes. First, it implements a property that descendants use to access the parent adapter. Second, it defines a base class that TCustomLoginFormAdapter (see page 5701) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 5818)

TCustomLoginFormAdapter (see page 5701)

54.1.60.1 TCustomLoginFormAdapterAction Members

The following tables list the members exposed by TCustomLoginFormAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.

	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5721)	Returns the adapter that acts as a parent to this action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.

W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
R	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
R	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
R	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.60.2 TCustomLoginFormAdapterAction Properties

The properties of the TCustomLoginFormAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5721)	Returns the adapter that acts as a parent to this action.

54.1.60.2.1 TCustomLoginFormAdapterAction.Adapter Property

Returns the adapter that acts as a parent to this action.

Pascal

```
property Adapter: TCustomLoginFormAdapter;
```

C++

```
__property TCustomLoginFormAdapter Adapter;
```

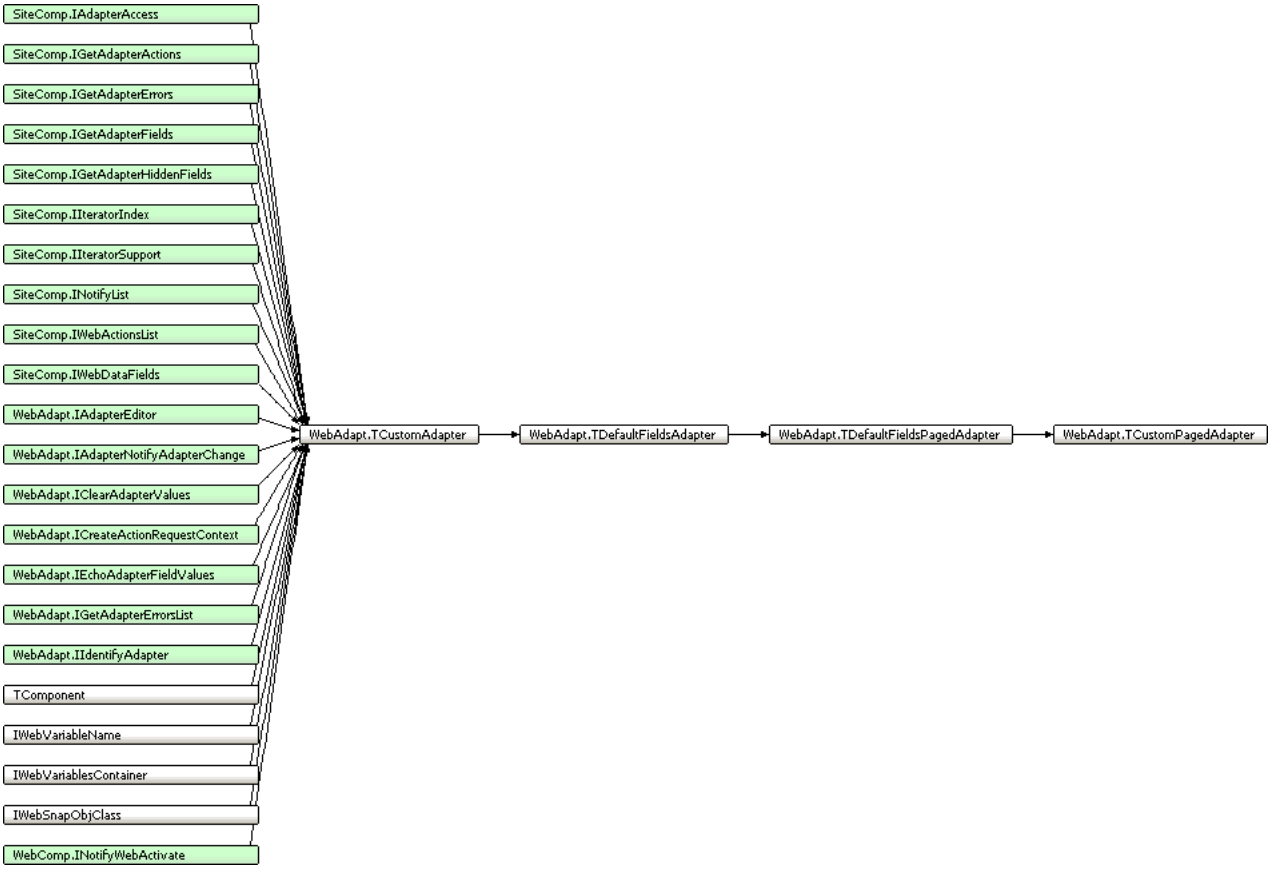
Description

Descendant classes use this property to access the parent adapter.

54.1.61 TCustomPagedAdapter Class

TCustomPagedAdapter is a base class for adapters that support paging.

Class Hierarchy



Pascal

```
TCustomPagedAdapter = class(TDefaultFieldsPagedAdapter);
```

C++

```
class TCustomPagedAdapter : public TDefaultFieldsPagedAdapter;
```

File

WebAdapt

Description

Derive from TCustomPagedAdapter to define a new adapter class that support paging. Paging is enabled when the PageSize (see page 5761) property is set.

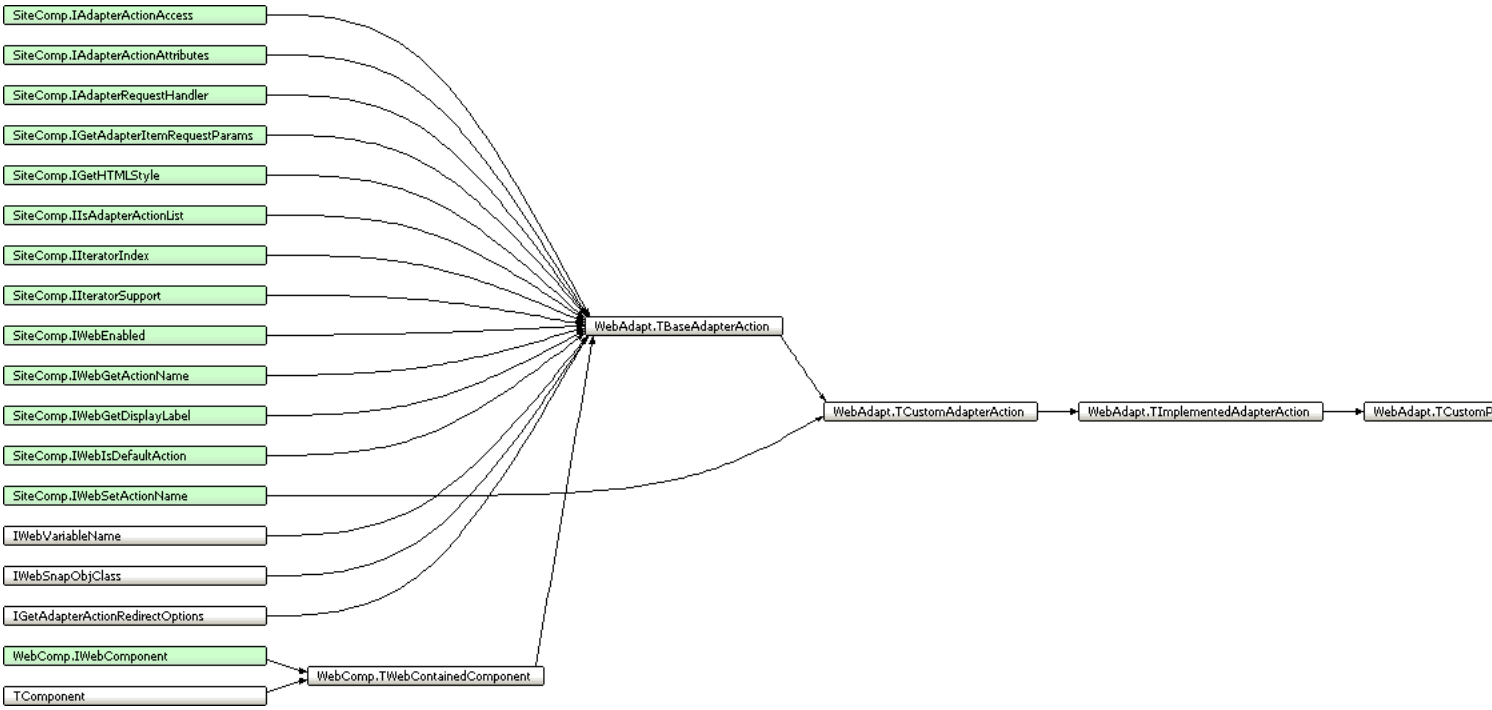
See Also

TPagedAdapter (see page 5844)

54.1.62 TCustomPagedAdapterAction Class

TCustomPagedAdapterAction is a base class for adapter actions that work with descendants of TDefaultFieldsPagedAdapter (see page 5747).

Class Hierarchy



Pascal

```
TCustomPagedAdapterAction = class(TImplementedAdapterAction);
```

C++

```
class TCustomPagedAdapterAction : public TImplementedAdapterAction;
```

File

WebAdapt

Description

TCustomPagedAdapterAction has two purposes. First, it provides a property that is used by descendants to access the parent adapter. Second, it defines a base class that TDefaultFieldsPagedAdapter (see page 5747) checks for when determining whether an adapter action can be added as a child.

See Also

TImplementedAdapterAction (see page 5818)

TDefaultFieldsPagedAdapter (see page 5747)

54.1.62.1 TCustomPagedAdapterAction Members

The following tables list the members exposed by TCustomPagedAdapterAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.

	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5727)	Returns the parent adapter of this adapter action.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultActionName (see page 5456)	Gets the default name used to reference this adapter action with server-side script.
	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
	ImplExecuteActionRequest (see page 5464)	Executes the adapter action using parameters passed in the HTTP request.
	ImplExecuteActionRequest (see page 5584)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.

	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
	Adapter (see page 5727)	Returns the parent adapter of this adapter action.
	Errors (see page 5475)	Returns the parent adapter's Errors list.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnAfterExecute (see page 5477)	Called after the adapter action has executed.
	OnAfterExecute (see page 5587)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5477)	Called after the adapter action response is generated.
	OnAfterGetResponse (see page 5587)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5478)	Called before the adapter action is executed.
	OnBeforeExecute (see page 5588)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5478)	Called before the adapter action response is generated.
	OnBeforeGetResponse (see page 5588)	Called before the adapter action response is generated.
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	OnGetParams (see page 5478)	Specifies name/value pairs needed to execute this adapter action.

Public Methods

	Name	Description
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	DisplayLabel (see page 5474)	Suggests a display label for this adapter action.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

54.1.62.2 TCustomPagedAdapterAction Methods

The methods of the TCustomPagedAdapterAction class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 5727)	Returns the parent adapter of this adapter action.

54.1.62.2.1 TCustomPagedAdapterAction.GetAdapter Method

Returns the parent adapter of this adapter action.

Pascal

```
function GetAdapter: TDefaultFieldsPagedAdapter;
```

C++

```
__fastcall TDefaultFieldsPagedAdapter GetAdapter();
```

Description

GetAdapter implements the Adapter (see page 5727) property.

See Also

TDefaultFieldsPagedAdapter (see page 5747)

Adapter (see page 5727)

54.1.62.3 TCustomPagedAdapterAction Properties

The properties of the TCustomPagedAdapterAction class are listed here.

Protected Properties

	Name	Description
R	Adapter (see page 5727)	Returns the parent adapter of this adapter action.

54.1.62.3.1 TCustomPagedAdapterAction.Adapter Property

Returns the parent adapter of this adapter action.

Pascal

```
property Adapter: TDefaultFieldsPagedAdapter;
```

C++

```
__property TDefaultFieldsPagedAdapter Adapter;
```

Description

Descendant classes use this property to access the parent adapter.

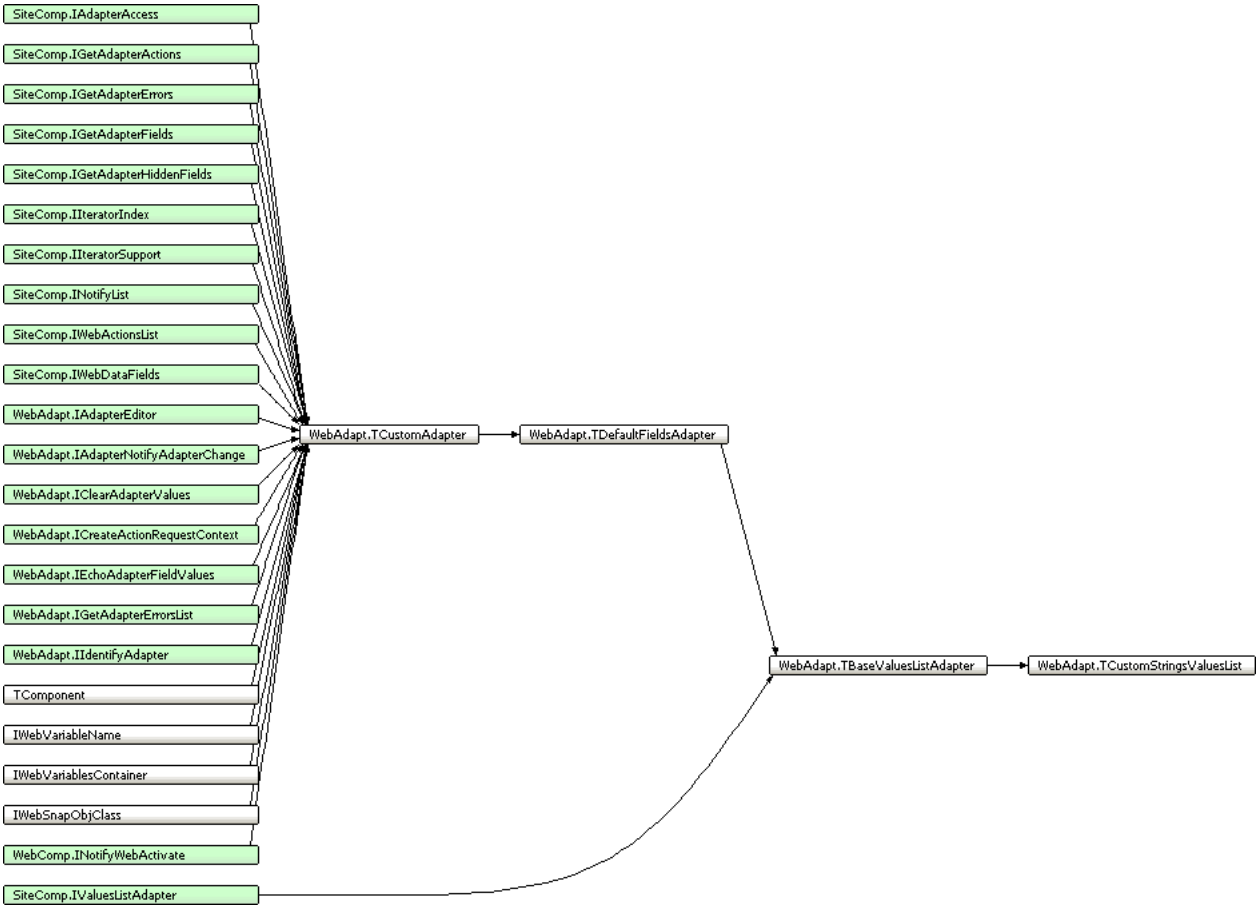
See Also

GetAdapter (see page 5727)

54.1.63 TCustomStringsValuesList Class

TCustomStringsValuesList is a base class for a values list that defines it's name/value pairs using a string list.

Class Hierarchy



Pascal

```
TCustomStringsValuesList = class(TBaseValuesListAdapter);
```

C++

```
class TCustomStringsValuesList : public TBaseValuesListAdapter;
```

File

WebAdapt

Description

Use descendants of TCustomStringsValuesList with adapter fields to define a list of name/value pairs that are possible values for the adapter field. Adapter fields reference a values list adapter using the adapter field's ValuesList property.

See Also

TBaseValuesListAdapter (see page 5517)

TStringsValuesList (see page 5857)

TAdapterUpdateField (see page 5419)

54.1.63.1 TCustomStringsValuesList Members

The following tables list the members exposed by TCustomStringsValuesList.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.

	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5522)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5523)	Looks up an image for a value.
	GetListName (see page 5523)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5523)	Looks up a name for a value.
	GetListValue (see page 5524)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5524)	Returns the class type for this adapter field.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5525)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5734)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5525)	Adds default field components to this adapter.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetListImage (see page 5525)	Responsible returning an image associated with the current name/value pair.
V	ImplGetListImageOfValue (see page 5526)	Looks up an image for a value.
V	ImplGetListName (see page 5735)	Responsible returning the name of the name/value pair.
V	ImplGetListNameOfValue (see page 5735)	Looks up a name for a value.
V	ImplGetListValue (see page 5735)	Responsible for returning the value of the name/value pair.
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
V	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
V	ImplNextIteration (see page 5736)	Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5736)	Clears the iteration index.

	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5736)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	PrepareStrings (see page 5737)	Customizes the content of the strings list.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 5737)	This flag indicates whether the strings list has been prepared or customized.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareStrings (see page 5738)	Implement OnPrepareStrings to customize the content of the strings list.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5733)	Create is the constructor for TCustomStringsValuesList (see page 5728).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5733)	Destroy is the destructor for TCustomStringsList.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetListImage (see page 1074)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1074)	Returns the image object associated with a specified value.
	GetListName (see page 1075)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1075)	Returns the name associated with a specified value.
	GetListValue (see page 1076)	Returns the value of the current name/value pair.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.

	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	Strings (see page 5737)	This string list defines the list of name/value pairs.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.63.2 TCustomStringsValuesList.Create Constructor

Create is the constructor for TCustomStringsValuesList (see page 5728).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomStringsValuesList(TComponent * AOwner);
```

Description

Create creates the strings list.

See Also

Strings (see page 5737)

54.1.63.3 TCustomStringsValuesList.Destroy Destructor

Destroy is the destructor for TCustomStringsList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomStringsValuesList();
```

Description

Destroy frees the strings list.

See Also

Strings (see page 5737)

54.1.63.4 TCustomStringsValuesList Methods

The methods of the TCustomStringsValuesList class are listed here.

Protected Methods

	Name	Description
	GetNameFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetValueFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.

	ImplEndIterator (see page 5734)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the records of the adapter.
	ImplGetListName (see page 5735)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 5735)	Looks up a name for a value.
	ImplGetListValue (see page 5735)	Responsible for returning the value of the name/value pair.
	ImplNextIteration (see page 5736)	Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyDeactivate (see page 5736)	Clears the iteration index.
	ImplStartIterator (see page 5736)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	PrepareStrings (see page 5737)	Customizes the content of the strings list.

54.1.63.4.1 TCustomStringsValuesList.GetNameFieldClass Method

Returns the class of adapter field that is responsible for providing the value of the name/value pair.

Pascal

```
function GetNameFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetNameFieldClass();
```

Description

GetNameFieldClass returns a TValuesListNameField (see page 5866).

See Also

GetValueFieldClass (see page 5734)

TValuesListNameField (see page 5866)

54.1.63.4.2 TCustomStringsValuesList.GetValueFieldClass Method

Returns the class of adapter field that is responsible for providing the name of the name/value pair.

Pascal

```
function GetValueFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns TValuesListValueField (see page 5869).

See Also

GetNameFieldClass (see page 5734)

TValuesListValueField (see page 5869)

54.1.63.4.3 TCustomStringsValuesList.ImplEndIterator Method

Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the records of the adapter.

See Also

EndIterator (see page 1056)

54.1.63.4.4 TCustomStringsValuesList.ImplGetListName Method

Responsible returning the name of the name/value pair.

Pascal

```
function ImplGetListName: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetListName();
```

Description

ImplGetListName uses the iteration index to retrieve a name from the strings list. The name may be blank.

See Also

Strings (see page 5737)

54.1.63.4.5 TCustomStringsValuesList.ImplGetListNameOfValue Method

Looks up a name for a value.

Pascal

```
function ImplGetListNameOfValue(Value: Variant): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplGetListNameOfValue(Variant Value);
```

Description

ImplGetListNameOfValue searches the string list for a matching value and returns the name associated with this value. An empty string is returned if the value is not found or if the name associated with the value is blank.

See Also

Strings (see page 5737)

54.1.63.4.6 TCustomStringsValuesList.ImplGetListValue Method

Responsible for returning the value of the name/value pair.

Pascal

```
function ImplGetListValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetListValue();
```

Description

ImplGetListValue uses the iteration index to retrieve a value from the strings list.

See Also

Strings (see page 5737)

54.1.63.4.7 TCustomStringsValuesList.ImplNextIteration Method

Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration increments the iteration index. false is returned if the iteration index is out of the range defined by the Strings (see page 5737) property's Count property. The iteration index is used when retrieving a name or value from the strings list.

See Also

NextIteration (see page 1056)

54.1.63.4.8 TCustomStringsValuesList.ImplNotifyDeactivate Method

Clears the iteration index.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The StringsPrepared (see page 5737) property is cleared. This ensures that the strings list will be prepared during the processing of the next HTTP request.

54.1.63.4.9 TCustomStringsValuesList.ImplStartIterator Method

Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator initializes the iteration index to 0. false is returned if the iteration index is out of the range defined by the Strings (see page 5737) property's Count property. The iteration index is used when retrieving a name or value from the strings list.

See Also

StartIterator (see page 1056)

54.1.63.4.10 TCustomStringsValuesList.PrepareStrings Method

Customizes the content of the strings list.

Pascal

```
procedure PrepareStrings; virtual;
```

C++

```
virtual __fastcall PrepareStrings();
```

Description

PrepareStrings calls the OnPrepareStrings (see page 5738) event handler.

See Also

StringsPrepared (see page 5737)

OnPrepareStrings (see page 5738)

54.1.63.5 TCustomStringsValuesList Properties

The properties of the TCustomStringsValuesList class are listed here.

Protected Properties

	Name	Description
	StringsPrepared (see page 5737)	This flag indicates whether the strings list has been prepared or customized.

Public Properties

	Name	Description
	Strings (see page 5737)	This string list defines the list of name/value pairs.

54.1.63.5.1 TCustomStringsValuesList.Strings Property

This string list defines the list of name/value pairs.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Each line in the string list represents a name/value pair. The name is optional. To specify a name/value enter a line with the name following by "=" followed by the value. For example, "Joe Smith=EMP111" is a named value. "EMP111" is an unnamed value.

54.1.63.5.2 TCustomStringsValuesList.StringsPrepared Property

This flag indicates whether the strings list has been prepared or customized.

Pascal

```
property StringsPrepared: Boolean;
```

C++

```
__property Boolean StringsPrepared;
```

Description

This flag is checked before calling PrepareStrings (see page 5737).

See Also

PrepareStrings (see page 5737)

54.1.63.6 TCustomStringsValuesList Events

The events of the TCustomStringsValuesList class are listed here.

Public Events

	Name	Description
	OnPrepareStrings (see page 5738)	Implement OnPrepareStrings to customize the content of the strings list.

54.1.63.6.1 TCustomStringsValuesList.OnPrepareStrings Event

Implement OnPrepareStrings to customize the content of the strings list.

Pascal

```
property OnPrepareStrings: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareStrings;
```

Description

OnPrepareStrings is called before the name/value pairs are retrieved by server-side script.

See Also

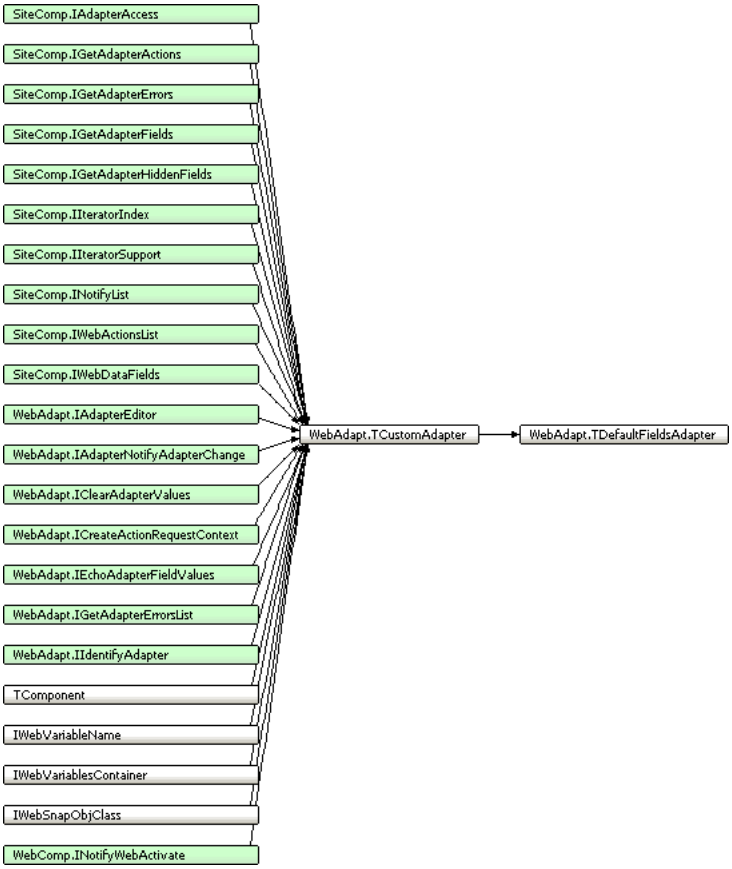
TNotifyEvent

PrepareStrings (see page 5737)

54.1.64 TDefaultFieldsAdapter Class

TDefaultFieldsAdapter is a base class for adapters that automatically adds default fields and/or actions.

Class Hierarchy



Pascal

```
TDefaultFieldsAdapter = class(TCustomAdapter);
```

C++

```
class TDefaultFieldsAdapter : public TCustomAdapter;
```

File

WebAdapt

Description

Derive from TDefaultFieldsAdapter to define adapter components that automatically add adapter field and/or adapter action components. Default fields are added to the adapter unless one or more field has been explicitly added.

Default actions are added to the adapter unless one or more action has been explicitly added.

See Also

TCustomAdapter (see page 5531)

54.1.64.1 TDefaultFieldsAdapter Members

The following tables list the members exposed by TDefaultFieldsAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.

	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.

	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.

	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.64.2 TDefaultFieldsAdapter Methods

The methods of the TDefaultFieldsAdapter class are listed here.

Protected Methods

	Name	Description
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
V	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
V	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
V	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.

54.1.64.2.1 TDefaultFieldsAdapter.AddDefaultActions Method

Adds default action components to this adapter.

Pascal

```
procedure AddDefaultActions(AComponents: TWebComponentList); virtual;
```

C++

```
virtual __fastcall AddDefaultActions(TWebComponentList AComponents);
```

Description

AddDefaultActions calls ImplGetActionsList (see page 5555). Descendant classes should override TCustomAdapter (see page 5531)'s ImplGetActionsList (see page 5555) method to provide the list of default actions that will be automatically added to this adapter.

See Also

TWebComponentList (see page 6025)

ImplGetActionsList (see page 5555)

54.1.64.2.2 TDefaultFieldsAdapter.AddDefaultFields Method

Adds default field components to this adapter.

Pascal

```
procedure AddDefaultFields(AComponents: TWebComponentList); virtual;
```

C++

```
virtual __fastcall AddDefaultFields(TWebComponentList AComponents);
```

Description

AddDefaultFields calls ImplGetFieldList. Descendant classes should override TCustomAdapter (see page 5531)'s ImplGetFieldList method to provide the list of default fields that will be automatically added to this adapter.

See Also

TWebComponentList (see page 6025)

ImplGetFieldsList (see page 5556)

54.1.64.2.3 TDefaultFieldsAdapter.DefaultWebComponentsChange Method

Notifies other components of changes to the list of default actions or fields.

Pascal

```
procedure DefaultWebComponentsChange(Sender: TObject);
```

C++

```
__fastcall DefaultWebComponentsChange(TObject * Sender);
```

Description

The list of default fields or actions may be based on properties of the adapter or properties of other components. For example, the default fields of TDataSetAdapter are based on the fields in the dataset. If the TDataSetAdapter references a different dataset then the default fields list will change. If the dataset's field list changes then TDataSetAdapter's default fields will change. DefaultWebCompoentsChange is called in these circumstances so that components that are using this adapter can be notified of changes to the list of fields or actions. DefaultWebComponentsChange calls NotifyAdapterChange (see page 5565).

See Also

NotifyAdapterChange (see page 5565)

54.1.64.2.4 TDefaultFieldsAdapter.GetActionsClass Method

Specifies the class to create as a container for adapter actions components.

Pascal

```
function GetActionsClass: TAdapterActionsClass; override;
```

C++

```
virtual __fastcall TAdapterActionsClass GetActionsClass();
```

Description

GetActionsClass returns TAdapterDefaultActions (see page 5231).

See Also

TAdapterDefaultActions (see page 5231)

54.1.64.2.5 TDefaultFieldsAdapter.GetDataClass Method

Specifies the class to create as a container for adapter field components.

Pascal

```
function GetDataClass: TAdapterFieldsClass; override;
```

C++

```
virtual __fastcall TAdapterFieldsClass GetDataClass();
```

Description

GetDataClass returns TAdapterDefaultFields (see page 5237).

See Also

TAdapterDefaultFields (see page 5237)

54.1.64.2.6 TDefaultFieldsAdapter.ImplGetVisibleActions Method

Returns a list of this adapter's action components.

Pascal

```
function ImplGetVisibleActions: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleActions();
```

Description

ImplGetVisibleActions returns a list of actions that have been added to the adapter. If no adapter fields have been explicitly added, ImplGetVisibleActions returns a list containing all default actions.

See Also

TWebComponentList (see page 6025)

ImplGetVisibleActions (see page 5557)

54.1.64.2.7 TDefaultFieldsAdapter.ImplGetVisibleFields Method

Returns a list of this adapter's field components.

Pascal

```
function ImplGetVisibleFields: TWebComponentList; override;
```

C++

```
virtual __fastcall TWebComponentList ImplGetVisibleFields();
```

Description

ImplGetVisibleFields returns a list of fields that have been added to the adapter. If no adapter fields have been explicitly added,

ImplGetVisibleFields returns a list containing all default fields.

See Also

TWebComponentList (see page 6025)

ImplGetVisibleFields (see page 5558)

54.1.64.2.8 TDefaultFieldsAdapter.ImplSetActions Method

Initializes the Actions (see page 5746) property.

Pascal

```
procedure ImplSetActions(const Value: TAdapterSubComponent); override;
```

C++

```
virtual __fastcall ImplSetActions(const TAdapterSubComponent Value);
```

Description

The Actions (see page 5746) property is the container for adapter action components.

See Also

ImplSetActions (see page 5561)

54.1.64.2.9 TDefaultFieldsAdapter.ImplSetData Method

Initializes the Data (see page 5747) property.

Pascal

```
procedure ImplSetData(const Value: TAdapterSubComponent); override;
```

C++

```
virtual __fastcall ImplSetData(const TAdapterSubComponent Value);
```

Description

The Data (see page 5747) property is the container for adapter field components.

See Also

ImplSetData (see page 5561)

54.1.64.3 TDefaultFieldsAdapter Properties

The properties of the TDefaultFieldsAdapter class are listed here.

Public Properties

	Name	Description
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5747)	Container for adapter field components.

54.1.64.3.1 TDefaultFieldsAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterDefaultActions;
```

C++

```
__property TAdapterDefaultActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5231)

54.1.64.3.2 TDefaultFieldsAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterDefaultFields;
```

C++

```
__property TAdapterDefaultFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

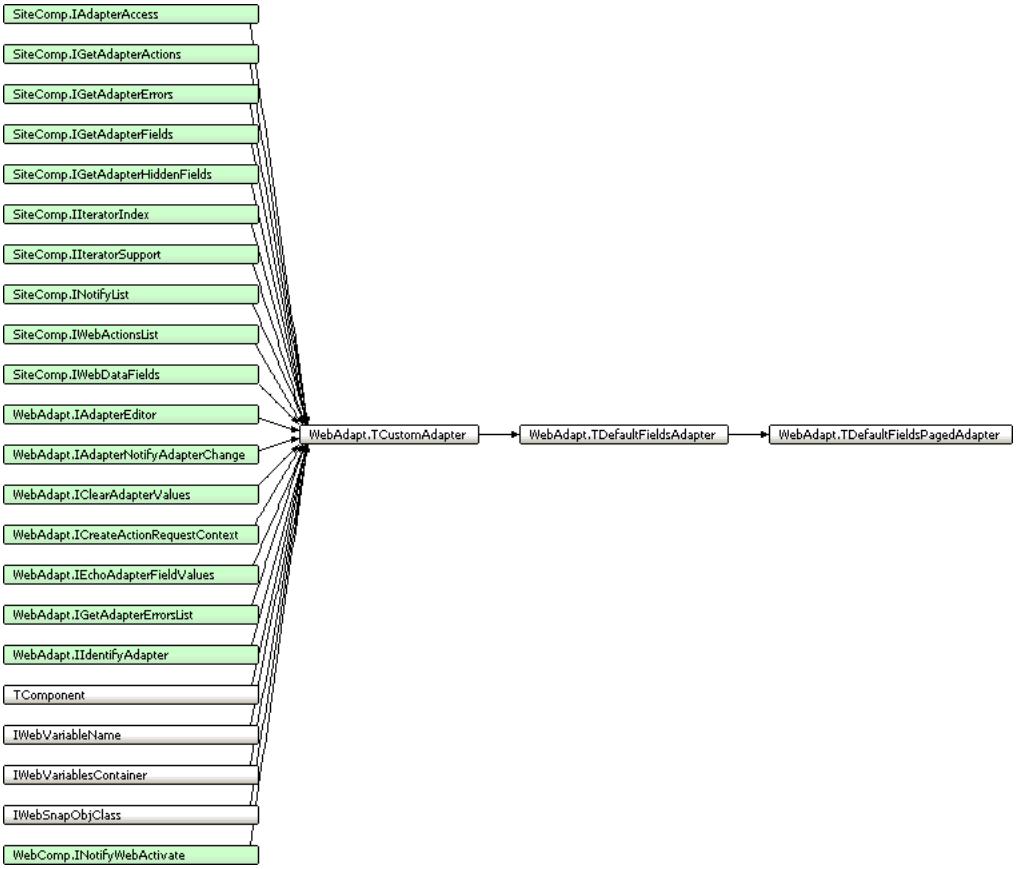
See Also

TAdapterDefaultFields (see page 5237)

54.1.65 TDefaultFieldsPagedAdapter Class

TDefaultFieldsPagedAdapter is a base class for adapters that support paging.

Class Hierarchy



Pascal

```
TDefaultFieldsPagedAdapter = class(TDefaultFieldsAdapter);
```

C++

```
class TDefaultFieldsPagedAdapter : public TDefaultFieldsAdapter;
```

File

WebAdapt

Description

Enable paging by setting the PageSize (see page 5761) property. When paging is enabled, the way that the adapter enumerates records is affected. The adapter calculates the number of pages by dividing the number of records by the page size. It keeps track of a current page number. When the enumerating records, the records on the current page are enumerated. Server-side script simply enumerates the records. Server-side script does not need to be aware of the page size or current page number. When paging is enabled, default actions for moving between pages are automatically added to the adapter.

See Also

TDefaultFieldsAdapter (see page 5738)

TAdapterNextPageAction (see page 5370)

TAdapterPrevPageAction (see page 5396)

TAdapterGotoPageAction (see page 5291)

54.1.65.1 TDefaultFieldsPagedAdapter Members

The following tables list the members exposed by TDefaultFieldsPagedAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5753)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5754)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 5754)	Indicates whether the record index has gone past the last record in the current page.
V	ExtractRequestParams (see page 5754)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
V	GetEOF (see page 5755)	Indicates whether the record position has passed the last record.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetFirstRecord (see page 5755)	Set the record position to the first record.

	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetNextRecord (see page 5755)	Increments the record position.
	GetPageCount (see page 5756)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 5756)	Returns the total number of records.
	GetRecordIndex (see page 5756)	Returns the index of the current record.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5757)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5757)	Implements IteratorSupport's EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5757)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5758)	Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5758)	Clears the current page variable.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5759)	Implements IteratorSupport.StartIterator (see page 5567) to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.

	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
V	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	SetPageSize (see page 5759)	Updates the pagesize variable.
	SkipToPage (see page 5760)	Skips records until the current record is within the current page.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 5762)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5763)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5763)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 5763)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5763)	Implement this event to identify the zero-based current record.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.

	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
	CurrentPage (see page 5760)	Sets or gets the current page index.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
R	PageCount (see page 5761)	Calculates the number of pages needed to display all adapter records.
	PageSize (see page 5761)	Sets or gets the page size variable.
R	RecordCount (see page 5761)	Returns the total number of records.
R	RecordIndex (see page 5762)	Returns the index of the current record.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.

	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.65.2 TDefaultFieldsPagedAdapter Methods

The methods of the TDefaultFieldsPagedAdapter class are listed here.

Protected Methods

	Name	Description
	AddDefaultActions (see page 5753)	Adds default action components to this adapter.
	EncodeActionParams (see page 5754)	Writes name/value pairs that represent adapter state.
	EndOfPage (see page 5754)	Indicates whether the record index has gone past the last record in the current page.
	ExtractRequestParams (see page 5754)	Reads name/value pairs from the HTTP request.
	GetEOF (see page 5755)	Indicates whether the record position has passed the last record.
	GetFirstRecord (see page 5755)	Set the record position to the first record.
	GetNextRecord (see page 5755)	Increments the record position.
	GetPageCount (see page 5756)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 5756)	Returns the total number of records.
	GetRecordIndex (see page 5756)	Returns the index of the current record.
	ImplCanAddActionClass (see page 5757)	Indicates whether a type of adapter action can be added to this adapter.
	ImplEndIterator (see page 5757)	Implements IteratorSupport's EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplGetActionsList (see page 5757)	Provides a list of action types that can be added to this adapter.
	ImplNextIteration (see page 5758)	Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyDeactivate (see page 5758)	Clears the current page variable.
	ImplStartIterator (see page 5759)	Implements IteratorSupport.StartIterator (see page 5567) to support iterating over the records of the adapter.
	SetPageSize (see page 5759)	Updates the pagesize variable.
	SkipToPage (see page 5760)	Skips records until the current record is within the current page.

54.1.65.2.1 TDefaultFieldsPagedAdapter.AddDefaultActions Method

Adds default action components to this adapter.

Pascal

```
procedure AddDefaultActions(AComponents: TWebComponentList); override;
```

C++

```
virtual __fastcall AddDefaultActions(TWebComponentList AComponents);
```

Description

AddDefaultActions adds TAdapterNextPageAction (see page 5370), TAdapterGotoPageAction (see page 5291), and TAdapterPrevPageAction (see page 5396).

See Also

TWebComponentList (see page 6025)

TAdapterNextPageAction (see page 5370)

TAdapterGotoPageAction (see page 5291)

TAdapterPrevPageAction (see page 5396)

54.1.65.2.2 TDefaultFieldsPagedAdapter.EncodeActionParams Method

Writes name/value pairs that represent adapter state.

Pascal

```
procedure EncodeActionParams(AParams: IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall EncodeActionParams(IAdapterItemRequestParams AParams);
```

Description

ExtractRequestParams (see page 5754) writes the current page number as a name/value pair.

See Also

ExtractRequestParams (see page 5543)

ExtractRequestParams (see page 5754)

54.1.65.2.3 TDefaultFieldsPagedAdapter.EndOfPage Method

Indicates whether the record index has gone past the last record in the current page.

Pascal

```
function EndOfPage: Boolean;
```

C++

```
__fastcall Boolean EndOfPage();
```

Description

EndOfPage tests for the end of page using PageSize (see page 5761), RecordIndex (see page 5762), and CurrentPage (see page 5760).

See Also

PageSize (see page 5761)

RecordIndex (see page 5762)

CurrentPage (see page 5760)

54.1.65.2.4 TDefaultFieldsPagedAdapter.ExtractRequestParams Method

Reads name/value pairs from the HTTP request.

Pascal

```
procedure ExtractRequestParams(ARequest: IUnknown); override;
```

C++

```
virtual __fastcall ExtractRequestParams(IUnknown ARequest);
```

Description

ExtractRequestParams is called by a child adapter action before executing the action. This gives the adapter an opportunity to extract values from the HTTP request that are important to the behavior of the adapter. ExtractRequestParams extracts the current page number from the request.

See Also

EncodeActionParams (see page 5754)

ExtractRequestParams (see page 5543)

54.1.65.2.5 TDefaultFieldsPagedAdapter.GetEOF Method

Indicates whether the record position has passed the last record.

Pascal

```
function GetEOF: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetEOF();
```

Description

GetEOF calls the OnGetFirstRecord (see page 5763) event handler. If OnGetFirstRecord (see page 5763) is not implemented then the RecordIndex (see page 5762) is compared to the RecordCount (see page 5761). Descendant classes may override this method. For example, TDataSetAdapter tests the dataset's EOF.

See Also

OnGetEOF (see page 5762)

RecordIndex (see page 5762)

RecordCount (see page 5761)

54.1.65.2.6 TDefaultFieldsPagedAdapter.GetFirstRecord Method

Set the record position to the first record.

Pascal

```
function GetFirstRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetFirstRecord();
```

Description

GetFirstRecord calls the OnGetFirstRecord (see page 5763) event handler. Descendant classes may override this method. For example, TDataSetAdapter sets the record position of a dataset.

See Also

OnGetFirstRecord (see page 5763)

54.1.65.2.7 TDefaultFieldsPagedAdapter.GetNextRecord Method

Increments the record position.

Pascal

```
function GetNextRecord: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetNextRecord();
```

Description

GetNextRecord calls the OnGetNextRecord (see page 5763) event handler. Descendant classes may override this method. For example, TDataSetAdapter increments the record position of a dataset.

See Also

OnGetNextRecord (see page 5763)

54.1.65.2.8 TDefaultFieldsPagedAdapter.GetPageCount Method

Calculates the number of pages needed to display all adapter records.

Pascal

```
function GetPageCount: Integer;
```

C++

```
__fastcall int GetPageCount();
```

Description

GetPageCount calculates the page count using PageCount (see page 5761), PageSize (see page 5761), and RecordCount (see page 5761).

See Also

RecordCount (see page 5761)

PageCount (see page 5761)

PageSize (see page 5761)

54.1.65.2.9 TDefaultFieldsPagedAdapter.GetRecordCount Method

Returns the total number of records.

Pascal

```
function GetRecordCount: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordCount();
```

Description

GetRecordCount calls the OnGetRecordCount (see page 5763) event handler. Descendant classes may override this method. For example, TDataSetAdapter returns the number of records in the dataset.

See Also

OnGetRecordCount (see page 5763)

RecordCount (see page 5761)

54.1.65.2.10 TDefaultFieldsPagedAdapter.GetRecordIndex Method

Returns the index of the current record.

Pascal

```
function GetRecordIndex: Integer; virtual;
```

C++

```
virtual __fastcall int GetRecordIndex();
```

Description

GetRecordIndex calls the OnGetRecordIndex (see page 5763) event handler.

See Also

OnGetRecordIndex (see page 5763)

54.1.65.2.11 TDefaultFieldsPagedAdapter.ImplCanAddActionClass Method

Indicates whether a type of adapter action can be added to this adapter.

Pascal

```
function ImplCanAddActionClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

ImplCanAddActionClass allows action classes that derive from TCustomPagedAdapterAction (see page 5722) in addition to those allowed by TCustomAdapter (see page 5531)'s ImplCanAddActionsClass method.

See Also

ImplCanAddActionClass (see page 5553)

ImplCanAddClass (see page 5212)

TCustomPagedAdapterAction (see page 5722)

54.1.65.2.12 TDefaultFieldsPagedAdapter.ImplEndIterator Method

Implements IteratorSupport's EndIterator (see page 5542) method to support iterating over the data records of the adapter.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

EndIterator (see page 5542) is called when record iteration is complete.

See Also

EndIterator (see page 1056)

54.1.65.2.13 TDefaultFieldsPagedAdapter.ImplGetActionsList Method

Provides a list of action types that can be added to this adapter.

Pascal

```
procedure ImplGetActionsList(AList: TStrings); override;
```

C++

```
virtual __fastcall ImplGetActionsList(TStrings AList);
```

Description

ImplGetActionsList lists PrevPage, GotoPage, and NextPage actions.

See Also

TAdapterNextPageAction (see page 5370)

TAdapterGotoPageAction (see page 5291)

TAdapterPrevPageAction (see page 5396)

ImplGetActionsList (see page 5555)

54.1.65.2.14 TDefaultFieldsPagedAdapter.ImplNextIteration Method

Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

ImplNextIteration calls GetNextRecord (see page 5755) if the RecordIndex (see page 5762) can be incremented without running past the end of the current page. false is returned if RecordIndex (see page 5762) can't be incremented.

See Also

NextIteration (see page 1056)

RecordIndex (see page 5762)

RecordCount (see page 5761)

GetNextRecord (see page 5755)

GetEOF (see page 5755)

EndOfPage (see page 5754)

54.1.65.2.15 TDefaultFieldsPagedAdapter.ImplNotifyDeactivate Method

Clears the current page variable.

Pascal

```
procedure ImplNotifyDeactivate; override;
```

C++

```
virtual __fastcall ImplNotifyDeactivate();
```

Description

The current page is reset to zero when the module is deactivated. This ensures that the current page set during the processing of an HTTP request will not affect another HTTP request. To keep track of the current page for a particular end-user, the current page is typically passed between end-user requests using action parameters.

See Also

ImplNotifyDeactivate (see page 5560)

EncodeActionParams (see page 5754)

ExtractRequestParams (see page 5754)

54.1.65.2.16 TDefaultFieldsPagedAdapter.ImplStartIterator Method

Implements IteratorSupport.StartIterator (see page 5567) to support iterating over the records of the adapter.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

ImplStartIterator locates the first record in the current page using the GetFirstRecord (see page 5755) and SkipToPage (see page 5760) methods.

See Also

StartIterator (see page 1056)

ImplNextIteration (see page 5758)

GetFirstRecord (see page 5755)

SkipToPage (see page 5760)

CurrentPage (see page 5760)

54.1.65.2.17 TDefaultFieldsPagedAdapter.SetPageSize Method

Updates the pagesize variable.

Pascal

```
procedure SetPageSize(const Value: Integer);
```

C++

```
__fastcall SetPageSize(const int Value);
```

Description

If the pagesize variable is not zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the pagesize variable is changed back to zero then the these actions are removed.

See Also

PageSize (see page 5761)

TAdapterNextPageAction (see page 5370)

TAdapterPrevPageAction (see page 5396)

TAdapterGotoPageAction (see page 5291)

54.1.65.2.18 TDefaultFieldsPagedAdapter.SkipToPage Method

Skips records until the current record is within the current page.

Pascal

```
procedure SkipToPage;
```

C++

```
__fastcall SkipToPage();
```

Description

SkipToPage calls GetNextRecord (see page 5755) to reach the first record in the current page.

See Also

CurrentPage (see page 5760)

GetNextRecord (see page 5755)

54.1.65.3 TDefaultFieldsPagedAdapter Properties

The properties of the TDefaultFieldsPagedAdapter class are listed here.

Public Properties

	Name	Description
	CurrentPage (see page 5760)	Sets or gets the current page index.
R	PageCount (see page 5761)	Calculates the number of pages needed to display all adapter records.
	PageSize (see page 5761)	Sets or gets the page size variable.
R	RecordCount (see page 5761)	Returns the total number of records.
R	RecordIndex (see page 5762)	Returns the index of the current record.

54.1.65.3.1 TDefaultFieldsPagedAdapter.CurrentPage Property

Sets or gets the current page index.

Pascal

```
property CurrentPage: Integer;
```

C++

```
__property int CurrentPage;
```

Description

Current page is a zero-based index that must be less than PageCount (see page 5761). Current page is used by the NextPage, GotoPage, and PrevPage actions to move between page.

See Also

PageCount (see page 5761)

TAdapterNextPageAction (see page 5370)

TAdapterPrevPageAction (see page 5396)

TAdapterGotoPageAction (see page 5291)

EncodeActionParams (see page 5754)

ExtractRequestParams (see page 5754)

54.1.65.3.2 TDefaultFieldsPagedAdapter.PageCount Property

Calculates the number of pages needed to display all adapter records.

Pascal

```
property PageCount: Integer;
```

C++

```
__property int PageCount;
```

Description

PageCount returns the result of GetPageCount (see page 5756).

See Also

GetPageCount (see page 5756)

RecordCount (see page 5761)

PageSize (see page 5761)

54.1.65.3.3 TDefaultFieldsPagedAdapter.PageSize Property

Sets or gets the page size variable.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

If the page size variable is set to a value greater than zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the page size variable is changed back to zero then the these actions are removed.

See Also

TAdapterNextPageAction (see page 5370)

TAdapterPrevPageAction (see page 5396)

TAdapterGotoPageAction (see page 5291)

54.1.65.3.4 TDefaultFieldsPagedAdapter.RecordCount Property

Returns the total number of records.

Pascal

```
property RecordCount: Integer;
```

C++

```
__property int RecordCount;
```

Description

The RecordCount property returns the result of GetRecordCount (see page 5756).

See Also

GetRecordCount (see page 5756)

54.1.65.3.5 TDefaultFieldsPagedAdapter.RecordIndex Property

Returns the index of the current record.

Pascal

```
property RecordIndex: Integer;
```

C++

```
__property int RecordIndex;
```

Description

RecordIndex returns the result of GetRecordIndex (see page 5756).

See Also

OnGetRecordIndex (see page 5763)

GetRecordIndex (see page 5756)

54.1.65.4 TDefaultFieldsPagedAdapter Events

The events of the TDefaultFieldsPagedAdapter class are listed here.

Public Events

	Name	Description
	OnGetEOF (see page 5762)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5763)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5763)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 5763)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5763)	Implement this event to identify the zero-based current record.

54.1.65.4.1 TDefaultFieldsPagedAdapter.OnGetEOF Event

Implement this event to indicate when the current record index has exceeded the total record count.

Pascal

```
property OnGetEOF: TGetEOFEvent;
```

C++

```
__property TGetEOFEvent OnGetEOF;
```

Description

OnGetEOF is called to determine when to stop the iteration. If OnGetEOF is not implemented, the result of the OnGetRecordCount (see page 5763) event handler will be compared to the result of the OnGetRecordIndex (see page 5763) event handler.

54.1.65.4.2 TDefaultFieldsPagedAdapter.OnGetFirstRecord Event

Implement this method to initialize the iteration index to zero.

Pascal

```
property OnGetFirstRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetFirstRecord;
```

Description

This method is called when the iteration is started. If there are no records then set the EOF flag.

54.1.65.4.3 TDefaultFieldsPagedAdapter.OnGetNextRecord Event

Implement this method to increment the iteration index.

Pascal

```
property OnGetNextRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetNextRecord;
```

Description

This method is called when the next record is requested. This method is not a next then set the EOF flag.

54.1.65.4.4 TDefaultFieldsPagedAdapter.OnGetRecordCount Event

Implement this event to identify the total number of records that can be iterated.

Pascal

```
property OnGetRecordCount: TGetRecordCountEvent;
```

C++

```
__property TGetRecordCountEvent OnGetRecordCount;
```

Description

OnGetRecordCount is called to calculate the number of pages.

54.1.65.4.5 TDefaultFieldsPagedAdapter.OnGetRecordIndex Event

Implement this event to identify the zero-based current record.

Pascal

```
property OnGetRecordIndex: TGetRecordIndexEvent;
```

C++

```
__property TGetRecordIndexEvent OnGetRecordIndex;
```

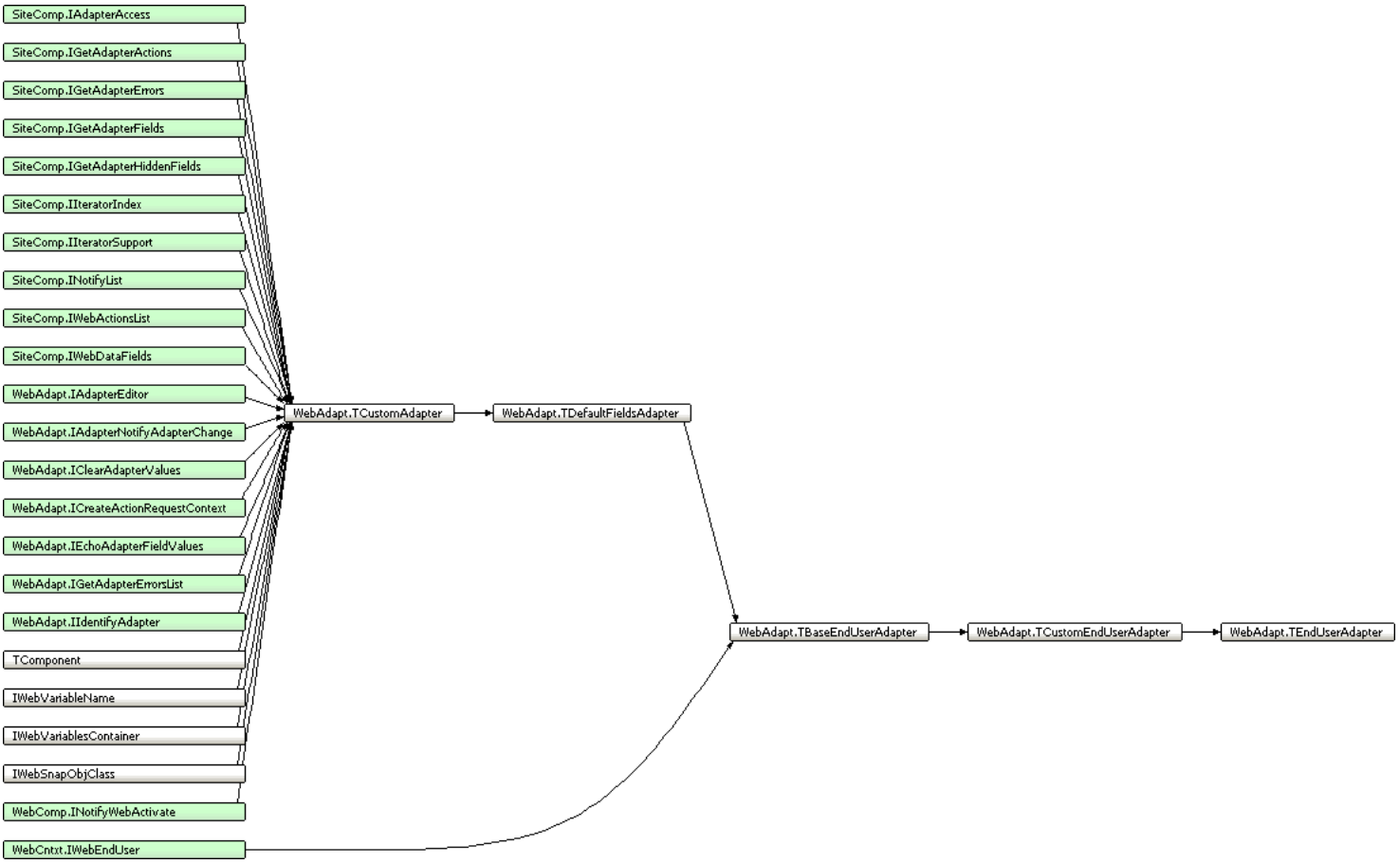
Description

OnGetRecordIndex is called to calculate the current page.

54.1.66 TEndUserAdapter Class

TEndUserAdapter provides information about the current end user such as the user name, user ID, and whether the user is logged in.

Class Hierarchy



Pascal

```
TEndUserAdapter = class(TCustomEndUserAdapter);
```

C++

```
class TEndUserAdapter : public TCustomEndUserAdapter;
```

File

WebAdapt

Description

Add TEndUserSessionAdapter (see page 5792) to a WebSnap application where users must log in before accessing Web pages. TEndUserSessionAdapter (see page 5792) handles login and logout actions on behalf of the end user by generating events where you can retrieve user information and execute user actions

TEndUserAdapter supports two default Adapter fields: DisplayName and LoggedIn. These fields provide access to the user's

name and login state. Implement the OnGetDisplayName (see page 5773) and OnIsLoggedIn (see page 5774) events to provide values for these fields.

TEndUserAdapter contains two default actions: LoginForm and LogoutOut. These actions are used to display a login form and to log the user out. Use the LoginPage (see page 5771) property to specify the name of the login page. Implement the OnLogin (see page 5775) and OnLogout (see page 5775) events to change the login state of an end user.

The EndUser script variable is used to by server-side script to display end user information and execute end user actions. The following JavaScript displays the name of the logged in end user, if any.

```
<% if (EndUser.LoggedIn) { %>
<h1>Welcome <%= EndUser.DisplayName %> </h1>
<% } %>
```

The following JavaScript displays a login or logout hyperlink.

```
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
```

TEndUserAdapter also determines access rights for a user. If the WebSnap application contains a TWebUserList or equivalent, TEndUserAdapter calls the TWebUserList component to check rights for a particular user ID. Implement the OnHasRights (see page 5774) event to check rights in an event handler.

54.1.66.1 TEndUserAdapter Members

The following tables list the members exposed by TEndUserAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5504)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5504)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5505)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	DoOnHasRights (see page 5505)	Determines whether the user has view rights.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExecuteLogin (see page 5505)	Starts the server-side login process.

	ExecuteLogin (see page 5683)	Generates an OnLogin (see page 5686) event.
	ExecuteLoginForm (see page 5506)	Creates the HTML login page.
	ExecuteLogout (see page 5506)	Performs server-side logout.
	ExecuteLogout (see page 5683)	Generates an OnLogout (see page 5687) event.
	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5506)	Retrieves the display name string.
	GetDisplayNameValue (see page 5507)	Gets the display name of the end user.
	GetDisplayNameValue (see page 5683)	Generates an OnGetDisplayName (see page 5685) event.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5507)	Gets the end-user adapter.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5507)	Determines if the user is logged in
	GetLoggedInValue (see page 5508)	Determines if the user is logged in.
	GetLoggedInValue (see page 5684)	Generates an OnIsLoggedIn (see page 5686) event.
	GetLoginFormAction (see page 5508)	Gets the action component for the login form.
	GetLogoutAction (see page 5508)	Gets the action component for the logout form.
	GetUserID (see page 5509)	Gets the UserID from this TBaseEndUserAdapter (see page 5497).
	GetUserID (see page 5684)	Generates an OnGetUserID (see page 5685) event.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasExecuteRights (see page 5509)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5497).
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasModifyRights (see page 5509)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5497).
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.

	HasViewRights (see page 5510)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5497).
	ImplCanAddActionClass (see page 5510)	Determines if the given action class can be added.
	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5510)	Determines if the given action class can be added.
	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5511)	Lists supported end-user adapter actions.
	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
	ImplGetDisplayName (see page 5511)	Gets the display name for this TBaseEndUserAdapter (see page 5497).
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetEndUserAdapter (see page 5511)	Returns this TBaseEndUserAdapter (see page 5497).
	ImplGetFieldsList (see page 5512)	Obtains the list of all of the fields for the end-user adapter.
	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetLoggedIn (see page 5512)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5512)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5497).
	ImplGetLogoutAction (see page 5513)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5497).
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasExecuteRights (see page 5513)	Checks user rights and determines if the user has execute rights.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasModifyRights (see page 5513)	Checks user rights and determines if the user has modify rights.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplHasViewRights (see page 5514)	Checks user rights and determines if the user has view rights.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplLogIn (see page 5514)	Processes the login for the given user.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplProduceLoginPage (see page 5514)	Generate the HTML login page.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.

	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5515)	Processes the server-side login for the given user.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
V	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5515)	Generates the login page.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
V	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
V	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
V	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
V	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
V	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
V	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
V	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5515)	Gets the user list.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetUserID (see page 5685)	Occurs when the adapter needs to provide the ID of the current end user.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.

	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5989)	Returns the value of the DisplayName (see page 5994) property.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5990)	Returns the value of the EndUserAdapter (see page 5994) property.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetLoggedIn (see page 5990)	Returns the value of the LoggedIn (see page 5994) property.
	GetLoginFormAction (see page 5990)	Returns the value of the LoginFormAction (see page 5995) property.
	GetLogoutAction (see page 5991)	Returns the value of the LogoutAction (see page 5996) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5991)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5992)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5992)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5993)	Logs in a specified user.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5993)	Generates the Web page for logging in users.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	DisplayName (see page 5994)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5994)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	LoggedIn (see page 5994)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5995)	Indicates the action component for the login form.
R	LogoutAction (see page 5996)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.

	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5771)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5772)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5772)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5772)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5773)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 5773)	Occurs when the adapter needs to provide the name of the current end user.
	OnHasRights (see page 5774)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 5774)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 5775)	Logs in the current end user.
	OnLogout (see page 5775)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 5776)	Allows implementation of a customized login page.

Published Properties

	Name	Description
	Actions (see page 5770)	Container for adapter actions components.
	Data (see page 5770)	Container for adapter field components.
	LoginPage (see page 5771)	Specifies the name of the page that acts as the login page.

54.1.66.2 TEndUserAdapter Properties

The properties of the TEndUserAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5770)	Container for adapter actions components.
	Data (see page 5770)	Container for adapter field components.
	LoginPage (see page 5771)	Specifies the name of the page that acts as the login page.

54.1.66.2.1 TEndUserAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5231)

54.1.66.2.2 TEndUserAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5237)

54.1.66.2.3 TEndUserAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property UnicodeString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter. The name of the login page is the name of the Web page module that generates the login page.

54.1.66.3 TEndUserAdapter Events

The events of the TEndUserAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5771)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5772)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5772)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5772)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5773)	Supplies name/value pairs needed to execute an adapter action.
	OnGetDisplayName (see page 5773)	Occurs when the adapter needs to provide the name of the current end user.
	OnHasRights (see page 5774)	Determines if the user rights match the rights being queried.
	OnIsLoggedIn (see page 5774)	Occurs when the adapter needs to indicate whether the current end user has logged in.
	OnLogin (see page 5775)	Logs in the current end user.
	OnLogout (see page 5775)	Occurs when the current user logs out.
	OnProduceLoginPage (see page 5776)	Allows implementation of a customized login page.

54.1.66.3.1 TEndUserAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.66.3.2 TEndUserAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.66.3.3 TEndUserAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5478)

OnAfterExecuteAction (see page 5576)

54.1.66.3.4 TEndUserAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5478)

OnAfterGetActionResponse (see page 5576)

54.1.66.3.5 TEndUserAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5478)

54.1.66.3.6 TEndUserAdapter.OnGetDisplayName Event

Occurs when the adapter needs to provide the name of the current end user.

Pascal

```
property OnGetDisplayName: TEndUserGetNameEvent;
```

C++

```
__property TEndUserGetNameEvent OnGetDisplayName;
```

Description

Write an OnGetDisplayName event handler to provide the value of the DisplayName field. When a server-side script (or a caller of the IWebEndUser interface) requests the display name for the current end user, TCustomEndUserAdapter generates an OnGetDisplayName event to retrieve that value.

Note: If you do not supply an OnGetDisplayName event handler, TCustomEndUserAdapter always returns an empty string as the display name for the current end user.

See Also

DisplayName (see page 5994)

EndUser (see page 5973)

WebContext (see page 6003)

54.1.66.3.7 TEndUserAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 5505)

ImplHasExecuteRights (see page 5513)

ImplHasModifyRights (see page 5513)

ImplHasViewRights (see page 5514)

54.1.66.3.8 TEndUserAdapter.OnIsLoggedIn Event

Occurs when the adapter needs to indicate whether the current end user has logged in.

Pascal

```
property OnIsLoggedIn: TEndUserIsLoggedInEvent;
```

C++

```
__property TEndUserIsLoggedInEvent OnIsLoggedIn;
```

Description

Write an OnIsLoggedIn event handler to provide the value of the LoggedIn field. When a server-side script (or a caller of the IWebEndUser interface) inquires whether the current user has logged in, TCustomEndUserAdapter generates an OnIsLoggedIn event to retrieve that information.

Note: If you do not supply an OnIsLoggedIn event handler, TCustomEndUserAdapter always returns false, indicating that the current end user has not logged in.

See Also

LoggedIn (see page 5994)

EndUser (see page 5973)

WebContext (see page 6003)

OnLogin (see page 5686)

OnLogout (see page 5687)

54.1.66.3.9 TEndUserAdapter.OnLogin Event

Logs in the current end user.

Pascal

```
property OnLogin: TEndUserUserIDEvent;
```

C++

```
__property TEndUserUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to allow the TCustomEndUserAdapter object to log users in. When a server-side script (or a caller of the IWebEndUser interface) attempts to log in a user, TCustomEndUserAdapter generates an OnLogin event to perform whatever steps are involved in the login process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter does nothing to log users in.

See Also

LogIn (see page 5993)

EndUser (see page 5973)

WebContext (see page 6003)

OnIsLoggedIn (see page 5686)

OnLogout (see page 5687)

TWebUserList (see page 6796)

OnGetUserID (see page 5685)

54.1.66.3.10 TEndUserAdapter.OnLogout Event

Occurs when the current user logs out.

Pascal

```
property OnLogout: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnLogout;
```

Description

Write an OnLogout event handler to allow the TCustomEndUserAdapter object to log users out. When a server-side script attempts to log a user out, TCustomEndUserAdapter generates an OnLogout event to perform whatever steps are involved in the logout process.

Note: If you do not supply an OnLogin event handler, TCustomEndUserAdapter does nothing to log users out.

See Also

OnIsLoggedIn (see page 5686)

OnLogin (see page 5686)

54.1.66.3.11 TEndUserAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage. If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 5514)

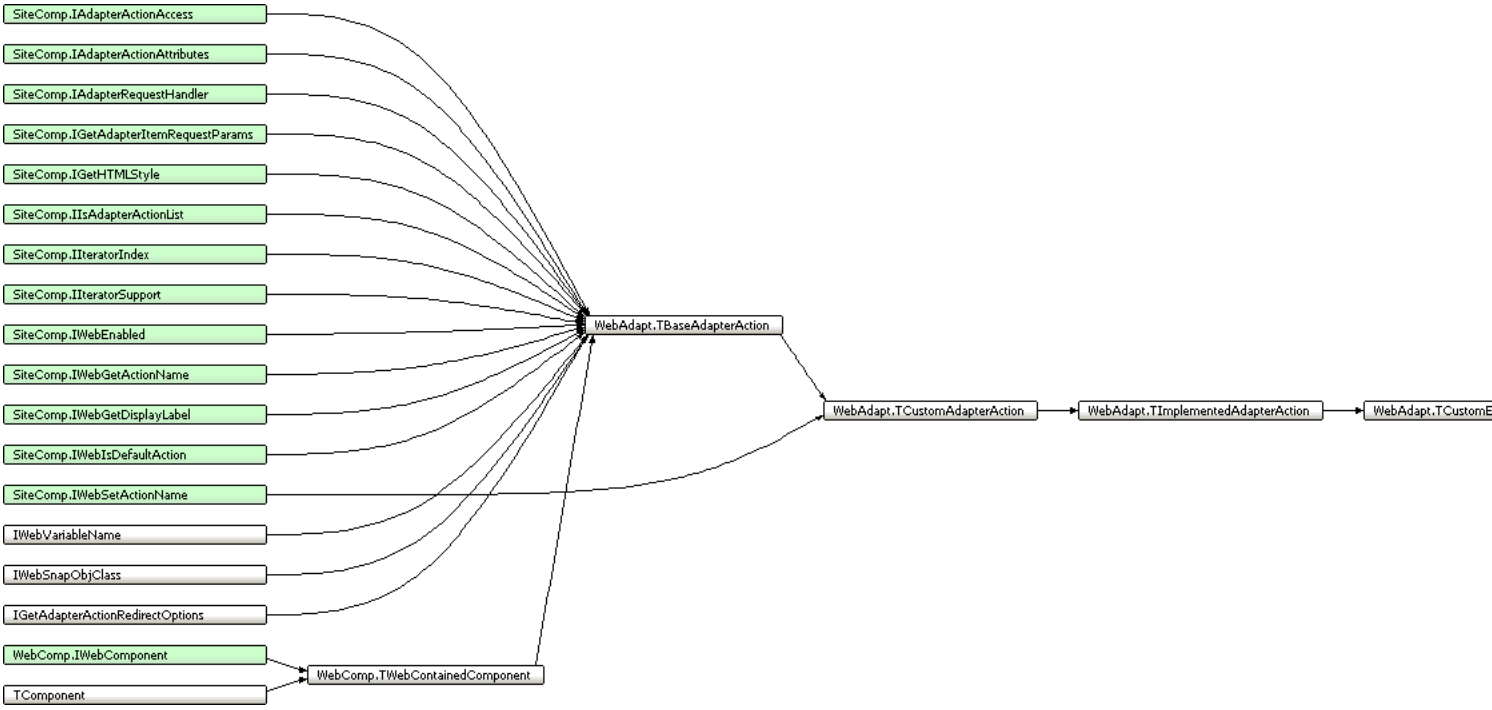
ProduceLoginPage (see page 5515)

CanProduceLoginPage (see page 5505)

54.1.67 TEndUserAdapterLoginFormAction Class

TEndUserAdapterLoginFormAction is an adapter action that displays a login form when executed.

Class Hierarchy



Pascal

```
TEndUserAdapterLoginFormAction = class(TCustomEndUserAdapterAction);
```

C++

```
class TEndUserAdapterLoginFormAction : public TCustomEndUserAdapterAction;
```

File

WebAdapt

Description

This adapter action is typically executed when the end-user clicks on a "Login" hyperlink displayed at the top of a web page. The parent adapter's LoginPage property is used to specify the page displayed when this adapter action is executed.

See Also

TCustomEndUserAdapterAction (see page 5687)

TBaseEndUserAdapter (see page 5497)

TLoginFormAdapter (see page 5824)

54.1.67.1 TEndUserAdapterLoginFormAction Members

The following tables list the members exposed by TEndUserAdapterLoginFormAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5780)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.

	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
	ImplExecuteActionRequest (see page 5781)	Executes the adapter action.
	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
	ImplWebEnabled (see page 5781)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5474)	Provides access to the parent adapter component.
R	Adapter (see page 5692)	Returns the parent adapter of this adapter action.
R	Errors (see page 5475)	Returns the parent adapter's Errors list.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
R	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
R	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.

	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5782)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5783)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5783)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5783)	Called before the adapter action response is generated.
	OnGetParams (see page 5784)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5782)	Suggests a display label for this adapter action.

54.1.67.2 TEndUserAdapterLoginFormAction Methods

The methods of the TEndUserAdapterLoginFormAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5780)	Gets the name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 5781)	Executes the adapter action.
	ImplWebEnabled (see page 5781)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

54.1.67.2.1 TEndUserAdapterLoginFormAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "LoginForm".

See Also

GetActionName (see page 5455)

GetDefaultActionName (see page 5456)

54.1.67.2.2 TEndUserAdapterLoginFormAction.ImplExecuteActionRequest Method

Executes the adapter action.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest calls the parent adapter's ExecuteLogin method. If an exception is raised, the exception message is added to the parent adapter's errors object.

See Also

IActionRequest

IActionResponse

ImplExecuteActionRequest (see page 5464)

ExecuteLogin (see page 5505)

Errors (see page 5573)

54.1.67.2.3 TEndUserAdapterLoginFormAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of the parent adapter's CanExecuteLoginForm function.

See Also

CanExecuteLoginForm (see page 5504)

WebEnabled (see page 5473)

54.1.67.3 TEndUserAdapterLoginFormAction Properties

The properties of the TEndUserAdapterLoginFormAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5782)	Suggests a display label for this adapter action.

54.1.67.3.1 TEndUserAdapterLoginFormAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5465)

SetDisplayLabel (see page 5472)

54.1.67.4 TEndUserAdapterLoginFormAction Events

The events of the TEndUserAdapterLoginFormAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5782)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5783)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5783)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5783)	Called before the adapter action response is generated.
	OnGetParams (see page 5784)	Specifies name/value pairs needed to execute this adapter action.

54.1.67.4.1 TEndUserAdapterLoginFormAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.67.4.2 TEndUserAdapterLoginFormAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.67.4.3 TEndUserAdapterLoginFormAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.67.4.4 TEndUserAdapterLoginFormAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains

the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.67.4.5 TEndUserAdapterLoginFormAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

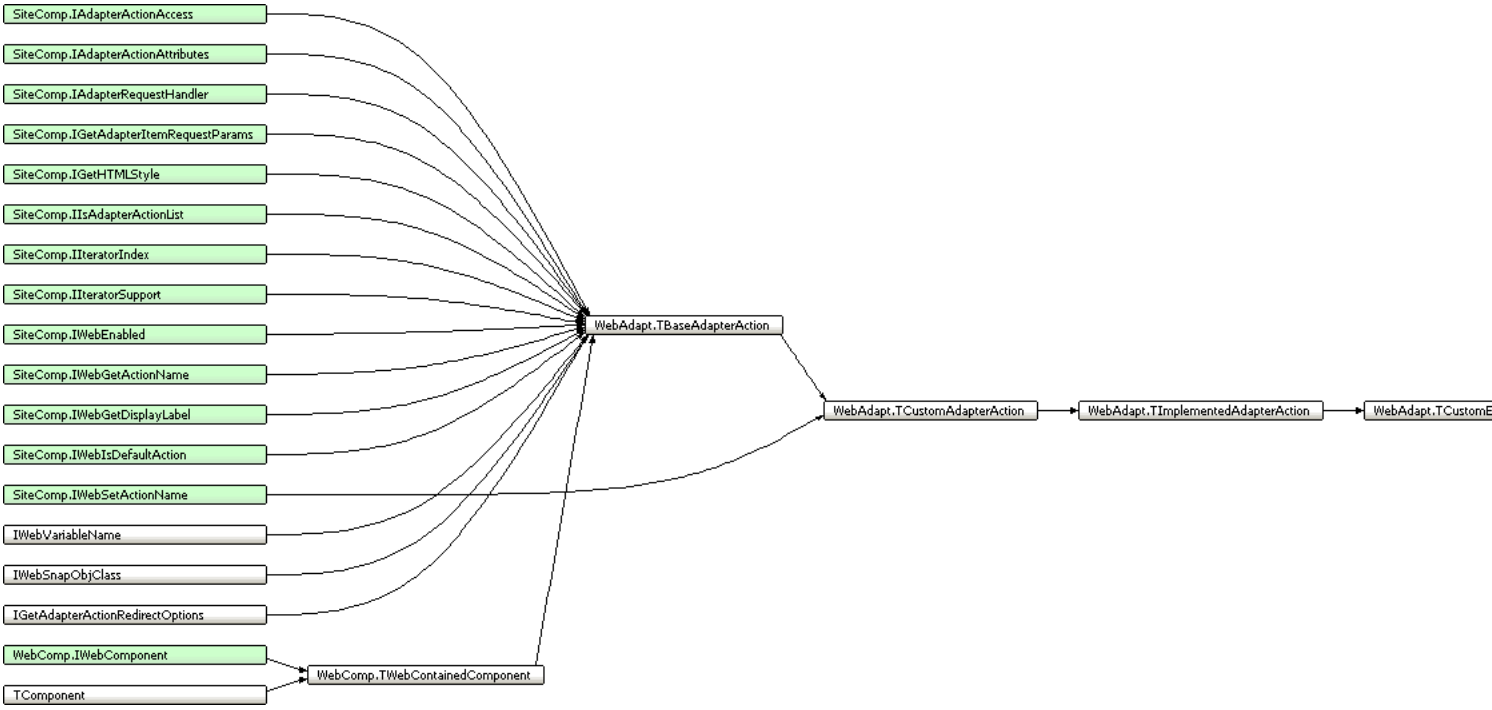
See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.68 TEndUserAdapterLogoutAction Class

TEndUserAdapterLogoutAction is an adapter action that logs the end-user out.

Class Hierarchy



Pascal

```
TEndUserAdapterLogoutAction = class(TCustomEndUserAdapterAction);
```

C++

```
class TEndUserAdapterLogoutAction : public TCustomEndUserAdapterAction;
```

File

WebAdapt

Description

This action is typically executed when the end-user clicks on a "Logout" hyperlink displayed at the top of a web page.

54.1.68.1 TEndUserAdapterLogoutAction Members

The following tables list the members exposed by TEndUserAdapterLogoutAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.

	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5789)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5789)	Executes the adapter action.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5465)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetAdapterItemRequestParams (see page 5584)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
V	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
V	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.

	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
V	ImplWebEnabled (see page 5789)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
V	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5474)	Provides access to the parent adapter component.
R	Adapter (see page 5692)	Returns the parent adapter of this adapter action.
R	Errors (see page 5475)	Returns the parent adapter's Errors list.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.

	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5791)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5791)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5791)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5792)	Called before the adapter action response is generated.
	OnGetParams (see page 5792)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5790)	Suggests a display label for this adapter action.

54.1.68.2 TEndUserAdapterLogoutAction Methods

The methods of the TEndUserAdapterLogoutAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5789)	Gets the name used to reference this adapter action with server-side script.

	ImplExecuteActionRequest (see page 5789)	Executes the adapter action.
	ImplWebEnabled (see page 5789)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

54.1.68.2.1 TEndUserAdapterLogoutAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "Logout".

See Also

GetActionName (see page 5455)

GetDefaultActionName (see page 5456)

54.1.68.2.2 TEndUserAdapterLogoutAction.ImplExecuteActionRequest Method

Executes the adapter action.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest calls the parent adapter's ExecuteLogout method. If an exception is raised, the exception message is added to the parent adapter's errors object.

See Also

IActionRequest

IActionResponse

ImplExecuteActionRequest (see page 5464)

ExecuteLogout (see page 5506)

Errors (see page 5573)

54.1.68.2.3 TEndUserAdapterLogoutAction.ImplWebEnabled Method

Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Pascal

```
function ImplWebEnabled: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplWebEnabled();
```

Description

ImplWebEnabled returns the result of the parent adapter's CanExecuteLogout function.

See Also

CanExecuteLogout (see page 5504)

WebEnabled (see page 5473)

54.1.68.3 TEndUserAdapterLogoutAction Properties

The properties of the TEndUserAdapterLogoutAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5790)	Suggests a display label for this adapter action.

54.1.68.3.1 TEndUserAdapterLogoutAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5465)

SetDisplayLabel (see page 5472)

54.1.68.4 TEndUserAdapterLogoutAction Events

The events of the TEndUserAdapterLogoutAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5791)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5791)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5791)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5792)	Called before the adapter action response is generated.
	OnGetParams (see page 5792)	Specifies name/value pairs needed to execute this adapter action.

54.1.68.4.1 TEndUserAdapterLogoutAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.68.4.2 TEndUserAdapterLogoutAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.68.4.3 TEndUserAdapterLogoutAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action

parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.68.4.4 TEndUserAdapterLogoutAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.68.4.5 TEndUserAdapterLogoutAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

See Also

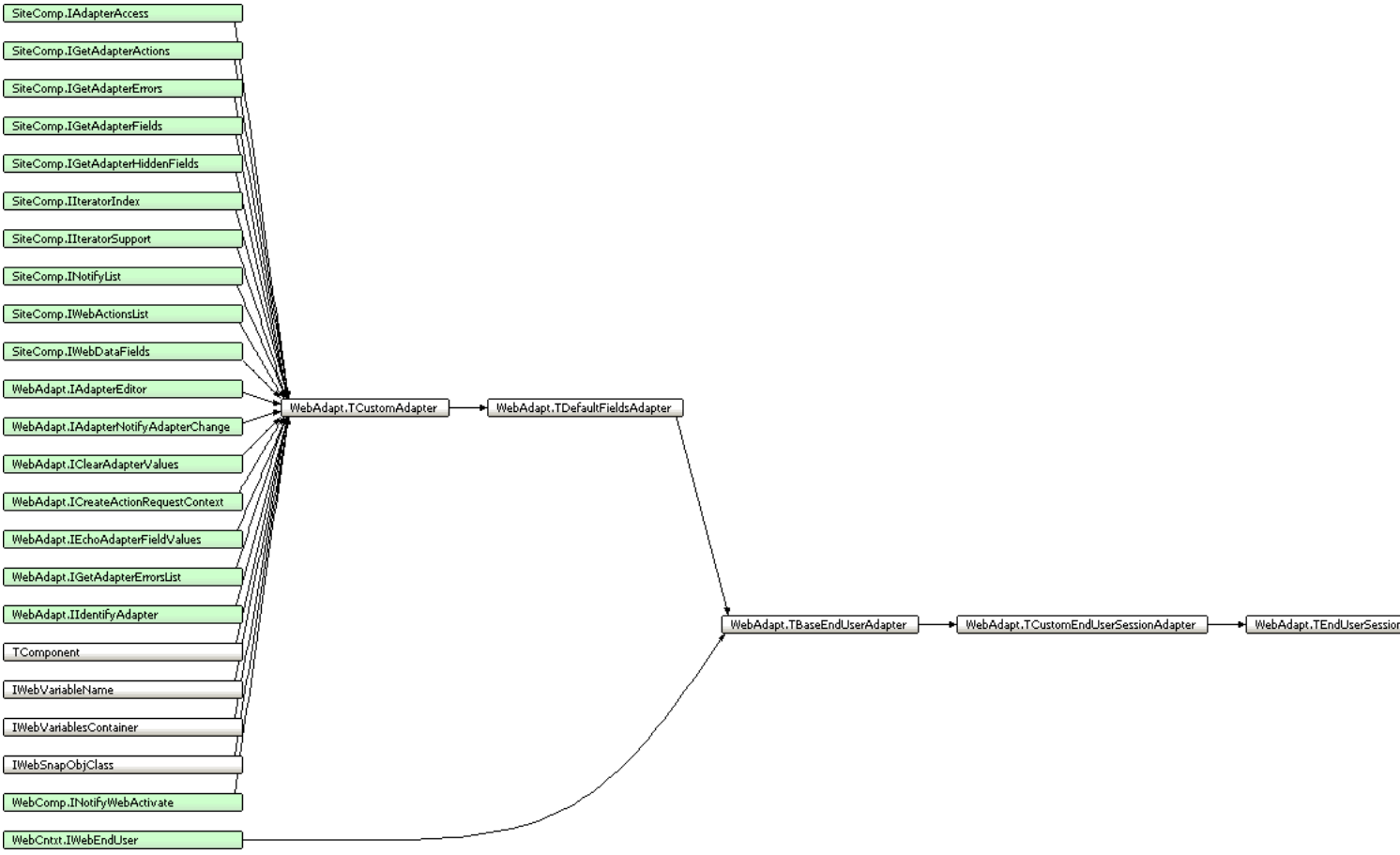
ImplGetAdapterItemRequestParams (see page 5465)

54.1.69 TEndUserSessionAdapter Class

TEndUserSessionAdapter provides information about the current end user such as the user name, user ID, and whether the user

is logged in.

Class Hierarchy



Pascal

```
TEndUserSessionAdapter = class(TCustomEndUserSessionAdapter);
```

C++

```
class TEndUserSessionAdapter : public TCustomEndUserSessionAdapter;
```

File

WebAdapt

Description

Add TEndUserSessionAdapter to a WebSnap application where users must log in before accessing Web pages. TEndUserSessionAdapter handles login and logout actions on behalf of the end user. TEndUserSessionAdapter uses the TSessionsService component to store information about the current end user. The WebSnap application must contain a TSessionsService component (or equivalent) in order for the TEndUserSessionAdapter to work properly.

TEndUserSessionAdapter supports two default Adapter fields: DisplayName and LoggedIn. These fields provide access to the user's name and login state. TEndUserSessionAdapter retrieves the values for these fields from a Web session, which is maintained by the TSessionsService component.

TEndUserSessionAdapter supports two default actions: LoginForm and LogoutOut. These actions are used to display a login form and to log the user out. Use the LoginPage (see page 5800) property to specify the name of the login page. The default

implementation of the logout action terminates the session.

The EndUser script variable is used by server-side script to display end user information and execute end user actions. The following JavaScript displays the name of the logged in end user, if any.

```
<% if (EndUser.LoggedIn) { %>
<h1>Welcome <%= EndUser.DisplayName %> </h1>
<% } %>
```

The following JavaScript displays a login or logout hyperlink.

```
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
```

TEndUserSessionAdapter also determines access rights for a user. If the WebSnap application contains a TWebUserList or equivalent, TEndUserAdapter (see page 5764) calls the TWebUserList component to check rights for a particular user ID. Implement the OnHasRights (see page 5802) event to check rights in an event handler.

See Also

TEndUserAdapter (see page 5764)

TSessionsService (see page 6756)

TLoginFormAdapter (see page 5824)

TWebUserList (see page 6796)

IWebEndUser (see page 5988)

Adding Login Support

54.1.69.1 TEndUserSessionAdapter Members

The following tables list the members exposed by TEndUserSessionAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
V	CanExecuteLoginForm (see page 5504)	Determines if there is a static login page.
V	CanExecuteLogout (see page 5504)	Determines if the user is logged in.
V	CanProduceLoginPage (see page 5505)	Determines whether there is a static login page or an automatically produced login page.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.

	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
	DoOnHasRights (see page 5505)	Determines whether the user has view rights.
	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
	ExecuteLogin (see page 5505)	Starts the server-side login process.
	ExecuteLogin (see page 5699)	Logs in an end user.
	ExecuteLoginForm (see page 5506)	Creates the HTML login page.
	ExecuteLogout (see page 5506)	Performs server-side logout.
	ExecuteLogout (see page 5699)	Logs out the current user.
	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetDisplayName (see page 5506)	Retrieves the display name string.
	GetDisplayNameValue (see page 5507)	Gets the display name of the end user.
	GetDisplayNameValue (see page 5699)	Returns the name of the current end user.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetEndUserAdapter (see page 5507)	Gets the end-user adapter.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetLoggedIn (see page 5507)	Determines if the user is logged in
	GetLoggedInValue (see page 5508)	Determines if the user is logged in.
	GetLoggedInValue (see page 5700)	Indicates whether the current end user has logged in.
	GetLoginFormAction (see page 5508)	Gets the action component for the login form.
	GetLogoutAction (see page 5508)	Gets the action component for the logout form.
	GetUserID (see page 5509)	Gets the UserID from this TBaseEndUserAdapter (see page 5497).
	GetUserID (see page 5700)	Returns the user ID of the current user.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.

	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasExecuteRights (see page 5509)	Determines if the user has execute rights for the TBaseEndUserAdapter (see page 5497).
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasModifyRights (see page 5509)	Determines if the user has modify rights for the TBaseEndUserAdapter (see page 5497).
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	HasViewRights (see page 5510)	Determine if the user has view rights to the TBaseEndUserAdapter (see page 5497).
V	ImplCanAddActionClass (see page 5510)	Determines if the given action class can be added.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5510)	Determines if the given action class can be added.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5511)	Lists supported end-user adapter actions.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetDisplayName (see page 5511)	Gets the display name for this TBaseEndUserAdapter (see page 5497).
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetEndUserAdapter (see page 5511)	Returns this TBaseEndUserAdapter (see page 5497).
V	ImplGetFieldsList (see page 5512)	Obtains the list of all of the fields for the end-user adapter.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetLoggedIn (see page 5512)	Checks the value of the LoggedIn field.
	ImplGetLoginFormAction (see page 5512)	Gets the LoginForm action component for this TBaseEndUserAdapter (see page 5497).
V	ImplGetLogoutAction (see page 5513)	Gets the Logout action component for this TBaseEndUserAdapter (see page 5497).
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasExecuteRights (see page 5513)	Checks user rights and determines if the user has execute rights.
V	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasModifyRights (see page 5513)	Checks user rights and determines if the user has modify rights.
V	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
V	ImplHasViewRights (see page 5514)	Checks user rights and determines if the user has view rights.
V	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
V	ImplLogIn (see page 5514)	Processes the login for the given user.
V	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
V	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
V	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
V	ImplProduceLoginPage (see page 5514)	Generate the HTML login page.
V	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
V	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
V	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.

	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	LogIn (see page 5515)	Processes the server-side login for the given user.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	ProduceLoginPage (see page 5515)	Generates the login page.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5515)	Gets the user list.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).

	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetDisplayName (see page 5989)	Returns the value of the DisplayName (see page 5994) property.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetEndUserAdapter (see page 5990)	Returns the value of the EndUserAdapter (see page 5994) property.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetLoggedIn (see page 5990)	Returns the value of the LoggedIn (see page 5994) property.
	GetLoginFormAction (see page 5990)	Returns the value of the LoginFormAction (see page 5995) property.
	GetLogoutAction (see page 5991)	Returns the value of the LogoutAction (see page 5996) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasExecuteRights (see page 5991)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasModifyRights (see page 5992)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	HasViewRights (see page 5992)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	LogIn (see page 5993)	Logs in a specified user.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	ProduceLoginPage (see page 5993)	Generates the Web page for logging in users.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	DisplayName (see page 5994)	Indicates the name of the end user as it should be used for display in Web pages.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	EndUserAdapter (see page 5994)	Provides access to the end user adapter that implements this interface.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.

	LoggedIn (see page 5994)	Indicates whether the current user is logged in.
	LoginFormAction (see page 5995)	Indicates the action component for the login form.
	LogoutAction (see page 5996)	Indicates the action component for logging a user out.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	UserID (see page 5701)	Specifies the user ID of the current end user.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5800)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5801)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5801)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5801)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5802)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5802)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 5803)	Allows implementation of a customized login page.

Published Properties

	Name	Description
	Actions (see page 5799)	Container for adapter actions components.
	Data (see page 5800)	Container for adapter field components.
	LoginPage (see page 5800)	Specifies the name of the page that acts as the login page.

54.1.69.2 TEndUserSessionAdapter Properties

The properties of the TEndUserSessionAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5799)	Container for adapter actions components.
	Data (see page 5800)	Container for adapter field components.
	LoginPage (see page 5800)	Specifies the name of the page that acts as the login page.

54.1.69.2.1 TEndUserSessionAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5231)

54.1.69.2.2 TEndUserSessionAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5237)

54.1.69.2.3 TEndUserSessionAdapter.LoginPage Property

Specifies the name of the page that acts as the login page.

Pascal

```
property LoginPage: string;
```

C++

```
__property UnicodeString LoginPage;
```

Description

Use LoginPage to read or set the name of the login page for this TBaseEndUserAdapter. The name of the login page is the name of the Web page module that generates the login page.

54.1.69.3 TEndUserSessionAdapter Events

The events of the TEndUserSessionAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5800)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5801)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5801)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5801)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5802)	Supplies name/value pairs needed to execute an adapter action.
	OnHasRights (see page 5802)	Determines if the user rights match the rights being queried.
	OnProduceLoginPage (see page 5803)	Allows implementation of a customized login page.

54.1.69.3.1 TEndUserSessionAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.69.3.2 TEndUserSessionAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.69.3.3 TEndUserSessionAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5478)

OnAfterExecuteAction (see page 5576)

54.1.69.3.4 TEndUserSessionAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5478)

OnAfterGetActionResponse (see page 5576)

54.1.69.3.5 TEndUserSessionAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5478)

54.1.69.3.6 TEndUserSessionAdapter.OnHasRights Event

Determines if the user rights match the rights being queried.

Pascal

```
property OnHasRights: TEndUserRightsEvent;
```

C++

```
__property TEndUserRightsEvent OnHasRights;
```

Description

The OnHasRights event handler is called from DoOnHasRights to determine if the user rights match the rights being queried. A customized event handler should be written to determine the rights for your specific application.

See Also

DoOnHasRights (see page 5505)

ImplHasExecuteRights (see page 5513)

ImplHasModifyRights (see page 5513)

ImplHasViewRights (see page 5514)

54.1.69.3.7 TEndUserSessionAdapter.OnProduceLoginPage Event

Allows implementation of a customized login page.

Pascal

```
property OnProduceLoginPage: TEndUserHandledEvent;
```

C++

```
__property TEndUserHandledEvent OnProduceLoginPage;
```

Description

The OnProduceLoginPage event is triggered in the ImplProduceLoginPage. If a customized login page is desired for your application, a customized event handler can be written for this event.

See Also

ImplProduceLoginPage (see page 5514)

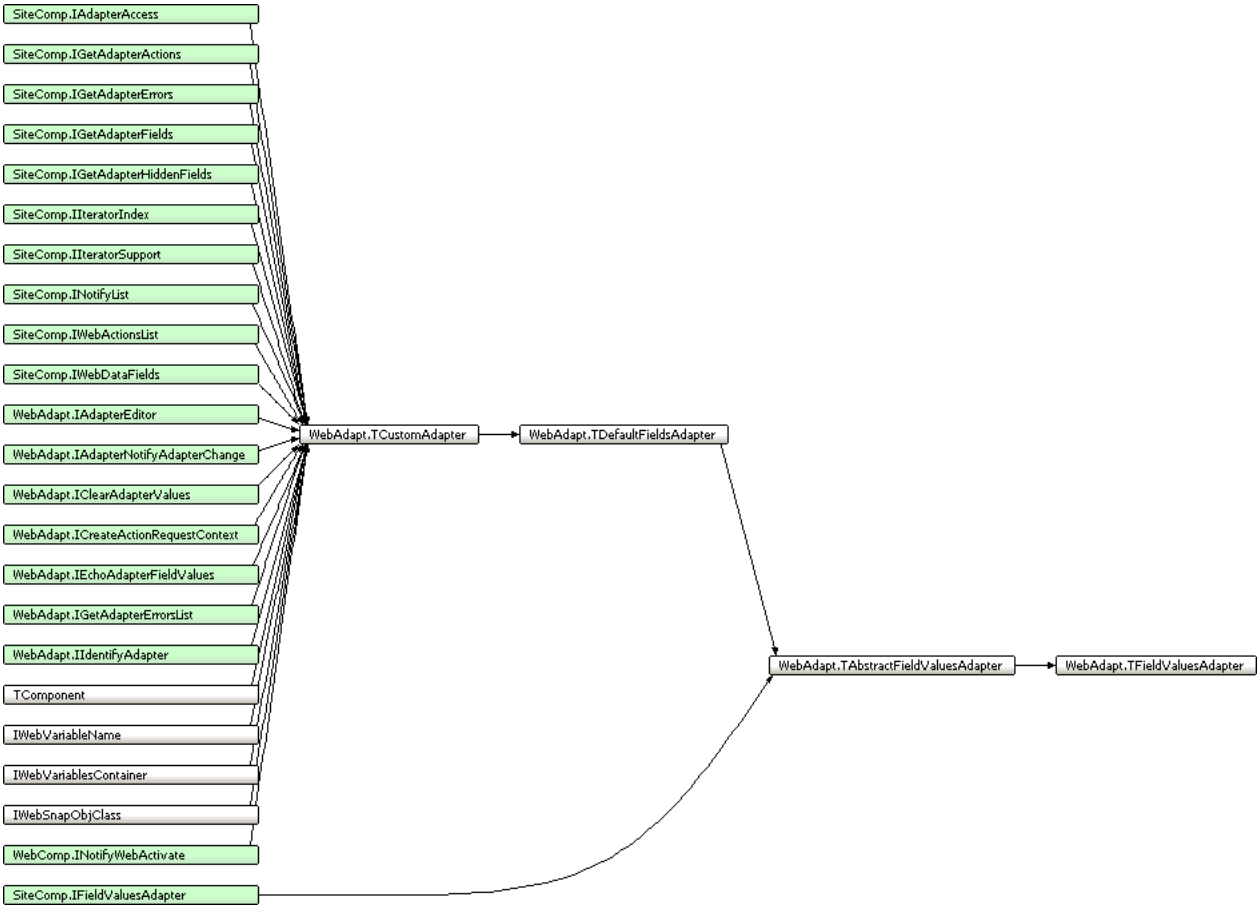
ProduceLoginPage (see page 5515)

CanProduceLoginPage (see page 5505)

54.1.70 TFieldValuesAdapter Class

TFieldValuesAdapter is an adapter that includes a single field value.

Class Hierarchy



Pascal

```
TFieldValuesAdapter = class(TAbstractFieldValuesAdapter);
```

C++

```
class TFieldValuesAdapter : public TAbstractFieldValuesAdapter;
```

File

WebAdapt

Description

Use TFieldValuesAdapter to represent a single field value, which is available as a field with the name 'ValueField'. The single field is represented by an adapter field object that is of type TFieldValuesValueField (see page 5813), or one of its descendants. This adapter field object is supplied to the TFieldValuesAdapter constructor.

The 'ValueField' field can have multiple values, such as when it represents a value in the column of a table. For example, the following server-side script iterates over the values of the 'ValueField' field and adds them to a table:

```
<table border=1>
<%
   var e = new Enumerator(FieldValuesAdapter1)
   for (; !e.atEnd(); e.moveNext())
   {
%>
```

```
  <tr><td><%=e.item().ValueField%></td></tr>
<%
   }
%>
</table>
```

See Also

TStringsValuesList (see page 5857)

TPagedAdapter (see page 5844)

TDataSetValuesList

TFieldValuesValueField (see page 5813)

54.1.70.1 TFieldValuesAdapter Members

The following tables list the members exposed by TFieldValuesAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetField (see page 5186)	Returns the object that represents the single field of this adapter.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
V	GetValueFieldClass (see page 5809)	Returns the class of the object that represents this adapter's ValueField field.
	GetValueIndex (see page 5187)	Returns an index that identifies the current adapter field value.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5187)	Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5810)	Provides the underlying implementation for the EndIterator (see page 5542) method.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetField (see page 5810)	Provides the underlying implementation for the GetField (see page 5186) method.
V	ImplGetFieldsList (see page 5188)	Provides the underlying implementation for the GetFieldsList (see page 5549) method.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetValueIndex (see page 5811)	Provides the underlying implementation for the GetValueIndex (see page 5187) method.
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
V	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
V	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
V	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
V	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
V	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
V	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
V	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
V	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.

	ImplNextIteration (see page 5811)	Provides the underlying implementation for the NextIteration (see page 5564) method.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5812)	Provides the underlying implementation for the StartIterator (see page 5567) method.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	Field (see page 5812)	Indicates the object that represents this adapter's single field.
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.

	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5809)	Creates and initializes an instance of TFieldValuesAdapter (see page 5803).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetField (see page 1016)	This is GetField, a member of class IFieldValuesAdapter.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetValueIndex (see page 1016)	This is GetValueIndex, a member of class IFieldValuesAdapter.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.
R	Actions (see page 5746)	Container for adapter actions components.
R	Data (see page 5572)	Data is the container for adapter field components.
R	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.

	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.70.2 TFieldValuesAdapter.Create Constructor

Creates and initializes an instance of TFieldValuesAdapter (see page 5803).

Pascal

```
constructor Create(AOwner: TComponent; AField: TObject); virtual; reintroduce;
```

C++

```
virtual __fastcall TFieldValuesAdapter(TComponent * AOwner, TObject * AField);
```

Description

Call Create to Create an instance of TFieldValuesAdapter (see page 5803) at runtime.

AOwner is the component that becomes the value of this adapter's Owner property. Typically, this is the Web module. The owner is responsible for freeing the TFieldValuesAdapter (see page 5803) object.

AField is the object that represents the 'ValueField' field of this adapter. It becomes the value of the protected Field (see page 5812) property.

See Also

Owner

Field (see page 5812)

54.1.70.3 TFieldValuesAdapter Methods

The methods of the TFieldValuesAdapter class are listed here.

Protected Methods

	Name	Description
	GetValueFieldClass (see page 5809)	Returns the class of the object that represents this adapter's ValueField field.
	ImplEndIterator (see page 5810)	Provides the underlying implementation for the EndIterator (see page 5542) method.
	ImplGetField (see page 5810)	Provides the underlying implementation for the GetField (see page 5186) method.
	ImplGetValueIndex (see page 5811)	Provides the underlying implementation for the GetValueIndex (see page 5187) method.
	ImplNextIteration (see page 5811)	Provides the underlying implementation for the NextIteration (see page 5564) method.
	ImplStartIterator (see page 5812)	Provides the underlying implementation for the StartIterator (see page 5567) method.

54.1.70.3.1 TFieldValuesAdapter.GetValueFieldClass Method

Returns the class of the object that represents this adapter's ValueField field.

Pascal

```
function GetValueFieldClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetValueFieldClass();
```

Description

GetValueFieldClass returns the class reference for the type of the single adapter field (ValueField) that is the value of the Field (see page 5812) property. This is TFieldValuesValueField (see page 5813).

See Also

Field (see page 5812)

TFieldValuesValueField (see page 5813)

54.1.70.3.2 TFieldValuesAdapter.ImplEndIterator Method

Provides the underlying implementation for the EndIterator (see page 5542) method.

Pascal

```
procedure ImplEndIterator(OwnerData: Pointer); override;
```

C++

```
virtual __fastcall ImplEndIterator(void * OwnerData);
```

Description

The EndIterator (see page 5542) method calls ImplEndIterator to stop the process of iterating through the adapter's field's possible values. This provides the underlying support that allows server-side script to iterate through adapter field values.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Each adapter can interpret this value in its own way.

ImplEndIterator obtains an IWebGetFieldValues (see page 1092) interface for the adapter field that is the value of the Field (see page 5812) property and calls its EndValueIterator method, passing on the value of the OwnerData parameter.

See Also

EndIterator (see page 5542)

ImplStartIterator (see page 5812)

ImplNextIteration (see page 5811)

EndValueIterator (see page 1093)

Field (see page 5812)

54.1.70.3.3 TFieldValuesAdapter.ImplGetField Method

Provides the underlying implementation for the GetField (see page 5186) method.

Pascal

```
function ImplGetField: TObject; override;
```

C++

```
virtual __fastcall TObject * ImplGetField();
```

Description

The GetField (see page 5186) method calls ImplGetField to obtain the object that represents this adapter's single field. This method returns the value of the Field (see page 5812) property.

See Also

GetField (see page 5186)

Field (see page 5812)

54.1.70.3.4 TFieldValuesAdapter.ImplGetValueIndex Method

Provides the underlying implementation for the GetValueIndex (see page 5187) method.

Pascal

```
function ImplGetValueIndex: Integer; override;
```

C++

```
virtual __fastcall int ImplGetValueIndex();
```

Description

The GetValueIndex (see page 5187) method calls ImplGetValueIndex to obtain the index of current value of the ValueField field. The first value has an index of 0, the second an index of 1, and so on.

This method returns the index of the current value, as indicated by the index that the GetIteratorIndex (see page 5550) method returns.

See Also

GetValueIndex (see page 5187)

GetIteratorIndex (see page 5550)

54.1.70.3.5 TFieldValuesAdapter.ImplNextIteration Method

Provides the underlying implementation for the NextIteration (see page 5564) method.

Pascal

```
function ImplNextIteration(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplNextIteration(void * OwnerData);
```

Description

The NextIteration (see page 5564) method calls ImplNextIteration when iterating over field values to determine whether there is another field value after the current one and if so, to change the current value to the next one.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. On entry, this value is typically the same as the value returned by the last call to NextIteration (see page 5564) or to StartIterator (see page 5567).

ImplNextIteration returns true if it moves to the next field value, false if it can't change to a new field value (usually because the field is on its last value).

In TFieldValuesAdapter (see page 5803), ImplNextIteration obtains an IWebGetFieldValues (see page 1092) interface for the adapter field that is the value of the Field (see page 5812) property and calls its NextValueIteration method, passing on the value of the OwnerData parameter.

See Also

NextIteration (see page 5564)

ImplStartIterator (see page 5812)

ImplEndIterator (see page 5810)

NextValueIteration (see page 1094)

54.1.70.3.6 TFieldValuesAdapter.ImplStartIterator Method

Provides the underlying implementation for the StartIterator (see page 5567) method.

Pascal

```
function ImplStartIterator(var OwnerData: Pointer): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplStartIterator(void * OwnerData);
```

Description

The StartIterator (see page 5567) method calls ImplStartIterator to start the process of iterating through the adapter's field's possible values. The return value indicates whether the adapter's field has any values over which to iterate.

OwnerData is a pointer to adapter-specific data that can affect the way it iterates over field values. Depending on the adapter field object, it may be supplied by the caller, or created by the adapter field object inside this method.

ImplStartIterator returns true if it positions the ValueField field on its first value, false if the adapter field can't supply a first value.

In TFieldValuesAdapter (see page 5803), ImplStartIterator obtains an IWebGetFieldValues (see page 1092) interface for the adapter field that is the value of the Field (see page 5812) property and calls its StartValueIterator method, passing on the value of the OwnerData parameter.

See Also

StartIterator (see page 5567)

ImplEndIterator (see page 5810)

ImplNextIteration (see page 5811)

StartValueIterator (see page 1095)

Field (see page 5812)

54.1.70.4 TFieldValuesAdapter Properties

The properties of the TFieldValuesAdapter class are listed here.

Protected Properties

	Name	Description
	Field (see page 5812)	Indicates the object that represents this adapter's single field.

54.1.70.4.1 TFieldValuesAdapter.Field Property

Indicates the object that represents this adapter's single field.

Pascal

```
property Field: TObject;
```

C++

```
__property TObject * Field;
```

Description

Read (see page 3634) Field to access the object that represents this adapter's ValueField field. Field is an instance of TFieldValuesValueField (see page 5813) or one of its descendants.

The TFieldValuesAdapter (see page 5803) constructor supplies the value for this property.

See Also

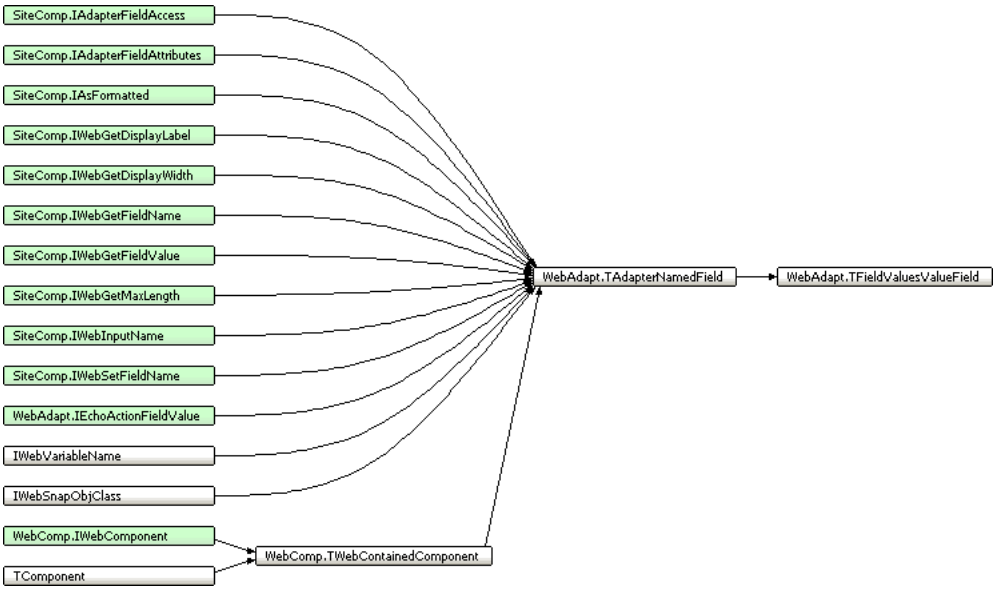
TFieldValuesValueField (see page 5813)

Create (see page 5809)

54.1.71 TFieldValuesValueField Class

TFieldValuesValueField is a adapter field child of TFieldValuesAdapter (see page 5803).

Class Hierarchy



Pascal

```
TFieldValuesValueField = class(TAdapterNamedField);
```

C++

```
class TFieldValuesValueField : public TAdapterNamedField;
```

File

WebAdapt

Description

The parent adapter, TFieldValuesAdapter (see page 5803), is used to iterate over the values associated with an adapter field. TFieldValuesValueField provides the value and display text of the current item.

See Also

TFieldValuesAdapter (see page 5803)

TBaseAdapterMultiValueField (see page 5488)

54.1.71.1 TFieldValuesValueField Members

The following tables list the members exposed by TFieldValuesValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5816)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5816)	Returns text used to display a read-only representation of the value associated with this adapter field.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5817)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.

	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
R	Adapter (see page 5368)	Returns the parent adapter component.
R	Field (see page 5817)	Returns the adapter field that is using TFieldValueAdapter.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.

	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.71.2 TFieldValuesValueField Methods

The methods of the TFieldValuesValueField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 5816)	Returns the name used to access this adapter field using server-side script.
	ImplAsFormatted (see page 5816)	Returns text used to display a read-only representation of the value associated with this adapter field.
	ImplGetValue (see page 5817)	Returns the value associated with this adapter field.

54.1.71.2.1 TFieldValuesValueField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'ValueField'. This allows server-side script to access this adapter field using the parent adapter's ValueField property.

See Also

ImplGetVariableName (see page 5364)

ImplSetFieldName (see page 5366)

GetDefaultFieldName (see page 5357)

54.1.71.2.2 TFieldValuesValueField.ImplAsFormatted Method

Returns text used to display a read-only representation of the value associated with this adapter field.

Pascal

```
function ImplAsFormatted: string; override;
```

C++

```
virtual __fastcall UnicodeString ImplAsFormatted();
```

Description

ImplAsFormatted determines whether the adapter field using TFieldValuesAdapter (see page 5803) has a values list. If a values list is available, the values list is used to look up a value name to return. If there is no values list or no value name, the result of ImplGetValue (see page 5817) is returned. Server-side script accesses the result of this function using the DisplayText property of this adapter field. ImplAsFormatted overrides TAdapterNamedField (see page 5351)'s ImplAsFormatted method.

See Also

Value (see page 5369)

ImplAsFormatted (see page 5361)

GetAdapterValuesList (see page 5426)

ImplGetValue (see page 5817)

54.1.71.2.3 TFieldValuesValueField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue returns the result of GetCurrentValue. GetCurrentValue is implemented by the adapter field using TFieldValuesAdapter (see page 5803). Server-side script accesses the result of this function using the Value (see page 5369) property of this adapter field.

See Also

GetCurrentValue (see page 5494)

Value (see page 5369)

54.1.71.3 TFieldValuesValueField Properties

The properties of the TFieldValuesValueField class are listed here.

Protected Properties

	Name	Description
	Field (see page 5817)	Returns the adapter field that is using TFieldValueAdapter.

54.1.71.3.1 TFieldValuesValueField.Field Property

Returns the adapter field that is using TFieldValueAdapter.

Pascal

```
property Field: TObject;
```

C++

```
__property TObject * Field;
```

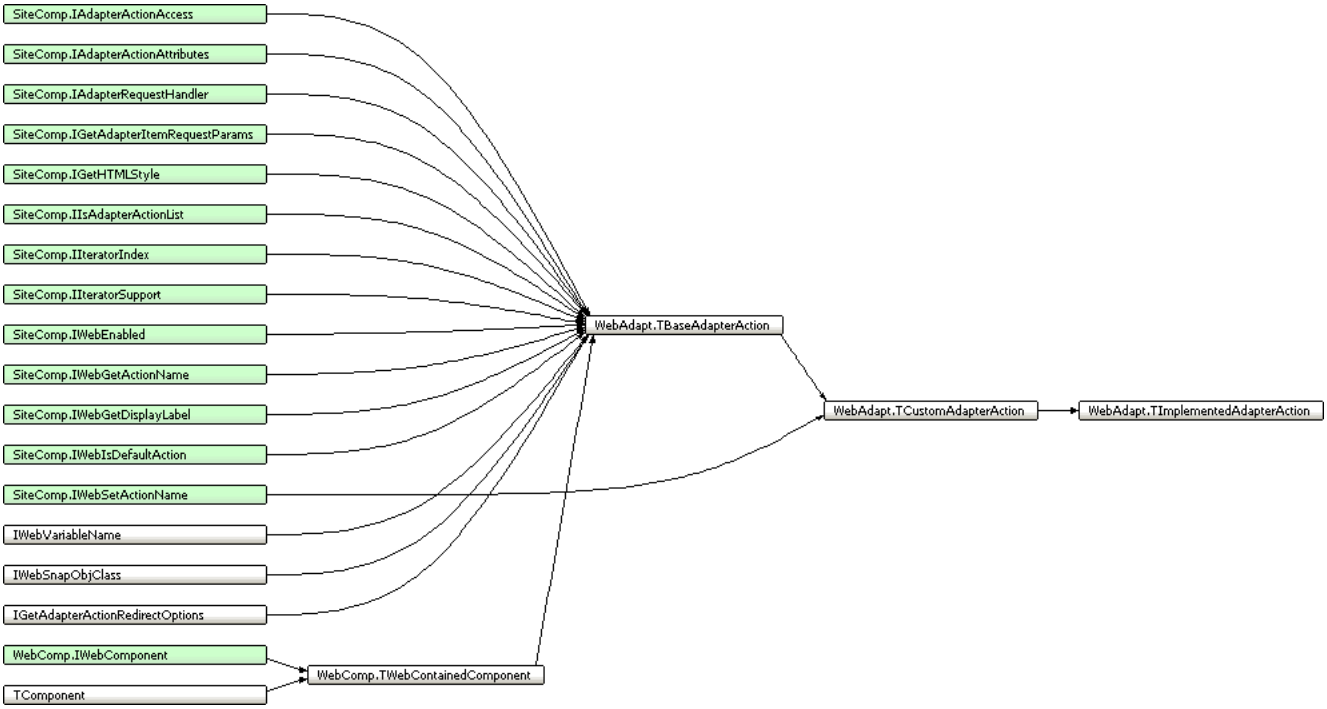
Description

Adapter (see page 5368) fields use the TFieldValuesAdapter (see page 5803) component to iterate over the values associated with the adapter field. The Field property returns the adapter field that is using the TFieldValuesAdapter (see page 5803).

54.1.72 TImplementedAdapterAction Class

TImplementedAdapterAction is a base class for adapter actions.

Class Hierarchy



Pascal

```
TImplementedAdapterAction = class(TCustomAdapterAction);
```

C++

```
class TImplementedAdapterAction : public TCustomAdapterAction;
```

File

WebAdapt

Description

Derive from TImplementedAdapterAction to define an adapter action component that works with specific adapter component types. Classes derived from TImplementedAdapterAction can only be added to adapter components that allow the particular class. On the other hand, classes derived from TCustomAdapterAction (see page 5579) (such as TAdapter (see page 5190)) can be added to any adapter component class.

See Also

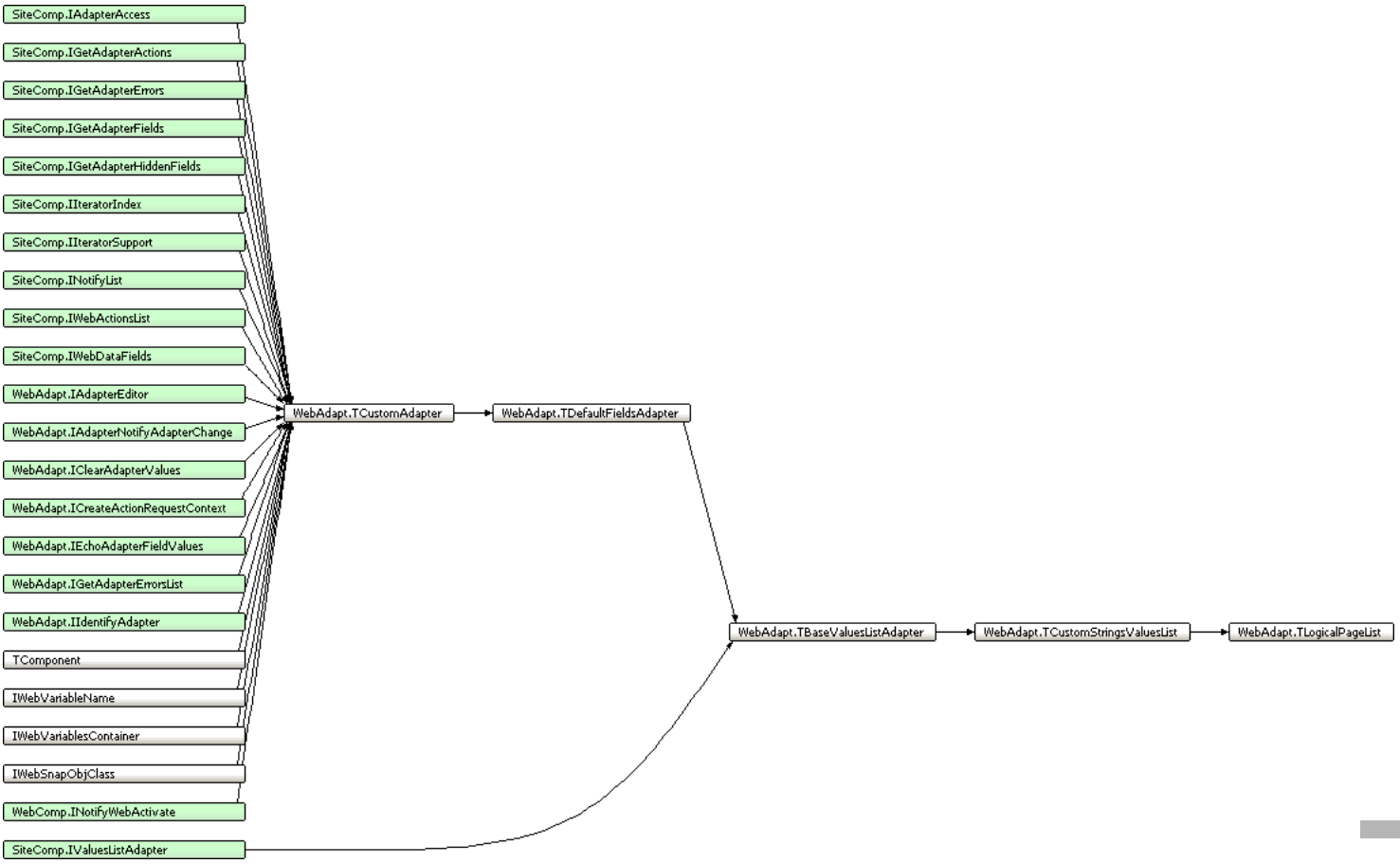
TCustomAdapterAction (see page 5579)

TAdapter (see page 5190)

TAdapterActions (see page 5209)

54.1.73 TLogicalPageList Class

Class Hierarchy



Pascal

```
TLogicalPageList = class(TCustomStringsValuesList);
```

C++

```
class TLogicalPageList : public TCustomStringsValuesList;
```

File

WebAdapt

Description

This is class WebAdapt.TLogicalPageList.

54.1.73.1 TLogicalPageList Members

The following tables list the members exposed by TLogicalPageList.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5522)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5523)	Looks up an image for a value.
	GetListName (see page 5523)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5523)	Looks up a name for a value.
	GetListValue (see page 5524)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.

	GetNameFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
	GetValueFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetValueFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5524)	Returns the class type for this adapter field.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5525)	Indicates whether a type of adapter field can be added to this adapter.
	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 5734)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5525)	Adds default field components to this adapter.
	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetListImage (see page 5525)	Responsible returning an image associated with the current name/value pair.
	ImplGetListImageOfValue (see page 5526)	Looks up an image for a value.
	ImplGetListName (see page 5526)	Responsible returning the name of the name/value pair.
	ImplGetListName (see page 5735)	Responsible returning the name of the name/value pair.
	ImplGetListNameOfValue (see page 5526)	Looks up a name for a value.
	ImplGetListNameOfValue (see page 5735)	Looks up a name for a value.
	ImplGetListValue (see page 5527)	Responsible for returning the value of the name/value pair.
	ImplGetListValue (see page 5735)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 5736)	Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.

	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5736)	Clears the iteration index.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 5736)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	PrepareStrings (see page 5737)	Customizes the content of the strings list.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 5737)	This flag indicates whether the strings list has been prepared or customized.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.

	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.
	OnPrepareStrings (see page 5738)	Implement OnPrepareStrings to customize the content of the strings list.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5824)	Create is the constructor for TCustomStringsValuesList.
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
V	Destroy (see page 5733)	Destroy is the destructor for TCustomStringsList.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetListImage (see page 1074)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1074)	Returns the image object associated with a specified value.
	GetListName (see page 1075)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1075)	Returns the name associated with a specified value.
	GetListValue (see page 1076)	Returns the value of the current name/value pair.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
R	Actions (see page 5572)	Actions is the container for adapter actions components.

	Actions (see page 5746)	Container for adapter actions components.
	Data (see page 5572)	Data is the container for adapter field components.
	Data (see page 5747)	Container for adapter field components.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	Strings (see page 5737)	This string list defines the list of name/value pairs.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

54.1.73.2 TLogicalPageList.Create Constructor

Create is the constructor for TCustomStringsValuesList.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TLogicalPageList(TComponent * AOwner);
```

Description

Create creates the strings list.

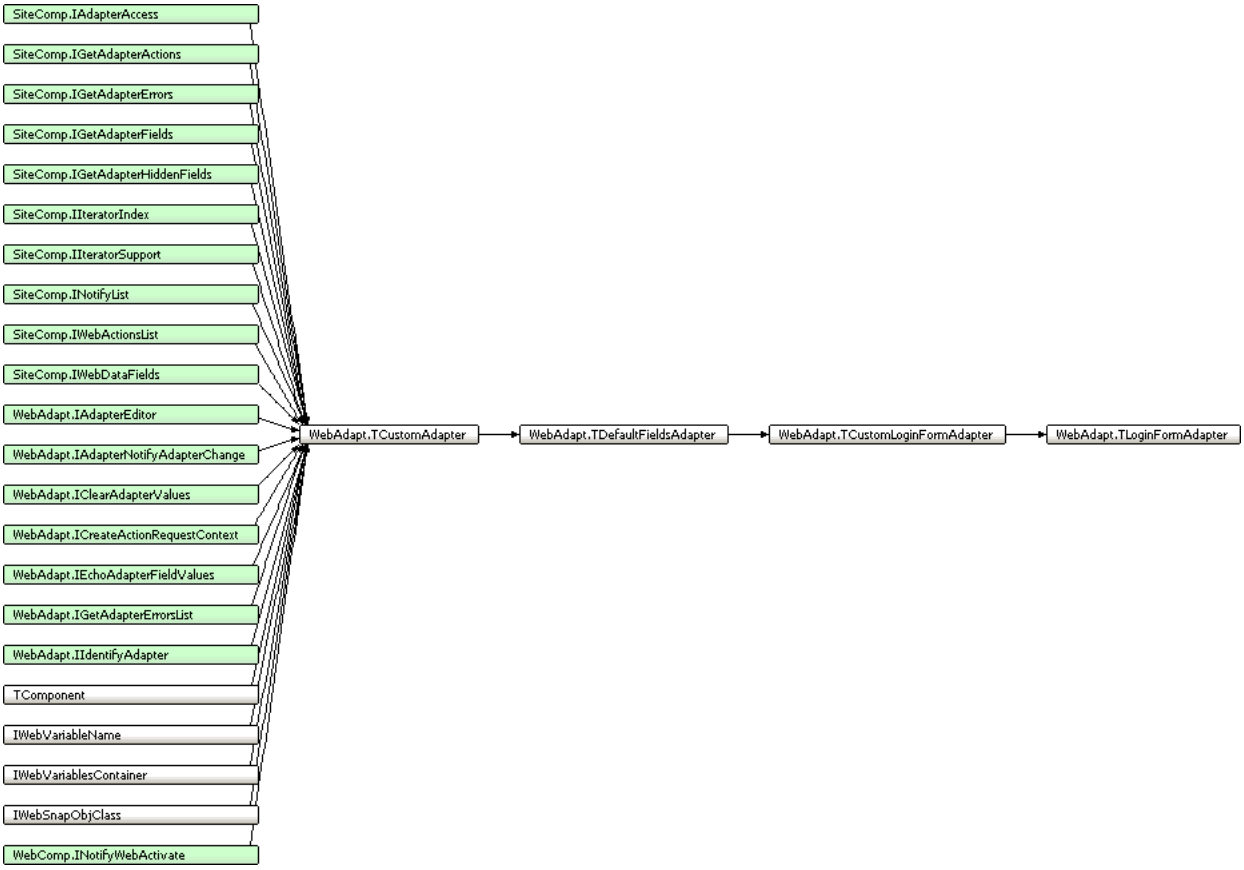
See Also

Strings (see page 5737)

54.1.74 TLoginFormAdapter Class

TLoginFormAdapter contains the adapter fields and adapter actions used to generate a login form.

Class Hierarchy



Pascal

```
TLoginFormAdapter = class(TCustomLoginFormAdapter);
```

C++

```
class TLoginFormAdapter : public TCustomLoginFormAdapter;
```

File

WebAdapt

Description

Use TLoginFormAdapter to generate a login form. TLoginFormAdapter has default fields for user name, password, and next page. The next page field lets the user select a Web page to see after logging in. TLoginFormAdapter has a single default action for logging in. If the WebSnap application also contains an end user adapter such as TEndUserAdapter (see page 5764) or TEndUserSessionAdapter (see page 5792), TLoginFormAdapter calls the end user adapter's LogIn method to log the user in. You can also implement an OnLogin (see page 5834) event handler to provide custom login support.

To build a simple login form

1. Create (see page 5536) a new Web page module that contains a TAdapterPageProducer component.
2. Add a TLoginFormAdapter component to that module.
3. Right click on the TAdapterPageProducer object and display the Web page editor.

4. In the Web page editor, add an adapter form, and to that form add an adapter field group and an adapter command group.
5. Select the field group, and set its Adapter property to the TLoginFormAdapter object.
6. Select the command group, and set its display component to the adapter field group.

You can embellish this simple form as you like. When you are done, connect the form to an end user adapter by setting the end user adapter's LoginPage property to the name of the page module.

To provide complete login support, add TWebUserList, TEndUserSessionAdapter (see page 5792), and TSessionsService components to the web application module to identify user names and passwords and to keep track of end user login state.

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

TAdapterPageProducer (see page 1136)

TWebUserList (see page 6796)

TSessionsService (see page 6756)

TAdapterControlGroup (see page 6341)

TAdapterFieldGroup (see page 6399)

54.1.74.1 TLoginFormAdapter Members

The following tables list the members exposed by TLoginFormAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
	EndUser (see page 5707)	Returns the interface of the WebSnap application's end user adapter.
	ExecuteLogin (see page 5707)	Logs in the user after the user name and password have been validated.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.

	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
V	GetNextPage (see page 5708)	Returns the value of the NextPage (see page 5713) property.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddActionClass (see page 5708)	Provides the underlying implementation for the CanAddActionClass (see page 5539) method.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplCanAddFieldClass (see page 5709)	Provides the underlying implementation for the CanAddFieldClass (see page 5540) method.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetActionsList (see page 5709)	Provides the underlying implementation for the GetActionsList (see page 5546) method.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetFieldsList (see page 5710)	Provides the underlying implementation for the GetFieldsList (see page 5549) method.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetVariable (see page 5556)	Returns an action or field.
V	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
V	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.

	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5710)	Provides the underlying implementation for the NotifyDeactivate (see page 5566) method.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	RaiseBlankPassword (see page 5711)	Raises an EAdapterFieldException (see page 5178) with a message that the supplied password is blank.
	RaiseBlankUserName (see page 5711)	Raises an EAdapterFieldException (see page 5178) with a message that the supplied user name is blank.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	UserList (see page 5711)	Returns the interface of the WebSnap application's Web user list.
	ValidatePassword (see page 5712)	Validates the password that the user supplied to the login form.
	ValidateUserName (see page 5712)	Validates the name that the user supplied to the login form.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	NextPage (see page 5713)	Names the Web page the user should see after successfully logging in.
R	Password (see page 5714)	Indicates the password that the user supplied to the login form.
R	UserName (see page 5714)	Indicates the name that the user supplied to the login form.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5832)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5832)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5833)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5833)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5833)	Supplies name/value pairs needed to execute an adapter action.
	OnLogin (see page 5834)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5834)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5835)	Occurs when the login form adapter must validate the name that the user supplied.

Published Properties

	Name	Description
	Actions (see page 5830)	Container for adapter actions components.
	Data (see page 5831)	Container for adapter field components.
	PasswordRequired (see page 5831)	Indicates whether the user must supply a password to log in.

54.1.74.2 TLoginFormAdapter Properties

The properties of the TLoginFormAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5830)	Container for adapter actions components.
	Data (see page 5831)	Container for adapter field components.
	PasswordRequired (see page 5831)	Indicates whether the user must supply a password to log in.

54.1.74.2.1 TLoginFormAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5231)

54.1.74.2.2 TLoginFormAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5237)

54.1.74.2.3 TLoginFormAdapter.PasswordRequired Property

Indicates whether the user must supply a password to log in.

Pascal

```
property PasswordRequired: Boolean;
```

C++

```
__property Boolean PasswordRequired;
```

Description

Set PasswordRequired to indicate whether users must supply a password to log in. If PasswordRequired is true and the OnValidatePassword event handler does not handle the validation of the password, then the login form adapter checks that the value of Password is not an empty string. If Password is an empty string, the login form adapter raises an EAdapterFieldException exception when it tries to validate the password. If the OnValidatePassword event handler takes care of validating the password, PasswordRequired is ignored.

Note: The login form adapter validates the password (and hence checks PasswordRequired) when the WebSnap application executes this adapter's LogIn action.

See Also

Password (see page 5714)

OnValidatePassword (see page 5716)

TLoginFormAdapterLoginAction (see page 5835)

EAdapterFieldException (see page 5178)

54.1.74.3 TLoginFormAdapter Events

The events of the TLoginFormAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5832)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5832)	Occurs after the adapter action response is generated.
	OnBeforeExecuteAction (see page 5833)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5833)	Occurs before the adapter action response is generated.
	OnGetActionParams (see page 5833)	Supplies name/value pairs needed to execute an adapter action.
	OnLogin (see page 5834)	Occurs after the password and user ID are validated.
	OnValidatePassword (see page 5834)	Occurs when the login form adapter must validate the password that the user supplied.
	OnValidateUserName (see page 5835)	Occurs when the login form adapter must validate the name that the user supplied.

54.1.74.3.1 TLoginFormAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.74.3.2 TLoginFormAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.74.3.3 TLoginFormAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5478)

OnAfterExecuteAction (see page 5576)

54.1.74.3.4 TLoginFormAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5478)

OnAfterGetActionResponse (see page 5576)

54.1.74.3.5 TLoginFormAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5478)

54.1.74.3.6 TLoginFormAdapter.OnLogin Event

Occurs after the password and user ID are validated.

Pascal

```
property OnLogin: TLoginUserIDEvent;
```

C++

```
__property TLoginUserIDEvent OnLogin;
```

Description

Write an OnLogin event handler to respond when a user has successfully logged in. This event allows your application to record the user login in an application-specific manner.

When the login form adapter's LogIn action executes, the adapter first receives an OnValidatePassword and an OnValidateUserName event. After validating the password and user name, the login form adapter receives an OnLogin event so that it can log the user in. After the OnLogin event, if the WebSnap application includes an end user adapter, the login form adapter calls the end user adapter's LogIn method. Thus, it may not be necessary to write an OnLogin event handler if the application includes an end user adapter.

See Also

OnValidatePassword (see page 5716)

OnValidateUserName (see page 5716)

LogIn (see page 5993)

TWebUserList (see page 6796)

UserName (see page 5714)

54.1.74.3.7 TLoginFormAdapter.OnValidatePassword Event

Occurs when the login form adapter must validate the password that the user supplied.

Pascal

```
property OnValidatePassword: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidatePassword;
```

Description

Write an OnValidatePassword event handler to validate the password that the user supplies to the login form. This event is especially important if the WebSnap application does not include a TWebUserList component for checking passwords against user names. The supplied Password is the value of the Value parameter on the event handler. You can check this against the user's name by reading the UserName property.

When the login form adapter's LogIn action executes, the adapter first receives an OnValidatePassword event to check the password. If the event handler sets the Handled parameter to true, indicating that the password is valid, the adapter goes on to validate the user name using an OnValidateUserName event, and then to log the user in using an OnLogin event (and, if the application includes an end user adapter, by calling the end user adapter's LogIn method). If the event handler finds the password to be invalid, it should raise an exception. If the event handler does not validate the supplied password (that is, if it sets

the Handled parameter to false), the login form adapter validates the password by checking whether it is non-blank if the PasswordRequired property is true.

See Also

OnLogin (see page 5715)

OnValidateUserName (see page 5716)

Password (see page 5714)

UserName (see page 5714)

PasswordRequired (see page 5714)

54.1.74.3.8 TLoginFormAdapter.OnValidateUserName Event

Occurs when the login form adapter must validate the name that the user supplied.

Pascal

```
property OnValidateUserName: TValidateAdapterFieldEvent;
```

C++

```
__property TValidateAdapterFieldEvent OnValidateUserName;
```

Description

Write an OnValidateUserName event handler to validate the name that the user supplies when logging in. This event occurs after the OnValidatePassword event and before the OnLogin event.

If the event handler sets the Handled parameter to true, the login form adapter assumes that the user name (which is the value of the Value parameter and of the UserName property) is valid. In this case, the adapter checks whether the WebSnap application includes a TWebUserList component, and if so, checks the user name against the password. If the event handler sets the Handled parameter to false, the adapter validates the user name by checking whether it is nonempty. In this case, the adapter raises an EAdapterFieldException exception if the value of UserName is an empty string. If the OnValidateUserName event handler finds the user name to be invalid, it should raise an exception.

See Also

OnLogin (see page 5715)

OnValidatePassword (see page 5716)

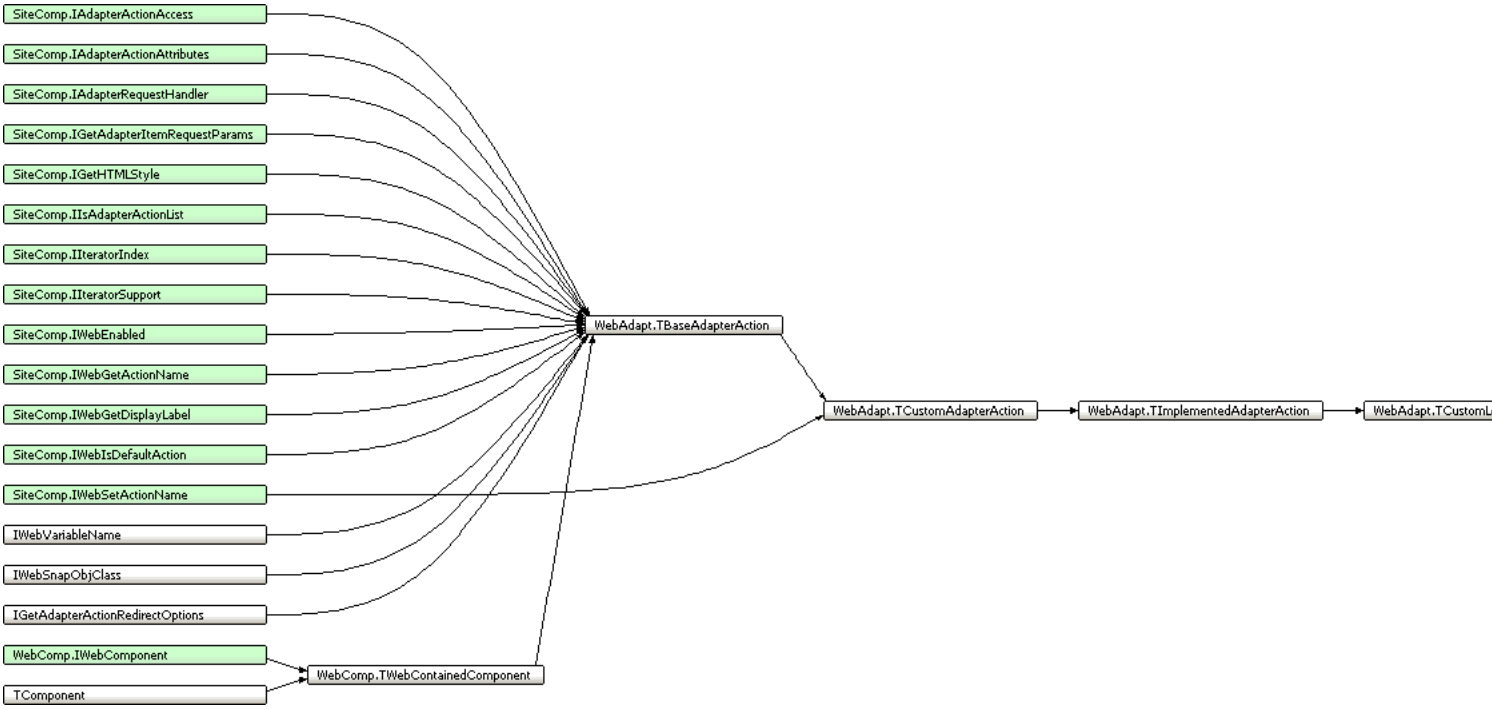
UserName (see page 5714)

EAdapterFieldException (see page 5178)

54.1.75 TLoginFormAdapterLoginAction Class

TLoginFormAdapterLoginAction executes the login command of a login form generated with TLoginFormAdapter (see page 5824).

Class Hierarchy



Pascal

```
TLoginFormAdapterLoginAction = class(TCustomLoginFormAdapterAction);
```

C++

```
class TLoginFormAdapterLoginAction : public TCustomLoginFormAdapterAction;
```

File

WebAdapt

Description

Add TLoginFormAdapterLoginAction to TLoginFormAdapter (see page 5824). The login form typically contains three fields: user name, password, and next page.

See Also

TCustomLoginFormAdapter (see page 5701)

TAdapterNextPageField (see page 5379)

TAdapterUserNameField (see page 5433)

TAdapterPasswordField (see page 5391)

54.1.75.1 TLoginFormAdapterLoginAction Members

The following tables list the members exposed by TLoginFormAdapterLoginAction.

Protected Methods

	Name	Description
	CheckExecuteAccess (see page 5452)	Checks end-user rights to execute this adapter action.
	CreateRequestContext (see page 5452)	Creates an object that describes the adapter action parameters and adapter field values.
V	DoAfterExecuteActionRequest (see page 5453)	Calls event handlers after the adapter action is executed.
V	DoAfterGetExecuteActionResponse (see page 5453)	Calls event handlers after the adapter action response is formulated.
V	DoBeforeExecuteActionRequest (see page 5453)	Calls event handlers before the adapter action is executed.
V	DoBeforeGetExecuteActionResponse (see page 5454)	Calls event handlers before the adapter action response is formulated.
	EndIterator (see page 5454)	Implements IIteratorSupport (see page 1054)'s EndIterator method.
	ExecuteActionRequest (see page 5455)	Executes the adapter action using parameters passed in the HTTP request.
	GetActionName (see page 5455)	Gets the name used to reference this adapter action with server-side script.
	GetAdapter (see page 5456)	GetAdapter returns the parent adapter component.
	GetAdapterItemRequestParams (see page 5456)	Provides the name/value pairs needed to execute this adapter action.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultActionName (see page 5840)	Gets the name used to reference this adapter action with server-side script.
V	GetDefaultDisplayLabel (see page 5457)	Provides the default display label for this adapter action.
	GetDisplayLabel (see page 5457)	Returns the suggested display label for this adapter action.
	GetDisplayStyle (see page 5457)	Returns a string value to suggest an HTML display style for this adapter action.
V	GetDisplayStyleType (see page 5458)	Suggests an HTML display type for this adapter action.
	GetErrors (see page 5458)	Returns the parent adapter's errors list.
	GetExecuteActionResponse (see page 5458)	Generates a response after the adapter action has been executed.
	GetFailureOptions (see page 5459)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
	GetInputStyle (see page 5459)	Returns a string value to suggest an HTML display style for this adapter action.
	GetIsAdapterActionList (see page 5460)	Indicates whether this adapter action may have multiple commands.
V	GetItemCount (see page 5460)	Returns the number of commands defined by this adapter action.
	GetIteratorIndex (see page 5460)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method.
	GetSuccessOptions (see page 5461)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	GetVariableName (see page 5461)	Gets the name used to reference this adapter action with server-side script.
	GetViewMode (see page 5462)	Returns a string value to suggest a view mode for this adapter action.
V	GetViewModeType (see page 5462)	Suggests a view mode for this adapter action.
	GetVisible (see page 5462)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	GetWebSnapObjClass (see page 5463)	Returns the class type for this adapter field.
	HandleRequest (see page 5463)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 5463)	Checks the end-user's right to execute this adapter action.
	ImplCreateRequestContext (see page 5464)	Creates an object that describes the action parameters and field values.
	ImplEndIterator (see page 5464)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5454) method.
V	ImplExecuteActionRequest (see page 5840)	Executes the adapter action using parameters passed in the HTTP request.
V	ImplGetActionName (see page 5465)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetActionName (see page 5584)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetAdapterItemRequestParams (see page 5841)	Provides the name/value pairs needed to execute this adapter action.
V	ImplGetDisplayLabel (see page 5465)	Returns the suggested display label for this adapter action.
V	ImplGetExecuteActionResponse (see page 5466)	Generates a response after the adapter action has been executed.
V	ImplGetFailureOptions (see page 5466)	Indicates whether to perform an HTTP redirect after execution of this adapter action fails.
V	ImplGetIsAdapterActionList (see page 5466)	Indicates whether this adapter action may have multiple commands.
V	ImplGetIteratorIndex (see page 5467)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5460) method.
V	ImplGetSuccessOptions (see page 5467)	Indicates whether to perform an HTTP redirect after successfully executing this action.
V	ImplGetSuccessOptions (see page 5585)	Indicates whether to perform an HTTP redirect after successfully executing this action.

	ImplGetVariableName (see page 5467)	Gets the name used to reference this adapter action with server-side script.
V	ImplGetVisible (see page 5468)	Indicates whether a button or hyperlink for this adapter action should be displayed on the HTML page.
V	ImplHandleRequest (see page 5468)	Executes the adapter action and generates a response.
V	ImplHasExecuteAccess (see page 5468)	Checks the end-user's right to execute this adapter action.
V	ImplInIterator (see page 5469)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5471) method.
V	ImplIsDefaultAction (see page 5469)	Indicates whether this adapter action should be added, by default, to the parent adapter.
	ImplNextIteration (see page 5470)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5472) method.
	ImplSetActionName (see page 5585)	Sets the name that is used to reference this adapter action with server-side script.
	ImplStartIterator (see page 5470)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5473) method.
V	ImplWebEnabled (see page 5470)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
V	ImplWebEnabled (see page 5585)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.
	InIterator (see page 5471)	Implements IIteratorIndex (see page 1052)'s InIterator method.
	IsDefaultAction (see page 5471)	Indicates whether this adapter action should be added, by default, to the parent adapter.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	NextIteration (see page 5472)	Implements IIteratorSupport (see page 1054)'s NextIteration method.
V	NextPage (see page 5841)	Identifies the name of the next page to display after login.
V	ParentChanged (see page 5472)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetActionName (see page 5586)	Sets the name that is used to reference this adapter action with server-side script.
	SetDisplayLabel (see page 5472)	Set the value of the DisplayLabel (see page 5474) property.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.
	StartIterator (see page 5473)	Implements IIteratorSupport (see page 1054)'s StartIterator method.
	WebEnabled (see page 5473)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Protected Properties

	Name	Description
R	Adapter (see page 5474)	Provides access to the parent adapter component.
R	Adapter (see page 5721)	Returns the adapter that acts as a parent to this action.
R	Errors (see page 5475)	Returns the parent adapter's Errors list.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Events

	Name	Description
	OnExecute (see page 5588)	Called to execute this action using parameters passed in the HTTP request.
	OnGetEnabled (see page 5589)	Indicates whether a button or hyperlink for this adapter action should be enabled on the HTML page.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	CreateRequestContext (see page 1013)	Creates an object that describes the adapter action parameters and adapter field values.

	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterItemRequestParams (see page 1024)	This is GetAdapterItemRequestParams, a member of class IGetAdapterItemRequestParams.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayStyle (see page 1032)	Returns the preferred display style for an adapter field or adapter action.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputStyle (see page 1033)	Returns the preferred editing style for an adapter field.
	GetIsAdapterActionList (see page 1050)	This is GetIsAdapterActionList, a member of class IIsAdapterActionList.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetViewMode (see page 1033)	Returns the preferred mode of an adapter field.
	GetVisible (see page 984)	This is GetVisible, a member of class IAdapterActionAttributes.
	HandleRequest (see page 1013)	Executes the adapter action and generates a response.
	HasExecuteAccess (see page 983)	This is HasExecuteAccess, a member of class IAdapterActionAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsDefaultAction (see page 1101)	This is IsDefaultAction, a member of class IWebIsDefaultAction.
	NextIteration (see page 1056)	Advances to the objects next item.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
	WebEnabled (see page 1084)	This is WebEnabled, a member of class IWebEnabled.

Public Properties

	Name	Description
	ActionName (see page 1087)	This is ActionName, a member of class IWebGetActionName.
	ActionName (see page 5474)	Identifies the name of this action when using server-side script.
	ActionName (see page 5586)	Identifies the name of this action when using server-side script.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	ExecuteAccess (see page 5475)	Defines the rights required to execute this action.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
	IsAdapterActionList (see page 1050)	This is IsAdapterActionList, a member of class IIsAdapterActionList.
	ItemCount (see page 5475)	Returns the number of items defined by an adapter action.
	ItemIndex (see page 5476)	Returns the index of the current item of an adapter action.
	IteratorIndex (see page 1054)	Indicates the index of the current item.
	RedirectOptions (see page 5476)	Indicates whether to perform an HTTP redirect after successfully executing this action.
	Visible (see page 984)	This is Visible, a member of class IAdapterActionAttributes.

Published Events

	Name	Description
	OnAfterExecute (see page 5842)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5843)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5843)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5844)	Called before the adapter action response is generated.
	OnGetParams (see page 5844)	Specifies name/value pairs needed to execute this adapter action.

Published Properties

	Name	Description
	DisplayLabel (see page 5842)	Suggests a display label for this adapter action.

54.1.75.2 TLoginFormAdapterLoginAction Methods

The methods of the TLoginFormAdapterLoginAction class are listed here.

Protected Methods

	Name	Description
	GetDefaultActionName (see page 5840)	Gets the name used to reference this adapter action with server-side script.
	ImplExecuteActionRequest (see page 5840)	Executes the adapter action using parameters passed in the HTTP request.
	ImplGetAdapterItemRequestParams (see page 5841)	Provides the name/value pairs needed to execute this adapter action.
	NextPage (see page 5841)	Identifies the name of the next page to display after login.

54.1.75.2.1 TLoginFormAdapterLoginAction.GetDefaultActionName Method

Gets the name used to reference this adapter action with server-side script.

Pascal

```
function GetDefaultActionName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultActionName();
```

Description

GetDefaultActionName returns "Login".

See Also

GetActionName (see page 5455)

GetDefaultActionName (see page 5456)

54.1.75.2.2 TLoginFormAdapterLoginAction.ImplExecuteActionRequest Method

Executes the adapter action using parameters passed in the HTTP request.

Pascal

```
procedure ImplExecuteActionRequest(AActionRequest: IActionRequest; AActionResponse:
IActionResponse); override;
```

C++

```
virtual __fastcall ImplExecuteActionRequest(IActionRequest AActionRequest, IActionResponse
AActionResponse);
```

Description

ImplExecuteActionRequest uses the user name and password passed in the HTTP request to log the user in. Exceptions are raised if the user name or password contain invalid values or if the user name/password is not known. If an exception is raised the exception message is added to the parent adapter's error list and the echo flag is set to true. If the login is successful then ImplExecuteActionRequest causes an HTTP redirect to the page specified in the next page field.

See Also

ImplExecuteActionRequest (see page 5464)

IActionRequest

IActionResponse

ValidatePassword (see page 5712)

ValidateUserName (see page 5712)

ValidateUser (see page 1117)

Errors (see page 5573)

EchoActionFieldValues (see page 5572)

54.1.75.2.3 TLoginFormAdapterLoginAction.ImplGetAdapterItemRequestParams Method

Provides the name/value pairs needed to execute this adapter action.

Pascal

```
procedure ImplGetAdapterItemRequestParams(var AIdentifier: string; Params:
IAdapterItemRequestParams); override;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(UnicodeString AIdentifier,
IAdapterItemRequestParams Params);
```

Description

ImplGetAdapterItemRequestParams adds a name/value pair identifying the name of the next page that should be displayed after login. When the login action is executed, the name/value specified by ImplGetAdapterItemRequestParams will be used to generate the HTTP response unless the login form contains a next page field. In that case, the value entered by the end-user will be used to the generate the HTTP response.

See Also

NextPage (see page 5713)

RequestedPageName (see page 5974)

ImplGetAdapterItemRequestParams (see page 5465)

IAdapterItemRequestParams (see page 1005)

TAdapterNextPageField (see page 5379)

54.1.75.2.4 TLoginFormAdapterLoginAction.NextPage Method

Identifies the name of the next page to display after login.

Pascal

```
function NextPage: string; virtual;
```

C++

```
virtual __fastcall UnicodeString NextPage();
```

Description

This value is used when generating the login form. NextPage returns the result of TCustomLoginFormAdapter.NextPage.

See Also

NextPage (see page 5713)

54.1.75.3 TLoginFormAdapterLoginAction Properties

The properties of the TLoginFormAdapterLoginAction class are listed here.

Published Properties

	Name	Description
	DisplayLabel (see page 5842)	Suggests a display label for this adapter action.

54.1.75.3.1 TLoginFormAdapterLoginAction.DisplayLabel Property

Suggests a display label for this adapter action.

Pascal

```
property DisplayLabel: string;
```

C++

```
__property UnicodeString DisplayLabel;
```

Description

Use this property to suggest a display label to use when generating HTML to display the a button or hyperlink to execute this adapter action. Typically the adapter action is displayed using server-side script, either generated or custom written. Server-side script uses the DisplayLabel property to access this value.

See Also

ImplGetDisplayLabel (see page 5465)

SetDisplayLabel (see page 5472)

54.1.75.4 TLoginFormAdapterLoginAction Events

The events of the TLoginFormAdapterLoginAction class are listed here.

Published Events

	Name	Description
	OnAfterExecute (see page 5842)	Called after the adapter action has executed.
	OnAfterGetResponse (see page 5843)	Called after the adapter action response is generated.
	OnBeforeExecute (see page 5843)	Called before the adapter action is executed.
	OnBeforeGetResponse (see page 5844)	Called before the adapter action response is generated.
	OnGetParams (see page 5844)	Specifies name/value pairs needed to execute this adapter action.

54.1.75.4.1 TLoginFormAdapterLoginAction.OnAfterExecute Event

Called after the adapter action has executed.

Pascal

```
property OnAfterExecute: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterExecute;
```

Description

Implement OnAfterExecute to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterExecute is called by DoAfterExecuteActionRequest.

See Also

DoAfterExecuteActionRequest (see page 5453)

OnAfterExecuteAction (see page 5576)

OnBeforeExecute (see page 5478)

54.1.75.4.2 TLoginFormAdapterLoginAction.OnAfterGetResponse Event

Called after the adapter action response is generated.

Pascal

```
property OnAfterGetResponse: TActionExecuteEvent;
```

C++

```
__property TActionExecuteEvent OnAfterGetResponse;
```

Description

Implement OnAfterGetResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. OnAfterGetResponse is called by DoAfterGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnBeforeGetActionResponse (see page 5578)

54.1.75.4.3 TLoginFormAdapterLoginAction.OnBeforeExecute Event

Called before the adapter action is executed.

Pascal

```
property OnBeforeExecute: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeExecute;
```

Description

Implement OnBeforeExecute to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed. OnBeforeExecute is called by DoBeforeExecuteActionRequest.

See Also

DoBeforeExecuteActionRequest (see page 5453)

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.75.4.4 TLoginFormAdapterLoginAction.OnBeforeGetResponse Event

Called before the adapter action response is generated.

Pascal

```
property OnBeforeGetResponse: TActionExecuteHandledEvent;
```

C++

```
__property TActionExecuteHandledEvent OnBeforeGetResponse;
```

Description

Implement OnBeforeGetResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed. OnBeforeGetResponse is called by DoBeforeGetExecuteActionReponse.

See Also

DoBeforeGetExecuteActionResponse (see page 5454)

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.75.4.5 TLoginFormAdapterLoginAction.OnGetParams Event

Specifies name/value pairs needed to execute this adapter action.

Pascal

```
property OnGetParams: TActionGetParamsEvent;
```

C++

```
__property TActionGetParamsEvent OnGetParams;
```

Description

Implement this event handler to specify the name/value needed to execute this action. For example, an action that deletes a row in a table will need a name/value pair that identifies the row to be deleted such as "Key=123".

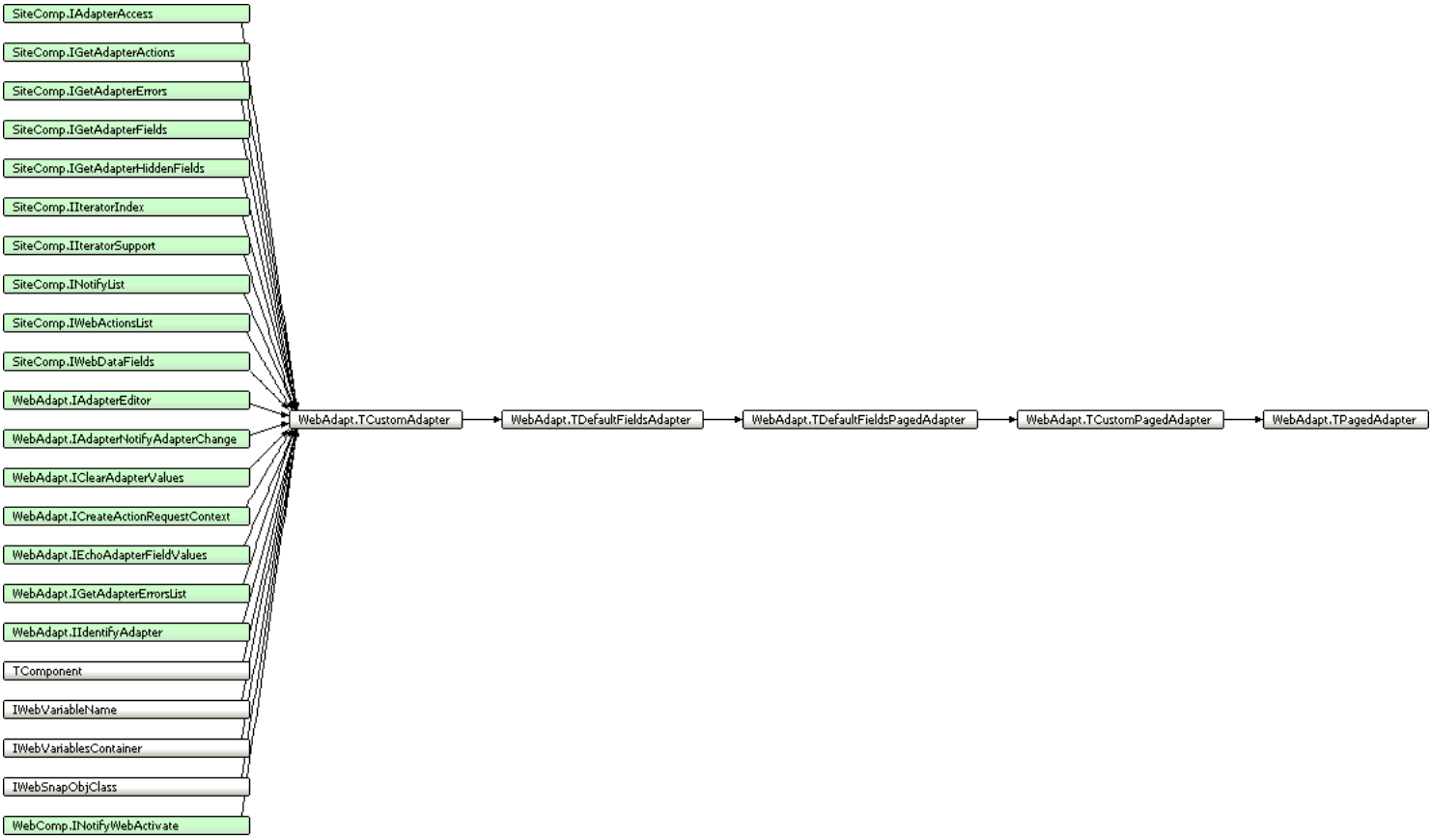
See Also

ImplGetAdapterItemRequestParams (see page 5465)

54.1.76 TPagedAdapter Class

TPagedAdapter divides a large amount of data into several pages, each page displaying a portion of the data records.

Class Hierarchy



Pascal

```
TPagedAdapter = class(TCustomPagedAdapter);
```

C++

```
class TPagedAdapter : public TCustomPagedAdapter;
```

File

WebAdapt

Description

Use TPagedAdapter like TAdapter (see page 5190) to define a custom interface to application logic. Set the PageSize (see page 5851) property to indicate the number of rows that a page displays. When the PageSize (see page 5851) is not 0, the TPagedAdapter component automatically adds built-in actions called TAdapterNextPageAction (see page 5370), TAdapterPrevPageAction (see page 5396), and TAdapterGotoPageAction (see page 5291) for moving between pages. To generate HTML buttons for these actions, use TAdapterPageProducer.

The TDataSetAdapter component also supports paging.

54.1.76.1 TPagedAdapter Members

The following tables list the members exposed by TPagedAdapter.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultActions (see page 5753)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
V	EncodeActionParams (see page 5754)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
	EndOfPage (see page 5754)	Indicates whether the record index has gone past the last record in the current page.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
V	ExtractRequestParams (see page 5754)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.
	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
V	GetEOF (see page 5755)	Indicates whether the record position has passed the last record.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetFirstRecord (see page 5755)	Set the record position to the first record.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.

	GetNextRecord (see page 5755)	Increments the record position.
	GetPageCount (see page 5756)	Calculates the number of pages needed to display all adapter records.
	GetRecordCount (see page 5756)	Returns the total number of records.
	GetRecordIndex (see page 5756)	Returns the index of the current record.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddActionClass (see page 5757)	Indicates whether a type of adapter action can be added to this adapter.
	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplEndIterator (see page 5757)	Implements IteratorSupport's EndIterator (see page 5542) method to support iterating over the data records of the adapter.
	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
	ImplGetActionsList (see page 5757)	Provides a list of action types that can be added to this adapter.
	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 5758)	Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5758)	Clears the current page variable.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.

	ImplStartIterator (see page 5759)	Implements IteratorSupport.StartIterator (see page 5567) to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	SetPageSize (see page 5759)	Updates the pagesize variable.
	SkipToPage (see page 5760)	Skips records until the current record is within the current page.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.
	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.

	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	CurrentPage (see page 5760)	Sets or gets the current page index.
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
R	PageCount (see page 5761)	Calculates the number of pages needed to display all adapter records.
R	RecordCount (see page 5761)	Returns the total number of records.
R	RecordIndex (see page 5762)	Returns the index of the current record.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5852)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5853)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5853)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5853)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5854)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5854)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5854)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5855)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5855)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 5856)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5856)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5856)	Implement this method to increment the iteration index.

	OnGetRecordCount (see page 5856)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5857)	Implement this event to identify the zero-based current record.

Published Properties

	Name	Description
	Actions (see page 5850)	Container for adapter actions components.
	Data (see page 5850)	Container for adapter field components.
	ModifyAccess (see page 5851)	Identifies the rights required to update this adapter's fields.
	PageSize (see page 5851)	Sets or gets the page size variable.
	ViewAccess (see page 5851)	Identifies the rights required to view this adapter's fields.

54.1.76.2 TPagedAdapter Properties

The properties of the TPagedAdapter class are listed here.

Published Properties

	Name	Description
	Actions (see page 5850)	Container for adapter actions components.
	Data (see page 5850)	Container for adapter field components.
	ModifyAccess (see page 5851)	Identifies the rights required to update this adapter's fields.
	PageSize (see page 5851)	Sets or gets the page size variable.
	ViewAccess (see page 5851)	Identifies the rights required to view this adapter's fields.

54.1.76.2.1 TPagedAdapter.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5231)

54.1.76.2.2 TPagedAdapter.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5237)

54.1.76.2.3 TPagedAdapter.ModifyAccess Property

Identifies the rights required to update this adapter's fields.

Pascal

```
property ModifyAccess: string;
```

C++

```
__property UnicodeString ModifyAccess;
```

Description

The ModifyAccess property defines the access rights required to modify this fields of this adapter. When server-side script accesses the CanModify property, this ModifyAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6796)

HasModifyAccess (see page 5552)

ImplHasModifyAccess (see page 5558)

54.1.76.2.4 TPagedAdapter.PageSize Property

Sets or gets the page size variable.

Pascal

```
property PageSize: Integer;
```

C++

```
__property int PageSize;
```

Description

If the page size variable is set to a value greater than zero and no actions have been explicitly added to the adapter then the NextPage, PrevPage, and GotoPage are automatically added. If the page size variable is changed back to zero then the these actions are removed.

See Also

TAdapterNextPageAction (see page 5370)

TAdapterPrevPageAction (see page 5396)

TAdapterGotoPageAction (see page 5291)

54.1.76.2.5 TPagedAdapter.ViewAccess Property

Identifies the rights required to view this adapter's fields.

Pascal

```
property ViewAccess: string;
```

C++

```
__property UnicodeString ViewAccess;
```

Description

The ViewAccess property identifies the access rights required to view fields of this adapter. When server-side script accesses the CanView property, the ViewAccess property value is checked against the end user's rights.

See Also

TWebUserList (see page 6796)

HasViewAccess (see page 5553)

ImplHasViewAccess (see page 5559)

54.1.76.3 TPagedAdapter Events

The events of the TPagedAdapter class are listed here.

Published Events

	Name	Description
	OnAfterExecuteAction (see page 5852)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5853)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5853)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5853)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5854)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5854)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5854)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5855)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5855)	Supplies name/value pairs needed to execute an adapter action.
	OnGetEOF (see page 5856)	Implement this event to indicate when the current record index has exceeded the total record count.
	OnGetFirstRecord (see page 5856)	Implement this method to initialize the iteration index to zero.
	OnGetNextRecord (see page 5856)	Implement this method to increment the iteration index.
	OnGetRecordCount (see page 5856)	Implement this event to identify the total number of records that can be iterated.
	OnGetRecordIndex (see page 5857)	Implement this event to identify the zero-based current record.

54.1.76.3.1 TPagedAdapter.OnAfterExecuteAction Event

Occurs after an adapter action has executed.

Pascal

```
property OnAfterExecuteAction: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterExecuteAction;
```

Description

Implement OnAfterExecuteAction to do something after the action has executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterExecute (see page 5477)

OnBeforeExecuteAction (see page 5577)

54.1.76.3.2 TPagedAdapter.OnAfterGetActionResponse Event

Occurs after the adapter action response is generated.

Pascal

```
property OnAfterGetActionResponse: TAdapterActionExecuteEvent;
```

C++

```
__property TAdapterActionExecuteEvent OnAfterGetActionResponse;
```

Description

Implement OnAfterGetActionResponse to do something after the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request.

See Also

OnAfterGetResponse (see page 5477)

OnBeforeGetActionResponse (see page 5578)

54.1.76.3.3 TPagedAdapter.OnAfterUpdateFields Event

Occurs after values in the HTTP request are used to update adapter fields.

Pascal

```
property OnAfterUpdateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterUpdateFields;
```

Description

Implement this event handler to perform custom updates. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record update errors.

See Also

UpdateFieldsUpdateValues (see page 5568)

OnBeforeUpdateFields (see page 5578)

Errors (see page 5573)

54.1.76.3.4 TPagedAdapter.OnAfterValidateFields Event

Occurs after values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnAfterValidateFields: TActionRequestFieldsEvent;
```

C++

```
__property TActionRequestFieldsEvent OnAfterValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. The adapter's errors object is used to record validation errors.

See Also

UpdateFieldsValidateValues (see page 5569)

OnBeforeValidateFields (see page 5578)

Errors (see page 5573)

54.1.76.3.5 TPagedAdapter.OnBeforeExecuteAction Event

Occurs before an adapter action is executed.

Pascal

```
property OnBeforeExecuteAction: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeExecuteAction;
```

Description

Implement OnBeforeExecuteAction to do something before the action is executed. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to prevent the action from being executed.

See Also

OnBeforeExecute (see page 5478)

OnAfterExecuteAction (see page 5576)

54.1.76.3.6 TPagedAdapter.OnBeforeGetActionResponse Event

Occurs before the adapter action response is generated.

Pascal

```
property OnBeforeGetActionResponse: TAdapterActionExecuteHandledEvent;
```

C++

```
__property TAdapterActionExecuteHandledEvent OnBeforeGetActionResponse;
```

Description

Implement OnBeforeGetActionResponse to do something before the action response is generated. The Params parameter contains the adapter action parameters (name/value pairs) that were passed in the HTTP request. Set the Handled flag to indicate that the response has been generated and no further processing is needed.

See Also

OnBeforeGetResponse (see page 5478)

OnAfterGetActionResponse (see page 5576)

54.1.76.3.7 TPagedAdapter.OnBeforeUpdateFields Event

Occurs before values in the HTTP request are used to update adapter fields.

Pascal

```
property OnBeforeUpdateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeUpdateFields;
```

Description

Implement this event handler to perform custom updates of adapter fields. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further updates. Add update errors to the adapter's errors object.

See Also

UpdateFieldsUpdateValues (see page 5568)

OnAfterUpdateFields (see page 5576)

Errors (see page 5573)

54.1.76.3.8 TPagedAdapter.OnBeforeValidateFields Event

Occurs before values in the HTTP request are validated by the adapter's fields.

Pascal

```
property OnBeforeValidateFields: TActionRequestFieldsHandledEvent;
```

C++

```
__property TActionRequestFieldsHandledEvent OnBeforeValidateFields;
```

Description

Implement this event handler to perform custom validation of the HTTP request. The AdapterFields parameter lists all of the adapter fields that have corresponding values in the HTTP request. Set the Handled flag to prevent any further validation. Add validation errors to the adapter's errors object.

See Also

UpdateFieldsValidateValues (see page 5569)

OnAfterValidateFields (see page 5577)

Errors (see page 5573)

54.1.76.3.9 TPagedAdapter.OnGetActionParams Event

Supplies name/value pairs needed to execute an adapter action.

Pascal

```
property OnGetActionParams: TAdapterActionGetParamsEvent;
```

C++

```
__property TAdapterActionGetParamsEvent OnGetActionParams;
```

Description

Implement this event handler to specify the name/value pairs needed to execute an action.

See Also

OnGetParams (see page 5478)

54.1.76.3.10 TPagedAdapter.OnGetEOF Event

Implement this event to indicate when the current record index has exceeded the total record count.

Pascal

```
property OnGetEOF: TGetEOFEvent;
```

C++

```
__property TGetEOFEvent OnGetEOF;
```

Description

OnGetEOF is called to determine when to stop the iteration. If OnGetEOF is not implemented, the result of the OnGetRecordCount event handler will be compared to the result of the OnGetRecordIndex event handler.

54.1.76.3.11 TPagedAdapter.OnGetFirstRecord Event

Implement this method to initialize the iteration index to zero.

Pascal

```
property OnGetFirstRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetFirstRecord;
```

Description

This method is called when the iteration is started. If there are no records then set the EOF flag.

54.1.76.3.12 TPagedAdapter.OnGetNextRecord Event

Implement this method to increment the iteration index.

Pascal

```
property OnGetNextRecord: TGetRecordEvent;
```

C++

```
__property TGetRecordEvent OnGetNextRecord;
```

Description

This method is called when the next record is requested. This method is not a next then set the EOF flag.

54.1.76.3.13 TPagedAdapter.OnGetRecordCount Event

Implement this event to identify the total number of records that can be iterated.

Pascal

```
property OnGetRecordCount: TGetRecordCountEvent;
```

C++

```
__property TGetRecordCountEvent OnGetRecordCount;
```

Description

OnGetRecordCount is called to calculate the number of pages.

54.1.76.3.14 TPagedAdapter.OnGetRecordIndex Event

Implement this event to identify the zero-based current record.

Pascal

```
property OnGetRecordIndex: TGetRecordIndexEvent;
```

C++

```
__property TGetRecordIndexEvent OnGetRecordIndex;
```

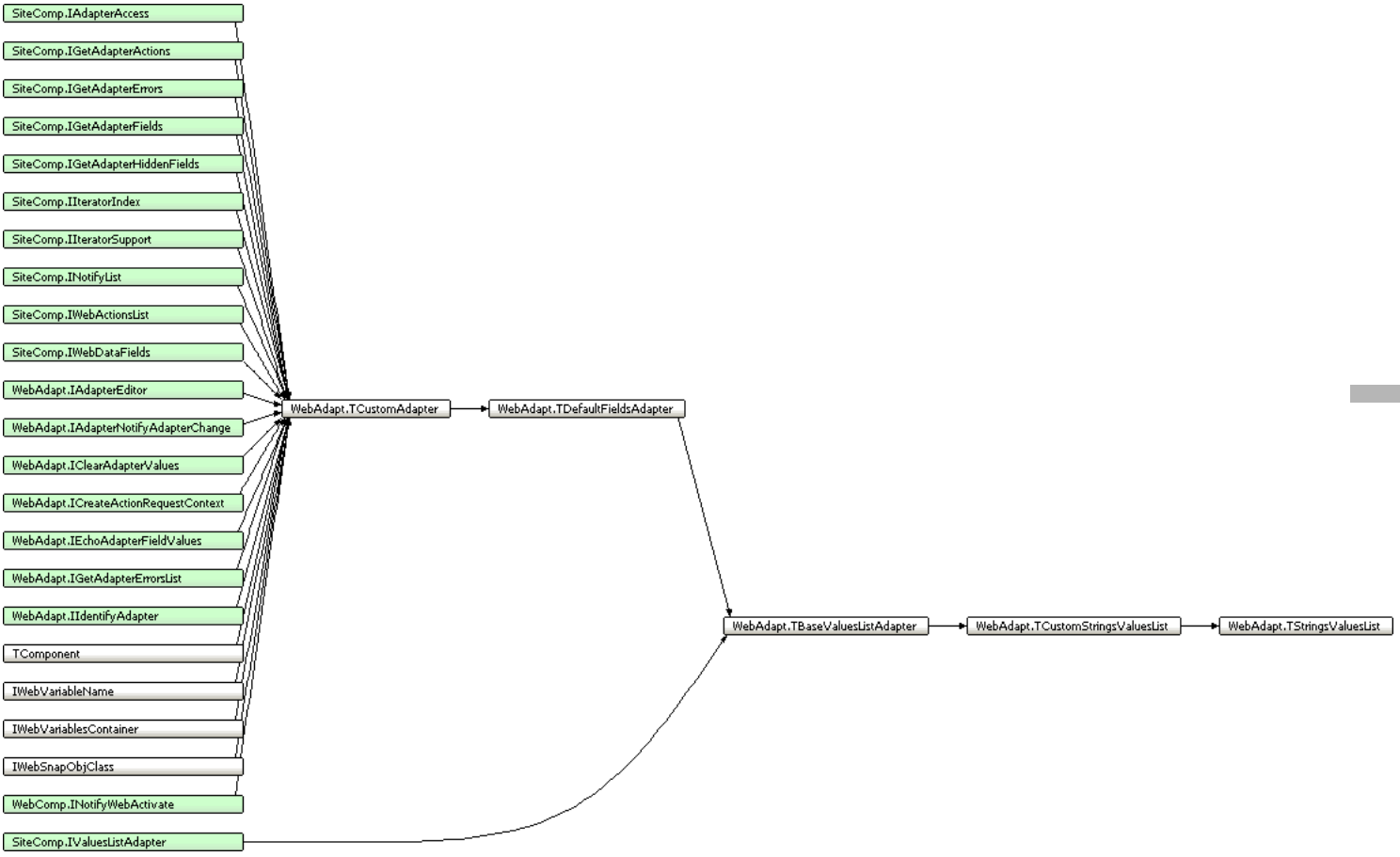
Description

OnGetRecordIndex is called to calculate the current page.

54.1.77 TStringsValuesList Class

TStringsValuesList is an adapter component that provides a list of name/value pairs defined by items in a TStrings property.

Class Hierarchy



Pascal

```
TStringsValuesList = class(TCustomStringsValuesList);
```

C++

```
class TStringsValuesList : public TCustomStringsValuesList;
```

File

WebAdapt

Description

TStringsValuesList defines a list of possible values for an adapter field by assigning the TStringsValuesList component to the ValuesList property of the adapter field. Typically, the values are used to populate HTML elements, such as a select list or radio buttons, when generating HTML to edit an adapter field. The TAdapterPageProducer will automatically generate appropriate HTML/JavaScript for an editable field when the ValuesList is set.

54.1.77.1 TStringsValuesList Members

The following tables list the members exposed by TStringsValuesList.

Protected Methods

	Name	Description
	AddAdapterModeHiddenField (see page 5538)	Defines a hidden field that passes the adapter's mode value in the HTTP request when an HTML form is submitted.
V	AddDefaultActions (see page 5743)	Adds default action components to this adapter.
V	AddDefaultFields (see page 5744)	Adds default field components to this adapter.
	AddOrigValueHiddenFields (see page 5539)	Defines hidden field(s) that pass the original values of adapter fields in the HTTP request when an HTML form is submitted.
	AddRecordKeysHiddenFields (see page 5539)	Defines hidden field(s) that pass key values in the HTTP request when an HTML form is submitted.
	CanAddActionClass (see page 5539)	Indicates whether a type of adapter action can be added to this adapter.
	CanAddFieldClass (see page 5540)	Indicates whether a type of adapter field can be added to this adapter.
	CheckModifyAccess (see page 5540)	Checks end-user rights to modify the adapter's field values.
	ClearAdapterValues (see page 5540)	Clears adapter state information.
	ClearErrors (see page 5541)	Clears the content of the adapter's Errors (see page 5573) property.
	CreateActionRequestContext (see page 5541)	Creates an object that describes the adapter action parameters and adapter field values.
V	CreateErrorsList (see page 5542)	Creates the object used to manage errors associated with this adapter.
	DefaultWebComponentsChange (see page 5744)	Notifies other components of changes to the list of default actions or fields.
V	EncodeActionParams (see page 5542)	Writes name/value pairs that represent adapter state.
	EndIterator (see page 5542)	Implements IIteratorSupport (see page 1054)'s EndIterator method to support iterating over the data records of the adapter.
V	ExtractRequestParams (see page 5543)	Reads name/value pairs from the HTTP request.
	FindAction (see page 5543)	Returns an action component with a matching action name.
	FindField (see page 5543)	Returns an adapter field component with a matching field name.
	FindNamedAction (see page 5544)	Finds an action with a matching variable name.
	FindNamedField (see page 5544)	Finds an adapter field with a matching variable name.
V	FindNamedVariable (see page 5544)	Finds a field or action with a matching variable name.
	FindVariable (see page 5545)	Finds an adapter action or adapter field with the given variable name.
	FindVariableContainer (see page 5545)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	GetActionsClass (see page 5545)	Specifies the class to create as a container for adapter actions components.
V	GetActionsClass (see page 5744)	Specifies the class to create as a container for adapter actions components.
	GetActionsList (see page 5546)	Provides a list of action types that can be added to this adapter.
	GetAdapterActions (see page 5546)	Returns the adapter actions container object.
	GetAdapterErrors (see page 5546)	Returns the adapter's errors object.
	GetAdapterErrorsList (see page 5547)	<insert text here>
	GetAdapterFields (see page 5547)	Returns the adapter fields container object.

	GetAdapterHiddenFields (see page 5547)	Returns a list of hidden fields that hold adapter state information.
	GetAdapterHiddenRecordFields (see page 5548)	Returns a list of hidden fields that associate HTML form fields and adapter field.
V	GetChildren (see page 5548)	Provides the interface for a method that adapters use to return their child components.
V	GetDataClass (see page 5548)	Specifies the class to create as a container for adapter fields components.
V	GetDataClass (see page 5745)	Specifies the class to create as a container for adapter field components.
	GetEchoActionFieldValues (see page 5549)	Returns the value of the echo flag.
	GetFieldsList (see page 5549)	Provides a list of field types that can be added to this adapter.
V	GetHiddenFieldOptions (see page 5549)	Indicates the types of hidden fields that this adapter requires in a submitted HTML form.
	GetIteratorIndex (see page 5550)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex method to support iterating over the data records of the adapter.
	GetListImage (see page 5522)	Returns an image associated with the current name/value pair.
	GetListImageOfValue (see page 5523)	Looks up an image for a value.
	GetListName (see page 5523)	Responsible returning the name of the name/value pair.
	GetListNameOfValue (see page 5523)	Looks up a name for a value.
	GetListValue (see page 5524)	Responsible for returning the value of the name/value pair.
V	GetNameFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetNameFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the value of the name/value pair.
V	GetValueFieldClass (see page 5524)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
V	GetValueFieldClass (see page 5734)	Returns the class of adapter field that is responsible for providing the name of the name/value pair.
	GetVariable (see page 5550)	Returns an action or field.
	GetVariableCount (see page 5550)	Returns the total count of actions and fields.
	GetVariableName (see page 5551)	Gets the name used to reference this adapter with server-side script.
	GetVisibleActions (see page 5551)	Returns a list of this adapter's action components.
	GetVisibleFields (see page 5551)	Returns a list of this adapter's field components.
V	GetWebSnapObjClass (see page 5524)	Returns the class type for this adapter field.
V	GetWebSnapObjClass (see page 5552)	Returns the class type for this adapter field.
	HasExecuteAccess (see page 5552)	Checks the end-user's right to execute this adapter's actions.
	HasModifyAccess (see page 5552)	Checks the end-user's right to modify this adapter's fields.
	HasViewAccess (see page 5553)	Checks the end-user's right to view this adapter's fields.
V	ImplCanAddActionClass (see page 5553)	Indicates whether a type of adapter action can be added to this adapter.
V	ImplCanAddFieldClass (see page 5525)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplCanAddFieldClass (see page 5553)	Indicates whether a type of adapter field can be added to this adapter.
V	ImplClearAdapterValues (see page 5554)	Clears adapter state information.
V	ImplEndIterator (see page 5554)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the data records of the adapter.
V	ImplEndIterator (see page 5734)	Implements IIteratorSupport (see page 1054)'s EndIterator (see page 5542) method to support iterating over the records of the adapter.
V	ImplFindVariable (see page 5554)	Finds an adapter action or adapter field child with a matching variable name.
V	ImplFindVariableContainer (see page 5555)	Finds an adapter action or adapter field that is a container for other scriptable objects.
V	ImplGetActionsList (see page 5555)	Provides a list of action types that can be added to this adapter.
V	ImplGetEchoActionFieldValues (see page 5555)	Returns the value of the echo flag.
V	ImplGetFieldsList (see page 5525)	Adds default field components to this adapter.
V	ImplGetFieldsList (see page 5556)	Provides a list of field types that can be added to this adapter.
V	ImplGetIteratorIndex (see page 5556)	Implements IIteratorIndex (see page 1052)'s GetIteratorIndex (see page 5550) method to support iterating over the data records of the adapter.
V	ImplGetListImage (see page 5525)	Responsible returning an image associated with the current name/value pair.
V	ImplGetListImageOfValue (see page 5526)	Looks up an image for a value.
V	ImplGetListName (see page 5526)	Responsible returning the name of the name/value pair.
V	ImplGetListName (see page 5735)	Responsible returning the name of the name/value pair.
V	ImplGetListNameOfValue (see page 5526)	Looks up a name for a value.
V	ImplGetListNameOfValue (see page 5735)	Looks up a name for a value.

	ImplGetListValue (see page 5527)	Responsible for returning the value of the name/value pair.
	ImplGetListValue (see page 5735)	Responsible for returning the value of the name/value pair.
	ImplGetVariable (see page 5556)	Returns an action or field.
	ImplGetVariableCount (see page 5557)	Returns the total count of actions and fields.
	ImplGetVariableName (see page 5557)	Gets the name used to reference this adapter with server-side script.
	ImplGetVisibleActions (see page 5557)	Returns a list of this adapter's action components.
	ImplGetVisibleActions (see page 5745)	Returns a list of this adapter's action components.
	ImplGetVisibleFields (see page 5558)	Returns a list of this adapter's field components.
	ImplGetVisibleFields (see page 5745)	Returns a list of this adapter's field components.
	ImplHasExecuteAccess (see page 5558)	Checks the end-user's right to execute this adapter's actions.
	ImplHasModifyAccess (see page 5558)	Checks the end-user's right to modify this adapter's fields.
	ImplHasViewAccess (see page 5559)	Checks the end-user's right to view this adapter's fields.
	ImplInIterator (see page 5559)	Implements IIteratorIndex (see page 1052)'s InIterator (see page 5562) to support iterating over the data records of the adapter.
	ImplNextIteration (see page 5560)	Implements IIteratorSupport (see page 1054)'s NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNextIteration (see page 5736)	Implements IteratorSupport's NextIteration (see page 5564) method to support iterating over the records of the adapter.
	ImplNotifyActivate (see page 5560)	Called when the web module is activated.
	ImplNotifyDeactivate (see page 5560)	Called when the web module is deactivated.
	ImplNotifyDeactivate (see page 5736)	Clears the iteration index.
	ImplSetActions (see page 5561)	Initializes the Actions (see page 5572) property.
	ImplSetActions (see page 5746)	Initializes the Actions (see page 5746) property.
	ImplSetData (see page 5561)	Initializes the Data (see page 5572) property.
	ImplSetData (see page 5746)	Initializes the Data (see page 5747) property.
	ImplSetEchoActionFieldValues (see page 5561)	Sets the echo flag for this adapter's fields.
	ImplStartIterator (see page 5562)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	ImplStartIterator (see page 5736)	Implements IIteratorSupport (see page 1054)'s StartIterator (see page 5567) method to support iterating over the records of the adapter.
	InIterator (see page 5562)	Implements IIteratorIndex (see page 1052)'s InIterator method to support iterating over the data records of the adapter.
	IsActionInUse (see page 5563)	Indicates whether a particular action has been added to the adapter.
	IsFieldInUse (see page 5563)	Indicates whether a particular field has been added to the adapter.
	ListHiddenFields (see page 5563)	Specifies the HTML form hidden fields needed by the adapter.
	ListHiddenRecordFields (see page 5564)	Specifies the HTML form hidden fields needed by the adapter.
	NextIteration (see page 5564)	Implements IIteratorSupport (see page 1054)'s NextIteration method to support iterating over the data records of the adapter.
	Notification (see page 5565)	Clears references to deleted components.
	NotifyActivate (see page 5565)	Called when the web module is activated.
	NotifyAdapterChange (see page 5565)	Notifies components that the adapter has changed in some way.
	NotifyDeactivate (see page 5566)	Called when the web module is deactivated.
	PrepareStrings (see page 5737)	Customizes the content of the strings list.
	SetActions (see page 5566)	Initializes the Actions (see page 5572) property.
	SetData (see page 5566)	Initializes the Data (see page 5572) property.
	SetEchoActionFieldValues (see page 5567)	Sets the echo flag for this adapter's fields.
	StartIterator (see page 5567)	Implements IIteratorSupport (see page 1054)'s StartIterator method to support iterating over the records of the adapter.
	UpdateFieldsExecuteValues (see page 5567)	Executes the adapter's fields with values passed in the HTTP request.
	UpdateFieldsGetAnyChanges (see page 5568)	Determines whether the values in the HTTP request change any adapter fields.
	UpdateFieldsUpdateValues (see page 5568)	Updates the adapter's fields with values passed in the HTTP request.
	UpdateFieldsValidateValues (see page 5569)	Validate the adapter's field values passed in the HTTP request.
	UpdateRecordsGetAnyChanges (see page 5570)	Determines whether values passed in the HTTP request will change any adapter fields.
	UpdateRecordsUpdateValues (see page 5570)	Updates adapter fields with values passed in the HTTP request.
	UpdateRecordsValidateValues (see page 5571)	Validate the adapter's field values passed in the HTTP request.

	WebComponentsChange (see page 5571)	WebComponentsChange is called when the list of adapter fields or adapter actions changes.

Protected Properties

	Name	Description
R	NotifyList (see page 5574)	Keeps track of components that are to be notified when the adapter is changed.
	StringsPrepared (see page 5737)	This flag indicates whether the strings list has been prepared or customized.
R	VisibleActions (see page 5575)	Returns a list of this adapter's action components.
R	VisibleFields (see page 5575)	Returns a list of this adapter's field components.

Public Events

	Name	Description
	OnAfterExecuteAction (see page 5576)	Occurs after an adapter action has executed.
	OnAfterGetActionResponse (see page 5576)	Occurs after the adapter action response is generated.
	OnAfterUpdateFields (see page 5576)	Occurs after values in the HTTP request are used to update adapter fields.
	OnAfterValidateFields (see page 5577)	Occurs after values in the HTTP request are validated by the adapter's fields.
	OnBeforeExecuteAction (see page 5577)	Occurs before an adapter action is executed.
	OnBeforeGetActionResponse (see page 5578)	Occurs before the adapter action response is generated.
	OnBeforeUpdateFields (see page 5578)	Occurs before values in the HTTP request are used to update adapter fields.
	OnBeforeValidateFields (see page 5578)	Occurs before values in the HTTP request are validated by the adapter's fields.
	OnGetActionParams (see page 5579)	Supplies name/value pairs needed to execute an adapter action.
	OnIterateRecords (see page 5579)	Occurs when the adapter needs to iterate over its data records.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.
V	Create (see page 5536)	Constructor for TCustomAdapter (see page 5531).
V	Create (see page 5733)	Create is the constructor for TCustomStringsValuesList (see page 5728).
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.
V	Destroy (see page 5536)	Destructor for TCustomAdapter (see page 5531).
V	Destroy (see page 5733)	Destroy is the destructor for TCustomStringsList.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterActions (see page 1018)	This is GetAdapterActions, a member of class IGetAdapterActions.
	GetAdapterErrors (see page 1020)	This is GetAdapterErrors, a member of class IGetAdapterErrors.
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.
	GetAdapterFields (see page 1021)	This is GetAdapterFields, a member of class IGetAdapterFields.
	GetAdapterHiddenFields (see page 1022)	This is GetAdapterHiddenFields, a member of class IGetAdapterHiddenFields.
	GetAdapterHiddenRecordFields (see page 1022)	This is GetAdapterHiddenRecordFields, a member of class IGetAdapterHiddenFields.
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetIteratorIndex (see page 1053)	Returns the value of the IteratorIndex (see page 1054) property.
	GetListImage (see page 1074)	Returns the image object associated with the current name/value pair.
	GetListImageOfValue (see page 1074)	Returns the image object associated with a specified value.
	GetListName (see page 1075)	Returns the name part of the current name/value pair.
	GetListNameOfValue (see page 1075)	Returns the name associated with a specified value.
	GetListValue (see page 1076)	Returns the value of the current name/value pair.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.

	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	HasExecuteAccess (see page 982)	This is HasExecuteAccess, a member of class IAdapterAccess.
	HasModifyAccess (see page 982)	This is HasModifyAccess, a member of class IAdapterAccess.
	HasViewAccess (see page 982)	This is HasViewAccess, a member of class IAdapterAccess.
	InIterator (see page 1053)	Indicates whether the object is iterating over its items.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.
V	UpdateRecords (see page 5569)	Updates the adapter's fields with the field values passed in an HTTP request.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	EchoActionFieldValues (see page 5572)	Modifies the adapter's echo flag.
R	Errors (see page 5573)	Returns an object used to track errors on behalf of this adapter.
	ExecuteAccess (see page 5573)	Defines the rights required to execute this adapter's action.
R	IteratorIndex (see page 1054)	Indicates the index of the current item.
	ModifyAccess (see page 5574)	Identifies the rights required to update this adapter's fields.
	ViewAccess (see page 5574)	Identifies the rights required to view this adapter's fields.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.

Published Events

	Name	Description
	OnPrepareStrings (see page 5864)	Implement OnPrepareStrings to customize the content of the strings list.

Published Properties

	Name	Description
	Actions (see page 5862)	Container for adapter actions components.
	Data (see page 5863)	Container for adapter field components.
	Strings (see page 5863)	This string list defines the list of name/value pairs.

54.1.77.2 TStringsValuesList Properties

The properties of the TStringsValuesList class are listed here.

Published Properties

	Name	Description
	Actions (see page 5862)	Container for adapter actions components.
	Data (see page 5863)	Container for adapter field components.
	Strings (see page 5863)	This string list defines the list of name/value pairs.

54.1.77.2.1 TStringsValuesList.Actions Property

Container for adapter actions components.

Pascal

```
property Actions: TAdapterActions;
```

C++

```
__property TAdapterDefaultActions Actions;__property TAdapterActions Actions;
```

Description

The Actions property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter actions by name.

See Also

TAdapterDefaultActions (see page 5231)

54.1.77.2.2 TStringsValuesList.Data Property

Container for adapter field components.

Pascal

```
property Data: TAdapterFields;
```

C++

```
__property TAdapterDefaultFields Data;__property TAdapterFields Data;
```

Description

The Data property is responsible for managing a list of child adapter fields. Responsibilities include streaming, insertion, deletion, and lookup of adapter fields by name.

See Also

TAdapterDefaultFields (see page 5237)

54.1.77.2.3 TStringsValuesList.Strings Property

This string list defines the list of name/value pairs.

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

Each line in the string list represents a name/value pair. The name is optional. To specify a name/value enter a line with the name following by "=" followed by the value. For example, "Joe Smith=EMP111" is a named value. "EMP111" is an unnamed value.

54.1.77.3 TStringsValuesList Events

The events of the TStringsValuesList class are listed here.

Published Events

	Name	Description
	OnPrepareStrings (see page 5864)	Implement OnPrepareStrings to customize the content of the strings list.

54.1.77.3.1 TStringsValuesList.OnPrepareStrings Event

Implement OnPrepareStrings to customize the content of the strings list.

Pascal

```
property OnPrepareStrings: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnPrepareStrings;
```

Description

OnPrepareStrings is called before the name/value pairs are retrieved by server-side script.

See Also

TNotifyEvent

PrepareStrings (see page 5737)

54.1.78 TUpdateFileList Class

TUpdateFileList is a container for uploaded files.

Class Hierarchy

Pascal

```
TUpdateFileList = class;
```

C++

```
class TUpdateFileList;
```

File

WebAdapt

Description

TUpdateFileList contains the files that are being uploaded to a particular adapter field. TUpdateFileList is passed as a parameter to event handlers that operate on a list of uploaded files.

See Also

TCustomAdapterFileField (see page 5604)

54.1.78.1 TUpdateFileList Members

The following tables list the members exposed by TUpdateFileList.

Public Methods

	Name	Description
	Create (see page 5865)	Creates an instance of TUpdateFileList (see page 5864).

Public Properties

	Name	Description
	Count (see page 5865)	Returns the count of uploaded files.
	Files (see page 5865)	Returns a particular uploaded file.

54.1.78.2 TUpdateFileList.Create Constructor

Creates an instance of TUpdateFileList (see page 5864).

Pascal

```
constructor Create(AValue: IActionFieldValue);
```

C++

```
__fastcall TUpdateFileList(IActionFieldValue AValue);
```

Description

```
constructor Create(AValue: IActionFieldValueIActionFieldValue);
```

Create initializes an instance of TUpdateFileList (see page 5864) and calls the inherited Create method.

54.1.78.3 TUpdateFileList Properties

The properties of the TUpdateFileList class are listed here.

Public Properties

	Name	Description
	Count (see page 5865)	Returns the count of uploaded files.
	Files (see page 5865)	Returns a particular uploaded file.

54.1.78.3.1 TUpdateFileList.Count Property

Returns the count of uploaded files.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use Count with the Files (see page 5865) property to loop through the uploaded files.

See Also

Files (see page 5865)

54.1.78.3.2 TUpdateFileList.Files Property

Returns a particular uploaded file.

Pascal

```
property Files [I: Integer]: TAbstractWebRequestFile;
```

C++

```
__property TAbstractWebRequestFile Files[int I];
```

Description

Use Files with the Count (see page 5865) property to loop through the uploaded files.

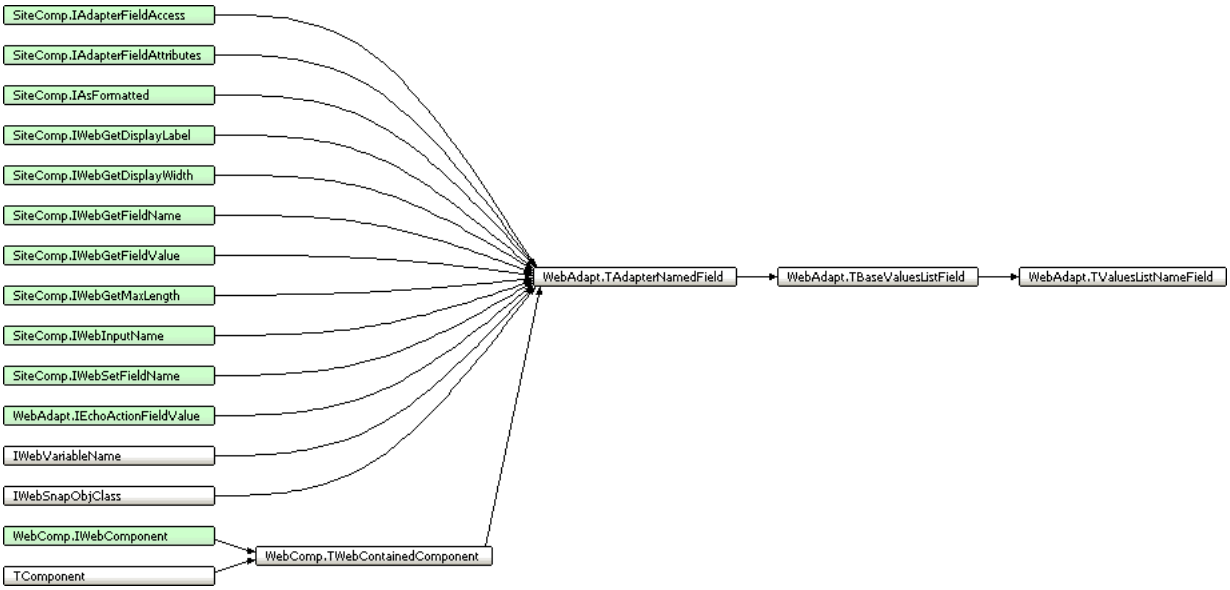
See Also

Count (see page 5865)

54.1.79 TValuesListNameField Class

TValuesListNameField is an adapter field used to access the Name of the current list item of a values list adapter.

Class Hierarchy



Pascal

```
TValuesListNameField = class(TBaseValuesListField);
```

C++

```
class TValuesListNameField : public TBaseValuesListField;
```

File

WebAdapt

Description

Values list adapters such as TStringValuesList contain at least two adapter fields. One adapter field is responsible for name of the current list item. The other adapter field is responsible for providing the value of the current list item. TValuesListNameField is an adapter field that returns the name of the current list item.

See Also

TBaseValuesListField (see page 5527)

TStringsValuesList (see page 5857)

TValuesListValueField (see page 5869)

54.1.79.1 TValuesListNameField Members

The following tables list the members exposed by TValuesListNameField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
V	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.
	GetAdapter (see page 5530)	Retrieves the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5869)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5869)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.

	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	Adapter (see page 5368)	Returns the parent adapter component.
	Adapter (see page 5531)	Retrieves the parent adapter component.
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.

	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.79.2 TValuesListNameField Methods

The methods of the TValuesListNameField class are listed here.

Protected Methods

	Name	Description
V	GetDefaultFieldName (see page 5869)	Returns the name used to access this adapter field using server-side script.
V	ImplGetValue (see page 5869)	Returns the value associated with this adapter field.

54.1.79.2.1 TValuesListNameField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'NameField'. Server-side script can access this adapter field using the NameField property.

54.1.79.2.2 TValuesListNameField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue calls the parent adapter's GetListName function to retrieve the name of the current list item.

See Also

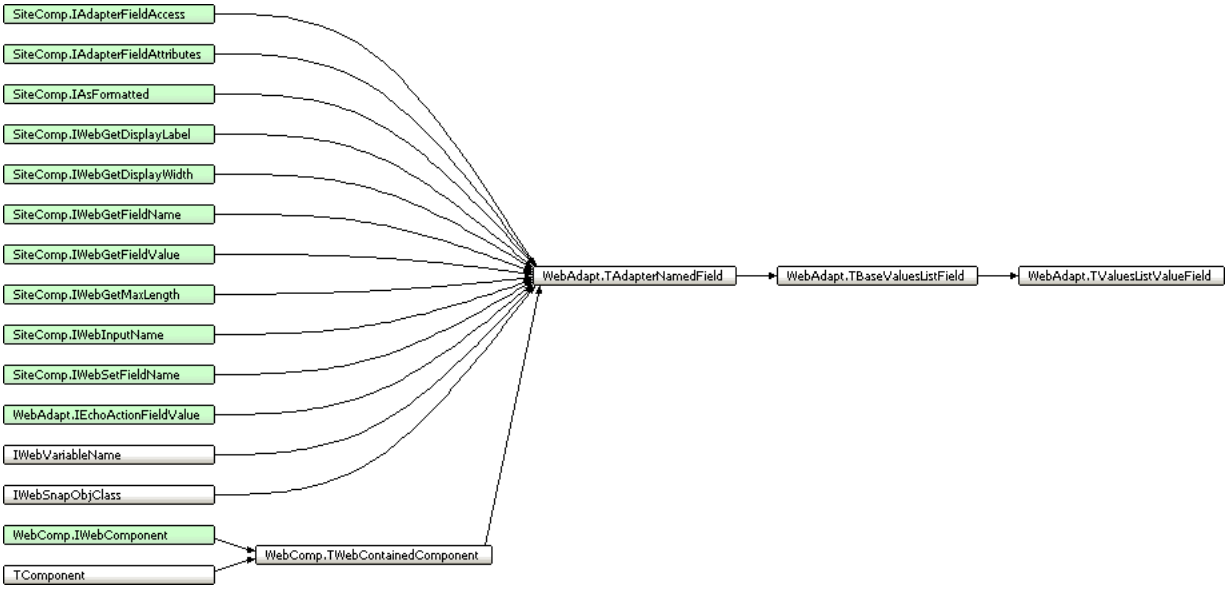
GetListName (see page 5523)

54.1.80 TValuesListValueField Class

TValuesListValueField is an adapter field used to access the Value (see page 5369) of the current list item of a values list

adapter.

Class Hierarchy



Pascal

```
TValuesListValueField = class(TBaseValuesListField);
```

C++

```
class TValuesListValueField : public TBaseValuesListField;
```

File

WebAdapt

Description

Values list adapters such as TStringValuesList contain at least two adapter fields. One adapter field is responsible for name of the current list item. The other adapter field is responsible for providing the value of the current list item. TValuesListValueField is an adapter field that returns the value of the current list item.

See Also

TBaseValuesListField (see page 5527)

TCustomStringsValuesList (see page 5728)

54.1.80.1 TValuesListValueField Members

The following tables list the members exposed by TValuesListValueField.

Protected Methods

	Name	Description
	AsFormatted (see page 5355)	Returns text used to generate a representation of the adapter field value.
	CheckModifyAccess (see page 5355)	Checks end-user rights to modify the adapter field value.
	CheckViewAccess (see page 5356)	Checks end-user rights to view the adapter field value.
	FullyQualifyInputName (see page 5356)	Indicates whether the HTML input names must be qualified with the name of the adapter.
	GetAdapter (see page 5356)	GetAdapter returns the parent adapter component.

	GetAdapter (see page 5530)	Retrieves the parent adapter component.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	GetDefaultDisplayLabel (see page 5357)	Provides the default display label for this adapter field.
V	GetDefaultFieldName (see page 5873)	Returns the name used to access this adapter field using server-side script.
	GetDisplayLabel (see page 5357)	Returns the suggested display label for this adapter field.
	GetDisplayWidth (see page 5358)	Returns the suggested display width for this adapter field.
	GetEchoActionFieldValue (see page 5358)	Returns the value of the echo flag.
	GetInputName (see page 5358)	Identifies the name to use when generating an HTML input for this adapter field.
	GetMaxLength (see page 5359)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
	GetRequired (see page 5359)	Indicates whether the end-user must enter a value for this adapter field.
	GetValue (see page 5359)	Gets the value of this adapter field as a Variant (see page 3421).
	GetVariableName (see page 5360)	Gets the name used to reference this adapter field with server-side script.
	GetVisible (see page 5360)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	GetWebSnapObjClass (see page 5360)	Returns the class type for this adapter field.
	HasModifyAccess (see page 5361)	Checks the end-user's right to modify this adapter field.
	HasViewAccess (see page 5361)	Checks the end-user's right to view this adapter field.
V	ImplAsFormatted (see page 5361)	Returns text used to generate a representation of the adapter field value.
V	ImplGetDisplayLabel (see page 5361)	Returns the suggested display label for this adapter field.
V	ImplGetDisplayWidth (see page 5362)	Returns the suggested display width for this adapter field.
V	ImplGetEchoActionFieldValue (see page 5362)	Returns the value of the echo flag.
V	ImplGetInputName (see page 5362)	Identifies the name to use when generating the HTML input for this adapter field.
V	ImplGetMaxLength (see page 5363)	Returns the maximum length, in characters, that the user should be allowed to enter when entering a value for this adapter field.
V	ImplGetRequired (see page 5363)	Indicates whether the end-user must enter a value for this adapter field.
V	ImplGetValue (see page 5873)	Returns the value associated with this adapter field.
V	ImplGetVariableName (see page 5364)	Gets the name used to reference this adapter field with server-side script.
V	ImplGetVisible (see page 5364)	Indicates whether the value of this adapter field should be displayed to the end-user.
V	ImplHasModifyAccess (see page 5365)	Checks the end-user's right to modify this adapter field.
V	ImplHasViewAccess (see page 5365)	Checks the end-user's right to view this adapter field.
V	ImplSetEchoActionFieldValue (see page 5365)	Sets the echo flag for this adapter field.
V	ImplSetFieldName (see page 5366)	Sets the name that is used to reference this adapter field with server-side script.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 5366)	Handles movement of the component to a different parent.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetEchoActionFieldValue (see page 5366)	Sets the echo flag for this adapter field.
	SetFieldName (see page 5367)	Sets the name that is used to reference this adapter field with server-side script.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
R	Adapter (see page 5368)	Returns the parent adapter component.
R	Adapter (see page 5531)	Retrieves the parent adapter component.
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	AsFormatted (see page 1014)	This is AsFormatted, a member of class IAsFormatted.
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDisplayLabel (see page 1088)	This is GetDisplayLabel, a member of class IWebGetDisplayLabel.
	GetDisplayWidth (see page 1089)	This is GetDisplayWidth, a member of class IWebGetDisplayWidth.
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetInputName (see page 1099)	This is GetInputName, a member of class IWebInputName.
	GetMaxLength (see page 1096)	This is GetMaxLength, a member of class IWebGetMaxLength.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	GetRequired (see page 1000)	This is GetRequired, a member of class IAdapterFieldAttributes.
	GetValue (see page 1092)	This is GetValue, a member of class IWebGetFieldValue.
	GetVisible (see page 1000)	This is GetVisible, a member of class IAdapterFieldAttributes.
	HasModifyAccess (see page 999)	This is HasModifyAccess, a member of class IAdapterFieldAccess.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	HasViewAccess (see page 999)	This is HasViewAccess, a member of class IAdapterFieldAccess.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
R	ActionOrigValue (see page 5367)	Retrieve the original value of this adapter field.
R	ActionValue (see page 5368)	Retrieve the value for this adapter field entered by the end-user.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.
	EchoActionFieldValue (see page 5368)	Used to echo the value(s) entered by the end-user when generating an HTML form.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 5369)	Identifies the name of this field when using server-side script.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.
R	InputName (see page 1100)	This is InputName, a member of class IWebInputName.
	ModifyAccess (see page 5369)	Identifies the rights required to update this field.
R	Required (see page 1001)	This is Required, a member of class IAdapterFieldAttributes.
R	Value (see page 1092)	This is Value, a member of class IWebGetFieldValue.
R	Value (see page 5369)	Provides the value of an adapter field as a variant.
	ViewAccess (see page 5370)	Identifies the rights required to view this field.
R	Visible (see page 1001)	This is Visible, a member of class IAdapterFieldAttributes.

54.1.80.2 TValuesListValueField Methods

The methods of the TValuesListValueField class are listed here.

Protected Methods

	Name	Description
	GetDefaultFieldName (see page 5873)	Returns the name used to access this adapter field using server-side script.
	ImplGetValue (see page 5873)	Returns the value associated with this adapter field.

54.1.80.2.1 TValuesListValueField.GetDefaultFieldName Method

Returns the name used to access this adapter field using server-side script.

Pascal

```
function GetDefaultFieldName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDefaultFieldName();
```

Description

GetDefaultFieldName returns 'ValueField'. Server-side script can access this adapter field using the ValueField property.

54.1.80.2.2 TValuesListValueField.ImplGetValue Method

Returns the value associated with this adapter field.

Pascal

```
function ImplGetValue: Variant; override;
```

C++

```
virtual __fastcall Variant ImplGetValue();
```

Description

ImplGetValue calls the parent adapter's GetListValue function to retrieve the value of the current list item.

See Also

GetListValue (see page 5524)

54.2 Interfaces

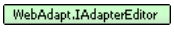
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterEditor (see page 5874)	This is class WebAdapt.IAdapterEditor.
	IAdapterNotifyAdapterChange (see page 5875)	This is class WebAdapt.IAdapterNotifyAdapterChange.
	ICheckOrigValue (see page 5876)	This is class WebAdapt.ICheckOrigValue.
	ICheckValueChange (see page 5877)	This is class WebAdapt.ICheckValueChange.
	IClearAdapterValues (see page 5878)	This is class WebAdapt.IClearAdapterValues.
	ICreateActionRequestContext (see page 5879)	This is class WebAdapt.ICreateActionRequestContext.
	IEchoActionFieldValue (see page 5880)	This is class WebAdapt.IEchoActionFieldValue.
	IEchoAdapterFieldValues (see page 5882)	This is class WebAdapt.IEchoAdapterFieldValues.
	IExecuteValue (see page 5883)	IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.
	IFullyQualifyInputNames (see page 5885)	This is class WebAdapt.IFullyQualifyInputNames.
	IGetAdapterErrorsList (see page 5886)	This is class WebAdapt.IGetAdapterErrorsList.
	IGetAdapterModeNames (see page 5887)	This is class WebAdapt.IGetAdapterModeNames.
	IGetRecordKeys (see page 5888)	This is class WebAdapt.IGetRecordKeys.
	IIdentifyAdapter (see page 5889)	This is class WebAdapt.IIdentifyAdapter.
	ISetSubComponents (see page 5889)	This is class WebAdapt.ISetSubComponents.
	ISetWebFieldValue (see page 5890)	This is class WebAdapt.ISetWebFieldValue.
	IUpdateValue (see page 5891)	This is class WebAdapt.IUpdateValue.
	IValidateValue (see page 5892)	This is class WebAdapt.IValidateValue.
	IWebActionsSubcomponent (see page 5893)	This is class WebAdapt.IWebActionsSubcomponent.
	IWebFieldsSubcomponent (see page 5893)	This is class WebAdapt.IWebFieldsSubcomponent.
	IWebGetFieldPrevValue (see page 5894)	This is class WebAdapt.IWebGetFieldPrevValue.

54.2.1 IAdapterEditor Interface

Class Hierarchy



Pascal

```
IAdapterEditor = interface;
```

C++

```
__interface IAdapterEditor;
```

File

WebAdapt

Description

This is class WebAdapt.IAdapterEditor.

54.2.1.1 IAdapterEditor Members

The following tables list the members exposed by IAdapterEditor.

Public Methods

	Name	Description
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.

54.2.1.2 IAdapterEditor Methods

The methods of the IAdapterEditor class are listed here.

Public Methods

	Name	Description
	CanAddActionClass (see page 5875)	This is CanAddActionClass, a member of class IAdapterEditor.
	CanAddFieldClass (see page 5875)	This is CanAddFieldClass, a member of class IAdapterEditor.

54.2.1.2.1 IAdapterEditor.CanAddActionClass Method

Pascal

```
function CanAddActionClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddActionClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddActionClass, a member of class IAdapterEditor.

54.2.1.2.2 IAdapterEditor.CanAddFieldClass Method

Pascal

```
function CanAddFieldClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

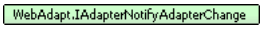
```
__fastcall Boolean CanAddFieldClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddFieldClass, a member of class IAdapterEditor.

54.2.2 IAdapterNotifyAdapterChange Interface

Class Hierarchy



Pascal

```
IAdapterNotifyAdapterChange = interface;
```

C++

```
__interface IAdapterNotifyAdapterChange;
```

File

WebAdapt

Description

This is class WebAdapt.IAdapterNotifyAdapterChange.

54.2.2.1 IAdapterNotifyAdapterChange Members

The following tables list the members exposed by IAdapterNotifyAdapterChange.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

54.2.2.2 IAdapterNotifyAdapterChange Methods

The methods of the IAdapterNotifyAdapterChange class are listed here.

Public Methods

	Name	Description
	NotifyAdapterChange (see page 5876)	This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

54.2.2.2.1 IAdapterNotifyAdapterChange.NotifyAdapterChange Method

Pascal

```
procedure NotifyAdapterChange;
```

C++

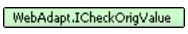
```
__fastcall NotifyAdapterChange();
```

Description

This is NotifyAdapterChange, a member of class IAdapterNotifyAdapterChange.

54.2.3 ICheckOrigValue Interface

Class Hierarchy



Pascal

```
ICheckOrigValue = interface;
```

C++

```
__interface ICheckOrigValue;
```

File

WebAdapt

Description

This is class WebAdapt.ICheckOrigValue.

54.2.3.1 ICheckOrigValue Members

The following tables list the members exposed by ICheckOrigValue.

Public Methods

	Name	Description
	CheckOrigValue (see page 5877)	This is CheckOrigValue, a member of class ICheckOrigValue.

54.2.3.2 ICheckOrigValue Methods

The methods of the ICheckOrigValue class are listed here.

Public Methods

	Name	Description
	CheckOrigValue (see page 5877)	This is CheckOrigValue, a member of class ICheckOrigValue.

54.2.3.2.1 ICheckOrigValue.CheckOrigValue Method

Pascal

```
procedure CheckOrigValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

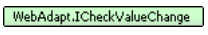
```
__fastcall CheckOrigValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is CheckOrigValue, a member of class ICheckOrigValue.

54.2.4 ICheckValueChange Interface

Class Hierarchy



Pascal

```
ICheckValueChange = interface;
```

C++

```
__interface ICheckValueChange;
```

File

WebAdapt

Description

This is class WebAdapt.ICheckValueChange.

54.2.4.1 ICheckValueChange Members

The following tables list the members exposed by ICheckValueChange.

Public Methods

	Name	Description
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.

54.2.4.2 ICheckValueChange Methods

The methods of the ICheckValueChange class are listed here.

Public Methods

	Name	Description
	CheckValueChange (see page 5878)	This is CheckValueChange, a member of class ICheckValueChange.

54.2.4.2.1 ICheckValueChange.CheckValueChange Method

Pascal

```
function CheckValueChange(AActionRequest: IActionRequest; AFieldIndex: Integer): Boolean;
```

C++

```
__fastcall Boolean CheckValueChange(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is CheckValueChange, a member of class ICheckValueChange.

54.2.5 IClearAdapterValues Interface

Class Hierarchy

WebAdapt.IClearAdapterValues

Pascal

```
IClearAdapterValues = interface;
```

C++

```
__interface IClearAdapterValues;
```

File

WebAdapt

Description

This is class WebAdapt.IClearAdapterValues.

54.2.5.1 IClearAdapterValues Members

The following tables list the members exposed by IClearAdapterValues.

Public Methods

	Name	Description
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.

54.2.5.2 IClearAdapterValues Methods

The methods of the IClearAdapterValues class are listed here.

Public Methods

	Name	Description
	ClearAdapterValues (see page 5879)	This is ClearAdapterValues, a member of class IClearAdapterValues.

54.2.5.2.1 IClearAdapterValues.ClearAdapterValues Method

Pascal

```
procedure ClearAdapterValues;
```

C++

```
__fastcall ClearAdapterValues();
```

Description

This is ClearAdapterValues, a member of class IClearAdapterValues.

54.2.6 ICreateActionRequestContext Interface

Class Hierarchy

WebAdapt.ICreateActionRequestContext

Pascal

```
ICreateActionRequestContext = interface;
```

C++

```
__interface ICreateActionRequestContext;
```

File

WebAdapt

Description

This is class WebAdapt.ICreateActionRequestContext.

54.2.6.1 ICreateActionRequestContext Members

The following tables list the members exposed by ICreateActionRequestContext.

Public Methods

	Name	Description
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

54.2.6.2 ICreateActionRequestContext Methods

The methods of the ICreateActionRequestContext class are listed here.

Public Methods

	Name	Description
	CreateActionRequestContext (see page 5880)	This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

54.2.6.2.1 ICreateActionRequestContext.CreateActionRequestContext Method

Pascal

```
procedure CreateActionRequestContext(DispatchParams: IAdapterDispatchParams);
```

C++

```
__fastcall CreateActionRequestContext(IAdapterDispatchParams DispatchParams);
```

Description

This is CreateActionRequestContext, a member of class ICreateActionRequestContext.

54.2.7 IEchoActionFieldValue Interface

Class Hierarchy

WebAdapt.IEchoActionFieldValue

Pascal

```
IEchoActionFieldValue = interface;
```

C++

```
__interface IEchoActionFieldValue;
```

File

WebAdapt

Description

This is class WebAdapt.IEchoActionFieldValue.

54.2.7.1 IEchoActionFieldValue Members

The following tables list the members exposed by IEchoActionFieldValue.

Public Methods

	Name	Description
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

Public Properties

	Name	Description
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

54.2.7.2 IEchoActionFieldValue Methods

The methods of the IEchoActionFieldValue class are listed here.

Public Methods

	Name	Description
	GetEchoActionFieldValue (see page 5881)	This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.
	SetEchoActionFieldValue (see page 5881)	This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

54.2.7.2.1 IEchoActionFieldValue.GetEchoActionFieldValue Method

Pascal

```
function GetEchoActionFieldValue: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValue();
```

Description

This is GetEchoActionFieldValue, a member of class IEchoActionFieldValue.

54.2.7.2.2 IEchoActionFieldValue.SetEchoActionFieldValue Method

Pascal

```
procedure SetEchoActionFieldValue(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValue(Boolean AValue);
```

Description

This is SetEchoActionFieldValue, a member of class IEchoActionFieldValue.

54.2.7.3 IEchoActionFieldValue Properties

The properties of the IEchoActionFieldValue class are listed here.

Public Properties

	Name	Description
	EchoActionFieldValue (see page 5882)	This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

54.2.7.3.1 IEchoActionFieldValue.EchoActionFieldValue Property

Pascal

```
property EchoActionFieldValue: Boolean;
```

C++

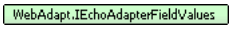
```
__property Boolean EchoActionFieldValue;
```

Description

This is EchoActionFieldValue, a member of class IEchoActionFieldValue.

54.2.8 IEchoAdapterFieldValues Interface

Class Hierarchy



Pascal

```
IEchoAdapterFieldValues = interface;
```

C++

```
__interface IEchoAdapterFieldValues;
```

File

WebAdapt

Description

This is class WebAdapt.IEchoAdapterFieldValues.

54.2.8.1 IEchoAdapterFieldValues Members

The following tables list the members exposed by IEchoAdapterFieldValues.

Public Methods

	Name	Description
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.
	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

54.2.8.2 IEchoAdapterFieldValues Methods

The methods of the IEchoAdapterFieldValues class are listed here.

Public Methods

	Name	Description
	GetEchoActionFieldValues (see page 5883)	This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

	SetEchoActionFieldValues (see page 5883)	This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

54.2.8.2.1 IEchoAdapterFieldValues.GetEchoActionFieldValues Method

Pascal

```
function GetEchoActionFieldValues: Boolean;
```

C++

```
__fastcall Boolean GetEchoActionFieldValues();
```

Description

This is GetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

54.2.8.2.2 IEchoAdapterFieldValues.SetEchoActionFieldValues Method

Pascal

```
procedure SetEchoActionFieldValues(AValue: Boolean);
```

C++

```
__fastcall SetEchoActionFieldValues(Boolean AValue);
```

Description

This is SetEchoActionFieldValues, a member of class IEchoAdapterFieldValues.

54.2.8.3 IEchoAdapterFieldValues Properties

The properties of the IEchoAdapterFieldValues class are listed here.

Public Properties

	Name	Description
	EchoActionFieldValues (see page 5883)	This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

54.2.8.3.1 IEchoAdapterFieldValues.EchoActionFieldValues Property

Pascal

```
property EchoActionFieldValues: Boolean;
```

C++

```
__property Boolean EchoActionFieldValues;
```

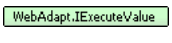
Description

This is EchoActionFieldValues, a member of class IEchoAdapterFieldValues.

54.2.9 IExecuteValue Interface

IExecuteValue is the interface for executing an adapter's fields with values passed in an HTTP request.

Class Hierarchy



Pascal

```
IExecuteValue = interface;
```

C++

```
__interface IExecuteValue;
```

File

WebAdapt

Description

Adapter fields implement the IExecuteValue interface when they are executable. When a field is executable, HTTP requests can supply field values that are used to execute a command rather than update a value. For example, an HTML form may contain a check box or radio button control that indicates an action the user wants to take. When the form is submitted, the values for the field that underlies the check box or radio buttons indicate the user's choice. When the WebSnap application receives the HTTP request, it uses the IExecuteValue interface to call the field's ExecuteValue (see page 5884) method.

Note: Method declarations that use IExecuteValue use the _di_IExecuteValue type instead. This type is a DelphiInterface (see page 3256) wrapper around the IExecuteValue interface:

Note: typedef System::DelphiInterface (see page 3256)< IExecuteValue > _di_IExecuteValue;

See Also

TCustomAdapter (see page 5531)

54.2.9.1 IExecuteValue Members

The following tables list the members exposed by IExecuteValue.

Public Methods

	Name	Description
	ExecuteValue (see page 5884)	Executes the field based on a value supplied in the current HTTP request.

54.2.9.2 IExecuteValue Methods

The methods of the IExecuteValue class are listed here.

Public Methods

	Name	Description
	ExecuteValue (see page 5884)	Executes the field based on a value supplied in the current HTTP request.

54.2.9.2.1 IExecuteValue.ExecuteValue Method

Executes the field based on a value supplied in the current HTTP request.

Pascal

```
procedure ExecuteValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

```
__fastcall ExecuteValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

An adapter calls ExecuteValue when responding to an HTTP request that supplies values from executable fields.

AActionRequest is the current request object. ExecuteValue uses this interface to extract the value to use when executing the field.

AFieldIndex is the index of the target field in the list of field values that the request object supplies.

See Also

UpdateFieldsExecuteValues (see page 5567)

54.2.10 IFullyQualifyInputNames Interface

Class Hierarchy

WebAdapt.IFullyQualifyInputNames

Pascal

```
IFullyQualifyInputNames = interface;
```

C++

```
__interface IFullyQualifyInputNames;
```

File

WebAdapt

Description

This is class WebAdapt.IFullyQualifyInputNames.

54.2.10.1 IFullyQualifyInputNames Members

The following tables list the members exposed by IFullyQualifyInputNames.

Public Methods

	Name	Description
	FullyQualifyInputNames (see page 5885)	This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

54.2.10.2 IFullyQualifyInputNames Methods

The methods of the IFullyQualifyInputNames class are listed here.

Public Methods

	Name	Description
	FullyQualifyInputNames (see page 5885)	This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

54.2.10.2.1 IFullyQualifyInputNames.FullyQualifyInputNames Method

Pascal

```
function FullyQualifyInputNames: Boolean;
```

C++

```
__fastcall Boolean FullyQualifyInputNames();
```

Description

This is FullyQualifyInputNames, a member of class IFullyQualifyInputNames.

54.2.11 IGetAdapterErrorsList Interface

Class Hierarchy

Pascal

```
IGetAdapterErrorsList = interface;
```

C++

```
__interface IGetAdapterErrorsList;
```

File

WebAdapt

Description

This is class WebAdapt.IGetAdapterErrorsList.

54.2.11.1 IGetAdapterErrorsList Members

The following tables list the members exposed by IGetAdapterErrorsList.

Public Methods

	Name	Description
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

54.2.11.2 IGetAdapterErrorsList Methods

The methods of the IGetAdapterErrorsList class are listed here.

Public Methods

	Name	Description
	GetAdapterErrorsList (see page 5886)	This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

54.2.11.2.1 IGetAdapterErrorsList.GetAdapterErrorsList Method

Pascal

```
function GetAdapterErrorsList: TBaseAdapterErrorsList;
```

C++

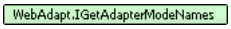
```
__fastcall TBaseAdapterErrorsList GetAdapterErrorsList();
```

Description

This is GetAdapterErrorsList, a member of class IGetAdapterErrorsList.

54.2.12 IGetAdapterModeNames Interface

Class Hierarchy



Pascal

```
IGetAdapterModeNames = interface;
```

C++

```
__interface IGetAdapterModeNames;
```

File

WebAdapt

Description

This is class WebAdapt.IGetAdapterModeNames.

54.2.12.1 IGetAdapterModeNames Members

The following tables list the members exposed by IGetAdapterModeNames.

Public Methods

	Name	Description
	GetAdapterModeNames (see page 5887)	This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

54.2.12.2 IGetAdapterModeNames Methods

The methods of the IGetAdapterModeNames class are listed here.

Public Methods

	Name	Description
	GetAdapterModeNames (see page 5887)	This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

54.2.12.2.1 IGetAdapterModeNames.GetAdapterModeNames Method

Pascal

```
procedure GetAdapterModeNames(AStrings: TStrings);
```

C++

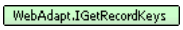
```
__fastcall GetAdapterModeNames(TStrings AStrings);
```

Description

This is GetAdapterModeNames, a member of class IGetAdapterModeNames.

54.2.13 IGetRecordKeys Interface

Class Hierarchy



Pascal

```
IGetRecordKeys = interface;
```

C++

```
__interface IGetRecordKeys;
```

File

WebAdapt

Description

This is class WebAdapt.IGetRecordKeys.

54.2.13.1 IGetRecordKeys Members

The following tables list the members exposed by IGetRecordKeys.

Public Methods

	Name	Description
	GetRecordKeys (see page 5888)	This is GetRecordKeys, a member of class IGetRecordKeys.

54.2.13.2 IGetRecordKeys Methods

The methods of the IGetRecordKeys class are listed here.

Public Methods

	Name	Description
	GetRecordKeys (see page 5888)	This is GetRecordKeys, a member of class IGetRecordKeys.

54.2.13.2.1 IGetRecordKeys.GetRecordKeys Method

Pascal

```
procedure GetRecordKeys(AParams: TStrings; var AFullyQualify: Boolean);
```

C++

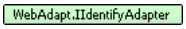
```
__fastcall GetRecordKeys(TStrings AParams, Boolean AFullyQualify);
```

Description

This is GetRecordKeys, a member of class IGetRecordKeys.

54.2.14 IIdentifyAdapter Interface

Class Hierarchy



Pascal

```
IIdentifyAdapter = interface;
```

C++

```
__interface IIdentifyAdapter;
```

File

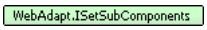
WebAdapt

Description

This is class WebAdapt.IIdentifyAdapter.

54.2.15 ISetSubComponents Interface

Class Hierarchy



Pascal

```
ISetSubComponents = interface;
```

C++

```
__interface ISetSubComponents;
```

File

WebAdapt

Description

This is class WebAdapt.ISetSubComponents.

54.2.15.1 ISetSubComponents Members

The following tables list the members exposed by ISetSubComponents.

Public Methods

	Name	Description
	SetActions (see page 5890)	This is SetActions, a member of class ISetSubComponents.
	SetData (see page 5890)	This is SetData, a member of class ISetSubComponents.

54.2.15.2 ISetSubComponents Methods

The methods of the ISetSubComponents class are listed here.

Public Methods

	Name	Description
	SetActions (see page 5890)	This is SetActions, a member of class ISetSubComponents.
	SetData (see page 5890)	This is SetData, a member of class ISetSubComponents.

54.2.15.2.1 ISetSubComponents.SetActions Method

Pascal

```
procedure SetActions(const Value: TAdapterSubComponent);
```

C++

```
__fastcall SetActions(const TAdapterSubComponent Value);
```

Description

This is SetActions, a member of class ISetSubComponents.

54.2.15.2.2 ISetSubComponents.SetData Method

Pascal

```
procedure SetData(const Value: TAdapterSubComponent);
```

C++

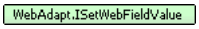
```
__fastcall SetData(const TAdapterSubComponent Value);
```

Description

This is SetData, a member of class ISetSubComponents.

54.2.16 ISetWebFieldValue Interface

Class Hierarchy



Pascal

```
ISetWebFieldValue = interface;
```

C++

```
__interface ISetWebFieldValue;
```

File

WebAdapt

Description

This is class WebAdapt.ISetWebFieldValue.

54.2.16.1 ISetWebFieldValue Members

The following tables list the members exposed by ISetWebFieldValue.

Public Methods

	Name	Description
	SetValue (see page 5891)	This is SetValue, a member of class ISetWebFieldValue.

54.2.16.2 ISetWebFieldValue Methods

The methods of the ISetWebFieldValue class are listed here.

Public Methods

	Name	Description
	SetValue (see page 5891)	This is SetValue, a member of class ISetWebFieldValue.

54.2.16.2.1 ISetWebFieldValue.SetValue Method

Pascal

```
procedure SetValue(const AValue: Variant);
```

C++

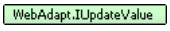
```
__fastcall SetValue(const Variant AValue);
```

Description

This is SetValue, a member of class ISetWebFieldValue.

54.2.17 IUpdateValue Interface

Class Hierarchy



Pascal

```
IUpdateValue = interface;
```

C++

```
__interface IUpdateValue;
```

File

WebAdapt

Description

This is class WebAdapt.IUpdateValue.

54.2.17.1 IUpdateValue Members

The following tables list the members exposed by IUpdateValue.

Public Methods

	Name	Description
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.

54.2.17.2 IUpdateValue Methods

The methods of the IUpdateValue class are listed here.

Public Methods

	Name	Description
	UpdateValue (see page 5892)	This is UpdateValue, a member of class IUpdateValue.

54.2.17.2.1 IUpdateValue.UpdateValue Method

Pascal

```
procedure UpdateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

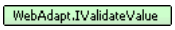
```
__fastcall UpdateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is UpdateValue, a member of class IUpdateValue.

54.2.18 IValidateValue Interface

Class Hierarchy



Pascal

```
IValidateValue = interface;
```

C++

```
__interface IValidateValue;
```

File

WebAdapt

Description

This is class WebAdapt.IValidateValue.

54.2.18.1 IValidateValue Members

The following tables list the members exposed by IValidateValue.

Public Methods

	Name	Description
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

54.2.18.2 IValidateValue Methods

The methods of the IValidateValue class are listed here.

Public Methods

	Name	Description
	ValidateValue (see page 5893)	This is ValidateValue, a member of class IValidateValue.

54.2.18.2.1 IValidateValue.ValidateValue Method

Pascal

```
procedure ValidateValue(AActionRequest: IActionRequest; AFieldIndex: Integer);
```

C++

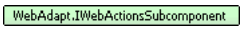
```
__fastcall ValidateValue(IActionRequest AActionRequest, int AFieldIndex);
```

Description

This is ValidateValue, a member of class IValidateValue.

54.2.19 IWebActionsSubcomponent Interface

Class Hierarchy



Pascal

```
IWebActionsSubcomponent = interface;
```

C++

```
__interface IWebActionsSubcomponent;
```

File

WebAdapt

Description

This is class WebAdapt.IWebActionsSubcomponent.

54.2.20 IWebFieldsSubcomponent Interface

Class Hierarchy

Pascal

```
IWebFieldsSubcomponent = interface;
```

C++

```
__interface IWebFieldsSubcomponent;
```

File

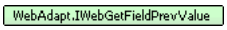
WebAdapt

Description

This is class WebAdapt.IWebFieldsSubcomponent.

54.2.21 IWebGetFieldPrevValue Interface

Class Hierarchy



Pascal

```
IWebGetFieldPrevValue = interface;
```

C++

```
__interface IWebGetFieldPrevValue;
```

File

WebAdapt

Description

This is class WebAdapt.IWebGetFieldPrevValue.

54.2.21.1 IWebGetFieldPrevValue Members

The following tables list the members exposed by IWebGetFieldPrevValue.

Public Methods

	Name	Description
	GetNeedPrevValue (see page 5895)	This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.
	GetPrevValue (see page 5895)	This is GetPrevValue, a member of class IWebGetFieldPrevValue.

Public Properties

	Name	Description
R	NeedPrevValue (see page 5895)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
R	PrevValue (see page 5895)	This is PrevValue, a member of class IWebGetFieldPrevValue.

54.2.21.2 IWebGetFieldPrevValue Methods

The methods of the IWebGetFieldPrevValue class are listed here.

Public Methods

	Name	Description
	GetNeedPrevValue (see page 5895)	This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.
	GetPrevValue (see page 5895)	This is GetPrevValue, a member of class IWebGetFieldPrevValue.

54.2.21.2.1 IWebGetFieldPrevValue.GetNeedPrevValue Method

Pascal

```
function GetNeedPrevValue: Boolean;
```

C++

```
__fastcall Boolean GetNeedPrevValue();
```

Description

This is GetNeedPrevValue, a member of class IWebGetFieldPrevValue.

54.2.21.2.2 IWebGetFieldPrevValue.GetPrevValue Method

Pascal

```
function GetPrevValue: Variant;
```

C++

```
__fastcall Variant GetPrevValue();
```

Description

This is GetPrevValue, a member of class IWebGetFieldPrevValue.

54.2.21.3 IWebGetFieldPrevValue Properties

The properties of the IWebGetFieldPrevValue class are listed here.

Public Properties

	Name	Description
	NeedPrevValue (see page 5895)	This is NeedPrevValue, a member of class IWebGetFieldPrevValue.
	PrevValue (see page 5895)	This is PrevValue, a member of class IWebGetFieldPrevValue.

54.2.21.3.1 IWebGetFieldPrevValue.NeedPrevValue Property

Pascal

```
property NeedPrevValue: Boolean;
```

C++

```
__property Boolean NeedPrevValue;
```

Description

This is NeedPrevValue, a member of class IWebGetFieldPrevValue.

54.2.21.3.2 IWebGetFieldPrevValue.PrevValue Property

Pascal

```
property PrevValue: Variant;
```

C++

```
__property Variant PrevValue;
```

Description

This is PrevValue, a member of class IWebGetFieldPrevValue.

54.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AddDefaultActions (see page 5897)	This is function WebAdapt.AddDefaultActions.
	AddDefaultFields (see page 5897)	This is function WebAdapt.AddDefaultFields.
	FindAdapterAction (see page 5898)	This is function WebAdapt.FindAdapterAction.
	FindAdapterField (see page 5898)	This is function WebAdapt.FindAdapterField.
	FindAdapterInParent (see page 5898)	This is function WebAdapt.FindAdapterInParent.
	MakeAdapterRequestIdentifier (see page 5899)	This is function WebAdapt.MakeAdapterRequestIdentifier.
	MakeValidAdapterFieldName (see page 5899)	This is function WebAdapt.MakeValidAdapterFieldName.
	RaiseFileUploadNotSupported (see page 5899)	This is function WebAdapt.RaiseFileUploadNotSupported.
	RaiseMultipleFilesException (see page 5900)	This is function WebAdapt.RaiseMultipleFilesException.
	RaiseMultipleValuesException (see page 5900)	This is function WebAdapt.RaiseMultipleValuesException.

54.3.1 WebAdapt.AddDefaultActions Function

Pascal

```
procedure AddDefaultActions(AList: TStrings; AComponents: TWebComponentList);
```

C++

```
AddDefaultActions(TStrings AList, TWebComponentList AComponents);
```

File

WebAdapt

Description

This is function WebAdapt.AddDefaultActions.

54.3.2 WebAdapt.AddDefaultFields Function

Pascal

```
procedure AddDefaultFields(AList: TStrings; AComponents: TWebComponentList);
```

C++

```
AddDefaultFields(TStrings AList, TWebComponentList AComponents);
```

File

WebAdapt

Description

This is function WebAdapt.AddDefaultFields.

54.3.3 WebAdapt.FindAdapterAction Function

Pascal

```
function FindAdapterAction(AWebComponents: TWebComponentList; const AName: string): TComponent;
```

C++

```
TComponent * FindAdapterAction(TWebComponentList AWebComponents, const UnicodeString AName);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterAction.

54.3.4 WebAdapt.FindAdapterField Function

Pascal

```
function FindAdapterField(AWebComponents: TWebComponentList; const AName: string): TComponent;
```

C++

```
TComponent * FindAdapterField(TWebComponentList AWebComponents, const UnicodeString AName);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterField.

54.3.5 WebAdapt.FindAdapterInParent Function

Pascal

```
function FindAdapterInParent(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindAdapterInParent(TComponent * AComponent);
```

File

WebAdapt

Description

This is function WebAdapt.FindAdapterInParent.

54.3.6 WebAdapt.MakeAdapterRequestIdentifier Function

Pascal

```
function MakeAdapterRequestIdentifier(AComponent: TComponent): string;
```

C++

```
UnicodeString MakeAdapterRequestIdentifier(TComponent * AComponent);
```

File

WebAdapt

Description

This is function WebAdapt.MakeAdapterRequestIdentifier.

54.3.7 WebAdapt.MakeValidAdapterFieldName Function

Pascal

```
function MakeValidAdapterFieldName(const AFieldName: string): string;
```

C++

```
UnicodeString MakeValidAdapterFieldName(const UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.MakeValidAdapterFieldName.

54.3.8 WebAdapt.RaiseFileUploadNotSupported Function

Pascal

```
procedure RaiseFileUploadNotSupported(AFieldName: string);
```

C++

```
RaiseFileUploadNotSupported(UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseFileUploadNotSupported.

54.3.9 WebAdapt.RaiseMultipleFilesException Function

Pascal

```
procedure RaiseMultipleFilesException(AFieldName: string);
```

C++

```
RaiseMultipleFilesException(UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseMultipleFilesException.

54.3.10 WebAdapt.RaiseMultipleValuesException Function

Pascal

```
procedure RaiseMultipleValuesException(AFieldName: string);
```

C++

```
RaiseMultipleValuesException(UnicodeString AFieldName);
```

File

WebAdapt

Description

This is function WebAdapt.RaiseMultipleValuesException.

54.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TAdapterHiddenFieldOption (see page 5901)	TAdapterHiddenFieldOptions (see page 5919) identifies the types of hidden fields an adapter adds to generated HTML forms.
	TIteratorMethod (see page 5902)	TIteratorMethod identifies a record of data to which an adapter should navigate.

54.4.1 WebAdapt.TAdapterHiddenFieldOption Enumeration

TAdapterHiddenFieldOptions (see page 5919) identifies the types of hidden fields an adapter adds to generated HTML forms.

Pascal

```
TAdapterHiddenFieldOption = (
  hoOrigValues,
  hoMode,
  hoRecordKeys
);
```

C++

```
enum TAdapterHiddenFieldOption {
  hoOrigValues,
  hoMode,
  hoRecordKeys
};
```

File

WebAdapt

Description

Adapters can add hidden fields to HTML forms that are then passed back in HTML request messages that originate from the form. TAdapterHiddenFieldOptions (see page 5919) describes the types of hidden fields that an adapter needs to add to function properly. It represents zero or more TAdapterHiddenFieldOption values.

The following table lists the possible values:

Value	Meaning
hoOrigValues	Hidden fields represent fields from the adapter and the values of those fields at the point when the form is generated.
hoMode	Hidden fields indicate the mode of the adapter.
hoRecordKeys	Hidden fields represent name/value pairs that allow the adapter to locate a record.

54.4.2 WebAdapt.TIteratorMethod Enumeration

TIteratorMethod identifies a record of data to which an adapter should navigate.

Pascal

```
TIteratorMethod = (
  itStart,
  itNext,
  itEnd
);
```

C++

```
enum TIteratorMethod {
  itStart,
  itNext,
  itEnd
};
```

File

WebAdapt

Description

TIteratorMethod indicates a target record from the set of all data records that belong to an adapter. The following table describes the possible values:

The different values of TIteratorMethod are explained below:

Value	Action
itStart	Reset (see page 3643) the iteration pointer to the first record. Set EOF if there are no records.
itNext	Increment the iteration pointer to the next record. Set EOF if the iteration pointer is out of range.
itEnd	The iteration has completed.

54.5 Types

The following table lists types in this documentation.

Types

Name	Description
IActionFieldValue (see page 5904)	This is type WebAdapt.IActionFieldValue.
IActionRequest (see page 5905)	This is type WebAdapt.IActionRequest.
TAbstractWebRequestFile (see page 5905)	This is type WebAdapt.TAbstractWebRequestFile.
TActionExecuteEvent (see page 5905)	TActionExecuteEvent is the type for event handlers on adapter action objects that are called when the action executes.
TActionExecuteHandledEvent (see page 5906)	TActionExecuteHandledEvent is the type for event handlers on adapter action objects that can handle the execution of the action.
TActionGetEnabledEvent (see page 5906)	TActionGetEnabledEvent is the type for event handlers on adapter action objects that are called when the action executes.
TActionGetParamsEvent (see page 5907)	TActionGetParamsEvent is the type for event handlers that supply the parameters for an adapter action.
TActionRequestFieldsEvent (see page 5907)	TActionRequestFieldsEvent is the type for event handlers on adapter action objects that are called after the adapter performs some operation on its fields in response to an HTTP request.
TActionRequestFieldsHandledEvent (see page 5908)	TActionRequestFieldsHandledEvent is the type for event handlers on adapter action objects that are called when the adapter performs some operation on its fields in response to an HTTP request.
TAdapterActionExecuteEvent (see page 5909)	TAdapterActionExecuteEvent is the type for event handlers on adapter objects that are called after one of its actions executes.
TAdapterActionExecuteHandledEvent (see page 5909)	TAdapterActionExecuteHandledEvent is the type for event handlers on adapter objects that are called when one of its actions executes.
TAdapterActionGetParamsEvent (see page 5910)	TAdapterActionGetParamsEvent is the type for event handlers that supply the parameters needed to execute an adapter action.
TAdapterActionsClass (see page 5911)	TAdapterActionsClass defines the metaclass for TAdapterActions (see page 5209).
TAdapterFieldBooleanValueEvent (see page 5911)	TAdapterFieldBooleanValueEvent is the type for event handlers that respond when a boolean value is assigned to an adapter field.
TAdapterFieldFileEvent (see page 5912)	TAdapterFieldFileEvent is the type for event handlers that perform some action on an uploaded file.
TAdapterFieldFileListEvent (see page 5912)	TAdapterFieldFileListEvent is the type for event handlers that perform some action on a set of uploaded files.
TAdapterFieldGetBooleanEvent (see page 5913)	TAdapterFieldGetBooleanEvent is the type for event handlers that provide a boolean value associated with an adapter field.
TAdapterFieldGetMultiValueEvent (see page 5913)	TAdapterFieldGetMultiValueEvent is the type for event handlers that supply a set of values for an adapter field.
TAdapterFieldGetMultiValueStringEvent (see page 5914)	TAdapterFieldGetMultiValueStringEvent is the type for event handlers that supply a set of string values for an adapter field.
TAdapterFieldGetStringEvent (see page 5914)	TAdapterFieldGetStringEvent is the type for event handlers that provide a string associated with an adapter field.
TAdapterFieldGetValueCountEvent (see page 5915)	TAdapterFieldGetValueCountEvent is the type for event handlers that indicate the number of values associated with an adapter field.
TAdapterFieldGetValueEvent (see page 5915)	TAdapterFieldGetValueEvent is the type for event handlers that provide the value associated with an adapter field.
TAdapterFieldHasValueEvent (see page 5916)	TAdapterFieldHasValueEvent is the type for event handlers that indicate whether a value is assigned to an adapter field.
TAdapterFieldsClass (see page 5916)	This is type WebAdapt.TAdapterFieldsClass.
TAdapterFieldStringEvent (see page 5917)	TAdapterFieldStringEvent is the type for event handlers that respond when a string value is assigned to an adapter field.
TAdapterFieldStringsEvent (see page 5917)	TAdapterFieldStringsEvent is the type for event handlers that respond when a set of string values is assigned to an adapter field.
TAdapterFieldValidateValueEvent (see page 5918)	TAdapterFieldValidateValueEvent is the type for event handlers that validate a value for an adapter field.

TAdapterFieldValueEvent (see page 5918)	TAdapterFieldValueEvent is the type for event handlers that respond when an adapter field's value changes.
TAdapterHiddenFieldOptions (see page 5919)	This is type WebAdapt.TAdapterHiddenFieldOptions.
TEndUserGetNameEvent (see page 5919)	TEndUserGetNameEvent is the type for event handlers that supply the name of the current end user.
TEndUserGetUserIDEvent (see page 5920)	TEndUserGetUserIDEvent is the type for event handlers that supply the user ID of the current end user.
TEndUserHandledEvent (see page 5920)	TEndUserHandledEvent is the type for event handlers that can optionally handle a task for an end user adapter.
TEndUserIsLoggedInEvent (see page 5921)	TEndUserIsLoggedInEvent is the type for event handlers that indicate whether the current end user has logged in.
TEndUserRightsEvent (see page 5921)	TEndUserRightsEvent is the type for event handlers that check whether a user has rights to perform a specific operation.
TEndUserUserIDEvent (see page 5922)	TEndUserUserIDEvent is the type for event handlers that log in a user given the user ID.
TGetEOFEvent (see page 5922)	TGetEOFEvent is the type for event handlers on adapters that indicate when to stop iterating over values.
TGetRecordCountEvent (see page 5923)	TGetRecordCountEvent is the type for event handlers that indicate how many values an adapter has.
TGetRecordEvent (see page 5923)	TGetRecordEvent is the type for event handlers on adapters that position the adapter on one of multiple values.
TGetRecordIndexEvent (see page 5924)	TGetRecordIndexEvent is the type for event handlers that indicate which of multiple values an adapter is positioned on.
THandledEvent (see page 5924)	This is type WebAdapt.THandledEvent.
TImageFieldGetHREFEvent (see page 5925)	TImageFieldGetHREFEvent is the type for event handlers that supply an URL that can be used to access an image field associated with an adapter field..
TImageFieldGetImageEvent (see page 5925)	TImageFieldGetImageEvent is the type for event handlers that supply a stream from which to read an image.
TImageFieldGetParamsEvent (see page 5926)	TImageFieldGetParamsEvent is the type for event handlers that supply a set of parameters to identify the value of an image field.
TIterateRecordsEvent (see page 5926)	TIterateRecordsEvent is the type for event handlers that navigate the records of an adapter.
TLoginUserIDEvent (see page 5927)	TLoginUserIDEvent is the type for event handlers that respond after a user logs in.
TObjectList (see page 5928)	This is type WebAdapt.TObjectList.
TValidateAdapterFieldEvent (see page 5928)	TValidateAdapterFieldEvent is the type for event handlers that validate the values supplied for adapter fields.
TWebFormGetPageNameEvent (see page 5928)	TWebFormGetPageNameEvent is the type for event handlers that supply labels for the controls that navigate to Web pages.

54.5.1 WebAdapt.IActionFieldValue Type

Pascal

```
IActionFieldValue = AdaptReq.IActionFieldValue;
```

C++

```
AdaptReq.IActionFieldValue IActionFieldValue;
```

File

WebAdapt

Description

This is type WebAdapt.IActionFieldValue.

54.5.2 WebAdapt.IActionRequest Type

Pascal

```
IActionRequest = AdaptReq.IActionRequest;
```

C++

```
AdaptReq.IActionRequest IActionRequest;
```

File

WebAdapt

Description

This is type WebAdapt.IActionRequest.

54.5.3 WebAdapt.TAbstractWebRequestFile Type

Pascal

```
TAbstractWebRequestFile = HTTPApp.TAbstractWebRequestFile;
```

C++

```
HTTPApp.TAbstractWebRequestFile TAbstractWebRequestFile;
```

File

WebAdapt

Description

This is type WebAdapt.TAbstractWebRequestFile.

54.5.4 WebAdapt.TActionExecuteEvent Type

TActionExecuteEvent is the type for event handlers on adapter action objects that are called when the action executes.

Pascal

```
TActionExecuteEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TActionExecuteEvent)();
```

File

WebAdapt

Description

TActionExecuteEvent is the type for event handlers of an adapter action that can perform some or all of the action's tasks when the action executes.

Sender is the object that represents the adapter action.

Params lists the parameters for the action, in 'Name=Value' format.

See Also

TCustomDataSetAdapterAction

TBaseAdapterAction (see page 5447)

TCustomAdapterAction (see page 5579)

Procedural types

Creating events

54.5.5 WebAdapt.TActionExecuteHandledEvent Type

TActionExecuteHandledEvent is the type for event handlers on adapter action objects that can handle the execution of the action.

Pascal

```
TActionExecuteHandledEvent = procedure (Sender: TObject; Params: TStrings; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; Params: TStrings; var Handled: Boolean) ( TActionExecuteHandledEvent)();
```

File

WebAdapt

Description

TActionExecuteHandledEvent is the type for event handlers of an adapter action that can perform some or all of the action's tasks when the action executes and prevent the default execution.

Sender is the object that represents the adapter action.

Params lists the parameters for the action, in 'Name=Value' format.

Handled returns true if the event handler fully handled the response. If Handled returns false, the adapter action object proceeds with its default response.

See Also

TBaseAdapterAction (see page 5447)

Procedural types

Creating events

54.5.6 WebAdapt.TActionGetEnabledEvent Type

TActionGetEnabledEvent is the type for event handlers on adapter action objects that are called when the action executes.

Pascal

```
TActionGetEnabledEvent = procedure (Sender: TObject; var Enabled: Boolean) of object;
```

C++

```
(Sender: TObject; var Enabled: Boolean) ( TActionGetEnabledEvent)();
```

File

WebAdapt

Description

TActionGetEnabledEvent is the type for event handlers that let you enable or disable controls that represent an adapter action.

Sender is the object that represents the adapter action.

Enabled returns true if the control for the action should be enabled in the current HTML form, false if it should be disabled.

See Also

TCustomDataSetAdapterAction

TCustomAdapterAction (see page 5579)

Procedural types

Creating events

54.5.7 WebAdapt.TActionGetParamsEvent Type

TActionGetParamsEvent is the type for event handlers that supply the parameters for an adapter action.

Pascal

```
TActionGetParamsEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TActionGetParamsEvent)();
```

File

WebAdapt

Description

TActionGetParamsEvent is the type for event handlers of an adapter action that supply the parameters needed to execute the action.

Sender is the object that represents the adapter action.

Params is a string list that the event handler can fill with parameters. Each parameter should have the format ParamName=ParamValue.

See Also

TBaseAdapterAction (see page 5447)

Procedural types

Creating events

54.5.8 WebAdapt.TActionRequestFieldsEvent Type

TActionRequestFieldsEvent is the type for event handlers on adapter action objects that are called after the adapter performs some operation on its fields in response to an HTTP request.

Pascal

```
TActionRequestFieldsEvent = procedure (Sender: TObject; ActionRequest: IActionRequest;
AdapterFields: TObjectList) of object;
```

C++

```
(Sender: TObject; ActionRequest: IActionRequest; AdapterFields: TObjectList) (
TActionRequestFieldsEvent)();
```

File

WebAdapt

Description

TActionRequestFieldsEvent is the type for event handlers that let you perform your own operations on adapter fields in response to an HTTP request.

Sender is the adapter to which the fields belong.

ActionRequest is an interface to the request object for the current HTTP request.

AdapterFields lists the fields of the adapter for which the current HTTP request includes values.

See Also

TCustomAdapter (see page 5531)

Procedural types

Creating events

54.5.9 WebAdapt.TActionRequestFieldsHandledEvent Type

TActionRequestFieldsHandledEvent is the type for event handlers on adapter action objects that are called when the adapter performs some operation on its fields in response to an HTTP request.

Pascal

```
TActionRequestFieldsHandledEvent = procedure (Sender: TObject; ActionRequest: IActionRequest;
AdapterFields: TObjectList; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; ActionRequest: IActionRequest; AdapterFields: TObjectList; var Handled:
Boolean) ( TActionRequestFieldsHandledEvent)();
```

File

WebAdapt

Description

TActionRequestFieldsHandledEvent is the type for event handlers that let you augment or replace the default operations on adapter fields in response to an HTTP request.

Sender is the adapter to which the fields belong.

ActionRequest is an interface to the request object for the current HTTP request.

AdapterFields lists the fields of the adapter for which the current HTTP request includes values.

Handled returns true if the event handler performs the operation on the field, false if the adapter should continue with its default processing.

See Also

TCustomAdapter (see page 5531)

Procedural types

Creating events

54.5.10 WebAdapt.TAdapterActionExecuteEvent Type

TAdapterActionExecuteEvent is the type for event handlers on adapter objects that are called after one of its actions executes.

Pascal

```
TAdapterActionExecuteEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TAdapterActionExecuteEvent)();
```

File

WebAdapt

Description

TAdapterActionExecuteEvent is the type for event handlers of an adapter that can perform some or all of the tasks required after an action executes.

Sender is the adapter component.

Action is the action object that is executing.

Params lists the parameters for the action, in 'Name=Value' format.

See Also

TCustomAdapter (see page 5531)

Procedural types

Creating events

54.5.11 WebAdapt.TAdapterActionExecuteHandledEvent Type

TAdapterActionExecuteHandledEvent is the type for event handlers on adapter objects that are called when one of its actions executes.

Pascal

```
TAdapterActionExecuteHandledEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) ( TAdapterActionExecuteHandledEvent)();
```

File

WebAdapt

Description

TAdapterActionExecuteHandledEvent is the type for event handlers of an adapter that can perform some or all of the tasks required when an action executes.

Sender is the adapter component.

Action is the action object that is executing.

Params lists the parameters for the action, in 'Name=Value' format.

Handled returns true if the event handler executes the action, false if the adapter should continue with its default processing of the action.

See Also

TCustomAdapter (see page 5531)

Procedural types

Creating events

54.5.12 WebAdapt.TAdapterActionGetParamsEvent Type

TAdapterActionGetParamsEvent is the type for event handlers that supply the parameters needed to execute an adapter action.

Pascal

```
TAdapterActionGetParamsEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings)
of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TAdapterActionGetParamsEvent)();
```

File

WebAdapt

Description

TAdapterActionGetParamsEvent is the type for event handlers of an adapter that can supply the parameters needed to execute one of the adapter's actions.

Sender is the adapter to which the action belongs.

Action is the adapter whose parameters are requested.

Params is a string list to which you can add parameter values in Name=Value format.

See Also

TCustomAdapter (see page 5531)

Procedural types

Creating events

54.5.13 WebAdapt.TAdapterActionsClass Type

TAdapterActionsClass defines the metaclass for TAdapterActions (see page 5209).

Pascal

```
TAdapterActionsClass = class of TAdapterActions;
```

C++

```
TAdapterActions TAdapterActionsClass;
```

File

WebAdapt

Description

TAdapterActionsClass is the metaclass for TAdapterActions (see page 5209). Its value is the class reference for TAdapterActions (see page 5209) or for one of its descendants.

See Also

Class References

54.5.14 WebAdapt.TAdapterFieldBooleanValueEvent Type

TAdapterFieldBooleanValueEvent is the type for event handlers that respond when a boolean value is assigned to an adapter field.

Pascal

```
TAdapterFieldBooleanValueEvent = procedure (Sender: TObject; Value: Boolean) of object;
```

C++

```
(Sender: TObject; Value: Boolean) ( TAdapterFieldBooleanValueEvent)();
```

File

WebAdapt

Description

TAdapterFieldBooleanValueEvent is the type for event handlers that let you store a boolean value that is assigned to an adapter field.

Sender is the adapter field component whose value has just been assigned.

Value is the boolean value assigned to Sender.

See Also

Procedural types

Creating events

54.5.15 WebAdapt.TAdapterFieldFileEvent Type

TAdapterFieldFileEvent is the type for event handlers that perform some action on an uploaded file.

Pascal

```
TAdapterFieldFileEvent = procedure (Sender: TObject; AFile: TAbstractWebRequestFile) of object;
```

C++

```
(Sender: TObject; AFile: TAbstractWebRequestFile) ( TAdapterFieldFileEvent)();
```

File

WebAdapt

Description

TAdapterFieldFileEvent is the type for event handlers that let you perform some function on a file uploaded to an adapter field.

Sender is the adapter field component to which the file is directed.

AFile describes the uploaded file.

See Also

Procedural types

Creating events

54.5.16 WebAdapt.TAdapterFieldFileListEvent Type

TAdapterFieldFileListEvent is the type for event handlers that perform some action on a set of uploaded files.

Pascal

```
TAdapterFieldFileListEvent = procedure (Sender: TObject; Files: TUpdateFileList) of object;
```

C++

```
(Sender: TObject; Files: TUpdateFileList) ( TAdapterFieldFileListEvent)();
```

File

WebAdapt

Description

TAdapterFieldFileListEvent is the type for event handlers that let you perform some function on a set of files uploaded to an adapter field.

Sender is the adapter field component to which the files are directed.

Files lists the uploaded files.

See Also

Procedural types

Creating events

54.5.17 WebAdapt.TAdapterFieldGetBooleanEvent Type

TAdapterFieldGetBooleanEvent is the type for event handlers that provide a boolean value associated with an adapter field.

Pascal

```
TAdapterFieldGetBooleanEvent = procedure (Sender: TObject; var Value: Boolean) of object;
```

C++

```
(Sender: TObject; var Value: Boolean) ( TAdapterFieldGetBooleanEvent)();
```

File

WebAdapt

Description

TAdapterFieldGetBooleanEvent is the type for event handlers that let you provide a boolean value that is associated with an adapter field component.

Sender is the adapter field component whose value is requested.

Value returns the boolean value associated with Sender. Typically, this is the value of the adapter field.

See Also

Procedural types

Creating events

54.5.18 WebAdapt.TAdapterFieldGetMultiValueEvent Type

TAdapterFieldGetMultiValueEvent is the type for event handlers that supply a set of values for an adapter field.

Pascal

```
TAdapterFieldGetMultiValueEvent = procedure (Sender: TObject; Index: Integer; var Value:
Variant) of object;
```

C++

```
(Sender: TObject; Index: Integer; var Value: Variant) ( TAdapterFieldGetMultiValueEvent)();
```

File

WebAdapt

Description

TAdapterFieldGetMultiValueEvent is the type for event handlers that let you provide multiple values associated with an adapter field.

Sender is the adapter field component for which you are supplying values.

Index indicates which value the event handler should supply, where 0 indicates the first value, 1 indicates the second value, and so on.

Value returns the value indicated by the Index parameter.

See Also

Procedural types

Creating events

54.5.19 WebAdapt.TAdapterFieldGetMultiValueStringEvent Type

TAdapterFieldGetMultiValueStringEvent is the type for event handlers that supply a set of string values for an adapter field.

Pascal

```
TAdapterFieldGetMultiValueStringEvent = procedure (Sender: TObject; Index: Integer; var Value: string) of object;
```

C++

```
(Sender: TObject; Index: Integer; var Value: string) (TAdapterFieldGetMultiValueStringEvent)();
```

File

WebAdapt

Description

TAdapterFieldGetMultiValueStringEvent is the type for event handlers that let you provide multiple values associated with an adapter field.

Sender is the adapter field component for which you are supplying values.

Index indicates which value the event handler should supply, where 0 indicates the first value, 1 indicates the second value, and so on.

Value returns the value indicated by the Index parameter.

See Also

Procedural types

Creating events

54.5.20 WebAdapt.TAdapterFieldGetStringEvent Type

TAdapterFieldGetStringEvent is the type for event handlers that provide a string associated with an adapter field.

Pascal

```
TAdapterFieldGetStringEvent = procedure (Sender: TObject; var Value: string) of object;
```

C++

```
(Sender: TObject; var Value: string) ( TAdapterFieldGetStringEvent)();
```

File

WebAdapt

Description

TAdapterFieldGetStringEvent is the type for event handlers that let you provide a string that is associated with an adapter field component.

Sender is the adapter field component whose string is requested.

Value returns the string associated with Sender. Typically, this is the value of the adapter field.

See Also

Procedural types

Creating events

54.5.21 WebAdapt.TAdapterFieldGetValueCountEvent Type

TAdapterFieldGetValueCountEvent is the type for event handlers that indicate the number of values associated with an adapter field.

Pascal

```
TAdapterFieldGetValueCountEvent = procedure (Sender: TObject; var Count: Integer) of object;
```

C++

```
(Sender: TObject; var Count: Integer) ( TAdapterFieldGetValueCountEvent)();
```

File

WebAdapt

Description

TAdapterFieldGetValueCountEvent is the type for event handlers that let you provide a count of the values associated with an adapter field.

Sender is the adapter field component for which you want a count of values.

Count returns the number of values associated with Sender.

See Also

Procedural types

Creating events

54.5.22 WebAdapt.TAdapterFieldGetValueEvent Type

TAdapterFieldGetValueEvent is the type for event handlers that provide the value associated with an adapter field.

Pascal

```
TAdapterFieldGetValueEvent = procedure (Sender: TObject; var Value: Variant) of object;
```

C++

```
(Sender: TObject; var Value: Variant) ( TAdapterFieldGetValueEvent)();
```

File

WebAdapt

Description

TAdapterFieldGetValueEvent is the type for event handlers that let you provide the value of an adapter field component.

Sender is the adapter field component whose value is requested.

Value returns a Variant (see page 3421) that contains the current value.

See Also

Procedural types

Creating events

54.5.23 WebAdapt.TAdapterFieldHasValueEvent Type

TAdapterFieldHasValueEvent is the type for event handlers that indicate whether a value is assigned to an adapter field.

Pascal

```
TAdapterFieldHasValueEvent = procedure (Sender: TObject; const Value: Variant; var HasValue:
Boolean) of object;
```

C++

```
(Sender: TObject; const Value: Variant; var HasValue: Boolean) ( TAdapterFieldHasValueEvent)();
```

File

WebAdapt

Description

TAdapterFieldHasValueEvent is the type for event handlers that let you indicate whether a specified value is assigned to an adapter field.

Sender is the adapter field component to check.

Value is the value that may or may not be one of the adapter field's values.

HasValue returns true if Value specifies a value of the adapter field, false otherwise.

See Also

Procedural types

Creating events

54.5.24 WebAdapt.TAdapterFieldsClass Type

Pascal

```
TAdapterFieldsClass = class of TAdapterFields;
```

C++

```
TAdapterFields TAdapterFieldsClass;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterFieldsClass.

54.5.25 WebAdapt.TAdapterFieldStringEvent Type

TAdapterFieldStringEvent is the type for event handlers that respond when a string value is assigned to an adapter field.

Pascal

```
TAdapterFieldStringEvent = procedure (Sender: TObject; Value: string) of object;
```

C++

```
(Sender: TObject; Value: string) ( TAdapterFieldStringEvent)();
```

File

WebAdapt

Description

TAdapterFieldStringEvent is the type for event handlers that let you respond when a string is assigned to an adapter field component.

Sender is the adapter field component to which the string is assigned.

Value is the newly-assigned string.

See Also

Procedural types

Creating events

54.5.26 WebAdapt.TAdapterFieldStringsEvent Type

TAdapterFieldStringsEvent is the type for event handlers that respond when a set of string values is assigned to an adapter field.

Pascal

```
TAdapterFieldStringsEvent = procedure (Sender: TObject; Values: TStrings) of object;
```

C++

```
(Sender: TObject; Values: TStrings) ( TAdapterFieldStringsEvent)();
```

File

WebAdapt

Description

TAdapterFieldStringsEvent is the type for event handlers that let you respond when a string list is assigned to an adapter field component.

Sender is the adapter field component to which the strings are assigned.

Values is a string list that contains the new set of string values.

See Also

Procedural types

Creating events

54.5.27 WebAdapt.TAdapterFieldValidateValueEvent Type

TAdapterFieldValidateValueEvent is the type for event handlers that validate a value for an adapter field.

Pascal

```
TAdapterFieldValidateValueEvent = procedure (Sender: TObject; AValue: IActionFieldValue; var
Handled: Boolean) of object;
```

C++

```
(Sender: TObject; AValue: IActionFieldValue; var Handled: Boolean) (
TAdapterFieldValidateValueEvent)();
```

File

WebAdapt

Description

TAdapterFieldValidateValueEvent is the type for event handlers that let you check whether a value is valid for a given adapter field. The event handler raises an exception if the value is not valid.

Sender is the adapter field component whose value needs to be validated.

AValue is an interface for obtaining the value of the adapter field.

Handled returns true if the event handler has checked the value, false if the event handler did not validate the value. Note that if the event handler checks the value and finds it invalid, it does not set Handled to false: rather, it raises an exception.

See Also

Procedural types

Creating events

54.5.28 WebAdapt.TAdapterFieldValueEvent Type

TAdapterFieldValueEvent is the type for event handlers that respond when an adapter field's value changes.

Pascal

```
TAdapterFieldValueEvent = procedure (Sender: TObject; Value: Variant) of object;
```

C++

```
(Sender: TObject; Value: Variant) ( TAdapterFieldValueEvent)();
```

File

WebAdapt

Description

TAdapterFieldValueEvent is the type for event handlers that let you respond when a new value has been assigned to an adapter field component.

Sender is the adapter field component whose value was just changed.

Value contains the newly-assigned value.

See Also

Procedural types

Creating events

54.5.29 WebAdapt.TAdapterHiddenFieldOptions Type

Pascal

```
TAdapterHiddenFieldOptions = set of TAdapterHiddenFieldOption;
```

C++

```
TAdapterHiddenFieldOption TAdapterHiddenFieldOptions;
```

File

WebAdapt

Description

This is type WebAdapt.TAdapterHiddenFieldOptions.

54.5.30 WebAdapt.TEndUserGetNameEvent Type

TEndUserGetNameEvent is the type for event handlers that supply the name of the current end user.

Pascal

```
TEndUserGetNameEvent = procedure (Sender: TObject; var Name: string) of object;
```

C++

```
(Sender: TObject; var Name: string) ( TEndUserGetNameEvent)();
```

File

WebAdapt

Description

TEndUserGetNameEvent is the type for event handlers of an adapter that can supply the name that should appear in Web pages for the current end user.

Sender is the end user adapter.

Name returns the display name of the current end user.

See Also

Procedural types

Creating events

54.5.31 WebAdapt.TEndUserGetUserIDEvent Type

TEndUserGetUserIDEvent is the type for event handlers that supply the user ID of the current end user.

Pascal

```
TEndUserGetUserIDEvent = procedure (Sender: TObject; var UserID: Variant) of object;
```

C++

```
(Sender: TObject; var UserID: Variant) ( TEndUserGetUserIDEvent)();
```

File

WebAdapt

Description

TEndUserGetUserIDEvent is the type for event handlers of an adapter that can supply the user ID that identifies the current end user.

Sender is the end user adapter.

UserID returns the ID of the current end user. In applications that include a TWebUserList component, this is the ID of a user maintained by that TWebUserList object.

See Also

Procedural types

Creating events

54.5.32 WebAdapt.TEndUserHandledEvent Type

TEndUserHandledEvent is the type for event handlers that can optionally handle a task for an end user adapter.

Pascal

```
TEndUserHandledEvent = procedure (Sender: TObject; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; var Handled: Boolean) ( TEndUserHandledEvent)();
```

File

WebAdapt

Description

TEndUserHandledEvent is the type for event handlers that can perform some task that is otherwise handled by an end user adapter.

Sender is the end user adapter that will perform the task if the event handler does not do so.

Handled returns true if the event handler peformed the task. It returns false if the event handler did not perform the task. If Handled returns false, the end user adapter performs the task in its default manner.

See Also

Procedural types

Creating events

54.5.33 WebAdapt.TEndUserIsLoggedInEvent Type

TEndUserIsLoggedInEvent is the type for event handlers that indicate whether the current end user has logged in.

Pascal

```
TEndUserIsLoggedInEvent = procedure (Sender: TObject; var IsLoggedIn: Boolean) of object;
```

C++

```
(Sender: TObject; var IsLoggedIn: Boolean) ( TEndUserIsLoggedInEvent)();
```

File

WebAdapt

Description

TEndUserIsLoggedInEvent is the type for event handlers of an adapter that indicate whether the current end user has logged in.

Sender is the end user adapter.

IsLoggedIn returns true of the current user has logged in, false if the current user has not logged in.

See Also

Procedural types

Creating events

54.5.34 WebAdapt.TEndUserRightsEvent Type

TEndUserRightsEvent is the type for event handlers that check whether a user has rights to perform a specific operation.

Pascal

```
TEndUserRightsEvent = procedure (Sender: TObject; Rights: TStrings; var HasRights, Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; Rights: TStrings; var HasRights, Handled: Boolean) ( TEndUserRightsEvent)();
```

File

WebAdapt

Description

TEndUserRightsEvent is the type for event handlers that indicate whether the current end user has the rights that match a specified set of rights strings. Note that the event handler can obtain a reference to the current end user information from the global WebContext (see page 6003).

Sender is the end user adapter that needs to check the current end user's rights.

Rights is a string list, where each string in the list indicates the access rights being queried. Typically, these values come from

the ViewRights property of a Web page information object that describes the page that the user wants to access or of an adapter that the user wants to call.

HasRights returns true if the event handler verifies that the current end user has the rights indicated by the Rights parameter, false if the user does not have sufficient rights.

Handled returns true if the event handler ascertained whether the user had the desired rights, false if the event handler did not determine the current user's rights. If Handled returns false, the end user adapter attempts its own mechanism for verifying rights. Typically, if the application contains a Web user list (TWebUserList), the end user adapter uses that Web user list to check rights; otherwise, it simply grants the user rights.

See Also

WebContext (see page 6003)

Procedural types

Creating events

54.5.35 WebAdapt.TEndUserUserIDEvent Type

TEndUserUserIDEvent is the type for event handlers that log in a user given the user ID.

Pascal

```
TEndUserUserIDEvent = procedure (Sender: TObject; UserID: Variant) of object;
```

C++

```
(Sender: TObject; UserID: Variant) ( TEndUserUserIDEvent)();
```

File

WebAdapt

Description

TEndUserUserIDEvent is the type for event handlers of an adapter that take a user ID and log in the identified user.

Sender is the end user adapter.

UserID is the identifier of the user to log in. If the application includes a TWebUserList object, it is the value of the UserID property of the entry for the user in the WebUserList component.

See Also

Procedural types

Creating events

54.5.36 WebAdapt.TGetEOFEvent Type

TGetEOFEvent is the type for event handlers on adapters that indicate when to stop iterating over values.

Pascal

```
TGetEOFEvent = procedure (Sender: TObject; var Eof: Boolean) of object;
```

C++

```
(Sender: TObject; var Eof: Boolean) ( TGetEOFEvent)();
```

File

WebAdapt

Description

Sender is the adapter over whose values the application is iterating.

EOF returns true if the adapter is positioned on the last value, false if it can move to another value.

See Also

Procedural types

Creating events

54.5.37 WebAdapt.TGetRecordCountEvent Type

TGetRecordCountEvent is the type for event handlers that indicate how many values an adapter has.

Pascal

```
TGetRecordCountEvent = procedure (Sender: TObject; var Count: Integer) of object;
```

C++

```
(Sender: TObject; var Count: Integer) ( TGetRecordCountEvent)();
```

File

WebAdapt

Description

Sender is the adapter that has multiple values over which the application can iterate.

Count returns the number of values that the adapter has.

See Also

Procedural types

Creating events

54.5.38 WebAdapt.TGetRecordEvent Type

TGetRecordEvent is the type for event handlers on adapters that position the adapter on one of multiple values.

Pascal

```
TGetRecordEvent = procedure (Sender: TObject; var Eof: Boolean) of object;
```

C++

```
(Sender: TObject; var Eof: Boolean) ( TGetRecordEvent)();
```

File

WebAdapt

Description

Sender is the adapter over whose values the application is iterating.

EOF returns true if, after moving to a new value, the adapter is positioned on the last value.

See Also

Procedural types

Creating events

54.5.39 WebAdapt.TGetRecordIndexEvent Type

TGetRecordIndexEvent is the type for event handlers that indicate which of multiple values an adapter is positioned on.

Pascal

```
TGetRecordIndexEvent = procedure (Sender: TObject; var Index: Integer) of object;
```

C++

```
(Sender: TObject; var Index: Integer) ( TGetRecordIndexEvent)();
```

File

WebAdapt

Description

Sender is the adapter that has multiple values over which the application can iterate.

Index returns the index of the current value. This is 0 if the adapter is positioned on the first value, 1 if it is positioned on the second value, and so on.

See Also

Procedural types

Creating events

54.5.40 WebAdapt.THandledEvent Type

Pascal

```
THandledEvent = procedure (Sender: TObject; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; var Handled: Boolean) ( THandledEvent)();
```

File

WebAdapt

Description

This is type WebAdapt.THandledEvent.

54.5.41 WebAdapt.TImageFieldGetHREFEvent Type

TImageFieldGetHREFEvent is the type for event handlers that supply an URL that can be used to access an image field associated with an adapter field..

Pascal

```
TImageFieldGetHREFEvent = procedure (Sender: TObject; var HREF: string) of object;
```

C++

```
(Sender: TObject; var HREF: string) ( TImageFieldGetHREFEvent)();
```

File

WebAdapt

Description

TImageFieldGetHREFEvent is the type for the OnGetHREF event handler of an adapter field whose value is an image. This event handler lets you supply the URL that clients use to retrieve the image associated with the adapter field. The event occurs when server-side script uses the AsHREF property of the adapter field.

Sender is the adapter field object that represents an image.

Handled returns an indication of how the adapter dispatcher should proceed after the event handler exits. If the HREF returns the URL that clients use to request the field's image from the adapter field. If the TImageFieldGetHREFEvent event handler does not supply a URL, the adapter field generates an URL automatically based on the request parameters for the image file.

See Also

OnGetHREF (see page 5317)

Procedural types

Creating events

54.5.42 WebAdapt.TImageFieldGetImageEvent Type

TImageFieldGetImageEvent is the type for event handlers that supply a stream from which to read an image.

Pascal

```
TImageFieldGetImageEvent = procedure (Sender: TObject; Params: TStrings; var MimeType: string;
var Image: TStream; var Owned: Boolean) of object;
```

C++

```
(Sender: TObject; Params: TStrings; var MimeType: string; var Image: TStream; var Owned:
Boolean) ( TImageFieldGetImageEvent)();
```

File

WebAdapt

Description

TImageFieldGetImageEvent is the type for the OnGetImage event handler of an adapter field whose value is an image. This event handler lets you supply an image that is encoded using a MIME format.

Sender is the adapter field object that represents an image.

Params is a string list containing parameters that identify the desired image. This list contains strings of the form ParamName=ParamValue.

MimeType is the MIME type/subtype string that indicates how the image is encoded.

Image returns a stream from which the adapter field can read the image.

Owned returns true if the adapter field is not responsible for freeing the stream that Image returns. Owned returns false if the adapter field should free the stream after reading the image.

See Also

OnGetImage (see page 5318)

Procedural types

Creating events

54.5.43 WebAdapt.TImageFieldGetParamsEvent Type

TImageFieldGetParamsEvent is the type for event handlers that supply a set of parameters to identify the value of an image field.

Pascal

```
TImageFieldGetParamsEvent = procedure (Sender: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Params: TStrings) ( TImageFieldGetParamsEvent)();
```

File

WebAdapt

Description

TImageFieldGetParamsEvent is the type for the OnGetParams event handler of an adapter field whose value is an image. This event handler lets you supply a set of parameters that uniquely identify the image that is the adapter field's value.

Sender is the adapter field object that represents an image.

Params is a string list to which the event handler adds parameter values. The strings should have the form ParamName=ParamValue.

See Also

OnGetParams (see page 5318)

Procedural types

Creating events

54.5.44 WebAdapt.TIterateRecordsEvent Type

TIterateRecordsEvent is the type for event handlers that navigate the records of an adapter.

Pascal

```
TIterateRecordsEvent = procedure (Sender: TObject; Action: TIteratorMethod; var Eof: Boolean)
of object;
```

C++

```
(Sender: TObject; Action: TIteratorMethod; var Eof: Boolean) ( TIterateRecordsEvent)();
```

File

WebAdapt

Description

TIterateRecordsEvent is the type for event handlers that allow an adapter to iterate over multiple records of data.

Sender is the adapter to which the action belongs.

Action indicates which record the event handler should move to.

EOF returns true if the event handler moved to the last record of the adapter, false otherwise.

See Also

Procedural types

Creating events

54.5.45 WebAdapt.TLoginUserIDEvent Type

TLoginUserIDEvent is the type for event handlers that respond after a user logs in.

Pascal

```
TLoginUserIDEvent = procedure (Sender: TObject; UserID: Variant) of object;
```

C++

```
(Sender: TObject; UserID: Variant) ( TLoginUserIDEvent)();
```

File

WebAdapt

Description

Sender is the login form adapter.

UserID is the ID of the user that just logged in. If the WebSnap application includes a TWebUserList component, this is the ID that the Web user list associates with the supplied password and user name. If the WebSnap application does not include a TWebUserList component, this is the same as the value of the UserName property.

See Also

Procedural types

Creating events

54.5.46 WebAdapt.TObjectList Type

Pascal

```
TObjectList = Contnrs.TObjectList;
```

C++

```
Contnrs.TObjectList TObjectList;
```

File

WebAdapt

Description

This is type WebAdapt.TObjectList.

54.5.47 WebAdapt.TValidateAdapterFieldEvent Type

TValidateAdapterFieldEvent is the type for event handlers that validate the values supplied for adapter fields.

Pascal

```
TValidateAdapterFieldEvent = procedure (Sender: TObject; Value: Variant; var Handled: Boolean)
of object;
```

C++

```
(Sender: TObject; Value: Variant; var Handled: Boolean) ( TValidateAdapterFieldEvent)();
```

File

WebAdapt

Description

TValidateAdapterFieldEvent is the type for event handlers that check whether the value supplied for an adapter field is valid.

ASender is the adapter component that needs to check the field value.

Value is the value supplied for the adapter field. (The particular field to which this value is assigned is determined by the specific event that uses this type for its event handler.)

Handled returns true if the event handler found Value to be valid. If Value is not valid, the event handler raises an exception. If the event handler can't determine whether Value is valid or invalid, Handled returns false.

See Also

Procedural types

Creating events

54.5.48 WebAdapt.TWebFormGetPageNameEvent Type

TWebFormGetPageNameEvent is the type for event handlers that supply labels for the controls that navigate to Web pages.

Pascal

```
TWebFormGetPageNameEvent = procedure (Sender: TObject; APageIndex: Integer; var Value: string)
of object;
```

C++

```
(Sender: TObject; APageIndex: Integer; var Value: string) ( TWebFormGetPageNameEvent)();
```

File

WebAdapt

Description

TWebFormGetPageNameEvent is the type for event handlers of an adapter that return the name of a Web page.

Sender is the adapter action that needs the Web page name.

APageIndex is the index of the Web page for which a name is needed..

Value returns the name of the specified Web page.

See Also

Procedural types

Creating events

54.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
NotifyPossibleHTMLChange (see page 5930)	This is variable WebAdapt.NotifyPossibleHTMLChange.

54.6.1 WebAdapt.NotifyPossibleHTMLChange Variable

Pascal

```
NotifyPossibleHTMLChange: procedure;
```

C++

```
procedure NotifyPossibleHTMLChange;
```

File

WebAdapt

Description

This is variable WebAdapt.NotifyPossibleHTMLChange.

54.7 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sActionRequestLong (see page 5931)	This is constant WebAdapt.sActionRequestLong.
sCurrentPage (see page 5931)	This is constant WebAdapt.sCurrentPage.
sGotoPageAction (see page 5932)	This is constant WebAdapt.sGotoPageAction.
sImageRequest (see page 5932)	This is constant WebAdapt.sImageRequest.
sLoginPassword (see page 5932)	This is constant WebAdapt.sLoginPassword.
sLoginUserName (see page 5933)	This is constant WebAdapt.sLoginUserName.
sNextPageAction (see page 5933)	This is constant WebAdapt.sNextPageAction.
sPrevPageAction (see page 5933)	This is constant WebAdapt.sPrevPageAction.

54.7.1 WebAdapt.sActionRequestLong Constant

Pascal

```
sActionRequestLong = '__action';
```

C++

```
sActionRequestLong = '__action';
```

File

WebAdapt

Description

This is constant WebAdapt.sActionRequestLong.

54.7.2 WebAdapt.sCurrentPage Constant

Pascal

```
sCurrentPage = '_cp';
```

C++

```
sCurrentPage = '_cp';
```

File

WebAdapt

Description

This is constant WebAdapt.sCurrentPage.

54.7.3 WebAdapt.sGotoPageAction Constant

Pascal

```
sGotoPageAction = 'GotoPage';
```

C++

```
sGotoPageAction = 'GotoPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sGotoPageAction.

54.7.4 WebAdapt.sImageRequest Constant

Pascal

```
sImageRequest = '__img';
```

C++

```
sImageRequest = '__img';
```

File

WebAdapt

Description

This is constant WebAdapt.sImageRequest.

54.7.5 WebAdapt.sLoginPassword Constant

Pascal

```
sLoginPassword = 'Password';
```

C++

```
sLoginPassword = 'Password';
```

File

WebAdapt

Description

This is constant WebAdapt.sLoginPassword.

54.7.6 WebAdapt.sLoginUserName Constant

Pascal

```
sLoginUserName = 'UserName';
```

C++

```
sLoginUserName = 'UserName';
```

File

WebAdapt

Description

This is constant WebAdapt.sLoginUserName.

54.7.7 WebAdapt.sNextPageAction Constant

Pascal

```
sNextPageAction = 'NextPage';
```

C++

```
sNextPageAction = 'NextPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sNextPageAction.

54.7.8 WebAdapt.sPrevPageAction Constant

Pascal

```
sPrevPageAction = 'PrevPage';
```

C++

```
sPrevPageAction = 'PrevPage';
```

File

WebAdapt

Description

This is constant WebAdapt.sPrevPageAction.

55 WebBroker Namespace

This is namespace WebBroker.

Topics

Name	Description
Classes (see page 5936)	The following table lists classes in this documentation.
Types (see page 5941)	The following table lists types in this documentation.
Variables (see page 5942)	The following table lists variables in this documentation.

Classes

	Name	Description
	TWebApplication (see page 5936)	TWebApplication encapsulates a Web server application.

Types

Name	Description
THandleShutdownException (see page 5941)	This is type WebBroker.THandleShutdownException.
TServerExceptionEvent (see page 5941)	This is type WebBroker.TServerExceptionEvent.

Variables

Name	Description
Application (see page 5942)	Represents a web server application for an Internet project.
HandleShutdownException (see page 5942)	This is variable WebBroker.HandleShutdownException.

55.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebApplication (see page 5936)	TWebApplication encapsulates a Web server application.

55.1.1 TWebApplication Class

TWebApplication encapsulates a Web server application.

Class Hierarchy

Pascal

```
TWebApplication = class(TWebRequestHandler);
```

C++

```
class TWebApplication : public TWebRequestHandler;
```

File

WebBroker

Description

TWebApplication provides the fundamental behavior of a Web server application. TWebApplication provides methods to create TWebRequest and TWebResponse objects that represent HTTP request messages and their responses, pass these objects to the dispatcher so that the response can be filled in, and send the response back to the Web client.

Each Web server project automatically declares an Application (see page 5942) variable of type TWebApplication as the instance of the application. For CGI applications, this variable is initialized to a TCGIApplication object in the CGIApp unit. For Apache applications, this variable is initialized to a TApacheApplication object in the ApacheApp unit. For ISAPI applications, this variable is initialized to a TISAPIApplication object in the ISAPIApp unit.

The Application (see page 5942) variable is added to a project by selecting File | New (see page 3619) from the menu of the IDE main window and selecting Web Server Application (see page 5942) in the New (see page 3619) Items dialog. Do not add the Forms unit to the uses list of the project (Delphi) or include Forms.hpp in the project source file (C++). These actions declare a conflicting Application (see page 5942) variable of type TApplication.In addition, do not add the ApacheApp or ISAPIApp unit to a CGI application or the CGIApp unit to an Apache, ISAPI or NSAPI application. The CGIApp unit, the ISAPIApp unit, and the ApacheApp unit each create the initial instance of the Application (see page 5942) variable. Including more than one of these units in the uses list causes the memory for the object created in the first unit to be lost.

See Also

Application (see page 5942)

TCGIApplication

TISAPIApplication

TWebDispatcher

TWebRequest

TWebResponse

55.1.1.1 TWebApplication Members

The following tables list the members exposed by TWebApplication.

Protected Methods

	Name	Description
	ActivateWebModules (see page 6636)	Activates a Web module for the application and returns it.
	DeactivateWebModules (see page 6636)	Deactivates the Web module of the application.
	HandleRequest (see page 6637)	Responds to an HTTP request message received by the server.

Public Methods

	Name	Description
	AddWebModuleFactory (see page 6636)	Adds a Web module factory to the request handler.
V	Create (see page 5937)	Creates an instance of TWebApplication (see page 5936).
V	CreateForm (see page 5938)	Registers the single Web module with the Web application.
V	Destroy (see page 5938)	Destroys an instance of TWebApplication (see page 5936).
	HandleException (see page 6637)	Responds to exceptions related to TWebRequestHandler (see page 6633) objects.
V	Initialize (see page 5939)	Introduces an interface for a method that initializes the application.
V	Run (see page 5939)	Executes the Web application.

Public Properties

	Name	Description
R	ActiveCount (see page 6638)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 6638)	Determines whether the Web modules used by the application are cached for re-use.
R	InactiveCount (see page 6639)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 6640)	Sets or returns the MaxConnections used by the socket.
	Title (see page 5940)	Specifies a title for the Web application.
	WebModuleClass (see page 6640)	Contains a reference to a Web module.
R	WebModuleFactory (see page 6640)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 6641)	The number of Web module factories in use by the request handler.

55.1.1.2 TWebApplication.Create Constructor

Creates an instance of TWebApplication (see page 5936).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebApplication(TComponent * AOwner);
```

Description

Do not call Create. The single Web application object named Application (see page 5942) is instantiated automatically when the CGIApp unit, the ISAPIApp unit, or the ApacheApp unit is included in the uses list of the project.

Create allocates the helper objects used by TWebApplication (see page 5936).

See Also

Create

55.1.1.3 TWebApplication.Destroy Destructor

Destroys an instance of TWebApplication (see page 5936).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebApplication();
```

Description

Never call the Destroy or Free method. The application is freed automatically when the application stops running.

Destroy frees the helper objects that were created at startup, as well as all Web modules created by the application.

See Also

System::TObject::Free (see page 3372)

Run (see page 5939)

55.1.1.4 TWebApplication Methods

The methods of the TWebApplication class are listed here.

Public Methods

	Name	Description
	CreateForm (see page 5938)	Registers the single Web module with the Web application.
	Initialize (see page 5939)	Introduces an interface for a method that initializes the application.
	Run (see page 5939)	Executes the Web application.

55.1.1.4.1 TWebApplication.CreateForm Method

Registers the single Web module with the Web application.

Pascal

```
procedure CreateForm(InstanceClass: TComponentClass; var Reference); virtual;
```

C++

```
virtual __fastcall CreateForm(TComponentClass InstanceClass,  Reference);
```

Description

Do not call CreateForm in an application. The module manager calls CreateForm internally to register the Web module for the Web application.

CreateForm registers the class of the Web module for the application.

See Also

TWebModule

55.1.1.4.2 TWebApplication.Initialize Method

Introduces an interface for a method that initializes the application.

Pascal

```
procedure Initialize; virtual;
```

C++

```
virtual __fastcall Initialize();
```

Description

The Initialize method for TWebApplication (see page 5936) does nothing. Initialize is the first method called for every project. Descendants of TWebApplication (see page 5936) that must perform any initialization before the application starts running override this method for that purpose.

Note: Although Initialize is the first method called in the project source code, it is not the first code that is executed in an application.

See Also

Run (see page 5939)

55.1.1.4.3 TWebApplication.Run Method

Executes the Web application.

Pascal

```
procedure Run; virtual;
```

C++

```
virtual __fastcall Run();
```

Description

When the application is executed, the application's Run method is calledautomatically.

The Run method for TWebApplication (see page 5936) sets up the default error handler and shut down sequence for the application. Descendants of TWebApplication (see page 5936) override the Run method to receive HTTP request messages, create the TWebRequest and TWebResponse objects for incoming requests, and call the HandleRequest method to process those requests.

See Also

TWebRequest

TWebResponse

55.1.1.5 TWebApplication Properties

The properties of the TWebApplication class are listed here.

Public Properties

	Name	Description
	Title (see page 5940)	Specifies a title for the Web application.

55.1.1.5.1 TWebApplication.Title Property

Specifies a title for the Web application.

Pascal

```
property Title: string;
```

C++

```
__property UnicodeString Title;
```

Description

Use the Title property to assign a title to your Web application. This title can be used in HTTP messages that the application sends to identify the application. The default value is taken from the executable DLL/EXE .so filename at runtime.

Do not confuse Title with the Name property. Name is the identifier for the Web application when it is referred to in code.

See Also

Name

55.2 Types

The following table lists types in this documentation.

Types

Name	Description
THandleShutdownException (see page 5941)	This is type WebBroker.THandleShutdownException.
TServerExceptionEvent (see page 5941)	This is type WebBroker.TServerExceptionEvent.

55.2.1 WebBroker.THandleShutdownException Type

Pascal

```
THandleShutdownException = procedure (E: Exception);
```

C++

```
(E: Exception) ( THandleShutdownException)();
```

File

WebBroker

Description

This is type WebBroker.THandleShutdownException.

55.2.2 WebBroker.TServerExceptionEvent Type

Pascal

```
TServerExceptionEvent = procedure (E: Exception; wr: TWebResponse) of object;
```

C++

```
(E: Exception; wr: TWebResponse) ( TServerExceptionEvent)();
```

File

WebBroker

Description

This is type WebBroker.TServerExceptionEvent.

55.3 Variables

The following table lists variables in this documentation.

Variables

Name	Description
Application (see page 5942)	Represents a web server application for an Internet project.
HandleShutdownException (see page 5942)	This is variable WebBroker.HandleShutdownException.

55.3.1 WebBroker.Application Variable

Represents a web server application for an Internet project.

Pascal

```
Application: TWebApplication = nil;
```

C++

```
TWebApplication Application = nil;
```

File

WebBroker

Description

By default, when a new Internet server project is created, an Application object is created and assigned to Application. If the project represents a CGI or Win-CGI server Application, Application is initialized to a TCGIApplication object. If the project represents an Apache server Application, Application is initialized to a TApacheApplication object. If the project represents an ISAPI or NSAPI server Application, Application is initialized to a TISAPIApplication object. Which type of TWebApplication (see page 5936) object gets created depends on whether the project includes the ISAPIApp unit, the ApacheApp unit, or the CGIApp unit.

See Also

TISAPIApplication

TCGIApplication

55.3.2 WebBroker.HandleShutdownException Variable

Pascal

```
HandleShutdownException: THandleShutdownException;
```

C++

```
THandleShutdownException HandleShutdownException;
```

File

WebBroker

Description

This is variable WebBroker.HandleShutdownException.

56 WebBrokerSOAP Namespace

This is namespace WebBrokerSOAP.

Topics

Name	Description
Classes (see page 5946)	The following table lists classes in this documentation.
Functions (see page 5951)	The following table lists functions in this documentation.

Classes

	Name	Description
	THTTPSoapDispatcher (see page 5946)	THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

Functions

	Name	Description
	GetSOAPWebModule (see page 5951)	Returns the Web module that contains the SOAP dispatcher that is processing the current request.

56.1 Classes

The following table lists classes in this documentation.

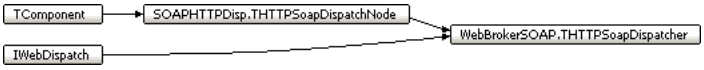
Classes

	Name	Description
	THTTPSoapDispatcher (see page 5946)	THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

56.1.1 THTTPSoapDispatcher Class

THTTPSoapDispatcher responds to SOAP messages by forwarding them to an invoker for interpretation.

Class Hierarchy



Pascal

```
THTTPSoapDispatcher = class(THTTPSoapDispatchNode, IWebDispatch);
```

C++

```
class THTTPSoapDispatcher : public THTTPSoapDispatchNode, public IWebDispatch;
```

File

WebBrokerSOAP

Description

Use THTTPSoapDispatcher in an application that publishes a Web Service using SOAP. THTTPSoapDispatcher responds to all SOAP-based calls made to the invokable interfaces that are registered in the application. These interfaces are registered by calling the RegisterInterface method of the invocation registry (InvRegistry).

THTTPSoapDispatcher only acts as a dispatcher, receiving incoming messages and forwarding them on to another component that handles the task of interpreting and executing them. The component that interprets and executes SOAP message is identified by the Dispatcher property. Dispatcher is the interface of an invoker component, which interprets the SOAP message, identifies the invokable interface that is the target of the call, executes the call, and assembles the content of the response message.

THTTPSoapDispatcher automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the SOAP dispatcher without requiring the use of Web action items.

See Also

THTTPSoapPascalInvoker (see page 1267)

THTTPSoapCppInvoker (see page 1266)

IInvokable (see page 3504)

InvRegistry

TInvokableClassRegistry

Using the SOAP Application Wizard

Writing Servers that Support Web Services

56.1.1.1 THTTPSoapDispatcher Members

The following tables list the members exposed by THTTPSoapDispatcher.

Protected Fields

	Name	Description
	FSoapDispatcher (see page 1259)	This is FSoapDispatcher, a member of class THTTPSoapDispatchNode.

Protected Methods

	Name	Description
	DispatchEnabled (see page 5948)	Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.
	DispatchMask (see page 5949)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 5949)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 5949)	Responds to an incoming message and generates the appropriate response message.
V	Notification (see page 1260)	Responds when objects are added or removed from the Web module.

Public Methods

	Name	Description
V	Create (see page 5947)	Creates a new THTTPSoapDispatcher (see page 5946) object.
V	Destroy (see page 5948)	Frees the memory associated with the THTTPSoapDispatcher (see page 5946) object.
V	DispatchSOAP (see page 1259)	Forwards the SOAP message on to the invoker for interpretation and execution.

Published Properties

	Name	Description
	Dispatcher (see page 1261)	Specifies the interface of the component that interprets SOAP messages.
	WebDispatch (see page 5950)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

56.1.1.2 THTTPSoapDispatcher.Create Constructor

Creates a new THTTPSoapDispatcher (see page 5946) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall THTTPSoapDispatcher(TComponent * AOwner);
```

Description

Most applications do not need to call the THTTPSoapDispatcher (see page 5946) constructor. Typically the HTTP SOAP dispatcher is added to a Web module at design time, so that the constructor is called automatically.

The Owner parameter specifies the component (usually the Web module) that is responsible for freeing the THTTPSoapDispatcher (see page 5946) instance.

See Also

Owner

56.1.1.3 THTTPSoapDispatcher.Destroy Destructor

Frees the memory associated with the THTTPSoapDispatcher (see page 5946) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTTPSoapDispatcher();
```

Description

Do not call Destroy directly. Instead,call Free, which verifies that the THTTPSoapDispatcher (see page 5946) reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

56.1.1.4 THTTPSoapDispatcher Methods

The methods of the THTTPSoapDispatcher class are listed here.

Protected Methods

	Name	Description
	DispatchEnabled (see page 5948)	Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.
	DispatchMask (see page 5949)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 5949)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 5949)	Responds to an incoming message and generates the appropriate response message.

56.1.1.4.1 THTTPSoapDispatcher.DispatchEnabled Method

Indicates whether the HTTP SOAP dispatcher is enabled to respond to incoming SOAP messages.

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

Description

The DispatchEnabled method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to determine whether it can receive incoming messages. In THTTPSoapDispatcher (see page 5946), DispatchEnabled always returns true, because the HTTP SOAP dispatcher can't be disabled.

See Also

WebDispatch (see page 5950)

56.1.1.4.2 THTTPSoapDispatcher.DispatchMask Method

Returns a mask that matches the path portion of any URL to which this object responds.

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

Description

The DispatchMask method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns the Mask property of the TWebDispatch object specified by the WebDispatch (see page 5950) property.

See Also

WebDispatch (see page 5950)

Mask

56.1.1.4.3 THTTPSoapDispatcher.DispatchMethodType Method

Returns the method type of all HTTP messages to which this component responds.

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

Description

The DispatchMethodType method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns the MethodType property of the TWebDispatch object specified by the WebDispatch (see page 5950) property.

See Also

MethodType

WebDispatch (see page 5950)

56.1.1.4.4 THTTPSoapDispatcher.DispatchRequest Method

Responds to an incoming message and generates the appropriate response message.

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

The Web Module (or Web Dispatcher) calls DispatchRequest when it identifies an HTTP message that matches the properties of

the WebDispatch (see page 5950) property. DispatchRequest passes the request on to the component whose interface is the value of the Dispatcher property. That object interprets the request message, calls the appropriate invokable interface, and returns the content of the response message.

Sender is not used in the DispatchRequest method.

Request contains information about the incoming SOAP message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

DispatchRequest returns true if it successfully passes the request on to an invoker and that invoker handles the request and fills out the Response parameter. It returns false if it could not handle the request.

See Also

Dispatcher (see page 1261)

WebDispatch (see page 5950)

56.1.1.5 THTTPSoapDispatcher Properties

The properties of the THTTPSoapDispatcher class are listed here.

Published Properties

	Name	Description
	WebDispatch (see page 5950)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

56.1.1.5.1 THTTPSoapDispatcher.WebDispatch Property

Lets the Web dispatcher identify what HTTP messages should be forwarded to the HTTP SOAP dispatcher.

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

Use WebDispatch to specify the SOAP messages to which this component responds.

The TWebDispatch object that implements WebDispatch includes properties that let you specify the path portion of URLs to which the Web Service application responds, as well as the values allowed in the types of messages it accepts. By default, WebDispatch enables the HTTP SOAP dispatcher to respond to any URL whose path portion begins with the string "soap".

Note: You can't use WebDispatch to disable the THTTPSoapDispatcher (see page 5946) object. Setting its Enabled property to false does nothing.

See Also

TWebModule

TWebDispatcher

56.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetSOAPWebModule (see page 5951)	Returns the Web module that contains the SOAP dispatcher that is processing the current request.

56.2.1 WebBrokerSOAP.GetSOAPWebModule Function

Returns the Web module that contains the SOAP dispatcher that is processing the current request.

Pascal

```
function GetSOAPWebModule: TWebModule;
```

C++

```
TWebModule GetSOAPWebModule();
```

File

WebBrokerSOAP

Description

Call GetSOAPWebModule to access the application Web module of a Web Services application. This is the Web module that contains the SOAP dispatcher (THTTPSoapDispatcher (see page 5946)).

See Also

THTTPSoapDispatcher (see page 5946)

57 WebCntxt Namespace

This is namespace WebCntxt.

Topics

Name	Description
Classes (see page 5955)	The following table lists classes in this documentation.
Interfaces (see page 5988)	The following table lists interfaces in this documentation.
Functions (see page 6003)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6004)	The following table lists structs, records, enums in this documentation.
Types (see page 6005)	The following table lists types in this documentation.
Variables (see page 6007)	The following table lists variables in this documentation.

Classes

	Name	Description
	TAbstractAdapterRequest (see page 5955)	TAbstractAdapterRequest is the base class for adapter request objects.
	TAbstractAdapterResponse (see page 5958)	TAbstractAdapterResponse is the base class for adapter response objects.
	TAbstractWebContext (see page 5960)	TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.
	TWebContext (see page 5976)	TWebContext represents the current state of a WebSnap application.

Functions

	Name	Description
	WebContext (see page 6003)	Returns the current Web context

Interfaces

	Name	Description
	IWebEndUser (see page 5988)	IWebEndUser provides access information about the current end user in a WebSnap application.
	IWebPageInfo (see page 5996)	IWebPageInfo describes a Web page in a WebSnap application.

Structs, Records, Enums

	Name	Description
	TFindPageOption (see page 6004)	TFindPageOptions (see page 6005) indicates the attributes that must be present on a Web page module that is the target of a search.

Types

Name	Description
TFindPageOptions (see page 6005)	This is type WebCntxt.TFindPageOptions.
TGetWebContextProc (see page 6005)	This is type WebCntxt.TGetWebContextProc.
TGetWebEndUserProc (see page 6006)	This is type WebCntxt.TGetWebEndUserProc.
TSetWebContextProc (see page 6006)	This is type WebCntxt.TSetWebContextProc.

Variables

Name	Description
GetModuleFileNameProc (see page 6007)	This is variable WebCntxt.GetModuleFileNameProc.
GetWebContextProc (see page 6007)	This is variable WebCntxt.GetWebContextProc.
GetWebEndUserProc (see page 6008)	This is variable WebCntxt.GetWebEndUserProc.
SetWebContextProc (see page 6008)	This is variable WebCntxt.SetWebContextProc.

57.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractAdapterRequest (see page 5955)	TAbstractAdapterRequest is the base class for adapter request objects.
	TAbstractAdapterResponse (see page 5958)	TAbstractAdapterResponse is the base class for adapter response objects.
	TAbstractWebContext (see page 5960)	TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.
	TWebContext (see page 5976)	TWebContext represents the current state of a WebSnap application.

57.1.1 TAbstractAdapterRequest Class

TAbstractAdapterRequest is the base class for adapter request objects.

Class Hierarchy

Pascal

```
TAbstractAdapterRequest = class(TInterfacedObject);
```

C++

```
class TAbstractAdapterRequest : public TInterfacedObject;
```

File

WebCntxt

Description

TAbstractAdapterRequest is the base class for the object that adapters use to obtain information about the current HTTP request message. An application interacts with an adapter request object through interfaces implemented by various descendants of TAbstractAdapterRequest. These descendant classes are considered internal to WebSnap and are not documented.

See Also

TAbstractWebContext (see page 5960)

IActionRequest

IAdapterModes

IActionFieldValues

IEnumActionFieldValueRecords

IAdapterRecordKeys

IActionFieldValuesOfAdapter

IAdapterDispatchParams (see page 987)

57.1.1.1 TAbstractAdapterRequest Members

The following tables list the members exposed by TAbstractAdapterRequest.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Request (see page 5957)	This is Request, a member of class TAbstractAdapterRequest.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5957)	Create an adapter request object.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.

	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

57.1.1.2 TAbstractAdapterRequest.Create Constructor

Create an adapter request object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractAdapterRequest();
```

Description

Creating an adapter request object automatically assigns that object to the AdapterRequest property of the WebContext (see page 6003) object.

57.1.1.3 TAbstractAdapterRequest Properties

The properties of the TAbstractAdapterRequest class are listed here.

Protected Properties

	Name	Description
R	Request (see page 5957)	This is Request, a member of class TAbstractAdapterRequest.

57.1.1.3.1 TAbstractAdapterRequest.Request Property

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

This is Request, a member of class TAbstractAdapterRequest.

57.1.2 TAbstractAdapterResponse Class

TAbstractAdapterResponse is the base class for adapter response objects.

Class Hierarchy

Pascal

```
TAbstractAdapterResponse = class(TInterfacedObject);
```

C++

```
class TAbstractAdapterResponse : public TInterfacedObject;
```

File

WebCntxt

Description

TAbstractAdapterResponse is the base class for the object that adapters use to obtain information about the current HTTP response message. An application interacts with an adapter response object through interfaces implemented by various descendants of TAbstractAdapterResponse. These descendant classes are considered internal to WebSnap and are not documented.

See Also

TAbstractWebContext (see page 5960)

IActionResponse

57.1.2.1 TAbstractAdapterResponse Members

The following tables list the members exposed by TAbstractAdapterResponse.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Protected Properties

	Name	Description
R	Response (see page 5960)	This is Response, a member of class TAbstractAdapterResponse.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5959)	Create an adapter response object.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

57.1.2.2 TAbstractAdapterResponse.Create Constructor

Create an adapter response object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractAdapterResponse();
```

Description

Creating an adapter response object automatically assigns that object to the AdapterResponse property of the WebContext (see page 6003) object.

57.1.2.3 TAbstractAdapterResponse Properties

The properties of the TAbstractAdapterResponse class are listed here.

Protected Properties

	Name	Description
	Response (see page 5960)	This is Response, a member of class TAbstractAdapterResponse.

57.1.2.3.1 TAbstractAdapterResponse.Response Property

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

This is Response, a member of class TAbstractAdapterResponse.

57.1.3 TAbstractWebContext Class

TAbstractWebContext is the base class for objects that represent the current state of a WebSnap application.

Class Hierarchy

Pascal

```
TAbstractWebContext = class(TObject);
```

C++

```
class TAbstractWebContext : public TObject;
```

File

WebCntxt

Description

A Web context stores information about the components that are available in a WebSnap application and about the current request that is being processed. Applications do not create instances of the Web context. Rather, they access the current Web context object by calling the global WebContext (see page 6003) function.

See Also

WebContext (see page 6003)

TWebContext (see page 5976)

57.1.3.1 TAbstractWebContext Members

The following tables list the members exposed by TAbstractWebContext.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5967)	This is GetAdapterRequest, a member of class TAbstractWebContext.
	GetAdapterResponse (see page 5967)	This is GetAdapterResponse, a member of class TAbstractWebContext.
	GetDispatchedPageName (see page 5968)	This is GetDispatchedPageName, a member of class TAbstractWebContext.
	GetEndUser (see page 5968)	This is GetEndUser, a member of class TAbstractWebContext.
	GetPageCount (see page 5968)	
	GetPageInfo (see page 5968)	
	GetRequestedPageName (see page 5969)	
	GetSession (see page 5969)	
	GetWebModules (see page 5969)	
	GetWebRequest (see page 5969)	
	GetWebResponse (see page 5970)	
	SetAdapterRequest (see page 5970)	This is SetAdapterRequest, a member of class TAbstractWebContext.
	SetAdapterResponse (see page 5970)	This is SetAdapterResponse, a member of class TAbstractWebContext.
	SetDispatchedPageName (see page 5970)	This is SetDispatchedPageName, a member of class TAbstractWebContext.
	SetRequestedPageName (see page 5970)	This is SetRequestedPageName, a member of class TAbstractWebContext.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5962)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 5963)	Frees an instance of TAbstractWebContext (see page 5960).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindApplicationModule (see page 5964)	Returns the application module instance.
	FindDefaultPageInfo (see page 5964)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 5965)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 5965)	Returns an instance of a Web module, given its class name.
	FindPage (see page 5966)	Retrieves a Web page module given its name.
	FindPageInfo (see page 5966)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 5967)	Returns an instance of a Web data module, given its class name.

	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AdapterRequest (see page 5971)	Provides access to the adapter request object that manages access to the current request.
R	AdapterResponse (see page 5972)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 5972)	Specifies the name of the Web page module that is producing content to respond to the current request.
R	EndUser (see page 5973)	Provides access to information about the end user that sent the current request.
R	PageCount (see page 5973)	Indicates the number of registered Web pages available in the WebSnap application.
R	Pages (see page 5973)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
R	Request (see page 5974)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 5974)	Indicates the name of the Web page module to which the current request is addressed.
R	Response (see page 5975)	Represents the current HTTP response that the application is assembling.
R	Session (see page 5975)	Provides access to the current Web session.
R	WebModules (see page 5975)	Lists all the Modules in the WebSnap application.

57.1.3.2 TAbstractWebContext.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAbstractWebContext();
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that Creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

new

57.1.3.3 TAbstractWebContext.Destroy Destructor

Frees an instance of TAbstractWebContext (see page 5960).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractWebContext();
```

Description

Do not free the global Web context object. This object is automatically maintained by the WebSnap application.

The TAbstractWebContext (see page 5960) destructor unregisters itself with the global WebContext (see page 6003) function so that the WebContext (see page 6003) function never returns a Web context that no longer exists.

See Also

WebContext (see page 6003)

57.1.3.4 TAbstractWebContext Methods

The methods of the TAbstractWebContext class are listed here.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5967)	This is GetAdapterRequest, a member of class TAbstractWebContext.
	GetAdapterResponse (see page 5967)	This is GetAdapterResponse, a member of class TAbstractWebContext.
	GetDispatchedPageName (see page 5968)	This is GetDispatchedPageName, a member of class TAbstractWebContext.
	GetEndUser (see page 5968)	This is GetEndUser, a member of class TAbstractWebContext.
	GetPageCount (see page 5968)	
	GetPageInfo (see page 5968)	
	GetRequestedPageName (see page 5969)	
	GetSession (see page 5969)	
	GetWebModules (see page 5969)	
	GetWebRequest (see page 5969)	
	GetWebResponse (see page 5970)	
	SetAdapterRequest (see page 5970)	This is SetAdapterRequest, a member of class TAbstractWebContext.
	SetAdapterResponse (see page 5970)	This is SetAdapterResponse, a member of class TAbstractWebContext.
	SetDispatchedPageName (see page 5970)	This is SetDispatchedPageName, a member of class TAbstractWebContext.
	SetRequestedPageName (see page 5970)	This is SetRequestedPageName, a member of class TAbstractWebContext.

Public Methods

	Name	Description
	FindApplicationModule (see page 5964)	Returns the application module instance.
	FindDefaultPageInfo (see page 5964)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 5965)	Returns an instance of a Web module, given its class reference.

	FindModuleName (see page 5965)	Returns an instance of a Web module, given its class name.
	FindPage (see page 5966)	Retrieves a Web page module given its name.
	FindPageInfo (see page 5966)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 5967)	Returns an instance of a Web data module, given its class name.

57.1.3.4.1 TAbstractWebContext.FindApplicationModule Method

Returns the application module instance.

Pascal

```
function FindApplicationModule(WebModuleContext: TWebModuleContext): TComponent; virtual;
abstract;
```

C++

```
virtual __fastcall TComponent * FindApplicationModule(TWebModuleContext WebModuleContext) = 0;
```

Description

Call FindApplicationModule to access the main Web module in the WebSnap application. FindApplicationModule returns a reference to the module that the WebSnap application wizard creates. This can be an instance of TWebAppPageModule or TWebAppDataModule.

The Application module is the module that holds the application-wide components such as the page dispatcher, end user adapter, application adapter, and so on.

WebModuleContext is a context object that holds design-time information about the application modules. It is only used at design time.

See Also

WebModules (see page 5975)

TWebAppPageModule (see page 6588)

TWebAppDataModule (see page 6585)

57.1.3.4.2 TAbstractWebContext.FindDefaultPageInfo Method

Retrieves the Web page information object for a specified class of Web module.

Pascal

```
function FindDefaultPageInfo(AComponentClass: TComponentClass; out APageInfo:
TAbstractWebPageInfo): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindDefaultPageInfo(TComponentClass AComponentClass,
TAbstractWebPageInfo APageInfo) = 0;
```

Description

Call FindDefaultPageInfo to obtain the Web page information object for a Web module. FindDefaultPageInfo retrieves the information object that is registered for the Web page module class specified by AComponentClass. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

AComponentClass is a class reference to the class that implements the Web page module. This class is declared in code that the Web page module wizard generates.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and

that provides access to the factory object that you can use to create instances of the Web page module.

FindDefaultPageInfo returns false if it can't return a value for the APageInfo parameter. FindDefaultPageInfo returns true if it locates a registered Web page module for the class specified by AComponentClass.

See Also

FindPageInfo (see page 5966)

57.1.3.4.3 TAbstractWebContext.FindModuleClass Method

Returns an instance of a Web module, given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass) = 0;
```

Description

Call FindModuleClass to obtain a reference to a Web module given its class reference. If an instance of the specified class already exists, FindModuleClass returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass creates an instance of the specified class and returns the new instance.

AClass is the class reference to a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleName (see page 5965)

FindPage (see page 5966)

57.1.3.4.4 TAbstractWebContext.FindModuleName Method

Returns an instance of a Web module, given its class name.

Pascal

```
function FindModuleName(const AClass: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const UnicodeString AClass) = 0;
```

Description

Call FindModuleName to obtain a reference to a Web module given the name of its implementation class. If an instance of the specified class already exists, FindModuleName returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass (see page 5965) creates an instance of the specified class and returns the new instance.

AClass is the name of a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleClass (see page 5965)

FindPage (see page 5966)

FindVariableContainer (see page 5967)

57.1.3.4.5 TAbstractWebContext.FindPage Method

Retrieves a Web page module given its name.

Pascal

```
function FindPage(const APage: string; AOptions: TFindPageOptions; out AComponent: TObject):
Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindPage(const UnicodeString APage, TFindPageOptions AOptions,
TObject * AComponent) = 0;
```

Description

Call FindPage to obtain an instance of a registered Web page module given its name. If an instance of the specified Web page module already exists, FindPage returns that instance. Otherwise, if the named Web page module is registered, FindPage creates an instance of the module and returns the new instance.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPage considers. For any option included in this set, FindPage only considers Web page modules that have the corresponding attribute.

AComponent returns the instance of the specified Web page module.

FindPage returns true if it can find or create an instance of the specified Web page module, false otherwise.

See Also

FindModuleName (see page 5965)

FindPageInfo (see page 5966)

FindModuleClass (see page 5965)

57.1.3.4.6 TAbstractWebContext.FindPageInfo Method

Retrieves a Web page information object given the name of the corresponding Web page module.

Pascal

```
function FindPageInfo(const APage: string; AOptions: TFindPageOptions; out APageInfo:
TAbstractWebPageInfo): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean FindPageInfo(const UnicodeString APage, TFindPageOptions AOptions,
TAbstractWebPageInfo APageInfo) = 0;
```

Description

Call FindPageInfo to obtain the Web page information object for a Web page module. FindPageInfo retrieves the information object that is registered for the Web page module that has a specified name. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPageInfo considers. For any option included in this set, FindPageInfo only considers Web page modules that have the corresponding attribute.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindPageInfo returns false if it can't return a value for the APageInfo parameter. FindPageInfo returns true if it locates a registered Web page module with the specified name and attributes.

See Also

FindPage (see page 5966)

FindDefaultPageInfo (see page 5964)

Pages (see page 5973)

57.1.3.4.7 TAbstractWebContext.FindVariableContainer Method

Returns an instance of a Web data module, given its class name.

Pascal

```
function FindVariableContainer(const AIdent: string): TComponent; virtual; abstract;
```

C++

```
virtual __fastcall TComponent * FindVariableContainer(const UnicodeString AIdent) = 0;
```

Description

Call FindVariableContainer to obtain a reference to a Web data module given the name of its implementation class. If an instance of the specified class already exists, FindVariableContainer returns that instance. Otherwise, if the specified class represents a registered Web module class, FindVariableContainer creates an instance of the specified class and returns the new instance.

AIdent is the name of a registered Web data module class. This class is declared in code that the Web data module wizard generates.

See Also

FindModuleName (see page 5965)

57.1.3.4.8 TAbstractWebContext.GetAdapterRequest Method

Pascal

```
function GetAdapterRequest: IUnknown; virtual; abstract;
```

C++

```
virtual __fastcall IUnknown GetAdapterRequest() = 0;
```

Description

This is GetAdapterRequest, a member of class TAbstractWebContext.

57.1.3.4.9 TAbstractWebContext.GetAdapterResponse Method

Pascal

```
function GetAdapterResponse: IUnknown; virtual; abstract;
```

C++

```
virtual __fastcall IUnknown GetAdapterResponse() = 0;
```

Description

This is GetAdapterResponse, a member of class TAbstractWebContext.

57.1.3.4.10 TAbstractWebContext.GetDispatchedPageName Method

Pascal

```
function GetDispatchedPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetDispatchedPageName() = 0;
```

Description

This is GetDispatchedPageName, a member of class TAbstractWebContext.

57.1.3.4.11 TAbstractWebContext.GetEndUser Method

Pascal

```
function GetEndUser: IWebEndUser; virtual; abstract;
```

C++

```
virtual __fastcall IWebEndUser GetEndUser() = 0;
```

Description

This is GetEndUser, a member of class TAbstractWebContext.

57.1.3.4.12 TAbstractWebContext.GetPageCount Method

Pascal

```
function GetPageCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetPageCount() = 0;
```

See Also

GetPageInfo (see page 5968)

PageCount (see page 5973)

57.1.3.4.13 TAbstractWebContext.GetPageInfo Method

Pascal

```
function GetPageInfo(I: Integer): IWebPageInfo; virtual; abstract;
```

C++

```
virtual __fastcall IWebPageInfo GetPageInfo(int I) = 0;
```

See Also

GetPageCount (see page 5968)

Pages (see page 5973)

57.1.3.4.14 TAbstractWebContext.GetRequestedPageName Method

Pascal

```
function GetRequestedPageName: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString GetRequestedPageName() = 0;
```

See Also

GetPageCount (see page 5968)

Pages (see page 5973)

57.1.3.4.15 TAbstractWebContext.GetSession Method

Pascal

```
function GetSession: TAbstractWebSession; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebSession GetSession() = 0;
```

See Also

GetPageCount (see page 5968)

Pages (see page 5973)

57.1.3.4.16 TAbstractWebContext.GetWebModules Method

Pascal

```
function GetWebModules: TAbstractWebModuleList; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractWebModuleList GetWebModules() = 0;
```

See Also

WebModules (see page 5975)

57.1.3.4.17 TAbstractWebContext.GetWebRequest Method

Pascal

```
function GetWebRequest: TWebRequest; virtual; abstract;
```

C++

```
virtual __fastcall TWebRequest GetWebRequest() = 0;
```

See Also

GetWebResponse (see page 5970)

Request (see page 5974)

57.1.3.4.18 TAbstractWebContext.GetWebResponse Method

Pascal

```
function GetWebResponse: TWebResponse; virtual; abstract;
```

C++

```
virtual __fastcall TWebResponse GetWebResponse() = 0;
```

See Also

GetWebRequest (see page 5969)

Response (see page 5975)

57.1.3.4.19 TAbstractWebContext.SetAdapterRequest Method

Pascal

```
procedure SetAdapterRequest(Intf: IUnknown); virtual; abstract;
```

C++

```
virtual __fastcall SetAdapterRequest(IUnknown Intf) = 0;
```

Description

This is SetAdapterRequest, a member of class TAbstractWebContext.

57.1.3.4.20 TAbstractWebContext.SetAdapterResponse Method

Pascal

```
procedure SetAdapterResponse(Intf: IUnknown); virtual; abstract;
```

C++

```
virtual __fastcall SetAdapterResponse(IUnknown Intf) = 0;
```

Description

This is SetAdapterResponse, a member of class TAbstractWebContext.

57.1.3.4.21 TAbstractWebContext.SetDispatchedPageName Method

Pascal

```
procedure SetDispatchedPageName(const APageName: string); virtual; abstract;
```

C++

```
virtual __fastcall SetDispatchedPageName(const UnicodeString APageName) = 0;
```

Description

This is SetDispatchedPageName, a member of class TAbstractWebContext.

57.1.3.4.22 TAbstractWebContext.SetRequestedPageName Method

Pascal

```
procedure SetRequestedPageName(const APageName: string); virtual; abstract;
```

C++

```
virtual __fastcall SetRequestedPageName(const UnicodeString APageName) = 0;
```

Description

This is SetRequestedPageName, a member of class TAbstractWebContext.

57.1.3.5 TAbstractWebContext Properties

The properties of the TAbstractWebContext class are listed here.

Public Properties

	Name	Description
R	AdapterRequest (see page 5971)	Provides access to the adapter request object that manages access to the current request.
R	AdapterResponse (see page 5972)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 5972)	Specifies the name of the Web page module that is producing content to respond to the current request.
R	EndUser (see page 5973)	Provides access to information about the end user that sent the current request.
R	PageCount (see page 5973)	Indicates the number of registered Web pages available in the WebSnap application.
R	Pages (see page 5973)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
R	Request (see page 5974)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 5974)	Indicates the name of the Web page module to which the current request is addressed.
R	Response (see page 5975)	Represents the current HTTP response that the application is assembling.
R	Session (see page 5975)	Provides access to the current Web session.
R	WebModules (see page 5975)	Lists all the Modules in the WebSnap application.

57.1.3.5.1 TAbstractWebContext.AdapterRequest Property

Provides access to the adapter request object that manages access to the current request.

Pascal

```
property AdapterRequest: IUnknown;
```

C++

```
__property IUnknown AdapterRequest;
```

Description

AdapterRequest is the interface of the object that adapters use to obtain information about the current HTTP request message. The adapter request object obtains its information from the TWebRequest object that is the value of the Request (see page 5974) property.

Note: In Delphi, this interface can typically be cast to an IActionRequest interface.

Note: In C++, you can generally obtain an IActionRequest interface from this interface.

See Also

AdapterResponse (see page 5972)

TAbstractAdapterRequest (see page 5955)

Request (see page 5974)

IActionRequest

57.1.3.5.2 TAbstractWebContext.AdapterResponse Property

Provides access to the adapter response object that manages access to the current response.

Pascal

```
property AdapterResponse: IUnknown;
```

C++

```
__property IUnknown AdapterResponse;
```

Description

AdapterResponse is the interface of the object that adapters use to obtain information about the current HTTP response message. The adapter response object obtains its information from the TWebResponse object that is the value of the Response (see page 5975) property.

Note: In Delphi, this interface can typically be cast to an IActionResponse interface.

Note: In C++, you can generally obtain an IActionResponse interface from this interface.

See Also

AdapterRequest (see page 5971)

TAbstractAdapterResponse (see page 5958)

Response (see page 5975)

IActionResponse

57.1.3.5.3 TAbstractWebContext.DispatchedPageName Property

Specifies the name of the Web page module that is producing content to respond to the current request.

Pascal

```
property DispatchedPageName: string;
```

C++

```
__property UnicodeString DispatchedPageName;
```

Description

Read (see page 3634) DispatchedPageName to determine the Web page module that is handling the current HTTP request message. Although this is typically the same as the page that the RequestedPageName (see page 5974) property specifies, it can differ if the request was redirected to another Web page or if its use was delayed because the caller had not yet logged in.

See Also

RequestedPageName (see page 5974)

FindPageInfo (see page 5966)

PageName

57.1.3.5.4 TAbstractWebContext.EndUser Property

Provides access to information about the end user that sent the current request.

Pascal

```
property EndUser: IWebEndUser;
```

C++

```
__property IWebEndUser EndUser;
```

Description

Use EndUser to obtain information about the caller that sent the current request. You can use the IWebEndUser (see page 5988) interface to determine whether the end user has logged in, whether the user has access rights to perform specified operations on pages that have a specified access string, the display name for the end user, and so on.

EndUser is nil (Delphi) or NULL (C++) if the WebSnap application does not include an end user adapter such as TEndUserAdapter or TEndUserSessionAdapter.

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

57.1.3.5.5 TAbstractWebContext.PageCount Property

Indicates the number of registered Web pages available in the WebSnap application.

Pascal

```
property PageCount: Integer;
```

C++

```
__property int PageCount;
```

Description

PageCount indicates the number of entries in the Pages (see page 5973) property array. Use this value as an upper bound for the index when iterating over all the pages in the WebSnap application.

See Also

Pages (see page 5973)

57.1.3.5.6 TAbstractWebContext.Pages Property

Lists the page information interfaces of the registered Web pages available in the WebSnap application.

Pascal

```
property Pages [I: Integer]: IWebPageInfo;
```

C++

```
__property IWebPageInfo Pages[int I];
```

Description

Use Pages to obtain an IWebPageInfo (see page 5996) interface for a registered Web page module. The IWebPageInfo (see page 5996) interface describes the characteristics of a Web page, including its name, title, HREF, description, an indication of whether users must log in to access the page, an indication of whether the current user has access rights to view the page,

and so on.

Pages provides indexed access to page information interfaces. To access the information about a Web page given its name, use the FindPageInfo (see page 5966) method instead. To access the information given the class of the Web page module, use the FindDefaultPageInfo (see page 5964) method instead. To access the page module itself (rather than the page information interface), use the WebModules (see page 5975) property instead.

See Also

PageCount (see page 5973)

FindPageInfo (see page 5966)

WebModules (see page 5975)

FindDefaultPageInfo (see page 5964)

57.1.3.5.7 TAbstractWebContext.Request Property

Represents the current HTTP request that the application is handling.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3634) Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 5975)

57.1.3.5.8 TAbstractWebContext.RequestedPageName Property

Indicates the name of the Web page module to which the current request is addressed.

Pascal

```
property RequestedPageName: string;
```

C++

```
__property UnicodeString RequestedPageName;
```

Description

When the page dispatcher receives an HTTP request message, it sets RequestedPageName to the name of the target Web page module. This is taken from the pathinfo portion of the request message's target URL.

Note: There is no guarantee that RequestedPageName refers to a registered Web page module, because this property is set before the dispatcher verifies that the requested page exists.

See Also

DispatchedPageName (see page 5972)

FindPageInfo (see page 5966)

Request (see page 5974)

TPageDispatcher (see page 6236)

57.1.3.5.9 TAbstractWebContext.Response Property

Represents the current HTTP response that the application is assembling.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3634) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 5974) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 5974)

57.1.3.5.10 TAbstractWebContext.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Use Session to get or set information about the current session. Sessions store information about an end user that is needed for a short period of time. This can be used for such tasks as login/logout support, recording user preferences, and so on.

Note: If you want to use a session to store end user information, the WebSnap application should include a TSessionsService component.

See Also

TSessionsService (see page 6756)

57.1.3.5.11 TAbstractWebContext.WebModules Property

Lists all the Modules in the WebSnap application.

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

Use WebModules to access one of the Web modules that make up the WebSnap application. WebModules lists all of the currently instantiated Web modules. It also provides access to the factory objects for all Web modules in the application, including those that are not yet instantiated.

See Also

Pages (see page 5973)

FindModuleClass (see page 5965)

FindModuleName (see page 5965)

FindApplicationModule (see page 5964)

57.1.4 TWebContext Class

TWebContext represents the current state of a WebSnap application.

Class Hierarchy

Pascal

```
TWebContext = class(TAbstractWebContext);
```

C++

```
class TWebContext : public TAbstractWebContext;
```

File

WebCntxt

Description

A Web context stores information about the components that are available in a WebSnap application and about the current request that is being processed. Applications do not create instances of the Web context. Rather, they access the current Web context object by calling the global WebContext (see page 6003) function.

TWebContext provides the implementation of the abstract or (in C++ terminology) pure virtual methods introduced by its immediate ancestor, TAbstractWebContext (see page 5960).

See Also

WebContext (see page 6003)

Dispatching Requests and Responses

57.1.4.1 TWebContext Members

The following tables list the members exposed by TWebContext.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5983)	This is GetAdapterRequest, a member of class TWebContext.
	GetAdapterResponse (see page 5983)	This is GetAdapterResponse, a member of class TWebContext.
	GetDispatchedPageName (see page 5984)	This is GetDispatchedPageName, a member of class TWebContext.
	GetEndUser (see page 5984)	This is GetEndUser, a member of class TWebContext.
	GetPageCount (see page 5984)	
	GetPageInfo (see page 5984)	
	GetRequestedPageName (see page 5985)	

	GetSession (see page 5985)	
	GetWebModules (see page 5985)	
	GetWebRequest (see page 5985)	
	GetWebResponse (see page 5986)	
	SetAdapterRequest (see page 5986)	This is SetAdapterRequest, a member of class TWebContext.
	SetAdapterResponse (see page 5986)	This is SetAdapterResponse, a member of class TWebContext.
	SetDispatchedPageName (see page 5986)	This is SetDispatchedPageName, a member of class TWebContext.
	SetRequestedPageName (see page 5986)	This is SetRequestedPageName, a member of class TWebContext.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 5978)	Creates an initializes an instance of TWebContext (see page 5976).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 5979)	Frees an instance of TWebContext (see page 5976).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindApplicationModule (see page 5980)	Returns the application module instance.
	FindDefaultPageInfo (see page 5980)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 5981)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 5981)	Returns an instance of a Web module, given its class name.
	FindPage (see page 5982)	Retrieves a Web page module given its name.
	FindPageInfo (see page 5982)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 5983)	Returns an instance of a Web data module, given its class name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AdapterRequest (see page 5971)	Provides access to the adapter request object that manages access to the current request.
	AdapterResponse (see page 5972)	Provides access to the adapter response object that manages access to the current response.
	DispatchedPageName (see page 5972)	Specifies the name of the Web page module that is producing content to respond to the current request.
	EndUser (see page 5973)	Provides access to information about the end user that sent the current request.
	PageCount (see page 5973)	Indicates the number of registered Web pages available in the WebSnap application.
	Pages (see page 5973)	Lists the page information interfaces of the registered Web pages available in the WebSnap application.
	Request (see page 5974)	Represents the current HTTP request that the application is handling.
	RequestedPageName (see page 5974)	Indicates the name of the Web page module to which the current request is addressed.
	Response (see page 5975)	Represents the current HTTP response that the application is assembling.
	Session (see page 5975)	Provides access to the current Web session.
	WebModules (see page 5975)	Lists all the Modules in the WebSnap application.

57.1.4.2 TWebContext.Create Constructor

Creates an initializes an instance of TWebContext (see page 5976).

Pascal

```
constructor Create(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest; AResponse:
TWebResponse; ASession: TAbstractWebSession);
```

C++

```
__fastcall TWebContext(TAbstractWebModuleList AWebModuleList, TWebRequest ARequest,
TWebResponse AResponse, TAbstractWebSession ASession);
```

Description

Do not create instances of TWebContext (see page 5976). WebSnap applications create a global Web context automatically to represent the current state of the application. To access this global Web context object, use the global WebContext (see page 6003) function.

The TWebContext (see page 5976) constructor registers itself with the global WebContext (see page 6003) function so that the application can access the Web context using that function, and initializes the properties of the Web context to the values passed in as parameters.

AWebModuleList is the component that manages the Web modules in the application. It becomes the value of the WebModules (see page 5975) property.

ARequest is the current HTTP request message. It becomes the value of the Request (see page 5974) property.

AResponse is the current response object that the application is filling out. It becomes the value of the Response (see page 5975) property.

ASession is the Session (see page 5975) object associated with ARequest. It becomes the value of the Session (see page 5975) property.

See Also

WebContext (see page 6003)

WebModules (see page 5975)

Request (see page 5974)

Response (see page 5975)

Session (see page 5975)

57.1.4.3 TWebContext.Destroy Destructor

Frees an instance of TWebContext (see page 5976).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContext();
```

Description

Do not free the global Web context object. This object is automatically maintained by the WebSnap application.

The TWebContext (see page 5976) destructor unregisters itself with the global WebContext (see page 6003) function so that the WebContext (see page 6003) function never returns a Web context that no longer exists.

See Also

WebContext (see page 6003)

57.1.4.4 TWebContext Methods

The methods of the TWebContext class are listed here.

Protected Methods

	Name	Description
	GetAdapterRequest (see page 5983)	This is GetAdapterRequest, a member of class TWebContext.
	GetAdapterResponse (see page 5983)	This is GetAdapterResponse, a member of class TWebContext.
	GetDispatchedPageName (see page 5984)	This is GetDispatchedPageName, a member of class TWebContext.
	GetEndUser (see page 5984)	This is GetEndUser, a member of class TWebContext.
	GetPageCount (see page 5984)	
	GetPageInfo (see page 5984)	
	GetRequestedPageName (see page 5985)	
	GetSession (see page 5985)	
	GetWebModules (see page 5985)	
	GetWebRequest (see page 5985)	
	GetWebResponse (see page 5986)	
	SetAdapterRequest (see page 5986)	This is SetAdapterRequest, a member of class TWebContext.
	SetAdapterResponse (see page 5986)	This is SetAdapterResponse, a member of class TWebContext.
	SetDispatchedPageName (see page 5986)	This is SetDispatchedPageName, a member of class TWebContext.
	SetRequestedPageName (see page 5986)	This is SetRequestedPageName, a member of class TWebContext.

Public Methods

	Name	Description
	FindApplicationModule (see page 5980)	Returns the application module instance.
	FindDefaultPageInfo (see page 5980)	Retrieves the Web page information object for a specified class of Web module.
	FindModuleClass (see page 5981)	Returns an instance of a Web module, given its class reference.
	FindModuleName (see page 5981)	Returns an instance of a Web module, given its class name.
	FindPage (see page 5982)	Retrieves a Web page module given its name.

	FindPageInfo (see page 5982)	Retrieves a Web page information object given the name of the corresponding Web page module.
	FindVariableContainer (see page 5983)	Returns an instance of a Web data module, given its class name.

57.1.4.4.1 TWebContext.FindApplicationModule Method

Returns the application module instance.

Pascal

```
function FindApplicationModule(WebModuleContext: TWebModuleContext): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindApplicationModule(TWebModuleContext WebModuleContext);
```

Description

Call FindApplicationModule to access the main Web module in the WebSnap application. FindApplicationModule returns a reference to the module that the WebSnap application wizard creates. This can be an instance of TWebAppPageModule or TWebAppDataModule.

The Application module is the module that holds the application-wide components such as the page dispatcher, end user adapter, application adapter, and so on.

WebModuleContext is a context object that holds design-time information about the application modules. It is only used at design time.

See Also

WebModules (see page 5975)

TWebAppPageModule (see page 6588)

TWebAppDataModule (see page 6585)

57.1.4.4.2 TWebContext.FindDefaultPageInfo Method

Retrieves the Web page information object for a specified class of Web module.

Pascal

```
function FindDefaultPageInfo(AComponentClass: TComponentClass; out APageInfo:
TAbstractWebPageInfo): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindDefaultPageInfo(TComponentClass AComponentClass,
TAbstractWebPageInfo APageInfo);
```

Description

Call FindDefaultPageInfo to obtain the Web page information object for a Web module. FindDefaultPageInfo retrieves the information object that is registered for the Web page module class specified by AComponentClass. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

AComponentClass is a class reference to the class that implements the Web page module. This class is declared in code that the Web page module wizard generates.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindDefaultPageInfo returns false if it can't return a value for the APageInfo parameter. FindDefaultPageInfo returns true if it

locates a registered Web page module for the class specified by AComponentClass.

See Also

FindPageInfo (see page 5982)

57.1.4.4.3 TWebContext.FindModuleClass Method

Returns an instance of a Web module, given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass);
```

Description

Call FindModuleClass to obtain a reference to a Web module given its class reference. If an instance of the specified class already exists, FindModuleClass returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass creates an instance of the specified class and returns the new instance.

AClass is the class reference to a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleName (see page 5981)

FindPage (see page 5982)

57.1.4.4.4 TWebContext.FindModuleName Method

Returns an instance of a Web module, given its class name.

Pascal

```
function FindModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const UnicodeString AName);
```

Description

Call FindModuleName to obtain a reference to a Web module given the name of its implementation class. If an instance of the specified class already exists, FindModuleName returns that instance. Otherwise, if the specified class represents a registered Web module class, FindModuleClass (see page 5981) creates an instance of the specified class and returns the new instance.

AClass is the name of a registered Web module class. This class is declared in code that the Web page module wizard or Web data module wizard generates.

See Also

FindModuleClass (see page 5981)

FindPage (see page 5982)

FindVariableContainer (see page 5983)

57.1.4.4.5 TWebContext.FindPage Method

Retrieves a Web page module given its name.

Pascal

```
function FindPage(const APage: string; AOptions: TFindPageOptions; out AComponent: TObject):
Boolean; override;
```

C++

```
virtual __fastcall Boolean FindPage(const UnicodeString APage, TFindPageOptions AOptions,
TObject * AComponent);
```

Description

Call FindPage to obtain an instance of a registered Web page module given its name. If an instance of the specified Web page module already exists, FindPage returns that instance. Otherwise, if the named Web page module is registered, FindPage creates an instance of the module and returns the new instance.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPage considers. For any option included in this set, FindPage only considers Web page modules that have the corresponding attribute.

AComponent returns the instance of the specified Web page module.

FindPage returns true if it can find or create an instance of the specified Web page module, false otherwise.

See Also

FindModuleName (see page 5981)

FindPageInfo (see page 5982)

FindModuleClass (see page 5981)

57.1.4.4.6 TWebContext.FindPageInfo Method

Retrieves a Web page information object given the name of the corresponding Web page module.

Pascal

```
function FindPageInfo(const APage: string; AOptions: TFindPageOptions; out APageInfo:
TAbstractWebPageInfo): Boolean; override;
```

C++

```
virtual __fastcall Boolean FindPageInfo(const UnicodeString APage, TFindPageOptions AOptions,
TAbstractWebPageInfo APageInfo);
```

Description

Call FindPageInfo to obtain the Web page information object for a Web page module. FindPageInfo retrieves the information object that is registered for the Web page module that has a specified name. The Web page information object describes the characteristics of the Web page, including its name, title, HREF, view access string, an indication of whether users must log in to access the page, and so on.

APage is the name of the desired Web page module. The name is specified using the Web page module wizard, and forms the basis of the Web page module's class name. Thus, a Web page module with a name MyPage has the class name TMyPage.

AOptions is a set of options that can limit the page modules that FindPageInfo considers. For any option included in this set, FindPageInfo only considers Web page modules that have the corresponding attribute.

APageInfo returns the Web page information object that describes the characteristics of the specified Web page module, and that provides access to the factory object that you can use to create instances of the Web page module.

FindPageInfo returns false if it can't return a value for the APageInfo parameter. FindPageInfo returns true if it locates a registered Web page module with the specified name and attributes.

See Also

FindPage (see page 5982)

FindDefaultPageInfo (see page 5980)

Pages (see page 5973)

57.1.4.4.7 TWebContext.FindVariableContainer Method

Returns an instance of a Web data module, given its class name.

Pascal

```
function FindVariableContainer(const AIdent: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindVariableContainer(const UnicodeString AIdent);
```

Description

Call FindVariableContainer to obtain a reference to a Web data module given the name of its implementation class. If an instance of the specified class already exists, FindVariableContainer returns that instance. Otherwise, if the specified class represents a registered Web module class, FindVariableContainer creates an instance of the specified class and returns the new instance.

AIdent is the name of a registered Web data module class. This class is declared in code that the Web data module wizard generates.

See Also

FindModuleName (see page 5981)

57.1.4.4.8 TWebContext.GetAdapterRequest Method

Pascal

```
function GetAdapterRequest: IUnknown; override;
```

C++

```
virtual __fastcall IUnknown GetAdapterRequest();
```

Description

This is GetAdapterRequest, a member of class TWebContext.

57.1.4.4.9 TWebContext.GetAdapterResponse Method

Pascal

```
function GetAdapterResponse: IUnknown; override;
```

C++

```
virtual __fastcall IUnknown GetAdapterResponse();
```

Description

This is GetAdapterResponse, a member of class TWebContext.

57.1.4.4.10 TWebContext.GetDispatchedPageName Method

Pascal

```
function GetDispatchedPageName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDispatchedPageName();
```

Description

This is GetDispatchedPageName, a member of class TWebContext.

57.1.4.4.11 TWebContext.GetEndUser Method

Pascal

```
function GetEndUser: IWebEndUser; override;
```

C++

```
virtual __fastcall IWebEndUser GetEndUser();
```

Description

This is GetEndUser, a member of class TWebContext.

57.1.4.4.12 TWebContext.GetPageCount Method

Pascal

```
function GetPageCount: Integer; override;
```

C++

```
virtual __fastcall int GetPageCount();
```

See Also

GetPageInfo (see page 5968)

PageCount (see page 5973)

57.1.4.4.13 TWebContext.GetPageInfo Method

Pascal

```
function GetPageInfo(I: Integer): IWebPageInfo; override;
```

C++

```
virtual __fastcall IWebPageInfo GetPageInfo(int I);
```

See Also

GetPageCount (see page 5968)

Pages (see page 5973)

57.1.4.4.14 TWebContext.GetRequestedPageName Method

Pascal

```
function GetRequestedPageName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetRequestedPageName();
```

See Also

GetPageCount (see page 5968)

Pages (see page 5973)

57.1.4.4.15 TWebContext.GetSession Method

Pascal

```
function GetSession: TAbstractWebSession; override;
```

C++

```
virtual __fastcall TAbstractWebSession GetSession();
```

See Also

GetPageCount (see page 5968)

Pages (see page 5973)

57.1.4.4.16 TWebContext.GetWebModules Method

Pascal

```
function GetWebModules: TAbstractWebModuleList; override;
```

C++

```
virtual __fastcall TAbstractWebModuleList GetWebModules();
```

See Also

WebModules (see page 5975)

57.1.4.4.17 TWebContext.GetWebRequest Method

Pascal

```
function GetWebRequest: TWebRequest; override;
```

C++

```
virtual __fastcall TWebRequest GetWebRequest();
```

See Also

GetWebResponse (see page 5970)

Request (see page 5974)

57.1.4.4.18 TWebContext.GetWebResponse Method

Pascal

```
function GetWebResponse: TWebResponse; override;
```

C++

```
virtual __fastcall TWebResponse GetWebResponse();
```

See Also

GetWebRequest (see page 5969)

Response (see page 5975)

57.1.4.4.19 TWebContext.SetAdapterRequest Method

Pascal

```
procedure SetAdapterRequest(Intf: IUnknown); override;
```

C++

```
virtual __fastcall SetAdapterRequest(IUnknown Intf);
```

Description

This is SetAdapterRequest, a member of class TWebContext.

57.1.4.4.20 TWebContext.SetAdapterResponse Method

Pascal

```
procedure SetAdapterResponse(Intf: IUnknown); override;
```

C++

```
virtual __fastcall SetAdapterResponse(IUnknown Intf);
```

Description

This is SetAdapterResponse, a member of class TWebContext.

57.1.4.4.21 TWebContext.SetDispatchedPageName Method

Pascal

```
procedure SetDispatchedPageName(const APageName: string); override;
```

C++

```
virtual __fastcall SetDispatchedPageName(const UnicodeString APageName);
```

Description

This is SetDispatchedPageName, a member of class TWebContext.

57.1.4.4.22 TWebContext.SetRequestedPageName Method

Pascal

```
procedure SetRequestedPageName(const APageName: string); override;
```

C++

```
virtual __fastcall SetRequestedPageName(const UnicodeString APageName);
```

Description

This is SetRequestedPageName, a member of class TWebContext.

57.2 Interfaces

The following table lists interfaces in this documentation.

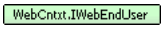
Interfaces

	Name	Description
	IWebEndUser (see page 5988)	IWebEndUser provides access information about the current end user in a WebSnap application.
	IWebPageInfo (see page 5996)	IWebPageInfo describes a Web page in a WebSnap application.

57.2.1 IWebEndUser Interface

IWebEndUser provides access information about the current end user in a WebSnap application.

Class Hierarchy



Pascal

```
IWebEndUser = interface;
```

C++

```
__interface IWebEndUser;
```

File

WebCntxt

Description

Use IWebEndUser to obtain information about the current end user, such as the user name, user ID, and whether the user is logged in. In addition to properties that provide information about the current end user, IWebEndUser has methods to log in an end user.

End user adapter components such as TEndUserAdapter and TEndUserSessionAdapter implement the IWebEndUser interface. If a WebSnap application includes an end user adapter, you can access its IWebEndUser interface by reading the EndUser property of the Web context.

Note: In C++, method declarations that use IWebEndUser use the _di_IWebEndUser type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebEndUser interface:

Note: typedef System::DelphiInterface (see page 3256)< IWebEndUser > _di_IWebEndUser;

See Also

TAbstractWebContext (see page 5960)

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

57.2.1.1 IWebEndUser Members

The following tables list the members exposed by IWebEndUser.

Public Methods

	Name	Description
	GetDisplayName (see page 5989)	Returns the value of the DisplayName (see page 5994) property.
	GetEndUserAdapter (see page 5990)	Returns the value of the EndUserAdapter (see page 5994) property.
	GetLoggedIn (see page 5990)	Returns the value of the LoggedIn (see page 5994) property.
	GetLoginFormAction (see page 5990)	Returns the value of the LoginFormAction (see page 5995) property.
	GetLogoutAction (see page 5991)	Returns the value of the LogoutAction (see page 5996) property.
	HasExecuteRights (see page 5991)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyRights (see page 5992)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewRights (see page 5992)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	LogIn (see page 5993)	Logs in a specified user.
	ProduceLoginPage (see page 5993)	Generates the Web page for logging in users.

Public Properties

	Name	Description
R	DisplayName (see page 5994)	Indicates the name of the end user as it should be used for display in Web pages.
R	EndUserAdapter (see page 5994)	Provides access to the end user adapter that implements this interface.
R	LoggedIn (see page 5994)	Indicates whether the current user is logged in.
R	LoginFormAction (see page 5995)	Indicates the action component for the login form.
R	LogoutAction (see page 5996)	Indicates the action component for logging a user out.

57.2.1.2 IWebEndUser Methods

The methods of the IWebEndUser class are listed here.

Public Methods

	Name	Description
	GetDisplayName (see page 5989)	Returns the value of the DisplayName (see page 5994) property.
	GetEndUserAdapter (see page 5990)	Returns the value of the EndUserAdapter (see page 5994) property.
	GetLoggedIn (see page 5990)	Returns the value of the LoggedIn (see page 5994) property.
	GetLoginFormAction (see page 5990)	Returns the value of the LoginFormAction (see page 5995) property.
	GetLogoutAction (see page 5991)	Returns the value of the LogoutAction (see page 5996) property.
	HasExecuteRights (see page 5991)	Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.
	HasModifyRights (see page 5992)	Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.
	HasViewRights (see page 5992)	Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.
	LogIn (see page 5993)	Logs in a specified user.
	ProduceLoginPage (see page 5993)	Generates the Web page for logging in users.

57.2.1.2.1 IWebEndUser.GetDisplayName Method

Returns the value of the DisplayName (see page 5994) property.

Pascal

```
function GetDisplayName: string;
```

C++

```
__fastcall UnicodeString GetDisplayName();
```

Description

GetDisplayName is the read implementation of the DisplayName (see page 5994) property.

See Also

DisplayName (see page 5994)

57.2.1.2.2 IWebEndUser.GetEndUserAdapter Method

Returns the value of the EndUserAdapter (see page 5994) property.

Pascal

```
function GetEndUserAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetEndUserAdapter();
```

Description

GetEndUserAdapter is the read implementation of the EndUserAdapter (see page 5994) property.

See Also

EndUserAdapter (see page 5994)

57.2.1.2.3 IWebEndUser.GetLoggedIn Method

Returns the value of the LoggedIn (see page 5994) property.

Pascal

```
function GetLoggedIn: Boolean;
```

C++

```
__fastcall Boolean GetLoggedIn();
```

Description

GetLoggedIn is the read implementation of the LoggedIn (see page 5994) property.

See Also

LoggedIn (see page 5994)

57.2.1.2.4 IWebEndUser.GetLoginFormAction Method

Returns the value of the LoginFormAction (see page 5995) property.

Pascal

```
function GetLoginFormAction: TComponent;
```

C++

```
__fastcall TComponent * GetLoginFormAction();
```

Description

GetLoginFormAction is the read implementation of the LoginFormAction (see page 5995) property.

See Also

LoginFormAction (see page 5995)

57.2.1.2.5 IWebEndUser.GetLogoutAction Method

Returns the value of the LogoutAction (see page 5996) property.

Pascal

```
function GetLogoutAction: TComponent;
```

C++

```
__fastcall TComponent * GetLogoutAction();
```

Description

GetLogoutAction is the read implementation of the LogoutAction (see page 5996) property.

See Also

LogoutAction (see page 5996)

57.2.1.2.6 IWebEndUser.HasExecuteRights Method

Indicates whether the current user has access rights to execute Web pages or adapters associated with a specified access string.

Pascal

```
function HasExecuteRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasExecuteRights(UnicodeString ARights);
```

Description

Call HasExecuteRights to determine whether the current end user is authorized to execute Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasExecuteRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasExecuteRights can determine the user's access rights.

See Also

LoggedIn (see page 5994)

HasViewRights (see page 5992)

HasModifyRights (see page 5992)

ViewAccess

ViewAccess (see page 5574)

TWebUserList (see page 6796)

AccessRightsStrings (see page 6791)

57.2.1.2.7 IWebEndUser.HasModifyRights Method

Indicates whether the current user has access rights to modify Web pages or adapters associated with a specified access string.

Pascal

```
function HasModifyRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasModifyRights(UnicodeString ARights);
```

Description

Call HasModifyRights to determine whether the current end user is authorized to modify Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasModifyRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasModifyRights can determine the user's access rights.

See Also

LoggedIn (see page 5994)

HasViewRights (see page 5992)

HasExecuteRights (see page 5991)

ViewAccess

ViewAccess (see page 5574)

TWebUserList (see page 6796)

AccessRightsStrings (see page 6791)

57.2.1.2.8 IWebEndUser.HasViewRights Method

Indicates whether the current user has access rights to view Web pages or adapters associated with a specified access string.

Pascal

```
function HasViewRights(ARights: string): Boolean;
```

C++

```
__fastcall Boolean HasViewRights(UnicodeString ARights);
```

Description

Call HasViewRights to determine whether the current end user is authorized to view Web pages or adapters whose ViewRights property is the value specified by ARights.

Typically, the end user adapter that implements this interface implements HasViewRights by checking the AccessRightsStrings property of the entry for this end user in a TWebUserList component. Note that in this case, the end user must be logged in before HasViewRights can determine the user's access rights.

See Also

LoggedIn (see page 5994)

HasExecuteRights (see page 5991)

HasModifyRights (see page 5992)

ViewAccess

ViewAccess (see page 5574)

TWebUserList (see page 6796)

AccessRightsStrings (see page 6791)

57.2.1.2.9 IWebEndUser.LogIn Method

Logs in a specified user.

Pascal

```
procedure LogIn(UserID: Variant);
```

C++

```
__fastcall LogIn(Variant UserID);
```

Description

Call LogIn to record a user as logged in. Typically, LogIn is called in response to a message from the login page that ProduceLoginPage (see page 5993) generates.

UserID is the identifier of the user that logged in. It is the value of the UserID property of the entry for the user in a WebUserList component, or, in applications that implement their own login scheme, any other Variant (see page 3421) that identifies an end user.

See Also

LoggedIn (see page 5994)

ProduceLoginPage (see page 5993)

TWebUserList (see page 6796)

UserID (see page 6791)

TLoginFormAdapter (see page 5824)

57.2.1.2.10 IWebEndUser.ProduceLoginPage Method

Generates the Web page for logging in users.

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

Call ProduceLoginPage to generate a Web page for logging in users.

ProduceLoginPage returns true if it generates a login form page and updates the current response message so that it will be displayed to the user. ProduceLoginPage returns false if the end user adapter is unable to generate a login page.

See Also

LoggedIn (see page 5994)

LogIn (see page 5993)

LoginFormAction (see page 5995)

57.2.1.3 IWebEndUser Properties

The properties of the IWebEndUser class are listed here.

Public Properties

	Name	Description
	DisplayName (see page 5994)	Indicates the name of the end user as it should be used for display in Web pages.
	EndUserAdapter (see page 5994)	Provides access to the end user adapter that implements this interface.
	LoggedIn (see page 5994)	Indicates whether the current user is logged in.
	LoginFormAction (see page 5995)	Indicates the action component for the login form.
	LogoutAction (see page 5996)	Indicates the action component for logging a user out.

57.2.1.3.1 IWebEndUser.DisplayName Property

Indicates the name of the end user as it should be used for display in Web pages.

Pascal

```
property DisplayName: string;
```

C++

```
__property UnicodeString DisplayName;
```

Description

Read (see page 3634) DisplayName to obtain a string that can be added to Web pages to refer to the current user.

See Also

UserName (see page 6791)

57.2.1.3.2 IWebEndUser.EndUserAdapter Property

Provides access to the end user adapter that implements this interface.

Pascal

```
property EndUserAdapter: TComponent;
```

C++

```
__property TComponent * EndUserAdapter;
```

Description

Read (see page 3634) EndUserAdapter to obtain access to the end user adapter. This is the component that implements this interface. Typically, it is a TEndUserAdapter or TEndUserSessionAdapter instance.

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

57.2.1.3.3 IWebEndUser.LoggedIn Property

Indicates whether the current user is logged in.

Pascal

```
property LoggedIn: Boolean;
```

C++

```
__property Boolean LoggedIn;
```

Description

Read (see page 3634) LoggedIn to determine whether the current user has logged in to the WebSnap application. By logging in, a caller can be identified as one of the users that a TWebUserList component lists.

Once the user has logged in, the HasViewRights (see page 5992), HasExecuteRights (see page 5991), and HasModifyRights (see page 5992) methods can be used to determine whether the user is authorized to view, execute actions, and modify data on restricted Web pages.

To log in a user that is not logged in, call the ProduceLoginPage (see page 5993) method to display a login Web page. That page should allow the user to provide some sort of UserID, which can then be used to log the user in by a call to the LogIn (see page 5993) method.

See Also

LogIn (see page 5993)

ProduceLoginPage (see page 5993)

TWebUserList (see page 6796)

HasExecuteRights (see page 5991)

HasModifyRights (see page 5992)

HasViewRights (see page 5992)

57.2.1.3.4 IWebEndUser.LoginFormAction Property

Indicates the action component for the login form.

Pascal

```
property LoginFormAction: TComponent;
```

C++

```
__property TComponent * LoginFormAction;
```

Description

Read (see page 3634) LoginFormAction to access the login form action component. The login form action component can be called from server-side scripts to display a login form Web page.

Note: To display the login form Web page using the IWebEndUser (see page 5988) interface (rather than from a server-side script), call the ProduceLoginPage (see page 5993) method.

See Also

ProduceLoginPage (see page 5993)

TEndUserAdapterLoginFormAction (see page 5776)

LogoutAction (see page 5996)

57.2.1.3.5 IWebEndUser.LogoutAction Property

Indicates the action component for logging a user out.

Pascal

```
property LogoutAction: TComponent;
```

C++

```
__property TComponent * LogoutAction;
```

Description

Read (see page 3634) LogoutAction to access the logout action component. The logout action component can be called from server-side scripts to perform any necessary tasks when the user logs out.

See Also

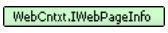
LoginFormAction (see page 5995)

TEndUserAdapterLogoutAction (see page 5784)

57.2.2 IWebPageInfo Interface

IWebPageInfo describes a Web page in a WebSnap application.

Class Hierarchy



Pascal

```
IWebPageInfo = interface;
```

C++

```
__interface IWebPageInfo;
```

File

WebCntxt

Description

Use IWebPageInfo to obtain information about a Web page, including its page name, page title, descriptive text, publish status, and access rights. This information is taken from a TWebPageInfo object that is stored with the Web page module factory.

Note: VCL Method declarations that use IWebPageInfo use the _di_IWebPageInfo type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebPageInfo interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWebPageInfo >  _di_IWebPageInfo;
```

See Also

TWebPageInfo (see page 6301)

TAbstractWebContext (see page 5960)

57.2.2.1 IWebPageInfo Members

The following tables list the members exposed by IWebPageInfo.

Public Methods

	Name	Description
	GetHasViewAccess (see page 5997)	Returns the value of the HasViewAccess (see page 6000) property.
	GetLoginRequired (see page 5998)	Returns the value of the LoginRequired (see page 6000) property.
	GetPageDescription (see page 5998)	Returns the value of the PageDescription (see page 6000) property.
	GetPageHREF (see page 5998)	Returns the value of the PageHREF (see page 6001) property.
	GetPageIsPublished (see page 5999)	Returns the value of the PageIsPublished (see page 6001) property.
	GetPageName (see page 5999)	Returns the value of the PageName (see page 6002) property.
	GetPageTitle (see page 5999)	Returns the value of the PageTitle (see page 6002) property.

Public Properties

	Name	Description
R	HasViewAccess (see page 6000)	Indicates whether the current user has access rights to view the Web page.
R	LoginRequired (see page 6000)	Indicates whether callers must log in before accessing this Web page.
R	PageDescription (see page 6000)	Describes the Web page.
R	PageHREF (see page 6001)	Identifies the HREF to use for references to the associated Web page.
R	PageIsPublished (see page 6001)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 6002).
R	PageName (see page 6002)	Denotes the logical name for the Web page.
R	PageTitle (see page 6002)	Indicates the title that appears on the Web page.

57.2.2.2 IWebPageInfo Methods

The methods of the IWebPageInfo class are listed here.

Public Methods

	Name	Description
	GetHasViewAccess (see page 5997)	Returns the value of the HasViewAccess (see page 6000) property.
	GetLoginRequired (see page 5998)	Returns the value of the LoginRequired (see page 6000) property.
	GetPageDescription (see page 5998)	Returns the value of the PageDescription (see page 6000) property.
	GetPageHREF (see page 5998)	Returns the value of the PageHREF (see page 6001) property.
	GetPageIsPublished (see page 5999)	Returns the value of the PageIsPublished (see page 6001) property.
	GetPageName (see page 5999)	Returns the value of the PageName (see page 6002) property.
	GetPageTitle (see page 5999)	Returns the value of the PageTitle (see page 6002) property.

57.2.2.2.1 IWebPageInfo.GetHasViewAccess Method

Returns the value of the HasViewAccess (see page 6000) property.

Pascal

```
function GetHasViewAccess: Boolean;
```

C++

```
__fastcall Boolean GetHasViewAccess();
```

Description

GetHasViewAccess is the read implementation of the HasViewAccess (see page 6000) property.

See Also

HasViewAccess (see page 6000)

57.2.2.2.2 IWebPageInfo.GetLoginRequired Method

Returns the value of the LoginRequired (see page 6000) property.

Pascal

```
function GetLoginRequired: Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired();
```

Description

GetLoginRequired is the read implementation of the LoginRequired (see page 6000) property.

See Also

LoginRequired (see page 6000)

57.2.2.2.3 IWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription (see page 6000) property.

Pascal

```
function GetPageDescription: string;
```

C++

```
__fastcall UnicodeString GetPageDescription();
```

Description

GetPageDescription is the read implementation of the PageDescription (see page 6000) property.

See Also

PageDescription (see page 6000)

57.2.2.2.4 IWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF (see page 6001) property.

Pascal

```
function GetPageHREF: string;
```

C++

```
__fastcall UnicodeString GetPageHREF();
```

Description

GetPageHREF is the read implementation of the PageHREF (see page 6001) property.

See Also

PageHREF (see page 6001)

57.2.2.2.5 IWebPageInfo.GetPageIsPublished Method

Returns the value of the PageIsPublished (see page 6001) property.

Pascal

```
function GetPageIsPublished: Boolean;
```

C++

```
__fastcall Boolean GetPageIsPublished();
```

Description

GetPageIsPublished is the read implementation of the PageIsPublished (see page 6001) property.

See Also

PageIsPublished (see page 6001)

57.2.2.2.6 IWebPageInfo.GetPageName Method

Returns the value of the PageName (see page 6002) property.

Pascal

```
function GetPageName: string;
```

C++

```
__fastcall UnicodeString GetPageName();
```

Description

GetPageName is the read implementation of the PageName (see page 6002) property.

See Also

PageName (see page 6002)

57.2.2.2.7 IWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle (see page 6002) property.

Pascal

```
function GetPageTitle: string;
```

C++

```
__fastcall UnicodeString GetPageTitle();
```

Description

GetPageTitle is the read implementation of the PageTitle (see page 6002) property.

See Also

PageTitle (see page 6002)

57.2.2.3 IWebPageInfo Properties

The properties of the IWebPageInfo class are listed here.

Public Properties

	Name	Description
	HasViewAccess (see page 6000)	Indicates whether the current user has access rights to view the Web page.
	LoginRequired (see page 6000)	Indicates whether callers must log in before accessing this Web page.
	PageDescription (see page 6000)	Describes the Web page.
	PageHREF (see page 6001)	Identifies the HREF to use for references to the associated Web page.
	PageIsPublished (see page 6001)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 6002).
	PageName (see page 6002)	Denotes the logical name for the Web page.
	PageTitle (see page 6002)	Indicates the title that appears on the Web page.

57.2.2.3.1 IWebPageInfo.HasViewAccess Property

Indicates whether the current user has access rights to view the Web page.

Pascal

```
property HasViewAccess: Boolean;
```

C++

```
__property Boolean HasViewAccess;
```

Description

HasViewAccess is true if the current end user has view rights for the associated Web page, false otherwise.

See Also

HasViewRights (see page 5992)

57.2.2.3.2 IWebPageInfo.LoginRequired Property

Indicates whether callers must log in before accessing this Web page.

Pascal

```
property LoginRequired: Boolean;
```

C++

```
__property Boolean LoginRequired;
```

Description

Read (see page 3634) LoginRequired to determine whether users must log in before they can access this Web page. The application can overrule the LoginRequired property by supplying an OnIsLoginRequired event handler to the page dispatcher.

See Also

OnIsLoginRequired (see page 6199)

TPageDispatcher (see page 6236)

57.2.2.3.3 IWebPageInfo.PageDescription Property

Describes the Web page.

Pascal

```
property PageDescription: string;
```

C++

```
__property UnicodeString PageDescription;
```

Description

Read (see page 3634) PageDescription to obtain a brief description of the Web page.

See Also

PageName (see page 6002)

PageTitle (see page 6002)

57.2.2.3.4 IWebPageInfo.PageHREF Property

Identifies the HREF to use for references to the associated Web page.

Pascal

```
property PageHREF: string;
```

C++

```
__property UnicodeString PageHREF;
```

Description

Read (see page 3634) PageHREF to obtain the HTTP path string that should be inserted in HTML code to create a hypertext link to the associated Web page.

See Also

PageName (see page 6002)

57.2.2.3.5 IWebPageInfo.PageIsPublished Property

Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 6002).

Pascal

```
property PageIsPublished: Boolean;
```

C++

```
__property Boolean PageIsPublished;
```

Description

Read (see page 3634) PageIsPublished to determine whether the page dispatcher (TPageDispatcher) automatically calls on the associated Web page module when it receives HTTP request messages that address this Web page by name. HTTP request messages address a Web page when the pathinfo portion of their target URL matches the value of the PageName (see page 6002) property.

When IsPublished is false, the WebSnap application must explicitly retrieve the content for the Web page and add it to a response message.

See Also

PageName (see page 6002)

TPageDispatcher (see page 6236)

57.2.2.3.6 IWebPageInfo.PageName Property

Denotes the logical name for the Web page.

Pascal

```
property PageName: string;
```

C++

```
__property UnicodeString PageName;
```

Description

Read (see page 3634) PageName to determine the name of the Web page. This name is used to identify the page in server-side scripts, and as the pathinfo portion of the URL for the page when the PageIsPublished (see page 6001) property is true.

See Also

PageTitle (see page 6002)

PageHREF (see page 6001)

PageIsPublished (see page 6001)

57.2.2.3.7 IWebPageInfo.PageTitle Property

Indicates the title that appears on the Web page.

Pascal

```
property PageTitle: string;
```

C++

```
__property UnicodeString PageTitle;
```

Description

Read (see page 3634) PageTitle to determine the title that appears on the Web page when viewed from a user's browser.

See Also

PageName (see page 6002)

57.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WebContext (see page 6003)	Returns the current Web context

57.3.1 WebCntxt.WebContext Function

Returns the current Web context

Pascal

```
function WebContext: TAbstractWebContext;
```

C++

```
TAbstractWebContext WebContext();
```

File

WebCntxt

Description

Use the WebContext function to obtain a reference to the global Web context. This object stores information about the components that are available in the Web application as well as about the current request that is being processed.

57.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TFindPageOption (see page 6004)	TFindPageOptions (see page 6005) indicates the attributes that must be present on a Web page module that is the target of a search.

57.4.1 WebCntxt.TFindPageOption Enumeration

TFindPageOptions (see page 6005) indicates the attributes that must be present on a Web page module that is the target of a search.

Pascal

```
TFindPageOption = (
  fpPublished,
  fpLoginRequired
);
```

C++

```
enum TFindPageOption {
  fpPublished,
  fpLoginRequired
};
```

File

WebCntxt

Description

When searching for Web modules using a given Web context, TFindPageOption values represent attributes of a Web module that must be present before a given Web module is considered. When a TFindPageOption is included in the set of options, only those Web modules that have the corresponding attribute are considered. Note that if an option is not included, Web modules that have the corresponding attribute are still considered in the search operation.

Option	Meaning
fpPublished	Only published Web page modules are considered.
fpLoginRequired	Only Web page modules that require the user to log in before they can be accessed are considered.

Note: These options are added to a Web page module using properties of the Web page information object with which the module is registered. The Web page information object is supplied as a parameter to the corresponding factory object's constructor in code that the Web page module wizard generates.

See Also

TAbstractWebContext (see page 5960)

TWebPageInfo (see page 6301)

57.5 Types

The following table lists types in this documentation.

Types

Name	Description
TFindPageOptions (see page 6005)	This is type WebCntxt.TFindPageOptions.
TGetWebContextProc (see page 6005)	This is type WebCntxt.TGetWebContextProc.
TGetWebEndUserProc (see page 6006)	This is type WebCntxt.TGetWebEndUserProc.
TSetWebContextProc (see page 6006)	This is type WebCntxt.TSetWebContextProc.

57.5.1 WebCntxt.TFindPageOptions Type

Pascal

```
TFindPageOptions = set of TFindPageOption;
```

C++

```
TFindPageOption TFindPageOptions;
```

File

WebCntxt

Description

This is type WebCntxt.TFindPageOptions.

57.5.2 WebCntxt.TGetWebContextProc Type

Pascal

```
TGetWebContextProc = function : TAbstractWebContext;
```

C++

```
: TAbstractWebContext ( TGetWebContextProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TGetWebContextProc.

57.5.3 WebCntxt.TGetWebEndUserProc Type

Pascal

```
TGetWebEndUserProc = function : IWebEndUser;
```

C++

```
: IWebEndUser ( TGetWebEndUserProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TGetWebEndUserProc.

57.5.4 WebCntxt.TSetWebContextProc Type

Pascal

```
TSetWebContextProc = procedure (AWebContext: TAbstractWebContext);
```

C++

```
(AWebContext: TAbstractWebContext) ( TSetWebContextProc)();
```

File

WebCntxt

Description

This is type WebCntxt.TSetWebContextProc.

57.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
GetModuleFileNameProc (see page 6007)	This is variable WebCntxt.GetModuleFileNameProc.
GetWebContextProc (see page 6007)	This is variable WebCntxt.GetWebContextProc.
GetWebEndUserProc (see page 6008)	This is variable WebCntxt.GetWebEndUserProc.
SetWebContextProc (see page 6008)	This is variable WebCntxt.SetWebContextProc.

57.6.1 WebCntxt.GetModuleFileNameProc Variable

Pascal

```
GetModuleFileNameProc: function: string = nil;
```

C++

```
function: string GetModuleFileNameProc = nil;
```

File

WebCntxt

Description

This is variable WebCntxt.GetModuleFileNameProc.

57.6.2 WebCntxt.GetWebContextProc Variable

Pascal

```
GetWebContextProc: TGetWebContextProc;
```

C++

```
TGetWebContextProc GetWebContextProc;
```

File

WebCntxt

Description

This is variable WebCntxt.GetWebContextProc.

57.6.3 WebCntxt.GetWebEndUserProc Variable

Pascal

```
GetWebEndUserProc: TGetWebEndUserProc;
```

C++

```
TGetWebEndUserProc GetWebEndUserProc;
```

File

WebCntxt

Description

This is variable WebCntxt.GetWebEndUserProc.

57.6.4 WebCntxt.SetWebContextProc Variable

Pascal

```
SetWebContextProc: TSetWebContextProc;
```

C++

```
TSetWebContextProc SetWebContextProc;
```

File

WebCntxt

Description

This is variable WebCntxt.SetWebContextProc.

58 WebComp Namespace

This is namespace WebComp.

Topics

Name	Description
Classes (see page 6011)	The following table lists classes in this documentation.
Interfaces (see page 6079)	The following table lists interfaces in this documentation.
Functions (see page 6106)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6109)	The following table lists structs, records, enums in this documentation.
Types (see page 6111)	The following table lists types in this documentation.
Constants (see page 6114)	The following table lists constants in this documentation.

Classes

	Name	Description
	TLayoutAttributes (see page 6011)	TLayoutAttributes describes how an HTML control should be laid out.
	TLayoutWebContent (see page 6014)	TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	TWebComponentClassDescription (see page 6023)	This is class WebComp.TWebComponentClassDescription.
	TWebComponentList (see page 6025)	TWebComponentList maintains a list of Web item components.
	TWebComponentsEditorHelper (see page 6035)	This is class WebComp.TWebComponentsEditorHelper.
	TWebContainedComponent (see page 6038)	TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.
	TWebContainedComponentHelper (see page 6045)	TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.
	TWebContainedContainerComponent (see page 6052)	TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.
	TWebContainerComponent (see page 6063)	TWebContainerComponent is the base class for components that use Web items to generate HTML.
	TWebContainerComponentHelper (see page 6068)	TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.
	TWebContentOptions (see page 6076)	TWebContentOptions controls what a Web item can include in its generated HTML.

Constants

Name	Description
EnumRegisteredWebComponentsProc (see page 6114)	This is constant WebComp.EnumRegisteredWebComponentsProc.
FindWebComponentEditorHelperProc (see page 6114)	This is constant WebComp.FindWebComponentEditorHelperProc.

RegisterWebComponentsProc (see page 6115)	This is constant WebComp.RegisterWebComponentsProc.
UnRegisterWebComponentsProc (see page 6115)	This is constant WebComp.UnRegisterWebComponentsProc.

Functions

	Name	Description
	DesigningComponent (see page 6106)	This is function WebComp.DesigningComponent.
	EnumRegisteredWebComponents (see page 6106)	This is function WebComp.EnumRegisteredWebComponents.
	FindWebComponentEditorHelper (see page 6107)	This is function WebComp.FindWebComponentEditorHelper.
	QualifyFileName (see page 6107)	This is function WebComp.QualifyFileName.
	RegisterWebComponents (see page 6107)	This is function WebComp.RegisterWebComponents.
	UnRegisterWebComponents (see page 6108)	This is function WebComp.UnRegisterWebComponents.
	ValidateWebParent (see page 6108)	This is function WebComp.ValidateWebParent.

Interfaces

	Name	Description
	IAddFieldsEditor (see page 6079)	This is class WebComp.IAddFieldsEditor.
	IGetWebComponentList (see page 6081)	This is class WebComp.IGetWebComponentList.
	IGetWebComponentNodeName (see page 6082)	This is class WebComp.IGetWebComponentNodeName.
	IGetXMLStream (see page 6083)	This is class WebComp.IGetXMLStream.
	ILayoutWebContent (see page 6084)	ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	INotifyActivateModule (see page 6087)	This is class WebComp.INotifyActivateModule.
	INotifyWebActivate (see page 6088)	This is class WebComp.INotifyWebActivate.
	IProducerEditorViewSupport (see page 6089)	This is class WebComp.IProducerEditorViewSupport.
	ISetWebContentOptions (see page 6091)	This is class WebComp.ISetWebContentOptions.
	ITopLevelWebComponent (see page 6092)	This is class WebComp.ITopLevelWebComponent.
	IWebComponent (see page 6093)	IWebComponent is the interface supported by components that TWebComponentList (see page 6025) manages.
	IWebComponentCommands (see page 6097)	This is class WebComp.IWebComponentCommands.
	IWebComponentContainer (see page 6098)	This is class WebComp.IWebComponentContainer.
	IWebComponentEditor (see page 6100)	IWebComponentEditor enables a Web item to be assigned children in the IDE.
	IWebComponentEditorHelper (see page 6102)	This is class WebComp.IWebComponentEditorHelper.
	IWebContent (see page 6103)	IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.
	IWebVariablesContainerName (see page 6104)	This is class WebComp.IWebVariablesContainerName.

Structs, Records, Enums

	Name	Description
	TLabelPosition (see page 6109)	This is record WebComp.TLabelPosition.
	TWebContentFlag (see page 6109)	TWebContentFlags (see page 6113) indicates the type of HTML a Web item can produce.

Types

Name	Description
TEnumWebComponentsProc (see page 6111)	This is type WebComp.TEnumWebComponentsProc.
TGetComponentListEvent (see page 6111)	TGetComponentListEvent is the type for event handlers that supply a list of child Web items.
TLayout (see page 6112)	TLayout is the type for helper objects that generate layout code for a Web item.
TSetComponentListEvent (see page 6112)	This is type WebComp.TSetComponentListEvent.
TWebComponentContainer (see page 6113)	TWebComponentContainer is the type for components that maintain a list of Web items.
TWebContentFlags (see page 6113)	This is type WebComp.TWebContentFlags.

58.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TLayoutAttributes (see page 6011)	TLayoutAttributes describes how an HTML control should be laid out.
	TLayoutWebContent (see page 6014)	TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	TWebComponentClassDescription (see page 6023)	This is class WebComp.TWebComponentClassDescription.
	TWebComponentList (see page 6025)	TWebComponentList maintains a list of Web item components.
	TWebComponentsEditorHelper (see page 6035)	This is class WebComp.TWebComponentsEditorHelper.
	TWebContainedComponent (see page 6038)	TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.
	TWebContainedComponentHelper (see page 6045)	TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.
	TWebContainedContainerComponent (see page 6052)	TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.
	TWebContainerComponent (see page 6063)	TWebContainerComponent is the base class for components that use Web items to generate HTML.
	TWebContainerComponentHelper (see page 6068)	TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.
	TWebContentOptions (see page 6076)	TWebContentOptions controls what a Web item can include in its generated HTML.

58.1.1 TLayoutAttributes Class

TLayoutAttributes describes how an HTML control should be laid out.

Class Hierarchy

Pascal

```
TLayoutAttributes = class(TObject);
```

C++

```
class TLayoutAttributes : public TObject;
```

File

WebComp

Description

TLayoutAttributes embodies a set of flags that describe how an HTML control should be laid out. These flags form the value of several public fields of the object, including

LabelAttributes (see page 6013): a string that describes style attributes that apply to the control's label (if any).

LabelPosition (see page 6014): an enumerated value describing the the position of the label (if any) relative to the control. Possible values are lposLeft (to the left of the control), lposRight (to the right of the control), lposAbove (above the control) and lposBelow (below the control).

ControlAttributes (see page 6013): a string that describes style attributes that apply to the control.

See Also

IWebContent (see page 6103)

58.1.1.1 TLayoutAttributes Members

The following tables list the members exposed by TLayoutAttributes.

Public Fields

	Name	Description
	ControlAttributes (see page 6013)	This is ControlAttributes, a member of class TLayoutAttributes.
	LabelAttributes (see page 6013)	This is LabelAttributes, a member of class TLayoutAttributes.
	LabelPosition (see page 6014)	This is LabelPosition, a member of class TLayoutAttributes.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6013)	Creates a new TLayoutAttributes (see page 6011) object.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

58.1.1.2 TLayoutAttributes.Create Constructor

Creates a new TLayoutAttributes (see page 6011) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TLayoutAttributes();
```

Description

Typically, applications do not instantiate TLayoutAttributes (see page 6011). Instead, each Web item that generates HTML for an InternetExpress application instantiates an internal TLayoutAttributes (see page 6011) object and uses it to encode property values that affect the layout of its generated HTML.

58.1.1.3 TLayoutAttributes Fields

The fields of the TLayoutAttributes class are listed here.

Public Fields

	Name	Description
	ControlAttributes (see page 6013)	This is ControlAttributes, a member of class TLayoutAttributes.
	LabelAttributes (see page 6013)	This is LabelAttributes, a member of class TLayoutAttributes.
	LabelPosition (see page 6014)	This is LabelPosition, a member of class TLayoutAttributes.

58.1.1.3.1 TLayoutAttributes.ControlAttributes Field

Pascal

```
ControlAttributes: string;
```

C++

```
UnicodeString ControlAttributes;
```

Description

This is ControlAttributes, a member of class TLayoutAttributes.

58.1.1.3.2 TLayoutAttributes.LabelAttributes Field

Pascal

```
LabelAttributes: string;
```

C++

```
UnicodeString LabelAttributes;
```

Description

This is LabelAttributes, a member of class TLayoutAttributes.

58.1.1.3.3 TLayoutAttributes.LabelPosition Field

Pascal

```
LabelPosition: TLabelPosition;
```

C++

```
TLabelPosition LabelPosition;
```

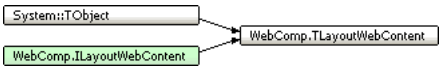
Description

This is LabelPosition, a member of class TLayoutAttributes.

58.1.2 TLayoutWebContent Class

TLayoutWebContent lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy



Pascal

```
TLayoutWebContent = class(TObject, ILayoutWebContent);
```

C++

```
class TLayoutWebContent : public TObject, public ILayoutWebContent;
```

File

WebComp

Description

TLayoutWebContent is the base class for objects that implement the ILayoutWebContent (see page 6084) interface. Parent components pass a TLayoutWebContent descendant to child Web items as the ParentLayout (see page 6022) parameter of their Content method. This object provides helper functions that the child items use to fit their generated content into the parent object's HTML structure (typically a <TABLE> tag). For example, the TLayoutWebContent methods may add <TD> tags around generated HTML that fills the cells of a parent's HTML table.

Do not create instances of TLayoutWebContent. This implementation of the ILayoutWebContent (see page 6084) methods consists of abstract or, in C++ terminology, pure virtual methods. Instead, use a TLayoutWebContent descendant such as TFormLayout or TAdapterFormLayout, which supplies an implementation for those methods.

See Also

TLayout (see page 6112)

IWebContent (see page 6103)

ILayoutWebContent (see page 6084)

TFormLayout

TAdapterFormLayout (see page 6408)

58.1.2.1 TLayoutWebContent Members

The following tables list the members exposed by TLayoutWebContent.

Protected Methods

	Name	Description
	_AddRef (see page 6017)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 6017)	Responds when the caller releases a reference to this object's interface.
A	ImplLayoutButton (see page 6017)	Provides the underlying implementation for the LayoutButton (see page 6020) method.
A	ImplLayoutField (see page 6018)	Provides the underlying implementation for the LayoutField (see page 6020) method.
A	ImplLayoutLabelAndField (see page 6018)	Provides the underlying implementation for the LayoutLabelAndField (see page 6020) method.
A	ImplLayoutTable (see page 6019)	Provides the underlying implementation for the LayoutTable (see page 6021) method.
	LayoutButton (see page 6020)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 6020)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 6020)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 6021)	Converts the HTML declaring a table into a form the associated Web item can use.
V	QueryInterface (see page 6021)	Returns a reference to a specified interface if the object supports that interface.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6016)	Creates a new TLayoutWebContent (see page 6014) object.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.

	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ParentLayout (see page 6022)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

58.1.2.2 TLayoutWebContent.Create Constructor

Creates a new TLayoutWebContent (see page 6014) object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TLayoutWebContent(TLayout AParentLayout);
```

Description

Call Create to instantiate a TLayoutWebContent (see page 6014) descendant to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout (see page 6022) property.

See Also

ParentLayout (see page 6022)

58.1.2.3 TLayoutWebContent Methods

The methods of the TLayoutWebContent class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6017)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 6017)	Responds when the caller releases a reference to this object's interface.
A	ImplLayoutButton (see page 6017)	Provides the underlying implementation for the LayoutButton (see page 6020) method.
A	ImplLayoutField (see page 6018)	Provides the underlying implementation for the LayoutField (see page 6020) method.
A	ImplLayoutLabelAndField (see page 6018)	Provides the underlying implementation for the LayoutLabelAndField (see page 6020) method.
A	ImplLayoutTable (see page 6019)	Provides the underlying implementation for the LayoutTable (see page 6021) method.
	LayoutButton (see page 6020)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 6020)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 6020)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 6021)	Converts the HTML declaring a table into a form the associated Web item can use.

	QueryInterface (see page 6021)	Returns a reference to a specified interface if the object supports that interface.

58.1.2.3.1 TLayoutWebContent._AddRef Method

Responds when the caller acquires a reference to this object's interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

In Delphi, when an application uses an interface that is implemented by TLayoutWebContent (see page 6014) or one of its descendants, it automatically calls __AddRef. Typically, interfaced objects use the __AddRef method to increment a reference count, which is then decremented when the application releases the interface. In this typical scenario, when the reference count drops to zero, the object frees itself.

TLayoutWebContent (see page 6014) implements its own version of the interface methods __AddRef and _Release (see page 6017) that avoid this typical behavior. This prevents the object from being freed when its reference count drops to zero. In TLayoutWebContent (see page 6014), __AddRef always returns –1.

See Also

_Release (see page 6017)

_AddRef (see page 3503)

58.1.2.3.2 TLayoutWebContent._Release Method

Responds when the caller releases a reference to this object's interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

In Delphi, when an application releases an interface that is implemented by TLayoutWebContent (see page 6014) or one of its descendants, it automatically calls __Release. Typically, interfaced objects use the __Release method to decrement a reference count. When the reference count drops to zero, the object is automatically freed.

TLayoutWebContent (see page 6014) implements its own version of the interface methods _AddRef (see page 6017) and __Release that avoid this typical behavior. This prevents the object from being freed when its reference count drops to zero. In TLayoutWebContent (see page 6014), __Release always returns –1.

See Also

_AddRef (see page 6017)

_Release (see page 3503)

58.1.2.3.3 TLayoutWebContent.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton (see page 6020) method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutButton(const UnicodeString HTMLButton,
TLayoutAttributes Attributes) = 0;
```

Description

The LayoutButton (see page 6020) method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton (see page 6020) returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent (see page 6014), ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton (see page 6020) method.

See Also

LayoutButton (see page 6020)

58.1.2.3.4 TLayoutWebContent.ImplLayoutField Method

Provides the underlying implementation for the LayoutField (see page 6020) method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutField(const UnicodeString HTMLField,
TLayoutAttributes Attributes) = 0;
```

Description

The LayoutField (see page 6020) method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField (see page 6020) returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent (see page 6014), ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField (see page 6020) method.

See Also

LayoutField (see page 6020)

58.1.2.3.5 TLayoutWebContent.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField (see page 6020) method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes) = 0;
```

Description

The LayoutLabelAndField (see page 6020) method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField (see page 6020) returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent (see page 6014), ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField (see page 6020) method.

See Also

LayoutLabelAndField (see page 6020)

58.1.2.3.6 TLayoutWebContent.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable (see page 6021) method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutTable(const UnicodeString HTMLTable,
TLayoutAttributes Attributes) = 0;
```

Description

The LayoutTable (see page 6021) method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable (see page 6021) returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent (see page 6014), ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable (see page 6021) method.

See Also

LayoutTable (see page 6021)

58.1.2.3.7 TLayoutWebContent.LayoutButton Method

Converts the HTML declaring a button into a form the associated Web item can use.

Pascal

```
function LayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutButton(const UnicodeString HTMLButton, TLayoutAttributes
Attributes);
```

Description

Call LayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

LayoutButton delegates the task of converting the HTML for the button to the ImplLayoutButton (see page 6017) method.

See Also

ImplLayoutButton (see page 6017)

58.1.2.3.8 TLayoutWebContent.LayoutField Method

Converts the HTML declaring a field into a form the associated Web item can use.

Pascal

```
function LayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutField(const UnicodeString HTMLField, TLayoutAttributes
Attributes);
```

Description

Call LayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

LayoutField delegates the task of converting the HTML for the field to the ImplLayoutField (see page 6018) method.

See Also

ImplLayoutField (see page 6018)

58.1.2.3.9 TLayoutWebContent.LayoutLabelAndField Method

Converts the HTML declaring a labeled field into a form the associated Web item can use.

Pascal

```
function LayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes);
```

Description

Call LayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

LayoutLabelAndField delegates the task of converting the HTML for the label and field to the ImplLayoutLabelAndField (see page 6018) method.

See Also

ImplLayoutLabelAndField (see page 6018)

58.1.2.3.10 TLayoutWebContent.LayoutTable Method

Converts the HTML declaring a table into a form the associated Web item can use.

Pascal

```
function LayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutTable(const UnicodeString HTMLTable, TLayoutAttributes
Attributes);
```

Description

Call LayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable returns the HTML for the table in a form that the associated Web item can use.

LayoutTable delegates the task of converting the HTML for the table to the ImplLayoutTable (see page 6019) method.

See Also

ImplLayoutTable (see page 6019)

58.1.2.3.11 TLayoutWebContent.QueryInterface Method

Returns a reference to a specified interface if the object supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the TLayoutWebContent (see page 6014) object supports the interface specified by IID. If so, QueryInterface sets the Obj parameter so that it points to an instance of the specified interface and returns 0 to indicate success.

If the layout object does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

IID is the desired interface.

Obj returns a reference to the interface if the TLayoutWebContent (see page 6014) object supports it.

See Also

_AddRef (see page 6017)

_Release (see page 6017)

58.1.2.4 TLayoutWebContent Properties

The properties of the TLayoutWebContent class are listed here.

Public Properties

	Name	Description
	ParentLayout (see page 6022)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

58.1.2.4.1 TLayoutWebContent.ParentLayout Property

Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.

Pascal

```
property ParentLayout: TLayout;
```

C++

```
__property TLayout ParentLayout;
```

Description

Web items form a hierarchy, where each child Web item generates its content on behalf of a parent object. This TLayoutWebContent (see page 6014) object is associated with a Web item, and supplies methods to help lay out the content of its child Web items. The associated Web item, in turn, may be the child Web item of another Web item. ParentLayout is the layout object of the associated Web item's parent item.

The TLayoutWebContent (see page 6014) object uses ParentLayout in the implementation of its methods so that they reflect the needs of the parent item as well as the needs of the associated Web item.

ParentLayout supports the ILayoutWebContent (see page 6084) interface. Typically, it is another TLayoutWebContent (see page 6014) descendant.

See Also

Create (see page 6016)

ILayoutWebContent (see page 6084)

58.1.3 TWebComponentClassDescription Class

Class Hierarchy

Pascal

```
TWebComponentClassDescription = class(TObject);
```

C++

```
class TWebComponentClassDescription : public TObject;
```

File

WebComp

Description

This is class WebComp.TWebComponentClassDescription.

58.1.3.1 TWebComponentClassDescription Members

The following tables list the members exposed by TWebComponentClassDescription.

Public Fields

	Name	Description
	ComponentClass (see page 6024)	This is ComponentClass, a member of class TWebComponentClassDescription.
	ComponentClassName (see page 6024)	This is ComponentClassName, a member of class TWebComponentClassDescription.
	Editor (see page 6025)	This is Editor, a member of class TWebComponentClassDescription.
	FrameClass (see page 6025)	This is FrameClass, a member of class TWebComponentClassDescription.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.

	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

58.1.3.2 TWebComponentClassDescription Fields

The fields of the TWebComponentClassDescription class are listed here.

Public Fields

	Name	Description
	ComponentClass (see page 6024)	This is ComponentClass, a member of class TWebComponentClassDescription.
	ComponentClassName (see page 6024)	This is ComponentClassName, a member of class TWebComponentClassDescription.
	Editor (see page 6025)	This is Editor, a member of class TWebComponentClassDescription.
	FrameClass (see page 6025)	This is FrameClass, a member of class TWebComponentClassDescription.

58.1.3.2.1 TWebComponentClassDescription.ComponentClass Field

Pascal

```
ComponentClass: TComponentClass;
```

C++

```
TComponentClass ComponentClass;
```

Description

This is ComponentClass, a member of class TWebComponentClassDescription.

58.1.3.2.2 TWebComponentClassDescription.ComponentClassName Field

Pascal

```
ComponentClassName: string;
```

C++

```
UnicodeString ComponentClassName;
```

Description

This is ComponentClassName, a member of class TWebComponentClassDescription.

58.1.3.2.3 TWebComponentClassDescription.Editor Field

Pascal

```
Editor: TObject;
```

C++

```
TObject * Editor;
```

Description

This is Editor, a member of class TWebComponentClassDescription.

58.1.3.2.4 TWebComponentClassDescription.FrameClass Field

Pascal

```
FrameClass: Boolean;
```

C++

```
Boolean FrameClass;
```

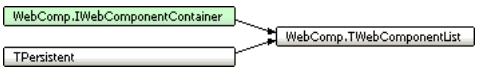
Description

This is FrameClass, a member of class TWebComponentClassDescription.

58.1.4 TWebComponentList Class

TWebComponentList maintains a list of Web item components.

Class Hierarchy



Pascal

```
TWebComponentList = class(TPersistent, IWebComponentContainer);
```

C++

```
class TWebComponentList : public TPersistent, public IWebComponentContainer;
```

File

WebComp

Description

TWebComponentList provides the structure for the hierarchy of Web items that generate HTML documents in a WebSnap or InternetExpress application. At the top of the hierarchy is a page producer (TINetXPageProducer or TAdapterPageProducer) or an adapter. The page producer assembles an HTML document, or the adapter provides information and performs tasks on behalf of an adapter page producer or server-side script. To perform their tasks, the root objects use a set of helper components. Each of these components, in turn, uses one or more helper components, and so on. Each component that uses helper components maintains a list of its helper components. That list is a TWebComponentList object.

Each component in the list must support the IWebComponent (see page 6093) interface. TWebComponentList uses this interface to coordinate the objects it manages. In addition, children of a page producer should support the IWebContent (see page 6103) interface. IWebContent (see page 6103) allows the parent component to extract HTML generated by its children.

Use the properties and methods of TWebComponentList to

Add (see page 6029) or delete the components in the list.

Locate and access components in the list.

Rearrange components in the list.

Access the parent component that uses the list.

See Also

IWebContent (see page 6103)

TBasePageItemsProducer

TWebControlGroup

TWebForm

TInetXPageProducer

IWebComponent (see page 6093)

TAdapterPageProducer (see page 1136)

58.1.4.1 TWebComponentList Members

The following tables list the members exposed by TWebComponentList.

Protected Methods

	Name	Description
	_AddRef (see page 6028)	This is _AddRef, a member of class TWebComponentList.
	_Release (see page 6028)	This is _Release, a member of class TWebComponentList.
	Changed (see page 6029)	This is Changed, a member of class TWebComponentList.
	GetComponent (see page 6030)	This is GetComponent, a member of class TWebComponentList.
	GetCount (see page 6030)	This is GetCount, a member of class TWebComponentList.
V	GetOwner (see page 6030)	Returns the owner of an object.
V	NewComponent (see page 6031)	This is NewComponent, a member of class TWebComponentList.
V	QueryInterface (see page 6032)	This is QueryInterface, a member of class TWebComponentList.
	WebComponentContainer_GetComponent (see page 6033)	This is WebComponentContainer_GetComponent, a member of class TWebComponentList.
	WebComponentContainer_GetComponentCount (see page 6033)	This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

Public Events

	Name	Description
	OnChange (see page 6035)	Occurs when a change occurs in the web component list.

Public Methods

	Name	Description
	Add (see page 6029)	Inserts a new component at the end of the list.
	Clear (see page 6029)	Deletes all components from the list.
	Create (see page 6027)	Creates and initializes new TWebComponentList (see page 6025) object.
V	Destroy (see page 6027)	Frees an instance of TWebComponentList (see page 6025).
	GetComponentCount (see page 6099)	This is GetComponentCount, a member of class IWebComponentContainer.
	IndexOf (see page 6030)	Returns the index of a specified component in the WebComponents (see page 6034) array.
	LockNotify (see page 6031)	This is LockNotify, a member of class TWebComponentList.

	NotifyLocked (see page 6031)	This is NotifyLocked, a member of class TWebComponentList.
	Remove (see page 6032)	Removes a component from the WebComponents (see page 6034) array.
	SetComponentIndex (see page 6032)	Moves a component to a new position in the WebComponents (see page 6034) array.
	UnlockNotify (see page 6033)	This is UnlockNotify, a member of class TWebComponentList.

Public Properties

	Name	Description
R	ComponentCount (see page 6100)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 6100)	This is Components, a member of class IWebComponentContainer.
R	Count (see page 6033)	Indicates the number of components in the list.
R	ParentComponent (see page 6034)	Identifies the component that uses this Web component list to maintain its Web Items.
R	WebComponents (see page 6034)	Lists the Web items in the list.

58.1.4.2 TWebComponentList.Create Constructor

Creates and initializes new TWebComponentList (see page 6025) object.

Pascal

```
constructor Create(AParentComponent: TComponent);
```

C++

```
__fastcall TWebComponentList(TComponent * AParentComponent);
```

Description

Call Create to instantiate TWebComponentList (see page 6025) at runtime. Most applications do not need to instantiate TWebComponentList (see page 6025). Instead, the Web items that use TWebComponentList (see page 6025) to manage their own Web items instantiate a TWebComponentList (see page 6025) from their constructors.

58.1.4.3 TWebComponentList.Destroy Destructor

Frees an instance of TWebComponentList (see page 6025).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebComponentList();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TWebComponentList (see page 6025) reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the list of WebComponents (see page 6034).

Note: Destroy does not free the components in the list.

See Also

System::TObject::Free (see page 3372)

58.1.4.4 TWebComponentList Methods

The methods of the TWebComponentList class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6028)	This is _AddRef, a member of class TWebComponentList.
	_Release (see page 6028)	This is _Release, a member of class TWebComponentList.
	Changed (see page 6029)	This is Changed, a member of class TWebComponentList.
	GetComponent (see page 6030)	This is GetComponent, a member of class TWebComponentList.
	GetCount (see page 6030)	This is GetCount, a member of class TWebComponentList.
V	GetOwner (see page 6030)	Returns the owner of an object.
V	NewComponent (see page 6031)	This is NewComponent, a member of class TWebComponentList.
V	QueryInterface (see page 6032)	This is QueryInterface, a member of class TWebComponentList.
	WebComponentContainer_GetComponent (see page 6033)	This is WebComponentContainer_GetComponent, a member of class TWebComponentList.
	WebComponentContainer_GetComponentCount (see page 6033)	This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

Public Methods

	Name	Description
	Add (see page 6029)	Inserts a new component at the end of the list.
	Clear (see page 6029)	Deletes all components from the list.
	IndexOf (see page 6030)	Returns the index of a specified component in the WebComponents (see page 6034) array.
	LockNotify (see page 6031)	This is LockNotify, a member of class TWebComponentList.
	NotifyLocked (see page 6031)	This is NotifyLocked, a member of class TWebComponentList.
	Remove (see page 6032)	Removes a component from the WebComponents (see page 6034) array.
	SetComponentIndex (see page 6032)	Moves a component to a new position in the WebComponents (see page 6034) array.
	UnlockNotify (see page 6033)	This is UnlockNotify, a member of class TWebComponentList.

58.1.4.4.1 TWebComponentList._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TWebComponentList.

58.1.4.4.2 TWebComponentList._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TWebComponentList.

58.1.4.4.3 TWebComponentList.Add Method

Inserts a new component at the end of the list.

Pascal

```
procedure Add(Component: TComponent);
```

C++

```
__fastcall Add(TComponent * Component);
```

Description

Call Add to insert a component at the end of the WebComponents (see page 6034) array.

Add increments the Count (see page 6033) property. After calling Add, the value of Count (see page 6033) is one greater than the index of the inserted component in the WebComponents (see page 6034) array. To move this component to another position, use the SetComponentIndex (see page 6032) method.

See Also

WebComponents (see page 6034)

Count (see page 6033)

SetComponentIndex (see page 6032)

Remove (see page 6032)

58.1.4.4.4 TWebComponentList.Changed Method

Pascal

```
procedure Changed;
```

C++

```
__fastcall Changed();
```

Description

This is Changed, a member of class TWebComponentList.

58.1.4.4.5 TWebComponentList.Clear Method

Deletes all components from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the WebComponents (see page 6034) array and set Count (see page 6033) to 0.

See Also

Count (see page 6033)

Remove (see page 6032)

WebComponents (see page 6034)

58.1.4.4.6 TWebComponentList.GetComponent Method

Pascal

```
function GetComponent(Index: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(int Index);
```

Description

This is GetComponent, a member of class TWebComponentList.

58.1.4.4.7 TWebComponentList.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TWebComponentList.

58.1.4.4.8 TWebComponentList.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

58.1.4.4.9 TWebComponentList.IndexOf Method

Returns the index of a specified component in the WebComponents (see page 6034) array.

Pascal

```
function IndexOf(Component: TComponent): Integer;
```

C++

```
__fastcall int IndexOf(TComponent * Component);
```

Description

Call IndexOf to get the index for a component in the WebComponents (see page 6034) array. Specify the desired component as the Component parameter.

The first component in the array has index 0, the second component has index 1, and so on. If a component is not in the list, IndexOf returns -1. If a component appears more than once in the array, IndexOf returns the index of the first appearance.

See Also

WebComponents (see page 6034)

58.1.4.4.10 TWebComponentList.LockNotify Method

Pascal

```
procedure LockNotify;
```

C++

```
__fastcall LockNotify();
```

Description

This is LockNotify, a member of class TWebComponentList.

58.1.4.4.11 TWebComponentList.NewComponent Method

Pascal

```
function NewComponent(AOwner: TComponent): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * NewComponent(TComponent * AOwner);
```

Description

This is NewComponent, a member of class TWebComponentList.

58.1.4.4.12 TWebComponentList.NotifyLocked Method

Pascal

```
function NotifyLocked: Boolean;
```

C++

```
__fastcall Boolean NotifyLocked();
```

Description

This is NotifyLocked, a member of class TWebComponentList.

58.1.4.4.13 TWebComponentList.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TWebComponentList.

58.1.4.4.14 TWebComponentList.Remove Method

Removes a component from the WebComponents (see page 6034) array.

Pascal

```
procedure Remove(Component: TComponent);
```

C++

```
__fastcall Remove(TComponent * Component);
```

Description

Call Remove to remove a specific component from the WebComponents (see page 6034) array. After a component is removed, all the WebComponents (see page 6034) that follow it are moved up in index position and Count (see page 6033) is reduced by one.

See Also

Add (see page 6029)

Count (see page 6033)

Clear (see page 6029)

WebComponents (see page 6034)

58.1.4.4.15 TWebComponentList.SetComponentIndex Method

Moves a component to a new position in the WebComponents (see page 6034) array.

Pascal

```
procedure SetComponentIndex(Component: TComponent; Value: Integer);
```

C++

```
__fastcall SetComponentIndex(TComponent * Component, int Value);
```

Description

Call SetComponentIndex to move a component to a new position in the list. Component specifies the component to move. Value specifies the index it should have after it is moved. If Value is less than 0, the component is moved to the first position. If Value is greater than or equal to count, the component is moved to the last position.

See Also

WebComponents (see page 6034)

58.1.4.4.16 TWebComponentList.UnlockNotify Method

Pascal

```
procedure UnlockNotify;
```

C++

```
__fastcall UnlockNotify();
```

Description

This is UnlockNotify, a member of class TWebComponentList.

58.1.4.4.17 TWebComponentList.WebComponentContainer_GetComponent Method

Pascal

```
function WebComponentContainer_GetComponent(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * WebComponentContainer_GetComponent(int I);
```

Description

This is WebComponentContainer_GetComponent, a member of class TWebComponentList.

58.1.4.4.18 TWebComponentList.WebComponentContainer_GetComponentCount Method

Pascal

```
function WebComponentContainer_GetComponentCount: Integer;
```

C++

```
__fastcall int WebComponentContainer_GetComponentCount();
```

Description

This is WebComponentContainer_GetComponentCount, a member of class TWebComponentList.

58.1.4.5 TWebComponentList Properties

The properties of the TWebComponentList class are listed here.

Public Properties

	Name	Description
R	Count (see page 6033)	Indicates the number of components in the list.
R	ParentComponent (see page 6034)	Identifies the component that uses this Web component list to maintain its Web Items.
R	WebComponents (see page 6034)	Lists the Web items in the list.

58.1.4.5.1 TWebComponentList.Count Property

Indicates the number of components in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3634) Count to determine the number of components in the WebComponents (see page 6034) array. Count is a read-only property. To change the size of the WebComponents (see page 6034) array, add or remove components using the Add (see page 6029) and Remove (see page 6032) methods.

See Also

WebComponents (see page 6034)

Add (see page 6029)

Remove (see page 6032)

58.1.4.5.2 TWebComponentList.ParentComponent Property

Identifies the component that uses this Web component list to maintain its Web Items.

Pascal

```
property ParentComponent: TComponent;
```

C++

```
__property TComponent * ParentComponent;
```

Description

Read (see page 3634) ParentComponent to access the parent of the Web Items in the list. The value of ParentComponent is assigned in the constructor when the parent first instantiates the Web component list.

See Also

Create (see page 6027)

58.1.4.5.3 TWebComponentList.WebComponents Property

Lists the Web items in the list.

Pascal

```
property WebComponents [Index: Integer]: TComponent;
```

C++

```
__property TComponent * WebComponents[int Index];
```

Description

Use WebComponents to access the Web components listed by TWebComponentList (see page 6025). Use WebComponents with the Count (see page 6033) property to iterate through all of the components in the list. Use the Add (see page 6029) or Remove (see page 6032) method to add or remove components from the list.

WebComponents is the default property for TWebComponentList (see page 6025). This means that the property name can be omitted when accessing a component in the list. Thus, the line

```
WebItem := InetXPageProducer1.WebPageItems.WebComponents[0];
WebItem = InetXPageProducer1->WebPageItems->WebComponents[0];
```

can be written

```
WebItem := InetXPageProducer1.WebPageItems[0];
WebItem = (*(InetXPageProducer1->WebPageItems))[0];
```

See Also

Count (see page 6033)

Add (see page 6029)

Remove (see page 6032)

58.1.4.6 TWebComponentList Events

The events of the TWebComponentList class are listed here.

Public Events

	Name	Description
	OnChange (see page 6035)	Occurs when a change occurs in the web component list.

58.1.4.6.1 TWebComponentList.OnChange Event

Occurs when a change occurs in the web component list.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

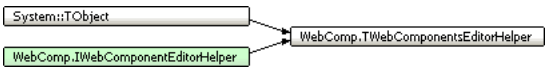
```
__property TNotifyEvent OnChange;
```

Description

Use OnChange to write an event handler to process an action when a change occurs in the web component list, such as when a new component is added.

58.1.5 TWebComponentsEditorHelper Class

Class Hierarchy



Pascal

```
TWebComponentsEditorHelper = class(TObject, IWebComponentEditorHelper);
```

C++

```
class TWebComponentsEditorHelper : public TObject, public IWebComponentEditorHelper;
```

File

WebComp

Description

This is class WebComp.TWebComponentsEditorHelper.

58.1.5.1 TWebComponentsEditorHelper Members

The following tables list the members exposed by TWebComponentsEditorHelper.

Protected Methods

	Name	Description
	_AddRef (see page 6037)	This is _AddRef, a member of class TWebComponentsEditorHelper.
	_Release (see page 6037)	This is _Release, a member of class TWebComponentsEditorHelper.
	CanAddClassHelper (see page 6037)	This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.
	ImplCanAddClassHelper (see page 6037)	This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.
	QueryInterface (see page 6038)	This is QueryInterface, a member of class TWebComponentsEditorHelper.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

58.1.5.2 TWebComponentsEditorHelper Methods

The methods of the TWebComponentsEditorHelper class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 6037)	This is _AddRef, a member of class TWebComponentsEditorHelper.
	_Release (see page 6037)	This is _Release, a member of class TWebComponentsEditorHelper.
	CanAddClassHelper (see page 6037)	This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.
	ImplCanAddClassHelper (see page 6037)	This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.
	QueryInterface (see page 6038)	This is QueryInterface, a member of class TWebComponentsEditorHelper.

58.1.5.2.1 TWebComponentsEditorHelper._AddRef Method

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

This is _AddRef, a member of class TWebComponentsEditorHelper.

58.1.5.2.2 TWebComponentsEditorHelper._Release Method

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

This is _Release, a member of class TWebComponentsEditorHelper.

58.1.5.2.3 TWebComponentsEditorHelper.CanAddClassHelper Method

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass
AClass);
```

Description

This is CanAddClassHelper, a member of class TWebComponentsEditorHelper.

58.1.5.2.4 TWebComponentsEditorHelper.ImplCanAddClassHelper Method

Pascal

```
function ImplCanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass):
```

```
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClassHelper(TComponent * AEditor, TComponent * AParent,
TClass AClass);
```

Description

This is ImplCanAddClassHelper, a member of class TWebComponentsEditorHelper.

58.1.5.2.5 TWebComponentsEditorHelper.QueryInterface Method

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; virtual; stdcall;
```

C++

```
virtual __fastcall __stdcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

This is QueryInterface, a member of class TWebComponentsEditorHelper.

58.1.6 TWebContainedComponent Class

TWebContainedComponent is the base class for Web items that generate HTML on behalf of another component.

Class Hierarchy

Pascal

```
TWebContainedComponent = class(TComponent, IWebComponent);
```

C++

```
class TWebContainedComponent : public TComponent, public IWebComponent;
```

File

WebComp

Description

Use TWebContainedComponent as a base class for Web items. Web items are items that generate HTML statements on behalf of another component. The component that uses the generated HTML has a property whose type is a TWebComponentList (see page 6025) object that lists a set of its Web items.

TWebContainedComponent implements the logic that fits a component into the hierarchy of Web items. TWebContainedComponent descendants act as leaf nodes in this hierarchy. That is, they do not have their own Web items that they use to generate HTML.

In order to allow TWebComponentList (see page 6025) to manage the Web item, TWebContainedComponent implements the IWebComponent (see page 6093) interface.

Do not create instances of TWebContainedComponent. TWebContainedComponent does not produce any HTML, but merely handles the relationship to a managing TWebComponentList (see page 6025) object.

See Also

TWebComponentList (see page 6025)

IWebComponent (see page 6093)

58.1.6.1 TWebContainedComponent Members

The following tables list the members exposed by TWebContainedComponent.

Protected Methods

	Name	Description
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
V	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
V	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.

58.1.6.2 TWebContainedComponent.Create Constructor

Creates and initializes an instance of TWebContainedComponent (see page 6038).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebContainedComponent(TComponent * AOwner);
```

Description

Call Create to instantiate a contained component at runtime. Web items added at design time are instantiated automatically.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainedComponent (see page 6038) instance.

See Also

Owner

58.1.6.3 TWebContainedComponent.Destroy Destructor

Frees an instance of TWebContainedComponent (see page 6038).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedComponent();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TWebContainedComponent (see page 6038).

58.1.6.4 TWebContainedComponent Methods

The methods of the TWebContainedComponent class are listed here.

Protected Methods

	Name	Description
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Public Methods

	Name	Description
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.

58.1.6.4.1 TWebContainedComponent.GetContainerList Method

Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.

Pascal

```
function GetContainerList: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetContainerList();
```

Description

Applications can't access the protected GetContainerList method. It is used internally to access the object manages this item along with the other Web items of the Web parent.

See Also

IsDefaultField (see page 6042)

WebParent (see page 6044)

58.1.6.4.2 TWebContainedComponent.GetParentComponent Method

Returns the object that uses this component to generate HTML.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TWebContainedComponent (see page 6038) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the WebParent (see page 6044) property (the object that uses the HTML generated by this component when assembling its own portion of an HTML document.)

See Also

GetParentComponent

HasParent (see page 6041)

WebParent (see page 6044)

58.1.6.4.3 TWebContainedComponent.HasParent Method

Indicates whether this component appears in another component's list of Web items.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TWebContainedComponent (see page 6038) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the WebParent (see page 6044) property is not nil

(Delphi) or NULL (C++).

If the contained component has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 6041)

WebParent (see page 6044)

58.1.6.4.4 TWebContainedComponent.IsDefaultField Method

Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponent (see page 6038) uses IsDefaultField internally to locate the property on the Web parent that lists this item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 6025) that contains this item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if this component does not belong to such a default list.

See Also

GetContainerList (see page 6041)

WebParent (see page 6044)

TWebComponentList (see page 6025)

58.1.6.4.5 TWebContainedComponent.ParentChanged Method

Responds when the WebParent (see page 6044) property changes.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

The protected ParentChanged method is called automatically when the WebParent (see page 6044) property changes. As implemented in TWebContainedComponent (see page 6038), ParentChanged does nothing. Descendant classes can override this method to perform any necessary updates in response to the change in Web parent.

See Also

WebParent (see page 6044)

58.1.6.4.6 TWebContainedComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TWebContainedComponent (see page 6038) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the WebParent (see page 6044) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 6041)

WebParent (see page 6044)

58.1.6.4.7 TWebContainedComponent.SetParentComponent Method

Sets the WebParent (see page 6044) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TWebContainedComponent (see page 6038) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the WebParent (see page 6044) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 6041)

WebParent (see page 6044)

58.1.6.4.8 TWebContainedComponent.SetWebParent Method

Sets the value of the WebParent (see page 6044) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent is used internally to set the read-only WebParent (see page 6044) property. To assign a Web parent in an application, add this component to the TWebComponentList (see page 6025) object that implements a property on the new Web parent.

See Also

WebParent (see page 6044)

Add (see page 6029)

58.1.6.5 TWebContainedComponent Properties

The properties of the TWebContainedComponent class are listed here.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

58.1.6.5.1 TWebContainedComponent.WebContainedComponentHelper Property

Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.

Pascal

```
property WebContainedComponentHelper: TWebContainedComponentHelper;
```

C++

```
__property TWebContainedComponentHelper WebContainedComponentHelper;
```

Description

TWebContainedComponent (see page 6038) uses WebContainedComponentHelper internally to access the separate helper object that implements the IWebComponent (see page 6093) interface. In addition to providing the implementation for the IWebComponent (see page 6093) interface, WebContainedComponentHelper provides the underlying implementation for most of the other properties and methods that are common to components that appear in a list of Web items.

See Also

IWebComponent (see page 6093)

58.1.6.5.2 TWebContainedComponent.WebParent Property

Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

TWebContainedComponent (see page 6038) uses WebParent to access the component on whose behalf it generates HTML. WebParent uses a TWebComponentList (see page 6025) object that includes this TWebContainedComponent (see page 6038) object to represent its Web items.

See Also

TWebComponentList (see page 6025)

58.1.7 TWebContainedComponentHelper Class

TWebContainedComponentHelper implements the common behavior of components that generate HTML on behalf of another component.

Class Hierarchy

Pascal

```
TWebContainedComponentHelper = class(TObject);
```

C++

```
class TWebContainedComponentHelper : public TObject;
```

File

WebComp

Description

Components that generate HTML for another component use TWebContainedComponentHelper to provide the underlying implementation of many of their properties and methods. TWebContainedComponentHelper implements the IWebComponent (see page 6093) interface, which allows a TWebComponentList (see page 6025) object manage a list of child components. In addition, it implements a number of properties and methods that allow a child Web item to interact with its parent item.

See Also

IWebComponent (see page 6093)

TWebContainedComponent (see page 6038)

TWebContainedContainerComponent (see page 6052)

58.1.7.1 TWebContainedComponentHelper Members

The following tables list the members exposed by TWebContainedComponentHelper.

Protected Events

	Name	Description
	OnParentChanged (see page 6052)	Occurs when the WebParent (see page 6051) property changes.

Protected Methods

	Name	Description
	GetContainer (see page 6048)	Returns the value of the WebParent (see page 6051) property.
	GetIndex (see page 6048)	Returns the position of the associated Web item in the parent item's array of Web items.
V	ParentChanged (see page 6049)	Generates an OnParentChanged (see page 6052) event.
	SetComponentList (see page 6049)	Sets the value of the ContainerList (see page 6051) property.
	SetContainer (see page 6050)	Sets the value of the WebParent (see page 6051) property.
	SetIndex (see page 6050)	Sets the position of the associated component in the list specified by ContainerList (see page 6051).
	SetWebParent (see page 6050)	Sets the value of the WebParent (see page 6051) property.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6047)	Creates an instance of TWebContainedComponentHelper (see page 6045).
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6047)	Destroys an instance of TWebContainedComponentHelper (see page 6045).
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	IsDefaultField (see page 6048)	Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	ContainerList (see page 6051)	Indicates the list that contains the associated component.

	WebParent (see page 6051)	Indicates the component that uses the list of Web items that includes the associated component.

58.1.7.2 TWebContainedComponentHelper.Create Constructor

Creates an instance of TWebContainedComponentHelper (see page 6045).

Pascal

```
constructor Create(AComponent: TComponent; AOnParentChanged: TNotifyEvent);
```

C++

```
__fastcall TWebContainedComponentHelper(TComponent * AComponent, TNotifyEvent
AOnParentChanged);
```

Description

Applications do not create instances of TWebContainedComponentHelper (see page 6045). The component that uses TWebContainedComponentHelper (see page 6045) so that it can appear as a child item in a TWebComponentList (see page 6025) object instantiates TWebContainedComponentHelper (see page 6045).

AComponent is the component that uses TWebContainedComponentHelper (see page 6045) to implement those properties and methods that allow it to appear in a TWebComponentList (see page 6025) object.

AOnParentChanged is an event handler that AComponent supplies so that it can update itself when it is moved to the list of a different parent component. This becomes the event handler for the OnParentChanged (see page 6052) event.

See Also

OnParentChanged (see page 6052)

58.1.7.3 TWebContainedComponentHelper.Destroy Destructor

Destroys an instance of TWebContainedComponentHelper (see page 6045).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedComponentHelper();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWebContainedComponentHelper (see page 6045) object is not nil and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

58.1.7.4 TWebContainedComponentHelper Methods

The methods of the TWebContainedComponentHelper class are listed here.

Protected Methods

	Name	Description
	GetContainer (see page 6048)	Returns the value of the WebParent (see page 6051) property.

	GetIndex (see page 6048)	Returns the position of the associated Web item in the parent item's array of Web items.
V	ParentChanged (see page 6049)	Generates an OnParentChanged (see page 6052) event.
	SetComponentList (see page 6049)	Sets the value of the ContainerList (see page 6051) property.
	SetContainer (see page 6050)	Sets the value of the WebParent (see page 6051) property.
	SetIndex (see page 6050)	Sets the position of the associated component in the list specified by ContainerList (see page 6051).
	SetWebParent (see page 6050)	Sets the value of the WebParent (see page 6051) property.

Public Methods

	Name	Description
	IsDefaultField (see page 6048)	Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

58.1.7.4.1 TWebContainedComponentHelper.GetContainer Method

Returns the value of the WebParent (see page 6051) property.

Pascal

```
function GetContainer: TWebComponentContainer;
```

C++

```
__fastcall TWebComponentContainer GetContainer();
```

Description

Use GetContainer to access the component that uses this helper object's associated component to assemble its HTML output. The component that this method returns uses the list specified by ContainerList (see page 6051) to manage its list of Web items.

See Also

WebParent (see page 6051)

ContainerList (see page 6051)

58.1.7.4.2 TWebContainedComponentHelper.GetIndex Method

Returns the position of the associated Web item in the parent item's array of Web items.

Pascal

```
function GetIndex: Integer;
```

C++

```
__fastcall int GetIndex();
```

Description

GetIndex returns the index of the associated component in the list that is the value of the ContainerList (see page 6051) property. The index of the first item is 0, the index of the second item is 1, and so on.

See Also

ContainerList (see page 6051)

58.1.7.4.3 TWebContainedComponentHelper.IsDefaultField Method

Indicates whether the associated component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean;
```

C++

```
__fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponentHelper (see page 6045) uses IsDefaultField internally to locate the property on the Web parent that lists the associated item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 6025) that contains the associated item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if the associated component does not belong to such a default list.

See Also

ContainerList (see page 6051)

WebParent (see page 6051)

TWebComponentList (see page 6025)

58.1.7.4.4 TWebContainedComponentHelper.ParentChanged Method

Generates an OnParentChanged (see page 6052) event.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

ParentChanged is called automatically when the WebParent (see page 6051) property changes. It generates an OnParentChanged (see page 6052) event, which is handled by the event handler that the associated component passes to the TWebContainedComponentHelper (see page 6045) constructor.

See Also

OnParentChanged (see page 6052)

Create (see page 6047)

58.1.7.4.5 TWebContainedComponentHelper.SetComponentList Method

Sets the value of the ContainerList (see page 6051) property.

Pascal

```
procedure SetComponentList(List: TObject);
```

C++

```
__fastcall SetComponentList(TObject * List);
```

Description

SetComponentList assigns the object specified by List as the value of the ContainerList (see page 6051) property. Note that List must be an instance of TWebComponentList (see page 6025) (or nil (Delphi) or NULL (C++)).

See Also

ContainerList (see page 6051)

58.1.7.4.6 TWebContainedComponentHelper.SetContainer Method

Sets the value of the WebParent (see page 6051) property.

Pascal

```
procedure SetContainer(Container: TWebComponentContainer);
```

C++

```
__fastcall SetContainer(TWebComponentContainer Container);
```

Description

SetContainer assigns a new value to the WebParent (see page 6051) property.

Container is the new Web parent.

See Also

WebParent (see page 6051)

58.1.7.4.7 TWebContainedComponentHelper.SetIndex Method

Sets the position of the associated component in the list specified by ContainerList (see page 6051).

Pascal

```
procedure SetIndex(Value: Integer);
```

C++

```
__fastcall SetIndex(int Value);
```

Description

SetIndex moves the associated component to a specified position in the list of Web items that contains it.

Value is the new index of the associated component, where 0 specifies the first position, 1 specifies the second position, and so on.

See Also

ContainerList (see page 6051)

58.1.7.4.8 TWebContainedComponentHelper.SetWebParent Method

Sets the value of the WebParent (see page 6051) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent moves the associated component to the component list of a new parent. To do so, it removes the associated component from the current parent's list of Web items, adds it to the new Web parent's list of Web items, and then generates an OnParentChanged (see page 6052) event.

Value is the new index of the associated component, where 0 specifies the first position, 1 specifies the second position, and so on.

Value is the new Web parent.

See Also

WebParent (see page 6051)

OnParentChanged (see page 6052)

ContainerList (see page 6051)

58.1.7.5 TWebContainedComponentHelper Properties

The properties of the TWebContainedComponentHelper class are listed here.

Public Properties

	Name	Description
	ContainerList (see page 6051)	Indicates the list that contains the associated component.
	WebParent (see page 6051)	Indicates the component that uses the list of Web items that includes the associated component.

58.1.7.5.1 TWebContainedComponentHelper.ContainerList Property

Indicates the list that contains the associated component.

Pascal

```
property ContainerList: TWebComponentList;
```

C++

```
__property TWebComponentList ContainerList;
```

Description

Use ContainerList to access the list in which the associated component appears. The parent item uses the components in the list to generate HTML, which it then assembles into its own HTML output.

See Also

WebParent (see page 6051)

58.1.7.5.2 TWebContainedComponentHelper.WebParent Property

Indicates the component that uses the list of Web items that includes the associated component.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

WebParent specifies the component on whose behalf the associated component generates HTML. WebParent is the component that uses the list that is the value of the ContainerList (see page 6051) property.

See Also

ContainerList (see page 6051)

58.1.7.6 TWebContainedComponentHelper Events

The events of the TWebContainedComponentHelper class are listed here.

Protected Events

	Name	Description
	OnParentChanged (see page 6052)	Occurs when the WebParent (see page 6051) property changes.

58.1.7.6.1 TWebContainedComponentHelper.OnParentChanged Event

Occurs when the WebParent (see page 6051) property changes.

Pascal

```
property OnParentChanged: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnParentChanged;
```

Description

Components that use TWebContainedComponentHelper (see page 6045) so that they can act as a child Web item in a hierarchy of Web items assign an event handler to OnParentChanged.

The associated component uses TWebContainedComponentHelper (see page 6045) to implement the IWebComponent (see page 6093) interface. This interface can be used to change the value of WebParent (see page 6051). By assigning an OnParentChanged event handler, the associated component can update itself when it moves from one parent item to another.

See Also

WebParent (see page 6051)

IWebComponent (see page 6093)

58.1.8 TWebContainedContainerComponent Class

TWebContainedContainerComponent is the base class for Web items that generate HTML on behalf of another component and also use Web items to generate HTML.

Class Hierarchy

Pascal

```
TWebContainedContainerComponent = class(TComponent, IGetWebComponentList, IWebComponent);
```

C++

```
class TWebContainedContainerComponent : public TComponent, public IGetWebComponentList, public
IWebComponent;
```

File

WebComp

Description

Use TWebContainedContainerComponent as a base class for Web items that have their own subitems. Web items are items that generate HTML statements on behalf of another component. The component that uses the generated HTML has a property whose type is a TWebComponentList (see page 6025) object that lists a set of its Web items. In addition, the TWebContainedContainerComponent descendant uses a TWebComponentList (see page 6025) object to store a set of Web items that it uses.

TWebContainedContainerComponent implements the logic that fits a component into the hierarchy of Web items. TWebContainedContainerComponent descendants act as internal nodes in this hierarchy. That is, they have both a Web parent and a set of child Web items.

In order to allow the parent's TWebComponentList (see page 6025) object to manage this Web item, TWebContainedContainerComponent implements the IWebComponent (see page 6093) interface.

Do not create instances of TWebContainedContainerComponent. TWebContainedContainerComponent does not produce any HTML, but merely handles the relationships to parent and child TWebComponentList (see page 6025) objects.

See Also

TWebComponentList (see page 6025)

IWebComponent (see page 6093)

58.1.8.1 TWebContainedContainerComponent Members

The following tables list the members exposed by TWebContainedContainerComponent.

Protected Methods

	Name	Description
	GetChildren (see page 6055)	Executes a callback for each child Web item.
	GetContainerList (see page 6056)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 6056)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6057)	Returns the list of Web items that this item uses to generate its HTML.
	ImplGetDefaultComponentList (see page 6058)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	IsDefaultField (see page 6058)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 6058)	Responds when the WebParent (see page 6062) property changes.
	ReadState (see page 6059)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetChildOrder (see page 6059)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 6060)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetParentComponent (see page 6060)	Sets the WebParent (see page 6062) property to be the same as this component's parent.
	SetWebParent (see page 6061)	Sets the value of the WebParent (see page 6062) property.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6061)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.

	WebContainerComponentHelper (see page 6062)	Provides access to the helper object that allows this object to manage sub Web items.
	WebParent (see page 6062)	Indicates the component that uses this TWebContainedContainerComponent (see page 6052) as one of its Web items.

Public Methods

	Name	Description
	Create (see page 6054)	Creates and initializes an instance of TWebContainedContainerComponent (see page 6052).
	Destroy (see page 6054)	Frees an instance of TWebContainedContainerComponent (see page 6052).
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetParentComponent (see page 6056)	Returns the object that uses this component to generate HTML.
	HasParent (see page 6057)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.

58.1.8.2 TWebContainedContainerComponent.Create Constructor

Creates and initializes an instance of TWebContainedContainerComponent (see page 6052).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebContainedContainerComponent(TComponent * AOwner);
```

Description

Call Create to instantiate a contained component at runtime. Web items added at design time using the Web Page editor are instantiated automatically.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainedContainerComponent (see page 6052) instance.

See Also

Owner

58.1.8.3 TWebContainedContainerComponent.Destroy Destructor

Frees an instance of TWebContainedContainerComponent (see page 6052).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainedContainerComponent();
```

Description

Destroy removes this component from its parent's list of components that generate HTML before freeing the memory used by TWebContainedContainerComponent (see page 6052).

58.1.8.4 TWebContainedContainerComponent Methods

The methods of the TWebContainedContainerComponent class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 6055)	Executes a callback for each child Web item.
	GetContainerList (see page 6056)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDefaultWebComponents (see page 6056)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6057)	Returns the list of Web items that this item uses to generate its HTML.
	ImplGetDefaultComponentList (see page 6058)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	IsDefaultField (see page 6058)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	ParentChanged (see page 6058)	Responds when the WebParent (see page 6062) property changes.
	ReadState (see page 6059)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetChildOrder (see page 6059)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 6060)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetParentComponent (see page 6060)	Sets the WebParent (see page 6062) property to be the same as this component's parent.
	SetWebParent (see page 6061)	Sets the value of the WebParent (see page 6062) property.

Public Methods

	Name	Description
	GetParentComponent (see page 6056)	Returns the object that uses this component to generate HTML.
	HasParent (see page 6057)	Indicates whether this component appears in another component's list of Web items.

58.1.8.4.1 TWebContainedContainerComponent.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TWebContainedContainerComponent (see page 6052) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It executes the Proc procedure for each child that in the list that

GetWebComponents (see page 6057) returns, as long as that component has Root for an Owner. This allows the sub Web items to be loaded and saved with the TWebContainedContainerComponent (see page 6052) object.

See Also

GetWebComponents (see page 6057)

SetChildOrder (see page 6059)

58.1.8.4.2 TWebContainedContainerComponent.GetContainerList Method

Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.

Pascal

```
function GetContainerList: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetContainerList();
```

Description

Applications can't access the protected GetContainerList method. It is used internally to access the object manages this item along with the other Web items of the Web parent.

See Also

IsDefaultField (see page 6058)

WebParent (see page 6062)

58.1.8.4.3 TWebContainedContainerComponent.GetDefaultWebComponents Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this Web item. As implemented in TWebContainedContainerComponent (see page 6052), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++). Descendant classes override this method to supply a set of default items.

See Also

GetWebComponents (see page 6057)

IsDefaultField (see page 6058)

58.1.8.4.4 TWebContainedContainerComponent.GetParentComponent Method

Returns the object that uses this component to generate HTML.

Pascal

```
function GetParentComponent: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetParentComponent();
```

Description

TWebContainedContainerComponent (see page 6052) overrides the GetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It returns the value of the WebParent (see page 6062) property (the object that uses the HTML generated by this component when assembling its own portion of an HTML document.)

See Also

GetParentComponent

HasParent (see page 6057)

WebParent (see page 6062)

58.1.8.4.5 TWebContainedContainerComponent.GetWebComponents Method

Returns the list of Web items that this item uses to generate its HTML.

Pascal

```
function GetWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetWebComponents();
```

Description

GetWebComponents returns the TWebComponentList (see page 6025) object that manages all of the child items of this component.

See Also

GetDefaultWebComponents (see page 6056)

58.1.8.4.6 TWebContainedContainerComponent.HasParent Method

Indicates whether this component appears in another component's list of Web items.

Pascal

```
function HasParent: Boolean; override;
```

C++

```
virtual __fastcall Boolean HasParent();
```

Description

TWebContainedContainerComponent (see page 6052) overrides the HasParent method that is introduced in TComponent for the streaming system that loads and saves components. It returns true if the WebParent (see page 6062) property is not nil (Delphi) or NULL (C++).

If the contained component has a parent, that parent component is responsible for writing this component to a stream.

See Also

HasParent

GetParentComponent (see page 6056)

WebParent (see page 6062)

58.1.8.4.7 TWebContainedContainerComponent.ImplGetDefaultComponentList Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function ImplGetDefaultComponentList: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetDefaultComponentList();
```

Description

ImplGetDefaultComponentList supplies the default set of child items for this Web item. ImplGetDefaultComponentList calls the virtual GetDefaultWebComponents (see page 6056) method, which descendant classes override to return this list.

See Also

GetDefaultWebComponents (see page 6056)

IsDefaultField (see page 6058)

58.1.8.4.8 TWebContainedContainerComponent.IsDefaultField Method

Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.

Pascal

```
function IsDefaultField: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean IsDefaultField();
```

Description

TWebContainedComponent (see page 6038) uses IsDefaultField internally to locate the property on the Web parent that lists this item along with other Web items.

IsDefaultField returns true if the TWebComponentList (see page 6025) that contains this item represents a set of Web items that are used when the application does not explicitly assign Web items to the Web parent. IsDefaultField returns false if this component does not belong to such a default list.

See Also

GetContainerList (see page 6056)

WebParent (see page 6062)

TWebComponentList (see page 6025)

GetDefaultWebComponents (see page 6056)

GetWebComponents (see page 6057)

58.1.8.4.9 TWebContainedContainerComponent.ParentChanged Method

Responds when the WebParent (see page 6062) property changes.

Pascal

```
procedure ParentChanged; virtual;
```

C++

```
virtual __fastcall ParentChanged();
```

Description

The protected ParentChanged method is called automatically when the WebParent (see page 6062) property changes. As implemented in TWebContainedContainerComponent (see page 6052), ParentChanged does nothing. Descendant classes can override this method to perform any necessary updates in response to the change in Web parent.

See Also

WebParent (see page 6062)

58.1.8.4.10 TWebContainedContainerComponent.ReadState Method

Reads the component's published properties, stored data, and owned components from a reader object.

Pascal

```
procedure ReadState(Reader: TReader); override;
```

C++

```
virtual __fastcall ReadState(TReader Reader);
```

Description

ReadState is part of a sequence of calls used by the streaming system that loads and saves components. It reads the values of all the component's published properties, stored data, and owned components from the reader object passed in Reader. Do not call ReadState directly.

TWebContainedContainerComponent (see page 6052) overrides ReadState to obtain the parent component from the reader, which it assigns as the value of the WebParent (see page 6062) property.

Note: Any descendant classes overriding ReadState should end with a call to the inherited ReadState method.

See Also

GetParentComponent (see page 6056)

WebParent (see page 6062)

58.1.8.4.11 TWebContainedContainerComponent.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder is part of the streaming system that loads and saves components. It positions a newly loaded child component in the list that GetWebComponents (see page 6057) returns.

Child is the newly loaded child component.

Order indicates its position in the list of child Web items, where 0 is the first position, 1 is the second position, and so on.

See Also

GetWebComponents (see page 6057)

SetChildOrder

58.1.8.4.12 TWebContainedContainerComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); virtual;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents assigns the default set of child items for this Web item. This default list can be retrieved by calling the GetDefaultWebComponents (see page 6056) method.

As implemented in TWebContainedContainerComponent (see page 6052), SetDefaultWebComponents does nothing. Descendant classes override this method to store the list of default items in a class-specific manner.

See Also

GetDefaultWebComponents (see page 6056)

IsDefaultField (see page 6058)

58.1.8.4.13 TWebContainedContainerComponent.SetParentComponent Method

Sets the WebParent (see page 6062) property to be the same as this component's parent.

Pascal

```
procedure SetParentComponent(AParent: TComponent); override;
```

C++

```
virtual __fastcall SetParentComponent(TComponent * AParent);
```

Description

TWebContainedContainerComponent (see page 6052) overrides the SetParentComponent method that is introduced in TComponent for the streaming system that loads and saves components. It sets the WebParent (see page 6062) property to AParent, so that it is the same as the parent that loads and saves this component.

See Also

SetParentComponent

HasParent (see page 6057)

WebParent (see page 6062)

58.1.8.4.14 TWebContainedContainerComponent.SetWebParent Method

Sets the value of the WebParent (see page 6062) property.

Pascal

```
procedure SetWebParent(Value: TComponent);
```

C++

```
__fastcall SetWebParent(TComponent * Value);
```

Description

SetWebParent is used internally to set the read-only WebParent (see page 6062) property. To assign a Web parent in an application, add this component to the TWebComponentList (see page 6025) object that implements a property on the new Web parent.

See Also

WebParent (see page 6062)

Add (see page 6029)

58.1.8.5 TWebContainedContainerComponent Properties

The properties of the TWebContainedContainerComponent class are listed here.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6061)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebContainerComponentHelper (see page 6062)	Provides access to the helper object that allows this object to manage sub Web items.
	WebParent (see page 6062)	Indicates the component that uses this TWebContainedContainerComponent (see page 6052) as one of its Web items.

58.1.8.5.1 TWebContainedContainerComponent.WebContainedComponentHelper Property

Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.

Pascal

```
property WebContainedComponentHelper: TWebContainedComponentHelper;
```

C++

```
__property TWebContainedComponentHelper WebContainedComponentHelper;
```

Description

TWebContainedContainerComponent (see page 6052) uses WebContainedComponentHelper internally to access the separate helper object that implements the IWebComponent (see page 6093) interface. In addition to providing the implementation for the IWebComponent (see page 6093) interface, WebContainedComponentHelper provides the underlying implementation for other properties and methods that are common to components that appear in a list of Web items.

See Also

IWebComponent (see page 6093)

58.1.8.5.2 TWebContainedContainerComponent.WebContainerComponentHelper Property

Provides access to the helper object that allows this object to manage sub Web items.

Pascal

```
property WebContainerComponentHelper: TWebContainerComponentHelper;
```

C++

```
__property TWebContainerComponentHelper WebContainerComponentHelper;
```

Description

TWebContainedContainerComponent (see page 6052) uses WebContainerComponentHelper internally to handle two lists of child Web items:

One list contains the default Web items that this component uses if the application does not explicitly assign sub Web items. The GetDefaultWebComponents (see page 6056) method returns this list.

The other list contains explicitly assigned Web items that this component uses. The GetWebComponents (see page 6057) method returns this list.

See Also

GetDefaultWebComponents (see page 6056)

GetWebComponents (see page 6057)

58.1.8.5.3 TWebContainedContainerComponent.WebParent Property

Indicates the component that uses this TWebContainedContainerComponent (see page 6052) as one of its Web items.

Pascal

```
property WebParent: TComponent;
```

C++

```
__property TComponent * WebParent;
```

Description

TWebContainedContainerComponent (see page 6052) uses WebParent to access the component on whose behalf it generates HTML. WebParent uses a TWebComponentList (see page 6025) object that includes this TWebContainedContainerComponent (see page 6052) object to represent its child Web items.

See Also

TWebComponentList (see page 6025)

58.1.9 TWebContainerComponent Class

TWebContainerComponent is the base class for components that use Web items to generate HTML.

Class Hierarchy

Pascal

```
TWebContainerComponent = class(TComponent, IGetWebComponentList);
```

C++

```
class TWebContainerComponent : public TComponent, public IGetWebComponentList;
```

File

WebComp

Description

Use TWebContainerComponent as a base class for objects that maintain a collection of Web items but that do not have a parent Web item. TWebContainerComponent uses a TWebComponentList (see page 6025) object to store the set of child items that it manages.

Do not create instances of TWebContainerComponent, it is intended only as a base class. It only provides support for storing and retrieving child Web items.

See Also

TWebComponentList (see page 6025)

IWebComponent (see page 6093)

TWebContainedContainerComponent (see page 6052)

58.1.9.1 TWebContainerComponent Members

The following tables list the members exposed by TWebContainerComponent.

Protected Methods

	Name	Description
	GetChildren (see page 6065)	Executes a callback for each child Web item.
	GetDefaultWebComponents (see page 6065)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6065)	Returns the list of Web items that this item manages.
	ImplGetDefaultComponentList (see page 6066)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	SetChildOrder (see page 6066)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 6067)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Protected Properties

	Name	Description
	WebContainerComponentHelper (see page 6067)	Provides access to the helper object that allows this object to manage its Web items.

Public Methods

	Name	Description
	Create (see page 6064)	Creates and initializes an instance of TWebContainerComponent (see page 6063).
	Destroy (see page 6064)	Frees an instance of TWebContainerComponent (see page 6063).
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

58.1.9.2 TWebContainerComponent.Create Constructor

Creates and initializes an instance of TWebContainerComponent (see page 6063).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebContainerComponent(TComponent * AOwner);
```

Description

Call Create to instantiate a contained component at runtime.

AOwner is the component, typically a Web module, that becomes the value of the Owner property. The Owner is responsible for initiating the loading and saving of this component, and for freeing the TWebContainerComponent (see page 6063) instance.

See Also

Owner

58.1.9.3 TWebContainerComponent.Destroy Destructor

Frees an instance of TWebContainerComponent (see page 6063).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainerComponent();
```

Description

Destroy frees the memory used by TWebContainerComponent (see page 6063).

58.1.9.4 TWebContainerComponent Methods

The methods of the TWebContainerComponent class are listed here.

Protected Methods

	Name	Description
	GetChildren (see page 6065)	Executes a callback for each child Web item.
	GetDefaultWebComponents (see page 6065)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.
	GetWebComponents (see page 6065)	Returns the list of Web items that this item manages.
	ImplGetDefaultComponentList (see page 6066)	Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

	SetChildOrder (see page 6066)	Positions a child component in the list of child Web items.
	SetDefaultWebComponents (see page 6067)	Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

58.1.9.4.1 TWebContainerComponent.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent); override;
```

C++

```
virtual __fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

TWebContainerComponent (see page 6063) overrides the GetChildren method that is introduced in TComponent for the streaming system that loads and saves components. It executes the Proc procedure for each child that in the list that GetWebComponents (see page 6065) returns, as long as that component has Root for an Owner. This allows the child Web items to be loaded and saved with the TWebContainerComponent (see page 6063) object.

See Also

GetWebComponents (see page 6065)

GetParentComponent

SetChildOrder (see page 6066)

58.1.9.4.2 TWebContainerComponent.GetDefaultWebComponents Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function GetDefaultWebComponents: TWebComponentList; virtual;
```

C++

```
virtual __fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents supplies the default set of child items for this component. As implemented in TWebContainerComponent (see page 6063), GetDefaultWebComponents always returns nil (Delphi) or NULL (C++). Descendant classes override this method to supply a set of default items.

See Also

GetWebComponents (see page 6065)

IsDefaultField (see page 6042)

58.1.9.4.3 TWebContainerComponent.GetWebComponents Method

Returns the list of Web items that this item manages.

Pascal

```
function GetWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetWebComponents();
```

Description

GetWebComponents returns the TWebComponentList (see page 6025) object that manages all of the child items of this component.

See Also

GetDefaultWebComponents (see page 6065)

58.1.9.4.4 TWebContainerComponent.ImplGetDefaultComponentList Method

Returns a list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
function ImplGetDefaultComponentList: TObject; virtual;
```

C++

```
virtual __fastcall TObject * ImplGetDefaultComponentList();
```

Description

ImplGetDefaultComponentList supplies the default set of child items for this Web item. ImplGetDefaultComponentList calls the virtual GetDefaultWebComponents (see page 6065) method, which descendant classes override to return this list.

See Also

GetDefaultWebComponents (see page 6065)

IsDefaultField (see page 6042)

58.1.9.4.5 TWebContainerComponent.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer); override;
```

C++

```
virtual __fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder is part of the streaming system that loads and saves components. It positions a newly loaded child component in the list that GetWebComponents (see page 6065) returns.

Child is the newly loaded child component.

Order indicates its position in the list of child Web items, where 0 is the first position, 1 is the second position, and so on.

See Also

GetWebComponents (see page 6065)

SetChildOrder

58.1.9.4.6 TWebContainerComponent.SetDefaultWebComponents Method

Sets the list of Web items that are used by default if an application does not explicitly assign child items to this object.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList); virtual;
```

C++

```
virtual __fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

SetDefaultWebComponents assigns the default set of child items for this Web item. This default list can be retrieved by calling the GetDefaultWebComponents (see page 6065) method.

As implemented in TWebContainerComponent (see page 6063), SetDefaultWebComponents does nothing. Descendant classes override this method to store the list of default items in a class-specific manner.

See Also

GetDefaultWebComponents (see page 6065)

IsDefaultField (see page 6042)

58.1.9.5 TWebContainerComponent Properties

The properties of the TWebContainerComponent class are listed here.

Protected Properties

	Name	Description
	WebContainerComponentHelper (see page 6067)	Provides access to the helper object that allows this object to manage its Web items.

58.1.9.5.1 TWebContainerComponent.WebContainerComponentHelper Property

Provides access to the helper object that allows this object to manage its Web items.

Pascal

```
property WebContainerComponentHelper: TWebContainerComponentHelper;
```

C++

```
__property TWebContainerComponentHelper WebContainerComponentHelper;
```

Description

TWebContainerComponent (see page 6063) uses WebContainerComponentHelper internally to handle two lists of child Web items:

One list contains the default Web items that this component uses if the application does not explicitly assign child Web items. The GetDefaultWebComponents (see page 6065) method returns this list.

The other list contains explicitly assigned Web items that this component uses. The GetWebComponents (see page 6065) method returns this list.

See Also

GetDefaultWebComponents (see page 6065)

GetWebComponents (see page 6065)

58.1.10 TWebContainerComponentHelper Class

TWebContainerComponentHelper implements the common behavior of components that maintain a list of Web items to generate HTML.

Class Hierarchy

Pascal

```
TWebContainerComponentHelper = class(TObject);
```

C++

```
class TWebContainerComponentHelper : public TObject;
```

File

WebComp

Description

Components that maintain a list of Web items to help them generate HTML use TWebContainerComponentHelper to provide the common implementation of properties and methods for managing the list of sub components. This includes

The logic for handling two lists of Web items: one that represents the Web items that the application assigns, and one that represents a set of default items that can be used when the application does not explicitly assign Web items.

The logic for interacting with the component streaming system, so that child Web items are correctly loaded and saved with the component that uses TWebContainerComponentHelper.

See Also

TWebContainedContainerComponent (see page 6052)

TWebContainerComponent (see page 6063)

TWebComponentList (see page 6025)

58.1.10.1 TWebContainerComponentHelper Members

The following tables list the members exposed by TWebContainerComponentHelper.

Protected Events

	Name	Description
	OnGetDefaultComponentList (see page 6075)	Occurs when TWebContainerComponentHelper (see page 6068) needs to access the value of DefaultWebComponents (see page 6074).
	OnSetDefaultComponentList (see page 6075)	Occurs when TWebContainerComponentHelper (see page 6068) needs to assign the value of DefaultWebComponents (see page 6074).

Protected Methods

	Name	Description
	GetComponentList (see page 6071)	Returns the value of the Webcomponents (see page 6074) property.
	GetDefaultComponentList (see page 6072)	Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.
	GetDefaultWebComponents (see page 6072)	Generates an OnGetDefaultComponentList (see page 6075) event.

	SetDefaultWebComponents (see page 6073)	Generates an OnSetDefaultComponentList (see page 6075) event.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6069)	Creates an instance of TWebContainerComponentHelper (see page 6068).
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6070)	Destroys an instance of TWebContainerComponentHelper (see page 6068).
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetChildren (see page 6071)	Executes a callback for each child Web item.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetChildOrder (see page 6072)	Positions a child component in the list of child Web items.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultWebComponents (see page 6074)	Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.
R	Webcomponents (see page 6074)	Lists the Web items of the associated component.

58.1.10.2 TWebContainerComponentHelper.Create Constructor

Creates an instance of TWebContainerComponentHelper (see page 6068).

Pascal

```
constructor Create(AComponent: TComponent; AOnGetDefaultComponentList: TGetComponentListEvent;
AOnSetDefaultComponentList: TSetComponentListEvent);
```

C++

```
__fastcall TWebContainerComponentHelper(TComponent * AComponent, TGetComponentListEvent
AOnGetDefaultComponentList, TSetComponentListEvent AOnSetDefaultComponentList);
```

Description

Applications do not create instances of TWebContainerComponentHelper (see page 6068). The component that uses TWebContainerComponentHelper (see page 6068) so that it can manage a set of Web items instantiates TWebContainerComponentHelper (see page 6068).

AComponent is the component that uses TWebContainerComponentHelper (see page 6068) to manage its list of Web items and to interact with the component streaming system.

AOnGetDefaultComponentList is an event handler that AComponent supplies to provide the read implementation of the DefaultWebComponents (see page 6074) property. It is assigned as the event handler for OnGetDefaultComponentList (see page 6075).

AOnSetDefaultComponentList is an event handler that AComponent supplies to provide the write implementation of the DefaultWebComponents (see page 6074) property. It is assigned as the event handler for OnSetDefaultComponentList (see page 6075).

See Also

OnGetDefaultComponentList (see page 6075)

OnSetDefaultComponentList (see page 6075)

DefaultWebComponents (see page 6074)

58.1.10.3 TWebContainerComponentHelper.Destroy Destructor

Destroys an instance of TWebContainerComponentHelper (see page 6068).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebContainerComponentHelper();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWebContainerComponentHelper (see page 6068) object is not nil and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

58.1.10.4 TWebContainerComponentHelper Methods

The methods of the TWebContainerComponentHelper class are listed here.

Protected Methods

	Name	Description
	GetComponentList (see page 6071)	Returns the value of the Webcomponents (see page 6074) property.
	GetDefaultComponentList (see page 6072)	Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.

	GetDefaultWebComponents (see page 6072)	Generates an OnGetDefaultComponentList (see page 6075) event.
	SetDefaultWebComponents (see page 6073)	Generates an OnSetDefaultComponentList (see page 6075) event.

Public Methods

	Name	Description
	GetChildren (see page 6071)	Executes a callback for each child Web item.
	SetChildOrder (see page 6072)	Positions a child component in the list of child Web items.

58.1.10.4.1 TWebContainerComponentHelper.GetChildren Method

Executes a callback for each child Web item.

Pascal

```
procedure GetChildren(Proc: TGetChildProc; Root: TComponent);
```

C++

```
__fastcall GetChildren(TGetChildProc Proc, TComponent * Root);
```

Description

GetChildren provides the underlying implementation of the associated component's GetChildren method. GetChildren is introduced in TComponent for the streaming system that loads and saves components.

GetChildren executes the Proc procedure for each child that in the list specified by the Webcomponents (see page 6074) property, as long as that component has Root for an Owner. This allows the sub Web items to be loaded and saved with the component that uses TWebContainerComponentHelper (see page 6068).

See Also

Webcomponents (see page 6074)

SetChildOrder (see page 6072)

GetChildren

58.1.10.4.2 TWebContainerComponentHelper.GetComponentList Method

Returns the value of the Webcomponents (see page 6074) property.

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

GetComponentList returns the list of Web items that have been assigned to the associated component. The return value is an instance of TWebComponentList (see page 6025).

See Also

Webcomponents (see page 6074)

GetDefaultComponentList (see page 6072)

TWebComponentList (see page 6025)

58.1.10.4.3 TWebContainerComponentHelper.GetDefaultComponentList Method

Returns the list of Web items that is used if an application does not explicitly assign any Web items to the associated component.

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

GetComponentList (see page 6071) returns the value of the DefaultWebComponents (see page 6074) property. The return value is an instance of TWebComponentList (see page 6025).

See Also

DefaultWebComponents (see page 6074)

Webcomponents (see page 6074)

TWebComponentList (see page 6025)

58.1.10.4.4 TWebContainerComponentHelper.GetDefaultWebComponents Method

Generates an OnGetDefaultComponentList (see page 6075) event.

Pascal

```
function GetDefaultWebComponents: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetDefaultWebComponents();
```

Description

GetDefaultWebComponents is the read implementation of the DefaultWebComponents (see page 6074) property. It generates an OnGetDefaultComponentList (see page 6075) event, so that the associated component can supply this list of default items. If there is no OnGetDefaultComponentList (see page 6075) event handler, GetDefaultWebComponents returns nil (Delphi) or NULL (C++).

See Also

OnGetDefaultComponentList (see page 6075)

DefaultWebComponents (see page 6074)

58.1.10.4.5 TWebContainerComponentHelper.SetChildOrder Method

Positions a child component in the list of child Web items.

Pascal

```
procedure SetChildOrder(Component: TComponent; Order: Integer);
```

C++

```
__fastcall SetChildOrder(TComponent * Component, int Order);
```

Description

SetChildOrder provides the underlying implementation of the associated component's SetChildOrder method. SetChildOrder is introduced in TComponent for the streaming system that loads and saves components. It positions a newly loaded child component in the list specified by Webcomponents (see page 6074).

Child is the newly loaded child component.

Order indicates its position in Webcomponents (see page 6074), where 0 is the first position, 1 is the second position, and so on.

See Also

Webcomponents (see page 6074)

SetChildOrder

GetChildren (see page 6071)

58.1.10.4.6 TWebContainerComponentHelper.SetDefaultWebComponents Method

Generates an OnSetDefaultComponentList (see page 6075) event.

Pascal

```
procedure SetDefaultWebComponents(AList: TWebComponentList);
```

C++

```
__fastcall SetDefaultWebComponents(TWebComponentList AList);
```

Description

GetDefaultWebComponents (see page 6072) is the write implementation of the DefaultWebComponents (see page 6074) property. It generates an OnSetDefaultComponentList (see page 6075) event, so that the associated component can assign the new list of default items.

See Also

Create (see page 6069)

OnSetDefaultComponentList (see page 6075)

DefaultWebComponents (see page 6074)

58.1.10.5 TWebContainerComponentHelper Properties

The properties of the TWebContainerComponentHelper class are listed here.

Public Properties

	Name	Description
	DefaultWebComponents (see page 6074)	Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.
R	Webcomponents (see page 6074)	Lists the Web items of the associated component.

58.1.10.5.1 TWebContainerComponentHelper.DefaultWebComponents Property

Lists the Web items that the associated component uses by default if the application does not explicitly assign any Web items.

Pascal

```
property DefaultWebComponents: TWebComponentList;
```

C++

```
__property TWebComponentList DefaultWebComponents;
```

Description

Use DefaultWebComponents to access the list of default Web items that can be used when no items have been explicitly assigned.

TWebContainerComponentHelper (see page 6068) does not actually store this list. It leaves that task to the component that uses TWebContainerComponentHelper (see page 6068). Instead, TWebContainerComponentHelper (see page 6068) uses two events, OnGetDefaultComponentList (see page 6075) and OnSetDefaultComponentList (see page 6075) for reading and writing this list. The associated component that uses TWebContainerComponentHelper (see page 6068) assigns event handlers to these two events, thereby providing the implementation for this property.

See Also

Webcomponents (see page 6074)

OnGetDefaultComponentList (see page 6075)

58.1.10.5.2 TWebContainerComponentHelper.Webcomponents Property

Lists the Web items of the associated component.

Pascal

```
property Webcomponents: TWebComponentList;
```

C++

```
__property TWebComponentList Webcomponents;
```

Description

Webcomponents is the list of Web items that the associated component uses to generate HTML. TWebContainerComponentHelper (see page 6068) stores this list on behalf of the object that uses it.

See Also

DefaultWebComponents (see page 6074)

58.1.10.6 TWebContainerComponentHelper Events

The events of the TWebContainerComponentHelper class are listed here.

Protected Events

	Name	Description
	OnGetDefaultComponentList (see page 6075)	Occurs when TWebContainerComponentHelper (see page 6068) needs to access the value of DefaultWebComponents (see page 6074).
	OnSetDefaultComponentList (see page 6075)	Occurs when TWebContainerComponentHelper (see page 6068) needs to assign the value of DefaultWebComponents (see page 6074).

58.1.10.6.1 TWebContainerComponentHelper.OnGetDefaultComponentList Event

Occurs when TWebContainerComponentHelper (see page 6068) needs to access the value of DefaultWebComponents (see page 6074).

Pascal

```
property OnGetDefaultComponentList: TGetComponentListEvent;
```

C++

```
__property TGetComponentListEvent OnGetDefaultComponentList;
```

Description

The component that uses TWebContainerComponentHelper (see page 6068) assigns an event handler for OnGetDefaultComponentList when it creates the TWebContainerComponentHelper (see page 6068) instance. This event handler provides the underlying read implementation of the DefaultWebComponents (see page 6074) property.

If the component that uses TWebContainerComponentHelper (see page 6068) does not provide an OnGetDefaultComponentList event handler, then the value of the DefaultWebComponents (see page 6074) property is nil (Delphi) or NULL (C++).

See Also

OnSetDefaultComponentList (see page 6075)

DefaultWebComponents (see page 6074)

Create (see page 6069)

58.1.10.6.2 TWebContainerComponentHelper.OnSetDefaultComponentList Event

Occurs when TWebContainerComponentHelper (see page 6068) needs to assign the value of DefaultWebComponents (see page 6074).

Pascal

```
property OnSetDefaultComponentList: TSetComponentListEvent;
```

C++

```
__property TSetComponentListEvent OnSetDefaultComponentList;
```

Description

The component that uses TWebContainerComponentHelper (see page 6068) assigns an event handler for OnSetDefaultComponentList when it creates the TWebContainerComponentHelper (see page 6068) instance. This event handler provides the underlying write implementation of the DefaultWebComponents (see page 6074) property.

If the component that uses TWebContainerComponentHelper (see page 6068) does not provide an OnSetDefaultComponentList event handler, then the value of the DefaultWebComponents (see page 6074) can't be changed.

See Also

OnGetDefaultComponentList (see page 6075)

DefaultWebComponents (see page 6074)

Create (see page 6069)

58.1.11 TWebContentOptions Class

TWebContentOptions controls what a Web item can include in its generated HTML.

Class Hierarchy

Pascal

```
TWebContentOptions = class(TObject);
```

C++

```
class TWebContentOptions : public TObject;
```

File

WebComp

Description

Use TWebContentOptions to influence the HTML generated by a component that supports the IWebContent (see page 6103) interface. Each TWebContentOptions represents a set of flags that are passed to the constructor.

See Also

IWebContent (see page 6103)

58.1.11.1 TWebContentOptions Members

The following tables list the members exposed by TWebContentOptions.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6077)	Creates a new TWebContentOptions (see page 6076) object.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.

	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Flags (see page 6077)	Lists the flags that indicate what type of generated HTML is permitted.

58.1.11.2 TWebContentOptions.Create Constructor

Creates a new TWebContentOptions (see page 6076) object.

Pascal

```
constructor Create(AFlags: TWebContentFlags);
```

C++

```
__fastcall TWebContentOptions(TWebContentFlags AFlags);
```

Description

Call Create to create a TWebContentOptions (see page 6076) instance at runtime. Pass the flags that the object should represent as the value of the AFlags parameter. This value initializes the Flags (see page 6077) property.

See Also

Flags (see page 6077)

58.1.11.3 TWebContentOptions Properties

The properties of the TWebContentOptions class are listed here.

Public Properties

	Name	Description
R	Flags (see page 6077)	Lists the flags that indicate what type of generated HTML is permitted.

58.1.11.3.1 TWebContentOptions.Flags Property

Lists the flags that indicate what type of generated HTML is permitted.

Pascal

```
property Flags: TWebContentFlags;
```

C++

```
__property TWebContentFlags Flags;
```

Description

Read (see page 3634) Flags to determine what types of HTML a Web item can generate.

See Also

Create (see page 6077)

58.2 Interfaces

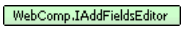
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAddFieldsEditor (see page 6079)	This is class WebComp.IAddFieldsEditor.
	IGetWebComponentList (see page 6081)	This is class WebComp.IGetWebComponentList.
	IGetWebComponentNodeName (see page 6082)	This is class WebComp.IGetWebComponentNodeName.
	IGetXMLStream (see page 6083)	This is class WebComp.IGetXMLStream.
	ILayoutWebContent (see page 6084)	ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.
	INotifyActivateModule (see page 6087)	This is class WebComp.INotifyActivateModule.
	INotifyWebActivate (see page 6088)	This is class WebComp.INotifyWebActivate.
	IProducerEditorViewSupport (see page 6089)	This is class WebComp.IProducerEditorViewSupport.
	ISetWebContentOptions (see page 6091)	This is class WebComp.ISetWebContentOptions.
	ITopLevelWebComponent (see page 6092)	This is class WebComp.ITopLevelWebComponent.
	IWebComponent (see page 6093)	IWebComponent is the interface supported by components that TWebComponentList (see page 6025) manages.
	IWebComponentCommands (see page 6097)	This is class WebComp.IWebComponentCommands.
	IWebComponentContainer (see page 6098)	This is class WebComp.IWebComponentContainer.
	IWebComponentEditor (see page 6100)	IWebComponentEditor enables a Web item to be assigned children in the IDE.
	IWebComponentEditorHelper (see page 6102)	This is class WebComp.IWebComponentEditorHelper.
	IWebContent (see page 6103)	IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.
	IWebVariablesContainerName (see page 6104)	This is class WebComp.IWebVariablesContainerName.

58.2.1 IAddFieldsEditor Interface

Class Hierarchy



Pascal

```
IAddFieldsEditor = interface;
```

C++

```
__interface IAddFieldsEditor;
```

File

WebComp

Description

This is class WebComp.IAddFieldsEditor.

58.2.1.1 IAddFieldsEditor Members

The following tables list the members exposed by IAddFieldsEditor.

Public Methods

	Name	Description
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

58.2.1.2 IAddFieldsEditor Methods

The methods of the IAddFieldsEditor class are listed here.

Public Methods

	Name	Description
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

58.2.1.2.1 IAddFieldsEditor.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.

58.2.1.2.2 IAddFieldsEditor.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.

58.2.1.2.3 IAddFieldsEditor.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class IAddFieldsEditor.

58.2.1.2.4 IAddFieldsEditor.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

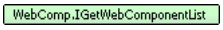
```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class IAddFieldsEditor.

58.2.2 IGetWebComponentList Interface

Class Hierarchy



Pascal

```
IGetWebComponentList = interface;
```

C++

```
__interface IGetWebComponentList;
```

File

WebComp

Description

This is class WebComp.IGetWebComponentList.

58.2.2.1 IGetWebComponentList Members

The following tables list the members exposed by IGetWebComponentList.

Public Methods

	Name	Description
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

58.2.2.2 IGetWebComponentList Methods

The methods of the IGetWebComponentList class are listed here.

Public Methods

	Name	Description
	GetComponentList (see page 6082)	This is GetComponentList, a member of class IGetWebComponentList.
	GetDefaultComponentList (see page 6082)	This is GetDefaultComponentList, a member of class IGetWebComponentList.

58.2.2.2.1 IGetWebComponentList.GetComponentList Method

Pascal

```
function GetComponentList: TObject;
```

C++

```
__fastcall TObject * GetComponentList();
```

Description

This is GetComponentList, a member of class IGetWebComponentList.

58.2.2.2.2 IGetWebComponentList.GetDefaultComponentList Method

Pascal

```
function GetDefaultComponentList: TObject;
```

C++

```
__fastcall TObject * GetDefaultComponentList();
```

Description

This is GetDefaultComponentList, a member of class IGetWebComponentList.

58.2.3 IGetWebComponentNodeName Interface

Class Hierarchy

Pascal

```
IGetWebComponentNodeName = interface;
```

C++

```
__interface IGetWebComponentNodeName;
```

File

WebComp

Description

This is class WebComp.IGetWebComponentNodeName.

58.2.3.1 IGetWebComponentNodeName Members

The following tables list the members exposed by IGetWebComponentNodeName.

Public Methods

	Name	Description
	GetWebComponentNodeName (see page 6083)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

58.2.3.2 IGetWebComponentNodeName Methods

The methods of the IGetWebComponentNodeName class are listed here.

Public Methods

	Name	Description
	GetWebComponentNodeName (see page 6083)	This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

58.2.3.2.1 IGetWebComponentNodeName.GetWebComponentNodeName Method

Pascal

```
function GetWebComponentNodeName: string;
```

C++

```
__fastcall UnicodeString GetWebComponentNodeName();
```

Description

This is GetWebComponentNodeName, a member of class IGetWebComponentNodeName.

58.2.4 IGetXMLStream Interface

Class Hierarchy

WebComp.IGetXMLStream

Pascal

```
IGetXMLStream = interface;
```

C++

```
__interface IGetXMLStream;
```

File

WebComp

Description

This is class WebComp.IGetXMLStream.

58.2.4.1 IGetXMLStream Members

The following tables list the members exposed by IGetXMLStream.

Public Methods

	Name	Description
	GetXMLStream (see page 6084)	This is GetXMLStream, a member of class IGetXMLStream.

58.2.4.2 IGetXMLStream Methods

The methods of the IGetXMLStream class are listed here.

Public Methods

	Name	Description
	GetXMLStream (see page 6084)	This is GetXMLStream, a member of class IGetXMLStream.

58.2.4.2.1 IGetXMLStream.GetXMLStream Method

Pascal

```
function GetXMLStream(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLStream(Boolean Owned);
```

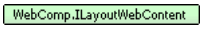
Description

This is GetXMLStream, a member of class IGetXMLStream.

58.2.5 ILayoutWebContent Interface

ILayoutWebContent lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy



Pascal

```
ILayoutWebContent = interface;
```

C++

```
__interface ILayoutWebContent;
```

File

WebComp

Description

The Content method, implemented by all Web items to return the generated HTML, takes as a parameter an object of type TLayout (see page 6112). TLayout (see page 6112) is any object that implements the ILayoutWebContent interface. Typically, this is a descendant of TLayoutWebContent (see page 6014).

Parent components pass the TLayout (see page 6112) object to child Web items to provide helper functions that the child items use to fit their generated content into the parent object's HTML structure (typically a <TABLE> tag). For example, the ILayoutWebContent methods may add <TD> tags around generated HTML that fills the cells of a parent's HTML table.

See Also

TLayout (see page 6112)

IWebContent (see page 6103)

TLayoutWebContent (see page 6014)

58.2.5.1 ILayoutWebContent Members

The following tables list the members exposed by ILayoutWebContent.

Public Methods

	Name	Description
	LayoutButton (see page 6085)	Converts the HTML declaring a button into a form the Web item's parent can use.
	LayoutField (see page 6085)	Converts the HTML declaring a field into a form the Web item's parent can use.
	LayoutLabelAndField (see page 6086)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 6086)	Converts the HTML declaring a table into a form the Web item's parent can use.

58.2.5.2 ILayoutWebContent Methods

The methods of the ILayoutWebContent class are listed here.

Public Methods

	Name	Description
	LayoutButton (see page 6085)	Converts the HTML declaring a button into a form the Web item's parent can use.
	LayoutField (see page 6085)	Converts the HTML declaring a field into a form the Web item's parent can use.
	LayoutLabelAndField (see page 6086)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 6086)	Converts the HTML declaring a table into a form the Web item's parent can use.

58.2.5.2.1 ILayoutWebContent.LayoutButton Method

Converts the HTML declaring a button into a form the Web item's parent can use.

Pascal

```
function LayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutButton(const UnicodeString HTMLButton, TLayoutAttributes
Attributes);
```

Description

Call LayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form required by the Web item's parent.

HTMLButton is the HTML that declares a button. This is produced by the Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that can be used by the Web item's parent.

58.2.5.2.2 ILayoutWebContent.LayoutField Method

Converts the HTML declaring a field into a form the Web item's parent can use.

Pascal

```
function LayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutField(const UnicodeString HTMLField, TLayoutAttributes
Attributes);
```

Description

Call LayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form required by the Web item's parent.

HTMLField is the HTML that declares the field. This is produced by the Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that can be used by the Web item's parent.

58.2.5.2.3 ILayoutWebContent.LayoutLabelAndField Method

Converts the HTML declaring a labeled field into a form the Web item's parent can use.

Pascal

```
function LayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes);
```

Description

Call LayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form required by the Web item's parent.

HTMLLabel is the HTML that declares the label. This is produced by the Web item.

HTMLField is the HTML that declares the field. This is produced by the Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that can be used by the Web item's parent.

58.2.5.2.4 ILayoutWebContent.LayoutTable Method

Converts the HTML declaring a table into a form the Web item's parent can use.

Pascal

```
function LayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
```

C++

```
__fastcall UnicodeString LayoutTable(const UnicodeString HTMLTable, TLayoutAttributes
Attributes);
```

Description

Call LayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the Web item's children) into the form required by the Web item's parent.

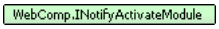
HTMLTable is the HTML that declares the table. This is produced by the Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable returns the HTML for the table in a form that can be used by the Web item's parent.

58.2.6 INotifyActivateModule Interface

Class Hierarchy



Pascal

```
INotifyActivateModule = interface;
```

C++

```
__interface INotifyActivateModule;
```

File

WebComp

Description

This is class WebComp.INotifyActivateModule.

58.2.6.1 INotifyActivateModule Members

The following tables list the members exposed by INotifyActivateModule.

Public Methods

	Name	Description
	NotifyActivateModule (see page 6087)	This is NotifyActivateModule, a member of class INotifyActivateModule.
	NotifyDeactivateModule (see page 6088)	This is NotifyDeactivateModule, a member of class INotifyActivateModule.

58.2.6.2 INotifyActivateModule Methods

The methods of the INotifyActivateModule class are listed here.

Public Methods

	Name	Description
	NotifyActivateModule (see page 6087)	This is NotifyActivateModule, a member of class INotifyActivateModule.
	NotifyDeactivateModule (see page 6088)	This is NotifyDeactivateModule, a member of class INotifyActivateModule.

58.2.6.2.1 INotifyActivateModule.NotifyActivateModule Method

Pascal

```
procedure NotifyActivateModule(AWebModule: TComponent);
```

C++

```
__fastcall NotifyActivateModule(TComponent * AWebModule);
```

Description

This is NotifyActivateModule, a member of class INotifyActivateModule.

58.2.6.2.2 INotifyActivateModule.NotifyDeactivateModule Method

Pascal

```
procedure NotifyDeactivateModule(AWebModule: TComponent);
```

C++

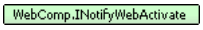
```
__fastcall NotifyDeactivateModule(TComponent * AWebModule);
```

Description

This is NotifyDeactivateModule, a member of class INotifyActivateModule.

58.2.7 INotifyWebActivate Interface

Class Hierarchy



Pascal

```
INotifyWebActivate = interface;
```

C++

```
__interface INotifyWebActivate;
```

File

WebComp

Description

This is class WebComp.INotifyWebActivate.

58.2.7.1 INotifyWebActivate Members

The following tables list the members exposed by INotifyWebActivate.

Public Methods

	Name	Description
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.

58.2.7.2 INotifyWebActivate Methods

The methods of the INotifyWebActivate class are listed here.

Public Methods

	Name	Description
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.

58.2.7.2.1 INotifyWebActivate.NotifyActivate Method

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

This is NotifyActivate, a member of class INotifyWebActivate.

58.2.7.2.2 INotifyWebActivate.NotifyDeactivate Method

Pascal

```
procedure NotifyDeactivate;
```

C++

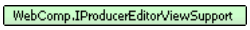
```
__fastcall NotifyDeactivate();
```

Description

This is NotifyDeactivate, a member of class INotifyWebActivate.

58.2.8 IProducerEditorViewSupport Interface

Class Hierarchy



Pascal

```
IProducerEditorViewSupport = interface;
```

C++

```
__interface IProducerEditorViewSupport;
```

File

WebComp

Description

This is class WebComp.IProducerEditorViewSupport.

58.2.8.1 IProducerEditorViewSupport Members

The following tables list the members exposed by IProducerEditorViewSupport.

Public Methods

	Name	Description
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.

	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

58.2.8.2 IProducerEditorViewSupport Methods

The methods of the IProducerEditorViewSupport class are listed here.

Public Methods

	Name	Description
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

58.2.8.2.1 IProducerEditorViewSupport.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class IProducerEditorViewSupport.

58.2.8.2.2 IProducerEditorViewSupport.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class IProducerEditorViewSupport.

58.2.8.2.3 IProducerEditorViewSupport.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class IProducerEditorViewSupport.

58.2.8.2.4 IProducerEditorViewSupport.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.

58.2.8.2.5 IProducerEditorViewSupport.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class IProducerEditorViewSupport.

58.2.8.2.6 IProducerEditorViewSupport.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.

58.2.8.2.7 IProducerEditorViewSupport.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

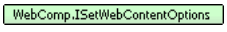
```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.

58.2.9 ISetWebContentOptions Interface

Class Hierarchy



Pascal

```
ISetWebContentOptions = interface;
```

C++

```
__interface ISetWebContentOptions;
```

File

WebComp

Description

This is class WebComp.ISetWebContentOptions.

58.2.9.1 ISetWebContentOptions Members

The following tables list the members exposed by ISetWebContentOptions.

Public Methods

	Name	Description
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

58.2.9.2 ISetWebContentOptions Methods

The methods of the ISetWebContentOptions class are listed here.

Public Methods

	Name	Description
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.

58.2.9.2.1 ISetWebContentOptions.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

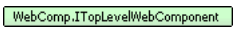
```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class ISetWebContentOptions.

58.2.10 ITopLevelWebComponent Interface

Class Hierarchy



Pascal

```
ITopLevelWebComponent = interface;
```

C++

```
__interface ITopLevelWebComponent;
```

File

WebComp

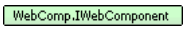
Description

This is class WebComp.ITopLevelWebComponent.

58.2.11 IWebComponent Interface

IWebComponent is the interface supported by components that TWebComponentList (see page 6025) manages.

Class Hierarchy



Pascal

```
IWebComponent = interface;
```

C++

```
__interface IWebComponent;
```

File

WebComp

Description

TWebComponentList (see page 6025) uses IWebComponent to ensure that all components in its list have access to information about how they fit in the hierarchy of Web items.

Note: VCL methods that work with the IWebComponent interface wrap it using the DelphiInterface (see page 3256) class:

Note: typedef System::DelphiInterface (see page 3256)< IWebComponent > _di_IWebComponent;

See Also

TWebComponentList (see page 6025)

58.2.11.1 IWebComponent Members

The following tables list the members exposed by IWebComponent.

Public Methods

	Name	Description
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.

58.2.11.2 IWebComponent Methods

The methods of the IWebComponent class are listed here.

Public Methods

	Name	Description
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

58.2.11.2.1 IWebComponent.GetContainer Method

Returns the value of the Container (see page 6096) property.

Pascal

```
function GetContainer: TWebComponentContainer;
```

C++

```
__fastcall TWebComponentContainer GetContainer();
```

Description

GetContainer is the read method for the Container (see page 6096) property.

See Also

Container (see page 6096)

58.2.11.2.2 IWebComponent.GetIndex Method

Returns the value of the Index (see page 6097) property.

Pascal

```
function GetIndex: Integer;
```

C++

```
__fastcall int GetIndex();
```

Description

GetIndex is the read method for the Index (see page 6097) property.

See Also

Index (see page 6097)

58.2.11.2.3 IWebComponent.SetComponentList Method

Sets the value of the ComponentList (see page 6096) property.

Pascal

```
procedure SetComponentList(List: TObject);
```

C++

```
__fastcall SetComponentList(TObject * List);
```

Description

SetComponentList is the write method for the ComponentList (see page 6096) property. The List parameter specifies the value to assign to ComponentList (see page 6096).

See Also

ComponentList (see page 6096)

58.2.11.2.4 IWebComponent.SetContainer Method

Sets the value of the Container (see page 6096) property.

Pascal

```
procedure SetContainer(Container: TWebComponentContainer);
```

C++

```
__fastcall SetContainer(TWebComponentContainer Container);
```

Description

SetContainer is the write method for the Container (see page 6096) property. The Container (see page 6096) parameter specifies the value to assign to Container (see page 6096).

See Also

Container (see page 6096)

58.2.11.2.5 IWebComponent.SetIndex Method

Sets the value of the Index (see page 6097) property.

Pascal

```
procedure SetIndex(Value: Integer);
```

C++

```
__fastcall SetIndex(int Value);
```

Description

SetIndex is the write method for the Index (see page 6097) property. The Value parameter specifies the value to assign to Index (see page 6097).

See Also

Index (see page 6097)

58.2.11.3 IWebComponent Properties

The properties of the IWebComponent class are listed here.

Public Properties

	Name	Description
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.

58.2.11.3.1 IWebComponent.ComponentList Property

Indicates the component list that contains the Web item.

Pascal

```
property ComponentList: TObject;
```

C++

```
__property TObject * ComponentList;
```

Description

The Web component list assigns a value to ComponentList when the component is added to it. The component can use this value to access the list and obtain information about the other components in the list.

Note that the IWebComponent (see page 6093) interface does not include any methods for reading the value of ComponentList.

See Also

Container (see page 6096)

Index (see page 6097)

TWebComponentList (see page 6025)

58.2.11.3.2 IWebComponent.Container Property

Specifies the component that uses content generated by this Web item.

Pascal

```
property Container: TWebComponentContainer;
```

C++

```
__property TWebComponentContainer Container;
```

Description

Web items are arranged in a hierarchy, where each item generates its content using content produced by its children. Container specifies the parent that uses the content of this Web item. Container maintains a list of all its children using a TWebComponentList (see page 6025) object.

See Also

ComponentList (see page 6096)

ParentComponent (see page 6034)

58.2.11.3.3 IWebComponent.Index Property

Indicates where the Web item appears in the Web component list's array of Web items.

Pascal

```
property Index: Integer;
```

C++

```
__property int Index;
```

Description

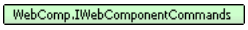
The Web component list sets Index to indicate where the component sits in its WebComponents property array.

See Also

WebComponents (see page 6034)

58.2.12 IWebComponentCommands Interface

Class Hierarchy



Pascal

```
IWebComponentCommands = interface;
```

C++

```
__interface IWebComponentCommands;
```

File

WebComp

Description

This is class WebComp.IWebComponentCommands.

58.2.12.1 IWebComponentCommands Members

The following tables list the members exposed by IWebComponentCommands.

Public Methods

	Name	Description
	ExecuteVerb (see page 6098)	This is ExecuteVerb, a member of class IWebComponentCommands.
	GetVerb (see page 6098)	This is GetVerb, a member of class IWebComponentCommands.
	GetVerbCount (see page 6098)	This is GetVerbCount, a member of class IWebComponentCommands.

58.2.12.2 IWebComponentCommands Methods

The methods of the IWebComponentCommands class are listed here.

Public Methods

	Name	Description
	ExecuteVerb (see page 6098)	This is ExecuteVerb, a member of class IWebComponentCommands.
	GetVerb (see page 6098)	This is GetVerb, a member of class IWebComponentCommands.
	GetVerbCount (see page 6098)	This is GetVerbCount, a member of class IWebComponentCommands.

58.2.12.2.1 IWebComponentCommands.ExecuteVerb Method

Pascal

```
procedure ExecuteVerb(AComponent: TComponent; Index: Integer);
```

C++

```
__fastcall ExecuteVerb(TComponent * AComponent, int Index);
```

Description

This is ExecuteVerb, a member of class IWebComponentCommands.

58.2.12.2.2 IWebComponentCommands.GetVerb Method

Pascal

```
function GetVerb(AComponent: TComponent; Index: Integer): string;
```

C++

```
__fastcall UnicodeString GetVerb(TComponent * AComponent, int Index);
```

Description

This is GetVerb, a member of class IWebComponentCommands.

58.2.12.2.3 IWebComponentCommands.GetVerbCount Method

Pascal

```
function GetVerbCount(AComponent: TComponent): Integer;
```

C++

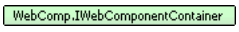
```
__fastcall int GetVerbCount(TComponent * AComponent);
```

Description

This is GetVerbCount, a member of class IWebComponentCommands.

58.2.13 IWebComponentContainer Interface

Class Hierarchy



Pascal

```
IWebComponentContainer = interface;
```

C++

```
__interface IWebComponentContainer;
```

File

WebComp

Description

This is class WebComp.IWebComponentContainer.

58.2.13.1 IWebComponentContainer Members

The following tables list the members exposed by IWebComponentContainer.

Public Methods

	Name	Description
	GetComponent (see page 6099)	This is GetComponent, a member of class IWebComponentContainer.
	GetComponentCount (see page 6099)	This is GetComponentCount, a member of class IWebComponentContainer.

Public Properties

	Name	Description
R	ComponentCount (see page 6100)	This is ComponentCount, a member of class IWebComponentContainer.
R	Components (see page 6100)	This is Components, a member of class IWebComponentContainer.

58.2.13.2 IWebComponentContainer Methods

The methods of the IWebComponentContainer class are listed here.

Public Methods

	Name	Description
	GetComponent (see page 6099)	This is GetComponent, a member of class IWebComponentContainer.
	GetComponentCount (see page 6099)	This is GetComponentCount, a member of class IWebComponentContainer.

58.2.13.2.1 IWebComponentContainer.GetComponent Method

Pascal

```
function GetComponent(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetComponent(int I);
```

Description

This is GetComponent, a member of class IWebComponentContainer.

58.2.13.2.2 IWebComponentContainer.GetComponentCount Method

Pascal

```
function GetComponentCount: Integer;
```

C++

```
__fastcall int GetComponentCount();
```

Description

This is GetComponentCount, a member of class IWebComponentContainer.

58.2.13.3 IWebComponentContainer Properties

The properties of the IWebComponentContainer class are listed here.

Public Properties

	Name	Description
	ComponentCount (see page 6100)	This is ComponentCount, a member of class IWebComponentContainer.
	Components (see page 6100)	This is Components, a member of class IWebComponentContainer.

58.2.13.3.1 IWebComponentContainer.ComponentCount Property

Pascal

```
property ComponentCount: Integer;
```

C++

```
__property int ComponentCount;
```

Description

This is ComponentCount, a member of class IWebComponentContainer.

58.2.13.3.2 IWebComponentContainer.Components Property

Pascal

```
property Components [I: Integer]: TComponent;
```

C++

```
__property TComponent * Components[int I];
```

Description

This is Components, a member of class IWebComponentContainer.

58.2.14 IWebComponentEditor Interface

IWebComponentEditor enables a Web item to be assigned children in the IDE.

Class Hierarchy

WebComp.IWebComponentEditor

Pascal

```
IWebComponentEditor = interface;
```

C++

```
__interface IWebComponentEditor;
```

File

WebComp

Description

When the Web Page editor builds the list of new items that can be added as the child of a Web item, it calls the

IWebComponentEditor interface. Components that contain child items for generating portions of their HTML content must implement this interface so that the Web Page editor can display the appropriate list of components to add.

IWebComponentEditor defines a single method, CanAddClass (see page 6101), which indicates whether a specified class can be used for a child Web item by the Web item that implements the interface.

See Also

IRestoreDefaults

TAdapterPageProducer (see page 1136)

TInetXPageProducer

TWebForm

TWebControlGroup

TAdapterSubComponent (see page 5406)

58.2.14.1 IWebComponentEditor Members

The following tables list the members exposed by IWebComponentEditor.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.

58.2.14.2 IWebComponentEditor Methods

The methods of the IWebComponentEditor class are listed here.

Public Methods

	Name	Description
	CanAddClass (see page 6101)	Indicates whether a specified class can be used as a child Web item.

58.2.14.2.1 IWebComponentEditor.CanAddClass Method

Indicates whether a specified class can be used as a child Web item.

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

The Web page editor calls CanAddClass to determine what Web items can be added to supply content for the Web item that implements this interface.

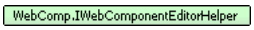
AParent is the Web item that will act as a parent to the added Web item.

AClass is the class of the proposed child Web item.

CanAddClass returns true if the Web item can use content generated by a component of type AClass, false otherwise.

58.2.15 IWebComponentEditorHelper Interface

Class Hierarchy



Pascal

```
IWebComponentEditorHelper = interface;
```

C++

```
__interface IWebComponentEditorHelper;
```

File

WebComp

Description

This is class WebComp.IWebComponentEditorHelper.

58.2.15.1 IWebComponentEditorHelper Members

The following tables list the members exposed by IWebComponentEditorHelper.

Public Methods

	Name	Description
	CanAddClassHelper (see page 6102)	This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

58.2.15.2 IWebComponentEditorHelper Methods

The methods of the IWebComponentEditorHelper class are listed here.

Public Methods

	Name	Description
	CanAddClassHelper (see page 6102)	This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

58.2.15.2.1 IWebComponentEditorHelper.CanAddClassHelper Method

Pascal

```
function CanAddClassHelper(AEditor: TComponent; AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClassHelper(TComponent * AEditor, TComponent * AParent, TClass
AClass);
```

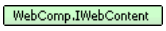
Description

This is CanAddClassHelper, a member of class IWebComponentEditorHelper.

58.2.16 IWebContent Interface

IWebContent is an interface that allows components page producers and Web items to access the HTML generated by other components.

Class Hierarchy



Pascal

```
IWebContent = interface;
```

C++

```
__interface IWebContent;
```

File

WebComp

Description

TInetXPageProducer, TAdapterPageProducer, and TComponentsPageProducer create HTML documents that include content generated by other components, called Web items. Similarly, the Web items that generate content can make use of other Web items when producing their content.

For a component to act as a Web item, it must support the IWebContent interface. This interface defines one new method, Content (see page 6104), which returns the HTML generated by the Web item.

Note: The only other required interface for Web items is IWebComponent (see page 6093).

See Also

TAdapterPageProducer (see page 1136)

TInetXPageProducer

TBasePageItemsProducer

TWebComponentList (see page 6025)

IWebComponent (see page 6093)

Customizing the InternetExpress Page Producer Template

58.2.16.1 IWebContent Members

The following tables list the members exposed by IWebContent.

Public Methods

	Name	Description
	Content (see page 6104)	Returns the HTML that the Web item generates.

58.2.16.2 IWebContent Methods

The methods of the IWebContent class are listed here.

Public Methods

	Name	Description
	Content (see page 6104)	Returns the HTML that the Web item generates.

58.2.16.2.1 IWebContent.Content Method

Returns the HTML that the Web item generates.

Pascal

```
function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout ParentLayout);
```

Description

Call Content to access the HTML produced by the Web item.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web item should lay out its constituent parts. This object supports the ILayoutWebContent (see page 6084) interface, which the Web item uses to fit its own content into the parent's HTML structures.

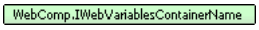
Content returns a string of HTML produced by the Web item.

See Also

ILayoutWebContent (see page 6084)

58.2.17 IWebVariablesContainerName Interface

Class Hierarchy



Pascal

```
IWebVariablesContainerName = interface;
```

C++

```
__interface IWebVariablesContainerName;
```

File

WebComp

Description

This is class WebComp.IWebVariablesContainerName.

58.2.17.1 IWebVariablesContainerName Members

The following tables list the members exposed by IWebVariablesContainerName.

Public Methods

	Name	Description
	GetContainerName (see page 6105)	This is GetContainerName, a member of class IWebVariablesContainerName.

Public Properties

	Name	Description
R	ContainerName (see page 6105)	This is ContainerName, a member of class IWebVariablesContainerName.

58.2.17.2 IWebVariablesContainerName Methods

The methods of the IWebVariablesContainerName class are listed here.

Public Methods

	Name	Description
	GetContainerName (see page 6105)	This is GetContainerName, a member of class IWebVariablesContainerName.

58.2.17.2.1 IWebVariablesContainerName.GetContainerName Method

Pascal

```
function GetContainerName: string;
```

C++

```
__fastcall UnicodeString GetContainerName();
```

Description

This is GetContainerName, a member of class IWebVariablesContainerName.

58.2.17.3 IWebVariablesContainerName Properties

The properties of the IWebVariablesContainerName class are listed here.

Public Properties

	Name	Description
R	ContainerName (see page 6105)	This is ContainerName, a member of class IWebVariablesContainerName.

58.2.17.3.1 IWebVariablesContainerName.ContainerName Property

Pascal

```
property ContainerName: string;
```

C++

```
__property UnicodeString ContainerName;
```

Description

This is ContainerName, a member of class IWebVariablesContainerName.

58.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DesigningComponent (see page 6106)	This is function WebComp.DesigningComponent.
	EnumRegisteredWebComponents (see page 6106)	This is function WebComp.EnumRegisteredWebComponents.
	FindWebComponentEditorHelper (see page 6107)	This is function WebComp.FindWebComponentEditorHelper.
	QualifyFileName (see page 6107)	This is function WebComp.QualifyFileName.
	RegisterWebComponents (see page 6107)	This is function WebComp.RegisterWebComponents.
	UnRegisterWebComponents (see page 6108)	This is function WebComp.UnRegisterWebComponents.
	ValidateWebParent (see page 6108)	This is function WebComp.ValidateWebParent.

58.3.1 WebComp.DesigningComponent Function

Pascal

```
function DesigningComponent(AComponent: TComponent): Boolean;
```

C++

```
Boolean DesigningComponent(TComponent * AComponent);
```

File

WebComp

Description

This is function WebComp.DesigningComponent.

58.3.2 WebComp.EnumRegisteredWebComponents Function

Pascal

```
procedure EnumRegisteredWebComponents(Proc: TEnumWebComponentsProc; Info: Pointer);
```

C++

```
EnumRegisteredWebComponents(TEnumWebComponentsProc Proc, void * Info);
```

File

WebComp

Description

This is function WebComp.EnumRegisteredWebComponents.

58.3.3 WebComp.FindWebComponentEditorHelper Function

Pascal

```
function FindWebComponentEditorHelper(AClass: TClass): TObject;
```

C++

```
TObject * FindWebComponentEditorHelper(TClass AClass);
```

File

WebComp

Description

This is function WebComp.FindWebComponentEditorHelper.

58.3.4 WebComp.QualifyFileName Function

Pascal

```
function QualifyFileName(const AFileName: string): string;
```

C++

```
UnicodeString QualifyFileName(const UnicodeString AFileName);
```

File

WebComp

Description

This is function WebComp.QualifyFileName.

58.3.5 WebComp.RegisterWebComponents Function

Pascal

```
procedure RegisterWebComponents(const AClasses: array of TComponentClass; Editor: TObject =
nil);
```

C++

```
RegisterWebComponents(const array of TComponentClass AClasses, TObject * Editor = nil);
```

File

WebComp

Description

This is function WebComp.RegisterWebComponents.

58.3.6 WebComp.UnRegisterWebComponents Function

Pascal

```
procedure UnRegisterWebComponents(const AClasses: array of TComponentClass);
```

C++

```
UnRegisterWebComponents(const array of TComponentClass AClasses);
```

File

WebComp

Description

This is function WebComp.UnRegisterWebComponents.

58.3.7 WebComp.ValidateWebParent Function

Pascal

```
procedure ValidateWebParent(AComponent: TComponent; AParent: TComponent);
```

C++

```
ValidateWebParent(TComponent * AComponent, TComponent * AParent);
```

File

WebComp

Description

This is function WebComp.ValidateWebParent.

58.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TLabelPosition (see page 6109)	This is record WebComp.TLabelPosition.
	TWebContentFlag (see page 6109)	TWebContentFlags (see page 6113) indicates the type of HTML a Web item can produce.

58.4.1 WebComp.TLabelPosition Enumeration

Pascal

```
TLabelPosition = (
  lposLeft,
  lposRight,
  lposAbove,
  lposBelow
);
```

C++

```
enum TLabelPosition {
  lposLeft,
  lposRight,
  lposAbove,
  lposBelow
};
```

File

WebComp

Description

This is record WebComp.TLabelPosition.

58.4.2 WebComp.TWebContentFlag Enumeration

TWebContentFlags (see page 6113) indicates the type of HTML a Web item can produce.

Pascal

```
TWebContentFlag = (
  coNoScript,
  coLocalPaths,
  coNoExecuteScript
);
```

C++

```
enum TWebContentFlag {
  coNoScript,
  coLocalPaths,
```

```
  coNoExecuteScript
};
```

File

WebComp

Description

TWebContentFlags (see page 6113) is a set of TWebContentFlag values. It can include zero or more of the following values:

Value	Meaning
coNoScript	The generated HTML can't contain any client-side script such as javascript.
coLocalPaths	The generated HTML can use local paths to included resources rather than requiring URLs. This option is for HTML documents that are only viewed on the same machine as the included resources.
coNoExecuteScript	The item generates its HTML in a class-specific manner, ignoring any server-side script that may be included in the application.

58.5 Types

The following table lists types in this documentation.

Types

Name	Description
TEnumWebComponentsProc (see page 6111)	This is type WebComp.TEnumWebComponentsProc.
TGetComponentListEvent (see page 6111)	TGetComponentListEvent is the type for event handlers that supply a list of child Web items.
TLayout (see page 6112)	TLayout is the type for helper objects that generate layout code for a Web item.
TSetComponentListEvent (see page 6112)	This is type WebComp.TSetComponentListEvent.
TWebComponentContainer (see page 6113)	TWebComponentContainer is the type for components that maintain a list of Web items.
TWebContentFlags (see page 6113)	This is type WebComp.TWebContentFlags.

58.5.1 WebComp.TEnumWebComponentsProc Type

Pascal

```
TEnumWebComponentsProc = procedure (WebComponentClass: TWebComponentClassDescription; Info:
Pointer) of object;
```

C++

```
(WebComponentClass: TWebComponentClassDescription; Info: Pointer) ( TEnumWebComponentsProc)();
```

File

WebComp

Description

This is type WebComp.TEnumWebComponentsProc.

58.5.2 WebComp.TGetComponentListEvent Type

TGetComponentListEvent is the type for event handlers that supply a list of child Web items.

Pascal

```
TGetComponentListEvent = procedure (Sender: TObject; var AList: TWebComponentList) of object;
```

C++

```
(Sender: TObject; var AList: TWebComponentList) ( TGetComponentListEvent)();
```

File

WebComp

Description

TGetComponentListEvent is the type for event handlers of helper objects that manage the lists of child Web items for a component in a WebSnap application.

Sender is the helper object that needs to access the list of child Web items.

AList returns the list of child Web items.

See Also

Procedural types

Creating events

58.5.3 WebComp.TLayout Type

TLayout is the type for helper objects that generate layout code for a Web item.

Pascal

```
TLayout = TObject;
```

C++

```
TObject * TLayout;
```

File

WebComp

Description

TLayout is the type for objects that Web items use to transmit common functions that control the layout of HTML controls. When a Web item generates content that includes HTML produced by child components, it passes a TLayout object to those children. The TLayout object implements an interface (ILayoutWebContent (see page 6084)) that generates HTML to lay out the controls which the child components generate.

Note: Typically, TLayout objects are descendants of TLayoutWebContent (see page 6014).

See Also

IWebContent (see page 6103)

ILayoutWebContent (see page 6084)

TLayoutWebContent (see page 6014)

58.5.4 WebComp.TSetComponentListEvent Type

Pascal

```
TSetComponentListEvent = procedure (Sender: TObject; AList: TWebComponentList) of object;
```

C++

```
(Sender: TObject; AList: TWebComponentList) ( TSetComponentListEvent)();
```

File

WebComp

Description

This is type WebComp.TSetComponentListEvent.

58.5.5 WebComp.TWebComponentContainer Type

TWebComponentContainer is the type for components that maintain a list of Web items.

Pascal

```
TWebComponentContainer = TObject;
```

C++

```
TObject * TWebComponentContainer;
```

File

WebComp

Description

TWebComponentContainer is the type for objects that make use of Web items (components that generate HTML and support the IWebContent (see page 6103) interface). Each TWebComponentContainer has a property of type TWebComponentList (see page 6025) that lists its Web items. The TWebComponentContainer uses these Web items to generate constituent parts of the HTML that it generates.

See Also

TWebComponentList (see page 6025)

IWebContent (see page 6103)

58.5.6 WebComp.TWebContentFlags Type

Pascal

```
TWebContentFlags = set of TWebContentFlag;
```

C++

```
TWebContentFlag TWebContentFlags;
```

File

WebComp

Description

This is type WebComp.TWebContentFlags.

58.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
EnumRegisteredWebComponentsProc (see page 6114)	This is constant WebComp.EnumRegisteredWebComponentsProc.
FindWebComponentEditorHelperProc (see page 6114)	This is constant WebComp.FindWebComponentEditorHelperProc.
RegisterWebComponentsProc (see page 6115)	This is constant WebComp.RegisterWebComponentsProc.
UnRegisterWebComponentsProc (see page 6115)	This is constant WebComp.UnRegisterWebComponentsProc.

58.6.1 WebComp.EnumRegisteredWebComponentsProc Constant

Pascal

```
EnumRegisteredWebComponentsProc: procedure (Proc: TEnumWebComponentsProc; Info: Pointer) = nil;
```

C++

```
procedure (Proc: TEnumWebComponentsProc; Info: Pointer) EnumRegisteredWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.EnumRegisteredWebComponentsProc.

58.6.2 WebComp.FindWebComponentEditorHelperProc Constant

Pascal

```
FindWebComponentEditorHelperProc: function (AClass: TClass): TObject = nil;
```

C++

```
function (AClass: TClass): TObject FindWebComponentEditorHelperProc = nil;
```

File

WebComp

Description

This is constant WebComp.FindWebComponentEditorHelperProc.

58.6.3 WebComp.RegisterWebComponentsProc Constant

Pascal

```
RegisterWebComponentsProc: procedure ( const AClasses: array of TComponentClass; Editor:
TObject) = nil;
```

C++

```
procedure ( const AClasses: array of TComponentClass; Editor: TObject)
RegisterWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.RegisterWebComponentsProc.

58.6.4 WebComp.UnRegisterWebComponentsProc Constant

Pascal

```
UnRegisterWebComponentsProc: procedure (const AClasses: array of TComponentClass) = nil;
```

C++

```
procedure (const AClasses: array of TComponentClass) UnRegisterWebComponentsProc = nil;
```

File

WebComp

Description

This is constant WebComp.UnRegisterWebComponentsProc.

59 WebContnrs Namespace

This is namespace WebContnrs.

Topics

Name	Description
Classes (see page 6118)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6140)	The following table lists structs, records, enums in this documentation.
Types (see page 6141)	The following table lists types in this documentation.

Classes

	Name	Description
	ENamedVariantsError (see page 6118)	ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 6134) object.
	TAbstractNamedVariants (see page 6118)	TAbstractNamedVariants is the base class for persistent lists of Variant (see page 3421) values.
	TNamedVariantsList (see page 6134)	TNamedVariantsList is a persistent list of name/Variant (see page 3421) pairs.

Structs, Records, Enums

	Name	Description
	TNamedVariantItem (see page 6140)	This is record WebContnrs.TNamedVariantItem.

Types

Name	Description
PNamedVariantItem (see page 6141)	This is type WebContnrs.PNamedVariantItem.
PNamedVariantList (see page 6141)	This is type WebContnrs.PNamedVariantList.
TNamedVariantList (see page 6141)	This is type WebContnrs.TNamedVariantList.

59.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ENamedVariantsError (see page 6118)	ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 6134) object.
	TAbstractNamedVariants (see page 6118)	TAbstractNamedVariants is the base class for persistent lists of Variant (see page 3421) values.
	TNamedVariantsList (see page 6134)	TNamedVariantsList is a persistent list of name/Variant (see page 3421) pairs.

59.1.1 ENamedVariantsError Class

ENamedVariantsError is the exception for errors encountered by a TNamedVariantsList (see page 6134) object.

Class Hierarchy

Pascal

```
ENamedVariantsError = class(Exception);
```

C++

```
class ENamedVariantsError : public Exception;
```

File

WebContnrs

Description

ENamedVariantsError is raised when an error occurs in TNamedVariantsList (see page 6134). For example, when an application supplies an invalid index to one of the TNamedVariantsList (see page 6134) methods, it raises an ENamedVariantsError exception.

See Also

TNamedVariantsList (see page 6134)

59.1.2 TAbstractNamedVariants Class

TAbstractNamedVariants is the base class for persistent lists of Variant (see page 3421) values.

Class Hierarchy

Pascal

```
TAbstractNamedVariants = class(TPersistent);
```

C++

```
class TAbstractNamedVariants : public TPersistent;
```

File

WebContnrs

Description

Use TAbstractNamedVariants as a base class for objects that represents a set of name/value pairs where the values are all Variants (see page 6133). TAbstractNamedVariants is a descendant of TPersistent, which means that if it is the type of a published property on a component, the named variants list is saved with the component.

Do not create instances of TAbstractNamedVariants: This class contains abstract or, in C++ terminology, pure virtual methods. Instead, use a descendant of TAbstractNamedVariants such as TNamedVariantsList (see page 6134). TNamedVariantsList (see page 6134) provides an implementation for the abstract or pure virtual methods introduced in TAbstractNamedVariants.

See Also

TStrings

TCollection

TList

TNamedVariantsList (see page 6134)

59.1.2.1 TAbstractNamedVariants Members

The following tables list the members exposed by TAbstractNamedVariants.

Protected Methods

	Name	Description
	CompareStrings (see page 6123)	Compares two strings.
	DefineProperties (see page 6124)	Allows the list of names and their corresponding Variant (see page 3421) values to be saved with the named Variants (see page 6133) list.
	Error (see page 6126)	Raises an ENamedVariantsError (see page 6118) exception.
	Get (see page 6126)	Retrieves the name and Variant (see page 3421) stored at a specified position in the list.
	GetCapacity (see page 6127)	Returns the value of the Capacity (see page 6131) property.
	GetCount (see page 6127)	Returns the value of the Count (see page 6131) property.
	Put (see page 6129)	Stores a name/Variant (see page 3421) pair at a specified position in the list.
	SetCapacity (see page 6130)	Sets the value of the Capacity (see page 6131) property.
	SetUpdateState (see page 6130)	Responds when an update operation begins or ends.

Protected Properties

	Name	Description
	UpdateCount (see page 6132)	Indicates the number of calls to BeginUpdate (see page 6122) that have not been matched by a corresponding call to EndUpdate (see page 6125).

Public Methods

	Name	Description
	Add (see page 6121)	Adds a named Variant (see page 3421) to the end of the list.
	Append (see page 6122)	Adds a named Variant (see page 3421) to the end of the list.
	Assign (see page 6122)	Copies the list of named Variants (see page 6133) from another object.
	BeginUpdate (see page 6122)	Signals the start of an update operation.
	Clear (see page 6123)	Empties the list of all its name/Variant (see page 3421) pairs.
	Delete (see page 6124)	Deletes an entry from the list.

	Destroy (see page 6120)	Disposes of an object instance.
	EndUpdate (see page 6125)	Signals the end of an update operation.
	Equals (see page 6125)	Compares this named Variants (see page 6133) list with another one.
	Exchange (see page 6126)	Swaps the positions of two items in the list.
	IndexOfName (see page 6128)	Returns the index of the first item in the list with a specified name.
	Insert (see page 6128)	Inserts a name/Variant (see page 3421) pair at a specified position.
	Move (see page 6129)	Moves a name/Variant (see page 3421) pair to a new position in the list.

Public Properties

	Name	Description
	Capacity (see page 6131)	Specifies the allocated size of the named variants list.
	Count (see page 6131)	Indicates the number of name/value pairs in the list.
	Names (see page 6132)	Lists the names of all the Variants (see page 6133) in the list.
	Values (see page 6133)	Lists the Variants (see page 6133) in the list by name.
	Variants (see page 6133)	Lists the Variants in the list by position.

59.1.2.2 TAbstractNamedVariants.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractNamedVariants();
```

Description

Do not call ~Destroy directly. Instead, use the delete keyword when destroying VCL objects.

~Destroy deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release (see page 3641) method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

59.1.2.3 TAbstractNamedVariants Methods

The methods of the TAbstractNamedVariants class are listed here.

Protected Methods

	Name	Description
	CompareStrings (see page 6123)	Compares two strings.

	DefineProperties (see page 6124)	Allows the list of names and their corresponding Variant (see page 3421) values to be saved with the named Variants (see page 6133) list.
	Error (see page 6126)	Raises an ENamedVariantsError (see page 6118) exception.
	Get (see page 6126)	Retrieves the name and Variant (see page 3421) stored at a specified position in the list.
	GetCapacity (see page 6127)	Returns the value of the Capacity (see page 6131) property.
	GetCount (see page 6127)	Returns the value of the Count (see page 6131) property.
	Put (see page 6129)	Stores a name/Variant (see page 3421) pair at a specified position in the list.
	SetCapacity (see page 6130)	Sets the value of the Capacity (see page 6131) property.
	SetUpdateState (see page 6130)	Responds when an update operation begins or ends.

Public Methods

	Name	Description
	Add (see page 6121)	Adds a named Variant (see page 3421) to the end of the list.
	Append (see page 6122)	Adds a named Variant (see page 3421) to the end of the list.
	Assign (see page 6122)	Copies the list of named Variants (see page 6133) from another object.
	BeginUpdate (see page 6122)	Signals the start of an update operation.
	Clear (see page 6123)	Empties the list of all its name/Variant (see page 3421) pairs.
	Delete (see page 6124)	Deletes an entry from the list.
	EndUpdate (see page 6125)	Signals the end of an update operation.
	Equals (see page 6125)	Compares this named Variants (see page 6133) list with another one.
	Exchange (see page 6126)	Swaps the positions of two items in the list.
	IndexOfName (see page 6128)	Returns the index of the first item in the list with a specified name.
	Insert (see page 6128)	Inserts a name/Variant (see page 3421) pair at a specified position.
	Move (see page 6129)	Moves a name/Variant (see page 3421) pair to a new position in the list.

59.1.2.3.1 TAbstractNamedVariants.Add Method

Adds a named Variant (see page 3421) to the end of the list.

Pascal

```
function Add(const S: string; const AValue: Variant): Integer; virtual;
```

C++

```
virtual __fastcall int Add(const UnicodeString S, const Variant AValue);
```

Description

Call Add to add a name/Variant (see page 3421) pair to the end of the list. Add adds a new name/Variant (see page 3421) pair to the list with a position index equal to the value of the Count (see page 6131) property.

S is the name of the Variant (see page 3421) value. This can be used to look up the value using the Values (see page 6133) property.

AValue is the Variant (see page 3421) value to associate with S.

Note: Unlike setting the Values (see page 6133) property, calling Add adds a duplicate name/Variant (see page 3421) pair to the list if S matches the name of a Variant (see page 3421) that is already in the list.

See Also

Values (see page 6133)

Append (see page 6122)

Insert (see page 6128)

Delete (see page 6124)

Count (see page 6131)

59.1.2.3.2 TAbstractNamedVariants.Append Method

Adds a named Variant (see page 3421) to the end of the list.

Pascal

```
procedure Append(const S: string; const AValue: Variant);
```

C++

```
__fastcall Append(const UnicodeString S, const Variant AValue);
```

Description

Call Append to add a name/Variant (see page 3421) pair to the end of the list. Append simply calls the Add (see page 6121) method to add the specified name and its associated Variant (see page 3421) value.

S is the name of the Variant (see page 3421) value. This can be used to look up the value using the Values (see page 6133) property.

AValue is the Variant (see page 3421) value to associate with S.

See Also

Values (see page 6133)

Add (see page 6121)

Insert (see page 6128)

Delete (see page 6124)

59.1.2.3.3 TAbstractNamedVariants.Assign Method

Copies the list of named Variants (see page 6133) from another object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Call Assign to copy another list to this named Variants (see page 6133) list. If the source object is another named Variants (see page 6133) list, Assign discards any name/Variant (see page 3421) pairs currently in the list and replaces them with the name/Variant (see page 3421) pairs in the source list. If the source is any other type of object, Assign calls the inherited method, which copies the list from any object that implements the AssignTo method so that it copies its items to a named Variants (see page 6133) list.

Source is the object from which to copy the list.

See Also

Values (see page 6133)

AssignTo

59.1.2.3.4 TAbstractNamedVariants.BeginUpdate Method

Signals the start of an update operation.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before starting an operation that makes multiple changes to the named Variants (see page 6133) list. After completing all of the changes, call the EndUpdate (see page 6125) method to signal the end of the operation.

Every call to BeginUpdate must be matched by a corresponding call to the EndUpdate (see page 6125) method.

Note: TAbstractNamedVariants (see page 6118) calls BeginUpdate and EndUpdate (see page 6125) internally when performing operations that affect multiple entries in the list.

See Also

EndUpdate (see page 6125)

59.1.2.3.5 TAbstractNamedVariants.Clear Method

Empties the list of all its name/Variant (see page 3421) pairs.

Pascal

```
procedure Clear; virtual; abstract;
```

C++

```
virtual __fastcall Clear() = 0;
```

Description

Call Clear to empty the named Variants (see page 6133) list. After a call to Clear, the value of the Count (see page 6131) property is zero.

TAbstractNamedVariants (see page 6118) declares Clear as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Count (see page 6131)

Delete (see page 6124)

59.1.2.3.6 TAbstractNamedVariants.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: string; const S2: string): Integer; virtual;
```

C++

```
virtual __fastcall int CompareStrings(const UnicodeString S1, const UnicodeString S2);
```

Description

TAbstractNamedVariants (see page 6118) calls CompareStrings internally to locate names in the list (for example, in the implementation of the Values (see page 6133) property). Descendant classes can use this method to sort the list entries by name.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 = S2, and a value greater than 0 if S1 > S2.

In TAbstractNamedVariants (see page 6118), CompareStrings calls the AnsiCompareText (see page 4051) function. This performs a locale-based case-insensitive comparison. Override this method in a descendant class to change the way the named Variants (see page 6133) list compares names. For example, by changing CompareStrings to a case sensitive function, the Values (see page 6133) property would distinguish two names that differed only in case.

See Also

AnsiCompareText (see page 4051)

Values (see page 6133)

Equals (see page 6125)

59.1.2.3.7 TAbstractNamedVariants.DefineProperties Method

Allows the list of names and their corresponding Variant (see page 3421) values to be saved with the named Variants (see page 6133) list.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

The component streaming system calls DefineProperties when it loads or saves the named Variants (see page 6133) list. TAbstractNamedVariants (see page 6118) overrides the inherited method to specify how to load and save the list of names and Variant (see page 3421) values.

Filer is the current TReader or TWriter object that is loading or saving the named Variants (see page 6133) list.

See Also

Values (see page 6133)

DefineProperties

TReader

TWriter

59.1.2.3.8 TAbstractNamedVariants.Delete Method

Deletes an entry from the list.

Pascal

```
procedure Delete(Index: Integer); virtual; abstract;
```

C++

```
virtual __fastcall Delete(int Index) = 0;
```

Description

Call Delete to remove a specified name/Variant (see page 3421) pair from the list. After a call to Delete, any name/Variant (see page 3421) pairs that appeared after the deleted item in the list move up by one (getting a new index).

Index is the position of the item to delete.

TAbstractNamedVariants (see page 6118) declares Delete as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6132)

Variants (see page 6133)

Clear (see page 6123)

59.1.2.3.9 TAbstractNamedVariants.EndUpdate Method

Signals the end of an update operation.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after completing an operation that was preceded by a call to the BeginUpdate (see page 6122) method. Every call to BeginUpdate (see page 6122) must be matched by a corresponding call to the EndUpdate method. Calls to BeginUpdate (see page 6122) and EndUpdate can be nested.

Note: TAbstractNamedVariants (see page 6118) calls BeginUpdate (see page 6122) and EndUpdate internally when performing operations that affect multiple entries in the list.

See Also

BeginUpdate (see page 6122)

59.1.2.3.10 TAbstractNamedVariants.Equals Method

Compares this named Variants (see page 6133) list with another one.

Pascal

```
function Equals(ANamedVariants: TAbstractNamedVariants): Boolean; reintroduce;
```

C++

```
__fastcall Boolean Equals(TAbstractNamedVariants ANamedVariants);
```

Description

Call Equals to determine whether another named Variants (see page 6133) list contains the same name/Variant (see page 3421) pairs as this one.

ANamedVariants is the named Variants (see page 6133) list to compare to this one.

Equals returns true if every entry in ANamedVariants has the same name and Variant (see page 3421) value as the corresponding entry in this list. Equals returns false if ANamedVariants has a different number of entries, if its entries appear in a different order, or if it contains different entries.

Equals considers two name to be the same if they differ only by case.

59.1.2.3.11 TAbstractNamedVariants.Error Method

Raises an ENamedVariantsError (see page 6118) exception.

Pascal

```
procedure Error(const Msg: string; Data: Integer); overload;
```

C++

```
__fastcall Error(const UnicodeString Msg, int Data);
```

Description

TAbstractNamedVariants (see page 6118) calls Error internally when it encounters an invalid index. For example, if an application tries to delete or change a name/Variant (see page 3421) pair and supplies an index that is greater than Count (see page 6131), TAbstractNamedVariants (see page 6118) calls Error to generate the exception about the out-of-bounds index.

Msg is the error message for the exception. This message contains a format specifier for the integer that represents the index that caused the problem.

Data is the index that caused the problem.

See Also

ENamedVariantsError (see page 6118)

Count (see page 6131)

59.1.2.3.12 TAbstractNamedVariants.Exchange Method

Swaps the positions of two items in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); virtual;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to interchange the positions of two name/Variant (see page 3421) pairs.

Index1 and Index2 are the indexes of the two entries to swap. Exchange copies the name and Variant (see page 3421) stored at Index1 to the position specified by Index2, and the name and Variant (see page 3421) from Index2 to the position specified by Index1.

See Also

Move (see page 6129)

59.1.2.3.13 TAbstractNamedVariants.Get Method

Retrieves the name and Variant (see page 3421) stored at a specified position in the list.

Pascal

```
function Get(Index: Integer; out AName: string; out AValue: Variant): Boolean; virtual;
abstract;
```

C++

```
virtual __fastcall Boolean Get(int Index, UnicodeString AName, Variant AValue) = 0;
```

Description

Get provides the underlying implementation of all methods that extract information from the named Variants (see page 6133) list based on item position. For example, the property read methods for the Names (see page 6132) and Variants (see page 6133) properties both call Get to extract the relevant information from the list.

Index is the index of the name and Variant (see page 3421) to read.

AName returns the name stored at the specified position.

AValue returns the Variant (see page 3421) value stored at the specified position.

Get returns true if Index is the index of an item in the list, false if there is no name/Variant (see page 3421) pair at the specified position.

TAbstractNamedVariants (see page 6118) declares Get as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6132)

Variants (see page 6133)

Put (see page 6129)

59.1.2.3.14 TAbstractNamedVariants.GetCapacity Method

Returns the value of the Capacity (see page 6131) property.

Pascal

```
function GetCapacity: Integer; virtual;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity (see page 6131) property.

In TAbstractNamedVariants (see page 6118), GetCapacity always returns the value of the Count (see page 6131) property, because TAbstractNamedVariants (see page 6118) does not implement support for preallocating memory to hold name/Variant (see page 3421) pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override GetCapacity to reflect the capacity of the list.

See Also

Capacity (see page 6131)

Count (see page 6131)

59.1.2.3.15 TAbstractNamedVariants.GetCount Method

Returns the value of the Count (see page 6131) property.

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCapacity (see page 6127) is the protected read implementation of the Count (see page 6131) property.

TAbstractNamedVariants (see page 6118) declares GetCount as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation for the Count (see page 6131) property.

See Also

Count (see page 6131)

59.1.2.3.16 TAbstractNamedVariants.IndexOfName Method

Returns the index of the first item in the list with a specified name.

Pascal

```
function IndexOfName(const Name: string): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfName(const UnicodeString Name);
```

Description

Call IndexOfName to determine where a name appears in the list, if it appears at all.

Name is the name to locate in the list.

If Name appears as the name part of a name/Variant (see page 3421) pair in the list, IndexOfName returns the index of the first item with that name. If there are no items with the specified name, IndexOfName returns –1.

Note: In TAbstractNamedVariants (see page 6118), IndexOfName uses a case-insensitive comparison to locate the name. In descendant classes that treat strings case sensitively, IndexOfName makes a case sensitive comparison.

See Also

Names (see page 6132)

59.1.2.3.17 TAbstractNamedVariants.Insert Method

Inserts a name/Variant (see page 3421) pair at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall Insert(int Index, const UnicodeString S, const Variant AValue) = 0;
```

Description

Call Insert to add a name/Variant (see page 3421) pair to the list at a specified position. After a call to Insert, any name/Variant (see page 3421) pairs that appeared at or after the specified position move down by one (getting a new index).

Index is the position where the new name and Variant (see page 3421) should be added.

S is the name of the new entry.

AValue is the Variant (see page 3421) value of the new entry.

TAbstractNamedVariants (see page 6118) declares Insert as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Values (see page 6133)

Append (see page 6122)

Add (see page 6121)

Delete (see page 6124)

Count (see page 6131)

59.1.2.3.18 TAbstractNamedVariants.Move Method

Moves a name/Variant (see page 3421) pair to a new position in the list.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer); virtual;
```

C++

```
virtual __fastcall Move(int CurIndex, int NewIndex);
```

Description

Call move to reposition an item in the list. Move removes the item from its current position and inserts it in the new position, readjusting the positions (indexes) of other items as necessary.

CurIndex is the current position of the item to move.

NewIndex is the position to which the item is moved.

See Also

Exchange (see page 6126)

59.1.2.3.19 TAbstractNamedVariants.Put Method

Stores a name/Variant (see page 3421) pair at a specified position in the list.

Pascal

```
procedure Put(Index: Integer; const AName: string; const AValue: Variant); virtual; abstract;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString AName, const Variant AValue) = 0;
```

Description

Put provides the underlying implementation of all methods that change the information stored in the named Variants (see page 6133) list based on item position. For example, the property write methods for the Names (see page 6132) and Variants (see page 6133) properties both call Put to assign new values to items in the list.

Index is the index of the name and Variant (see page 3421) to write.

AName is the name to store at the specified position.

AValue is the Variant (see page 3421) value to store at the specified position.

TAbstractNamedVariants (see page 6118) declares Put as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6132)

Variants (see page 6133)

Get (see page 6126)

59.1.2.3.20 TAbstractNamedVariants.SetCapacity Method

Sets the value of the Capacity (see page 6131) property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); virtual;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected write implementation of the Capacity (see page 6131) property.

In TAbstractNamedVariants (see page 6118), SetCapacity does nothing, because TAbstractNamedVariants (see page 6118) does not implement support for preallocating memory to hold name/Variant (see page 3421) pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override SetCapacity to reallocate the memory used to hold list items.

See Also

Capacity (see page 6131)

GetCapacity (see page 6127)

59.1.2.3.21 TAbstractNamedVariants.SetUpdateState Method

Responds when an update operation begins or ends.

Pascal

```
procedure SetUpdateState(Updating: Boolean); virtual;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

The BeginUpdate (see page 6122) and EndUpdate (see page 6125) methods call SetUpdate state when an update operation first starts or finally stops. If the calls to BeginUpdate (see page 6122) and EndUpdate (see page 6125) are nested, SetUpdate is only called by the first BeginUpdate (see page 6122) call and the last EndUpdate (see page 6125) call.

In TAbstractNamedVariants (see page 6118), SetUpdateState does nothing. Descendant classes can override this method to respond in some way (such as enabling or suppressing repaints of a control that displays the values in the list) when an update operation begins or ends.

See Also

BeginUpdate (see page 6122)

EndUpdate (see page 6125)

59.1.2.4 TAbstractNamedVariants Properties

The properties of the TAbstractNamedVariants class are listed here.

Protected Properties

	Name	Description
	UpdateCount (see page 6132)	Indicates the number of calls to BeginUpdate (see page 6122) that have not been matched by a corresponding call to EndUpdate (see page 6125).

Public Properties

	Name	Description
	Capacity (see page 6131)	Specifies the allocated size of the named variants list.
	Count (see page 6131)	Indicates the number of name/value pairs in the list.
	Names (see page 6132)	Lists the names of all the Variants (see page 6133) in the list.
	Values (see page 6133)	Lists the Variants (see page 6133) in the list by name.
	Variants (see page 6133)	Lists the Variants in the list by position.

59.1.2.4.1 TAbstractNamedVariants.Capacity Property

Specifies the allocated size of the named variants list.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Set Capacity to the number of named Variants (see page 6133) the list will need to contain. When a value is added to a list that is already filled to capacity, the Capacity property is automatically increased. Setting Capacity before adding values can reduce the number of memory reallocations and thereby improve performance. In addition, in descendants that allocate memory in chunks, setting Capacity can ensure that you do not allocate more memory for the list than your application needs.

Read (see page 3634) Capacity to learn number of values the list can hold without reallocating memory. Do not confuse Capacity with the Count (see page 6131) property, which is the number of entries in the list that are in use. The value of Capacity is always greater than or equal to the value of Count (see page 6131). When Capacity is greater than Count (see page 6131), the unused memory can be reclaimed by setting Capacity to Count (see page 6131).

See Also

Count (see page 6131)

Insert (see page 6128)

59.1.2.4.2 TAbstractNamedVariants.Count Property

Indicates the number of name/value pairs in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3634) Count to determine the number of values the list contains. Count acts as an upper bound to the index that you can use for the Names (see page 6132) and Variants (see page 6133) properties.

See Also

Capacity (see page 6131)

Names (see page 6132)

Variants (see page 6133)

59.1.2.4.3 TAbstractNamedVariants.Names Property

Lists the names of all the Variants (see page 6133) in the list.

Pascal

```
property Names [Index: Integer]: string;
```

C++

```
__property UnicodeString Names[int Index];
```

Description

Read (see page 3634) Names to obtain the name of a Variant (see page 3421) in the list.

Index is the position of the name (and its corresponding Variant (see page 3421)) in the named Variants (see page 6133) list. The value of Index falls in the range from 0 to Count (see page 6131) – 1.

See Also

Count (see page 6131)

Values (see page 6133)

Variants (see page 6133)

59.1.2.4.4 TAbstractNamedVariants.UpdateCount Property

Indicates the number of calls to BeginUpdate (see page 6122) that have not been matched by a corresponding call to EndUpdate (see page 6125).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether the named Variants (see page 6133) list is in the middle of an update operation (an operation that makes several changes to the list). Every time the application calls BeginUpdate (see page 6122), the value of UpdateCount is incremented. Every call to EndUpdate (see page 6125) decrements the value of UpdateCount.

When UpdateCount changes from 0 to 1, and when it changes from 1 to 0, TAbstractNamedVariants (see page 6118) calls the SetUpdateState (see page 6130) method. This allows TAbstractNamedVariants (see page 6118) descendants to respond in some way (such as enabling or suppressing repaints of a control that displays the values in the list) when an update operation begins or ends.

See Also

BeginUpdate (see page 6122)

EndUpdate (see page 6125)

SetUpdateState (see page 6130)

59.1.2.4.5 TAbstractNamedVariants.Values Property

Lists the Variants (see page 6133) in the list by name.

Pascal

```
property Values [const Name: string]: Variant;
```

C++

```
__property Variant Values[UnicodeString const Name];
```

Description

Read (see page 3634) Values to look up the Variant (see page 3421) that is stored under a particular name. Set Values to change the Variant (see page 3421) associated with a name, or to add a new name and Variant (see page 3421) to the list.

Name is the name of the Variant (see page 3421). If there is no Variant (see page 3421) in the list with the specified name, reading Values gives an unassigned Variant (see page 3421), and setting Values adds a new name and Variant (see page 3421) to the list.

See Also

Count (see page 6131)

Variants (see page 6133)

59.1.2.4.6 TAbstractNamedVariants.Variants Property

Lists the Variants in the list by position.

Pascal

```
property Variants [Index: Integer]: Variant;
```

C++

```
__property Variant Variants[int Index];
```

Description

Read (see page 3634) Variants to look up the Variant (see page 3421) that is stored at a particular position. Set Variants to change the Variant (see page 3421) stored in a particular position.

Index is the position of the Variant (see page 3421) in the named Variants list. The value of Index falls in the range from 0 to Count (see page 6131) – 1. If you specify a value of Index that is out of range, TAbstractNamedVariants (see page 6118) raises an ENamedVariantsError (see page 6118) exception.

See Also

Count (see page 6131)

Values (see page 6133)

Names (see page 6132)

ENamedVariantsError (see page 6118)

59.1.3 TNamedVariantsList Class

TNamedVariantsList is a persistent list of name/Variant (see page 3421) pairs.

Class Hierarchy

Pascal

```
TNamedVariantsList = class(TAbstractNamedVariants);
```

C++

```
class TNamedVariantsList : public TAbstractNamedVariants;
```

File

WebContnrs

Description

Use TNamedVariantsList to represent a set of name/value pairs where the values are all Variants (see page 4973). Because TNamedVariantsList is a descendant of TPersistent, it can be used as the type of a published property on a component.

TNamedVariantsList has properties and methods that let you:

Look up Variant (see page 3421) values by name or by index.

Locate a name in the list.

Add (see page 6121) or delete name/Variant (see page 3421) pairs.

Rearrange the items in the list.

Compare this list to another named variants list.

Note: TNamedVariantsList treats the names it stores in a case-insensitive manner. When locating a name in the list or assigning a Variant (see page 3421) value to an existing name, TNamedVariantsList treats names that differ only by case as the same.

See Also

TStringList

THashedStringList

TCollection

TList

TInterfaceList

59.1.3.1 TNamedVariantsList Members

The following tables list the members exposed by TNamedVariantsList.

Protected Methods

	Name	Description
	CompareStrings (see page 6123)	Compares two strings.

	DefineProperties (see page 6124)	Allows the list of names and their corresponding Variant (see page 3421) values to be saved with the named Variants (see page 6133) list.
	Error (see page 6126)	Raises an ENamedVariantsError (see page 6118) exception.
	Get (see page 6137)	Retrieves the name and Variant stored at a specified position in the list.
	GetCapacity (see page 6137)	Returns the value of the Capacity property.
	GetCount (see page 6138)	Returns the value of the Count property.
	Put (see page 6139)	Stores a name/Variant pair at a specified position in the list.
	SetCapacity (see page 6139)	Sets the value of the Capacity property.
	SetUpdateState (see page 6130)	Responds when an update operation begins or ends.

Protected Properties

	Name	Description
	UpdateCount (see page 6132)	Indicates the number of calls to BeginUpdate (see page 6122) that have not been matched by a corresponding call to EndUpdate (see page 6125).

Public Methods

	Name	Description
	Add (see page 6121)	Adds a named Variant (see page 3421) to the end of the list.
	Append (see page 6122)	Adds a named Variant (see page 3421) to the end of the list.
	Assign (see page 6122)	Copies the list of named Variants (see page 6133) from another object.
	BeginUpdate (see page 6122)	Signals the start of an update operation.
	Clear (see page 6136)	Deletes all the name/Variant (see page 3421) pairs from the list.
	Delete (see page 6136)	Removes a specified Variant (see page 3421) specified by the Index parameter.
	Destroy (see page 6135)	Frees an instance of TNamedVariantsList (see page 6134).
	EndUpdate (see page 6125)	Signals the end of an update operation.
	Equals (see page 6125)	Compares this named Variants (see page 6133) list with another one.
	Exchange (see page 6126)	Swaps the positions of two items in the list.
	IndexOfName (see page 6128)	Returns the index of the first item in the list with a specified name.
	Insert (see page 6138)	Inserts a name/Variant (see page 3421) pair at a specified position.
	Move (see page 6129)	Moves a name/Variant (see page 3421) pair to a new position in the list.

Public Properties

	Name	Description
	Capacity (see page 6131)	Specifies the allocated size of the named variants list.
	Count (see page 6131)	Indicates the number of name/value pairs in the list.
	Names (see page 6132)	Lists the names of all the Variants (see page 6133) in the list.
	Values (see page 6133)	Lists the Variants (see page 6133) in the list by name.
	Variants (see page 6133)	Lists the Variants in the list by position.

59.1.3.2 TNamedVariantsList.Destroy Destructor

Frees an instance of TNamedVariantsList (see page 6134).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TNamedVariantsList();
```

Description

Do not call the TNamedVariantsList (see page 6134) destructor directly. Instead, use the Free method, which first checks whether the reference to the object is nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

59.1.3.3 TNamedVariantsList Methods

The methods of the TNamedVariantsList class are listed here.

Protected Methods

	Name	Description
	Get (see page 6137)	Retrieves the name and Variant stored at a specified position in the list.
	GetCapacity (see page 6137)	Returns the value of the Capacity property.
	GetCount (see page 6138)	Returns the value of the Count property.
	Put (see page 6139)	Stores a name/Variant pair at a specified position in the list.
	SetCapacity (see page 6139)	Sets the value of the Capacity property.

Public Methods

	Name	Description
	Clear (see page 6136)	Deletes all the name/Variant (see page 3421) pairs from the list.
	Delete (see page 6136)	Removes a specified Variant (see page 3421) specified by the Index parameter.
	Insert (see page 6138)	Inserts a name/Variant (see page 3421) pair at a specified position.

59.1.3.3.1 TNamedVariantsList.Clear Method

Deletes all the name/Variant (see page 3421) pairs from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call Clear to empty the list. Clear frees the memory that the Variants (see page 4973) in the list use to store their values as well as the memory used to store the list itself.

After a call to Clear, the value of the Count (see page 6131) property is zero.

See Also

Delete (see page 6136)

Variants (see page 6133)

Count (see page 6131)

59.1.3.3.2 TNamedVariantsList.Delete Method

Removes a specified Variant (see page 3421) specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a specified name/Variant (see page 3421) pair from the list. Delete frees the Variant (see page 3421) when it removes the item from the list.

After a call to Delete, any name/Variant (see page 3421) pairs that appeared after the deleted item in the list move up by one (getting a new index).

Index is the position of the item to delete.

See Also

Clear (see page 6136)

IndexOfName (see page 6128)

59.1.3.3.3 TNamedVariantsList.Get Method

Retrieves the name and Variant stored at a specified position in the list.

Pascal

```
function Get(Index: Integer; out AName: string; out AValue: Variant): Boolean; override;
```

C++

```
virtual __fastcall Boolean Get(int Index, UnicodeString AName, Variant AValue);
```

Description

Get provides the underlying implementation of all methods that extract information from the named Variants list based on item position. For example, the property read methods for the Names and Variants properties both call Get to extract the relevant information from the list.

Index is the index of the name and Variant to read.

AName returns the name stored at the specified position.

AValue returns the Variant value stored at the specified position.

Get returns true if Index is the index of an item in the list, false if there is no name/Variant pair at the specified position.

TAbstractNamedVariants declares Get as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6132)

Variants (see page 6133)

Put (see page 6129)

59.1.3.3.4 TNamedVariantsList.GetCapacity Method

Returns the value of the Capacity property.

Pascal

```
function GetCapacity: Integer; override;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity property.

In TAbstractNamedVariants, GetCapacity always returns the value of the Count property, because TAbstractNamedVariants does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override GetCapacity to reflect the capacity of the list.

See Also

Capacity (see page 6131)

Count (see page 6131)

59.1.3.3.5 TNamedVariantsList.GetCount Method

Returns the value of the Count property.

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCapacity is the protected read implementation of the Count property.

TAbstractNamedVariants declares GetCount as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation for the Count property.

See Also

Count (see page 6131)

59.1.3.3.6 TNamedVariantsList.Insert Method

Inserts a name/Variant (see page 3421) pair at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall Insert(int Index, const UnicodeString S, const Variant AValue);
```

Description

Call Insert to add a name/Variant (see page 3421) pair to the list at a specified position. After a call to Insert, any name/Variant (see page 3421) pairs that appeared at or after the specified position move down by one (getting a new index).

Index is the position where the new name and Variant (see page 3421) should be added.

S is the name of the new entry.

AValue is the Variant (see page 3421) value of the new entry.

See Also

Values (see page 6133)

Append (see page 6122)

Add (see page 6121)

Delete (see page 6136)

Count (see page 6131)

59.1.3.3.7 TNamedVariantsList.Put Method

Stores a name/Variant pair at a specified position in the list.

Pascal

```
procedure Put(Index: Integer; const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall Put(int Index, const UnicodeString AName, const Variant AValue);
```

Description

Put provides the underlying implementation of all methods that change the information stored in the named Variants list based on item position. For example, the property write methods for the Names and Variants properties both call Put to assign new values to items in the list.

Index is the index of the name and Variant to write.

AName is the name to store at the specified position.

AValue is the Variant value to store at the specified position.

TAbstractNamedVariants declares Put as an abstract or (in C++ terminology) a pure virtual method, meaning that it provides no implementation. Descendant classes must override this method to provide an implementation.

See Also

Names (see page 6132)

Variants (see page 6133)

Get (see page 6126)

59.1.3.3.8 TNamedVariantsList.SetCapacity Method

Sets the value of the Capacity property.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); override;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected write implementation of the Capacity property.

In TAbstractNamedVariants, SetCapacity does nothing, because TAbstractNamedVariants does not implement support for preallocating memory to hold name/Variant pairs that have not been added to the list. Descendant classes that provide support for allocating memory before adding items to the list override SetCapacity to reallocate the memory used to hold list items.

See Also

Capacity (see page 6131)

GetCapacity (see page 6127)

59.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TNamedVariantItem (see page 6140)	This is record WebContnrs.TNamedVariantItem.

59.2.1 WebContnrs.TNamedVariantItem Record

Pascal

```
TNamedVariantItem = record
  FString: string;
  FVariant: Variant;
end;
```

C++

```
struct TNamedVariantItem {
  UnicodeString FString;
  Variant FVariant;
};
```

File

WebContnrs

Description

This is record WebContnrs.TNamedVariantItem.

59.3 Types

The following table lists types in this documentation.

Types

Name	Description
PNamedVariantItem (see page 6141)	This is type WebContnrs.PNamedVariantItem.
PNamedVariantList (see page 6141)	This is type WebContnrs.PNamedVariantList.
TNamedVariantList (see page 6141)	This is type WebContnrs.TNamedVariantList.

59.3.1 WebContnrs.PNamedVariantItem Type

Pascal

```
PNamedVariantItem = ^TNamedVariantItem;
```

C++

```
^TNamedVariantItem PNamedVariantItem;
```

File

WebContnrs

Description

This is type WebContnrs.PNamedVariantItem.

59.3.2 WebContnrs.PNamedVariantList Type

Pascal

```
PNamedVariantList = ^TNamedVariantList;
```

C++

```
^TNamedVariantList PNamedVariantList;
```

File

WebContnrs

Description

This is type WebContnrs.PNamedVariantList.

59.3.3 WebContnrs.TNamedVariantList Type

Pascal

```
TNamedVariantList = array[0..2000000000 div sizeof(TNamedVariantItem)] of TNamedVariantItem;
```

C++

```
array[0..2000000000 div sizeof(TNamedVariantItem)] of TNamedVariantItem TNamedVariantList;
```

File

WebContnrs

Description

This is type WebContnrs.TNamedVariantList.

60 WebDisp Namespace

This is namespace WebDisp.

Topics

Name	Description
Classes (see page 6146)	The following table lists classes in this documentation.
Interfaces (see page 6257)	The following table lists interfaces in this documentation.
Functions (see page 6259)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6265)	The following table lists structs, records, enums in this documentation.
Types (see page 6266)	The following table lists types in this documentation.
Constants (see page 6277)	The following table lists constants in this documentation.

Classes

	Name	Description
	EAdapterDispatcherException (see page 6146)	EAdapterDispatcherException is the exception class for WebSnap actions and image requests that can't be serviced.
	EPageDispatcherException (see page 6147)	EPageDispatcherException raises an exception when a page dispatcher encounters an error.
	ESessionException (see page 6147)	ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.
	TAdapterDispatcher (see page 6148)	TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TAdapterDispatchParams (see page 6153)	TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.
	TAdapterItemRequestParams (see page 6160)	TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.
	TBaseRequestHandler (see page 6166)	The TBaseRequestHandler component is a base class for WebSnap dispatcher components.
	TCustomAdapterDispatcher (see page 6168)	TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TCustomLocateFileService (see page 6181)	TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.
	TCustomPageDispatcher (see page 6188)	Dispatches incoming HTTP request messages in a WebSnap application.
	TCustomWebAppComponents (see page 6201)	TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.
	TInvalidWebSession (see page 6216)	Serves as a placeholder when an HTTP request could not be given a session.
	TLocateFileService (see page 6220)	TLocateFileService controls the access of templates and server-side script include files at runtime.

	TMultiModuleWebAppServices (see page 6223)	TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.
	TPageDispatcher (see page 6236)	TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.
	TWebAppComponents (see page 6243)	TWebAppComponents holds the application components that provide the WebSnap application's functionality.
	TWebSession (see page 6251)	TWebSession stores temporary information associated with the user in a WebSnap application.

Constants

Name	Description
sFailurePageParamName (see page 6277)	This is constant WebDisp.sFailurePageParamName.
sPageParamName (see page 6277)	This is constant WebDisp.sPageParamName.
sRequestIDParamName (see page 6278)	This is constant WebDisp.sRequestIDParamName.
sSessionCookieName (see page 6278)	This is constant WebDisp.sSessionCookieName.
sSuccessPageParamName (see page 6278)	This is constant WebDisp.sSuccessPageParamName.

Functions

	Name	Description
	AdapterRequestParamsAsStrings (see page 6259)	This is function WebDisp.AdapterRequestParamsAsStrings.
	DispatchPageName (see page 6259)	This is function WebDisp.DispatchPageName.
	EncodeParamNameValues (see page 6260)	This is function WebDisp.EncodeParamNameValues.
	EncodeParamsAsHTTPQuery (see page 6260)	This is function WebDisp.EncodeParamsAsHTTPQuery.
	ExpandScriptException (see page 6260)	This is function WebDisp.ExpandScriptException.
	ExtractParamNameValues (see page 6261)	This is function WebDisp.ExtractParamNameValues.
	FindDefaultAppPageName (see page 6261)	This is function WebDisp.FindDefaultAppPageName.
	PageNameOfRequest (see page 6261)	This is function WebDisp.PageNameOfRequest.
	RaiseDispatchBlankPageNameException (see page 6262)	This is function WebDisp.RaiseDispatchBlankPageNameException.
	RaisePageDoesNotProvideContent (see page 6262)	This is function WebDisp.RaisePageDoesNotProvideContent.
	RaisePageNotFound (see page 6262)	This is function WebDisp.RaisePageNotFound.
	RedirectToPageName (see page 6263)	This is function WebDisp.RedirectToPageName.
	RegisterAdapterRequestIdentifier (see page 6263)	This is function WebDisp.RegisterAdapterRequestIdentifier.
	SafeRedirect (see page 6263)	Redirects a browser to another specified page.

Interfaces

	Name	Description
	IWebSessionAccess (see page 6257)	This is class WebDisp.IWebSessionAccess.

Structs, Records, Enums

	Name	Description
	TPageAccessDenied (see page 6265)	TPageAccessDenied indicates why a user was denied access to a requested Web page.

Types

Name	Description
TActivateWebModuleEvent (see page 6266)	This is type WebDisp.TActivateWebModuleEvent.
TCanViewPageEvent (see page 6267)	TCanViewPageEvent is the type for event handlers that indicate whether the current user can view a specified Web page.
TDispatchActionEvent (see page 6267)	TDispatchActionEvent is the type for event handlers that respond when an adapter action executes.
TDispatchActionHandledEvent (see page 6268)	TDispatchActionHandledEvent is the type for event handlers that can handle requests to execute an adapter action.
TDispatchImageEvent (see page 6269)	TDispatchImageEvent is the type for event handlers that respond when an adapter action gets a request for an image field value.
TDispatchImageHandledEvent (see page 6269)	TDispatchImageHandledEvent is the type for event handlers that can handle requests for an image field value.
TDispatchPageEvent (see page 6270)	TDispatchPageEvent is the type for event handlers that are called after a Web page module finishes handling an HTTP request.

TDispatchPageHandledEvent (see page 6270)	TDispatchPageHandledEvent is the type for event handlers that can handle HTTP request messages.
TDispatchPageParamsHandledEvent (see page 6271)	TDispatchPageParamsHandledEvent is the type for event handlers that are called before a WebSnap application assembles a response message redirecting the caller to another Web page.
TGetAdapterRequestParamsEvent (see page 6272)	TGetAdapterRequestParamsEvent is the type for event handlers that retrieve parameters for an adapter dispatcher.
TLocateFileServiceFindFileEvent (see page 6273)	TLocateFileServiceFindFileEvent is the type for event handlers that locates a file on disk in a WebSnap application.
TLocateFileServiceFindStreamEvent (see page 6273)	TLocateFileServiceFindStreamEvent is the type for event handlers that provide access to a requested file.
TLoginPagedHandledEvent (see page 6274)	This is type WebDisp.TLoginPagedHandledEvent.
TLoginRequiredEvent (see page 6274)	TLoginRequiredEvent is the type for event handlers that indicate whether the current user must log in before accessing a specified Web page.
TPageAccessDeniedEvent (see page 6275)	TPageAccessDeniedEvent is the type for event handlers that respond when a page dispatcher determines that the current user can't access the current page.
TWebRequest (see page 6276)	This is type WebDisp.TWebRequest.
TWebResponse (see page 6276)	This is type WebDisp.TWebResponse.

60.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EAdapterDispatcherException (see page 6146)	EAdapterDispatcherException is the exception class for WebSnap actions and image requests that can't be serviced.
	EPageDispatcherException (see page 6147)	EPageDispatcherException raises an exception when a page dispatcher encounters an error.
	ESessionException (see page 6147)	ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.
	TAdapterDispatcher (see page 6148)	TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TAdapterDispatchParams (see page 6153)	TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.
	TAdapterItemRequestParams (see page 6160)	TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.
	TBaseRequestHandler (see page 6166)	The TBaseRequestHandler component is a base class for WebSnap dispatcher components.
	TCustomAdapterDispatcher (see page 6168)	TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.
	TCustomLocateFileService (see page 6181)	TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.
	TCustomPageDispatcher (see page 6188)	Dispatches incoming HTTP request messages in a WebSnap application.
	TCustomWebAppComponents (see page 6201)	TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.
	TInvalidWebSession (see page 6216)	Serves as a placeholder when an HTTP request could not be given a session.
	TLocateFileService (see page 6220)	TLocateFileService controls the access of templates and server-side script include files at runtime.
	TMultiModuleWebAppServices (see page 6223)	TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.
	TPageDispatcher (see page 6236)	TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.
	TWebAppComponents (see page 6243)	TWebAppComponents holds the application components that provide the WebSnap application's functionality.
	TWebSession (see page 6251)	TWebSession stores temporary information associated with the user in a WebSnap application.

60.1.1 EAdapterDispatcherException Class

EAdapterDispatcherException is the exception class for WebSnap actions and image requests that can't be serviced.

Class Hierarchy

Pascal

```
EAdapterDispatcherException = class(EWebBrokerException);
```

C++

```
class EAdapterDispatcherException : public EWebBrokerException;
```

File

WebDisp

Description

When the TAdapterDispatcher (see page 6148) component can't successfully dispatch an action or obtain a dynamic image in response to a request, it generates an OnActionRequestNotHandled or an OnImageRequestNotHandled event. If there is no event handler for that event, or if the event handler does not handle the situation, TAdapterDispatcher (see page 6148) raises an EAdapterDispatcherException exception.

See Also

TCustomAdapterDispatcher (see page 6168)

60.1.2 EPageDispatcherException Class

EPageDispatcherException raises an exception when a page dispatcher encounters an error.

Class Hierarchy

Pascal

```
EPageDispatcherException = class(EWebBrokerException);
```

C++

```
class EPageDispatcherException : public EWebBrokerException;
```

File

WebDisp

Description

EPageDispatcherException represents exceptions that occur when a page dispatcher tries to obtain the content of a Web page in response to an incoming HTTP request. These exceptions include

The incoming request does not request a specific Web page and the application does not provide a default Web page to handle this situation.

The current user does not have access rights for the requested page.

The page dispatcher could not locate a Web page module to provide the requested page.

The Web module with the same name as the requested page can't produce a Web page.

The application is attempting to send a response that redirects the user to a page that the dispatcher can't locate.

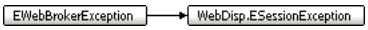
See Also

TPageDispatcher (see page 6236)

60.1.3 ESessionException Class

ESessionException is the exception that is raised when a WebSnap application tries to use an invalid session.

Class Hierarchy



Pascal

```
ESessionException = class(EWebBrokerException);
```

C++

```
class ESessionException : public EWebBrokerException;
```

File

WebDisp

Description

The TInvalidWebSession (see page 6216) class raises ESessionException when the application tries to use the current session and that session could not be created. This includes attempts to change the properties of the current session (such as setting the time-out period), attempts to get or set values, or attempts to terminate the current session.

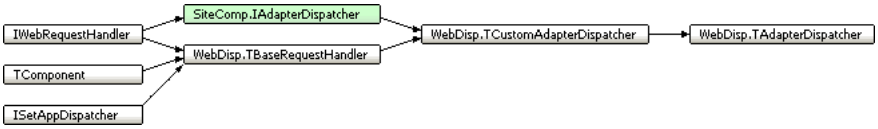
See Also

TInvalidWebSession (see page 6216)

60.1.4 TAdapterDispatcher Class

TAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.

Class Hierarchy



Pascal

```
TAdapterDispatcher = class(TCustomAdapterDispatcher);
```

C++

```
class TAdapterDispatcher : public TCustomAdapterDispatcher;
```

File

WebDisp

Description

TAdapterDispatcher is one of the default application components that is added when creating a WebSnap application. Events are triggered before and after dispatching an action or image, when adapter parameters are requested, and in cases when the action or image request was not handled.

Note: Your web application will be unable to execute adapter actions or obtain images from adapter fields unless it includes a TAdapterDispatcher component.

See Also

TAdapter (see page 5190)

TCustomWebAppComponents (see page 6201)

60.1.4.1 TAdapterDispatcher Members

The following tables list the members exposed by TAdapterDispatcher.

Protected Methods

	Name	Description
	AddSpecialParams (see page 6170)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
	DoAdapterRequestNotHandled (see page 6171)	Generates an OnActionRequestNotHandled (see page 6177) and an OnImageRequestNotHandled (see page 6180) event.
	DoAfterDispatchAdapterRequest (see page 6172)	Generates an OnAfterDispatchAction (see page 6178) and an OnAfterDispatchImage (see page 6178) event.
	DoBeforeDispatchAdapterRequest (see page 6172)	Generates an OnBeforeDispatchAction (see page 6179) and an OnBeforeDispatchImage (see page 6179) event.
	EncodeAdapterItemRequestAsFieldValue (see page 6173)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 6173)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 6174)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetDispatchParams (see page 6174)	Parses a request to determine the adapter to which it is directed.
	HandleRequest (see page 6167)	Handles an HTTP request.
	ImplEncodeAdapterItemRequestAsFieldValue (see page 6175)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6173) method.
	ImplEncodeAdapterItemRequestAsHREF (see page 6175)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6173) method.
	ImplGetAdapterItemRequestParams (see page 6176)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6174) method.
	ImplHandleRequest (see page 6167)	Handles application-specific processing of an HTTP request.
	ImplHandleRequest (see page 6176)	Provides the underlying implementation for the HandleRequest (see page 6167) method.
	SetAppDispatcher (see page 6167)	Sets the AppDispatcher (see page 6168) property.

Protected Properties

	Name	Description
	AppDispatcher (see page 6168)	Provides read-only access to the application dispatcher component.

Public Methods

	Name	Description
	EncodeAdapterItemRequestAsFieldValue (see page 986)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 986)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 987)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Published Events

	Name	Description
	OnActionRequestNotHandled (see page 6150)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6151)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 6151)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6151)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6152)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6152)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

	OnImageRequestNotHandled (see page 6153)	Occurs when the WebSnap application could not provide a requested image.

60.1.4.2 TAdapterDispatcher Events

The events of the TAdapterDispatcher class are listed here.

Published Events

	Name	Description
	OnActionRequestNotHandled (see page 6150)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6151)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 6151)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6151)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6152)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6152)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 6153)	Occurs when the WebSnap application could not provide a requested image.

60.1.4.2.1 TAdapterDispatcher.OnActionRequestNotHandled Event

Occurs when the WebSnap application could not execute a requested action.

Pascal

```
property OnActionRequestNotHandled: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnActionRequestNotHandled;
```

Description

Write an OnActionRequestNotHandled event handler to provide a response to actions that the adapter components in the application can't execute. This may occur because the adapter dispatcher could not find an adapter to handle the action, or it may occur because the identified adapter was unable to execute the action.

If the OnActionRequestNotHandled event handler does not set its Handled parameter to true, the adapter dispatcher generates an OnImageRequestNotHandled. If the OnImageRequestNotHandled event handler also does not handle the event, the dispatcher raises an EAdapterDispatcherException exception.

Tip: Use the Response property of the current Web context to access the response message from the OnActionRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchAction (see page 6178)

OnBeforeDispatchAction (see page 6179)

EAdapterDispatcherException (see page 6146)

WebContext (see page 6003)

Response (see page 5975)

OnImageRequestNotHandled (see page 6180)

60.1.4.2.2 TAdapterDispatcher.OnAfterDispatchAction Event

Occurs after the target adapter has finished executing an action.

Pascal

```
property OnAfterDispatchAction: TDispatchActionEvent;
```

C++

```
__property TDispatchActionEvent OnAfterDispatchAction;
```

Description

Write an OnAfterDispatchAction event handler to take specific action after an adapter has executed an action. Note that this event can occur even if the request does not request the execution of an action or if the adapter was not successful in its attempt to execute the action.

See Also

OnActionRequestNotHandled (see page 6177)

OnBeforeDispatchAction (see page 6179)

60.1.4.2.3 TAdapterDispatcher.OnAfterDispatchImage Event

Occurs after the target adapter has finished handling an image request.

Pascal

```
property OnAfterDispatchImage: TDispatchImageEvent;
```

C++

```
__property TDispatchImageEvent OnAfterDispatchImage;
```

Description

Write an OnAfterDispatchImage event handler to take specific action after an adapter has responded to an image request. Note that this event can occur even if the request does not request the value of an image field or if the adapter was not successful in its attempt to honor that request.

See Also

OnImageRequestNotHandled (see page 6180)

OnBeforeDispatchImage (see page 6179)

60.1.4.2.4 TAdapterDispatcher.OnBeforeDispatchAction Event

Occurs before the dispatcher forwards an action to an adapter for execution.

Pascal

```
property OnBeforeDispatchAction: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnBeforeDispatchAction;
```

Description

Write an OnBeforeDispatchAction event handler to respond to action requests before they are forwarded to the associated adapter for execution. Note that if the request does not request the execution of an action, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchAction event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Action parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for execution, and does not generate an OnAfterDispatchAction event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchAction event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnActionRequestNotHandled (see page 6177)

OnAfterDispatchAction (see page 6178)

WebContext (see page 6003)

Response (see page 5975)

60.1.4.2.5 TAdapterDispatcher.OnBeforeDispatchImage Event

Occurs before the dispatcher forwards an image request to an adapter for handling.

Pascal

```
property OnBeforeDispatchImage: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnBeforeDispatchImage;
```

Description

Write an OnBeforeDispatchImage event handler to respond to image requests before they are forwarded to the associated adapter for handling. Note that if the request does not ask for the value of an image field, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchImage event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Image parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for handling, and does not generate an OnAfterDispatchImage event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchImage event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnImageRequestNotHandled (see page 6180)

OnAfterDispatchImage (see page 6178)

WebContext (see page 6003)

Response (see page 5975)

60.1.4.2.6 TAdapterDispatcher.OnGetAdapterRequestParams Event

Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

Pascal

```
property OnGetAdapterRequestParams: TGetAdapterRequestParamsEvent;
```

C++

```
__property TGetAdapterRequestParamsEvent OnGetAdapterRequestParams;
```

Description

Write an OnGetAdapterRequestParams event handler to modify the list of parameters for an action or image request before they are used. The OnGetAdapterRequestParams event handler can modify the current parameters or add its own parameters to the list. This event occurs both when the dispatcher formats parameters that are added to HTML forms and when it is dispatching a request message that contains an action or image request.

See Also

GetAdapterItemRequestParams (see page 6174)

EncodeAdapterItemRequestAsFieldValue (see page 6173)

EncodeAdapterItemRequestAsHREF (see page 6173)

AddSpecialParams (see page 6170)

60.1.4.2.7 TAdapterDispatcher.OnImageRequestNotHandled Event

Occurs when the WebSnap application could not provide a requested image.

Pascal

```
property OnImageRequestNotHandled: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnImageRequestNotHandled;
```

Description

Write an OnImageRequestNotHandled event handler to provide a response to image requests that the adapter components in the application can't honor. This may occur because the adapter dispatcher could not find an adapter to handle the request, or it may occur because the identified adapter was unable to provide the requested image.

If the OnImageRequestNotHandled event handler exits with its Handled parameter set to false, the dispatcher raises an EAdapterDispatcherException exception.

Tip: Use the Response property of the current Web context to access the response message from the OnImageRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext function.

See Also

OnAfterDispatchImage (see page 6178)

OnBeforeDispatchImage (see page 6179)

EAdapterDispatcherException (see page 6146)

WebContext (see page 6003)

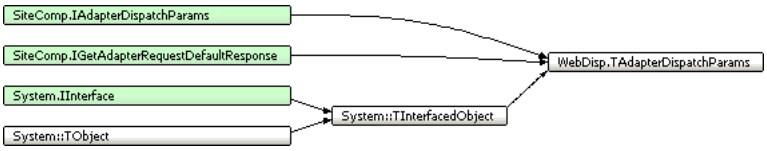
Response (see page 5975)

OnActionRequestNotHandled (see page 6177)

60.1.5 TAdapterDispatchParams Class

TAdapterDispatchParams parses a request to determine the adapter to which it is directed, and, if available, the Web pages that should appear when the request is successfully or unsuccessfully handled.

Class Hierarchy



Pascal

```
TAdapterDispatchParams = class(TInterfacedObject, IAdapterDispatchParams,
IGetAdapterRequestDefaultResponse);
```

C++

```
class TAdapterDispatchParams : public TInterfacedObject, public IAdapterDispatchParams, public
IGetAdapterRequestDefaultResponse;
```

File

WebDisp

Description

TAdapterDispatcher (see page 6148) uses TAdapterDispatchParams as a helper object for locating the adapter that can handle a request message. The TAdapterDispatchParams constructor takes a request message as an argument, and parses the request to determine the adapter, if any, that should handle it and the parameters that the adapter needs to handle the request.

The TAdapterDispatchParms properties identify the adapter that can process the request, the name of the action or field that the adapter uses to process the request, and the type of request (an action to execute or a field value to retrieve).

See Also

TAdapterDispatcher (see page 6148)

IAdapterDispatchParams (see page 987)

60.1.5.1 TAdapterDispatchParams Members

The following tables list the members exposed by TAdapterDispatchParams.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetFailurePage (see page 6157)	This is GetFailurePage, a member of class TAdapterDispatchParams.
	GetHandler (see page 6157)	Returns the value of the Handler property.
	GetObjectIdentifier (see page 6157)	Returns the value of the ObjectIdentifier property.
	GetParams (see page 6158)	Returns the value of the Params property.
	GetRequestIdentifier (see page 6158)	Returns the value of the RequestIdentifier property.
	GetSuccessPage (see page 6158)	This is GetSuccessPage, a member of class TAdapterDispatchParams.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6156)	Creates and initializes a TAdapterDispatchParams (see page 6153) object based on the information in an HTTP request.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6156)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	FailurePage (see page 1026)	This is FailurePage, a member of class IGetAdapterRequestDefaultResponse.
R	Handler (see page 6159)	Indicates the adapter object that can handle the current request.
R	ObjectIdentifier (see page 6159)	Specifies the name of the action to execute or the field to retrieve.
R	Params (see page 991)	Indicates the parameter values to use when executing the action or looking up the image.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
R	RequestIdentifier (see page 6160)	Identifies the type of the request identified by the ObjectIdentifier (see page 6159) property.

	SuccessPage (see page 1026)	This is SuccessPage, a member of class IGetAdapterRequestDefaultResponse.

60.1.5.2 TAdapterDispatchParams.Create Constructor

Creates and initializes a TAdapterDispatchParams (see page 6153) object based on the information in an HTTP request.

Pascal

```
constructor Create(ARequest: TWebRequest);
```

C++

```
__fastcall TAdapterDispatchParams(TWebRequest ARequest);
```

Description

Call Create to create a new TAdapterDispatchParams (see page 6153) object and use it to obtain information encoded in an HTTP request message.

The TAdapterDispatchParams (see page 6153) constructor parses the request message that is supplied as ARequest, locating the values that identify the adapter that must be used to handle the request, and the name and type of the information that that adapter supplies. Create uses this information to initialize the Handler (see page 6159), ObjectIdentifier (see page 6159), and RequestIdentifier (see page 6160) properties.

See Also

Handler (see page 6159)

ObjectIdentifier (see page 6159)

RequestIdentifier (see page 6160)

60.1.5.3 TAdapterDispatchParams.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDispatchParams();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a

safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

60.1.5.4 TAdapterDispatchParams Methods

The methods of the TAdapterDispatchParams class are listed here.

Protected Methods

	Name	Description
	GetFailurePage (see page 6157)	This is GetFailurePage, a member of class TAdapterDispatchParams.
	GetHandler (see page 6157)	Returns the value of the Handler property.
	GetObjectIdentifier (see page 6157)	Returns the value of the ObjectIdentifier property.
	GetParams (see page 6158)	Returns the value of the Params property.
	GetRequestIdentifier (see page 6158)	Returns the value of the RequestIdentifier property.
	GetSuccessPage (see page 6158)	This is GetSuccessPage, a member of class TAdapterDispatchParams.

60.1.5.4.1 TAdapterDispatchParams.GetFailurePage Method

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

This is GetFailurePage, a member of class TAdapterDispatchParams.

60.1.5.4.2 TAdapterDispatchParams.GetHandler Method

Returns the value of the Handler property.

Pascal

```
function GetHandler: TObject;
```

C++

```
__fastcall TObject * GetHandler();
```

Description

GetHandler is the read implementation of the Handler property.

See Also

Handler (see page 990)

60.1.5.4.3 TAdapterDispatchParams.GetObjectIdentifier Method

Returns the value of the ObjectIdentifier property.

Pascal

```
function GetObjectIdentifier: string;
```

C++

```
__fastcall UnicodeString GetObjectIdentifier();
```

Description

GetObjectIdentifier is the read implementation of the ObjectIdentifier property.

See Also

ObjectIdentifier (see page 990)

60.1.5.4.4 TAdapterDispatchParams.GetParams Method

Returns the value of the Params property.

Pascal

```
function GetParams: TStrings;
```

C++

```
__fastcall TStrings GetParams();
```

Description

GetParams is the read implementation of the Params property.

See Also

Params (see page 991)

60.1.5.4.5 TAdapterDispatchParams.GetRequestIdentifier Method

Returns the value of the RequestIdentifier property.

Pascal

```
function GetRequestIdentifier: string;
```

C++

```
__fastcall UnicodeString GetRequestIdentifier();
```

Description

GetRequestIdentifier is the read implementation of the RequestIdentifier property.

See Also

RequestIdentifier (see page 991)

60.1.5.4.6 TAdapterDispatchParams.GetSuccessPage Method

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

This is GetSuccessPage, a member of class TAdapterDispatchParams.

60.1.5.5 TAdapterDispatchParams Properties

The properties of the TAdapterDispatchParams class are listed here.

Public Properties

	Name	Description
	Handler (see page 6159)	Indicates the adapter object that can handle the current request.
	ObjectIdentifier (see page 6159)	Specifies the name of the action to execute or the field to retrieve.
	RequestIdentifier (see page 6160)	Identifies the type of the request identified by the ObjectIdentifier (see page 6159) property.

60.1.5.5.1 TAdapterDispatchParams.Handler Property

Indicates the adapter object that can handle the current request.

Pascal

```
property Handler: TObject;
```

C++

```
__property TObject * Handler;
```

Description

Read (see page 3634) Handler to access the adapter that can execute the action or supply the image that is requested. The TAdapterDispatchParams (see page 6153) constructor initializes this property based on the information it parses from the current request message.

See Also

Create (see page 6156)

ObjectIdentifier (see page 6159)

RequestIdentifier (see page 6160)

60.1.5.5.2 TAdapterDispatchParams.ObjectIdentifier Property

Specifies the name of the action to execute or the field to retrieve.

Pascal

```
property ObjectIdentifier: string;
```

C++

```
__property UnicodeString ObjectIdentifier;
```

Description

Read (see page 3634) ObjectIdentifier to determine the name of the requested action or field. The TAdapterDispatchParams (see page 6153) constructor initializes this property based on the information it parses from the current request message.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 6159) parameter.

See Also

Create (see page 6156)

Handler (see page 6159)

RequestIdentifier (see page 6160)

60.1.5.5.3 TAdapterDispatchParams.RequestIdentifier Property

Identifies the type of the request identified by the ObjectIdentifier (see page 6159) property.

Pascal

```
property RequestIdentifier: string;
```

C++

```
__property UnicodeString RequestIdentifier;
```

Description

Read (see page 3634) RequestIdentifier to determine the type of processing the current request requires. For example, the RequestIdentifier may identify the request as a request for the value of an image field or a request to execute an action. The TAdapterDispatchParams (see page 6153) constructor initializes this property based on the information it parses from the current request message.

The interpretation of this value is the responsibility of the object specified by the Handler (see page 6159) parameter.

See Also

Create (see page 6156)

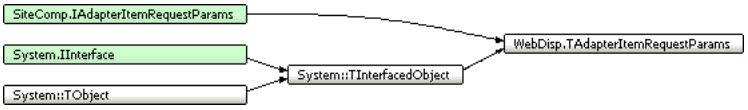
Handler (see page 6159)

ObjectIdentifier (see page 6159)

60.1.6 TAdapterItemRequestParams Class

TAdapterItemRequestParams holds the name/value pairs that must be included in request messages addressed to an adapter or its subcomponents.

Class Hierarchy



Pascal

```
TAdapterItemRequestParams = class(TInterfacedObject, IAdapterItemRequestParams);
```

C++

```
class TAdapterItemRequestParams : public TInterfacedObject, public IAdapterItemRequestParams;
```

File

WebDisp

Description

TAdapterItemRequestParams is the default implementation of the IAdapterItemRequestParams (see page 1005) interface. It holds the name/value pairs that should be included as request parameters to an adapter or one of its subcomponents.

The parameters, which can be accessed through the ParamValues (see page 6165) property, are a set of 'Name=Value' strings. In addition to holding these parameters, TAdapterItemRequestParams has two properties, SuccessPage (see page 6165) and FailurePage (see page 6165), which identify the Web pages that should be returned to the caller when the request succeeds or fails.

See Also

TAdapterDispatcher (see page 6148)

IAdapterItemRequestParams (see page 1005)

GetAdapterItemRequestParams (see page 5629)

60.1.6.1 TAdapterItemRequestParams Members

The following tables list the members exposed by TAdapterItemRequestParams.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetFailurePage (see page 6163)	Returns the value of the FailurePage property.
	GetParamValues (see page 6163)	Returns the value of the ParamValues property.
	GetSuccessPage (see page 6164)	Returns the value of the SuccessPage property.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	SetFailurePage (see page 6164)	Sets the value of the FailurePage property.
	SetSuccessPage (see page 6164)	Sets the value of the SuccessPage property.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6162)	Creates and initializes a TAdapterItemRequestParams (see page 6160) object.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6162)	Frees an instance of TAdapterItemRequestParams (see page 6160).
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.

	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	FailurePage (see page 6165)	Indicates the Web page that the application should return if the current request fails.
	ParamValues (see page 6165)	Lists the parameter values that are included in requests to the adapter action or field.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	SuccessPage (see page 6165)	Indicates the Web page that the application should return if the request succeeds.

60.1.6.2 TAdapterItemRequestParams.Create Constructor

Creates and initializes a TAdapterItemRequestParams (see page 6160) object.

Pascal

```
constructor Create;
```

C++

```
__fastcall TAdapterItemRequestParams();
```

Description

Typically, applications have no need to create TAdapterItemRequestParams (see page 6160) instances. The adapter dispatcher creates these objects to hold the parameters for a request.

See Also

TAdapterDispatcher (see page 6148)

60.1.6.3 TAdapterItemRequestParams.Destroy Destructor

Frees an instance of TAdapterItemRequestParams (see page 6160).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterItemRequestParams();
```

Description

Most applications have no need to create or free TAdapterItemRequestParams (see page 6160) instances. These objects are managed by the adapter dispatcher.

See Also

TAdapterDispatcher (see page 6148)

60.1.6.4 TAdapterItemRequestParams Methods

The methods of the TAdapterItemRequestParams class are listed here.

Protected Methods

	Name	Description
	GetFailurePage (see page 6163)	Returns the value of the FailurePage property.
	GetParamValues (see page 6163)	Returns the value of the ParamValues property.
	GetSuccessPage (see page 6164)	Returns the value of the SuccessPage property.
	SetFailurePage (see page 6164)	Sets the value of the FailurePage property.
	SetSuccessPage (see page 6164)	Sets the value of the SuccessPage property.

60.1.6.4.1 TAdapterItemRequestParams.GetFailurePage Method

Returns the value of the FailurePage property.

Pascal

```
function GetFailurePage: string;
```

C++

```
__fastcall UnicodeString GetFailurePage();
```

Description

GetFailurePage is the read implementation of the FailurePage property.

See Also

FailurePage (see page 1008)

60.1.6.4.2 TAdapterItemRequestParams.GetParamValues Method

Returns the value of the ParamValues property.

Pascal

```
function GetParamValues: TStrings;
```

C++

```
__fastcall TStrings GetParamValues();
```

Description

GetParamValues is the read implementation of the ParamValues property.

See Also

ParamValues (see page 1009)

60.1.6.4.3 TAdapterItemRequestParams.GetSuccessPage Method

Returns the value of the SuccessPage property.

Pascal

```
function GetSuccessPage: string;
```

C++

```
__fastcall UnicodeString GetSuccessPage();
```

Description

GetSuccessPage is the read implementation of the SuccessPage property.

See Also

SuccessPage (see page 1009)

60.1.6.4.4 TAdapterItemRequestParams.SetFailurePage Method

Sets the value of the FailurePage property.

Pascal

```
procedure SetFailurePage(const AValue: string);
```

C++

```
__fastcall SetFailurePage(const UnicodeString AValue);
```

Description

SetFailurePage is the write implementation of the FailurePage property.

AValue is the name of the Web page that should be returned if the adapter request fails.

See Also

FailurePage (see page 1008)

60.1.6.4.5 TAdapterItemRequestParams.SetSuccessPage Method

Sets the value of the SuccessPage property.

Pascal

```
procedure SetSuccessPage(const AValue: string);
```

C++

```
__fastcall SetSuccessPage(const UnicodeString AValue);
```

Description

SetSuccessPage is the write implementation of the SuccessPage property.

AValue is the name of the Web page that should be returned when the adapter request succeeds.

See Also

SuccessPage (see page 1009)

60.1.6.5 TAdapterItemRequestParams Properties

The properties of the TAdapterItemRequestParams class are listed here.

Public Properties

	Name	Description
	FailurePage (see page 6165)	Indicates the Web page that the application should return if the current request fails.
R	ParamValues (see page 6165)	Lists the parameter values that are included in requests to the adapter action or field.
	SuccessPage (see page 6165)	Indicates the Web page that the application should return if the request succeeds.

60.1.6.5.1 TAdapterItemRequestParams.FailurePage Property

Indicates the Web page that the application should return if the current request fails.

Pascal

```
property FailurePage: string;
```

C++

```
__property UnicodeString FailurePage;
```

Description

FailurePage is the name of the registered Web page module that should generate the content of the response message if the request to the adapter fails.

See Also

SuccessPage (see page 6165)

60.1.6.5.2 TAdapterItemRequestParams.ParamValues Property

Lists the parameter values that are included in requests to the adapter action or field.

Pascal

```
property ParamValues: TStrings;
```

C++

```
__property TStrings ParamValues;
```

Description

Use ParamValues to access the list of parameter values. Use the Names property of ParamValues to obtain the names of the parameters and the Values property to access the corresponding values.

See Also

Names

Values

60.1.6.5.3 TAdapterItemRequestParams.SuccessPage Property

Indicates the Web page that the application should return if the request succeeds.

Pascal

```
property SuccessPage: string;
```

C++

```
__property UnicodeString SuccessPage;
```

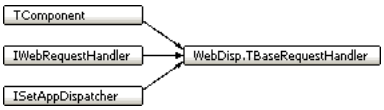
Description

SuccessPage is the name of the registered Web page module that should generate the content of the response message when the request to the adapter succeeds.

60.1.7 TBaseRequestHandler Class

The TBaseRequestHandler component is a base class for WebSnap dispatcher components.

Class Hierarchy



Pascal

```
TBaseRequestHandler = class(TComponent, IWebRequestHandler, ISetAppDispatcher);
```

C++

```
class TBaseRequestHandler : public TComponent, public IWebRequestHandler, public
ISetAppDispatcher;
```

File

WebDisp

Description

The TCustomAdapterDispatcher (see page 6168) and TCustomPageDispatcher (see page 6188) components are derived from TBaseRequestHandler.

60.1.7.1 TBaseRequestHandler Members

The following tables list the members exposed by TBaseRequestHandler.

Protected Methods

	Name	Description
	HandleRequest (see page 6167)	Handles an HTTP request.
V	ImplHandleRequest (see page 6167)	Handles application-specific processing of an HTTP request.
	SetAppDispatcher (see page 6167)	Sets the AppDispatcher (see page 6168) property.

Protected Properties

	Name	Description
R	AppDispatcher (see page 6168)	Provides read-only access to the application dispatcher component.

60.1.7.2 TBaseRequestHandler Methods

The methods of the TBaseRequestHandler class are listed here.

Protected Methods

	Name	Description
	HandleRequest (see page 6167)	Handles an HTTP request.
	ImplHandleRequest (see page 6167)	Handles application-specific processing of an HTTP request.
	SetAppDispatcher (see page 6167)	Sets the AppDispatcher (see page 6168) property.

60.1.7.2.1 TBaseRequestHandler.HandleRequest Method

Handles an HTTP request.

Pascal

```
function HandleRequest(ARequest: TWebRequest; AResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest ARequest, TWebResponse AResponse);
```

Description

HandleRequest returns true if it handles the request specified by ARequest. This function calls the ImplHandleRequest (see page 6167) to process the request. A customized overridden function should be written for ImplHandleRequest (see page 6167) to do the necessary processing specific to the application.

See Also

ImplHandleRequest (see page 6167)

60.1.7.2.2 TBaseRequestHandler.ImplHandleRequest Method

Handles application-specific processing of an HTTP request.

Pascal

```
function ImplHandleRequest(ARequest: TWebRequest; AResponse: TWebResponse): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest ARequest, TWebResponse AResponse);
```

Description

The ImplHandleRequest function is called from HandleRequest (see page 6167) to handle the application-specific processing of the HTTP request.

This base version of the function generates an assertion failure as a reminder that a customized overridden version of this function needs to be defined in order to handle the request processing specific to the application.

See Also

HandleRequest (see page 6167)

60.1.7.2.3 TBaseRequestHandler.SetAppDispatcher Method

Sets the AppDispatcher (see page 6168) property.

Pascal

```
procedure SetAppDispatcher(const AWebAppDispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * AWebAppDispatcher);
```

Description

SetAppDispatcher provides a way to set the AppDispatcher (see page 6168) property.

See Also

AppDispatcher (see page 6168)

60.1.7.3 TBaseRequestHandler Properties

The properties of the TBaseRequestHandler class are listed here.

Protected Properties

	Name	Description
	AppDispatcher (see page 6168)	Provides read-only access to the application dispatcher component.

60.1.7.3.1 TBaseRequestHandler.AppDispatcher Property

Provides read-only access to the application dispatcher component.

Pascal

```
property AppDispatcher: TComponent;
```

C++

```
__property TComponent * AppDispatcher;
```

Description

The application dispatcher component is typically TWebAppComponents (see page 6243). The application dispatcher component is used to access application objects such as TWebRequest (see page 6276) and application components such as TEndUserAdapter.

See Also

TCustomAdapterDispatcher (see page 6168)

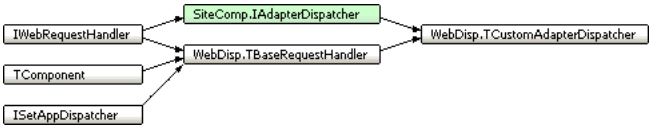
TCustomPageDispatcher (see page 6188)

TWebAppComponents (see page 6243)

60.1.8 TCustomAdapterDispatcher Class

TCustomAdapterDispatcher automatically handles HTML form submissions and requests for dynamic images by calling adapter action and field components.

Class Hierarchy



Pascal

```
TCustomAdapterDispatcher = class(TBaseRequestHandler, IAdapterDispatcher);
```

C++

```
class TCustomAdapterDispatcher : public TBaseRequestHandler, public IAdapterDispatcher;
```

File

WebDisp

Description

TCustomAdapterDispatcher is the base class for the adapter dispatcher. The adapter dispatcher is responsible for dispatching adapter actions and references to image fields. It also handles the encoding of parameters that must be included in HTML form submissions that invoke those adapter actions or field references. If a WebSnap application does not include an AdapterDispatcher component, then it can't execute adapter actions or obtain images from adapter fields.

See Also

TAdapter (see page 5190)

TAdapterDispatcher (see page 6148)

TCustomWebAppComponents (see page 6201)

60.1.8.1 TCustomAdapterDispatcher Members

The following tables list the members exposed by TCustomAdapterDispatcher.

Protected Methods

	Name	Description
	AddSpecialParams (see page 6170)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
	DoAdapterRequestNotHandled (see page 6171)	Generates an OnActionRequestNotHandled (see page 6177) and an OnImageRequestNotHandled (see page 6180) event.
	DoAfterDispatchAdapterRequest (see page 6172)	Generates an OnAfterDispatchAction (see page 6178) and an OnAfterDispatchImage (see page 6178) event.
	DoBeforeDispatchAdapterRequest (see page 6172)	Generates an OnBeforeDispatchAction (see page 6179) and an OnBeforeDispatchImage (see page 6179) event.
	EncodeAdapterItemRequestAsFieldValue (see page 6173)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 6173)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 6174)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetDispatchParams (see page 6174)	Parses a request to determine the adapter to which it is directed.
	HandleRequest (see page 6167)	Handles an HTTP request.
	ImplEncodeAdapterItemRequestAsFieldValue (see page 6175)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6173) method.
	ImplEncodeAdapterItemRequestAsHREF (see page 6175)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6173) method.

	ImplGetAdapterItemRequestParams (see page 6176)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6174) method.
	ImplHandleRequest (see page 6176)	Provides the underlying implementation for the HandleRequest (see page 6167) method.
	SetAppDispatcher (see page 6167)	Sets the AppDispatcher (see page 6168) property.

Protected Properties

	Name	Description
	AppDispatcher (see page 6168)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnActionRequestNotHandled (see page 6177)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6178)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 6178)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6179)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6179)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6180)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 6180)	Occurs when the WebSnap application could not provide a requested image.

60.1.8.2 TCustomAdapterDispatcher Methods

The methods of the TCustomAdapterDispatcher class are listed here.

Protected Methods

	Name	Description
	AddSpecialParams (see page 6170)	Adds parameters for the success and failure Web pages to the list of adapter parameters.
	DoAdapterRequestNotHandled (see page 6171)	Generates an OnActionRequestNotHandled (see page 6177) and an OnImageRequestNotHandled (see page 6180) event.
	DoAfterDispatchAdapterRequest (see page 6172)	Generates an OnAfterDispatchAction (see page 6178) and an OnAfterDispatchImage (see page 6178) event.
	DoBeforeDispatchAdapterRequest (see page 6172)	Generates an OnBeforeDispatchAction (see page 6179) and an OnBeforeDispatchImage (see page 6179) event.
	EncodeAdapterItemRequestAsFieldValue (see page 6173)	Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.
	EncodeAdapterItemRequestAsHREF (see page 6173)	Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.
	GetAdapterItemRequestParams (see page 6174)	Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.
	GetDispatchParams (see page 6174)	Parses a request to determine the adapter to which it is directed.
	ImplEncodeAdapterItemRequestAsFieldValue (see page 6175)	Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6173) method.
	ImplEncodeAdapterItemRequestAsHREF (see page 6175)	Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6173) method.
	ImplGetAdapterItemRequestParams (see page 6176)	Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6174) method.
	ImplHandleRequest (see page 6176)	Provides the underlying implementation for the HandleRequest (see page 6167) method.

60.1.8.2.1 TCustomAdapterDispatcher.AddSpecialParams Method

Adds parameters for the success and failure Web pages to the list of adapter parameters.

Pascal

```
procedure AddSpecialParams(AHandler: IUnknown; Params: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall AddSpecialParams(IUnknown AHandler, IAdapterItemRequestParams Params);
```

Description

The adapter dispatcher calls AddSpecialParams internally after retrieving a set of parameters from an adapter (handler) and before generating an OnGetAdapterRequestParams (see page 6180) event. This method adds parameters to the list that are not required by the adapter for handling the request.

AddSpecialParams retrieves the Web pages that should be displayed when the adapter succeeds or fails in processing the request with which the handler is associated. If the handler has any special pages to display, AddSpecialParams adds the names of these pages to the parameters, using the parameter names of '__sp' and '__fp' (for success page and failure page), or '__p' (if the same page should be used for both success and failure).

AHandler is the interface for the adapter associated with the action or image.

Params is an interface that provides access to the current list of parameters.

See Also

GetAdapterItemRequestParams (see page 6174)

OnGetAdapterRequestParams (see page 6180)

FailurePage (see page 1008)

SuccessPage (see page 1009)

60.1.8.2.2 TCustomAdapterDispatcher.DoAdapterRequestNotHandled Method

Generates an OnActionRequestNotHandled (see page 6177) and an OnImageRequestNotHandled (see page 6180) event.

Pascal

```
procedure DoAdapterRequestNotHandled(AObject: TObject; var AHandled: Boolean);
```

C++

```
__fastcall DoAdapterRequestNotHandled(TObject * AObject, Boolean AHandled);
```

Description

TCustomAdapterDispatcher (see page 6168) calls DoAdapterRequestNotHandled internally when an adapter request remains unhandled. The request may be unhandled because the adapter could not execute the requested action or obtain the requested image, or it may be unhandled because the dispatcher could not locate an adapter to handle the request.

DoAdapterRequestNotHandled assembles the parameters for an action request and generates an OnActionRequestNotHandled (see page 6177) event. It then assembles the parameters for an image request and generates an OnImageRequestNotHandled (see page 6180) event.

AObject is the adapter that failed to handle the request (or nil (Delphi) or NULL (C++) if the dispatcher could not locate an adapter for the request).

AHandled returns true if an event handler handled the request.

See Also

OnActionRequestNotHandled (see page 6177)

OnImageRequestNotHandled (see page 6180)

60.1.8.2.3 TCustomAdapterDispatcher.DoAfterDispatchAdapterRequest Method

Generates an OnAfterDispatchAction (see page 6178) and an OnAfterDispatchImage (see page 6178) event.

Pascal

```
procedure DoAfterDispatchAdapterRequest(AObject: TObject);
```

C++

```
__fastcall DoAfterDispatchAdapterRequest(TObject * AObject);
```

Description

TCustomAdapterDispatcher (see page 6168) calls DoAfterDispatchAdapterRequest internally after calling on an adapter to handle a request. It calls this method both when the adapter indicates success and when it indicates failure in handling the request.

DoAfterDispatchAdapterRequest assembles the parameters for an action request and generates an OnAfterDispatchAction (see page 6178) event. It then assembles the parameters for an image request and generates an OnAfterDispatchImage (see page 6178) event.

AObject is the adapter that just handled the current request.

See Also

OnAfterDispatchAction (see page 6178)

OnAfterDispatchImage (see page 6178)

60.1.8.2.4 TCustomAdapterDispatcher.DoBeforeDispatchAdapterRequest Method

Generates an OnBeforeDispatchAction (see page 6179) and an OnBeforeDispatchImage (see page 6179) event.

Pascal

```
procedure DoBeforeDispatchAdapterRequest(AObject: TObject; var AHandled: Boolean);
```

C++

```
__fastcall DoBeforeDispatchAdapterRequest(TObject * AObject, Boolean AHandled);
```

Description

TCustomAdapterDispatcher (see page 6168) calls DoBeforeDispatchAdapterRequest internally before calling on an adapter to handle a request. It calls this method only after it has identified the appropriate handler for responding to the current request.

DoBeforeDispatchAdapterRequest assembles the parameters for an action request and generates an OnBeforeDispatchAction (see page 6179) event. It then assembles the parameters for an image request and generates an OnBeforeDispatchImage (see page 6179) event.

AObject is the adapter for handling the current request.

AHandled returns true if an event handler handled the request.

See Also

OnBeforeDispatchAction (see page 6179)

OnBeforeDispatchImage (see page 6179)

60.1.8.2.5 TCustomAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue Method

Returns the hidden fields that can be added to an HTML form to represent the parameters of a request to an adapter action or image field.

Pascal

```
function EncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Adapter components call EncodeAdapterItemRequestAsFieldValue to encode the action or field parameters as they should appear as field values in form submissions. The resulting parameter string has a form such as '?param1=value1&param2=value2'.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 6174)

EncodeAdapterItemRequestAsHREF (see page 6173)

60.1.8.2.6 TCustomAdapterDispatcher.EncodeAdapterItemRequestAsHREF Method

Returns the HREF parameters that can be added to an anchor tag on an HTML form to represent the parameters of a request to an adapter.

Pascal

```
function EncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string;
```

C++

```
__fastcall UnicodeString EncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

Adapter components call EncodeAdapterItemRequestAsHREF to encode the action or image field parameters as they should appear as parameters to an anchor tag in an HTML form.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF returns the parameter string.

See Also

GetAdapterItemRequestParams (see page 6174)

EncodeAdapterItemRequestAsFieldValue (see page 6173)

60.1.8.2.7 TCustomAdapterDispatcher.GetAdapterItemRequestParams Method

Retrieves an interface for reading and writing the parameters of a request to an adapter action or image field.

Pascal

```
procedure GetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; Params: IAdapterItemRequestParams);
```

C++

```
__fastcall GetAdapterItemRequestParams(IUnknown AHandler, TAdapterItemRequestParamOptions
AOptions, IAdapterItemRequestParams Params);
```

Description

GetAdapterItemRequestParams retrieves the parameters for a call to execute an adapter action or to fetch the value of an image field.

AHandler is the interface of the adapter that requires the parameters to execute the action or produce the image field value.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. GetAdapterItemRequestParams uses this interface to write the parameter values.

GetAdapterItemRequestParams simply calls the ImplGetAdapterItemRequestParams (see page 6176) method, which fetches the parameter values and writes them to AParams.

See Also

EncodeAdapterItemRequestAsHREF (see page 6173)

EncodeAdapterItemRequestAsFieldValue (see page 6173)

ImplGetAdapterItemRequestParams (see page 6176)

60.1.8.2.8 TCustomAdapterDispatcher.GetDispatchParams Method

Parses a request to determine the adapter to which it is directed.

Pascal

```
function GetDispatchParams(Request: TWebRequest): TAdapterDispatchParams; virtual;
```

C++

```
virtual __fastcall TAdapterDispatchParams GetDispatchParams(TWebRequest Request);
```

Description

TCustomAdapterDispatcher (see page 6168) calls GetDispatchParams internally to obtain information about the adapter to which a request is directed. GetDispatchParams creates a new TAdapterDispatchParams (see page 6153) object, which parses the current request to determine the adapter to which it is directed.

Request is the current http request object.

GetDispatchParams returns the new TAdapterDispatchParams (see page 6153) object, whose properties have been filled out to reflect the information in the current request. It is the caller's responsibility to free this object.

60.1.8.2.9
TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsFieldValue Method

Provides the underlying implementation for the EncodeAdapterItemRequestAsFieldValue (see page 6173) method.

Pascal

```
function ImplEncodeAdapterItemRequestAsFieldValue(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplEncodeAdapterItemRequestAsFieldValue(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

The EncodeAdapterItemRequestAsFieldValue (see page 6173) method calls ImplEncodeAdapterItemRequestAsFieldValue to retrieve a set of parameters formatted as a set of field values for an HTTP request. ImplEncodeAdapterItemRequestAsFieldValue calls GetAdapterItemRequestParams (see page 6174) to fetch the parameter values, and then formats them as a set of field values.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItmRequestAsFieldValue returns the parameter string.

See Also

EncodeAdapterItemRequestAsFieldValue (see page 6173)

GetAdapterItemRequestParams (see page 6174)

60.1.8.2.10
TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsHREF Method

Provides the underlying implementation for the EncodeAdapterItemRequestAsHREF (see page 6173) method.

Pascal

```
function ImplEncodeAdapterItemRequestAsHREF(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplEncodeAdapterItemRequestAsHREF(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions);
```

Description

The EncodeAdapterItemRequestAsHREF (see page 6173) method calls ImplEncodeAdapterItemRequestAsFieldValue (see page 6175) to retrieve a set of parameters formatted as the parameters to an anchor tag in an HTML form. ImplEncodeAdapterItemRequestAsHREF calls GetAdapterItemRequestParams (see page 6174) to fetch the parameter values, and then formats them.

AHandler is the interface of the adapter whose action or image field requires the parameters.

AOptions is a set of flags that indicate what types of parameters to include.

EncodeAdapterItemRequestAsHREF (see page 6173) returns the parameter string.

See Also

EncodeAdapterItemRequestAsHREF (see page 6173)

GetAdapterItemRequestParams (see page 6174)

60.1.8.2.11 TCustomAdapterDispatcher.ImplGetAdapterItemRequestParams Method

Provides the underlying implementation for the GetAdapterItemRequestParams (see page 6174) method.

Pascal

```
procedure ImplGetAdapterItemRequestParams(AHandler: IUnknown; AOptions:
TAdapterItemRequestParamOptions; Params: IAdapterItemRequestParams); virtual;
```

C++

```
virtual __fastcall ImplGetAdapterItemRequestParams(IUnknown AHandler,
TAdapterItemRequestParamOptions AOptions, IAdapterItemRequestParams Params);
```

Description

GetAdapterItemRequestParams (see page 6174) calls ImplGetAdapterItemRequestParams to retrieve the parameters for a call to execute an adapter action or to fetch the value of an image field. This method fetches the requested parameters from the adapter interface, calls the AddSpecialParams (see page 6170) method to add additional parameters for the return Web page, and then generates an OnGetAdapterRequestParams (see page 6180) event so that you can make your own final adjustments to the parameter list.

AHandler is the interface of the adapter that requires the parameters to execute the action or produce the image field value.

AOptions is a set of flags that indicate what types of parameters should be included.

AParams is an interface for reading and writing parameter values. ImplGetAdapterItemRequestParams uses this interface to write the parameter values.

See Also

GetAdapterItemRequestParams (see page 6174)

AddSpecialParams (see page 6170)

OnGetAdapterRequestParams (see page 6180)

60.1.8.2.12 TCustomAdapterDispatcher.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 6167) method.

Pascal

```
function ImplHandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

The HandleRequest (see page 6167) method calls ImplHandleRequest to process an http request message and obtain the response that should be sent back to the caller. (The HandleRequest (see page 6167) method is called using the IWebRequestHandler interface.)

ImplHandleRequest

1. Calls GetDispatchParams (see page 6174) to parse the request and identify the adapter that should handle it.
2. If it finds an adapter, calls DoBeforeDispatchAdapterRequest (see page 6172) to generate the OnBeforeDispatchAction (see page 6179) and OnBeforeDispatchImage (see page 6179) events.
3. If the event handlers do not handle the event, forwards the request to the adapter for processing.
4. Calls DoAfterDispatchAdapterRequest (see page 6172) to generate the OnAfterDispatchAction (see page 6178) and OnAfterDispatchImage (see page 6178) events.
5. Calls DoAdapterRequestNotHandled (see page 6171) to generate the OnActionRequestNotHandled (see page 6177) and OnImageRequestNotHandled (see page 6180) events if the request has not yet been handled.

See Also

IWebRequestHandler

HandleRequest (see page 6167)

GetDispatchParams (see page 6174)

DoBeforeDispatchAdapterRequest (see page 6172)

DoAfterDispatchAdapterRequest (see page 6172)

DoAdapterRequestNotHandled (see page 6171)

60.1.8.3 TCustomAdapterDispatcher Events

The events of the TCustomAdapterDispatcher class are listed here.

Public Events

	Name	Description
	OnActionRequestNotHandled (see page 6177)	Occurs when the WebSnap application could not execute a requested action.
	OnAfterDispatchAction (see page 6178)	Occurs after the target adapter has finished executing an action.
	OnAfterDispatchImage (see page 6178)	Occurs after the target adapter has finished handling an image request.
	OnBeforeDispatchAction (see page 6179)	Occurs before the dispatcher forwards an action to an adapter for execution.
	OnBeforeDispatchImage (see page 6179)	Occurs before the dispatcher forwards an image request to an adapter for handling.
	OnGetAdapterRequestParams (see page 6180)	Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.
	OnImageRequestNotHandled (see page 6180)	Occurs when the WebSnap application could not provide a requested image.

60.1.8.3.1 TCustomAdapterDispatcher.OnActionRequestNotHandled Event

Occurs when the WebSnap application could not execute a requested action.

Pascal

```
property OnActionRequestNotHandled: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnActionRequestNotHandled;
```

Description

Write (see page 3707) an OnActionRequestNotHandled event handler to provide a response to actions that the adapter components in the application can't execute. This may occur because the adapter dispatcher could not find an adapter to handle the action, or it may occur because the identified adapter was unable to execute the action.

If the OnActionRequestNotHandled event handler does not set its Handled parameter to true, the adapter dispatcher generates

an OnImageRequestNotHandled (see page 6180). If the OnImageRequestNotHandled (see page 6180) event handler also does not handle the event, the dispatcher raises an EAdapterDispatcherException (see page 6146) exception.

Tip: Use the Response property of the current Web context to access the response message from the OnActionRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 6003) function.

See Also

OnAfterDispatchAction (see page 6178)

OnBeforeDispatchAction (see page 6179)

EAdapterDispatcherException (see page 6146)

WebContext (see page 6003)

Response (see page 5975)

OnImageRequestNotHandled (see page 6180)

60.1.8.3.2 TCustomAdapterDispatcher.OnAfterDispatchAction Event

Occurs after the target adapter has finished executing an action.

Pascal

```
property OnAfterDispatchAction: TDispatchActionEvent;
```

C++

```
__property TDispatchActionEvent OnAfterDispatchAction;
```

Description

Write (see page 3707) an OnAfterDispatchAction event handler to take specific action after an adapter has executed an action. Note that this event can occur even if the request does not request the execution of an action or if the adapter was not successful in its attempt to execute the action.

See Also

OnActionRequestNotHandled (see page 6177)

OnBeforeDispatchAction (see page 6179)

60.1.8.3.3 TCustomAdapterDispatcher.OnAfterDispatchImage Event

Occurs after the target adapter has finished handling an image request.

Pascal

```
property OnAfterDispatchImage: TDispatchImageEvent;
```

C++

```
__property TDispatchImageEvent OnAfterDispatchImage;
```

Description

Write (see page 3707) an OnAfterDispatchImage event handler to take specific action after an adapter has responded to an image request. Note that this event can occur even if the request does not request the value of an image field or if the adapter was not successful in its attempt to honor that request.

See Also

OnImageRequestNotHandled (see page 6180)

OnBeforeDispatchImage (see page 6179)

60.1.8.3.4 TCustomAdapterDispatcher.OnBeforeDispatchAction Event

Occurs before the dispatcher forwards an action to an adapter for execution.

Pascal

```
property OnBeforeDispatchAction: TDispatchActionHandledEvent;
```

C++

```
__property TDispatchActionHandledEvent OnBeforeDispatchAction;
```

Description

Write (see page 3707) an OnBeforeDispatchAction event handler to respond to action requests before they are forwarded to the associated adapter for execution. Note that if the request does not request the execution of an action, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchAction event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Action parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for execution, and does not generate an OnAfterDispatchAction (see page 6178) event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchAction event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 6003) function.

See Also

OnActionRequestNotHandled (see page 6177)

OnAfterDispatchAction (see page 6178)

WebContext (see page 6003)

Response (see page 5975)

60.1.8.3.5 TCustomAdapterDispatcher.OnBeforeDispatchImage Event

Occurs before the dispatcher forwards an image request to an adapter for handling.

Pascal

```
property OnBeforeDispatchImage: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnBeforeDispatchImage;
```

Description

Write (see page 3707) an OnBeforeDispatchImage event handler to respond to image requests before they are forwarded to the associated adapter for handling. Note that if the request does not ask for the value of an image field, the Params parameter of the event handler is nil (Delphi) or NULL (C++).

If the OnBeforeDispatchImage event handler does not set its Handled parameter to true, the adapter dispatcher forwards the request to the identified adapter that is the value of the Image parameter. If the event handler sets Handled to false, the dispatcher does not forward the request to the adapter for handling, and does not generate an OnAfterDispatchImage (see page 6178) event.

Tip: Use the Response property of the current Web context to access the response message from the OnBeforeDispatchImage event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 6003) function.

See Also

OnImageRequestNotHandled (see page 6180)

OnAfterDispatchImage (see page 6178)

WebContext (see page 6003)

Response (see page 5975)

60.1.8.3.6 TCustomAdapterDispatcher.OnGetAdapterRequestParams Event

Occurs when the adapter dispatcher needs to obtain the parameters for executing an action or retrieving an image.

Pascal

```
property OnGetAdapterRequestParams: TGetAdapterRequestParamsEvent;
```

C++

```
__property TGetAdapterRequestParamsEvent OnGetAdapterRequestParams;
```

Description

Write (see page 3707) an OnGetAdapterRequestParams event handler to modify the list of parameters for an action or image request before they are used. The OnGetAdapterRequestParams event handler can modify the current parameters or add its own parameters to the list. This event occurs both when the dispatcher formats parameters that are added to HTML forms and when it is dispatching a request message that contains an action or image request.

See Also

GetAdapterItemRequestParams (see page 6174)

EncodeAdapterItemRequestAsFieldValue (see page 6173)

EncodeAdapterItemRequestAsHREF (see page 6173)

AddSpecialParams (see page 6170)

60.1.8.3.7 TCustomAdapterDispatcher.OnImageRequestNotHandled Event

Occurs when the WebSnap application could not provide a requested image.

Pascal

```
property OnImageRequestNotHandled: TDispatchImageHandledEvent;
```

C++

```
__property TDispatchImageHandledEvent OnImageRequestNotHandled;
```

Description

Write (see page 3707) an OnImageRequestNotHandled event handler to provide a response to image requests that the adapter components in the application can't honor. This may occur because the adapter dispatcher could not find an adapter to handle the request, or it may occur because the identified adapter was unable to provide the requested image.

If the OnImageRequestNotHandled event handler exits with its Handled parameter set to false, the dispatcher raises an EAdapterDispatcherException (see page 6146) exception.

Tip: Use the Response property of the current Web context to access the response message from the OnImageRequestNotHandled event handler. You can obtain a reference to the Web context by calling the global WebContext (see page 6003) function.

See Also

OnAfterDispatchImage (see page 6178)

OnBeforeDispatchImage (see page 6179)

EAdapterDispatcherException (see page 6146)

WebContext (see page 6003)

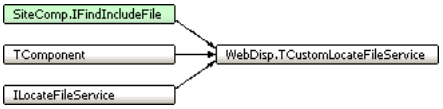
Response (see page 5975)

OnActionRequestNotHandled (see page 6177)

60.1.9 TCustomLocateFileService Class

TCustomLocateFileService uses event handlers to control access to templates and server-side script include files at runtime.

Class Hierarchy



Pascal

```
TCustomLocateFileService = class(TComponent, ILocateFileService, IFindIncludeFile);
```

C++

```
class TCustomLocateFileService : public TComponent, public ILocateFileService, public
IFindIncludeFile;
```

File

WebDisp

Description

TCustomLocateFileService is the base class for components that control access to files in a WebSnap application. TCustomLocateFileService defines events that indicate where to locate a requested file or that supply a stream object from which to read the contents of a file. By using a TCustomLocateFileService descendant, a WebSnap application can adjust to differences in the way files are deployed or even to use different files, depending on the current Web context.

Do not create instances of TCustomLocateFileService. All of the events it introduces are protected, and so unavailable to your application. Instead, use a TCustomLocateFileService descendant, such as TLocateFileService (see page 6220), which publishes those events.

See Also

TLocateFileService (see page 6220)

TStream

60.1.9.1 TCustomLocateFileService Members

The following tables list the members exposed by TCustomLocateFileService.

Protected Events

	Name	Description
	OnFindIncludeFile (see page 6186)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6187)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6187)	Occurs when the WebSnap application tries to locate the template file for a content producer.

Protected Methods

	Name	Description
	FindIncludeFile (see page 6182)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 6183)	Generates an OnFindStream (see page 6187) event.
	FindTemplateFile (see page 6183)	Generates an OnFindTemplateFile (see page 6187) event.
V	GetFileStream (see page 6184)	Returns a stream object from which to read a specified file.
	GetTemplateStream (see page 6184)	Returns a stream from which to read a template file.
V	ImplFindIncludeFile (see page 6185)	Provides the underlying implementation for the FindIncludeFile (see page 6182) method.
V	ImplGetTemplateStream (see page 6185)	Provides the underlying implementation for the GetTemplateStream (see page 6184) method.

60.1.9.2 TCustomLocateFileService Methods

The methods of the TCustomLocateFileService class are listed here.

Protected Methods

	Name	Description
	FindIncludeFile (see page 6182)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 6183)	Generates an OnFindStream (see page 6187) event.
	FindTemplateFile (see page 6183)	Generates an OnFindTemplateFile (see page 6187) event.
V	GetFileStream (see page 6184)	Returns a stream object from which to read a specified file.
	GetTemplateStream (see page 6184)	Returns a stream from which to read a template file.
V	ImplFindIncludeFile (see page 6185)	Provides the underlying implementation for the FindIncludeFile (see page 6182) method.
V	ImplGetTemplateStream (see page 6185)	Provides the underlying implementation for the GetTemplateStream (see page 6184) method.

60.1.9.2.1 TCustomLocateFileService.FindIncludeFile Method

Returns the fully-qualified path name for an include file for a server-side script.

Pascal

```
function FindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean;
```

C++

```
__fastcall Boolean FindIncludeFile(TComponent * AComponent, const UnicodeString AFileName,
UnicodeString AResult);
```

Description

WebSnap applications call FindIncludeFile when they encounter a reference to an include file in server-side script. This method returns the fully-qualified path name for the include file.

FindIncludeFile calls the virtual ImplFindIncludeFile (see page 6185) method to obtain the path and name of the include file. In TCustomLocateFileService (see page 6181), ImplFindIncludeFile (see page 6185) fetches the include file name by generating an OnFindIncludeFile (see page 6186) event. Component writers that which to locate include files in some other

way should override the ImplFindIncludeFile (see page 6185) method.

AComponent is the Web module context object that stores the reference to the include file.

AFileName is the name of the include file to locate.

AResult returns the fully-qualified path name where the WebSnap application can find the requested file.

FindIncludeFile returns true if it locates the requested file and returns its fully-qualified name as AResult.

See Also

OnFindIncludeFile (see page 6186)

ImplFindIncludeFile (see page 6185)

WebModuleContext

60.1.9.2.2 TCustomLocateFileService.FindStream Method

Generates an OnFindStream (see page 6187) event.

Pascal

```
function FindStream(AComponent: TComponent; const AFileName: string; out AResult: TStream; out AOwned: Boolean): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean FindStream(TComponent * AComponent, const UnicodeString AFileName, TStream AResult, Boolean AOwned);
```

Description

TCustomLocateFileService (see page 6181) calls FindStream internally to generate an OnFindStream (see page 6187) event.

AComponent is the object that uses or is associated with the requested file. Typically, this is a content producer.

AFileName is the name of the file to locate. This can be a "logical" file that does not exist on disk.

AResult returns a stream from which the caller can read the contents of the requested file.

AOwned indicates whether the caller should free AResult after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

FindStream returns true if it supplies a stream as the value of the AResult parameter. It returns false if there is no OnFindStream (see page 6187) event handler or if the event handler does not return a value for AResult.

See Also

OnFindStream (see page 6187)

60.1.9.2.3 TCustomLocateFileService.FindTemplateFile Method

Generates an OnFindTemplateFile (see page 6187) event.

Pascal

```
function FindTemplateFile(AComponent: TComponent; const AFileName: string; var AResult: string): Boolean;
```

C++

```
__fastcall Boolean FindTemplateFile(TComponent * AComponent, const UnicodeString AFileName, UnicodeString AResult);
```

Description

TCustomLocateFileService (see page 6181) calls FindTemplateFile internally to generate an OnFindTemplateFile (see page 6187) event.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the name of the file to locate.

AResult returns the fully-qualified path name for the requested file.

FindTemplateFile returns true if it supplies a file name as the value of the AResult parameter. It returns false if there is no OnFindTemplateFile (see page 6187) event handler or if the event handler does not return a value for AResult.

See Also

OnFindTemplateFile (see page 6187)

60.1.9.2.4 TCustomLocateFileService.GetFileStream Method

Returns a stream object from which to read a specified file.

Pascal

```
function GetFileStream(const AFileName: string; out AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream GetFileStream(const UnicodeString AFileName, Boolean AOwned);
```

Description

The GetTemplateStream (see page 6184) method calls GetFileStream internally to obtain the stream it returns when the requested file maps to a fully-qualified path name.

AFileName is the fully-qualified name of a file whose contents can be read from the returned stream.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetFileStream returns a stream object from which to read the contents of AFileName.

See Also

GetTemplateStream (see page 6184)

60.1.9.2.5 TCustomLocateFileService.GetTemplateStream Method

Returns a stream from which to read a template file.

Pascal

```
function GetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned: Boolean):
TStream;
```

C++

```
__fastcall TStream GetTemplateStream(TComponent * AComponent, UnicodeString AFileName, Boolean
AOwned);
```

Description

Producer components call GetTemplateStream to read the contents of the template files they use to generate their content. GetTemplateStream locates the specified template file (which can be a logical rather than a physical file name), and returns a stream from which the caller can read the file contents.

GetTemplateStream calls the ImplGetTemplateStream (see page 6185) method, which generates first an OnFindStream (see page 6187) event, and, if OnFindStream (see page 6187) is unsuccessful, an OnFindTemplateFile (see page 6187) event.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the logical name of the file to locate.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetTemplateStream returns a stream from which to read the requested file, or, if it could not locate the file, nil (Delphi) or NULL (C++).

See Also

ImplGetTemplateStream (see page 6185)

OnFindStream (see page 6187)

OnFindTemplateFile (see page 6187)

60.1.9.2.6 TCustomLocateFileService.ImplFindIncludeFile Method

Provides the underlying implementation for the FindIncludeFile (see page 6182) method.

Pascal

```
function ImplFindIncludeFile(AComponent: TComponent; const AFileName: string; var AResult:
string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplFindIncludeFile(TComponent * AComponent, const UnicodeString
AFileName, UnicodeString AResult);
```

Description

The FindIncludeFile (see page 6182) method calls ImplFindIncludeFile to locate the fully-qualified path name for an include file to a server-side script. ImplFindIncludeFile generates an OnFindIncludeFile (see page 6186) event to obtain the fully-qualified path name.

AComponent is the Web module context object that stores the reference to the include file.

AFileName is the name of the include file to locate.

AResult returns the fully-qualified path name where the WebSnap application can find the requested file.

FindIncludeFile (see page 6182) returns true if it locates the requested file and returns its fully-qualified name as AResult. It returns false if there is no OnFindIncludeFile (see page 6186) event handler or if the event handler does not return a value for AResult.

See Also

OnFindIncludeFile (see page 6186)

FindIncludeFile (see page 6182)

WebModuleContext

60.1.9.2.7 TCustomLocateFileService.ImplGetTemplateStream Method

Provides the underlying implementation for the GetTemplateStream (see page 6184) method.

Pascal

```
function ImplGetTemplateStream(AComponent: TComponent; AFileName: string; out AOwned:
Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream ImplGetTemplateStream(TComponent * AComponent, UnicodeString
AFileName, Boolean AOwned);
```

Description

The GetTemplateStream (see page 6184) method calls ImplGetTemplateStream to obtain a stream from which to read a specified template. ImplGetTemplateStream generates an OnFindStream (see page 6187) event, and, if OnFindStream (see page 6187) is unsuccessful, an OnFindTemplateFile (see page 6187) event. If it gets a file name from the OnFindTemplateFile (see page 6187) event handler, ImplGetTemplateStream calls the GetFileStream (see page 6184) method to obtain a stream object from which to read the file.

AComponent is the object that uses or is associated with the requested template. Typically, this is a content producer.

AFileName is the logical name of the file to locate.

AOwned indicates whether the caller should free the stream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

GetTemplateStream (see page 6184) returns a stream from which to read the requested file, or, if it could not locate the file, nil (Delphi) or NULL (C++).

See Also

GetTemplateStream (see page 6184)

OnFindStream (see page 6187)

OnFindTemplateFile (see page 6187)

GetFileStream (see page 6184)

60.1.9.3 TCustomLocateFileService Events

The events of the TCustomLocateFileService class are listed here.

Protected Events

	Name	Description
	OnFindIncludeFile (see page 6186)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6187)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6187)	Occurs when the WebSnap application tries to locate the template file for a content producer.

60.1.9.3.1 TCustomLocateFileService.OnFindIncludeFile Event

Occurs when the WebSnap application tries to locate an include file for a server-side script.

Pascal

```
property OnFindIncludeFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindIncludeFile;
```

Description

Write (see page 3707) an OnFindIncludeFile event handler to indicate the location on disk of include files for server-side scripts. In WebSnap applications that include a locate file service component, you must supply either an OnFindIncludeFile or an OnFindStream (see page 6187) event handler (or both) if the server-side scripts have include files. Otherwise the application can't locate these include files at all.

See Also

OnFindStream (see page 6187)

OnFindTemplateFile (see page 6187)

60.1.9.3.2 TCustomLocateFileService.OnFindStream Event

Occurs when the WebSnap application tries to locate a file.

Pascal

```
property OnFindStream: TLocateFileServiceFindStreamEvent;
```

C++

```
__property TLocateFileServiceFindStreamEvent OnFindStream;
```

Description

Write (see page 3707) an OnFindStream event handler to supply the WebSnap application with the contents of a file given its name. This event is particularly useful for applications that use "logical" files that do not exist physically on disk. When the WebSnap application tries to locate a file, the locate file service generates this event, along with an OnFindTemplateFile (see page 6187). One of these event handlers must be provided in order for the application to use the requested file.

See Also

OnFindIncludeFile (see page 6186)

OnFindTemplateFile (see page 6187)

60.1.9.3.3 TCustomLocateFileService.OnFindTemplateFile Event

Occurs when the WebSnap application tries to locate the template file for a content producer.

Pascal

```
property OnFindTemplateFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindTemplateFile;
```

Description

Write (see page 3707) an OnFindIncludeFile (see page 6186) event handler to indicate the location on disk of templates that a content producer component uses to generate its content. In WebSnap applications that include a locate file service component, you must supply either an OnFindTemplateFile or an OnFindStream (see page 6187) event handler (or both) if the application has content producers that use template files. Otherwise content producers can't locate their templates.

See Also

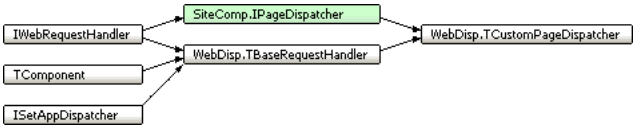
OnFindStream (see page 6187)

OnFindIncludeFile (see page 6186)

60.1.10 TCustomPageDispatcher Class

Dispatches incoming HTTP request messages in a WebSnap application.

Class Hierarchy



Pascal

```
TCustomPageDispatcher = class(TBaseRequestHandler, IPageDispatcher);
```

C++

```
class TCustomPageDispatcher : public TBaseRequestHandler, public IPageDispatcher;
```

File

WebDisp

Description

TCustomPageDispatcher is the base class for components that locate the appropriate page module for handling an incoming HTTP message and retrieve the content that the identified page module creates for the corresponding response message. Typically, the WebSnap application wizard adds a TCustomPageDispatcher descendant to the Web application module that it creates.

A WebSnap application only requires a single page dispatcher. The dispatcher examines all incoming HTTP request messages, checking the pathInfo portion of the URL to which a request is addressed for the name of a registered Web page module. It then forwards the request to the matching Web page module so that the page module can handle the request and produce the content of a response. If the request is not addressed to an URL with a pathinfo portion, the page dispatcher forwards the request to the Web page module identified by its DefaultPage (see page 6195) property.

See Also

TPageDispatcher (see page 6236)

60.1.10.1 TCustomPageDispatcher Members

The following tables list the members exposed by TCustomPageDispatcher.

Protected Methods

	Name	Description
	DispatchPageName (see page 6190)	Checks whether a specified user has a given access rights string.
	EndUser (see page 6190)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 6191)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 6191)	Returns the value of the DefaultPageName property.
	GetLoginRequired (see page 6191)	Indicates whether a user must log in before accessing a specified Web page.
	GetRequestedPage (see page 6192)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	HandleRequest (see page 6167)	Handles an HTTP request.
	ImplDispatchPageName (see page 6192)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
V	ImplGetCanView (see page 6192)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
V	ImplGetDefaultPageName (see page 6192)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.

	ImplGetLoginRequired (see page 6193)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
	ImplHandleRequest (see page 6193)	Handles application-specific processing of an HTTP request.
	ImplPageNameOfRequest (see page 6193)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 6193)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	IsLoggedIn (see page 6194)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	PageNameOfRequest (see page 6194)	Returns the name of the Web page to which an HTTP request is addressed.
	ProduceLoginPage (see page 6194)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	RedirectToPageName (see page 6194)	Fills out a response message so that it redirects the caller to another page in the application.
	SetAppDispatcher (see page 6167)	Sets the AppDispatcher (see page 6168) property.
	ValidatePage (see page 6195)	This is ValidatePage, a member of class TCustomPageDispatcher.

Protected Properties

	Name	Description
	AppDispatcher (see page 6168)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6196)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnAfterRedirectToPage (see page 6197)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeDispatchPage (see page 6197)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnBeforeRedirectToPage (see page 6198)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.
	OnCanViewPage (see page 6198)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6199)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6200)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6200)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Public Properties

	Name	Description
	DefaultPage (see page 6195)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.
	DefaultPageName (see page 1064)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

60.1.10.2 TCustomPageDispatcher Methods

The methods of the TCustomPageDispatcher class are listed here.

Protected Methods

	Name	Description
	DispatchPageName (see page 6190)	Checks whether a specified user has a given access rights string.
	EndUser (see page 6190)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 6191)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 6191)	Returns the value of the DefaultPageName property.
	GetLoginRequired (see page 6191)	Indicates whether a user must log in before accessing a specified Web page.
	GetRequestedPage (see page 6192)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	ImplDispatchPageName (see page 6192)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
	ImplGetCanView (see page 6192)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
	ImplGetDefaultPageName (see page 6192)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.

	ImplGetLoginRequired (see page 6193)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
	ImplHandleRequest (see page 6193)	Handles application-specific processing of an HTTP request.
	ImplPageNameOfRequest (see page 6193)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 6193)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	IsLoggedIn (see page 6194)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	PageNameOfRequest (see page 6194)	Returns the name of the Web page to which an HTTP request is addressed.
	ProduceLoginPage (see page 6194)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	RedirectToPageName (see page 6194)	Fills out a response message so that it redirects the caller to another page in the application.
	ValidatePage (see page 6195)	This is ValidatePage, a member of class TCustomPageDispatcher.

60.1.10.2.1 TCustomPageDispatcher.DispatchPageName Method

Checks whether a specified user has a given access rights string.

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean DispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

Call DispatchPageName to fill out a response message with the Web page that a specified Web page module provides. DispatchPageName locates the Web page module that produces the requested Web page and uses it to fill out the content of a response object.

APageName is the name of the Web page to fetch. If APageName is an empty string, DispatchPageName uses the first Web page module that is registered with the current Web context.

AResponse is the response object whose Content property receives the requested Web page.

AFlags indicates the attributes that must be present on the Web page module. For example, if AFlags includes dpPublished, APageName must specify a published Web page module or the page dispatcher won't fetch the Web page.

DispatchPageName returns true if it successfully fetches the requested page, false if it fails to obtain the requested page.

See Also

RedirectToPageName (see page 1063)

HandleRequest

Content

WebContext (see page 6003)

60.1.10.2.2 TCustomPageDispatcher.EndUser Method

Pascal

```
function EndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser EndUser();
```

Description

This is EndUser, a member of class TCustomPageDispatcher.

60.1.10.2.3 TCustomPageDispatcher.GetCanView Method

Indicates whether the current user is permitted to view a specified Web page.

Pascal

```
function GetCanView(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetCanView(const UnicodeString APageName);
```

Description

Call GetCanView to check whether the current user should be allowed to view a specified Web page. Typically, this method is called after the user logs in.

APageName is the name of the Web page the user wants to view.

See Also

GetLoginRequired (see page 1062)

HasViewRights (see page 5992)

OnCanViewPage (see page 6198)

60.1.10.2.4 TCustomPageDispatcher.GetDefaultPageName Method

Returns the value of the DefaultPageName property.

Pascal

```
function GetDefaultPageName: string;
```

C++

```
__fastcall UnicodeString GetDefaultPageName();
```

Description

GetDefaultPageName is the read implementation of the DefaultPageName property.

See Also

DefaultPageName (see page 1064)

60.1.10.2.5 TCustomPageDispatcher.GetLoginRequired Method

Indicates whether a user must log in before accessing a specified Web page.

Pascal

```
function GetLoginRequired(const APageName: string): Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired(const UnicodeString APageName);
```

Description

Call GetLoginRequired to check whether users must log in before accessing a specified Web page. GetLoginRequired can check the Web page information object for the target Web page, but can also use other means (such as an event handler) to obtain this information.

APageName is the name of the Web page to check whether it requires users to first log in.

See Also

GetCanView (see page 1062)

IsLoginRequired

OnIsLoginRequired (see page 6199)

60.1.10.2.6 TCustomPageDispatcher.GetRequestedPage Method

Pascal

```
function GetRequestedPage(const APageName: string; AFlags: TDispatchPageFlags): TObject;
```

C++

```
__fastcall TObject * GetRequestedPage(const UnicodeString APageName, TDispatchPageFlags
AFlags);
```

Description

This is GetRequestedPage, a member of class TCustomPageDispatcher.

60.1.10.2.7 TCustomPageDispatcher.ImplDispatchPageName Method

Pascal

```
function ImplDispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean ImplDispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

Description

This is ImplDispatchPageName, a member of class TCustomPageDispatcher.

60.1.10.2.8 TCustomPageDispatcher.ImplGetCanView Method

Pascal

```
function ImplGetCanView(const APageName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetCanView(const UnicodeString APageName);
```

Description

This is ImplGetCanView, a member of class TCustomPageDispatcher.

60.1.10.2.9 TCustomPageDispatcher.ImplGetDefaultPageName Method

Pascal

```
function ImplGetDefaultPageName: string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplGetDefaultPageName();
```

Description

This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.

60.1.10.2.10 TCustomPageDispatcher.ImplGetLoginRequired Method

Pascal

```
function ImplGetLoginRequired(const APageName: string): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplGetLoginRequired(const UnicodeString APageName);
```

Description

This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.

60.1.10.2.11 TCustomPageDispatcher.ImplHandleRequest Method

Handles application-specific processing of an HTTP request.

Pascal

```
function ImplHandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

The ImplHandleRequest function is called from HandleRequest to handle the application-specific processing of the HTTP request.

This base version of the function generates an assertion failure as a reminder that a customized overridden version of this function needs to be defined in order to handle the request processing specific to the application.

See Also

HandleRequest (see page 6167)

60.1.10.2.12 TCustomPageDispatcher.ImplPageNameOfRequest Method

Pascal

```
function ImplPageNameOfRequest(Request: TWebRequest): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplPageNameOfRequest(TWebRequest Request);
```

Description

This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.

60.1.10.2.13 TCustomPageDispatcher.ImplRedirectToPageName Method

Pascal

```
function ImplRedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean ImplRedirectToPageName(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.

60.1.10.2.14 TCustomPageDispatcher.IsLoggedIn Method

Pascal

```
function IsLoggedIn: Boolean;
```

C++

```
__fastcall Boolean IsLoggedIn();
```

Description

This is IsLoggedIn, a member of class TCustomPageDispatcher.

60.1.10.2.15 TCustomPageDispatcher.PageNameOfRequest Method

Returns the name of the Web page to which an HTTP request is addressed.

Pascal

```
function PageNameOfRequest(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString PageNameOfRequest(TWebRequest Request);
```

Description

Call PageNameOfRequest to obtain the name of the Web page that a request message is trying to fetch.

ARequest is the request object that describes the HTTP request message.

PageNameOfRequest returns the last string in the path info portion of the request message's target URL.

See Also

DefaultPageName (see page 1064)

60.1.10.2.16 TCustomPageDispatcher.ProduceLoginPage Method

Pascal

```
function ProduceLoginPage: Boolean;
```

C++

```
__fastcall Boolean ProduceLoginPage();
```

Description

This is ProduceLoginPage, a member of class TCustomPageDispatcher.

60.1.10.2.17 TCustomPageDispatcher.RedirectToPageName Method

Fills out a response message so that it redirects the caller to another page in the application.

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
__fastcall Boolean RedirectToPageName(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse, TDispatchPageFlags AFlags);
```

Description

Call RedirectToPageName to generate a response that redirects the caller to another Web page in the WebSnap application.

APageName is the name of the Web page to which the request should be redirected. If APageName is an empty string, RedirectToPageName uses the first Web page module that is registered with the current Web context.

AParams is a string list containing application-defined name-value pairs. These name-value pairs supply the query fields of the redirected request message, and are added to the URL that is the target of the SendRedirect call.

AResponse is the response object that the WebSnap uses to make the SendRedirect call.

AFlags indicates the attributes that must be present on the Web page module to which the current request is redirected. For example, if AFlags includes dpPublished, APageName must specify a published Web page module.

RedirectToPageName returns true if it successfully redirects to the specified page.

See Also

DispatchPageName (see page 1061)

HandleRequest

Content

WebContext (see page 6003)

60.1.10.2.18 TCustomPageDispatcher.ValidatePage Method

Pascal

```
function ValidatePage(const APage: string; var AHandled: Boolean): Boolean;
```

C++

```
__fastcall Boolean ValidatePage(const UnicodeString APage, Boolean AHandled);
```

Description

This is ValidatePage, a member of class TCustomPageDispatcher.

60.1.10.3 TCustomPageDispatcher Properties

The properties of the TCustomPageDispatcher class are listed here.

Public Properties

	Name	Description
	DefaultPage (see page 6195)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

60.1.10.3.1 TCustomPageDispatcher.DefaultPage Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPage: string;
```

C++

```
__property UnicodeString DefaultPage;
```

Description

DefaultPage is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPage is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a Web Page module.

60.1.10.4 TCustomPageDispatcher Events

The events of the TCustomPageDispatcher class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6196)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnAfterRedirectToPage (see page 6197)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeDispatchPage (see page 6197)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnBeforeRedirectToPage (see page 6198)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.
	OnCanViewPage (see page 6198)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6199)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6200)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6200)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

60.1.10.4.1 TCustomPageDispatcher.OnAfterDispatchPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write (see page 3707) an OnAfterDispatchPage event handler to take specific action after an HTTP request has been successfully handled by a Web page module. For example, an OnAfterDispatchPage event handler can maintain a log of all Web traffic handled by the WebSnap application.

OnAfterDispatchPage occurs only if the target Web page module succeeds in handling the request. If the target Web page module could not be validated, an OnPageAccessDenied (see page 6200) event occurs instead. If the page could not be located, an OnPageNotFound (see page 6200) event occurs instead.

See Also

OnBeforeDispatchPage (see page 6197)

OnPageNotFound (see page 6200)

OnCanViewPage (see page 6198)

OnIsLoginRequired (see page 6199)

OnPageAccessDenied (see page 6200)

60.1.10.4.2 TCustomPageDispatcher.OnAfterRedirectToPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

Write (see page 3707) an OnAfterRedirectToPage event handler to take specific action after an HTTP request has been successfully redirected to a Web page module. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost), and the new target Web page module succeeds in handling the request.

See Also

OnBeforeRedirectToPage (see page 6198)

OnAfterDispatchPage (see page 6196)

RedirectOptions (see page 5476)

60.1.10.4.3 TCustomPageDispatcher.OnBeforeDispatchPage Event

Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write (see page 3707) an OnBeforeDispatchPage event handler to process an HTTP request message before the dispatcher forwards it to the target Web page module for processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the page dispatcher from locating the target Web page module, verifying that it can handle the request, and forwarding the request to that Web page module.

If the event handler does not handle the request (for example, if it is merely maintaining a log of incoming requests), it can leave the Handled parameter as false and the dispatcher will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, call the global WebContext (see page 6003) function to access the Web context and read its Request and Response properties.

See Also

OnAfterDispatchPage (see page 6196)

OnPageNotFound (see page 6200)

OnCanViewPage (see page 6198)

OnIsLoginRequired (see page 6199)

OnPageAccessDenied (see page 6200)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

60.1.10.4.4 TCustomPageDispatcher.OnBeforeRedirectToPage Event

Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

Write (see page 3707) an OnBeforeRedirectToPage event handler to take specific action before an HTTP request is redirected to a Web page module. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost).

If the event handler sets Handled to true, the dispatcher assumes it has handled the request and performs no further processing. Otherwise, after the event handler exits, the page dispatcher calls on the Web Module to set up the response message and call SendRedirect.

Tip: To access the Web request and response objects from within an OnBeforeRedirectToPage event handler, call the global WebContext (see page 6003) function to access the Web context and read its Request and Response properties.

See Also

OnAfterRedirectToPage (see page 6197)

OnBeforeDispatchPage (see page 6197)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

SendRedirect

60.1.10.4.5 TCustomPageDispatcher.OnCanViewPage Event

Occurs just before the dispatcher passes a request message to a Web page module.

Pascal

```
property OnCanViewPage: TCanViewPageEvent;
```

C++

```
__property TCanViewPageEvent OnCanViewPage;
```

Description

Write (see page 3707) an OnCanViewPage event handler to customize the way the WebSnap application determines whether a given request can be passed to a Web page module for processing. This event is part of the mechanism by which the dispatcher validates a Web page for viewing. The validation process, which occurs after the OnBeforeDispatchPage (see page 6197) event, occurs as follows:

1. If the caller has not already logged in, the dispatcher generates an OnIsLoginRequired (see page 6199) event to check whether the specified Web page requires the caller to log in.
2. If a login is required, the dispatcher does not validate the page (and so does not forward the request to the specified Web page module). Rather, it initiates the process of creating a response to allow the user to log in.

a. First, the dispatcher generates an OnPageAccessDenied (see page 6200) event, where the application can handle the login process.

b. If the OnPageAccessDenied (see page 6200) event handler does not handle the login process, the dispatcher forwards the request to the login page specified by the end user adapter's LoginPage property.

3. If a login is not required (either because the caller already logged in or because there is no need to log in), the dispatcher generates an OnCanViewPage event. This event can validate a caller that has logged in, or provide any other type of validation that governs access to the requested page.
4. If there is no OnCanViewPage event handler, or if the OnCanViewPage event handler does not handle the validation, the dispatcher calls on a TEndUserAdapter component to check the user's access rights.

   Tip: To access the Web request and response objects from within an OnCanViewPage event handler, call the global WebContext (see page 6003) function to access the Web context and read its Request and Response properties. To access information about the access rights that the target page requires, call the Web context's FindPageInfo method to obtain a Web page info object for the target page and check its ViewAccess property.

See Also

OnIsLoginRequired (see page 6199)

OnBeforeDispatchPage (see page 6197)

OnPageAccessDenied (see page 6200)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

ViewAccess

FindPageInfo (see page 5966)

60.1.10.4.6 TCustomPageDispatcher.OnIsLoginRequired Event

Occurs when a request arrives from a caller that has not yet logged in.

Pascal

```
property OnIsLoginRequired: TLoginRequiredEvent;
```

C++

```
__property TLoginRequiredEvent OnIsLoginRequired;
```

Description

Write (see page 3707) an OnIsLoginRequired event handler to customize the way the WebSnap application determines whether a caller must log in before obtaining access to a requested page. This event occurs after the OnBeforeDispatchPage (see page 6197) event if the caller has not already logged in to the WebSnap application.

If there is no OnPageAccessDenied (see page 6200) event handler, or if the OnPageAccessDenied (see page 6200) event handler does not handle the login process, the dispatcher obtains a response message from the login page specified by the end user adapter.

See Also

OnCanViewPage (see page 6198)

OnBeforeDispatchPage (see page 6197)

OnPageAccessDenied (see page 6200)

LoginPage (see page 5516)

60.1.10.4.7 TCustomPageDispatcher.OnPageAccessDenied Event

Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.

Pascal

```
property OnPageAccessDenied: TPageAccessDeniedEvent;
```

C++

```
__property TPageAccessDeniedEvent OnPageAccessDenied;
```

Description

Write (see page 3707) an OnPageAccessDenied event handler to customize the response when a request for a specific Web page fails to be validated.

Tip: To access the Web request and response objects from within an OnPageAccessDenied event handler, call the global WebContext (see page 6003) function to access the Web context and read its Request and Response properties.

See Also

OnIsLoginRequired (see page 6199)

OnCanViewPage (see page 6198)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

LoginPage (see page 5516)

60.1.10.4.8 TCustomPageDispatcher.OnPageNotFound Event

Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Pascal

```
property OnPageNotFound: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnPageNotFound;
```

Description

Write (see page 3707) an OnPageNotFound event handler to generate a response when the dispatcher can't locate the Web page module to which an HTTP request message is addressed. For example, an OnPageNotFound event handler can provide the response object with content that indicates that the requested page does not exist or that represents simple content for a page that does not have an associated Web page module.

If the OnPageNotFound event handler sets its Handled parameter to true, the page dispatcher assumes it has updated the response message and returns it. If the OnPageNotFound leaves the Handled parameter set to false, the dispatcher raises an EPageDispatcherException (see page 6147) exception after the event handler exits.

Tip: To access the Web request and response objects from within an OnPageNotFound event handler, call the global WebContext (see page 6003) function to access the Web context and read its Request and Response properties.

See Also

OnBeforeDispatchPage (see page 6197)

OnAfterDispatchPage (see page 6196)

OnPageAccessDenied (see page 6200)

EPageDispatcherException (see page 6147)

WebContext (see page 6003)

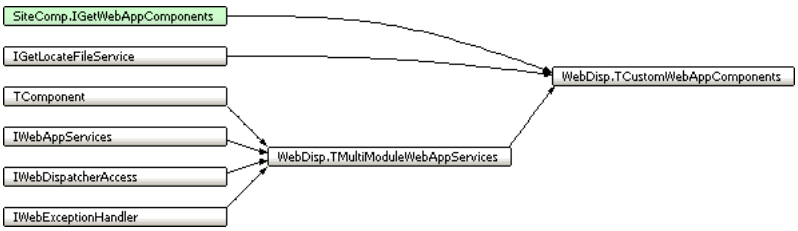
Request (see page 5974)

Response (see page 5975)

60.1.11 TCustomWebAppComponents Class

TCustomWebAppComponents manages the components that provide a WebSnap application's functionality.

Class Hierarchy



Pascal

```
TCustomWebAppComponents = class(TMultiModuleWebAppServices, IGetWebAppComponents,
IGetLocateFileService);
```

C++

```
class TCustomWebAppComponents : public TMultiModuleWebAppServices, public
IGetWebAppComponents, public IGetLocateFileService;
```

File

WebDisp

Description

Use TCustomWebAppComponents as a base class for the component that centralizes access to the main WebSnap components. It keeps track of the current request and response objects, maintains a list of the available Web modules, and locates the appropriate components to handle incoming request messages. TCustomWebAppComponents introduces properties that provide access to the components that perform many common tasks in a WebSnap application. These properties include:

PageDispatcher (see page 6214), which locates the Web page modules to process incoming request messages.

AdapterDispatcher (see page 6211), which encodes calls to adapter actions and fields, and which handles HTML form submissions and requests for dynamic images by calling adapter actions and fields.

Sessions (see page 6214), which manages the sessions for each caller.

ApplicationAdapter (see page 6212), which implements the fields and actions of the Application variable in server-side scripts.

EndUserAdapter (see page 6213), which implements the fields and actions of the EndUser variable in server-side scripts.

UserListService (see page 6215), which maintains a list of end users and their passwords and access privileges.

LocateFileService (see page 6213), which controls access to the templates and server-side script include files that are used while processing requests.

DispatchActions (see page 6212), which maintains a list of action items that can respond to request messages.

The components these properties reference are optional. The WebSnap Application wizard automatically creates components and fill in these properties for any components that you select.

See Also

TApplicationAdapter (see page 5439)

TEndUserAdapter (see page 5764)

TPageDispatcher (see page 6236)

TAdapterDispatcher (see page 6148)

TLocateFileService (see page 6220)

TSessionsService (see page 6756)

TWebUserList (see page 6796)

WebContext (see page 6003)

60.1.11.1 TCustomWebAppComponents Members

The following tables list the members exposed by TCustomWebAppComponents.

Protected Events

	Name	Description
	OnAfterDispatch (see page 6235)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6235)	Occurs before TMultiModuleWebAppServices (see page 6223) passes an HTTP request onto the other components in the application for processing.

Protected Methods

	Name	Description
	Access_Request (see page 6225)	Returns the value of the Access_Request property.
	Access_Response (see page 6225)	Returns the value of the Access_Response property.
V	CreateContext (see page 6204)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6226)	Generates an OnAfterDispatch (see page 6235) event.
	DoBeforeDispatch (see page 6226)	Generates an OnBeforeDispatch (see page 6235) event.
V	DoneModule (see page 6227)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6227)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetAdapterDispatcher (see page 6205)	Returns the value of the AdapterDispatcher (see page 6211) property.
	GetExceptionHandler (see page 6228)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetLocateFileService (see page 6205)	Returns the value of the LocateFileService (see page 6213) property.
	GetPageDispatcher (see page 6205)	Returns the value of the PageDispatcher (see page 6214) property.
	GetWebApplicationInfo (see page 6206)	Returns the value of the ApplicationAdapter (see page 6212) property.
	GetWebEndUser (see page 6206)	Returns the value of the EndUserAdapter (see page 6213) property.
	GetWebUserList (see page 6206)	Returns the value of the UserListService (see page 6215) property.
	HandleException (see page 6228)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6229)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6207)	Provides the underlying implementation of the FinishContext (see page 6227) method.
V	ImplHandleException (see page 6207)	Provides the underlying implementation of the HandleException (see page 6228) method.
V	ImplHandleRequest (see page 6208)	Provides the underlying implementation for the HandleRequest (see page 6229) method.
V	ImplInitContext (see page 6208)	Provides the underlying implementation for the InitContext (see page 6231) method.
	InitContext (see page 6231)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 6209)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6210)	Initializes all the currently active Web modules.
V	ListRequestHandlers (see page 6210)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
	ModuleAdded (see page 6233)	Initializes a Web module when it is activated while processing the current request.
V	Notification (see page 6211)	Updates the TCustomWebAppComponents (see page 6201) properties when components are added to or removed from the application.

Protected Properties

	Name	Description
R	Request (see page 6234)	Provides access to the current request object.
R	Response (see page 6234)	Provides access to the current response object.
R	WebModules (see page 6234)	Lists the Web modules available in the WebSnap application.

Public Events

	Name	Description
	OnException (see page 6215)	Occurs when an exception escapes all try blocks in the WebSnap application.

Public Properties

	Name	Description
	AdapterDispatcher (see page 6211)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6212)	Provides access to the application adapter.
	DispatchActions (see page 6212)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6213)	Provides access to an end-user adapter.
	LocateFileService (see page 6213)	Provides access to the file location service.
	PageDispatcher (see page 6214)	Provides access to the current page dispatcher object.
	Sessions (see page 6214)	Provides access to the sessions service.

	UserListService (see page 6215)	Provides access to the file location service.
R	WebApplicationInfo (see page 1039)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
R	WebEndUser (see page 1040)	This is WebEndUser, a member of class IGetWebAppComponents.

60.1.11.2 TCustomWebAppComponents Methods

The methods of the TCustomWebAppComponents class are listed here.

Protected Methods

	Name	Description
V	CreateContext (see page 6204)	Creates the Web context for the current request message.
	GetAdapterDispatcher (see page 6205)	Returns the value of the AdapterDispatcher (see page 6211) property.
	GetLocateFileService (see page 6205)	Returns the value of the LocateFileService (see page 6213) property.
	GetPageDispatcher (see page 6205)	Returns the value of the PageDispatcher (see page 6214) property.
	GetWebApplicationInfo (see page 6206)	Returns the value of the ApplicationAdapter (see page 6212) property.
	GetWebEndUser (see page 6206)	Returns the value of the EndUserAdapter (see page 6213) property.
	GetWebUserList (see page 6206)	Returns the value of the UserListService (see page 6215) property.
V	ImplFinishContext (see page 6207)	Provides the underlying implementation of the FinishContext (see page 6227) method.
V	ImplHandleException (see page 6207)	Provides the underlying implementation of the HandleException (see page 6228) method.
V	ImplHandleRequest (see page 6208)	Provides the underlying implementation for the HandleRequest (see page 6229) method.
V	ImplInitContext (see page 6208)	Provides the underlying implementation for the InitContext (see page 6231) method.
V	InitModule (see page 6209)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6210)	Initializes all the currently active Web modules.
V	ListRequestHandlers (see page 6210)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
V	Notification (see page 6211)	Updates the TCustomWebAppComponents (see page 6201) properties when components are added to or removed from the application.

60.1.11.2.1 TCustomWebAppComponents.CreateContext Method

Creates the Web context for the current request message.

Pascal

```
function CreateContext: TAbstractWebContext; override;
```

C++

```
virtual __fastcall TAbstractWebContext CreateContext();
```

Description

The ImplInitContext (see page 6208) method calls CreateContext to instantiate the Web context object for the current request. This context can then be accessed by the WebSnap application using the global WebContext (see page 6003) function.

In TCustomWebAppComponents (see page 6201), CreateContext creates a TWebContext (see page 5976) object, creating a new session object for the current request and passing the session object along with the values of the protected Request (see page 6234), Response (see page 6234), and WebModules (see page 6234) properties to the TWebContext (see page 5976) constructor.

See Also

WebContext (see page 6003)

ImplInitContext (see page 6208)

Create (see page 5978)

Request (see page 6234)

Response (see page 6234)

WebModules (see page 6234)

60.1.11.2.2 TCustomWebAppComponents.GetAdapterDispatcher Method

Returns the value of the AdapterDispatcher (see page 6211) property.

Pascal

```
function GetAdapterDispatcher: IAdapterDispatcher;
```

C++

```
__fastcall IAdapterDispatcher GetAdapterDispatcher();
```

Description

The protected GetAdapterDispatcher method is used internally to fetch the value of the AdapterDispatcher (see page 6211) property.

See Also

AdapterDispatcher (see page 6211)

60.1.11.2.3 TCustomWebAppComponents.GetLocateFileService Method

Returns the value of the LocateFileService (see page 6213) property.

Pascal

```
function GetLocateFileService: ILocateFileService;
```

C++

```
__fastcall ILocateFileService GetLocateFileService();
```

Description

The protected GetLocateFileService method is used internally to fetch the value of the LocateFileService (see page 6213) property.

See Also

LocateFileService (see page 6213)

60.1.11.2.4 TCustomWebAppComponents.GetPageDispatcher Method

Returns the value of the PageDispatcher (see page 6214) property.

Pascal

```
function GetPageDispatcher: IPageDispatcher;
```

C++

```
__fastcall IPageDispatcher GetPageDispatcher();
```

Description

The protected GetPageDispatcher method is used internally to fetch the value of the PageDispatcher (see page 6214) property.

See Also

PageDispatcher (see page 6214)

60.1.11.2.5 TCustomWebAppComponents.GetWebApplicationInfo Method

Returns the value of the ApplicationAdapter (see page 6212) property.

Pascal

```
function GetWebApplicationInfo: IWebApplicationInfo;
```

C++

```
__fastcall IWebApplicationInfo GetWebApplicationInfo();
```

Description

The protected GetWebApplicationInfo method is used internally to fetch the value of the ApplicationAdapter (see page 6212) property.

See Also

ApplicationAdapter (see page 6212)

60.1.11.2.6 TCustomWebAppComponents.GetWebEndUser Method

Returns the value of the EndUserAdapter (see page 6213) property.

Pascal

```
function GetWebEndUser: IWebEndUser;
```

C++

```
__fastcall IWebEndUser GetWebEndUser();
```

Description

The protected GetWebEndUser method is used internally to fetch the value of the EndUserAdapter (see page 6213) property.

See Also

EndUserAdapter (see page 6213)

60.1.11.2.7 TCustomWebAppComponents.GetWebUserList Method

Returns the value of the UserListService (see page 6215) property.

Pascal

```
function GetWebUserList: IWebUserList;
```

C++

```
__fastcall IWebUserList GetWebUserList();
```

Description

The protected GetWebUserList method is used internally to fetch the value of the UserListService (see page 6215) property.

See Also

UserListService (see page 6215)

60.1.11.2.8 TCustomWebAppComponents.ImplFinishContext Method

Provides the underlying implementation of the FinishContext (see page 6227) method.

Pascal

```
procedure ImplFinishContext; override;
```

C++

```
virtual __fastcall ImplFinishContext();
```

Description

ImplFinishContext performs any final tasks and cleanup after the WebSnap application has finished processing a request message.

ImplFinishContext

1. Instructs the Web dispatcher that implements the DispatchActions (see page 6212) property to perform any final tasks.
2. Calls the DoneModule (see page 6227) method for every active Web module so that it can be freed or deactivated.
3. Frees the Web context for the current request.
4. Adds the session ID of the current session to the response message so that subsequent requests from the same caller can use the same session.

See Also

FinishContext (see page 6227)

DoneModule (see page 6227)

WebContext (see page 6003)

DispatchActions (see page 6212)

Sessions (see page 6214)

60.1.11.2.9 TCustomWebAppComponents.ImplHandleException Method

Provides the underlying implementation of the HandleException (see page 6228) method.

Pascal

```
procedure ImplHandleException(E: Exception; var Handled: Boolean); override;
```

C++

```
virtual __fastcall ImplHandleException(Exception E, Boolean Handled);
```

Description

The HandleException (see page 6228) method calls ImplHandleException to handle any exception that passes through all the try blocks in the application code.

ImplHandleException generates an OnException (see page 6215) event, so that an event handler can take care of the exception. If the OnException (see page 6215) event handler does not handle the exception, ImplHandleException checks whether the response message has not yet been sent. If not, it adds the exception error message to the response.

E is the exception object for the exception that occurred.

Handled returns true if the OnException (see page 6215) event handler handled the exception or if ImplHandleException added the error message to the response object. If Handled returns false, the WebSnap application displays a dialog box that

indicates an error occurred.

See Also

HandleException (see page 6228)

OnException (see page 6215)

60.1.11.2.10 TCustomWebAppComponents.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 6229) method.

Pascal

```
function ImplHandleRequest: Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest();
```

Description

The HandleRequest (see page 6229) method calls ImplHandleRequest to handle the current HTTP request message.

ImpleHandleRequest

1.Generates an OnBeforeDispatch (see page 6235) event.

2.If the OnBeforeDispatch (see page 6235) event handler does not handle the request, checks the PageDispatcher (see page 6214), AdapterDispatcher (see page 6211), and DispatchActions (see page 6212) properties to find a dispatcher that can handle the request. If it finds one, the dispatcher forwards the request to the appropriate components for handling.

3.If the request was handled by the OnBeforeDispatch (see page 6235) event or by one of the dispatchers, and if the response was not already sent to the caller, generates an OnAfterDispatch (see page 6235) event.

ImplHandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

HandleRequest (see page 6229)

OnBeforeDispatch (see page 6235)

OnAfterDispatch (see page 6235)

PageDispatcher (see page 6214)

AdapterDispatcher (see page 6211)

DispatchActions (see page 6212)

60.1.11.2.11 TCustomWebAppComponents.ImplInitContext Method

Provides the underlying implementation for the InitContext (see page 6231) method.

Pascal

```
procedure ImplInitContext(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest;
AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall ImplInitContext(TAbstractWebModuleList AWebModuleList, TWebRequest
ARequest, TWebResponse AResponse);
```

Description

The InitContext (see page 6231) method, which is called when the WebSnap application first receives a request, calls ImplInitContext to perform any initializations and to create the Web context for the new request message.

ImplInitContext performs the following tasks:

1. If the DispatchActions (see page 6212) property is set, instructs the Web dispatcher to initialize itself.
2. Calls the CreateContext (see page 6204) method to create a Web context for the current request, supplying it with the values of the AWebModuleList, ARequest, and AResponse parameters. This Web context can then be accessed using the global WebContext (see page 6003) function.
3. Calls the InitModules (see page 6210) method to initialize all the Web modules that the AWebModuleList parameter indicates are currently instantiated. (These are the Web modules that are created for every request message)
4. Hooks up the InitModule (see page 6209) method so that it can initialize any Web modules that are instantiated in the course of processing the current request.

See Also

InitContext (see page 6231)

CreateContext (see page 6204)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

WebModules (see page 5975)

InitModules (see page 6210)

DispatchActions (see page 6212)

InitModule (see page 6209)

60.1.11.2.12 TCustomWebAppComponents.InitModule Method

Initializes a Web module before the WebSnap application uses it to handle the current request.

Pascal

```
procedure InitModule(AModule: TComponent); override;
```

C++

```
virtual __fastcall InitModule(TComponent * AModule);
```

Description

InitModule is called whenever the WebSnap application instantiates a Web module to handle the current request. InitModule handles any initializations, including assigning itself as the Web dispatcher to any content producers in the Web module, generating an OnActivate event on the Web module, and allowing the Web dispatcher that is the value of the DispatchActions (see page 6212) property to perform its own initializations.

AWebModule is the Web module component to initialize.

See Also

InitModules (see page 6210)

Dispatcher

DispatchActions (see page 6212)

OnActivate (see page 6556)

60.1.11.2.13 TCustomWebAppComponents.InitModules Method

Initializes all the currently active Web modules.

Pascal

```
procedure InitModules; override;
```

C++

```
virtual __fastcall InitModules();
```

Description

InitModules is called to initialize the currently active Web modules when the WebSnap application first sets up the Web context for a new request. It iterates over all the active Web modules that the WebModules (see page 6234) property lists, calling the InitModule (see page 6209) method to initialize each one. It then checks whether the session for the request is new (rather than a continuation of the session from a previous request), and if so, generates an OnStartSession event on the sessions service.

See Also

InitModule (see page 6209)

WebModules (see page 6234)

InitContext (see page 6231)

OnStartSession (see page 6755)

60.1.11.2.14 TCustomWebAppComponents.ListRequestHandlers Method

Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.

Pascal

```
procedure ListRequestHandlers(L: IInterfaceList); virtual;
```

C++

```
virtual __fastcall ListRequestHandlers(IInterfaceList L);
```

Description

The ImplHandleRequest (see page 6208) method calls ListRequestHandlers to obtain a list of all interfaces it can call to dispatch the request. ListRequestHandlers adds the interfaces that are the values of the PageDispatcher (see page 6214), AdapterDispatcher (see page 6211), and DispatchActions (see page 6212) properties to the list.

L is the interface list to which the dispatcher interfaces should be added.

See Also

ImplHandleRequest (see page 6208)

PageDispatcher (see page 6214)

AdapterDispatcher (see page 6211)

DispatchActions (see page 6212)

60.1.11.2.15 TCustomWebAppComponents.Notification Method

Updates the TCustomWebAppComponents (see page 6201) properties when components are added to or removed from the application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. It checks whether AComponent supports one of the interfaces that is the type of a TCustomWebAppComponents (see page 6201) property, and if so, sets that property to nil (Delphi) or NULL (C++) if the object is about to be removed, or sets the property to the AComponent interface if the object was just added.

See Also

Notification

AdapterDispatcher (see page 6211)

ApplicationAdapter (see page 6212)

DispatchActions (see page 6212)

EndUserAdapter (see page 6213)

LocateFileService (see page 6213)

Sessions (see page 6214)

PageDispatcher (see page 6214)

UserListService (see page 6215)

60.1.11.3 TCustomWebAppComponents Properties

The properties of the TCustomWebAppComponents class are listed here.

Public Properties

	Name	Description
	AdapterDispatcher (see page 6211)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6212)	Provides access to the application adapter.
	DispatchActions (see page 6212)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6213)	Provides access to an end-user adapter.
	LocateFileService (see page 6213)	Provides access to the file location service.
	PageDispatcher (see page 6214)	Provides access to the current page dispatcher object.
	Sessions (see page 6214)	Provides access to the sessions service.
	UserListService (see page 6215)	Provides access to the file location service.

60.1.11.3.1 TCustomWebAppComponents.AdapterDispatcher Property

Provides access to the current adapter dispatcher object.

Pascal

```
property AdapterDispatcher: IAdapterDispatcher;
```

C++

```
__property IAdapterDispatcher AdapterDispatcher;
```

Description

Use AdapterDispatcher to get or set the interface of the adapter dispatcher. The adapter dispatcher is responsible for encoding the hidden fields and HREF values that represent calls to adapter fields and actions, and for interpreting the fields in incoming requests by calling adapters to execute their actions or to fetch image field values.

Typically, AdapterDispatcher is the interface of a TAdapterDispatcher (see page 6148) object. If the WebSnap application does not include a TAdapterDispatcher (see page 6148) component, or some other component that implements the IAdapterDispatcher (see page 985) interface, the value of this property is nil (Delphi) or NULL (C++). When AdapterDispatcher is nil (Delphi) or NULL (C++), the WebSnap application can't execute adapter actions or fetch image fields from adapters.

See Also

TAdapterDispatcher (see page 6148)

DispatchActions (see page 6212)

60.1.11.3.2 TCustomWebAppComponents.ApplicationAdapter Property

Provides access to the application adapter.

Pascal

```
property ApplicationAdapter: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo ApplicationAdapter;
```

Description

Use ApplicationAdapter to get or set the interface of the application adapter. The application adapter implements the fields and actions of the Application variable in server-side scripts.

Typically, ApplicationAdapter is the interface of a TApplicationAdapter object. If the WebSnap application does not include a TApplicationAdapter component, or some other component that implements the IWebApplicationInfo (see page 1078) interface, the value of this property is nil (Delphi) or NULL (C++). When ApplicationAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the Application variable in server-side scripts.

See Also

TApplicationAdapter (see page 5439)

60.1.11.3.3 TCustomWebAppComponents.DispatchActions Property

Provides access to an action-based Web dispatcher.

Pascal

```
property DispatchActions: IWebDispatchActions;
```

C++

```
__property IWebDispatchActions DispatchActions;
```

Description

Read (see page 3634) DispatchActions to determine whether the WebSnap application contains a Web dispatcher. A Web dispatcher maintains a list of action items that respond to incoming request messages. Web dispatchers can appear in both WebSnap and Web broker applications.

The IWebDispatchActions interface that is the value of DispatchActions does nothing special other than identify its component as a Web dispatcher. You can, however, use this interface to obtain other interfaces that belong to the Web dispatcher.

Typically, DispatchActions is the interface of a TWebDispatcher object. If the WebSnap application does not include a TWebDispatcher component, or some other component that implements the IWebDispatchActions interface, the value of this property is nil (Delphi) or NULL (C++).

See Also

TWebModule

TWebDispatcher

60.1.11.3.4 TCustomWebAppComponents.EndUserAdapter Property

Provides access to an end-user adapter.

Pascal

```
property EndUserAdapter: IWebEndUser;
```

C++

```
__property IWebEndUser EndUserAdapter;
```

Description

Use EndUserAdapter to get or set the interface of the end user adapter. An end user adapter maintains information about the current end user, and supports the actions for logging users in and out. End user adapters implement the fields and actions of the EndUser variable in server-side scripts.

Typically, EndUserAdapter is the interface of a TEndUserAdapter or TEndUserSessionAdapter object. If the WebSnap application does not include a component that implements the IWebEndUser (see page 5988) interface, the value of this property is nil (Delphi) or NULL (C++). When EndUserAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the EndUser variable in server-side scripts or take advantage of the login support that end user adapters provide.

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

60.1.11.3.5 TCustomWebAppComponents.LocateFileService Property

Provides access to the file location service.

Pascal

```
property LocateFileService: ILocateFileService;
```

C++

```
__property ILocateFileService LocateFileService;
```

Description

Use LocateFileService to get or set the interface of the file location service. The file location service controls access to the

templates and server-side script include files that are used while processing requests.

Typically, LocateFileService is the interface of a TLocateFileService (see page 6220) object. If the WebSnap application does not include a component that implements the ILocateFileService interface, the value of this property is nil (Delphi) or NULL (C++). When LocateFileService is nil (Delphi) or NULL (C++), the WebSnap application can still use template files and include files, but you have less control over how the application locates these files (that is, they must reside in the current directory or references to them must be qualified).

See Also

TLocateFileService (see page 6220)

60.1.11.3.6 TCustomWebAppComponents.PageDispatcher Property

Provides access to the current page dispatcher object.

Pascal

```
property PageDispatcher: IPageDispatcher;
```

C++

```
__property IPageDispatcher PageDispatcher;
```

Description

Use PageDispatcher to get or set the interface of the page dispatcher. The page dispatcher is responsible for interpreting requests for specific Web pages, checking the access rights for requested Web pages, and dispatching Web page requests to the appropriate Web page module.

Typically, PageDispatcher is the interface of a TPageDispatcher (see page 6236) object. If the WebSnap application does not include a TPageDispatcher (see page 6236) component, or some other component that implements the IPageDispatcher (see page 1060) interface, the value of this property is nil (Delphi) or NULL (C++). When PageDispatcher is nil (Delphi) or NULL (C++), the WebSnap application does not automatically generate Web page content in response to requests that give the Web page name in their target URL.

See Also

TPageDispatcher (see page 6236)

AdapterDispatcher (see page 6211)

60.1.11.3.7 TCustomWebAppComponents.Sessions Property

Provides access to the sessions service.

Pascal

```
property Sessions: ISessionsService;
```

C++

```
__property ISessionsService Sessions;
```

Description

Use Sessions to get or set the interface of the sessions service. The sessions service maintains Web sessions so that the WebSnap application can store temporary information about an end user.

Typically, Sessions is the interface of a TSessionsService object. If the WebSnap application does not include a component that implements the ISessionsService (see page 1066) interface, the value of this property is nil (Delphi) or NULL (C++). When Sessions is nil (Delphi) or NULL (C++), the WebSnap application can't use Web sessions to store named information about the current end user.

See Also

TSessionsService (see page 6756)

60.1.11.3.8 TCustomWebAppComponents.UserListService Property

Provides access to the file location service.

Pascal

```
property UserListService: IWebUserList;
```

C++

```
__property IWebUserList UserListService;
```

Description

Use UserListService to get or set the interface of the user list service. The user list service maintains a list of user names, passwords, and access rights. Using this information, it can validate logins and check a user's access rights.

Typically, UserListService is the interface of a TWebUserList object. If the WebSnap application does not include a component that implements the IWebUserList (see page 1116) interface, the value of this property is nil (Delphi) or NULL (C++). When UserListService is nil (Delphi) or NULL (C++), the WebSnap application can still allow users to log in and check access rights, but the support for this must be coded using event handlers on the page dispatcher.

See Also

TWebUserList (see page 6796)

TPageDispatcher (see page 6236)

60.1.11.4 TCustomWebAppComponents Events

The events of the TCustomWebAppComponents class are listed here.

Public Events

	Name	Description
	OnException (see page 6215)	Occurs when an exception escapes all try blocks in the WebSnap application.

60.1.11.4.1 TCustomWebAppComponents.OnException Event

Occurs when an exception escapes all try blocks in the WebSnap application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Write (see page 3707) an OnException event handler respond to exceptions that arise while processing a request message. OnException occurs when an exception is not caught by the components that process the request message. If the OnException event handler does not handle the exception, TCustomWebAppComponents (see page 6201) updates the response message so that it describes the exception to the caller.

See Also

OnAfterDispatch (see page 6235)

60.1.12 TInvalidWebSession Class

Serves as a placeholder when an HTTP request could not be given a session.

Class Hierarchy

Pascal

```
TInvalidWebSession = class(TAbstractWebSession);
```

C++

```
class TInvalidWebSession : public TAbstractWebSession;
```

File

WebDisp

Description

TInvalidWebSession implements the Session property on the Web context when the WebSnap application fails to create an active session (TWebSession (see page 6251)) for the current request. The failure to create an active session can arise from various reasons, including the situation where the WebSnap application does not have a TSessionsService component or the situation where an incoming request message tries to reactivate a session that was terminated.

TInvalidWebSession acts as a placeholder, so that the WebSnap application does not need to raise an exception when the attempt to create an active session fails. Any attempt to use TInvalidWebSession, however, leads to an ESessionException (see page 6147) exception.

See Also

TSessionsService (see page 6756)

TAbstractWebContext (see page 5960)

TWebSession (see page 6251)

ESessionException (see page 6147)

60.1.12.1 TInvalidWebSession Members

The following tables list the members exposed by TInvalidWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 6217)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 6218)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 6218)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 6219)	Sets the value of the TimoutMinutes property.
	SetValue (see page 6219)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Create (see page 6217)	Creates and initializes an instance of TInvalidWebSession (see page 6216).
	Terminate (see page 6219)	Raises an ESessionException (see page 6147) exception.
	UpdateResponse (see page 6220)	Does nothing.

60.1.12.2 TInvalidWebSession.Create Constructor

Creates and initializes an instance of TInvalidWebSession (see page 6216).

Pascal

```
constructor Create(const AMsg: string);
```

C++

```
__fastcall TInvalidWebSession(const UnicodeString AMsg);
```

Description

Do not create instances of TInvalidWebSession (see page 6216). WebSnap applications automatically create TInvalidWebSession (see page 6216) objects when a call to the TWebSession (see page 6251) constructor fails.

AMsg is the error message from the exception that was raised within a call to the TWebSession (see page 6251) constructor. This error message is used for any ESessionException (see page 6147) exception that the TInvalidWebSession (see page 6216) object raises.

See Also

ESessionException (see page 6147)

Create (see page 6252)

60.1.12.3 TInvalidWebSession Methods

The methods of the TInvalidWebSession class are listed here.

Protected Methods

	Name	Description
	GetSessionID (see page 6217)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 6218)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 6218)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 6219)	Sets the value of the TimoutMinutes property.
	SetValue (see page 6219)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 6219)	Raises an ESessionException (see page 6147) exception.
	UpdateResponse (see page 6220)	Does nothing.

60.1.12.3.1 TInvalidWebSession.GetSessionID Method

Returns the value of the SessionID property.

Pascal

```
function GetSessionID: string; override;
```

C++

```
virtual __fastcall UnicodeString GetSessionID();
```

Description

GetSessionID is the protected read implementation of the SessionID property.

In TAbstractWebSession, GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID property.

See Also

SessionID

60.1.12.3.2 TInvalidWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes property.

Pascal

```
function GetTimeoutMinutes: Integer; override;
```

C++

```
virtual __fastcall int GetTimeoutMinutes();
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes property.

In TAbstractWebSession, GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

TimeoutMinutes

SetTimeoutMinutes

60.1.12.3.3 TInvalidWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; override;
```

C++

```
virtual __fastcall Variant GetValue(const UnicodeString AName);
```

Description

GetValue is the protected read implementation of the Values property.

AName is the name of the value to return.

In TAbstractWebSession, GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

Values

SetValue

60.1.12.3.4 TInvalidWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); override;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue);
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession, SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

GetTimeoutMinutes

TimeoutMinutes

60.1.12.3.5 TInvalidWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession, SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

GetValue

Values

60.1.12.3.6 TInvalidWebSession.Terminate Method

Raises an ESessionException (see page 6147) exception.

Pascal

```
procedure Terminate; override;
```

C++

```
virtual __fastcall Terminate();
```

Description

Do not call the Terminate method. TInvalidWebSession (see page 6216) does not represent an active Web session, so there is no Web session to terminate.

If the application calls the Terminate method, TInvalidWebSession (see page 6216) raises an ESessionException (see page 6147) object.

See Also

ESessionException (see page 6147)

60.1.12.3.7 TInvalidWebSession.UpdateResponse Method

Does nothing.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse);
```

Description

UpdateResponse is called automatically to allow the WebSnap application to add the session ID to the response message before it is sent to the user. In TInvalidWebSession (see page 6216), UpdateResponse does nothing, because invalid Web sessions do not have session ID's.

See Also

SessionID

60.1.13 TLocateFileService Class

TLocateFileService controls the access of templates and server-side script include files at runtime.

Class Hierarchy

Pascal

```
TLocateFileService = class(TCustomLocateFileService);
```

C++

```
class TLocateFileService : public TCustomLocateFileService;
```

File

WebDisp

Description

Add a TLocateFileService component to your WebSnap application when you want greater control over how template and

server-side script include files are accessed at runtime. Define event handlers that will be called when the application requires one of these files. Event handlers can specify a different directory location for a file or return a TStream containing the file.

See Also

TStream

60.1.13.1 TLocateFileService Members

The following tables list the members exposed by TLocateFileService.

Protected Methods

	Name	Description
	FindIncludeFile (see page 6182)	Returns the fully-qualified path name for an include file for a server-side script.
V	FindStream (see page 6183)	Generates an OnFindStream (see page 6187) event.
	FindTemplateFile (see page 6183)	Generates an OnFindTemplateFile (see page 6187) event.
V	GetFileStream (see page 6184)	Returns a stream object from which to read a specified file.
	GetTemplateStream (see page 6184)	Returns a stream from which to read a template file.
V	ImplFindIncludeFile (see page 6185)	Provides the underlying implementation for the FindIncludeFile (see page 6182) method.
V	ImplGetTemplateStream (see page 6185)	Provides the underlying implementation for the GetTemplateStream (see page 6184) method.

Public Methods

	Name	Description
	FindIncludeFile (see page 1017)	This is FindIncludeFile, a member of class IFindIncludeFile.

Published Events

	Name	Description
	OnFindIncludeFile (see page 6221)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6222)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6222)	Occurs when the WebSnap application tries to locate the template file for a content producer.

60.1.13.2 TLocateFileService Events

The events of the TLocateFileService class are listed here.

Published Events

	Name	Description
	OnFindIncludeFile (see page 6221)	Occurs when the WebSnap application tries to locate an include file for a server-side script.
	OnFindStream (see page 6222)	Occurs when the WebSnap application tries to locate a file.
	OnFindTemplateFile (see page 6222)	Occurs when the WebSnap application tries to locate the template file for a content producer.

60.1.13.2.1 TLocateFileService.OnFindIncludeFile Event

Occurs when the WebSnap application tries to locate an include file for a server-side script.

Pascal

```
property OnFindIncludeFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindIncludeFile;
```

Description

Write an OnFindIncludeFile event handler to indicate the location on disk of include files for server-side scripts. In WebSnap applications that include a locate file service component, you must supply either an OnFindIncludeFile or an OnFindStream event handler (or both) if the server-side scripts have include files. Otherwise the application can't locate these include files at all.

See Also

OnFindStream (see page 6187)

OnFindTemplateFile (see page 6187)

60.1.13.2.2 TLocateFileService.OnFindStream Event

Occurs when the WebSnap application tries to locate a file.

Pascal

```
property OnFindStream: TLocateFileServiceFindStreamEvent;
```

C++

```
__property TLocateFileServiceFindStreamEvent OnFindStream;
```

Description

Write an OnFindStream event handler to supply the WebSnap application with the contents of a file given its name. This event is particularly useful for applications that use "logical" files that do not exist physically on disk. When the WebSnap application tries to locate a file, the locate file service generates this event, along with an OnFindTemplateFile. One of these event handlers must be provided in order for the application to use the requested file.

See Also

OnFindIncludeFile (see page 6186)

OnFindTemplateFile (see page 6187)

60.1.13.2.3 TLocateFileService.OnFindTemplateFile Event

Occurs when the WebSnap application tries to locate the template file for a content producer.

Pascal

```
property OnFindTemplateFile: TLocateFileServiceFindFileEvent;
```

C++

```
__property TLocateFileServiceFindFileEvent OnFindTemplateFile;
```

Description

Write an OnFindIncludeFile event handler to indicate the location on disk of templates that a content producer component uses to generate its content. In WebSnap applications that include a locate file service component, you must supply either an OnFindTemplateFile or an OnFindStream event handler (or both) if the application has content producers that use template files. Otherwise content producers can't locate their templates.

See Also

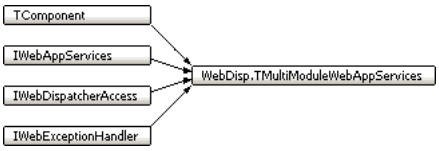
OnFindStream (see page 6187)

OnFindIncludeFile (see page 6186)

60.1.14 TMultiModuleWebAppServices Class

TMultiModuleWebAppServices coordinates the response to HTTP request messages by locating the appropriate components to handle various tasks.

Class Hierarchy



Pascal

```
TMultiModuleWebAppServices = class(TComponent, IWebAppServices, IWebDispatcherAccess,
IWebExceptionHandler);
```

C++

```
class TMultiModuleWebAppServices : public TComponent, public IWebAppServices, public
IWebDispatcherAccess, public IWebExceptionHandler;
```

File

WebDisp

Description

TMultiModuleWebAppServices is the base class for components that provide access to the services available in a WebSnap application. It keeps track of the current request and response objects, maintains a list of the available Web modules, and introduces methods for locating the appropriate components to handle incoming request messages.

Do not use TMultiModuleWebAppServices directly. The methods that process request messages and any exceptions that arise during that processing are simple stubs that should be overridden by a descendant class. Rather, use a TMultiModuleWebAppServices descendant such as TWebAppComponents (see page 6243), to provide access to the central components of the WebSnap application.

See Also

TWebRequest

TWebResponse

TWebRequestHandler (see page 6633)

TWebAppComponents (see page 6243)

IWebAppServices

60.1.14.1 TMultiModuleWebAppServices Members

The following tables list the members exposed by TMultiModuleWebAppServices.

Protected Events

	Name	Description
	OnAfterDispatch (see page 6235)	Occurs after the WebSnap application generates the content of a response message.

	OnBeforeDispatch (see page 6235)	Occurs before TMultiModuleWebAppServices (see page 6223) passes an HTTP request onto the other components in the application for processing.

Protected Methods

	Name	Description
	Access_Request (see page 6225)	Returns the value of the Access_Request property.
	Access_Response (see page 6225)	Returns the value of the Access_Response property.
V	CreateContext (see page 6225)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6226)	Generates an OnAfterDispatch (see page 6235) event.
	DoBeforeDispatch (see page 6226)	Generates an OnBeforeDispatch (see page 6235) event.
V	DoneModule (see page 6227)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6227)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 6228)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleException (see page 6228)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6229)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6229)	Provides the underlying implementation of the FinishContext (see page 6227) method.
V	ImplHandleException (see page 6230)	Provides the underlying implementation of the HandleException (see page 6228) method.
V	ImplHandleRequest (see page 6230)	Provides the underlying implementation for the HandleRequest (see page 6229) method.
V	ImplInitContext (see page 6231)	Provides the underlying implementation for the InitContext (see page 6231) method.
	InitContext (see page 6231)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 6232)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6233)	Initializes all the currently active Web modules.
	ModuleAdded (see page 6233)	Initializes a Web module when it is activated while processing the current request.

Protected Properties

	Name	Description
R	Request (see page 6234)	Provides access to the current request object.
R	Response (see page 6234)	Provides access to the current response object.
R	WebModules (see page 6234)	Lists the Web modules available in the WebSnap application.

60.1.14.2 TMultiModuleWebAppServices Methods

The methods of the TMultiModuleWebAppServices class are listed here.

Protected Methods

	Name	Description
	Access_Request (see page 6225)	Returns the value of the Access_Request property.
	Access_Response (see page 6225)	Returns the value of the Access_Response property.
V	CreateContext (see page 6225)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6226)	Generates an OnAfterDispatch (see page 6235) event.
	DoBeforeDispatch (see page 6226)	Generates an OnBeforeDispatch (see page 6235) event.
V	DoneModule (see page 6227)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6227)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetExceptionHandler (see page 6228)	Returns the object that handles exceptions that arise in the WebSnap application.
	HandleException (see page 6228)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6229)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6229)	Provides the underlying implementation of the FinishContext (see page 6227) method.

	ImplHandleException (see page 6230)	Provides the underlying implementation of the HandleException (see page 6228) method.
	ImplHandleRequest (see page 6230)	Provides the underlying implementation for the HandleRequest (see page 6229) method.
	ImplInitContext (see page 6231)	Provides the underlying implementation for the InitContext (see page 6231) method.
	InitContext (see page 6231)	Creates and initializes the Web context for the current Web request.
	InitModule (see page 6232)	Initializes a Web module before the WebSnap application uses it to handle the current request.
	InitModules (see page 6233)	Initializes all the currently active Web modules.
	ModuleAdded (see page 6233)	Initializes a Web module when it is activated while processing the current request.

60.1.14.2.1 TMultiModuleWebAppServices.Access_Request Method

Returns the value of the Access_Request property.

Pascal

```
function Access_Request: TWebRequest;
```

C++

```
__fastcall TWebRequest Access_Request();
```

Description

Access_Access_Request provides the TMultiModuleWebAppServices (see page 6223) object's implementation of the Access_Request method of its IWebDispatcherAccess interface. This allows content producer components in the application to access the current request message.

See Also

Request

Dispatcher

60.1.14.2.2 TMultiModuleWebAppServices.Access_Response Method

Returns the value of the Access_Response property.

Pascal

```
function Access_Response: TWebResponse;
```

C++

```
__fastcall TWebResponse Access_Response();
```

Description

Access_Access_Response provides the TMultiModuleWebAppServices (see page 6223) object's implementation of the Access_Response method of its IWebDispatcherAccess interface. This allows content producer components in the application to access the current Access_Response message.

See Also

Response

Dispatcher

60.1.14.2.3 TMultiModuleWebAppServices.CreateContext Method

Creates the Web context for the current request message.

Pascal

```
function CreateContext: TAbstractWebContext; virtual;
```

C++

```
virtual __fastcall TAbstractWebContext CreateContext();
```

Description

The InitContext (see page 6231) method calls CreateContext to instantiate the Web context object for the current request. This context can then be accessed by the WebSnap application using the global WebContext (see page 6003) function.

In TMultiModuleWebAppServices (see page 6223), CreateContext creates a TWebContext (see page 5976) object, passing the values of the Request (see page 6234), Response (see page 6234), and WebModules (see page 6234) properties to the constructor. The Web context is created without a session.

See Also

WebContext (see page 6003)

InitContext (see page 6231)

Create (see page 5978)

Request (see page 6234)

Response (see page 6234)

WebModules (see page 6234)

60.1.14.2.4 TMultiModuleWebAppServices.DoAfterDispatch Method

Generates an OnAfterDispatch (see page 6235) event.

Pascal

```
function DoAfterDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoAfterDispatch(TWebRequest Request, TWebResponse Response);
```

Description

TMultiModuleWebAppServices (see page 6223) descendants call DoAfterDispatch to generate an OnAfterDispatch (see page 6235) event after dispatching a request message to the other components in the WebSnap application.

Request (see page 6234) is the current request message that was just handled.

Response (see page 6234) is the response message that was filled in by the rest of the WebSnap application.

DoAfterDispatch returns true if the WebSnap application should send the response message back to the caller.

See Also

OnAfterDispatch (see page 6235)

60.1.14.2.5 TMultiModuleWebAppServices.DoBeforeDispatch Method

Generates an OnBeforeDispatch (see page 6235) event.

Pascal

```
function DoBeforeDispatch(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DoBeforeDispatch(TWebRequest Request, TWebResponse Response);
```

Description

TMultiModuleWebAppServices (see page 6223) descendants call DoBeforeDispatch to generate an OnBeforeDispatch (see page 6235) event before dispatching a request message to the other components in the WebSnap application.

Request (see page 6234) is the current request message that needs to be handled.

Response (see page 6234) is the response message that needs to be filled out.

DoBeforeDispatch returns true if the OnBeforeDispatch (see page 6235) event handler has handled the event. When DoBeforeDispatch returns true, the WebSnap application does not need to continue processing the request.

See Also

OnBeforeDispatch (see page 6235)

60.1.14.2.6 TMultiModuleWebAppServices.DoneModule Method

Performs any final tasks before deactivating a Web module.

Pascal

```
procedure DoneModule(AWebModule: TComponent); virtual;
```

C++

```
virtual __fastcall DoneModule(TComponent * AWebModule);
```

Description

The FinishContext (see page 6227) method calls DoneModule for every active Web module as part of the final cleanup before freeing the Web modules and Web context when the WebSnap application is finished handling an HTTP request. DoneModule notifies the Web module that it is about to be deactivated, so that it can perform any final tasks before it is freed or deactivated.

AWebModule is the Web module instance that is about to be deactivated.

See Also

WebModules (see page 6234)

InitModule (see page 6232)

FinishContext (see page 6227)

60.1.14.2.7 TMultiModuleWebAppServices.FinishContext Method

Performs any final tasks after the WebSnap application has finished processing a request.

Pascal

```
procedure FinishContext;
```

C++

```
__fastcall FinishContext();
```

Description

FinishContext is called as the last step before a WebSnap application sends a response message back to the caller. FinishContext performs any final tasks, and frees all the objects and components that were created to handle the request message.

FinishContext is not a virtual method. Descendant classes can't override this method to add additional final tasks. Instead, FinishContext calls the virtual ImplFinishContext (see page 6229) method, which descendant classes override to add their own final tasks.

See Also

InitContext (see page 6231)

ImplFinishContext (see page 6229)

60.1.14.2.8 TMultiModuleWebAppServices.GetExceptionHandler Method

Returns the object that handles exceptions that arise in the WebSnap application.

Pascal

```
function GetExceptionHandler: TObject;
```

C++

```
__fastcall TObject * GetExceptionHandler();
```

Description

GetExceptionHandler returns the object that the WebSnap application should call when an exception is raised while trying to process a request message. The request handler must implement the IWebExceptionHandler interface, which includes only a single method: HandleException (see page 6228).

In TMultiModuleWebAppServices (see page 6223), GetExceptionHandler returns self this, because TMultiModuleWebAppServices (see page 6223) implements the IWebExceptionHandler interface.

See Also

HandleException (see page 6228)

60.1.14.2.9 TMultiModuleWebAppServices.HandleException Method

Responds to any exceptions not caught by the WebSnap application.

Pascal

```
procedure HandleException(E: Exception; var Handled: Boolean);
```

C++

```
__fastcall HandleException(Exception E, Boolean Handled);
```

Description

HandleException provides the last opportunity for a WebSnap application to handle an exception that passes through all the try blocks in the application code. If the HandleException method does not handle the exception, the WebSnap application displays a dialog box that indicates an error occurred.

HandleException calls the ImplHandleException (see page 6230) method to handle the exception. In TMultiModuleWebAppServices (see page 6223), ImplHandleException (see page 6230) does nothing. However, descendant classes override ImplHandleException (see page 6230) to provide more meaningful responses, such as generating an OnException event or creating a response message that describes the exception to the caller.

E is the exception object for the exception that occurred.

Handled returns true if HandleException handled the exception. If Handled returns false, the WebSnap application displays a dialog box that indicates an error occurred.

See Also

ImplHandleException (see page 6230)

60.1.14.2.10 TMultiModuleWebAppServices.HandleRequest Method

Directs the processing of the current request message.

Pascal

```
function HandleRequest: Boolean;
```

C++

```
__fastcall Boolean HandleRequest();
```

Description

When the WebSnap application receives a request message, it makes the following calls:

1. The InitContext (see page 6231) method is called to generate a Web context for the request.
2. HandleRequest is called to generate a response to the request.
3. FinishContext (see page 6227) is called to perform any final cleanup.

HandleRequest simply calls the ImplHandleRequest (see page 6230) method, which directs the processing of the request. In TMultiModuleWebAppServices (see page 6223), ImplHandleRequest (see page 6230) simply declares the request as handled while doing no processing. Descendant classes override ImplHandleRequest (see page 6230) to locate the components that should handle the request and forwarding the request to those components for processing.

HandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

ImplHandleRequest (see page 6230)

Request (see page 6234)

InitContext (see page 6231)

FinishContext (see page 6227)

60.1.14.2.11 TMultiModuleWebAppServices.ImplFinishContext Method

Provides the underlying implementation of the FinishContext (see page 6227) method.

Pascal

```
procedure ImplFinishContext; virtual;
```

C++

```
virtual __fastcall ImplFinishContext();
```

Description

ImplFinishContext performs any final tasks and cleanup after the WebSnap application has finished processing a request message.

In TMultiModuleWebAppServices (see page 6223), ImplFinishContext calls the DoneModule (see page 6227) method for every active Web module so that it can be freed or deactivated, and then frees the Web context for the current request.

Descendant classes override ImplFinishContext to add their own final tasks.

See Also

FinishContext (see page 6227)

DoneModule (see page 6227)

WebContext (see page 6003)

60.1.14.2.12 TMultiModuleWebAppServices.ImplHandleException Method

Provides the underlying implementation of the HandleException (see page 6228) method.

Pascal

```
procedure ImplHandleException(E: Exception; var Handled: Boolean); virtual;
```

C++

```
virtual __fastcall ImplHandleException(Exception E, Boolean Handled);
```

Description

The HandleException (see page 6228) method calls ImplHandleException to handle any exception that passes through all the try blocks in the application code. If ImplHandleException does not handle the exception, the WebSnap application displays a dialog box that indicates an error occurred.

In TMultiModuleWebAppServices (see page 6223), ImplHandleException returns true, ignoring the exception that occurred. Descendant classes override this method to provide more meaningful responses, such as generating an OnException event or creating a response message that describes the exception to the caller.

E is the exception object for the exception that occurred.

Handled returns true if ImplHandleException handled the exception. If Handled returns false, the WebSnap application displays a dialog box that indicates an error occurred.

See Also

HandleException (see page 6228)

60.1.14.2.13 TMultiModuleWebAppServices.ImplHandleRequest Method

Provides the underlying implementation for the HandleRequest (see page 6229) method.

Pascal

```
function ImplHandleRequest: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplHandleRequest();
```

Description

The HandleRequest (see page 6229) method calls ImplHandleRequest to handle the current HTTP request message. In TMultiModuleWebAppServices (see page 6223), ImplHandleRequest returns true, indicating that the request is handled, but does nothing to fill out the response object. Descendant classes override this method to locate components in the WebSnap application that can process the request and forward the request to them for handling.

ImplHandleRequest returns true if the request is handled and the response should be sent back to the caller (if it was not already sent). It returns false if the WebSnap application could not handle the request.

See Also

HandleRequest (see page 6229)

Request (see page 6234)

Response (see page 6234)

60.1.14.2.14 TMultiModuleWebAppServices.ImplInitContext Method

Provides the underlying implementation for the InitContext (see page 6231) method.

Pascal

```
procedure ImplInitContext(AWebModuleList: TAbstractWebModuleList; ARequest: TWebRequest;
AResponse: TWebResponse); virtual;
```

C++

```
virtual __fastcall ImplInitContext(TAbstractWebModuleList AWebModuleList, TWebRequest
ARequest, TWebResponse AResponse);
```

Description

The InitContext (see page 6231) method, which is called when the WebSnap application first receives a request, calls ImplInitContext to perform any initializations and to create the Web context for the new request message.

ImplInitContext performs the following tasks:

1. Sets the WebModules (see page 6234), Request (see page 6234), and Response (see page 6234) properties to the values of the AWebModuleList, ARequest, and AResponse parameters, respectively.
2. Calls the CreateContext (see page 6225) method to create a Web context for the current request. This Web context can then be accessed using the global WebContext (see page 6003) function.
3. Calls the InitModules (see page 6233) method to initialize all the Web modules that the AWebModuleList parameter indicates are currently instantiated. (These are the Web modules that are created for every request message.)
4. Hooks up the ModuleAdded (see page 6233) method so that it can initialize any Web modules that are instantiated in the course of processing the current request.

See Also

InitContext (see page 6231)

CreateContext (see page 6225)

WebContext (see page 6003)

Request (see page 6234)

Response (see page 6234)

WebModules (see page 6234)

InitModules (see page 6233)

60.1.14.2.15 TMultiModuleWebAppServices.InitContext Method

Creates and initializes the Web context for the current Web request.

Pascal

```
procedure InitContext(WebModuleList: TAbstractWebModuleList; Request: TWebRequest; Response:
```

```
TWebResponse);
```

C++

```
__fastcall InitContext(TAbstractWebModuleList WebModuleList, TWebRequest Request, TWebResponse
Response);
```

Description

InitContext is called when the WebSnap application first receives a request message to set up the Web context before the application attempts to handle the request. It creates and sets up the global Web context, which can be accessed by calling the global WebContext (see page 6003) function. It also performs any other initializations that ready the application for handling a new request message.

WebModuleList lists the currently active Web modules in the WebSnap application and provides access to all Web module factories so that the application can create any new modules it needs for processing the request message. This becomes the value of the WebModules (see page 6234) property.

Request (see page 6234) describes the new request message. This becomes the value of the Request (see page 6234) property.

Response (see page 6234) is the response object that the WebSnap application should fill out when handling the current request. This becomes the value of the Response (see page 6234) property.

InitContext is not a virtual method. Descendant classes can't override this method to add additional initializations. Instead, InitContext calls the virtual ImplInitContext (see page 6231) method, which descendant classes override to add their own initial tasks.

See Also

ImplInitContext (see page 6231)

WebContext (see page 6003)

Request (see page 6234)

Response (see page 6234)

WebModules (see page 6234)

60.1.14.2.16 TMultiModuleWebAppServices.InitModule Method

Initializes a Web module before the WebSnap application uses it to handle the current request.

Pascal

```
procedure InitModule(AWebModule: TComponent); virtual;
```

C++

```
virtual __fastcall InitModule(TComponent * AWebModule);
```

Description

Both the ModuleAdded (see page 6233) and the InitModules (see page 6233) methods call InitModule to initialize a Web module after it is first created. InitModule assigns itself as the Web dispatcher to any content producers in the Web module, and notifies the Web module that it has been activated.

AWebModule is the Web module component to initialize.

See Also

InitModules (see page 6233)

ModuleAdded (see page 6233)

WebModules (see page 6234)

Dispatcher

60.1.14.2.17 TMultiModuleWebAppServices.InitModules Method

Initializes all the currently active Web modules.

Pascal

```
procedure InitModules; virtual;
```

C++

```
virtual __fastcall InitModules();
```

Description

InitModules is called to initialize the currently active Web modules when the WebSnap application first sets up the Web context for a new request. It iterates over all the active Web modules that the WebModules (see page 6234) property lists, calling the InitModule (see page 6232) method to initialize each one.

See Also

InitModule (see page 6232)

WebModules (see page 6234)

InitContext (see page 6231)

60.1.14.2.18 TMultiModuleWebAppServices.ModuleAdded Method

Initializes a Web module when it is activated while processing the current request.

Pascal

```
procedure ModuleAdded(AWebModule: TComponent);
```

C++

```
__fastcall ModuleAdded(TComponent * AWebModule);
```

Description

TMultiModuleWebAppServices (see page 6223) assigns ModuleAdded as the OnModuleAdded event handler on the TWebModuleList that is the value of the WebModules (see page 6234) property. As a result, this method is called every time the WebSnap application activates a new Web module while processing the current request.

ModuleAdded calls the InitModule (see page 6232) method to initialize the new Web module before it is used.

AWebModule is the newly activated Web module.

See Also

InitModule (see page 6232)

OnModuleAdded

WebModules (see page 6234)

60.1.14.3 TMultiModuleWebAppServices Properties

The properties of the TMultiModuleWebAppServices class are listed here.

Protected Properties

	Name	Description
	Request (see page 6234)	Provides access to the current request object.
	Response (see page 6234)	Provides access to the current response object.
	WebModules (see page 6234)	Lists the Web modules available in the WebSnap application.

60.1.14.3.1 TMultiModuleWebAppServices.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3634) Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 6234)

60.1.14.3.2 TMultiModuleWebAppServices.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3634) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 6234) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 6234)

60.1.14.3.3 TMultiModuleWebAppServices.WebModules Property

Lists the Web modules available in the WebSnap application.

Pascal

```
property WebModules: TAbstractWebModuleList;
```

C++

```
__property TAbstractWebModuleList WebModules;
```

Description

Use WebModules to access one of the Web modules that make up the WebSnap application. WebModules lists all of the

currently instantiated Web modules. It also provides access to the factory objects for all Web modules in the application, including those that are not yet instantiated.

When the application first receives a Web request message, the Web request handler initializes WebModules to include all Web modules that have the crAlways flag set in their factory's CreateMode property.

See Also

CreateMode

60.1.14.4 TMultiModuleWebAppServices Events

The events of the TMultiModuleWebAppServices class are listed here.

Protected Events

	Name	Description
	OnAfterDispatch (see page 6235)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6235)	Occurs before TMultiModuleWebAppServices (see page 6223) passes an HTTP request onto the other components in the application for processing.

60.1.14.4.1 TMultiModuleWebAppServices.OnAfterDispatch Event

Occurs after the WebSnap application generates the content of a response message.

Pascal

```
property OnAfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAfterDispatch;
```

Description

Write (see page 3707) an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the WebSnap application a request. OnAfterDispatch occurs after the other components in the WebSnap application have attempted to generate a response for the current request. It does not occur if the response message has already been sent back to the caller.

If the AfterDispatch event handler sets the Handled parameter to true, the WebSnap application sends the response back to the caller after the event handler exits. If it sets the Handled parameter to false, the WebSnap application does not send the response message.

Note: Although TMultiModuleWebAppServices (see page 6223) introduces the OnAfterDispatch event, it never calls the event handler. Descendant classes add the code to generate this event after dispatching a request message.

See Also

OnBeforeDispatch (see page 6235)

60.1.14.4.2 TMultiModuleWebAppServices.OnBeforeDispatch Event

Occurs before TMultiModuleWebAppServices (see page 6223) passes an HTTP request onto the other components in the application for processing.

Pascal

```
property OnBeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnBeforeDispatch;
```

Description

Write (see page 3707) an OnBeforeDispatch event handler to initialize the response to an HTTP request or to handle any messages that the other components in the WebSnap application can't handle. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing this component from trying to locate a component in the WebSnap application that can handle the message.

Note: Although TMultiModuleWebAppServices (see page 6223) introduces the OnBeforeDispatch event, it never calls the event handler. Descendant classes add the code to generate this event before dispatching a request message.

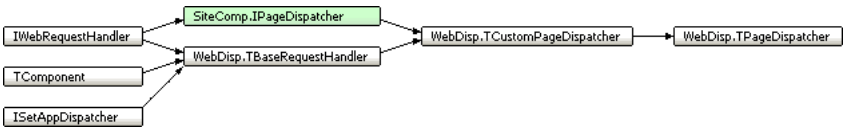
See Also

OnAfterDispatch (see page 6235)

60.1.15 TPageDispatcher Class

TPageDispatcher dispatches incoming HTTP request messages in a WebSnap application.

Class Hierarchy



Pascal

```
TPageDispatcher = class(TCustomPageDispatcher);
```

C++

```
class TPageDispatcher : public TCustomPageDispatcher;
```

File

WebDisp

Description

Add a single TPageDispatcher object to the Web application module of a WebSnap application. Typically, the WebSnap application wizard adds a page dispatcher to the Web application module that it creates.

A WebSnap application only requires a single page dispatcher. The page dispatcher examines all incoming HTTP request messages, checking the pathInfo portion of the URL to which a request is addressed for the name of a registered Web page module. It then forwards the request to the matching Web page module so that the page module can handle the request and produce the content of a response. If the request is not addressed to a URL with a pathinfo portion, the page dispatcher forwards the request to the Web page module identified by its DefaultPage (see page 6238) property.

See Also

TWebDispatcher

60.1.15.1 TPageDispatcher Members

The following tables list the members exposed by TPageDispatcher.

Protected Methods

	Name	Description
	DispatchPageName (see page 6190)	Checks whether a specified user has a given access rights string.
	EndUser (see page 6190)	This is EndUser, a member of class TCustomPageDispatcher.
	GetCanView (see page 6191)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 6191)	Returns the value of the DefaultPageName property.
	GetLoginRequired (see page 6191)	Indicates whether a user must log in before accessing a specified Web page.
	GetRequestedPage (see page 6192)	This is GetRequestedPage, a member of class TCustomPageDispatcher.
	HandleRequest (see page 6167)	Handles an HTTP request.
	ImplDispatchPageName (see page 6192)	This is ImplDispatchPageName, a member of class TCustomPageDispatcher.
V	ImplGetCanView (see page 6192)	This is ImplGetCanView, a member of class TCustomPageDispatcher.
V	ImplGetDefaultPageName (see page 6192)	This is ImplGetDefaultPageName, a member of class TCustomPageDispatcher.
V	ImplGetLoginRequired (see page 6193)	This is ImplGetLoginRequired, a member of class TCustomPageDispatcher.
V	ImplHandleRequest (see page 6167)	Handles application-specific processing of an HTTP request.
V	ImplHandleRequest (see page 6193)	Handles application-specific processing of an HTTP request.
V	ImplPageNameOfRequest (see page 6193)	This is ImplPageNameOfRequest, a member of class TCustomPageDispatcher.
	ImplRedirectToPageName (see page 6193)	This is ImplRedirectToPageName, a member of class TCustomPageDispatcher.
	IsLoggedIn (see page 6194)	This is IsLoggedIn, a member of class TCustomPageDispatcher.
	PageNameOfRequest (see page 6194)	Returns the name of the Web page to which an HTTP request is addressed.
	ProduceLoginPage (see page 6194)	This is ProduceLoginPage, a member of class TCustomPageDispatcher.
	RedirectToPageName (see page 6194)	Fills out a response message so that it redirects the caller to another page in the application.
	SetAppDispatcher (see page 6167)	Sets the AppDispatcher (see page 6168) property.
	ValidatePage (see page 6195)	This is ValidatePage, a member of class TCustomPageDispatcher.

Protected Properties

	Name	Description
R	AppDispatcher (see page 6168)	Provides read-only access to the application dispatcher component.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 6197)	Occurs after the page dispatcher successfully dispatches an HTTP request message in response to an internal redirection request.
	OnBeforeRedirectToPage (see page 6198)	Occurs before the page dispatcher dispatches an HTTP request message in response to an internal redirection request.

Public Methods

	Name	Description
	DispatchPageName (see page 1061)	Checks whether a specified user has a given access rights string.
	GetCanView (see page 1062)	Indicates whether the current user is permitted to view a specified Web page.
	GetDefaultPageName (see page 1062)	Returns the value of the DefaultPageName (see page 1064) property.
	GetLoginRequired (see page 1062)	Indicates whether a user must log in before accessing a specified Web page.
	PageNameOfRequest (see page 1063)	Returns the name of the Web page to which an HTTP request is addressed.
	RedirectToPageName (see page 1063)	Fills out a response message so that it redirects the caller to another page in the application.

Public Properties

	Name	Description
R	DefaultPageName (see page 1064)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Published Events

	Name	Description
	OnAfterDispatchPage (see page 6239)	Occurs after the page dispatcher successfully dispatches an HTTP request message.

	OnBeforeDispatchPage (see page 6239)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.
	OnCanViewPage (see page 6240)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6241)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6241)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6242)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Published Properties

	Name	Description
	DefaultPage (see page 6238)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

60.1.15.2 TPageDispatcher Properties

The properties of the TPageDispatcher class are listed here.

Published Properties

	Name	Description
	DefaultPage (see page 6238)	Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

60.1.15.2.1 TPageDispatcher.DefaultPage Property

Indicates the Web page module that should handle requests that do not include a Web page module name in the pathinfo portion of their target URL.

Pascal

```
property DefaultPage: string;
```

C++

```
__property UnicodeString DefaultPage;
```

Description

DefaultPage is the name of the registered Web page module that should handle incoming request messages that do not have a pathinfo portion on their target URL. This is the name that is supplied to the Web page module wizard when you created the default Web page module. It forms the basis of that Web page module's class name. For example, if the default Web page module has the name MyDefaultPage, the Web page module class is of type TMyDefaultPage.

If DefaultPage is an empty string, the page dispatcher uses the Web application module as a default Web page module, as long as it is a Web Page module.

60.1.15.3 TPageDispatcher Events

The events of the TPageDispatcher class are listed here.

Published Events

	Name	Description
	OnAfterDispatchPage (see page 6239)	Occurs after the page dispatcher successfully dispatches an HTTP request message.
	OnBeforeDispatchPage (see page 6239)	Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

	OnCanViewPage (see page 6240)	Occurs just before the dispatcher passes a request message to a Web page module.
	OnIsLoginRequired (see page 6241)	Occurs when a request arrives from a caller that has not yet logged in.
	OnPageAccessDenied (see page 6241)	Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.
	OnPageNotFound (see page 6242)	Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

60.1.15.3.1 TPageDispatcher.OnAfterDispatchPage Event

Occurs after the page dispatcher successfully dispatches an HTTP request message.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take specific action after an HTTP request has been successfully handled by a Web page module. For example, an OnAfterDispatchPage event handler can maintain a log of all Web traffic handled by the WebSnap application.

OnAfterDispatchPage occurs only if the target Web page module succeeds in handling the request. If the target Web page module could not be validated, an OnPageAccessDenied event occurs instead. If the page could not be located, an OnPageNotFound event occurs instead.

See Also

OnBeforeDispatchPage (see page 6197)

OnPageNotFound (see page 6200)

OnCanViewPage (see page 6198)

OnIsLoginRequired (see page 6199)

OnPageAccessDenied (see page 6200)

60.1.15.3.2 TPageDispatcher.OnBeforeDispatchPage Event

Occurs before the page dispatcher passes an HTTP request to a Web page module for processing.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the dispatcher forwards it to the target Web page module for processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the page dispatcher from locating the target Web page module, verifying that it can handle the request, and forwarding the request to that Web page module.

If the event handler does not handle the request (for example, if it is merely maintaining a log of incoming requests), it can leave

the Handled parameter as false and the dispatcher will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnAfterDispatchPage (see page 6196)

OnPageNotFound (see page 6200)

OnCanViewPage (see page 6198)

OnIsLoginRequired (see page 6199)

OnPageAccessDenied (see page 6200)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

60.1.15.3.3 TPageDispatcher.OnCanViewPage Event

Occurs just before the dispatcher passes a request message to a Web page module.

Pascal

```
property OnCanViewPage: TCanViewPageEvent;
```

C++

```
__property TCanViewPageEvent OnCanViewPage;
```

Description

Write an OnCanViewPage event handler to customize the way the WebSnap application determines whether a given request can be passed to a Web page module for processing. This event is part of the mechanism by which the dispatcher validates a Web page for viewing. The validation process, which occurs after the OnBeforeDispatchPage event, occurs as follows:

1. If the caller has not already logged in, the dispatcher generates an OnIsLoginRequired event to check whether the specified Web page requires the caller to log in.
2. If a login is required, the dispatcher does not validate the page (and so does not forward the request to the specified Web page module). Rather, it initiates the process of creating a response to allow the user to log in.

a. First, the dispatcher generates an OnPageAccessDenied event, where the application can handle the login process.

b. If the OnPageAccessDenied event handler does not handle the login process, the dispatcher forwards the request to the login page specified by the end user adapter's LoginPage property.

3. If a login is not required (either because the caller already logged in or because there is no need to log in), the dispatcher generates an OnCanViewPage event. This event can validate a caller that has logged in, or provide any other type of validation that governs access to the requested page.
4. If there is no OnCanViewPage event handler, or if the OnCanViewPage event handler does not handle the validation, the dispatcher calls on a TEndUserAdapter component to check the user's access rights.

   Tip: To access the Web request and response objects from within an OnCanViewPage event handler, call the global WebContext function to access the Web context and read its Request and Response properties. To access information about the access rights that the target page requires, call the Web context's FindPageInfo method to obtain a Web page info object for the target page and check its ViewAccess property.

See Also

OnIsLoginRequired (see page 6199)

OnBeforeDispatchPage (see page 6197)

OnPageAccessDenied (see page 6200)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

ViewAccess

FindPageInfo (see page 5966)

60.1.15.3.4 TPageDispatcher.OnIsLoginRequired Event

Occurs when a request arrives from a caller that has not yet logged in.

Pascal

```
property OnIsLoginRequired: TLoginRequiredEvent;
```

C++

```
__property TLoginRequiredEvent OnIsLoginRequired;
```

Description

Write an OnIsLoginRequired event handler to customize the way the WebSnap application determines whether a caller must log in before obtaining access to a requested page. This event occurs after the OnBeforeDispatchPage event if the caller has not already logged in to the WebSnap application.

If there is no OnPageAccessDenied event handler, or if the OnPageAccessDenied event handler does not handle the login process, the dispatcher obtains a response message from the login page specified by the end user adapter.

See Also

OnCanViewPage (see page 6198)

OnBeforeDispatchPage (see page 6197)

OnPageAccessDenied (see page 6200)

LoginPage (see page 5516)

60.1.15.3.5 TPageDispatcher.OnPageAccessDenied Event

Occurs when the page dispatcher fails to validate the caller's right to access a requested Web page.

Pascal

```
property OnPageAccessDenied: TPageAccessDeniedEvent;
```

C++

```
__property TPageAccessDeniedEvent OnPageAccessDenied;
```

Description

Write an OnPageAccessDenied event handler to customize the response when a request for a specific Web page fails to be validated.

Tip: To access the Web request and response objects from within an OnPageAccessDenied event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnIsLoginRequired (see page 6199)

OnCanViewPage (see page 6198)

WebContext (see page 6003)

Request (see page 5974)

Response (see page 5975)

LoginPage (see page 5516)

60.1.15.3.6 TPageDispatcher.OnPageNotFound Event

Occurs when the dispatcher can't locate a registered Web page module with the name specified in an HTTP request message.

Pascal

```
property OnPageNotFound: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnPageNotFound;
```

Description

Write an OnPageNotFound event handler to generate a response when the dispatcher can't locate the Web page module to which an HTTP request message is addressed. For example, an OnPageNotFound event handler can provide the response object with content that indicates that the requested page does not exist or that represents simple content for a page that does not have an associated Web page module.

If the OnPageNotFound event handler sets its Handled parameter to true, the page dispatcher assumes it has updated the response message and returns it. If the OnPageNotFound leaves the Handled parameter set to false, the dispatcher raises an EPageDispatcherException exception after the event handler exits.

Tip: To access the Web request and response objects from within an OnPageNotFound event handler, call the global WebContext function to access the Web context and read its Request and Response properties.

See Also

OnBeforeDispatchPage (see page 6197)

OnAfterDispatchPage (see page 6196)

OnPageAccessDenied (see page 6200)

EPageDispatcherException (see page 6147)

WebContext (see page 6003)

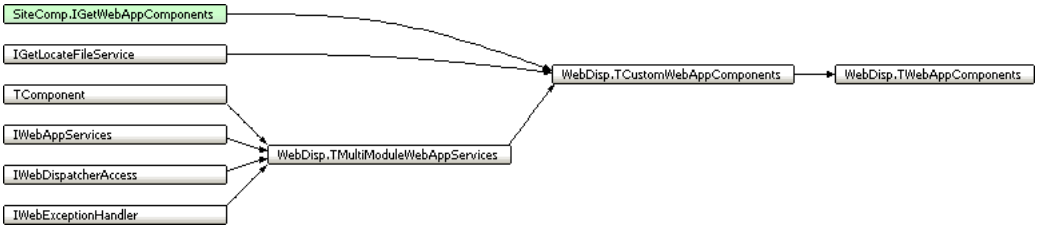
Request (see page 5974)

Response (see page 5975)

60.1.16 TWebAppComponents Class

TWebAppComponents holds the application components that provide the WebSnap application's functionality.

Class Hierarchy



Pascal

```
TWebAppComponents = class(TCustomWebAppComponents);
```

C++

```
class TWebAppComponents : public TCustomWebAppComponents;
```

File

WebDisp

Description

Use TWebAppComponents to centralize access to the main WebSnap components. It initializes the Web context for new request messages and locates the appropriate components to handle the request message. TWebAppComponents publishes properties that provide access to the components that perform many common tasks in a WebSnap application. These properties include

PageDispatcher (see page 6248), which locates the Web page modules to process incoming request messages.

AdapterDispatcher (see page 6245), which encodes calls to adapter actions and fields, and which handles HTML form submissions and requests for dynamic images by calling adapter actions and fields.

Sessions (see page 6248), which manages the sessions for each caller.

ApplicationAdapter (see page 6246), which implements the fields and actions of the Application variable in server-side scripts.

EndUserAdapter (see page 6247), which implements the fields and actions of the EndUser variable in server-side scripts.

UserListService (see page 6249), which maintains a list of end users and their passwords and access privileges.

LocateFileService (see page 6247), which controls access to the templates and server-side script include files that are used while processing requests.

DispatchActions (see page 6246), which maintains a list of action items that can respond to request messages.

The components these properties reference are optional. The WebSnap Application wizard automatically creates components and fills in these properties for any components that you select.

Add TWebAppComponents to the application module of the WebSnap application to provide centralized access to the other components that govern the behavior of the WebSnap application.

60.1.16.1 TWebAppComponents Members

The following tables list the members exposed by TWebAppComponents.

Protected Methods

	Name	Description
	Access_Request (see page 6225)	Returns the value of the Access_Request property.
	Access_Response (see page 6225)	Returns the value of the Access_Response property.
V	CreateContext (see page 6204)	Creates the Web context for the current request message.
V	CreateContext (see page 6225)	Creates the Web context for the current request message.
	DoAfterDispatch (see page 6226)	Generates an OnAfterDispatch (see page 6235) event.
	DoBeforeDispatch (see page 6226)	Generates an OnBeforeDispatch (see page 6235) event.
V	DoneModule (see page 6227)	Performs any final tasks before deactivating a Web module.
	FinishContext (see page 6227)	Performs any final tasks after the WebSnap application has finished processing a request.
	GetAdapterDispatcher (see page 6205)	Returns the value of the AdapterDispatcher (see page 6211) property.
	GetExceptionHandler (see page 6228)	Returns the object that handles exceptions that arise in the WebSnap application.
	GetLocateFileService (see page 6205)	Returns the value of the LocateFileService (see page 6213) property.
	GetPageDispatcher (see page 6205)	Returns the value of the PageDispatcher (see page 6214) property.
	GetWebApplicationInfo (see page 6206)	Returns the value of the ApplicationAdapter (see page 6212) property.
	GetWebEndUser (see page 6206)	Returns the value of the EndUserAdapter (see page 6213) property.
	GetWebUserList (see page 6206)	Returns the value of the UserListService (see page 6215) property.
	HandleException (see page 6228)	Responds to any exceptions not caught by the WebSnap application.
	HandleRequest (see page 6229)	Directs the processing of the current request message.
V	ImplFinishContext (see page 6207)	Provides the underlying implementation of the FinishContext (see page 6227) method.
V	ImplFinishContext (see page 6229)	Provides the underlying implementation of the FinishContext (see page 6227) method.
V	ImplHandleException (see page 6207)	Provides the underlying implementation of the HandleException (see page 6228) method.
V	ImplHandleException (see page 6230)	Provides the underlying implementation of the HandleException (see page 6228) method.
V	ImplHandleRequest (see page 6208)	Provides the underlying implementation for the HandleRequest (see page 6229) method.
V	ImplHandleRequest (see page 6230)	Provides the underlying implementation for the HandleRequest (see page 6229) method.
V	ImplInitContext (see page 6208)	Provides the underlying implementation for the InitContext (see page 6231) method.
V	ImplInitContext (see page 6231)	Provides the underlying implementation for the InitContext (see page 6231) method.
	InitContext (see page 6231)	Creates and initializes the Web context for the current Web request.
V	InitModule (see page 6209)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModule (see page 6232)	Initializes a Web module before the WebSnap application uses it to handle the current request.
V	InitModules (see page 6210)	Initializes all the currently active Web modules.
V	InitModules (see page 6233)	Initializes all the currently active Web modules.
V	ListRequestHandlers (see page 6210)	Fills an interface list with the interfaces of all dispatcher components in the WebSnap application.
	ModuleAdded (see page 6233)	Initializes a Web module when it is activated while processing the current request.
V	Notification (see page 6211)	Updates the TCustomWebAppComponents (see page 6201) properties when components are added to or removed from the application.

Protected Properties

	Name	Description
R	Request (see page 6234)	Provides access to the current request object.
R	Response (see page 6234)	Provides access to the current response object.

	WebModules (see page 6234)	Lists the Web modules available in the WebSnap application.

Public Methods

	Name	Description
	GetAdapterDispatcher (see page 1038)	This is GetAdapterDispatcher, a member of class IGetWebAppComponents.
	GetLocateFileService (see page 1038)	This is GetLocateFileService, a member of class IGetWebAppComponents.
	GetPageDispatcher (see page 1038)	This is GetPageDispatcher, a member of class IGetWebAppComponents.
	GetWebApplicationInfo (see page 1038)	This is GetWebApplicationInfo, a member of class IGetWebAppComponents.
	GetWebEndUser (see page 1039)	This is GetWebEndUser, a member of class IGetWebAppComponents.
	GetWebUserList (see page 1039)	This is GetWebUserList, a member of class IGetWebAppComponents.

Public Properties

	Name	Description
	WebApplicationInfo (see page 1039)	This is WebApplicationInfo, a member of class IGetWebAppComponents.
	WebEndUser (see page 1040)	This is WebEndUser, a member of class IGetWebAppComponents.

Published Events

	Name	Description
	OnAfterDispatch (see page 6249)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6250)	Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.
	OnException (see page 6250)	Occurs when an exception escapes all try blocks in the WebSnap application.

Published Properties

	Name	Description
	AdapterDispatcher (see page 6245)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6246)	Provides access to the application adapter.
	DispatchActions (see page 6246)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6247)	Provides access to an end-user adapter.
	LocateFileService (see page 6247)	Provides access to the file location service.
	PageDispatcher (see page 6248)	Provides access to the current page dispatcher object.
	Sessions (see page 6248)	Provides access to the sessions service.
	UserListService (see page 6249)	Provides access to the file location service.

60.1.16.2 TWebAppComponents Properties

The properties of the TWebAppComponents class are listed here.

Published Properties

	Name	Description
	AdapterDispatcher (see page 6245)	Provides access to the current adapter dispatcher object.
	ApplicationAdapter (see page 6246)	Provides access to the application adapter.
	DispatchActions (see page 6246)	Provides access to an action-based Web dispatcher.
	EndUserAdapter (see page 6247)	Provides access to an end-user adapter.
	LocateFileService (see page 6247)	Provides access to the file location service.
	PageDispatcher (see page 6248)	Provides access to the current page dispatcher object.
	Sessions (see page 6248)	Provides access to the sessions service.
	UserListService (see page 6249)	Provides access to the file location service.

60.1.16.2.1 TWebAppComponents.AdapterDispatcher Property

Provides access to the current adapter dispatcher object.

Pascal

```
property AdapterDispatcher: IAdapterDispatcher;
```

C++

```
__property IAdapterDispatcher AdapterDispatcher;
```

Description

Use AdapterDispatcher to get or set the interface of the adapter dispatcher. The adapter dispatcher is responsible for encoding the hidden fields and HREF values that represent calls to adapter fields and actions, and for interpreting the fields in incoming requests by calling adapters to execute their actions or to fetch image field values.

Typically, AdapterDispatcher is the interface of a TAdapterDispatcher object. If the WebSnap application does not include a TAdapterDispatcher component, or some other component that implements the IAdapterDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When AdapterDispatcher is nil (Delphi) or NULL (C++), the WebSnap application can't execute adapter actions or fetch image fields from adapters.

See Also

TAdapterDispatcher (see page 6148)

DispatchActions (see page 6212)

60.1.16.2.2 TWebAppComponents.ApplicationAdapter Property

Provides access to the application adapter.

Pascal

```
property ApplicationAdapter: IWebApplicationInfo;
```

C++

```
__property IWebApplicationInfo ApplicationAdapter;
```

Description

Use ApplicationAdapter to get or set the interface of the application adapter. The application adapter implements the fields and actions of the Application variable in server-side scripts.

Typically, ApplicationAdapter is the interface of a TApplicationAdapter object. If the WebSnap application does not include a TApplicationAdapter component, or some other component that implements the IWebApplicationInfo interface, the value of this property is nil (Delphi) or NULL (C++). When ApplicationAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the Application variable in server-side scripts.

See Also

TApplicationAdapter (see page 5439)

60.1.16.2.3 TWebAppComponents.DispatchActions Property

Provides access to an action-based Web dispatcher.

Pascal

```
property DispatchActions: IWebDispatchActions;
```

C++

```
__property IWebDispatchActions DispatchActions;
```

Description

Read DispatchActions to determine whether the WebSnap application contains a Web dispatcher. A Web dispatcher maintains a list of action items that respond to incoming request messages. Web dispatchers can appear in both WebSnap and Web broker applications.

The IWebDispatchActions interface that is the value of DispatchActions does nothing special other than identify its component as a Web dispatcher. You can, however, use this interface to obtain other interfaces that belong to the Web dispatcher.

Typically, DispatchActions is the interface of a TWebDispatcher object. If the WebSnap application does not include a TWebDispatcher component, or some other component that implements the IWebDispatchActions interface, the value of this property is nil (Delphi) or NULL (C++).

See Also

TWebModule

TWebDispatcher

60.1.16.2.4 TWebAppComponents.EndUserAdapter Property

Provides access to an end-user adapter.

Pascal

```
property EndUserAdapter: IWebEndUser;
```

C++

```
__property IWebEndUser EndUserAdapter;
```

Description

Use EndUserAdapter to get or set the interface of the end user adapter. An end user adapter maintains information about the current end user, and supports the actions for logging users in and out. End user adapters implement the fields and actions of the EndUser variable in server-side scripts.

Typically, EndUserAdapter is the interface of a TEndUserAdapter or TEndUserSessionAdapter object. If the WebSnap application does not include a component that implements the IWebEndUser interface, the value of this property is nil (Delphi) or NULL (C++). When EndUserAdapter is nil (Delphi) or NULL (C++), the WebSnap application can't use the EndUser variable in server-side scripts or take advantage of the login support that end user adapters provide.

See Also

TEndUserAdapter (see page 5764)

TEndUserSessionAdapter (see page 5792)

60.1.16.2.5 TWebAppComponents.LocateFileService Property

Provides access to the file location service.

Pascal

```
property LocateFileService: ILocateFileService;
```

C++

```
__property ILocateFileService LocateFileService;
```

Description

Use LocateFileService to get or set the interface of the file location service. The file location service controls access to the

templates and server-side script include files that are used while processing requests.

Typically, LocateFileService is the interface of a TLocateFileService object. If the WebSnap application does not include a component that implements the ILocateFileService interface, the value of this property is nil (Delphi) or NULL (C++). When LocateFileService is nil (Delphi) or NULL (C++), the WebSnap application can still use template files and include files, but you have less control over how the application locates these files (that is, they must reside in the current directory or references to them must be qualified).

See Also

TLocateFileService (see page 6220)

60.1.16.2.6 TWebAppComponents.PageDispatcher Property

Provides access to the current page dispatcher object.

Pascal

```
property PageDispatcher: IPageDispatcher;
```

C++

```
__property IPageDispatcher PageDispatcher;
```

Description

Use PageDispatcher to get or set the interface of the page dispatcher. The page dispatcher is responsible for interpreting requests for specific Web pages, checking the access rights for requested Web pages, and dispatching Web page requests to the appropriate Web page module.

Typically, PageDispatcher is the interface of a TPageDispatcher object. If the WebSnap application does not include a TPageDispatcher component, or some other component that implements the IPageDispatcher interface, the value of this property is nil (Delphi) or NULL (C++). When PageDispatcher is nil (Delphi) or NULL (C++), the WebSnap application does not automatically generate Web page content in response to requests that give the Web page name in their target URL.

See Also

TPageDispatcher (see page 6236)

AdapterDispatcher (see page 6211)

60.1.16.2.7 TWebAppComponents.Sessions Property

Provides access to the sessions service.

Pascal

```
property Sessions: ISessionsService;
```

C++

```
__property ISessionsService Sessions;
```

Description

Use Sessions to get or set the interface of the sessions service. The sessions service maintains Web sessions so that the WebSnap application can store temporary information about an end user.

Typically, Sessions is the interface of a TSessionsService object. If the WebSnap application does not include a component that implements the ISessionsService interface, the value of this property is nil (Delphi) or NULL (C++). When Sessions is nil (Delphi) or NULL (C++), the WebSnap application can't use Web sessions to store named information about the current end user.

See Also

TSessionsService (see page 6756)

60.1.16.2.8 TWebAppComponents.UserListService Property

Provides access to the file location service.

Pascal

```
property UserListService: IWebUserList;
```

C++

```
__property IWebUserList UserListService;
```

Description

Use UserListService to get or set the interface of the user list service. The user list service maintains a list of user names, passwords, and access rights. Using this information, it can validate logins and check a user's access rights.

Typically, UserListService is the interface of a TWebUserList object. If the WebSnap application does not include a component that implements the IWebUserList interface, the value of this property is nil (Delphi) or NULL (C++). When UserListService is nil (Delphi) or NULL (C++), the WebSnap application can still allow users to log in and check access rights, but the support for this must be coded using event handlers on the page dispatcher.

See Also

TWebUserList (see page 6796)

TPageDispatcher (see page 6236)

60.1.16.3 TWebAppComponents Events

The events of the TWebAppComponents class are listed here.

Published Events

	Name	Description
	OnAfterDispatch (see page 6249)	Occurs after the WebSnap application generates the content of a response message.
	OnBeforeDispatch (see page 6250)	Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.
	OnException (see page 6250)	Occurs when an exception escapes all try blocks in the WebSnap application.

60.1.16.3.1 TWebAppComponents.OnAfterDispatch Event

Occurs after the WebSnap application generates the content of a response message.

Pascal

```
property OnAfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnAfterDispatch;
```

Description

Write an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the WebSnap application a request. OnAfterDispatch occurs after the other components in the WebSnap application have attempted to generate a response for the current request. It does not occur if the response message has already been sent back

to the caller.

If the AfterDispatch event handler sets the Handled parameter to true, the WebSnap application sends the response back to the caller after the event handler exits. If it sets the Handled parameter to false, the WebSnap application does not send the response message.

Note: Although TMultiModuleWebAppServices introduces the OnAfterDispatch event, it never calls the event handler. Descendant classes add the code to generate this event after dispatching a request message.

See Also

OnBeforeDispatch (see page 6235)

60.1.16.3.2 TWebAppComponents.OnBeforeDispatch Event

Occurs before TMultiModuleWebAppServices passes an HTTP request onto the other components in the application for processing.

Pascal

```
property OnBeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent OnBeforeDispatch;
```

Description

Write an OnBeforeDispatch event handler to initialize the response to an HTTP request or to handle any messages that the other components in the WebSnap application can't handle. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing this component from trying to locate a component in the WebSnap application that can handle the message.

Note: Although TMultiModuleWebAppServices introduces the OnBeforeDispatch event, it never calls the event handler. Descendant classes add the code to generate this event before dispatching a request message.

See Also

OnAfterDispatch (see page 6235)

60.1.16.3.3 TWebAppComponents.OnException Event

Occurs when an exception escapes all try blocks in the WebSnap application.

Pascal

```
property OnException: TWebExceptionEvent;
```

C++

```
__property TWebExceptionEvent OnException;
```

Description

Write an OnException event handler respond to exceptions that arise while processing a request message. OnException occurs when an exception is not caught by the components that process the request message. If the OnException event handler does not handle the exception, TCustomWebAppComponents updates the response message so that it describes the exception to the caller.

See Also

OnAfterDispatch (see page 6235)

60.1.17 TWebSession Class

TWebSession stores temporary information associated with the user in a WebSnap application.

Class Hierarchy

Pascal

```
TWebSession = class(TAbstractWebSession, IWebSessionAccess);
```

C++

```
class TWebSession : public TAbstractWebSession, public IWebSessionAccess;
```

File

WebDisp

Description

TWebSession represents an active Web session. Use its Values property to get or set application-specific information about the current user. If your application uses sessions to store user information, you need to include a TSessionsService component in the Web application module.

A Web session stores information that persists over multiple calls from the same user. This information is stored as a set of Variant (see page 3421) values that are indexed by name. You can use these named Variants (see page 4973) to store application-defined information that is associated with a single user.

There is no need to instantiate session objects. A Session is created automatically when the WebSnap application receives an HTTP request message. The new session is given a unique identifier, which is added (in a cookie) to response messages, so that the same session can be used for subsequent requests from the same caller. The session persists until either the user logs out, a specified time elapses with no request messages from the caller, or the application explicitly calls the session's Terminate (see page 6255) method.

See Also

TSessionsService (see page 6756)

TAbstractWebContext (see page 5960)

TInvalidWebSession (see page 6216)

60.1.17.1 TWebSession Members

The following tables list the members exposed by TWebSession.

Protected Methods

	Name	Description
	GetSessionID (see page 6253)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 6253)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 6254)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 6254)	Sets the value of the TimoutMinutes property.
	SetValue (see page 6254)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Create (see page 6252)	This is the overview for the Create constructor overload.
	GetSessionsService (see page 6258)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 6258)	This is GetWebSession, a member of class IWebSessionAccess.
V	Terminate (see page 6255)	Terminates the session.
V	UpdateResponse (see page 6255)	Adds the session ID to a cookie in a response message.

60.1.17.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TWebSession.Create (ISessionsService, TSessionID) (see page 6252)	This is Create, a member of class TWebSession.
	TWebSession.Create (ISessionsService, TWebRequest, TWebResponse) (see page 6252)	This is Create, a member of class TWebSession.

60.1.17.2.1 TWebSession.Create Constructor (ISessionsService, TSessionID)

Pascal

```
constructor Create(ASessions: ISessionsService; AID: TSessionID); overload;
```

C++

```
__fastcall TWebSession(ISessionsService ASessions, TSessionID AID);
```

Description

This is Create, a member of class TWebSession.

60.1.17.2.2 TWebSession.Create Constructor (ISessionsService, TWebRequest, TWebResponse)

Pascal

```
constructor Create(ASessions: ISessionsService; ARequest: TWebRequest; AResponse:
TWebResponse); overload;
```

C++

```
__fastcall TWebSession(ISessionsService ASessions, TWebRequest ARequest, TWebResponse
AResponse);
```

Description

This is Create, a member of class TWebSession.

60.1.17.3 TWebSession Methods

The methods of the TWebSession class are listed here.

Protected Methods

	Name	Description
	GetSessionID (see page 6253)	Returns the value of the SessionID property.
	GetTimeoutMinutes (see page 6253)	Returns the value of the TimeoutMinutes property.
	GetValue (see page 6254)	Returns the value associated with a specified name.
	SetTimeoutMinutes (see page 6254)	Sets the value of the TimoutMinutes property.
	SetValue (see page 6254)	Assigns a value to a specified name.

Public Methods

	Name	Description
	Terminate (see page 6255)	Terminates the session.
	UpdateResponse (see page 6255)	Adds the session ID to a cookie in a response message.

60.1.17.3.1 TWebSession.GetSessionID Method

Returns the value of the SessionID property.

Pascal

```
function GetSessionID: string; override;
```

C++

```
virtual __fastcall UnicodeString GetSessionID();
```

Description

GetSessionID is the protected read implementation of the SessionID property.

In TAbstractWebSession, GetSessionID is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the SessionID property.

See Also

SessionID

60.1.17.3.2 TWebSession.GetTimeoutMinutes Method

Returns the value of the TimeoutMinutes property.

Pascal

```
function GetTimeoutMinutes: Integer; override;
```

C++

```
virtual __fastcall int GetTimeoutMinutes();
```

Description

GetTimeoutMinutes is the protected read implementation of the TimeoutMinutes property.

In TAbstractWebSession, GetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

TimeoutMinutes

SetTimeoutMinutes

60.1.17.3.3 TWebSession.GetValue Method

Returns the value associated with a specified name.

Pascal

```
function GetValue(const AName: string): Variant; override;
```

C++

```
virtual __fastcall Variant GetValue(const UnicodeString AName);
```

Description

GetValue is the protected read implementation of the Values property.

AName is the name of the value to return.

In TAbstractWebSession, GetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

Values

SetValue

60.1.17.3.4 TWebSession.SetTimeoutMinutes Method

Sets the value of the TimoutMinutes property.

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer); override;
```

C++

```
virtual __fastcall SetTimeoutMinutes(int AValue);
```

Description

SetTimeoutMinutes is the protected write implementation of the TimeoutMinutes property.

AValue is the number of minutes that can elapse before the session terminates itself.

In TAbstractWebSession, SetTimeoutMinutes is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the TimeoutMinutes property.

See Also

GetTimeoutMinutes

TimeoutMinutes

60.1.17.3.5 TWebSession.SetValue Method

Assigns a value to a specified name.

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant); override;
```

C++

```
virtual __fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

SetValue is the protected write implementation of the Value property.

AName is the name of the value to store.

AValue is the value to store under the specified name.

In TAbstractWebSession, SetValue is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to provide an implementation for the Values property.

See Also

GetValue

Values

60.1.17.3.6 TWebSession.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate; override;
```

C++

```
virtual __fastcall Terminate();
```

Description

Call Terminate to explicitly terminate a session. Once a session has been terminated, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimoutMinutes minutes.

See Also

TimeoutMinutes

Values

60.1.17.3.7 TWebSession.UpdateResponse Method

Adds the session ID to a cookie in a response message.

Pascal

```
procedure UpdateResponse(AResponse: TWebResponse); override;
```

C++

```
virtual __fastcall UpdateResponse(TWebResponse AResponse);
```

Description

Call UpdateResponse to add the value of the SessionID property to one of the response message's cookies. This allows the user's next request message to use the Values stored in this session.

AResponse is the Web response message to which a cookie with the session ID should be added.

See Also

SessionID

Values

60.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebSessionAccess (see page 6257)	This is class WebDisp.IWebSessionAccess.

60.2.1 IWebSessionAccess Interface

Class Hierarchy

WebDisp.IWebSessionAccess

Pascal

```
IWebSessionAccess = interface;
```

C++

```
__interface IWebSessionAccess;
```

File

WebDisp

Description

This is class WebDisp.IWebSessionAccess.

60.2.1.1 IWebSessionAccess Members

The following tables list the members exposed by IWebSessionAccess.

Public Methods

	Name	Description
	GetSessionsService (see page 6258)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 6258)	This is GetWebSession, a member of class IWebSessionAccess.

60.2.1.2 IWebSessionAccess Methods

The methods of the IWebSessionAccess class are listed here.

Public Methods

	Name	Description
	GetSessionsService (see page 6258)	This is GetSessionsService, a member of class IWebSessionAccess.
	GetWebSession (see page 6258)	This is GetWebSession, a member of class IWebSessionAccess.

60.2.1.2.1 IWebSessionAccess.GetSessionsService Method

Pascal

```
function GetSessionsService: ISessionsService;
```

C++

```
__fastcall ISessionsService GetSessionsService();
```

Description

This is GetSessionsService, a member of class IWebSessionAccess.

60.2.1.2.2 IWebSessionAccess.GetWebSession Method

Pascal

```
function GetWebSession: IWebSession;
```

C++

```
__fastcall IWebSession GetWebSession();
```

Description

This is GetWebSession, a member of class IWebSessionAccess.

60.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AdapterRequestParamsAsStrings (see page 6259)	This is function WebDisp.AdapterRequestParamsAsStrings.
	DispatchPageName (see page 6259)	This is function WebDisp.DispatchPageName.
	EncodeParamNameValues (see page 6260)	This is function WebDisp.EncodeParamNameValues.
	EncodeParamsAsHTTPQuery (see page 6260)	This is function WebDisp.EncodeParamsAsHTTPQuery.
	ExpandScriptException (see page 6260)	This is function WebDisp.ExpandScriptException.
	ExtractParamNameValues (see page 6261)	This is function WebDisp.ExtractParamNameValues.
	FindDefaultAppPageName (see page 6261)	This is function WebDisp.FindDefaultAppPageName.
	PageNameOfRequest (see page 6261)	This is function WebDisp.PageNameOfRequest.
	RaiseDispatchBlankPageNameException (see page 6262)	This is function WebDisp.RaiseDispatchBlankPageNameException.
	RaisePageDoesNotProvideContent (see page 6262)	This is function WebDisp.RaisePageDoesNotProvideContent.
	RaisePageNotFound (see page 6262)	This is function WebDisp.RaisePageNotFound.
	RedirectToPageName (see page 6263)	This is function WebDisp.RedirectToPageName.
	RegisterAdapterRequestIdentifier (see page 6263)	This is function WebDisp.RegisterAdapterRequestIdentifier.
	SafeRedirect (see page 6263)	Redirects a browser to another specified page.

60.3.1 WebDisp.AdapterRequestParamsAsStrings Function

Pascal

```
function AdapterRequestParamsAsStrings(AAdapterRequest: IUnknown): TStrings;
```

C++

```
TStrings AdapterRequestParamsAsStrings(IUnknown AAdapterRequest);
```

File

WebDisp

Description

This is function WebDisp.AdapterRequestParamsAsStrings.

60.3.2 WebDisp.DispatchPageName Function

Pascal

```
function DispatchPageName(const APageName: string; AResponse: TWebResponse; AFlags:
TDispatchPageFlags): Boolean;
```

C++

```
Boolean DispatchPageName(const UnicodeString APageName, TWebResponse AResponse,
TDispatchPageFlags AFlags);
```

File

WebDisp

Description

This is function WebDisp.DispatchPageName.

60.3.3 WebDisp.EncodeParamNameValues Function

Pascal

```
function EncodeParamNameValues(ParamValues: TStrings): string;
```

C++

```
UnicodeString EncodeParamNameValues(TStrings ParamValues);
```

File

WebDisp

Description

This is function WebDisp.EncodeParamNameValues.

60.3.4 WebDisp.EncodeParamsAsHTTPQuery Function

Pascal

```
function EncodeParamsAsHTTPQuery(AParams: TStrings): string;
```

C++

```
UnicodeString EncodeParamsAsHTTPQuery(TStrings AParams);
```

File

WebDisp

Description

This is function WebDisp.EncodeParamsAsHTTPQuery.

60.3.5 WebDisp.ExpandScriptException Function

Pascal

```
function ExpandScriptException(E: EScriptError): string;
```

C++

```
UnicodeString ExpandScriptException(EScriptError E);
```

File

WebDisp

Description

This is function WebDisp.ExpandScriptException.

60.3.6 WebDisp.ExtractParamNameValues Function

Pascal

```
procedure ExtractParamNameValues(const AParam: string; AList: TStrings);
```

C++

```
ExtractParamNameValues(const UnicodeString AParam, TStrings AList);
```

File

WebDisp

Description

This is function WebDisp.ExtractParamNameValues.

60.3.7 WebDisp.FindDefaultAppPageName Function

Pascal

```
function FindDefaultAppPageName: string;
```

C++

```
UnicodeString FindDefaultAppPageName();
```

File

WebDisp

Description

This is function WebDisp.FindDefaultAppPageName.

60.3.8 WebDisp.PageNameOfRequest Function

Pascal

```
function PageNameOfRequest(ARequest: TWebRequest): string;
```

C++

```
UnicodeString PageNameOfRequest(TWebRequest ARequest);
```

File

WebDisp

Description

This is function WebDisp.PageNameOfRequest.

60.3.9 WebDisp.RaiseDispatchBlankPageNameException Function

Pascal

```
procedure RaiseDispatchBlankPageNameException;
```

C++

```
RaiseDispatchBlankPageNameException();
```

File

WebDisp

Description

This is function WebDisp.RaiseDispatchBlankPageNameException.

60.3.10 WebDisp.RaisePageDoesNotProvideContent Function

Pascal

```
procedure RaisePageDoesNotProvideContent(const APageName: string);
```

C++

```
RaisePageDoesNotProvideContent(const UnicodeString APageName);
```

File

WebDisp

Description

This is function WebDisp.RaisePageDoesNotProvideContent.

60.3.11 WebDisp.RaisePageNotFound Function

Pascal

```
procedure RaisePageNotFound(const APageName: string);
```

C++

```
RaisePageNotFound(const UnicodeString APageName);
```

File

WebDisp

Description

This is function WebDisp.RaisePageNotFound.

60.3.12 WebDisp.RedirectToPageName Function

Pascal

```
function RedirectToPageName(const APageName: string; AParams: TStrings; AResponse:
TWebResponse; AFlags: TDispatchPageFlags): Boolean;
```

C++

```
Boolean RedirectToPageName(const UnicodeString APageName, TStrings AParams, TWebResponse
AResponse, TDispatchPageFlags AFlags);
```

File

WebDisp

Description

This is function WebDisp.RedirectToPageName.

60.3.13 WebDisp.RegisterAdapterRequestIdentifier Function

Pascal

```
procedure RegisterAdapterRequestIdentifier(const AIdentifier: string);
```

C++

```
RegisterAdapterRequestIdentifier(const UnicodeString AIdentifier);
```

File

WebDisp

Description

This is function WebDisp.RegisterAdapterRequestIdentifier.

60.3.14 WebDisp.SafeRedirect Function

Redirects a browser to another specified page.

Pascal

```
procedure SafeRedirect(AResponse: TWebResponse; const AURL: string);
```

C++

```
SafeRedirect(TWebResponse AResponse, const UnicodeString AURL);
```

File

WebDisp

Description

Use the SafeRedirect function to send a response which redirects a Web browser to another page. AResponse is the response object which will send the redirect. AURL is the URL for the page that the browser is being redirected to.

SafeRedirect differs from other redirect methods (such as an override of TWebResponse (see page 6276)'s SendRedirect method) in that it allows browsers to accept cookies. Other redirect methods don't allow cookies to be set during a redirect.

60.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TPageAccessDenied (see page 6265)	TPageAccessDenied indicates why a user was denied access to a requested Web page.

60.4.1 WebDisp.TPageAccessDenied Enumeration

TPageAccessDenied indicates why a user was denied access to a requested Web page.

Pascal

```
TPageAccessDenied = (
  adLoginRequired,
  adCantView
);
```

C++

```
enum TPageAccessDenied {
  adLoginRequired,
  adCantView
};
```

File

WebDisp

Description

The following table describes the possible values:

Value	Meaning
adLoginRequired	The caller has not logged in and the page specified by PageName requires a login. If the OnPageAccessDenied event handler does not set Handled to true, the dispatcher obtains a response message from the page specified by the end user adapter's LoginPage property. If the application does not include an end user adapter, or if that adapter does not specify a login page, the dispatcher raises an EPageDispatcherException (see page 6147) exception.
adCantView	The caller was denied access to the page by an OnCanViewPage event or because the caller's access rights do not match those required by the target Web page module. If the OnPageAccessDenied event handler does not set Handled to true, the dispatcher raises an EPageDispatcherException (see page 6147) exception.

60.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActivateWebModuleEvent (see page 6266)	This is type WebDisp.TActivateWebModuleEvent.
TCanViewPageEvent (see page 6267)	TCanViewPageEvent is the type for event handlers that indicate whether the current user can view a specified Web page.
TDispatchActionEvent (see page 6267)	TDispatchActionEvent is the type for event handlers that respond when an adapter action executes.
TDispatchActionHandledEvent (see page 6268)	TDispatchActionHandledEvent is the type for event handlers that can handle requests to execute an adapter action.
TDispatchImageEvent (see page 6269)	TDispatchImageEvent is the type for event handlers that respond when an adapter action gets a request for an image field value.
TDispatchImageHandledEvent (see page 6269)	TDispatchImageHandledEvent is the type for event handlers that can handle requests for an image field value.
TDispatchPageEvent (see page 6270)	TDispatchPageEvent is the type for event handlers that are called after a Web page module finishes handling an HTTP request.
TDispatchPageHandledEvent (see page 6270)	TDispatchPageHandledEvent is the type for event handlers that can handle HTTP request messages.
TDispatchPageParamsHandledEvent (see page 6271)	TDispatchPageParamsHandledEvent is the type for event handlers that are called before a WebSnap application assembles a response message redirecting the caller to another Web page.
TGetAdapterRequestParamsEvent (see page 6272)	TGetAdapterRequestParamsEvent is the type for event handlers that retrieve parameters for an adapter dispatcher.
TLocateFileServiceFindFileEvent (see page 6273)	TLocateFileServiceFindFileEvent is the type for event handlers that locates a file on disk in a WebSnap application.
TLocateFileServiceFindStreamEvent (see page 6273)	TLocateFileServiceFindStreamEvent is the type for event handlers that provide access to a requested file.
TLoginPagedHandledEvent (see page 6274)	This is type WebDisp.TLoginPagedHandledEvent.
TLoginRequiredEvent (see page 6274)	TLoginRequiredEvent is the type for event handlers that indicate whether the current user must log in before accessing a specified Web page.
TPageAccessDeniedEvent (see page 6275)	TPageAccessDeniedEvent is the type for event handlers that respond when a page dispatcher determines that the current user can't access the current page.
TWebRequest (see page 6276)	This is type WebDisp.TWebRequest.
TWebResponse (see page 6276)	This is type WebDisp.TWebResponse.

60.5.1 WebDisp.TActivateWebModuleEvent Type

Pascal

```
TActivateWebModuleEvent = procedure (Sender: TObject; WebModule: TComponent) of object;
```

C++

```
(Sender: TObject; WebModule: TComponent) ( TActivateWebModuleEvent)();
```

File

WebDisp

Description

This is type WebDisp.TActivateWebModuleEvent.

60.5.2 WebDisp.TCanViewPageEvent Type

TCanViewPageEvent is the type for event handlers that indicate whether the current user can view a specified Web page.

Pascal

```
TCanViewPageEvent = procedure (Sender: TObject; const PageName: string; var CanView: Boolean;
var AHandled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var CanView: Boolean; var AHandled: Boolean) (
TCanViewPageEvent)();
```

File

WebDisp

Description

TCanViewPageEvent is the type for event handlers of a page dispatcher that indicate whether a request can be forwarded to a Web page module.

Sender is the page dispatcher that is validating a request for the Web page module.

PageName is the name of the Web page module to which the request is redirected.

CanView returns an indication of whether the caller can access the specified page. For example, an OnCanViewPage event handler can set CanView to true when PageName specifies a page that does not require access rights.

Handled returns an indication of whether the CanView parameter accurately indicates whether the caller can access the requested page. If Handled returns false (the default), the dispatcher checks the callers access rights after the event handler exits.

See Also

Procedural types

Creating events

60.5.3 WebDisp.TDispatchActionEvent Type

TDispatchActionEvent is the type for event handlers that respond when an adapter action executes.

Pascal

```
TDispatchActionEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings) of
object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings) ( TDispatchActionEvent)();
```

File

WebDisp

Description

TDispatchActionEvent is the type for event handlers of an adapter dispatcher that respond when the adapter action executes a request.

Sender is the adapter dispatcher component that received the HTTP request message.

Action is the object that executes the action. If the dispatcher could not find an object to execute the action, this parameter is nil (Delphi) or NULL (C++).

Params lists the parameters for the action, in 'Name=Value' format. The '__id' parameter gives the name of the action to execute. If the current request is not a request to execute an action, Params is nil (Delphi) or NULL (C++).

See Also

Procedural types

Creating events

60.5.4 WebDisp.TDispatchActionHandledEvent Type

TDispatchActionHandledEvent is the type for event handlers that can handle requests to execute an adapter action.

Pascal

```
TDispatchActionHandledEvent = procedure (Sender: TObject; Action: TObject; Params: TStrings;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Action: TObject; Params: TStrings; var Handled: Boolean) (
TDispatchActionHandledEvent)();
```

File

WebDisp

Description

TDispatchActionHandledEvent is the type for event handlers of an adapter dispatcher that can handle requests to execute an adapter action. If the event handler executes the requested action, it can then prevent the default handling that would otherwise occur.

Sender is the adapter dispatcher component that received the HTTP request message.

Action is the object that would ordinarily execute the action. If the dispatcher could not find an object to execute the action, this parameter is nil (Delphi) or NULL (C++).

Params lists the parameters for the action, in 'Name=Value' format. The '__id' parameter gives the name of the action to execute. If the current request is not a request to execute an action, Params is nil (Delphi) or NULL (C++).

Handled returns an indication of how the adapter dispatcher should proceed after the event handler exits. If the event handler sets Handled to true, the dispatcher assumes that the event handler has taken care of the request. If the event handler leaves Handled set to false, the dispatcher continues with its normal processing of the request message.

See Also

TCustomAdapterDispatcher (see page 6168)

Procedural types

Creating events

60.5.5 WebDisp.TDispatchImageEvent Type

TDispatchImageEvent is the type for event handlers that respond when an adapter action gets a request for an image field value.

Pascal

```
TDispatchImageEvent = procedure (Sender: TObject; Image: TObject; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Image: TObject; Params: TStrings) ( TDispatchImageEvent)();
```

File

WebDisp

Description

TDispatchImageEvent is the type for event handlers of an adapter dispatcher that respond to requests for the image from an image field.

Sender is the adapter dispatcher component that received the HTTP request message.

Image is the object that provides the image. If the dispatcher could not find an object to provide the image, this parameter is nil (Delphi) or NULL (C++).

Params lists the parameters for the image, in 'Name=Value' format. The '__id' parameter gives the name of the requested image. If the current request is not a request to retrieve an image, Params is nil (Delphi) or NULL (C++).

See Also

Procedural types

Creating events

60.5.6 WebDisp.TDispatchImageHandledEvent Type

TDispatchImageHandledEvent is the type for event handlers that can handle requests for an image field value.

Pascal

```
TDispatchImageHandledEvent = procedure (Sender: TObject; Image: TObject; Params: TStrings; var
Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Image: TObject; Params: TStrings; var Handled: Boolean) (
TDispatchImageHandledEvent)();
```

File

WebDisp

Description

TDispatchImageHandledEvent is the type for event handlers of an adapter dispatcher that can handle requests for the image from an image field. If the event handler adds the requested image (or a substitute) to the current response message, it can then prevent the default handling that would otherwise occur.

Sender is the adapter dispatcher component that received the HTTP request message.

Image is the object that would ordinarily provide the image. If the dispatcher could not find an object to provide the image, this parameter is nil (Delphi) or NULL (C++).

Params lists the parameters for the image, in 'Name=Value' format. The '__id' parameter gives the name of the requested image. If the current request is not a request to retrieve an image, Params is nil (Delphi) or NULL (C++).

Handled returns an indication of how the adapter dispatcher should proceed after the event handler exits. If the event handler sets Handled to true, the dispatcher assumes that the event handler has taken care of the request. If the event handler leaves Handled set to false, the dispatcher continues with its normal processing of the request message.

See Also

TCustomAdapterDispatcher (see page 6168)

Procedural types

Creating events

60.5.7 WebDisp.TDispatchPageEvent Type

TDispatchPageEvent is the type for event handlers that are called after a Web page module finishes handling an HTTP request.

Pascal

```
TDispatchPageEvent = procedure (Sender: TObject; const PageName: string) of object;
```

C++

```
(Sender: TObject; const PageName: string) ( TDispatchPageEvent)();
```

File

WebDisp

Description

TDispatchPageEvent is the type for event handlers that are called after dispatching an HTTP request message to a Web page module.

Sender is the page dispatcher or Web page module that received the event.

PageName is the name of the Web page module that handled the request.

See Also

TCustomPageDispatcher (see page 6188)

TCustomWebPageModule (see page 6556)

Procedural types

Creating events

60.5.8 WebDisp.TDispatchPageHandledEvent Type

TDispatchPageHandledEvent is the type for event handlers that can handle HTTP request messages.

Pascal

```
TDispatchPageHandledEvent = procedure (Sender: TObject; const PageName: string; var Handled:
```

```
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Handled: Boolean) ( TDispatchPageHandledEvent)();
```

File

WebDisp

Description

TDispatchPageHandledEvent is the type for event handlers that are called before a Web page module handles an HTTP request message. The event handler allows the application handle the message in any way, and then prevent the default handling that would otherwise occur.

Sender is the page dispatcher or Web page module that receives the event.

PageName is the name of the Web page module to which the request was addressed.

Handled returns an indication of how the application should proceed after the event handler exits. If the event handler sets Handled to true, the application assumes that the event handler has taken care of the request and filled out the content of the response message. If the event handler leaves Handled set to false, the application continues with its normal processing of the request message.

See Also

TCustomPageDispatcher (see page 6188)

TCustomWebPageModule (see page 6556)

Procedural types

Creating events

60.5.9 WebDisp.TDispatchPageParamsHandledEvent Type

TDispatchPageParamsHandledEvent is the type for event handlers that are called before a WebSnap application assembles a response message redirecting the caller to another Web page.

Pascal

```
TDispatchPageParamsHandledEvent = procedure (Sender: TObject; const PageName: string; Params:
TStrings; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; Params: TStrings; var Handled: Boolean) (
TDispatchPageParamsHandledEvent)();
```

File

WebDisp

Description

TDispatchPageParamsHandledEvent is the type for event handlers that are called when an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost).

Sender is the page dispatcher or Web page module that receives the event.

PageName is the name of the Web page to which the request should be redirected.

Params is a set of name/value pairs that should be added to the redirect message for use by the Web page module when it

receives the redirected request.

Handled returns an indication of how the application should proceed after the event handler exits. If the event handler sets Handled to true, the application assumes that the event handler has taken care of setting up the response message to indicate the redirection. If the event handler leaves Handled set to false, the application continues with its normal processing of the redirection.

See Also

TCustomPageDispatcher (see page 6188)

TBaseAdapterAction (see page 5447)

TCustomWebPageModule (see page 6556)

Procedural types

Creating events

60.5.10 WebDisp.TGetAdapterRequestParamsEvent Type

TGetAdapterRequestParamsEvent is the type for event handlers that retrieve parameters for an adapter dispatcher.

Pascal

```
TGetAdapterRequestParamsEvent = procedure (Sender: TObject; Handler: IUnknown; var Identifer:
string; Params: TStrings) of object;
```

C++

```
(Sender: TObject; Handler: IUnknown; var Identifer: string; Params: TStrings) (
TGetAdapterRequestParamsEvent)();
```

File

WebDisp

Description

TGetAdapterRequestParamsEvent is the type for event handlers of an adapter dispatcher that fetch the parameters needed to execute an action or retrieve an image.

Sender is the adapter dispatcher component.

Handler is the interface of the adapter that has set up the Params parameter to reflect the current values.

Identifier is the name of the action to execute or the image to retrieve.

Params lists the parameters for the request. It lists the parameters in "Name=Value" strings. When the dispatcher is formatting parameters that are added to an HTML form, these values come from the adapter. When the dispatcher is handling an incoming request, these parameters come from the current request message. The event handler can change these parameters before they are added to the HTML form or passed to the adapter.

See Also

Procedural types

Creating events

60.5.11 WebDisp.TLocateFileServiceFindFileEvent Type

TLocateFileServiceFindFileEvent is the type for event handlers that locates a file on disk in a WebSnap application.

Pascal

```
TLocateFileServiceFindFileEvent = procedure (ASender: TObject; AComponent: TComponent; const
AFileName: string; var AFoundFile: string; var AHandled: Boolean) of object;
```

C++

```
(ASender: TObject; AComponent: TComponent; const AFileName: string; var AFoundFile: string;
var AHandled: Boolean) ( TLocateFileServiceFindFileEvent)();
```

File

WebDisp

Description

TLocateFileServiceFindFileEvent is the type for event handlers of a TLocateFileService (see page 6220) component that return a fully-qualified file name to indicate where a specified file can be found.

ASender is the TLocateFileService (see page 6220) component that is asked to locate a file.

AComponent is the component that uses or is otherwise associated with the file. For example, this can be a content producer component that uses a template file.

AFileName is the name of the file to locate. This is the name that the WebSnap application uses to refer to the desired file. Typically, it is the name of the Web page that the file is used to produce.

AFoundFile returns the fully-qualified file name where the requested file can be found.

AHandled returns true if the event handler supplies a file name as the value of AFoundFile.

See Also

TLocateFileService (see page 6220)

Procedural types

Creating events

60.5.12 WebDisp.TLocateFileServiceFindStreamEvent Type

TLocateFileServiceFindStreamEvent is the type for event handlers that provide access to a requested file.

Pascal

```
TLocateFileServiceFindStreamEvent = procedure (ASender: TObject; AComponent: TComponent; const
AFileName: string; var AFoundStream: TStream; var AOwned: Boolean; var AHandled: Boolean) of
object;
```

C++

```
(ASender: TObject; AComponent: TComponent; const AFileName: string; var AFoundStream: TStream;
var AOwned: Boolean; var AHandled: Boolean) ( TLocateFileServiceFindStreamEvent)();
```

File

WebDisp

Description

ASender is the TCustomLocateFileService (see page 6181) descendant that is asked to locate a file.

AComponent is the component that uses or is otherwise associated with the file. For example, this can be a content producer component that uses a template file.

AFileName is the name of the file to locate. This is the name that the WebSnap application uses to refer to the desired file. Typically, it is the name of a Web page.

AFoundStream returns a stream object from which the WebSnap application can read the contents of the requested file.

AOwned indicates whether the caller should free AFoundStream after reading the file contents. When AOwned returns true, the caller should free the stream object. When AOwned returns false, the caller should not free the stream object.

AHandled returns true if the event handler supplies a stream as the value of AFoundStream.

See Also

Procedural types

Creating events

60.5.13 WebDisp.TLoginPagedHandledEvent Type

Pascal

```
TLoginPagedHandledEvent = procedure (Sender: TObject; const PageName: string; var Handled:
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Handled: Boolean) ( TLoginPagedHandledEvent)();
```

File

WebDisp

Description

This is type WebDisp.TLoginPagedHandledEvent.

60.5.14 WebDisp.TLoginRequiredEvent Type

TLoginRequiredEvent is the type for event handlers that indicate whether the current user must log in before accessing a specified Web page.

Pascal

```
TLoginRequiredEvent = procedure (Sender: TObject; const PageName: string; var Required:
Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; var Required: Boolean) ( TLoginRequiredEvent)();
```

File

WebDisp

Description

TLoginRequiredEvent is the type for event handlers of a page dispatcher that indicate whether a specified Web page requires that the user be logged in.

Sender is the page dispatcher that is about to dispatch the HTTP request.

PageName is the name of the Web page module to which the request is redirected.

Required returns an indication of whether the caller must log in before accessing the specified page. On entry, this is set to the default associated with the specified Web page module. If the event handler returns with Required set to true, the dispatcher does not forward the request to the specified Web module. Rather, it generates an OnPageAccessDenied event, where the application can handle the login process.

See Also

Procedural types

Creating events

60.5.15 WebDisp.TPageAccessDeniedEvent Type

TPageAccessDeniedEvent is the type for event handlers that respond when a page dispatcher determines that the current user can't access the current page.

Pascal

```
TPageAccessDeniedEvent = procedure (Sender: TObject; const PageName: string; Reason:
TPageAccessDenied; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; const PageName: string; Reason: TPageAccessDenied; var Handled: Boolean) (
TPageAccessDeniedEvent)();
```

File

WebDisp

Description

TPageAccessDeniedEvent is the type for event handlers of a page dispatcher that respond when the current user can't be validated for accessing the requested page.

Sender is the page dispatcher that checked the caller's rights to access the requested page and found them insufficient.

PageName is the name of the Web page module to which the request was directed.

Reason indicates why the caller was not validated.

Handled returns an indication of whether the event handler has handled the situation (typically, by generating an appropriate response message). This value determines what the page dispatcher does next after the event handler exits, as indicated in the previous table.

See Also

Procedural types

Creating events

60.5.16 WebDisp.TWebRequest Type

Pascal

```
TWebRequest = HTTPApp.TWebRequest;
```

C++

```
HTTPApp.TWebRequest TWebRequest;
```

File

WebDisp

Description

This is type WebDisp.TWebRequest.

60.5.17 WebDisp.TWebResponse Type

Pascal

```
TWebResponse = HTTPApp.TWebResponse;
```

C++

```
HTTPApp.TWebResponse TWebResponse;
```

File

WebDisp

Description

This is type WebDisp.TWebResponse.

60.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sFailurePageParamName (see page 6277)	This is constant WebDisp.sFailurePageParamName.
sPageParamName (see page 6277)	This is constant WebDisp.sPageParamName.
sRequestIDParamName (see page 6278)	This is constant WebDisp.sRequestIDParamName.
sSessionCookieName (see page 6278)	This is constant WebDisp.sSessionCookieName.
sSuccessPageParamName (see page 6278)	This is constant WebDisp.sSuccessPageParamName.

60.6.1 WebDisp.sFailurePageParamName Constant

Pascal

```
sFailurePageParamName = '__fp';
```

C++

```
sFailurePageParamName = '__fp';
```

File

WebDisp

Description

This is constant WebDisp.sFailurePageParamName.

60.6.2 WebDisp.sPageParamName Constant

Pascal

```
sPageParamName = '__p';
```

C++

```
sPageParamName = '__p';
```

File

WebDisp

Description

This is constant WebDisp.sPageParamName.

60.6.3 WebDisp.sRequestIDParamName Constant

Pascal

```
sRequestIDParamName = '__id';
```

C++

```
sRequestIDParamName = '__id';
```

File

WebDisp

Description

This is constant WebDisp.sRequestIDParamName.

60.6.4 WebDisp.sSessionCookieName Constant

Pascal

```
sSessionCookieName = 'WebBrokerSessionID';
```

C++

```
sSessionCookieName = 'WebBrokerSessionID';
```

File

WebDisp

Description

This is constant WebDisp.sSessionCookieName.

60.6.5 WebDisp.sSuccessPageParamName Constant

Pascal

```
sSuccessPageParamName = '__sp';
```

C++

```
sSuccessPageParamName = '__sp';
```

File

WebDisp

Description

This is constant WebDisp.sSuccessPageParamName.

61 WebFact Namespace

This is namespace WebFact.

Topics

Name	Description
Classes (see page 6280)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6305)	The following table lists structs, records, enums in this documentation.
Types (see page 6306)	The following table lists types in this documentation.

Classes

	Name	Description
	TBaseWebModuleFactory (see page 6280)	TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.
	TBaseWebPageInfo (see page 6285)	TBaseWebPageInfo stores information about a page registered with a Web page module factory.
	TBaseWebPageModuleFactory (see page 6290)	TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TWebAppDataModuleFactory (see page 6295)	TWebAppDataModule is the factory object for WebSnap application data modules.
	TWebAppPageModuleFactory (see page 6297)	TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.
	TWebDataModuleFactory (see page 6299)	TWebDataModuleFactory is the factory class for WebSnap data modules.
	TWebPageInfo (see page 6301)	TWebPageInfo describes a WebSnap page.
	TWebPageModuleFactory (see page 6303)	TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

Structs, Records, Enums

	Name	Description
	TWebPageAccessFlags (see page 6305)	TWebPageAccessFlags and TWebPageAccess (see page 6306) determine what conditions must be met for a user to access the page that a Web Page module produces.

Types

Name	Description
TWebPageAccess (see page 6306)	This is type WebFact.TWebPageAccess.

61.1 Classes

The following table lists classes in this documentation.

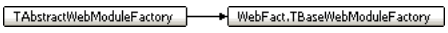
Classes

	Name	Description
	TBaseWebModuleFactory (see page 6280)	TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.
	TBaseWebPageInfo (see page 6285)	TBaseWebPageInfo stores information about a page registered with a Web page module factory.
	TBaseWebPageModuleFactory (see page 6290)	TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.
	TWebAppDataModuleFactory (see page 6295)	TWebAppDataModule is the factory object for WebSnap application data modules.
	TWebAppPageModuleFactory (see page 6297)	TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.
	TWebDataModuleFactory (see page 6299)	TWebDataModuleFactory is the factory class for WebSnap data modules.
	TWebPageInfo (see page 6301)	TWebPageInfo describes a WebSnap page.
	TWebPageModuleFactory (see page 6303)	TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

61.1.1 TBaseWebModuleFactory Class

TBaseWebModuleFactory is the base class for Web module factory objects whose generated Web modules do not create Web pages.

Class Hierarchy



Pascal

```
TBaseWebModuleFactory = class(TAbstractWebModuleFactory);
```

C++

```
class TBaseWebModuleFactory : public TAbstractWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating Web module instance. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instance for reuse with subsequent request messages.

TBaseWebModuleFactory descendants generate Web modules that do generate Web pages. These Web modules can contain components, such as adapters, that perform tasks for the WebSnap application.

Do not create instances of TBaseWebModuleFactory. Although TBaseWebModuleFactory provides an implementation for the abstract or, in C++ terminology, pure virtual methods introduced by its ancestors, it is intended only as a base class. Instead, create instances of a TBaseWebModuleFactory descendant such as TWebDataModuleFactory (see page 6299) or TWebAppDataModuleFactory (see page 6295).

See Also

TWebAppDataModuleFactory (see page 6295)

TWebDataModuleFactory (see page 6299)

TWebRequestHandler (see page 6633)

61.1.1.1 TBaseWebModuleFactory Members

The following tables list the members exposed by TBaseWebModuleFactory.

Protected Methods

	Name	Description
	Create (see page 6281)	Creates and initializes an instance of TBaseWebModuleFactory (see page 6280).
	GetCacheMode (see page 6282)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6282)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6283)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6283)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6284)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6283)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6284)	Prevents the WebSnap application from freeing the Web module instance.

61.1.1.2 TBaseWebModuleFactory.Create Constructor

Creates and initializes an instance of TBaseWebModuleFactory (see page 6280).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACreateMode: TWebModuleCreateMode;
ACacheMode: TWebModuleCacheMode);
```

C++

```
__fastcall TBaseWebModuleFactory(TComponentClass AComponentClass, TWebModuleCreateMode
ACreateMode, TWebModuleCacheMode ACacheMode);
```

Description

Call Create to instantiate a Web module factory. Typically, Web module factory objects are created in the initialization section of the unit for the Web module. The call to the constructor is added by the Web module wizard or WebSnap application wizard when it generates the unit for the Web module.

AComponentClass is the class of the Web module that this factory object creates. It becomes the value of the ComponentClass property.

ACreateMode describes whether the factory should create instances of the Web module for every request message, or only when the application explicitly requests the Web module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

ComponentClass

CreateMode

CacheMode

61.1.1.3 TBaseWebModuleFactory Methods

The methods of the TBaseWebModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 6282)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6282)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6283)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6283)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6284)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6283)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6284)	Prevents the WebSnap application from freeing the Web module instance.

61.1.1.3.1 TBaseWebModuleFactory.GetCacheMode Method

Returns the value of the CacheMode property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; override;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode();
```

Description

GetCacheMode is the protected read implementation of the CacheMode property. It returns the cache mode flag that was supplied to the constructor.

See Also

CacheMode

Create (see page 6281)

61.1.1.3.2 TBaseWebModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass property.

Pascal

```
function GetComponentClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass();
```

Description

GetComponentClass is the protected read implementation of the ComponentClass property. It returns the class of the Web page module this factory creates, as supplied to the Web page module factory constructor.

See Also

ComponentClass

Create (see page 6281)

61.1.1.3.3 TBaseWebModuleFactory.GetCreateMode Method

Returns the value of the CreateMode property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; override;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode();
```

Description

GetCreateMode is the protected read implementation of the CreateMode property. It returns the create mode flag that was supplied to the constructor.

See Also

CreateMode

Create (see page 6281)

61.1.1.3.4 TBaseWebModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebModuleFactory (see page 6280), GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppDataModuleFactory (see page 6295)) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6297)

61.1.1.3.5 TBaseWebModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetModule();
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. GetModule creates an instance of the class that the ComponentClass property specifies and returns it.

See Also

ComponentClass

CreateMode

CacheMode

61.1.1.3.6 TBaseWebModuleFactory.GetModuleName Method

Returns the value of the ModuleName property.

Pascal

```
function GetModuleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetModuleName();
```

Description

GetModuleName is the protected read implementation of the ModuleName property. It returns the name of the class that the ComponentClass property specifies.

See Also

ModuleName

ComponentClass

61.1.1.3.7 TBaseWebModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; override;
```

C++

```
virtual __fastcall PreventDestruction();
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TBaseWebModuleFactory (see page 6280), PreventDestruction changes the CacheMode property to caCache.

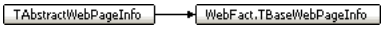
See Also

CacheMode

61.1.2 TBaseWebPageInfo Class

TBaseWebPageInfo stores information about a page registered with a Web page module factory.

Class Hierarchy



Pascal

```
TBaseWebPageInfo = class(TAbstractWebPageInfo);
```

C++

```
class TBaseWebPageInfo : public TAbstractWebPageInfo;
```

File

WebFact

Description

The TBaseWebPageInfo component stores basic information about an HTML page such as the filename, page name, page title, descriptive text, publish status, and access rights. This information is stored with the Web page module factory.

An instance of a TBaseWebPageInfo descendant, such as TWebPageInfo (see page 6301), is supplied to the constructor of the factory object that creates instances of a Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard. You can change this generated code, which appears in the generated unit for the Web page module, to alter the properties of the Web page information object. For example, you may want to add a ViewAccess string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageInfo (see page 6301)

TAbstractWebContext (see page 5960)

TWebAppPageModuleFactory (see page 6297)

TWebPageModuleFactory (see page 6303)

WebContext (see page 6003)

61.1.2.1 TBaseWebPageInfo Members

The following tables list the members exposed by TBaseWebPageInfo.

Protected Methods

	Name	Description
	Create (see page 6286)	Creates and initializes a TBaseWebPageInfo (see page 6285) object.
	GetIsLoginRequired (see page 6287)	Returns the value of the IsLoginRequired property.
	GetIsPublished (see page 6287)	Returns the value of the IsPublished property.
	GetPageDescription (see page 6287)	Returns the value of the PageDescription property.
	GetPageFile (see page 6288)	Returns the value of the PageFile property.
	GetPageHREF (see page 6288)	Returns the value of the PageHREF property.

	GetPageName (see page 6288)	Returns the value of the PageName property.
	GetPageTitle (see page 6289)	Returns the value of the PageTitle property.
	GetViewAccess (see page 6289)	Returns the value of the ViewAccess property.
	SetFactory (see page 6290)	Connects a specified Web page module factory with the Web page information object.

61.1.2.2 TBaseWebPageInfo.Create Constructor

Creates and initializes a TBaseWebPageInfo (see page 6285) object.

Pascal

```
constructor Create(AAccess: TWebPageAccess; const AFile: string; const AName: string; const
ATitle: string; const ADescription: string; const AViewAccess: string);
```

C++

```
__fastcall TBaseWebPageInfo(TWebPageAccess AAccess, const UnicodeString AFile, const
UnicodeString AName, const UnicodeString ATitle, const UnicodeString ADescription, const
UnicodeString AViewAccess);
```

Description

Typically, you do not need to create a call to the Web page information object constructor. The Web page module wizard generates code to make this call and pass the resulting object instance to the constructor of a Web page module factory. You may, however, want to edit the generated call to the constructor, changing some of the parameter values that the wizard supplies.

AAccess specifies access information for the associated Web page.

AFile specifies the name (or extension) of the file that the associated Web page module uses to generate its content. It becomes the value of the PageFile property.

AName is the logical name of the associated Web page. It becomes the value of the PageName property.

ATitle is the title that appears on the associated Web page. It becomes the value of the PageTitle property.

ADescription is a brief description of the associated Web page. It becomes the value of the PageDescription property.

AViewAccess is the view access string that represents a set of access rights needed to work with the associated Web page. It becomes the value of the ViewAccess property.

See Also

PageFile

PageName

PageTitle

PageDescription

ViewAccess

IsPublished

IsLoginRequired

61.1.2.3 TBaseWebPageInfo Methods

The methods of the TBaseWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetIsLoginRequired (see page 6287)	Returns the value of the IsLoginRequired property.
	GetIsPublished (see page 6287)	Returns the value of the IsPublished property.
	GetPageDescription (see page 6287)	Returns the value of the PageDescription property.
	GetPageFile (see page 6288)	Returns the value of the PageFile property.
	GetPageHREF (see page 6288)	Returns the value of the PageHREF property.
	GetPageName (see page 6288)	Returns the value of the PageName property.
	GetPageTitle (see page 6289)	Returns the value of the PageTitle property.
	GetViewAccess (see page 6289)	Returns the value of the ViewAccess property.
	SetFactory (see page 6290)	Connects a specified Web page module factory with the Web page information object.

61.1.2.3.1 TBaseWebPageInfo.GetIsLoginRequired Method

Returns the value of the IsLoginRequired property.

Pascal

```
function GetIsLoginRequired: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsLoginRequired();
```

Description

GetIsLoginRequired is the protected read implementation of the IsLoginRequired property. It returns true if wpLoginRequired is included in the access flags passed to the constructor.

See Also

IsLoginRequired

Create (see page 6286)

61.1.2.3.2 TBaseWebPageInfo.GetIsPublished Method

Returns the value of the IsPublished property.

Pascal

```
function GetIsPublished: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsPublished();
```

Description

GetIsPublished is the protected read implementation of the IsPublished property. It returns true if wpPublished is included in the access flags passed to the constructor.

See Also

IsPublished

Create (see page 6286)

61.1.2.3.3 TBaseWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription property.

Pascal

```
function GetPageDescription: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageDescription();
```

Description

GetPageDescription is the protected read implementation of the PageDescription property. It returns the description string that is passed to the constructor.

See Also

PageDescription

Create (see page 6286)

61.1.2.3.4 TBaseWebPageInfo.GetPageFile Method

Returns the value of the PageFile property.

Pascal

```
function GetPageFile: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageFile();
```

Description

GetPageFile is the protected read implementation of the PageFile property.

See Also

PageFile

61.1.2.3.5 TBaseWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF property.

Pascal

```
function GetPageHREF: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageHREF();
```

Description

GetPageHREF is the protected read implementation of the PageHREF property. It returns the HTTP path string that is passed to the constructor.

See Also

PageHREF

Create (see page 6286)

61.1.2.3.6 TBaseWebPageInfo.GetPageName Method

Returns the value of the PageName property.

Pascal

```
function GetPageName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageName();
```

Description

GetPageName is the protected read implementation of the PageName property. It returns the page name that is passed to the constructor.

See Also

PageName

Create (see page 6286)

61.1.2.3.7 TBaseWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle property.

Pascal

```
function GetPageTitle: string; override;
```

C++

```
virtual __fastcall UnicodeString GetPageTitle();
```

Description

GetPageTitle is the protected read implementation of the PageTitle property. It returns the title that is passed to the constructor.

See Also

PageTitle

Create (see page 6286)

61.1.2.3.8 TBaseWebPageInfo.GetViewAccess Method

Returns the value of the ViewAccess property.

Pascal

```
function GetViewAccess: string; override;
```

C++

```
virtual __fastcall UnicodeString GetViewAccess();
```

Description

GetViewAccess is the protected read implementation of the ViewAccess property. It returns the view access string that is passed to the constructor.

See Also

ViewAccess

Create (see page 6286)

61.1.2.3.9 TBaseWebPageInfo.SetFactory Method

Connects a specified Web page module factory with the Web page information object.

Pascal

```
procedure SetFactory(AFactory: TAbstractWebPageModuleFactory); override;
```

C++

```
virtual __fastcall SetFactory(TAbstractWebPageModuleFactory AFactory);
```

Description

SetFactory is the protected write implementation of the Factory property.

AFactory is the Web page module factory object that instantiates the Web page module that this object describes.

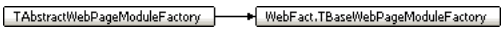
See Also

Factory

61.1.3 TBaseWebPageModuleFactory Class

TBaseWebPageModuleFactory is the base class for Web module factory objects whose generated Web modules create Web pages.

Class Hierarchy



Pascal

```
TBaseWebPageModuleFactory = class(TAbstractWebPageModuleFactory);
```

C++

```
class TBaseWebPageModuleFactory : public TAbstractWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating Web module instances. Applications can set flags for each factory object to indicate when it should create instances of its Web module and whether it can cache Web module instances for reuse with subsequent request messages.

TBaseWebPageModuleFactory descendants generate Web page modules. In addition to holding components that perform tasks for the WebSnap application, Web page modules can generate a Web page. The Web page is described by a Web page information object, which the Web page module factory makes available as its WebPageInfo property.

Do not create instances of TBaseWebPageModuleFactory. Although TBaseWebPageModuleFactory provides an implementation for the abstract or, in C++ terminology, pure virtual methods introduced by its ancestors, it is intended only as a base class. Instead, create instances of a TBaseWebPageModuleFactory descendant such as TWebPageModuleFactory (see page 6303) or TWebAppPageModuleFactory (see page 6297).

See Also

TWebAppPageModuleFactory (see page 6297)

TWebPageModuleFactory (see page 6303)

TWebRequestHandler (see page 6633)

TWebPageInfo (see page 6301)

61.1.3.1 TBaseWebPageModuleFactory Members

The following tables list the members exposed by TBaseWebPageModuleFactory.

Protected Methods

	Name	Description
	Create (see page 6291)	Creates and initializes an instance of TBaseWebPageModuleFactory (see page 6290).
	GetCacheMode (see page 6292)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6292)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6293)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6293)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6294)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6293)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6294)	Prevents the WebSnap application from freeing the Web module instance.

61.1.3.2 TBaseWebPageModuleFactory.Create Constructor

Creates and initializes an instance of TBaseWebPageModuleFactory (see page 6290).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACreateMode: TWebModuleCreateMode; ACacheMode: TWebModuleCacheMode);
```

C++

```
__fastcall TBaseWebPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCreateMode ACreateMode, TWebModuleCacheMode ACacheMode);
```

Description

Call Create to instantiate a Web page module factory. Typically, Web page module factory objects are created in the initialization section of the unit for the Web page module. The call to the constructor is added by the Web module wizard or WebSnap application wizard when it generates the unit for the Web page module.

AComponentClass is the class of the Web page module that this factory object creates. It becomes the value of the ComponentClass property.

AWebPageInfo is the Web page information object that describes the Web page that this factory's Web page module generates. It becomes the value of the WebPageInfo property.

ACreateMode describes whether the factory should create instances of the Web page module for every request message, or only when the application explicitly requests the Web page module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

WebPageInfo

ComponentClass

CreateMode

CacheMode

61.1.3.3 TBaseWebPageModuleFactory Methods

The methods of the TBaseWebPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetCacheMode (see page 6292)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6292)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6293)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6293)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6294)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	GetModule (see page 6293)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6294)	Prevents the WebSnap application from freeing the Web module instance.

61.1.3.3.1 TBaseWebPageModuleFactory.GetCacheMode Method

Returns the value of the CacheMode property.

Pascal

```
function GetCacheMode: TWebModuleCacheMode; override;
```

C++

```
virtual __fastcall TWebModuleCacheMode GetCacheMode();
```

Description

GetCacheMode is the protected read implementation of the CacheMode property. It returns the cache mode flag that was supplied to the constructor.

See Also

CacheMode

Create (see page 6291)

61.1.3.3.2 TBaseWebPageModuleFactory.GetComponentClass Method

Returns the value of the ComponentClass property.

Pascal

```
function GetComponentClass: TComponentClass; override;
```

C++

```
virtual __fastcall TComponentClass GetComponentClass();
```

Description

GetComponentClass is the protected read implementation of the ComponentClass property. It returns the class of the Web page module this factory creates, as supplied to the Web page module factory constructor.

See Also

ComponentClass

Create (see page 6291)

61.1.3.3.3 TBaseWebPageModuleFactory.GetCreateMode Method

Returns the value of the CreateMode property.

Pascal

```
function GetCreateMode: TWebModuleCreateMode; override;
```

C++

```
virtual __fastcall TWebModuleCreateMode GetCreateMode();
```

Description

GetCreateMode is the protected read implementation of the CreateMode property. It returns the create mode flag that was supplied to the constructor.

See Also

CreateMode

Create (see page 6291)

61.1.3.3.4 TBaseWebPageModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebPageModuleFactory (see page 6290), GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppPageModuleFactory (see page 6297)) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6297)

61.1.3.3.5 TBaseWebPageModuleFactory.GetModule Method

Creates an instance of the associated Web module.

Pascal

```
function GetModule: TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetModule();
```

Description

Call GetModule to obtain a new instance of the Web module that the factory creates. GetModule creates an instance of the class that the ComponentClass property specifies and returns it.

See Also

ComponentClass

CreateMode

CacheMode

61.1.3.3.6 TBaseWebPageModuleFactory.GetModuleName Method

Returns the value of the ModuleName property.

Pascal

```
function GetModuleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetModuleName();
```

Description

GetModuleName is the protected read implementation of the ModuleName property. It returns the name of the class that the ComponentClass property specifies.

See Also

ModuleName

ComponentClass

61.1.3.3.7 TBaseWebPageModuleFactory.PreventDestruction Method

Prevents the WebSnap application from freeing the Web module instance.

Pascal

```
procedure PreventDestruction; override;
```

C++

```
virtual __fastcall PreventDestruction();
```

Description

Call PreventDestruction if the WebSnap application should not free the Web module instance when the current HTTP request message has been fully processed. For example, if another Web module contains a reference to the Web module that this factory creates, you may not want to allow the WebSnap application to free the Web module that this factory creates while the other module exists.

In TBaseWebPageModuleFactory (see page 6290), PreventDestruction changes the CacheMode property to caCache.

See Also

CacheMode

61.1.4 TWebAppDataModuleFactory Class

TWebAppDataModule is the factory object for WebSnap application data modules.

Class Hierarchy

Pascal

```
TWebAppDataModuleFactory = class(TBaseWebModuleFactory);
```

C++

```
class TWebAppDataModuleFactory : public TBaseWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory is responsible for creating the module and also contains flags that specify how the module should be managed by the WebSnap application. TWebAppDataModuleFactory contains creation flags. Creation flags determine whether the module should be cached for future requests or destroyed after each request and created for each new request.

TWebAppDataModuleFactory identifies the module as an application module. The application module contains the components responsible for dispatching the HTTP request. WebSnap applications register a single application module class.

See Also

TWebAppDataModule (see page 6585)

TWebAppPageModuleFactory (see page 6297)

TWebDataModuleFactory (see page 6299)

TWebRequestHandler (see page 6633)

TWebPageModuleFactory (see page 6303)

61.1.4.1 TWebAppDataModuleFactory Members

The following tables list the members exposed by TWebAppDataModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6282)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6282)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6283)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6296)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6284)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6296)	Creates and initializes an instance of TWebAppDataModuleFactory (see page 6295).
	GetModule (see page 6283)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6284)	Prevents the WebSnap application from freeing the Web module instance.

61.1.4.2 TWebAppDataModuleFactory.Create Constructor

Creates and initializes an instance of TWebAppDataModuleFactory (see page 6295).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACacheMode: TWebModuleCacheMode =
caCache);
```

C++

```
__fastcall TWebAppDataModuleFactory(TComponentClass AComponentClass, TWebModuleCacheMode
ACacheMode = caCache);
```

Description

Call Create to instantiate a Web application module factory. Typically, the WebSnap application wizard generates the call to the constructor in the initialization section of the Web application module's unit.

AComponentClass is the class of the Web data module that this factory object creates. It becomes the value of the ComponentClass property.

ACacheMode indicates whether the application can cache Web application module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CacheMode

ComponentClass

61.1.4.3 TWebAppDataModuleFactory Methods

The methods of the TWebAppDataModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetIsAppModule (see page 6296)	Returns the value of the IsAppModule property.

61.1.4.3.1 TWebAppDataModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebModuleFactory, GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppDataModuleFactory) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6297)

61.1.5 TWebAppPageModuleFactory Class

TWebAppPageModuleFactory is the factory object for application Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TWebAppPageModuleFactory = class(TBaseWebPageModuleFactory);
```

C++

```
class TWebAppPageModuleFactory : public TBaseWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating the module and also contains flags that determine whether the module should be cached for future request or destroyed after each request and created for each new request.

Page module factories also contain information about the page such as the page name, description, and access rights. This information is available through the factory so that it can be retrieved without creating the module.

TWebAppPageModuleFactory identifies the module as an application module. The application module contains the components responsible for dispatching the HTTP request. WebSnap applications register a single application module class.

See Also

TWebAppPageModule (see page 6588)

TWebAppDataModuleFactory (see page 6295)

TWebPageModuleFactory (see page 6303)

TWebRequestHandler (see page 6633)

TWebPageInfo (see page 6301)

TWebDataModuleFactory (see page 6299)

61.1.5.1 TWebAppPageModuleFactory Members

The following tables list the members exposed by TWebAppPageModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6292)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6292)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6293)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6299)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6294)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6298)	Creates and initializes an instance of TWebAppPageModuleFactory (see page 6297).
	GetModule (see page 6293)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6294)	Prevents the WebSnap application from freeing the Web module instance.

61.1.5.2 TWebAppPageModuleFactory.Create Constructor

Creates and initializes an instance of TWebAppPageModuleFactory (see page 6297).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebAppPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCacheMode ACacheMode = caCache);
```

Description

Call Create to instantiate a Web page module factory. Typically, the WebSnap application wizard generates the call to the constructor in the initialization section of the Web application page module's unit.

AComponentClass is the class of the Web application page module that this factory object creates.

AWebPageInfo describes the attributes of the Web application page module, such as the page name, page title, access strings, and so on. This becomes the value of the WebPageInfo property.

ACacheMode indicates whether the factory can cache Web application page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CacheMode

ComponentClass

WebPageInfo

61.1.5.3 TWebAppPageModuleFactory Methods

The methods of the TWebAppPageModuleFactory class are listed here.

Protected Methods

	Name	Description
	GetIsAppModule (see page 6299)	Returns the value of the IsAppModule property.

61.1.5.3.1 TWebAppPageModuleFactory.GetIsAppModule Method

Returns the value of the IsAppModule property.

Pascal

```
function GetIsAppModule: Boolean; override;
```

C++

```
virtual __fastcall Boolean GetIsAppModule();
```

Description

GetIsAppModule is the protected read implementation of the IsAppModule property.

In TBaseWebPageModuleFactory, GetIsAppModule returns false. Descendant classes that generate Web application modules (such as TWebAppPageModuleFactory) override this method to return true.

See Also

IsAppModule

TWebAppPageModuleFactory (see page 6297)

61.1.6 TWebDataModuleFactory Class

TWebDataModuleFactory is the factory class for WebSnap data modules.

Class Hierarchy

Pascal

```
TWebDataModuleFactory = class(TBaseWebModuleFactory);
```

C++

```
class TWebDataModuleFactory : public TBaseWebModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. (In Delphi, the factory object is typically created in the initialization section of the module source file; in C++, it is added in a #pragma startup.)The factory is responsible for creating the module and also contains flags that specify how the module should be managed by the WebSnap application. TWebDataModuleFactory includes caching flags and creation flags. Caching flags determine whether the module should be cached for future request, or destroyed after each request and created for each new request. Creation flags determine whether the module should be automatically created for every request or created on demand.

The WebSnap application may register multiple Web data module classes. Each class is registered with an instance of TWebDataModuleFactory.

See Also

TWebDataModule (see page 6597)

TWebAppDataModuleFactory (see page 6295)

TWebAppPageModuleFactory (see page 6297)

TWebRequestHandler (see page 6633)

TWebPageModuleFactory (see page 6303)

61.1.6.1 TWebDataModuleFactory Members

The following tables list the members exposed by TWebDataModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6282)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6282)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6283)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6283)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6284)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6300)	Creates and initializes an instance of TWebDataModuleFactory (see page 6299).
	GetModule (see page 6283)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6284)	Prevents the WebSnap application from freeing the Web module instance.

61.1.6.2 TWebDataModuleFactory.Create Constructor

Creates and initializes an instance of TWebDataModuleFactory (see page 6299).

Pascal

```
constructor Create(AComponentClass: TComponentClass; ACreateMode: TWebModuleCreateMode =
crOnDemand; ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebDataModuleFactory(TComponentClass AComponentClass, TWebModuleCreateMode
ACreateMode = crOnDemand, TWebModuleCacheMode ACacheMode = caCache);
```

Description

Call Create to instantiate a Web data module factory. Typically, the Web module wizard generates the call to the constructor in the initialization section of the Web data module's unit.

AComponentClass is the class of the Web data module that this factory object creates.

ACreateMode describes whether the factory should create instances of the Web data module for every request message, or only when the application explicitly requests the Web data module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web data module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CreateMode

CacheMode

ComponentClass

61.1.7 TWebPageInfo Class

TWebPageInfo describes a WebSnap page.

Class Hierarchy

Pascal

```
TWebPageInfo = class(TBaseWebPageInfo);
```

C++

```
class TWebPageInfo : public TBaseWebPageInfo;
```

File

WebFact

Description

The TWebPageInfo module stores information relating to a web page, such as the template filename, page name, title, description, and access rights. This information is part of the web page module factory. This information is stored as part of the factory rather than as part of the module so that it can be accessed without creating the web page module.

An instance of a TWebPageInfo is supplied to the constructor of the factory object that creates instances of the Web page module. Typically, the code to create this object and supply it to the Web page factory's constructor is generated by a Web page module wizard (in Delphi, the generated code appears in the initialization section of the generated unit for the Web page module; in C++, it appears in a #pragma startup). You can change the generated code to alter the properties of the Web page information object. For example, you may want to add a ViewAccess string to allow your application to limit access to the Web page.

In a WebSnap application, you can access the Web page information object of any Web page module by calling the FindPageInfo method of the application's global Web context.

See Also

TWebPageModuleFactory (see page 6303)

TWebAppPageModuleFactory (see page 6297)

TWebModuleFactoryList (see page 6622)

TAbstractWebContext (see page 5960)

WebContext (see page 6003)

61.1.7.1 TWebPageInfo Members

The following tables list the members exposed by TWebPageInfo.

Protected Methods

	Name	Description
	GetIsLoginRequired (see page 6287)	Returns the value of the IsLoginRequired property.
	GetIsPublished (see page 6287)	Returns the value of the IsPublished property.
	GetPageDescription (see page 6287)	Returns the value of the PageDescription property.
	GetPageFile (see page 6288)	Returns the value of the PageFile property.
	GetPageHREF (see page 6288)	Returns the value of the PageHREF property.
	GetPageName (see page 6288)	Returns the value of the PageName property.
	GetPageTitle (see page 6289)	Returns the value of the PageTitle property.
	GetViewAccess (see page 6289)	Returns the value of the ViewAccess property.
	SetFactory (see page 6290)	Connects a specified Web page module factory with the Web page information object.

Public Methods

	Name	Description
	Create (see page 6302)	Creates and initializes a TWebPageInfo (see page 6301) object.

61.1.7.2 TWebPageInfo.Create Constructor

Creates and initializes a TWebPageInfo (see page 6301) object.

Pascal

```
constructor Create(AAccess: TWebPageAccess = [wpPublished]; const APageFile: string = '.html';
const APageName: string = ''; const ACaption: string = ''; const ADescription: string = '';
const AViewAccess: string = '');
```

C++

```
__fastcall TWebPageInfo(TWebPageAccess AAccess = [wpPublished], const UnicodeString APageFile
= '.html', const UnicodeString APageName = '', const UnicodeString ACaption = '', const
UnicodeString ADescription = '', const UnicodeString AViewAccess = '');
```

Description

Typically, you do not need create a call to the Web page information object constructor. The Web page module wizard generates code to make this call and pass the resulting object instance to the constructor of a Web page module factory. You may, however, want to edit the generated call to the constructor, changing some of the parameter values that the wizard supplies.

AAccess specifies access information for the associated Web page.

APageFile specifies the name (or extension) of the file that the associated Web page module uses to generate its content. It becomes the value of the PageFile property.

APageName is the logical name of the associated Web page. It becomes the value of the PageName property.

ACaption is the title that appears on the associated Web page. It becomes the value of the PageTitle property.

ADescription is a brief description of the associated Web page. It becomes the value of the PageDescription property.

AViewAccess is the view access string that represents a set of access rights needed to work with the associated Web page. It becomes the value of the ViewAccess property.

See Also

PageFile

PageName

PageTitle

PageDescription

ViewAccess

IsPublished

IsLoginRequired

61.1.8 TWebPageModuleFactory Class

TWebPageModuleFactory is the factory object for Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TWebPageModuleFactory = class(TBaseWebPageModuleFactory);
```

C++

```
class TWebPageModuleFactory : public TBaseWebPageModuleFactory;
```

File

WebFact

Description

All WebSnap modules are registered with the WebSnap application using a factory object. The factory object is responsible for creating the module and also contains flags that specify how the WebSnap application should manage the creation, destruction, and use of the Web module. TWebPageModuleFactory also contains flags that determine whether the module should be cached for future request or destroyed after each request and created for each new request.

Note: In the Delphi language, the factory object is created in the initialization section of the module source file.

Page module factories also contain information about the page such as the page name, description, and access rights. This information is available through the factory so that it can be retrieved without creating the module.

WebSnap applications can contain multiple Web page module factories (hence multiple Web page modules), each of which creates the Web page module for a different Web page that the application can create.

See Also

TWebPageModule (see page 6603)

TWebAppDataModuleFactory (see page 6295)

TWebAppPageModuleFactory (see page 6297)

TWebRequestHandler (see page 6633)

TWebPageInfo (see page 6301)

TWebDataModuleFactory (see page 6299)

61.1.8.1 TWebPageModuleFactory Members

The following tables list the members exposed by TWebPageModuleFactory.

Protected Methods

	Name	Description
	GetCacheMode (see page 6292)	Returns the value of the CacheMode property.
	GetComponentClass (see page 6292)	Returns the value of the ComponentClass property.
	GetCreateMode (see page 6293)	Returns the value of the CreateMode property.
	GetIsAppModule (see page 6293)	Returns the value of the IsAppModule property.
	GetModuleName (see page 6294)	Returns the value of the ModuleName property.

Public Methods

	Name	Description
	Create (see page 6304)	Creates and initializes an instance of TWebPageModuleFactory (see page 6303).
	GetModule (see page 6293)	Creates an instance of the associated Web module.
	PreventDestruction (see page 6294)	Prevents the WebSnap application from freeing the Web module instance.

61.1.8.2 TWebPageModuleFactory.Create Constructor

Creates and initializes an instance of TWebPageModuleFactory (see page 6303).

Pascal

```
constructor Create(AComponentClass: TComponentClass; AWebPageInfo: TAbstractWebPageInfo;
ACreateMode: TWebModuleCreateMode = crOnDemand; ACacheMode: TWebModuleCacheMode = caCache);
```

C++

```
__fastcall TWebPageModuleFactory(TComponentClass AComponentClass, TAbstractWebPageInfo
AWebPageInfo, TWebModuleCreateMode ACreateMode = crOnDemand, TWebModuleCacheMode ACacheMode =
caCache);
```

Description

Call Create to instantiate a Web page module factory. Typically, the Web page module wizard generates the call to the constructor in the Web page module's unit. In Delphi, the constructor call is placed in the initialization section of the unit. In C++, the call is placed in a #pragma startup.

AComponentClass is the class of the Web page module that this factory object creates.

AWebPageInfo describes the attributes of the Web page module, such as the page name, page title, access strings, and so on. This becomes the value of the WebPageInfo property.

ACreateMode describes whether the factory should create instances of the Web page module for every request message, or only when the application explicitly requests the Web page module. This becomes the value of the CreateMode property.

ACacheMode indicates whether the application can cache Web page module instances for reuse with subsequent request messages. This becomes the value of the CacheMode property.

See Also

CreateMode

CacheMode

ComponentClass

WebPageInfo

61.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWebPageAccessFlags (see page 6305)	TWebPageAccessFlags and TWebPageAccess (see page 6306) determine what conditions must be met for a user to access the page that a Web Page module produces.

61.2.1 WebFact.TWebPageAccessFlags Enumeration

TWebPageAccessFlags and TWebPageAccess (see page 6306) determine what conditions must be met for a user to access the page that a Web Page module produces.

Pascal

```
TWebPageAccessFlags = (
  wpPublished,
  wpLoginRequired
);
```

C++

```
enum TWebPageAccessFlags {
  wpPublished,
  wpLoginRequired
};
```

File

WebFact

Description

TWebPageAccessFlags are flags that signal limitations on how users can access a Web page that a Web page module produces. TWebPageAccess (see page 6306) is a set of zero or more TWebPageAccessFlags values. The following table lists the possible flags:

Flag	Meaning
wpPublished	The Web page module is automatically called in response to HTTP request messages that specify the Web page name as the pathinfo portion of their target URL. When included, the value of the IsPublished property is true. When not included, IsPublished is false.
wpLoginRequired	Users must log in before they can access the associated Web page. When included, the value of the IsLoginRequired property is true. When not included, IsLoginRequired is false.

61.3 Types

The following table lists types in this documentation.

Types

Name	Description
TWebPageAccess (see page 6306)	This is type WebFact.TWebPageAccess.

61.3.1 WebFact.TWebPageAccess Type

Pascal

```
TWebPageAccess = set of TWebPageAccessFlags;
```

C++

```
TWebPageAccessFlags TWebPageAccess;
```

File

WebFact

Description

This is type WebFact.TWebPageAccess.

62 WebForm Namespace

This is namespace WebForm.

Topics

Name	Description
Classes (see page 6310)	The following table lists classes in this documentation.
Interfaces (see page 6510)	The following table lists interfaces in this documentation.
Functions (see page 6519)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6522)	The following table lists structs, records, enums in this documentation.
Types (see page 6528)	The following table lists types in this documentation.
Constants (see page 6530)	The following table lists constants in this documentation.

Classes

	Name	Description
	TAbstractControlRequirements (see page 6311)	TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TAbstractFormRequirements (see page 6314)	This is class WebForm.TAbstractFormRequirements.
	TAdapterActionButton (see page 6316)	TAdapterActionButton generates HTML controls for executing adapter actions.
	TAdapterActionsListHelper (see page 6324)	TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.
	TAdapterCommandColumn (see page 6331)	TAdapterCommandColumn generates a column of action controls in an adapter grid.
	TAdapterCommandGroup (see page 6337)	TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.
	TAdapterControlGroup (see page 6341)	TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 6371) or TAdapterDisplayColumn (see page 6364).
	TAdapterDataDisplay (see page 6354)	TAdapterDataDisplay is the base class for components that display adapter fields as form fields.
	TAdapterDisplayColumn (see page 6364)	TAdapterDisplayColumn displays an Adapter field in a grid.
	TAdapterDisplayField (see page 6371)	TAdapterDisplayField displays an Adapter field as a form field.
	TAdapterDisplayReferenceGroup (see page 6381)	TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.
	TAdapterEditColumn (see page 6387)	TAdapterEditColumn displays an adapter field as a form field in a grid.
	TAdapterErrorList (see page 6397)	TAdapterErrorList is the error list display component.
	TAdapterFieldGroup (see page 6399)	TAdapterFieldGroup is a container for Adapter (see page 6402) display field components such as TAdapterDisplayField (see page 6371). TAdapterFieldGroup labels and fields in an HTML form.
	TAdapterForm (see page 6404)	TAdapterForm generates an HTML form displaying Adapter fields.

	TAdapterFormLayout (see page 6408)	TAdapterFormLayout lets child Web items adhere to layout rules imposed by their parent.
	TAdapterGrid (see page 6419)	TAdapterGrid is a container for Adapter (see page 6422) display field components such as TAdapterDisplayColumn (see page 6364). TAdapterGrid generates an HTML table.
	TControlRequirements (see page 6425)	TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TCustomAdapterActionButton (see page 6430)	TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.
	TCustomAdapterCommandColumn (see page 6442)	TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.
	TCustomAdapterCommandGroup (see page 6454)	TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 6337).
	TCustomAdapterDisplayColumn (see page 6461)	TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.
	TCustomAdapterDisplayField (see page 6467)	TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.
	TCustomAdapterEditColumn (see page 6475)	TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.
	TCustomAdapterErrorList (see page 6478)	TCustomAdapterErrorList is the base class adapter for components that display error lists.
	TCustomAdapterFieldGroup (see page 6483)	TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.
	TCustomAdapterForm (see page 6490)	TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.
	TCustomAdapterGrid (see page 6496)	TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.
	TDesigntimeWarnings (see page 6504)	This is class WebForm.TDesigntimeWarnings.
	TGeneratedFunctions (see page 6506)	This is class WebForm.TGeneratedFunctions.

Constants

Name	Description
sDisplayText (see page 6530)	This is constant WebForm.sDisplayText.
sFieldText (see page 6530)	This is constant WebForm.sFieldText.
sFormGet (see page 6531)	This is constant WebForm.sFormGet.
sFormPost (see page 6531)	This is constant WebForm.sFormPost.
sMultipartForm (see page 6531)	This is constant WebForm.sMultipartForm.

Functions

	Name	Description
	CalcDisplayFieldHTMLElementType (see page 6519)	This is function WebForm.CalcDisplayFieldHTMLElementType.
	CalcDisplayFieldViewMode (see page 6519)	This is function WebForm.CalcDisplayFieldViewMode.
	FindVariablesContainer (see page 6520)	This is function WebForm.FindVariablesContainer.
	FindVariablesContainerInParent (see page 6520)	This is function WebForm.FindVariablesContainerInParent.
	GetAdapterModeOfAdapterInParent (see page 6520)	This is function WebForm.GetAdapterModeOfAdapterInParent.
	MakeAdapterHTMLName (see page 6521)	This is function WebForm.MakeAdapterHTMLName.
	MakeAdapterVariableName (see page 6521)	This is function WebForm.MakeAdapterVariableName.

Interfaces

	Name	Description
	IAdapterDisplay (see page 6510)	This is class WebForm.IAdapterDisplay.
	IGetAdapterFormLayout (see page 6510)	This is class WebForm.IGetAdapterFormLayout.
	IGetAdapterModeOfDisplay (see page 6511)	This is class WebForm.IGetAdapterModeOfDisplay.
	IGetAdapterOfDisplay (see page 6512)	This is class WebForm.IGetAdapterOfDisplay.
	IGetAdapterVariableName (see page 6513)	This is class WebForm.IGetAdapterVariableName.
	IGetControlRequirements (see page 6515)	This is class WebForm.IGetControlRequirements.
	IGetFormRequirements (see page 6516)	This is class WebForm.IGetFormRequirements.
	IGetVariablesContainerOfDisplay (see page 6517)	This is class WebForm.IGetVariablesContainerOfDisplay.

Structs, Records, Enums

	Name	Description
	TActionButtonHideOption (see page 6522)	TActionButtonHideOptions (see page 6528) indicates when a control to represent an adapter action should not be generated.
	TColumnHTMLElementType (see page 6523)	This is record WebForm.TColumnHTMLElementType.
	TCommandHTMLElementType (see page 6523)	TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.
	TDisplayFieldHideOption (see page 6524)	TDisplayFieldHideOptions (see page 6528) indicates when a control to represent a dataset field should not be generated.
	TDisplayFieldHTMLElementType (see page 6525)	TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TDisplayFieldViewMode (see page 6525)	TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.
	TInputFieldHTMLElementType (see page 6526)	TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TLayoutState (see page 6527)	This is record WebForm.TLayoutState.

Types

Name	Description
TActionButtonHideOptions (see page 6528)	This is type WebForm.TActionButtonHideOptions.
TAdapterDataDisplayClass (see page 6528)	This is type WebForm.TAdapterDataDisplayClass.
TDisplayFieldHideOptions (see page 6528)	This is type WebForm.TDisplayFieldHideOptions.

62.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TAbstractControlRequirements (see page 6311)	TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TAbstractFormRequirements (see page 6314)	This is class WebForm.TAbstractFormRequirements.
	TAdapterActionButton (see page 6316)	TAdapterActionButton generates HTML controls for executing adapter actions.
	TAdapterActionsListHelper (see page 6324)	TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.
	TAdapterCommandColumn (see page 6331)	TAdapterCommandColumn generates a column of action controls in an adapter grid.
	TAdapterCommandGroup (see page 6337)	TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.
	TAdapterControlGroup (see page 6341)	TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 6371) or TAdapterDisplayColumn (see page 6364).
	TAdapterDataDisplay (see page 6354)	TAdapterDataDisplay is the base class for components that display adapter fields as form fields.
	TAdapterDisplayColumn (see page 6364)	TAdapterDisplayColumn displays an Adapter field in a grid.
	TAdapterDisplayField (see page 6371)	TAdapterDisplayField displays an Adapter field as a form field.
	TAdapterDisplayReferenceGroup (see page 6381)	TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.
	TAdapterEditColumn (see page 6387)	TAdapterEditColumn displays an adapter field as a form field in a grid.
	TAdapterErrorList (see page 6397)	TAdapterErrorList is the error list display component.
	TAdapterFieldGroup (see page 6399)	TAdapterFieldGroup is a container for Adapter (see page 6402) display field components such as TAdapterDisplayField (see page 6371). TAdapterFieldGroup labels and fields in an HTML form.
	TAdapterForm (see page 6404)	TAdapterForm generates an HTML form displaying Adapter fields.
	TAdapterFormLayout (see page 6408)	TAdapterFormLayout lets child Web items adhere to layout rules imposed by their parent.
	TAdapterGrid (see page 6419)	TAdapterGrid is a container for Adapter (see page 6422) display field components such as TAdapterDisplayColumn (see page 6364). TAdapterGrid generates an HTML table.
	TControlRequirements (see page 6425)	TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.
	TCustomAdapterActionButton (see page 6430)	TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.
	TCustomAdapterCommandColumn (see page 6442)	TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.
	TCustomAdapterCommandGroup (see page 6454)	TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 6337).
	TCustomAdapterDisplayColumn (see page 6461)	TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.
	TCustomAdapterDisplayField (see page 6467)	TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.
	TCustomAdapterEditColumn (see page 6475)	TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.
	TCustomAdapterErrorList (see page 6478)	TCustomAdapterErrorList is the base class adapter for components that display error lists.
	TCustomAdapterFieldGroup (see page 6483)	TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.
	TCustomAdapterForm (see page 6490)	TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.
	TCustomAdapterGrid (see page 6496)	TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.

	TDesigntimeWarnings (see page 6504)	This is class WebForm.TDesigntimeWarnings.
	TGeneratedFunctions (see page 6506)	This is class WebForm.TGeneratedFunctions.

62.1.1 TAbstractControlRequirements Class

TAbstractControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.

Class Hierarchy

Pascal

```
TAbstractControlRequirements = class;
```

C++

```
class TAbstractControlRequirements;
```

File

WebForm

Description

Components that generate HTML controls to represent the fields of an adapter (TAdapterControlGroup (see page 6341) and TAdapterDataDisplay (see page 6354) descendants) use descendants of TAbstractControlRequirements. The descendants of TAbstractControlRequirements store the variable and function declarations the adapter form must include before the control group can assemble HTML controls for the fields of the adapter it represents.

TAbstractControlRequirements introduces methods to

Store a specified function or variable (along with its value).

Query whether the function or variable declaration has already been added to the list of requirements.

Store an indication of whether the control requires that it be contained in a multipart form.

TAbstractControlRequirements is abstract or, in C++ terminology, pure virtual. It does not implement any of the methods it declares. WebSnap applications use TAbstractControlRequirements descendants such as TControlRequirements (see page 6425), which provide implementations for these methods.

See Also

TAdapterControlGroup (see page 6341)

TAdapterDataDisplay (see page 6354)

TAdapterForm (see page 6404)

TControlRequirements (see page 6425)

62.1.1.1 TAbstractControlRequirements Members

The following tables list the members exposed by TAbstractControlRequirements.

Public Methods

	Name	Description
	DefineFunction (see page 6312)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 6312)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 6313)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 6313)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 6314)	Stores the fact that the generated HTML form should be multipart.

62.1.1.2 TAbstractControlRequirements Methods

The methods of the TAbstractControlRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 6312)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 6312)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 6313)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 6313)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 6314)	Stores the fact that the generated HTML form should be multipart.

62.1.1.2.1 TAbstractControlRequirements.DefineFunction Method

Adds the declaration of a function to those that this control requirements object stores.

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineFunction(const UnicodeString AName, const UnicodeString ABody) = 0;
```

Description

DefineFunction stores the declaration of a function that has the name AName, where ABody is the text of the function body.

AName is the name of the function. This name is one of a standard set that the adapter form knows to call under specific circumstances. For example, the name can be the standard name of the function to generate an HTML button.

ABody is the body of the specified function. This value is typically customized to reflect the properties of a specific control in the form.

See Also

IsFunctionDefined (see page 6313)

DefineVariable (see page 6312)

62.1.1.2.2 TAbstractControlRequirements.DefineVariable Method

Adds the declaration of a variable to those that this control requirements object stores.

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string; AAdapter: Boolean);
virtual; abstract;
```

C++

```
virtual __fastcall DefineVariable(const UnicodeString AName, const UnicodeString AValue,
Boolean AAdapter) = 0;
```

Description

DefineVariable stores the name and value of a variable declaration.

AName is the name of the variable.

AValue is the value to assign to the variable.

AAdapter indicates whether this variable is the one that declares the adapter object whose field a control represents or whose set of fields a control group represents.

See Also

DefineFunction (see page 6312)

IsVariableDefined (see page 6313)

Adapter (see page 6352)

62.1.1.2.3 TAbstractControlRequirements.IsFunctionDefined Method

Indicates whether a function with the specified name has already been added to the list of control requirements.

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const UnicodeString AName) = 0;
```

Description

IsFunctionDefined returns true if the control requirements object already lists a function with the specified name. That is, IsFunctionDefined returns true if the DefineFunction (see page 6312) method has been called to define the specified function, and false otherwise.

AName is the name of the function.

See Also

DefineFunction (see page 6312)

IsVariableDefined (see page 6313)

62.1.1.2.4 TAbstractControlRequirements.IsVariableDefined Method

Indicates whether a variable with the specified name has already been added to the list of control requirements.

Pascal

```
function IsVariableDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const UnicodeString AName) = 0;
```

Description

IsVariableDefined returns true if the control requirements object already lists a variable with the specified name. That is, IsVariableDefined returns true if the DefineVariable (see page 6312) method has been called to define a value for the specified variable, and false otherwise.

AName is the name of the variable.

See Also

DefineVariable (see page 6312)

IsFunctionDefined (see page 6313)

62.1.1.2.5 TAbstractControlRequirements.RequiresMultipartForm Method

Stores the fact that the generated HTML form should be multipart.

Pascal

```
procedure RequiresMultipartForm; virtual; abstract;
```

C++

```
virtual __fastcall RequiresMultipartForm() = 0;
```

Description

Web items call this method if they need to be contained in a multipart form (for example, when a field in the form is represented by a reference to another file). A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

62.1.2 TAbstractFormRequirements Class

Class Hierarchy

Pascal

```
TAbstractFormRequirements = class;
```

C++

```
class TAbstractFormRequirements;
```

File

WebForm

Description

This is class WebForm.TAbstractFormRequirements.

62.1.2.1 TAbstractFormRequirements Members

The following tables list the members exposed by TAbstractFormRequirements.

Public Methods

	Name	Description
	DefineFunction (see page 6315)	This is DefineFunction, a member of class TAbstractFormRequirements.
	DefineVariable (see page 6315)	This is DefineVariable, a member of class TAbstractFormRequirements.
	IsFunctionDefined (see page 6315)	This is IsFunctionDefined, a member of class TAbstractFormRequirements.
	IsVariableDefined (see page 6316)	This is IsVariableDefined, a member of class TAbstractFormRequirements.

62.1.2.2 TAbstractFormRequirements Methods

The methods of the TAbstractFormRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 6315)	This is DefineFunction, a member of class TAbstractFormRequirements.
	DefineVariable (see page 6315)	This is DefineVariable, a member of class TAbstractFormRequirements.
	IsFunctionDefined (see page 6315)	This is IsFunctionDefined, a member of class TAbstractFormRequirements.
	IsVariableDefined (see page 6316)	This is IsVariableDefined, a member of class TAbstractFormRequirements.

62.1.2.2.1 TAbstractFormRequirements.DefineFunction Method

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineFunction(const UnicodeString AName, const UnicodeString ABody) = 0;
```

Description

This is DefineFunction, a member of class TAbstractFormRequirements.

62.1.2.2.2 TAbstractFormRequirements.DefineVariable Method

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string); virtual; abstract;
```

C++

```
virtual __fastcall DefineVariable(const UnicodeString AName, const UnicodeString AValue) = 0;
```

Description

This is DefineVariable, a member of class TAbstractFormRequirements.

62.1.2.2.3 TAbstractFormRequirements.IsFunctionDefined Method

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const UnicodeString AName) = 0;
```

Description

This is IsFunctionDefined, a member of class TAbstractFormRequirements.

62.1.2.2.4 TAbstractFormRequirements.IsVariableDefined Method

Pascal

```
function IsVariableDefined(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const UnicodeString AName) = 0;
```

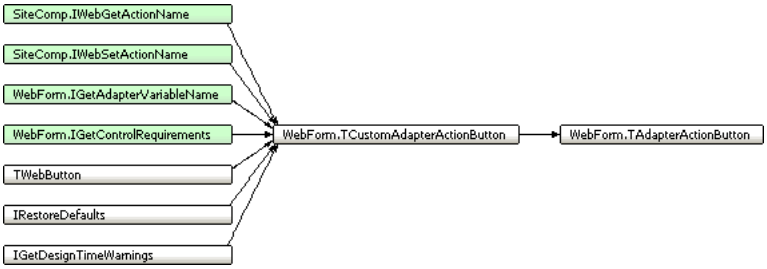
Description

This is IsVariableDefined, a member of class TAbstractFormRequirements.

62.1.3 TAdapterActionButton Class

TAdapterActionButton generates HTML controls for executing adapter actions.

Class Hierarchy



Pascal

```
TAdapterActionButton = class(TCustomAdapterActionButton);
```

C++

```
class TAdapterActionButton : public TCustomAdapterActionButton;
```

File

WebForm

Description

Add TAdapterActionButton to the hierarchy of Web items that an adapter page producer (TAdapterPageProducer) maintains to include a control in the HTML form that executes an adapter action. When the end user clicks the control, it sends a request message that executes the adapter action. Use the ActionName (see page 6318) property to specify the adapter action that the generated control executes.

TAdapterActionButton can generate a variety of controls, including labeled buttons, images that the user can click, and hypertext links. Use the DisplayType (see page 6320) property to specify the type of control you want. Other properties customize the appearance of the control, or indicate when it is included in HTML forms.

See Also

TAdapterPageProducer (see page 1136)

TCustomAdapterAction (see page 5579)

TAdapterForm (see page 6404)

62.1.3.1 TAdapterActionButton Members

The following tables list the members exposed by TAdapterActionButton.

Protected Methods

	Name	Description
	FindAssociatedAction (see page 6433)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 6433)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 6433)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 6433)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
V	GetCaption (see page 6434)	Returns the value of the Caption property.
	GetControlRequirements (see page 6434)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 6434)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 6434)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
V	GetLayoutAttributes (see page 6435)	Returns the layout object that specifies how the generated control should look.
V	ImplContent (see page 6435)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 6435)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
V	ImplRestoreDefaults (see page 6436)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
V	Notification (see page 6436)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 6436)	Returns the Web item to a default state.
	SetActionName (see page 6437)	This is SetActionName, a member of class TCustomAdapterActionButton.
V	SetCaption (see page 6437)	Sets the value of the Caption property.

Public Methods

	Name	Description
V	Create (see page 6431)	Creates and initializes an instance of TCustomAdapterActionButton (see page 6430).
V	Destroy (see page 6432)	Frees an instance of TCustomAdapterActionButton (see page 6430).
	GetActionName (see page 1087)	This is GetActionName, a member of class IWebGetActionName.
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	SetActionName (see page 1115)	This is SetActionName, a member of class IWebSetActionName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	ActionName (see page 6318)	Specifies the name of the action taken by the Web button.
	Align (see page 6318)	Specifies the horizontal alignment of the generated HTML control.
	Caption (see page 6319)	Specifies the caption that appears on the button.
	Custom (see page 6319)	Represents a tag option and its value in the generated INPUT tag.
	DisplayColumns (see page 6320)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6320)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6321)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6321)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6321)	Specifies the height of the generated element if it is a clickable image.

	ImageProducer (see page 6322)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6322)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 6323)	Specifies the name of the page to display after successfully executing the action.
	Style (see page 6323)	Adds style attributes to the generated button control.
	StyleRule (see page 6323)	Applies a style in the page producer's style sheet to the button.
	VAlign (see page 6324)	Specifies the vertical alignment of the generated HTML control.

62.1.3.2 TAdapterActionButton Properties

The properties of the TAdapterActionButton class are listed here.

Published Properties

	Name	Description
	ActionName (see page 6318)	Specifies the name of the action taken by the Web button.
	Align (see page 6318)	Specifies the horizontal alignment of the generated HTML control.
	Caption (see page 6319)	Specifies the caption that appears on the button.
	Custom (see page 6319)	Represents a tag option and its value in the generated INPUT tag.
	DisplayColumns (see page 6320)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6320)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6321)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6321)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6321)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 6322)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6322)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 6323)	Specifies the name of the page to display after successfully executing the action.
	Style (see page 6323)	Adds style attributes to the generated button control.
	StyleRule (see page 6323)	Applies a style in the page producer's style sheet to the button.
	VAlign (see page 6324)	Specifies the vertical alignment of the generated HTML control.

62.1.3.2.1 TAdapterActionButton.ActionName Property

Specifies the name of the action taken by the Web button.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;__property UnicodeString ActionName;
```

Description

Use ActionName to get or set the name of the adapter action that executes in response to requests sent when the user clicks this object's generated control. ActionName is the same as the ActionName property of the adapter action object that this control executes.

See Also

Caption

ActionName (see page 5586)

62.1.3.2.2 TAdapterActionButton.Align Property

Specifies the horizontal alignment of the generated HTML control.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the generated HTML control is positioned horizontally within its parent HTML element. If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each individual of the items within the grid generated to position them into rows.

Align indicates the desired value of the Align option of the HTML tag for the generated control.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6442)

DisplayColumns (see page 6438)

ItemCount (see page 5475)

62.1.3.2.3 TAdapterActionButton.Caption Property

Specifies the caption that appears on the button.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the caption that appears on the generated HTML button. The value of Caption is used as the value of a VALUE parameter on the HTML INPUT tag generated to implement the button.

See Also

Custom

GetCaption

SetCaption

62.1.3.2.4 TAdapterActionButton.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button. Each option and its value

has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Caption

Style

62.1.3.2.5 TAdapterActionButton.DisplayColumns Property

Indicates the number of columns to use when laying out the generated HTML control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

An adapter action may represent multiple commands. When the TCustomAdapterActionButton object generates controls for such an action, it must handle the layout of individual items for each of the action's commands. DisplayColumns controls how many items (controls) appear in a row before TCustomAdapterActionButton starts a new row of controls.

If the corresponding adapter action does not represent multiple commands (that is, if its ItemCount property is not greater than 1), the value of DisplayColumns has no effect.

If DisplayColumns is –1, the Web browser controls the layout of the adapter action's items.

See Also

ItemCount (see page 5475)

DisplayType (see page 6439)

62.1.3.2.6 TAdapterActionButton.DisplayType Property

Specifies the DisplayType of the generated HTML control.

Pascal

```
property DisplayType: TCommandHTMLElementType;
```

C++

```
__property TCommandHTMLElementType DisplayType;
```

Description

DisplayType controls the type of HTML element that this Web item generates.

When DisplayType is ctDefault (the default), TCustomAdapterActionButton uses the element type that the associated adapter action suggests. Change DisplayType to generate a different type of control or set of controls.

When DisplayType is ctImage or ctEventImages, you must set the ImageProducer property to a component that can provide the image or images.

See Also

ImageProducer (see page 6441)

Caption

GetDisplayStyle (see page 5457)

62.1.3.2.7 TAdapterActionButton.ErrorPageName Property

Specifies the name of the page to display when an error occurs executing the action.

Pascal

```
property ErrorPageName: string;
```

C++

```
__property UnicodeString ErrorPageName;
```

Description

Use ErrorPageName to indicate the Web page that should appear if the WebSnap application encounters an error when executing the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

PageName (see page 6441)

62.1.3.2.8 TAdapterActionButton.HideOptions Property

Controls when the generated HTML element appears in HTML forms.

Pascal

```
property HideOptions: TActionButtonHideOptions;
```

C++

```
__property TActionButtonHideOptions HideOptions;
```

Description

HideOptions controls when this Web item generates an HTML control for its associated action. This property is a set of flags that control when the adapter page producer suppresses the controls that this object generates. If HideOptions is an empty set, TCustomAdapterActionButton always generates a control for its associated adapter action.

See Also

GetVisible (see page 5462)

WebEnabled (see page 5473)

HasExecuteAccess (see page 5463)

62.1.3.2.9 TAdapterActionButton.ImageHeight Property

Specifies the height of the generated element if it is a clickable image.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

ImageHeight controls the height of the image that the user can click to execute the associated action. If ImageHeight is –1 (the default), the browser controls the height it uses to display the generated image.

ImageHeight is only used when the value of the DisplayType property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6441)

ImageWidth (see page 6441)

DisplayType (see page 6439)

62.1.3.2.10 TAdapterActionButton.ImageProducer Property

Specifies the interface that this adapter action button uses to generate images.

Pascal

```
property ImageProducer: IActionImageProducer;
```

C++

```
__property IActionImageProducer ImageProducer;
```

Description

Set ImageProducer when the value of the DisplayType is ctImage or ctEventImages. ImageProducer is the interface of the component that supplies the image or images that appear in HTML forms as the control that this Web item generates. When the user clicks on the generated image in the HTML form, the associated action executes.

See Also

DisplayType (see page 6439)

ImageHeight (see page 6440)

ImageWidth (see page 6441)

62.1.3.2.11 TAdapterActionButton.ImageWidth Property

Specifies the width of the generated element if it is a clickable image.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

ImageWidth controls the width of the image that the user can click to execute the associated action. If ImageWidth is –1 (the default), the browser controls the width it uses to display the generated image.

ImageWidth is only used when the value of the DisplayType property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6441)

ImageHeight (see page 6440)

DisplayType (see page 6439)

62.1.3.2.12 TAdapterActionButton.PageName Property

Specifies the name of the page to display after successfully executing the action.

Pascal

```
property PageName: string;
```

C++

```
__property UnicodeString PageName;
```

Description

Use PageName to indicate the Web page that should appear if the WebSnap application successfully executes the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

ErrorPageName (see page 6439)

62.1.3.2.13 TAdapterActionButton.Style Property

Adds style attributes to the generated button control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom

StyleRule

62.1.3.2.14 TAdapterActionButton.StyleRule Property

Applies a style in the page producer's style sheet to the button.

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style

Styles

StylesFile

Styles (see page 1157)

StylesFile (see page 1158)

62.1.3.2.15 TAdapterActionButton.VAlign Property

Specifies the vertical alignment of the generated HTML control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the generated HTML control is positioned vertically within its parent HTML element. VAlign indicates the desired value of the VAlign option of the HTML tag for the generated control. . If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each of the individual items within the grid generated to position them into rows.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 6438)

DisplayColumns (see page 6438)

ItemCount (see page 5475)

62.1.4 TAdapterActionsListHelper Class

TAdapterActionsListHelper assists a display component that represents a collection of adapter action buttons to work with the design-time editors.

Class Hierarchy

Pascal

```
TAdapterActionsListHelper = class(TObject);
```

C++

```
class TAdapterActionsListHelper : public TObject;
```

File

WebForm

Description

AdapterActionsListHelper is a helper object that works with components that generate a collection of controls for executing adapter actions. Such components include TAdapterCommandColumn (see page 6331) and TAdapterCommandGroup (see page 6337). TAdapterActionsListHelper handles common tasks that allow the display component to work with the IDE. These tasks include locating child Web items for specified actions, listing the child Web items (adapter action buttons) of the display component, and specifying the text of menu commands that operate on the set of child Web items.

See Also

TAdapterCommandColumn (see page 6331)

TAdapterCommandGroup (see page 6337)

TAdapterActionButton (see page 6316)

62.1.4.1 TAdapterActionsListHelper Members

The following tables list the members exposed by TAdapterActionsListHelper.

Protected Methods

	Name	Description
	FindAction (see page 6327)	Returns the child Web item that corresponds to the specified adapter action.
	GetActionsList (see page 6327)	Fills a list with the names of all the actions for which there are child Web items.
	GetAddAllFieldsItem (see page 6328)	Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.
	GetAddFieldsDlgCaption (see page 6328)	Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.
	GetAddFieldsItem (see page 6329)	Returns the string that the IDE uses for the command to display the Add Commands dialog.
	GetNewFieldPrefix (see page 6329)	Returns the string that is used to start the names of all automatically-generated Web items.
	GetVisibleActions (see page 6329)	Fills a list with the names of all the actions for which there are child Web items.
	IsActionInUse (see page 6330)	Indicates whether there is a child Web item that corresponds to a specified action.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6326)	Creates and initializes an instance of TAdapterActionsListHelper (see page 6324).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.

	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	AddDefaultActions (see page 6331)	Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

62.1.4.2 TAdapterActionsListHelper.Create Constructor

Creates and initializes an instance of TAdapterActionsListHelper (see page 6324).

Pascal

```
constructor Create(AComponent: TComponent; AHelper: TWebContainerComponentHelper);
```

C++

```
__fastcall TAdapterActionsListHelper(TComponent * AComponent, TWebContainerComponentHelper
AHelper);
```

Description

Do not instantiate TAdapterActionsListHelper (see page 6324). It is created by display components that use it to handle their interactions with the IDE.

AComponent is the adapter display component that uses TAdapterActionsListHelper (see page 6324) to interact with the IDE. This component is a Web item, such as TAdapterCommandColumn (see page 6331) or TAdapterCommandGroup (see page 6337), that assembles a set of HTML controls for executing adapter actions.

AHelper is the helper object that ACompnent uses to manage its list of child Web items.

See Also

TAdapterCommandColumn (see page 6331)

TAdapterCommandGroup (see page 6337)

62.1.4.3 TAdapterActionsListHelper Methods

The methods of the TAdapterActionsListHelper class are listed here.

Protected Methods

	Name	Description
	FindAction (see page 6327)	Returns the child Web item that corresponds to the specified adapter action.
	GetActionsList (see page 6327)	Fills a list with the names of all the actions for which there are child Web items.
	GetAddAllFieldsItem (see page 6328)	Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.
	GetAddFieldsDlgCaption (see page 6328)	Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.
	GetAddFieldsItem (see page 6329)	Returns the string that the IDE uses for the command to display the Add Commands dialog.
	GetNewFieldPrefix (see page 6329)	Returns the string that is used to start the names of all automatically-generated Web items.
	GetVisibleActions (see page 6329)	Fills a list with the names of all the actions for which there are child Web items.
	IsActionInUse (see page 6330)	Indicates whether there is a child Web item that corresponds to a specified action.

62.1.4.3.1 TAdapterActionsListHelper.FindAction Method

Returns the child Web item that corresponds to the specified adapter action.

Pascal

```
function FindAction(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindAction(const UnicodeString AName);
```

Description

FindAction searches through the list of child Web items for a component that corresponds to the specified adapter action.

AName is the name of the adapter action to locate.

FindAction returns the child Web item (typically a TAdapterActionButton (see page 6316) instance) that generates a control for the specified action. If there is no child Web item for the specified action, FindAction returns nil (Delphi) or NULL (C++).

See Also

GetActionsList (see page 6327)

GetVisibleActions (see page 6329)

IsActionInUse (see page 6330)

ActionName (see page 6438)

62.1.4.3.2 TAdapterActionsListHelper.GetActionsList Method

Fills a list with the names of all the actions for which there are child Web items.

Pascal

```
procedure GetActionsList(List: TStrings);
```

C++

```
__fastcall GetActionsList(TStrings List);
```

Description

GetActionsList searches through the variables that have been defined for child Web items that represent adapter actions. It adds the names of any actions it finds to the list, along with the class of the Web item that represents each action.

List is the list to fill with the names of adapter actions for which there are child Web items.

See Also

FindAction (see page 6327)

GetVisibleActions (see page 6329)

IsActionInUse (see page 6330)

ActionName (see page 6438)

62.1.4.3.3 TAdapterActionsListHelper.GetAddAllFieldsItem Method

Returns the string that the IDE uses for the command to add all default actions to the list of child Web items.

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

The IDE calls GetAddAllFieldsItem to customize the label of the command that adds all the default items to the list of child Web items. Because TAdapterActionsListHelper (see page 6324) is specialized for controls that contain adapter action buttons, it returns the string 'Add All Commands'.

See Also

GetAddFieldsDlgCaption (see page 6328)

GetAddFieldsItem (see page 6329)

GetNewFieldPrefix (see page 6329)

62.1.4.3.4 TAdapterActionsListHelper.GetAddFieldsDlgCaption Method

Returns the string that the IDE uses to label the dialog in which you can select child Web items to add.

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

The IDE calls GetAddFieldsDlgCaption to label the dialog that lists available child Web items. Because TAdapterActionsListHelper (see page 6324) is specialized for controls that contain adapter action buttons, it returns the string 'Add Commands'.

See Also

GetAddAllFieldsItem (see page 6328)

GetAddFieldsItem (see page 6329)

GetNewFieldPrefix (see page 6329)

62.1.4.3.5 TAdapterActionsListHelper.GetAddFieldsItem Method

Returns the string that the IDE uses for the command to display the Add Commands dialog.

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

The IDE calls GetAddFieldsItem to customize the label of the command that brings up a dialog where you can select child Web items to add. Because TAdapterActionsListHelper (see page 6324) is specialized for controls that contain adapter action buttons, it returns the string 'Add Commands ...'.

See Also

GetAddFieldsDlgCaption (see page 6328)

GetAddAllFieldsItem (see page 6328)

GetNewFieldPrefix (see page 6329)

62.1.4.3.6 TAdapterActionsListHelper.GetNewFieldPrefix Method

Returns the string that is used to start the names of all automatically-generated Web items.

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

When the IDE adds new child Web items, it generates names of the components it adds. Because TAdapterActionsListHelper (see page 6324) is specialized for controls that contain adapter action buttons, the names of those TAdapterActionButton (see page 6316) components are generated by prepending the string that this method returns to the name of the action the button executes.

GetNewFieldPrefix returns the string 'Cmd'.

See Also

GetAddFieldsDlgCaption (see page 6328)

GetAddAllFieldsItem (see page 6328)

GetAddFieldsItem (see page 6329)

62.1.4.3.7 TAdapterActionsListHelper.GetVisibleActions Method

Fills a list with the names of all the actions for which there are child Web items.

Pascal

```
function GetVisibleActions: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleActions();
```

Description

GetVisibleActions returns the list of child Web items that generate HTML controls for the associated component. Each element in this list is an adapter action button (TAdapterActionButton (see page 6316) object).

If you have explicitly added child Web items, the returned list contains the child Web items you added. Otherwise, if the AddDefaultActions (see page 6331) property is true, the list contains adapter action buttons for the default actions of the adapter. If AddDefaultActions (see page 6331) is false, the list is empty.

See Also

FindAction (see page 6327)

GetActionsList (see page 6327)

IsActionInUse (see page 6330)

TAdapterActionButton (see page 6316)

AddDefaultActions (see page 6331)

62.1.4.3.8 TAdapterActionsListHelper.IsActionInUse Method

Indicates whether there is a child Web item that corresponds to a specified action.

Pascal

```
function IsActionInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsActionInUse(const UnicodeString AName);
```

Description

IsActionInUse indicates whether the associated component already has a child Web item that executes a specified adapter action.

AName is the name of the adapter action.

IsActionInUse returns true if there is a child Web item that executes the specified action, false if there is no child Web item for the action.

See Also

FindAction (see page 6327)

GetActionsList (see page 6327)

GetVisibleActions (see page 6329)

ActionName (see page 6438)

62.1.4.4 TAdapterActionsListHelper Properties

The properties of the TAdapterActionsListHelper class are listed here.

Public Properties

	Name	Description
	AddDefaultActions (see page 6331)	Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

62.1.4.4.1 TAdapterActionsListHelper.AddDefaultActions Property

Indicates whether the display control represents all of the adapter's default actions if you do not explicitly add items to its WebFieldControls property.

Pascal

```
property AddDefaultActions: Boolean;
```

C++

```
__property Boolean AddDefaultActions;
```

Description

AddDefaultCommands determines what controls appear in the generated set of HTML controls if you do not explicitly add any adapter action buttons to the WebFieldControls property of the component that uses this TAdapterActionsListHelper (see page 6324). When AddDefaultCommands is true (the default), the associated component automatically adds controls for every default action of its adapter when you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the associated component only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls

DefaultWebComponents (see page 5237)

AddDefaultCommands (see page 6449)

AddDefaultCommands (see page 6459)

62.1.5 TAdapterCommandColumn Class

TAdapterCommandColumn generates a column of action controls in an adapter grid.

Class Hierarchy

Pascal

```
TAdapterCommandColumn = class(TCustomAdapterCommandColumn);
```

C++

```
class TAdapterCommandColumn : public TCustomAdapterCommandColumn;
```

File

WebForm

Description

Add TAdapterCommandColumn as a child Web item to an adapter grid (TAdapterGrid (see page 6419)). TAdapterCommandColumn adds a column to the grid that contains controls that execute adapter actions. Specify the controls in the cells of this column by adding adapter action buttons (TAdapterActionButton (see page 6316) objects) as child Web items of the TAdapterCommandColumn object. Each cell in the column contains all the adapter action buttons you add as child Web items. For each row in the grid, the column contains duplicates of the same set of adapter action buttons.

Use the TAdapterCommandColumn properties to specify which adapter action buttons appear in the column and to customize their layout and appearance. These properties include:

The AddDefaultCommands (see page 6334), WebFieldControls, and VisibleCommands (see page 6453) properties, which specify which adapter action buttons appear in the column.

The Caption (see page 6334) and CaptionAttributes (see page 6335) properties, which determine the appearance of the label at the top of the column.

The Align (see page 6334), VAlign (see page 6337), and DisplayColumns (see page 6335) properties, which determine the layout of controls inside each cell.

The Style (see page 6336) and StyleRule (see page 6336) properties, which let you apply a style from a stylesheet to the column.

See Also

TAdapterGrid (see page 6419)

TAdapterEditColumn (see page 6387)

TAdapterDisplayColumn (see page 6364)

TAdapterActionButton (see page 6316)

62.1.5.1 TAdapterCommandColumn Members

The following tables list the members exposed by TAdapterCommandColumn.

Protected Methods

	Name	Description
	FormatColumnData (see page 6445)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 6446)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 6446)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
V	GetLayoutAttributes (see page 6446)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6447)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6447)	Provides the underlying implementation for the Content method.
	NotifyAdapterChange (see page 6448)	Responds to changes that affect the associated adapter.

Protected Properties

	Name	Description
R	AdapterActionsListHelper (see page 6449)	Identifies the helper object that enables this component to work with the editor at design time.

Public Methods

	Name	Description
	Create (see page 6444)	Creates and initializes an instance of TCustomAdapterCommandColumn (see page 6442).
	Destroy (see page 6445)	Frees an instance of TCustomAdapterCommandColumn (see page 6442).
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterDisplayCharacteristics (see page 1019)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.

Public Properties

	Name	Description
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleCommands (see page 6453)	Lists the child Web items that generate controls that appear in cells of the column.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6334)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6334)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6334)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6335)	Specifies the attributes of the Caption property.
	Custom (see page 6335)	Represents a tag option and its value.
	DisplayColumns (see page 6335)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6336)	Adds style attributes to the generated caption.
	StyleRule (see page 6336)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6337)	Specifies the vertical alignment of the action control buttons within the column.

62.1.5.2 TAdapterCommandColumn Properties

The properties of the TAdapterCommandColumn class are listed here.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6334)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6334)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6334)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6335)	Specifies the attributes of the Caption property.
	Custom (see page 6335)	Represents a tag option and its value.
	DisplayColumns (see page 6335)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6336)	Adds style attributes to the generated caption.
	StyleRule (see page 6336)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6337)	Specifies the vertical alignment of the action control buttons within the column.

62.1.5.2.1 TAdapterCommandColumn.AddDefaultCommands Property

Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

Set AddDefaultCommands property to specify what should appear in the generated column when you do not explicitly add any adapter action buttons to the WebFieldControls property. When AddDefaultCommands is true (the default), the adapter command column automatically adds controls for every default action of the associated adapter if you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the adapter command column only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls

DefaultWebComponents (see page 5237)

VisibleCommands (see page 6453)

62.1.5.2.2 TAdapterCommandColumn.Align Property

Specifies the horizontal alignment of the action control buttons within the column.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the child HTML control are positioned horizontally within this column. Align indicates the desired value of the Align option of the HTML tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6453)

DisplayColumns (see page 6451)

62.1.5.2.3 TAdapterCommandColumn.Caption Property

Specifies the text label that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to get or set the caption for the column in the adapter grid. The caption is the text label that appears at the top of the column.

To change the appearance or layout of the caption, use the CaptionAttribute property.

See Also

CaptionAttributes (see page 6451)

62.1.5.2.4 TAdapterCommandColumn.CaptionAttributes Property

Specifies the attributes of the Caption property.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the column caption.

See Also

Caption (see page 6450)

62.1.5.2.5 TAdapterCommandColumn.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the column. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces.

TCustomAdapterCommandColumn inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 6452)

StyleRule (see page 6452)

62.1.5.2.6 TAdapterCommandColumn.DisplayColumns Property

Specifies the number of adapter action buttons that can appear in a single row.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Set DisplayColumns to organize the layout of adapter action buttons when the WebFieldControls property lists more than one adapter action button. TCustomAdapterCommandColumn places the adapter action buttons by rows in the cells of the adapter grid column. DisplayColumns limits the number of controls that can appear in a single row within each column cell. If the WebFieldControls property lists more adapter action buttons than DisplayColumns, then the cell contains multiple rows of controls.

See Also

WebFieldControls

62.1.5.2.7 TAdapterCommandColumn.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated column. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the column as the value of a STYLE option.

Note: Style affects the contents of the column cells. To specify display characteristics of the column's caption, use the CaptionAttributes property instead.

See Also

Custom (see page 6451)

StyleRule (see page 6452)

CaptionAttributes (see page 6451)

62.1.5.2.8 TAdapterCommandColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated column.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated column. The value

of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style property instead.

See Also

Custom (see page 6451)

Style (see page 6452)

Styles (see page 1157)

StylesFile (see page 1158)

62.1.5.2.9 TAdapterCommandColumn.VAlign Property

Specifies the vertical alignment of the action control buttons within the column.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the child HTML control are positioned vertically within the cells of this column. VAlign indicates the desired value of the VAlign option of the HTML tag.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

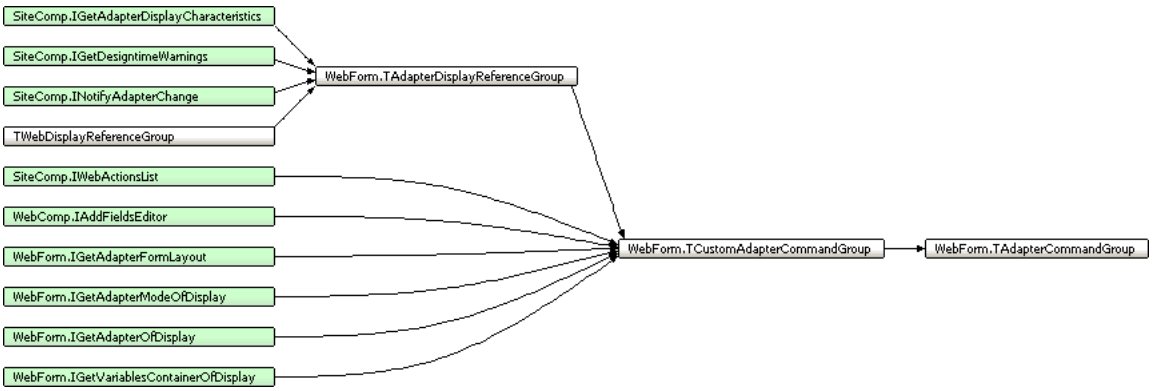
Align (see page 6450)

DisplayColumns (see page 6451)

62.1.6 TAdapterCommandGroup Class

TAdapterCommandGroup generates a group of buttons or other HTML elements that executes adapter actions.

Class Hierarchy



Pascal

```
TAdapterCommandGroup = class(TCustomAdapterCommandGroup);
```

C++

```
class TAdapterCommandGroup : public TCustomAdapterCommandGroup;
```

File

WebForm

Description

TAdapterCommandGroup is added to a TAdapterPageProducer to cause the page producer to generate a group of HTML elements that executes adapter actions. First add a TAdapterForm (see page 6404) to the TAdapterPageProducer and then add TAdapterCommandGroup as a child of TAdapterForm (see page 6404). TAdapterCommandGroup does not reference an adapter directly. Instead, it references a component that references an adapter. Typically, TAdapterCommandGroup references either TAdapterGrid (see page 6419) or TAdapterFieldGroup (see page 6399). TAdapterCommandGroup contains child components, such as TAdapterActionButton (see page 6316), that are responsible for rendering individual HTML elements, such as buttons. By default, TAdapterCommandGroup will automatically add one child component for each action in the adapter.

See Also

TAdapterDisplayReferenceGroup (see page 6381)

TCustomAdapterCommandGroup (see page 6454)

TAdapterForm (see page 6404)

TAdapterFieldGroup (see page 6399)

TAdapterGrid (see page 6419)

TAdapterActionButton (see page 6316)

62.1.6.1 TAdapterCommandGroup Members

The following tables list the members exposed by TAdapterCommandGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6383)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6384)	Responds when the DisplayComponent (see page 6386) property changes.
	GetAdapter (see page 6457)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterDisplayCharacteristics (see page 6384)	Indicates the display characteristics of the HTML this component generates.
	GetAdapterFormLayout (see page 6457)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 6457)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 6457)	Returns the adapter mode defined by the TAdapterPageProducer components.
	GetDesigntimeWarnings (see page 6384)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	ImplCanAddClass (see page 6458)	Checks the inheritance of the given component.
	ImplContent (see page 6458)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 6408) component.
	NotifyAdapterChange (see page 6385)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6385)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 6386)	Sets the value of the DisplayComponent (see page 6386) property.

Protected Properties

	Name	Description
	Adapter (see page 6459)	Provides read-only access to the Adapter that this command group uses.
	AdapterActionsListHelper (see page 6459)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Public Methods

	Name	Description
	Create (see page 6382)	Creates and initializes an instance of TAdapterDisplayReferenceGroup (see page 6381).
	Create (see page 6456)	Creator for TCustomAdapterCommandGroup (see page 6454).
	Destroy (see page 6382)	Frees an instance of TAdapterDisplayReferenceGroup (see page 6381).
	Destroy (see page 6456)	Calls the base class version, and then frees the memory used by the TAdapterActionsListHelper (see page 6324) component.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapter (see page 6513)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 1019)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 6511)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 6512)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6512)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.

Public Properties

	Name	Description
	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleCommands (see page 6460)	Provides read-only access to the TWebComponentList (see page 6025) that stores the list of visible components.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6340)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 6340)	Provides read and write access to the custom HTML element attributes, such as VAlign="top."
	DisplayComponent (see page 6340)	Associates the group with a component that generates the display controls.
	Style (see page 6341)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6341)	Provides read and write access to the internal string that stores the CSS style rule for the control group.

62.1.6.2 TAdapterCommandGroup Properties

The properties of the TAdapterCommandGroup class are listed here.

Published Properties

	Name	Description
	AddDefaultCommands (see page 6340)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 6340)	Provides read and write access to the custom HTML element attributes, such as VAlign="top."
	DisplayComponent (see page 6340)	Associates the group with a component that generates the display controls.
	Style (see page 6341)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6341)	Provides read and write access to the internal string that stores the CSS style rule for the control group.

62.1.6.2.1 TAdapterCommandGroup.AddDefaultCommands Property

Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

The AddDefaultCommands property provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

62.1.6.2.2 TAdapterCommandGroup.Custom Property

Provides read and write access to the custom HTML element attributes, such as VAlign="top."

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

The Custom property provides read and write access to the custom HTML element attributes, such as VAlign="top." The content of this string will be written as it is when generating HTML.

62.1.6.2.3 TAdapterCommandGroup.DisplayComponent Property

Associates the group with a component that generates the display controls.

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

Use DisplayComponent to associate the group of controls with a Web item that generates the display elements those controls influence. Typical values of DisplayComponent are TAdapterGrid or TAdapterFieldGroup.

TAdapterDisplayReferenceGroup descendants use the value of DisplayComponent to access the display controls to influence, and to access the adapter, whose information is displayed or manipulated.

See Also

TAdapterGrid (see page 6419)

TAdapterFieldGroup (see page 6399)

62.1.6.2.4 TAdapterCommandGroup.Style Property

Provides read and write access to the internal string that stores the CSS style, such as making the background color red.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

The Style property provides read and write access to the internal string that stores the CSS style, such as making the background color red.

62.1.6.2.5 TAdapterCommandGroup.StyleRule Property

Provides read and write access to the internal string that stores the CSS style rule for the control group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

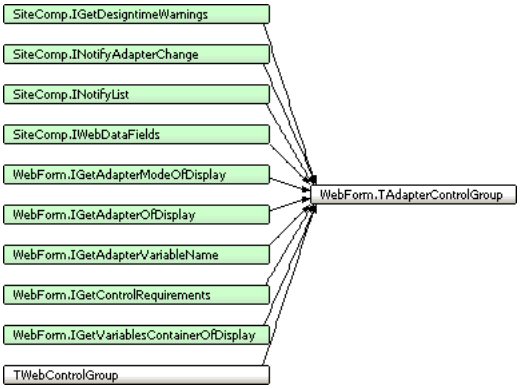
Description

The StyleRule property provides read and write access to the internal string that stores the CSS style rule for the control group. This property is used to reference a CSS style defined in TAdapterPageProducer. ::Styles.

62.1.7 TAdapterControlGroup Class

TAdapterControlGroup is a container for adapter display field components such as TAdapterDisplayField (see page 6371) or TAdapterDisplayColumn (see page 6364).

Class Hierarchy



Pascal

```
TAdapterControlGroup = class(TWebControlGroup, IWebDataFields, INotifyAdapterChange,
IGetAdapterVariableName, IGetAdapterOfDisplay, INotifyList, IGetVariablesContainerOfDisplay,
IGetDesigntimeWarnings, IGetAdapterModeOfDisplay, IGetControlRequirements);
```

C++

```
class TAdapterControlGroup : public TWebControlGroup, public IWebDataFields, public
INotifyAdapterChange, public IGetAdapterVariableName, public IGetAdapterOfDisplay, public
INotifyList, public IGetVariablesContainerOfDisplay, public IGetDesigntimeWarnings, public
IGetAdapterModeOfDisplay, public IGetControlRequirements;
```

File

WebForm

Description

Do not create instances of TAdapterControlGroup. Instead, use a descendant of TAdapterControlGroup that manages particular types of display field components. These descendants include TAdapterFieldGroup (see page 6399) and TAdapterGrid (see page 6419).

Descendants of TAdapterControlGroup are children of TAdapterPageProducer form components. Use the Web page editor to add TAdapterControlGroup descendants to an adapter form when building an HTML form.

See Also

TAdapterFieldGroup (see page 6399)

TAdapterGrid (see page 6419)

TAdapterPageProducer (see page 1136)

TAdapterForm (see page 6404)

TAdapter (see page 5190)

62.1.7.1 TAdapterControlGroup Members

The following tables list the members exposed by TAdapterControlGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6345)	Responds when the adapter associated with this control group changes.

	AddNotify (see page 6345)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6346)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6346)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6346)	Returns the value of the Adapter (see page 6352) property.
	GetAdapterModeOfAdapter (see page 6347)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6347)	Returns the value of the AdapterMode (see page 6353) property.
	GetAdapterVariableName (see page 6348)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetControlRequirements (see page 6348)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6348)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6349)	Fills a string list with the names of all of the associated adapter's fields.
	GetVisibleFields (see page 6349)	Returns the value of the VisibleFields (see page 6354) property.
V	ImplGetControlRequirements (see page 6350)	Provides the underlying implementation for the GetControlRequirements (see page 6348) method.
	IsFieldInUse (see page 6350)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
V	Notification (see page 6350)	Updates the TAdapterControlGroup (see page 6341) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6351)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6351)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6352)	Sets the value of the Adapter (see page 6352) property.

Public Methods

	Name	Description
V	Create (see page 6343)	Creates and initializes an instance of TAdapterControlGroup (see page 6341).
V	Destroy (see page 6344)	Frees an instance of TAdapterControlGroup (see page 6341).
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

Public Properties

	Name	Description
	Adapter (see page 6352)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6353)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 6353)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
R	VisibleFields (see page 6354)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

62.1.7.2 TAdapterControlGroup.Create Constructor

Creates and initializes an instance of TAdapterControlGroup (see page 6341).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterControlGroup(TComponent * AOwner);
```

Description

Call Create to instantiate TAdapterControlGroup (see page 6341) at runtime. TAdapterControlGroup (see page 6341) objects added at design time are instantiated automatically.

See Also

Owner

62.1.7.3 TAdapterControlGroup.Destroy Destructor

Frees an instance of TAdapterControlGroup (see page 6341).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterControlGroup();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

62.1.7.4 TAdapterControlGroup Methods

The methods of the TAdapterControlGroup class are listed here.

Protected Methods

	Name	Description
	AdapterChange (see page 6345)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6345)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6346)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6346)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6346)	Returns the value of the Adapter (see page 6352) property.
	GetAdapterModeOfAdapter (see page 6347)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6347)	Returns the value of the AdapterMode (see page 6353) property.
	GetAdapterVariableName (see page 6348)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetControlRequirements (see page 6348)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6348)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6349)	Fills a string list with the names of all of the associated adapter's fields.
	GetVisibleFields (see page 6349)	Returns the value of the VisibleFields (see page 6354) property.
	ImplGetControlRequirements (see page 6350)	Provides the underlying implementation for the GetControlRequirements (see page 6348) method.
	IsFieldInUse (see page 6350)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6350)	Updates the TAdapterControlGroup (see page 6341) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6351)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6351)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6352)	Sets the value of the Adapter (see page 6352) property.

62.1.7.4.1 TAdapterControlGroup.AdapterChange Method

Responds when the adapter associated with this control group changes.

Pascal

```
procedure AdapterChange; virtual;
```

C++

```
virtual __fastcall AdapterChange();
```

Description

AdapterChange is called automatically when the value of the Adapter (see page 6352) property changes, or when the application makes changes to the adapter that is the value of the Adapter (see page 6352) property.

AdapterChange performs the following tasks to respond to the change:

Updates the list of default adapter display field components that is the value of VisibleFields (see page 6354) when you have not explicitly assigned child Web items.

Notifies all of the child Web items listed in the WebFieldControls property that the adapter has changed.

Notifies any objects that registered interest by calling the AddNotify (see page 6345) method that the adapter has changed.

See Also

Adapter (see page 6352)

AddNotify (see page 6345)

WebFieldControls

VisibleFields (see page 6354)

62.1.7.4.2 TAdapterControlGroup.AddNotify Method

Registers interest so that the adapter control group notifies an object of changes to its adapter.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Other Web items in the adapter form call AddNotify to register an interest in when the control group's adapter changes. AddNotify adds the specified object to an internal list of interested objects. When the Adapter (see page 6352) property changes, it calls every object in this internal list to notify it of the change.

There is no need for the child Web items listed in the WebFieldControls property to call AddNotify. These objects are notified automatically.

ANotify is the object that registers an interest in change notifications.

See Also

AdapterChange (see page 6345)

RemoveNotify (see page 6351)

Adapter (see page 6352)

62.1.7.4.3 TAdapterControlGroup.CreateDefaultFields Method

Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.

Pascal

```
procedure CreateDefaultFields;
```

C++

```
__fastcall CreateDefaultFields();
```

Description

TAdapterControlGroup (see page 6341) calls CreateDefaultFields internally when it needs to generate child Web items for the fields of the adapter that the Adapter (see page 6352) property specifies. The resulting list becomes the value of the VisibleFields (see page 6354) property if you have not explicitly added any child Web items to the WebFieldControls property and AddDefaultFields (see page 6353) is true.

See Also

VisibleFields (see page 6354)

Adapter (see page 6352)

AddDefaultFields (see page 6353)

WebFieldControls

62.1.7.4.4 TAdapterControlGroup.FindField Method

Returns the child Web item that represents the value of a specified adapter field.

Pascal

```
function FindField(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindField(const UnicodeString AName);
```

Description

FindField searches the list of child Web items for one that represents the specified adapter field. If it finds a Web item that represents the specified field, it returns that adapter display field component. Otherwise, FindField returns nil (Delphi) or NULL (C++).

AName is the name of a field that the adapter control group's adapter contains.

See Also

VisibleFields (see page 6354)

GetFieldsList (see page 6349)

Adapter (see page 6352)

62.1.7.4.5 TAdapterControlGroup.GetAdapter Method

Returns the value of the Adapter (see page 6352) property.

Pascal

```
function GetAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetAdapter();
```

Description

GetAdapter is the protected read implementation of the Adapter (see page 6352) property.

See Also

Adapter (see page 6352)

62.1.7.4.6 TAdapterControlGroup.GetAdapterModeOfAdapter Method

Returns the current mode of the control group's adapter.

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

GetAdapterModeOfAdapter returns the current mode of the adapter that the Adapter (see page 6352) property specifies. If the adapter's mode is not currently set, GetAdapterModeOfAdapter returns the value of the AdapterMode (see page 6353) property.

See Also

AdapterMode (see page 6353)

Adapter (see page 6352)

GetAdapterModeOfDisplay (see page 6347)

62.1.7.4.7 TAdapterControlGroup.GetAdapterModeOfDisplay Method

Returns the value of the AdapterMode (see page 6353) property.

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfDisplay();
```

Description

GetAdapterModeOfDisplay returns the value of the AdapterMode (see page 6353) property. This is the value that influences the type and behavior of the HTML controls that child Web items generate. It may differ from the current mode of the control group's adapter, which the GetAdapterModeOfAdapter (see page 6347) method returns.

See Also

AdapterMode (see page 6353)

GetAdapterModeOfAdapter (see page 6347)

62.1.7.4.8 TAdapterControlGroup.GetAdapterVariableName Method

Returns the variable name that refers to the control group's adapter in server-side script.

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

TAdapterControlGroup (see page 6341) uses GetAdapterVariableName internally when generating server-side script that refers to its adapter.

See Also

Adapter (see page 6352)

62.1.7.4.9 TAdapterControlGroup.GetControlRequirements Method

Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

The adapter form object that includes this control group as a child Web item calls GetControlRequirements to obtain declarations for the variables and functions that the control group uses to assemble HTML controls for its adapter's fields.

ARequirements is the control requirements object that collects a list of required variables and functions.

GetControlRequirements simply calls the ImplGetControlRequirements (see page 6350) method, which adds a variable that represents the control group's adapter to ARequirements if it is not already there.

See Also

ImplGetControlRequirements (see page 6350)

TAdapterForm (see page 6404)

GetAdapterVariableName (see page 6348)

62.1.7.4.10 TAdapterControlGroup.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the adapter control group needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the adapter control group an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

If the Adapter (see page 6352) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. Otherwise, GetDesigntimeWarnings adds a reference to Adapter (see page 6352) to the design-time warnings object so that it can obtain any warning messages that the adapter wants to add.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

See Also

Adapter (see page 6352)

TAdapterPageProducer (see page 1136)

62.1.7.4.11 TAdapterControlGroup.GetFieldsList Method

Fills a string list with the names of all of the associated adapter's fields.

Pascal

```
procedure GetFieldsList(List: TStrings);
```

C++

```
__fastcall GetFieldsList(TStrings List);
```

Description

The CreateDefaultFields (see page 6346) method calls GetFieldsList to obtain a list of adapter fields for which it needs to create default child Web items. GetFieldsList obtains a list of the names of its fields from the adapter, and adds them to the specified string list. For any field that is a "default" field (as defined by the adapter), the string in the list is associated with the class of the child Web item that generates a control to represent that field.

List is a string list to fill with the names of (and Web item classes for) the adapter's default fields.

See Also

CreateDefaultFields (see page 6346)

Adapter (see page 6352)

62.1.7.4.12 TAdapterControlGroup.GetVisibleFields Method

Returns the value of the VisibleFields (see page 6354) property.

Pascal

```
function GetVisibleFields: TWebComponentList;
```

C++

```
__fastcall TWebComponentList GetVisibleFields();
```

Description

GetVisibleFields is the protected implementation of the VisibleFields (see page 6354) property.

See Also

VisibleFields (see page 6354)

62.1.7.4.13 TAdapterControlGroup.ImplGetControlRequirements Method

Provides the underlying implementation for the GetControlRequirements (see page 6348) method.

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

The GetControlRequirements (see page 6348) method calls ImplGetControlRequirements to add the variables and functions the control group needs to have declared to the control requirements object. This method adds the requirement of a variable with the name that the GetAdapterVariableName (see page 6348) method returns to represent the adapter whose fields the control group displays.

ARequirements is the control requirements object that collects a list of required variables and functions.

See Also

GetControlRequirements (see page 6348)

GetAdapterVariableName (see page 6348)

62.1.7.4.14 TAdapterControlGroup.IsFieldInUse Method

Indicates whether one of the adapter control group's child Web items represents a specified adapter field.

Pascal

```
function IsFieldInUse(const AName: string): Boolean;
```

C++

```
__fastcall Boolean IsFieldInUse(const UnicodeString AName);
```

Description

IsFieldInUse returns true if one of its child Web items represents the adapter field with the specified name, false otherwise.

AName is the name of a field that the adapter control group's adapter contains.

See Also

FindField (see page 6346)

VisibleFields (see page 6354)

Adapter (see page 6352)

62.1.7.4.15 TAdapterControlGroup.Notification Method

Updates the TAdapterControlGroup (see page 6341) object when components are added to or removed from the application.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation.

In TAdapterControlGroup (see page 6341), Notification checks whether AComponent is the value of the Adapter (see page 6352) property. If so, and Operation is opRemove, Notification sets the Adapter (see page 6352) property to nil (Delphi) or NULL (C++).

See Also

Adapter (see page 6352)

62.1.7.4.16 TAdapterControlGroup.NotifyAdapterChange Method

Responds when the control group's adapter notifies it of a change.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

NotifyAdapterChange is called automatically when the application makes changes to the adapter that is the value of the Adapter (see page 6352) property. This method calls the AdapterChange (see page 6345) method, which updates the list of default fields to be represented, and forwards the notification on to other Web items that may need to adjust themselves.

Sender is the component that sent the notification.

See Also

Adapter (see page 6352)

AdapterChange (see page 6345)

62.1.7.4.17 TAdapterControlGroup.RemoveNotify Method

Unregisters interest in changes to the control group's adapter.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Other Web items in the adapter form call RemoveNotify to unregister an interest in when the control group's adapter changes. The object must have previously registered interest by a call to the AddNotify (see page 6345) method. RemoveNotify removes the specified object from the internal list of interested objects.

ANotify is the object that had previously registered an interest in change notifications.

See Also

AdapterChange (see page 6345)

AddNotify (see page 6345)

62.1.7.4.18 TAdapterControlGroup.SetAdapter Method

Sets the value of the Adapter (see page 6352) property.

Pascal

```
procedure SetAdapter(const Value: TComponent);
```

C++

```
__fastcall SetAdapter(const TComponent * Value);
```

Description

SetAdapter is the protected write implementaiton of the Adapter (see page 6352) property.

See Also

Adapter (see page 6352)

62.1.7.5 TAdapterControlGroup Properties

The properties of the TAdapterControlGroup class are listed here.

Public Properties

	Name	Description
	Adapter (see page 6352)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6353)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6353)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
R	VisibleFields (see page 6354)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

62.1.7.5.1 TAdapterControlGroup.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 5549)

GetFieldsList (see page 6349)

AdapterMode (see page 6353)

WebFieldControls

62.1.7.5.2 TAdapterControlGroup.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert (see page 3601), Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter (see page 6352) property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode

Adapter (see page 6352)

WebFieldControls

62.1.7.5.3 TAdapterControlGroup.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter (see page 6352) page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter (see page 6352) property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 6354)

WebFieldControls

Adapter (see page 6352)

62.1.7.5.4 TAdapterControlGroup.VisibleFields Property

Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

Pascal

```
property VisibleFields: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleFields;
```

Description

Read (see page 3634) VisibleFields to determine the adapter display field components that generate HTML controls for this adapter control group. If you have explicitly added child Web items to the control group, or if the AddDefaultFields (see page 6353) property is false, VisibleFields is the same as the value of the WebFieldControls property. If you have not added any child Web items and AddDefaultFields (see page 6353) is true, VisibleFields is a list of adapter display field components that generate controls for the default fields of the control group's adapter.

See Also

WebFieldControls

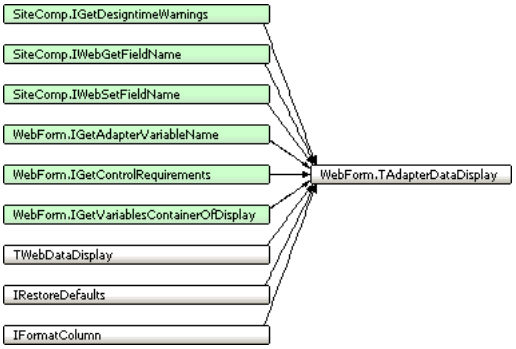
AddDefaultFields (see page 6353)

Adapter (see page 6352)

62.1.8 TAdapterDataDisplay Class

TAdapterDataDisplay is the base class for components that display adapter fields as form fields.

Class Hierarchy



Pascal

```
TAdapterDataDisplay = class(TWebDataDisplay, IWebGetFieldName, IWebSetFieldName,
IGetAdapterVariableName, IRestoreDefaults, IGetControlRequirements,
IGetVariablesContainerOfDisplay, IGetDesigntimeWarnings, IFormatColumn);
```

C++

```
class TAdapterDataDisplay : public TWebDataDisplay, public IWebGetFieldName, public
IWebSetFieldName, public IGetAdapterVariableName, public IRestoreDefaults, public
IGetControlRequirements, public IGetVariablesContainerOfDisplay, public
IGetDesigntimeWarnings, public IFormatColumn;
```

File

WebForm

Description

TAdapterDataDisplay is the base class for components that display adapter fields as HTML form fields. Its descendants include the following classes:

TAdapterDisplayField (see page 6371), which supports free-form display and editing of adapter field data.

TAdapterDisplayColumn (see page 6364), which supports grid display of adapter field data.

TAdapterEditColumn (see page 6387), which supports grid editing of adapter field data.

See Also

TAdapterDisplayField (see page 6371)

TAdapterDisplayColumn (see page 6364)

TAdapterEditColumn (see page 6387)

62.1.8.1 TAdapterDataDisplay Members

The following tables list the members exposed by TAdapterDataDisplay.

Protected Methods

	Name	Description
	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 6360)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6361)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6361)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 6361)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
	SetCaption (see page 6362)	Sets the value of the Caption property.
	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
	Create (see page 6356)	Creates an instance of TAdapterDataDisplay (see page 6354)

Public Properties

	Name	Description
	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6362)	Specifies the horizontal alignment of the component relative to its containing cell.
	FieldName (see page 6363)	Specifies the name of the associated dataset field.
	HideOptions (see page 6363)	When field should be hidden.
	VAlign (see page 6363)	Specifies the vertical alignment of the generated display control.

62.1.8.2 TAdapterDataDisplay.Create Constructor

Creates an instance of TAdapterDataDisplay (see page 6354)

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterDataDisplay(TComponent * AOwner);
```

Description

TAdapterDataDisplay (see page 6354) overrides Create in order to initialize the HideOptions (see page 6363) property to [hoHideOnFieldNotVisible].

62.1.8.3 TAdapterDataDisplay Methods

The methods of the TAdapterDataDisplay class are listed here.

Protected Methods

	Name	Description
	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 6360)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6361)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6361)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 6361)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
	SetCaption (see page 6362)	Sets the value of the Caption property.

	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

62.1.8.3.1 TAdapterDataDisplay.FieldConditions Method

Pascal

```
function FieldConditions(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString FieldConditions(TWebContentOptions Options);
```

Description

This is FieldConditions, a member of class TAdapterDataDisplay.

62.1.8.3.2 TAdapterDataDisplay.FindAssociatedField Method

Pascal

```
function FindAssociatedField: TComponent;
```

C++

```
__fastcall TComponent * FindAssociatedField();
```

Description

This is FindAssociatedField, a member of class TAdapterDataDisplay.

62.1.8.3.3 TAdapterDataDisplay.FormatCaption Method

Returns the HTML for the display control's caption.

Pascal

```
function FormatCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString FormatCaption();
```

Description

FormatCaption generates the HTML for the display control's caption, using the attributes specified by the CaptionAttributes property. FormatCaption returns the resulting string of HTML.

In addition, FormatCaption encodes the layout properties of the caption and the value of the CaptionPosition property into the LayoutAttributes property. The TWebDataDisplay component uses this value later to lay out the control and its caption appropriately.

See Also

Caption

CaptionAttributes

CaptionPosition

LayoutAttributes

62.1.8.3.4 TAdapterDataDisplay.FormatColumnData Method

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnData(const UnicodeString Content, TWebContentOptions
Options);
```

Description

This is FormatColumnData, a member of class TAdapterDataDisplay.

62.1.8.3.5 TAdapterDataDisplay.FormatColumnHeading Method

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnHeading(TWebContentOptions Options);
```

Description

This is FormatColumnHeading, a member of class TAdapterDataDisplay.

62.1.8.3.6 TAdapterDataDisplay.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.

62.1.8.3.7 TAdapterDataDisplay.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class TAdapterDataDisplay.

62.1.8.3.8 TAdapterDataDisplay.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption

62.1.8.3.9 TAdapterDataDisplay.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class TAdapterDataDisplay.

62.1.8.3.10 TAdapterDataDisplay.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the dataset values list needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the dataset values list an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DataSet property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. If DataSet is set, but the dataset is not active, GetDesigntimeWarnings adds a message that indicates this problem. Finally, if the ValueField property is not set, GetDesigntimeWarnings adds a message indicating this omission.

See Also

DataSet

Active

ValueField

62.1.8.3.11 TAdapterDataDisplay.GetFieldName Method

Pascal

```
function GetFieldName: string;
```

C++

```
__fastcall UnicodeString GetFieldName();
```

Description

This is GetFieldName, a member of class TAdapterDataDisplay.

62.1.8.3.12 TAdapterDataDisplay.GetVariablesContainer Method

Pascal

```
function GetVariablesContainer: TComponent;
```

C++

```
__fastcall TComponent * GetVariablesContainer();
```

Description

This is GetVariablesContainer, a member of class TAdapterDataDisplay.

62.1.8.3.13 TAdapterDataDisplay.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content

Caption

ILayoutWebContent (see page 6084)

62.1.8.3.14 TAdapterDataDisplay.ImplFormatColumnData Method

Pascal

```
function ImplFormatColumnData(const Content: string; Options: TWebContentOptions): string;
virtual;
```

C++

```
virtual __fastcall UnicodeString ImplFormatColumnData(const UnicodeString Content,
TWebContentOptions Options);
```

Description

This is ImplFormatColumnData, a member of class TAdapterDataDisplay.

62.1.8.3.15 TAdapterDataDisplay.ImplFormatColumnHeading Method

Pascal

```
function ImplFormatColumnHeading(Options: TWebContentOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplFormatColumnHeading(TWebContentOptions Options);
```

Description

This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.

62.1.8.3.16 TAdapterDataDisplay.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.

62.1.8.3.17 TAdapterDataDisplay.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.

62.1.8.3.18 TAdapterDataDisplay.RestoreDefaults Method

Returns the Web item to a default state.

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

The Web page editor calls RestoreDefaults when the user chooses the Restore Defaults command. Web items implement this

method to return property values to a default state.

62.1.8.3.19 TAdapterDataDisplay.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(Value: string); override;
```

C++

```
virtual __fastcall SetCaption(UnicodeString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption

62.1.8.3.20 TAdapterDataDisplay.SetFieldName Method

Pascal

```
procedure SetFieldName(const Value: string);
```

C++

```
__fastcall SetFieldName(const UnicodeString Value);
```

Description

This is SetFieldName, a member of class TAdapterDataDisplay.

62.1.8.4 TAdapterDataDisplay Properties

The properties of the TAdapterDataDisplay class are listed here.

Public Properties

	Name	Description
	Align (see page 6362)	Specifies the horizontal alignment of the component relative to its containing cell.
	FieldName (see page 6363)	Specifies the name of the associated dataset field.
	HideOptions (see page 6363)	When field should be hidden.
	VAlign (see page 6363)	Specifies the vertical alignment of the generated display control.

62.1.8.4.1 TAdapterDataDisplay.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6363)

62.1.8.4.2 TAdapterDataDisplay.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

62.1.8.4.3 TAdapterDataDisplay.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

62.1.8.4.4 TAdapterDataDisplay.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

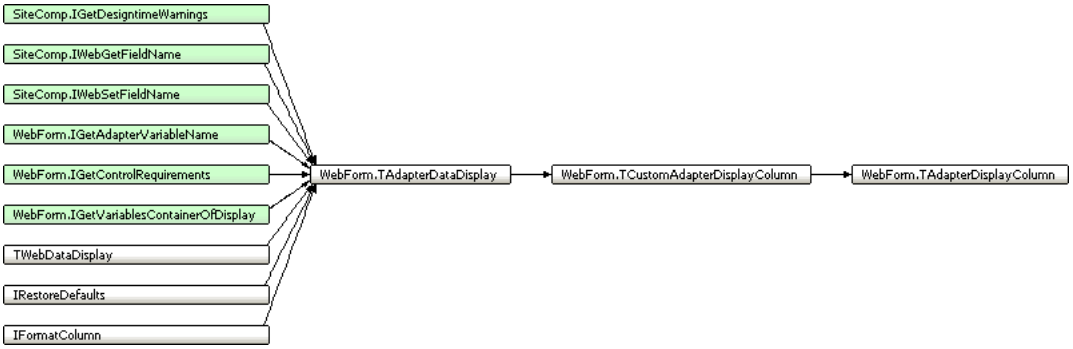
See Also

Align (see page 6362)

62.1.9 TAdapterDisplayColumn Class

TAdapterDisplayColumn displays an Adapter field in a grid.

Class Hierarchy



Pascal

```
TAdapterDisplayColumn = class(TCustomAdapterDisplayColumn);
```

C++

```
class TAdapterDisplayColumn : public TCustomAdapterDisplayColumn;
```

File

WebForm

Description

Add TAdapterDisplayColumn as a child of a TAdapterGrid (see page 6419) to display an Adapter field. The TAdapterGrid (see page 6419) specifies the Adapter used by TAdapterDisplayColumn.

Use the properties and methods of TAdapterDisplayColumn to:

Specify the Adapter field whose value the component displays.

Specify a caption for the column.

Specify when the column should be hidden.

Customize the appearance of the column.

TAdapterDisplayColumn supports multiple display presentation: text, list, and image. The display type can be set explicitly or TAdapterDisplayColumn will automatically choose a display type that is appropriate for the Adapter field.

TAdapterDisplayColumn is a minimal implementation of TCustomAdapterDisplayColumn (see page 6461). To extend this component, create your own concrete descendant of TCustomAdapterDisplayColumn (see page 6461).

See Also

TCustomAdapterDisplayColumn (see page 6461)

62.1.9.1 TAdapterDisplayColumn Members

The following tables list the members exposed by TAdapterDisplayColumn.

Protected Methods

	Name	Description
	CanAddClass (see page 6463)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
V	ControlContent (see page 6463)	Returns the HTML that defines the display control (without its caption).
V	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	FormatImage (see page 6464)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 6464)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 6464)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
V	ImplCanAddClass (see page 6465)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
V	ImplContent (see page 6360)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6361)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6465)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
V	ImplGetControlRequirements (see page 6361)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6465)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
V	ImplRestoreDefaults (see page 6361)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
V	ImplRestoreDefaults (see page 6465)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
V	SetCaption (see page 6362)	Sets the value of the Caption property.
	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
V	Create (see page 6356)	Creates an instance of TAdapterDataDisplay (see page 6354)
V	Create (see page 6463)	Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 6461).
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
V	IsColumn (see page 6466)	Indicates whether the component generates the controls that appear in the column of a data grid.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	Align (see page 6366)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6367)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6367)	Customizes the appearance of the generated control's label.
	Custom (see page 6367)	Represents a tag option and its value in the tag for the display control.
	DisplayType (see page 6368)	Sets the display type for the column.
	FieldName (see page 6368)	Specifies the name of the associated dataset field.
	HideOptions (see page 6368)	When field should be hidden.
	ImageHeight (see page 6369)	Specifies the height of the image in the column.
	ImageWidth (see page 6369)	Specifies the width of the image in the column.
	Style (see page 6369)	Adds style attributes to the generated display control.
	StyleRule (see page 6370)	Applies a style in the page producer's style sheet to the generated display control.
	VAlign (see page 6370)	Specifies the vertical alignment of the generated display control.

62.1.9.2 TAdapterDisplayColumn Properties

The properties of the TAdapterDisplayColumn class are listed here.

Published Properties

	Name	Description
	Align (see page 6366)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6367)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6367)	Customizes the appearance of the generated control's label.
	Custom (see page 6367)	Represents a tag option and its value in the tag for the display control.
	DisplayType (see page 6368)	Sets the display type for the column.
	FieldName (see page 6368)	Specifies the name of the associated dataset field.
	HideOptions (see page 6368)	When field should be hidden.
	ImageHeight (see page 6369)	Specifies the height of the image in the column.
	ImageWidth (see page 6369)	Specifies the width of the image in the column.
	Style (see page 6369)	Adds style attributes to the generated display control.
	StyleRule (see page 6370)	Applies a style in the page producer's style sheet to the generated display control.
	VAlign (see page 6370)	Specifies the vertical alignment of the generated display control.

62.1.9.2.1 TAdapterDisplayColumn.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6363)

62.1.9.2.2 TAdapterDisplayColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes

CaptionPosition

62.1.9.2.3 TAdapterDisplayColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes;
```

C++

```
__property CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption

CaptionPosition

HeadingAttributes

62.1.9.2.4 TAdapterDisplayColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator

key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style

StyleRule

CaptionAttributes

62.1.9.2.5 TAdapterDisplayColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TColumnHTMLElementType;
```

C++

```
__property TColumnHTMLElementType DisplayType;
```

Description

Set DisplayType to set the display type for the column to text or image.

See Also

ImageHeight (see page 6466)

ImageWidth (see page 6467)

62.1.9.2.6 TAdapterDisplayColumn.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

62.1.9.2.7 TAdapterDisplayColumn.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

62.1.9.2.8 TAdapterDisplayColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in pixels.

See Also

ImageWidth (see page 6467)

62.1.9.2.9 TAdapterDisplayColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in pixels.

See Also

ImageHeight (see page 6466)

62.1.9.2.10 TAdapterDisplayColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom

StyleRule

CaptionAttributes

Setting Web Item Properties

62.1.9.2.11 TAdapterDisplayColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated display control.

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated display control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style property instead.

Note: To apply a style to the control's label, use the CaptionAttributes property instead.

See Also

Style

Styles

Styles (see page 1157)

StylesFile

StylesFile (see page 1158)

CaptionAttributes

62.1.9.2.12 TAdapterDisplayColumn.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

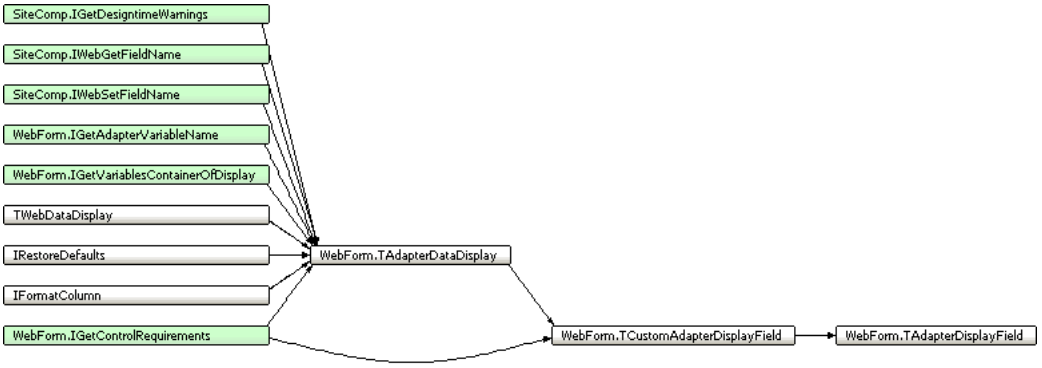
See Also

Align (see page 6362)

62.1.10 TAdapterDisplayField Class

TAdapterDisplayField displays an Adapter field as a form field.

Class Hierarchy



Pascal

```
TAdapterDisplayField = class(TCustomAdapterDisplayField);
```

C++

```
class TAdapterDisplayField : public TCustomAdapterDisplayField;
```

File

WebForm

Description

Add TAdapterDisplayField as a child of a TAdapterFieldGroup (see page 6399) to display an Adapter field. The TAdapterFieldGroup (see page 6399) specifies the Adapter used by TAdapterDisplayField.

Use the properties and methods of TAdapterDisplayField to:

Specify the Adapter field whose value the component displays.

Specify a caption for the field.

Specify when the field should be hidden.

Customize the appearance of the column.

Specify whether the field is used for input or display only.

TAdapterDisplayField supports multiple display type: text, list, and image. The display type can be set explicitly or TAdapterDisplayField will automatically choose a display type that is appropriate for the Adapter field.

TAdapterDisplayField supports multiple input types: Text, Password, Select, MultipleSelect, Radio, CheckBox, TextArea, and File. The input type can be set explicitly or TAdapterDisplayField will automatically choose an input type that is appropriate for the Adapter field.

TAdapterDisplayField instances are not used in grids. TAdapterEditColumn (see page 6387) provides equivalent functionality in a grid context.

TAdapterDisplayField is a minimal implementation of TCustomAdapterDisplayField (see page 6467). To extend the features of this component, define your own descendant of TCustomAdapterDisplayField (see page 6467).

See Also

TAdapterEditColumn (see page 6387)

62.1.10.1 TAdapterDisplayField Members

The following tables list the members exposed by TAdapterDisplayField.

Protected Methods

	Name	Description
	AddAttributes (see page 6470)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
	ControlContent (see page 6470)	Returns the HTML that defines the display control (without its caption).
	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetLayoutAttributes (see page 6470)	Returns the value of the LayoutAttributes property.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 6360)	Provides the underlying implementation for the Content method.
	ImplContent (see page 6471)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6361)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6361)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6471)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
	ImplRestoreDefaults (see page 6361)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 6471)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
	SetCaption (see page 6362)	Sets the value of the Caption property.
	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
	Create (see page 6356)	Creates an instance of TAdapterDataDisplay (see page 6354)
	Create (see page 6469)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6467).
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	Align (see page 6374)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6374)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6374)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6375)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6375)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6376)	Specifies the number of columns in a display field.
	DisplayRows (see page 6376)	Specifies the number of rows in a display field.
	DisplayType (see page 6376)	Sets the display type for the column.
	DisplayWidth (see page 6377)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6377)	Specifies the name of the associated dataset field.
	HideOptions (see page 6377)	When field should be hidden.
	ImageHeight (see page 6377)	Specifies the height of the image in the column.
	ImageWidth (see page 6378)	Specifies the width of the image in the column.
	InputType (see page 6378)	Specifies the input type for the display field.
	MaxLength (see page 6378)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6379)	Indicates the number of rows that can be selected.
	Style (see page 6379)	Adds style attributes to the generated display control.
	StyleRule (see page 6379)	Applies a style in the page producer's style sheet to the generated display control.
	TextAreaWrap (see page 6380)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6380)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6381)	Indicates the viewing state of the adapter field.

62.1.10.2 TAdapterDisplayField Properties

The properties of the TAdapterDisplayField class are listed here.

Published Properties

	Name	Description
	Align (see page 6374)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6374)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6374)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6375)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6375)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6376)	Specifies the number of columns in a display field.
	DisplayRows (see page 6376)	Specifies the number of rows in a display field.
	DisplayType (see page 6376)	Sets the display type for the column.
	DisplayWidth (see page 6377)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6377)	Specifies the name of the associated dataset field.
	HideOptions (see page 6377)	When field should be hidden.
	ImageHeight (see page 6377)	Specifies the height of the image in the column.
	ImageWidth (see page 6378)	Specifies the width of the image in the column.
	InputType (see page 6378)	Specifies the input type for the display field.
	MaxLength (see page 6378)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6379)	Indicates the number of rows that can be selected.

	Style (see page 6379)	Adds style attributes to the generated display control.
	StyleRule (see page 6379)	Applies a style in the page producer's style sheet to the generated display control.
	TextAreaWrap (see page 6380)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6380)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6381)	Indicates the viewing state of the adapter field.

62.1.10.2.1 TAdapterDisplayField.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6363)

62.1.10.2.2 TAdapterDisplayField.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes

CaptionPosition

62.1.10.2.3 TAdapterDisplayField.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes;
```

C++

```
__property CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption

CaptionPosition

HeadingAttributes

62.1.10.2.4 TAdapterDisplayField.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition;
```

C++

```
__property CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes

Caption

62.1.10.2.5 TAdapterDisplayField.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style

StyleRule

CaptionAttributes

62.1.10.2.6 TAdapterDisplayField.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 6472)

62.1.10.2.7 TAdapterDisplayField.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns to control the number of rows being displayed.

See Also

DisplayColumns (see page 6472)

62.1.10.2.8 TAdapterDisplayField.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 6473)

ImageWidth (see page 6474)

InputType (see page 6474)

62.1.10.2.9 TAdapterDisplayField.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField control values are one character (M for modified or I for inserted).

62.1.10.2.10 TAdapterDisplayField.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

62.1.10.2.11 TAdapterDisplayField.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

62.1.10.2.12 TAdapterDisplayField.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 6473)

ImageWidth (see page 6474)

62.1.10.2.13 TAdapterDisplayField.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in the column.

See Also

ImageHeight (see page 6473)

62.1.10.2.14 TAdapterDisplayField.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 6473)

ImageWidth (see page 6474)

DisplayType (see page 6473)

62.1.10.2.15 TAdapterDisplayField.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

62.1.10.2.16 TAdapterDisplayField.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

62.1.10.2.17 TAdapterDisplayField.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom

StyleRule

CaptionAttributes

Setting Web Item Properties

62.1.10.2.18 TAdapterDisplayField.StyleRule Property

Applies a style in the page producer's style sheet to the generated display control.

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated display control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style property instead.

Note: To apply a style to the control's label, use the CaptionAttributes property instead.

See Also

Style

Styles

Styles (see page 1157)

StylesFile

StylesFile (see page 1158)

CaptionAttributes

62.1.10.2.19 TAdapterDisplayField.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth.

See Also

DisplayWidth (see page 6473)

62.1.10.2.20 TAdapterDisplayField.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 6362)

62.1.10.2.21 TAdapterDisplayField.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

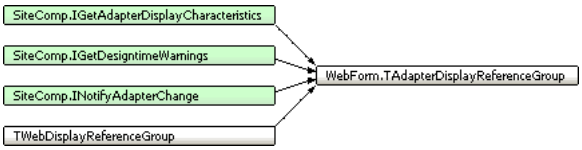
Description

Use ViewMode to specify the viewing state of the field.

62.1.11 TAdapterDisplayReferenceGroup Class

TAdapterDisplayReferenceGroup is the base class for components that generate HTML controls that refer to a control which displays data from an adapter.

Class Hierarchy



Pascal

```
TAdapterDisplayReferenceGroup = class(TWebDisplayReferenceGroup, IGetDesigntimeWarnings,
INotifyAdapterChange, IGetAdapterDisplayCharacteristics);
```

C++

```
class TAdapterDisplayReferenceGroup : public TWebDisplayReferenceGroup, public
IGetDesigntimeWarnings, public INotifyAdapterChange, public IGetAdapterDisplayCharacteristics;
```

File

WebForm

Description

Use TAdapterDisplayReferenceGroup as a base class for Web items that generate the HTML to arrange a collection of controls. The TAdapterDisplayReferenceGroup is associated with another component that generates HTML elements to display the data from an adapter. This component is the value of the DisplayComponent (see page 6386) property. The child Web items of TAdapterDisplayReferenceGroup generate the individual controls in the collection this class arranges. Each of those controls operates on the display element produced by the component that is the value of the DisplayComponent (see page 6386) property.

Do not create instances of TAdapterDisplayReferenceGroup. TAdapterDisplayReferenceGroup only manages the Web items that generate controls, the association to a display generator, and the logic for working with the WebSnap editors in the IDE. It does not include any special logic for working with the display generator's adapter. To add a component that organizes controls in an adapter form, use a descendant of TAdapterDisplayReferenceGroup such as TAdapterCommandGroup (see page 6337).

See Also

TAdapterCommandGroup (see page 6337)

62.1.11.1 TAdapterDisplayReferenceGroup Members

The following tables list the members exposed by TAdapterDisplayReferenceGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6383)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6384)	Responds when the DisplayComponent (see page 6386) property changes.
	GetAdapterDisplayCharacteristics (see page 6384)	Indicates the display characteristics of the HTML this component generates.
	GetDesigntimeWarnings (see page 6384)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	NotifyAdapterChange (see page 6385)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6385)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 6386)	Sets the value of the DisplayComponent (see page 6386) property.

Public Methods

	Name	Description
	Create (see page 6382)	Creates and initializes an instance of TAdapterDisplayReferenceGroup (see page 6381).
	Destroy (see page 6382)	Frees an instance of TAdapterDisplayReferenceGroup (see page 6381).

Public Properties

	Name	Description
	DisplayComponent (see page 6386)	Associates the group with a component that generates the display controls.

62.1.11.2 TAdapterDisplayReferenceGroup.Create Constructor

Creates and initializes an instance of TAdapterDisplayReferenceGroup (see page 6381).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TAdapterDisplayReferenceGroup(TComponent * AOwner);
```

Description

Call Create to instantiate an adapter display reference group at runtime. TAdapterDisplayReferenceGroup (see page 6381) descendants added to an adapter form at design time are instantiated automatically.

AOwner is a component, typically a Web page module, that becomes the value of the Owner property. The Owner is responsible for freeing the TAdapterDisplayReferenceGroup (see page 6381) instance.

See Also

Create (see page 6456)

Owner

62.1.11.3 TAdapterDisplayReferenceGroup.Destroy Destructor

Frees an instance of TAdapterDisplayReferenceGroup (see page 6381).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAdapterDisplayReferenceGroup();
```

Description

Do not call Destroy directly. Instead, call the Free method, which checks whether the reference to the producer is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

62.1.11.4 TAdapterDisplayReferenceGroup Methods

The methods of the TAdapterDisplayReferenceGroup class are listed here.

Protected Methods

	Name	Description
	AdapterChange (see page 6383)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6384)	Responds when the DisplayComponent (see page 6386) property changes.
	GetAdapterDisplayCharacteristics (see page 6384)	Indicates the display characteristics of the HTML this component generates.
	GetDesigntimeWarnings (see page 6384)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	NotifyAdapterChange (see page 6385)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6385)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 6386)	Sets the value of the DisplayComponent (see page 6386) property.

62.1.11.4.1 TAdapterDisplayReferenceGroup.AdapterChange Method

Responds to changes in the adapter on which the controls in this group operate.

Pascal

```
procedure AdapterChange; virtual;
```

C++

```
virtual __fastcall AdapterChange();
```

Description

TAdapterDisplayReferenceGroup (see page 6381) calls AdapterChange internally whenever the adapter associated with its display component changes. This happens in response to the following occurrences:

The component specified by DisplayComponent (see page 6386) notifies this one that it has changed.

Another component notifies this one of a change to the adapter by calling the NotifyAdapterChange (see page 6385) method.

AdapterChange responds to the change by regenerating its list of default child Web items and passing the notification on to every child Web item in its WebFieldControls property list.

See Also

NotifyAdapterChange (see page 6385)

DisplayComponent (see page 6386)

DisplayComponentChange (see page 6384)

WebFieldControls

62.1.11.4.2 TAdapterDisplayReferenceGroup.DisplayComponentChange Method

Responds when the DisplayComponent (see page 6386) property changes.

Pascal

```
procedure DisplayComponentChange; virtual;
```

C++

```
virtual __fastcall DisplayComponentChange();
```

Description

The component specified by the DisplayComponent (see page 6386) property calls DisplayComponentChange when its properties change. This allows the TAdapterDisplayReferenceGroup (see page 6381) object to update itself to reflect any changes.

In TAdapterDisplayReferenceGroup (see page 6381), DisplayComponentChange calls the AdapterChange (see page 6383) method, to ensure that its Web items are up-to-date in case the change was to the associated adapter.

See Also

AdapterChange (see page 6383)

DisplayComponent (see page 6386)

62.1.11.4.3 TAdapterDisplayReferenceGroup.GetAdapterDisplayCharacteristics Method

Indicates the display characteristics of the HTML this component generates.

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

Adapter actions call GetAdapterDisplayCharacteristics from their IsDefaultAction methods to determine whether their function should be included in the adapter's default set of actions. If the adapter action is appropriate to operate on an HTML form with the indicated characteristics, the adapter action returns true from its IsDefaultAction method.

In TAdapterDisplayReferenceGroup (see page 6381), GetAdapterDisplayCharacteristics delegates this task by fetching the display characteristics from the associated display component.

See Also

IsDefaultAction (see page 5471)

DisplayComponent (see page 6386)

62.1.11.4.4 TAdapterDisplayReferenceGroup.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the reference group needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the display reference group an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DisplayComponent (see page 6386) property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. Otherwise, GetDesigntimeWarnings adds the display component to the list of objects that can supply design-time warnings.

See Also

DisplayComponent (see page 6386)

TAdapterPageProducer (see page 1136)

62.1.11.4.5 TAdapterDisplayReferenceGroup.NotifyAdapterChange Method

Responds when other components notify this one of changes to the adapter.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

Other components in the WebSnap application call NotifyAdapterChange when they detect possible changes to the adapter on which this group's controls operate.

Sender is the component that detected the possible change and sent the notification.

NotifyAdapterChange calls the AdapterChange (see page 6383) method, which responds to the notification.

See Also

AdapterChange (see page 6383)

DisplayComponent (see page 6386)

62.1.11.4.6 TAdapterDisplayReferenceGroup.OnDisplayComponentChange Method

Responds when the associated display component changes.

Pascal

```
procedure OnDisplayComponentChange(ASender: TObject);
```

C++

```
__fastcall OnDisplayComponentChange(TObject * ASender);
```

Description

The component specified by the DisplayComponent (see page 6386) property calls OnDisplayComponentChange to notify this component of changes to its properties or child Web items.

ASender is the display component that calls this method.

OnDisplayComponentChange calls the DisplayComponentChange (see page 6384) method, which responds to the change.

See Also

DisplayComponentChange (see page 6384)

DisplayComponent (see page 6386)

62.1.11.4.7 TAdapterDisplayReferenceGroup.SetWebDisplayComponent Method

Sets the value of the DisplayComponent (see page 6386) property.

Pascal

```
procedure SetWebDisplayComponent(const AComponent: TComponent); override;
```

C++

```
virtual __fastcall SetWebDisplayComponent(const TComponent * AComponent);
```

Description

SetWebDisplayComponent is the protected write implementation of the DisplayComponent (see page 6386) property.

AComponent is the new value of the DisplayComponent (see page 6386) property.

See Also

DisplayComponent (see page 6386)

62.1.11.5 TAdapterDisplayReferenceGroup Properties

The properties of the TAdapterDisplayReferenceGroup class are listed here.

Public Properties

	Name	Description
	DisplayComponent (see page 6386)	Associates the group with a component that generates the display controls.

62.1.11.5.1 TAdapterDisplayReferenceGroup.DisplayComponent Property

Associates the group with a component that generates the display controls.

Pascal

```
property DisplayComponent: TComponent;
```

C++

```
__property TComponent * DisplayComponent;
```

Description

Use DisplayComponent to associate the group of controls with a Web item that generates the display elements those controls influence. Typical values of DisplayComponent are TAdapterGrid (see page 6419) or TAdapterFieldGroup (see page 6399).

TAdapterDisplayReferenceGroup (see page 6381) descendants use the value of DisplayComponent to access the display controls to influence, and to access the adapter, whose information is displayed or manipulated.

See Also

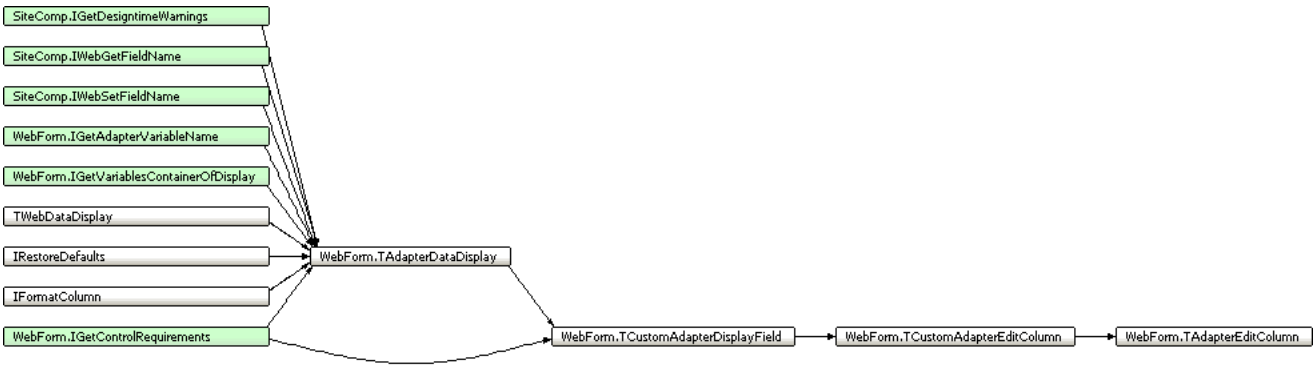
TAdapterGrid (see page 6419)

TAdapterFieldGroup (see page 6399)

62.1.12 TAdapterEditColumn Class

TAdapterEditColumn displays an adapter field as a form field in a grid.

Class Hierarchy



Pascal

```
TAdapterEditColumn = class(TCustomAdapterEditColumn);
```

C++

```
class TAdapterEditColumn : public TCustomAdapterEditColumn;
```

File

WebForm

Description

TAdapterEditColumn has the same functionality as TAdapterDisplayField (see page 6371), but is designed to work with grids. TAdapterEditColumn is a minimal implementation of TCustomAdapterEditColumn (see page 6475).

See Also

TCustomAdapterEditColumn (see page 6475)

TAdapterDisplayField (see page 6371)

62.1.12.1 TAdapterEditColumn Members

The following tables list the members exposed by TAdapterEditColumn.

Protected Methods

	Name	Description
	AddAttributes (see page 6470)	This is AddAttributes, a member of class TCustomAdapterDisplayField.

	ControlContent (see page 6470)	Returns the HTML that defines the display control (without its caption).
	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetLayoutAttributes (see page 6470)	Returns the value of the LayoutAttributes property.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 6360)	Provides the underlying implementation for the Content method.
	ImplContent (see page 6471)	Provides the underlying implementation for the Content method.
	ImplContent (see page 6478)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6361)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6361)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6471)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
	ImplRestoreDefaults (see page 6361)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 6471)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
	SetCaption (see page 6362)	Sets the value of the Caption property.
	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
	Create (see page 6356)	Creates an instance of TAdapterDataDisplay (see page 6354)
	Create (see page 6469)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6467).
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

Published Properties

	Name	Description
	Align (see page 6389)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6390)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6390)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6390)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6391)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6391)	Specifies the number of columns in a display field.

	DisplayRows (see page 6392)	Specifies the number of rows in a display field.
	DisplayType (see page 6392)	Sets the display type for the column.
	DisplayWidth (see page 6392)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6393)	Specifies the name of the associated dataset field.
	HideOptions (see page 6393)	When field should be hidden.
	ImageHeight (see page 6393)	Specifies the height of the image in the column.
	ImageWidth (see page 6393)	Specifies the width of the image in the column.
	InputType (see page 6394)	Specifies the input type for the display field.
	MaxLength (see page 6394)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6394)	Indicates the number of rows that can be selected.
	Style (see page 6395)	Adds style attributes to the generated display control.
	StyleRule (see page 6395)	Applies a style in the page producer's style sheet to the generated display control.
	TextAreaWrap (see page 6396)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6396)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6396)	Indicates the viewing state of the adapter field.

62.1.12.2 TAdapterEditColumn Properties

The properties of the TAdapterEditColumn class are listed here.

Published Properties

	Name	Description
	Align (see page 6389)	Specifies the horizontal alignment of the component relative to its containing cell.
	Caption (see page 6390)	Specifies the text of a label that indicates what information the generated control displays.
	CaptionAttributes (see page 6390)	Customizes the appearance of the generated control's label.
	CaptionPosition (see page 6390)	Specifies where the label appears relative to the generated display control.
	Custom (see page 6391)	Represents a tag option and its value in the tag for the display control.
	DisplayColumns (see page 6391)	Specifies the number of columns in a display field.
	DisplayRows (see page 6392)	Specifies the number of rows in a display field.
	DisplayType (see page 6392)	Sets the display type for the column.
	DisplayWidth (see page 6392)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 6393)	Specifies the name of the associated dataset field.
	HideOptions (see page 6393)	When field should be hidden.
	ImageHeight (see page 6393)	Specifies the height of the image in the column.
	ImageWidth (see page 6393)	Specifies the width of the image in the column.
	InputType (see page 6394)	Specifies the input type for the display field.
	MaxLength (see page 6394)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6394)	Indicates the number of rows that can be selected.
	Style (see page 6395)	Adds style attributes to the generated display control.
	StyleRule (see page 6395)	Applies a style in the page producer's style sheet to the generated display control.
	TextAreaWrap (see page 6396)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6396)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6396)	Indicates the viewing state of the adapter field.

62.1.12.2.1 TAdapterEditColumn.Align Property

Specifies the horizontal alignment of the component relative to its containing cell.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align specifies the component's horizontal alignment relative to the containing HTML table cell.

See Also

VAlign (see page 6363)

62.1.12.2.2 TAdapterEditColumn.Caption Property

Specifies the text of a label that indicates what information the generated control displays.

Pascal

```
property Caption;
```

C++

```
__property Caption;
```

Description

Use Caption to specify the text of the control's label.

See Also

CaptionAttributes

CaptionPosition

62.1.12.2.3 TAdapterEditColumn.CaptionAttributes Property

Customizes the appearance of the generated control's label.

Pascal

```
property CaptionAttributes;
```

C++

```
__property CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the control's label. CaptionAttributes defines a set of options that are added to an HTML SPAN tag surrounding the value of the Caption property.

See Also

Caption

CaptionPosition

HeadingAttributes

62.1.12.2.4 TAdapterEditColumn.CaptionPosition Property

Specifies where the label appears relative to the generated display control.

Pascal

```
property CaptionPosition;
```

C++

```
__property CaptionPosition;
```

Description

Use CaptionPosition to indicate where the label should appear relative to the control that displays information.

Note: CaptionPosition has no effect on column controls. The position of column labels is always in the table header.

See Also

CaptionAttributes

Caption

62.1.12.2.5 TAdapterEditColumn.Custom Property

Represents a tag option and its value in the tag for the display control.

Pascal

```
property Custom;
```

C++

```
__property Custom;
```

Description

TWebDataDisplay descendants generate different HTML tags to generate the control that displays information. Use Custom to add options that qualify the tag. Each option and its value has the form

OPTIONNAME=Value

The options specified by Custom should be separated by spaces. For example, to provide an INPUT control with an accelerator key and align text to the bottom, you can set Custom to

ACCESSKEY=A ALIGN=BOTTOM

Note: To add custom options to the control's label, use the CaptionAttributes property instead.

See Also

Style

StyleRule

CaptionAttributes

62.1.12.2.6 TAdapterEditColumn.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 6472)

62.1.12.2.7 TAdapterEditColumn.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns to control the number of rows being displayed.

See Also

DisplayColumns (see page 6472)

62.1.12.2.8 TAdapterEditColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 6473)

ImageWidth (see page 6474)

InputType (see page 6474)

62.1.12.2.9 TAdapterEditColumn.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField control values are one character (M for modified or I for inserted).

62.1.12.2.10 TAdapterEditColumn.FieldName Property

Specifies the name of the associated dataset field.

Pascal

```
property FieldName: string;
```

C++

```
__property UnicodeString FieldName;__property UnicodeString FieldName;
```

Description

FieldName specifies the name of the dataset field that the adapter field represents.

See Also

DataSet

62.1.12.2.11 TAdapterEditColumn.HideOptions Property

When field should be hidden.

Pascal

```
property HideOptions: TDisplayFieldHideOptions;
```

C++

```
__property TDisplayFieldHideOptions HideOptions;
```

Description

Set HideOptions to indicate when the field should be hidden from view.

62.1.12.2.12 TAdapterEditColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 6473)

ImageWidth (see page 6474)

62.1.12.2.13 TAdapterEditColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight to indicate the width of the image in the column.

See Also

ImageHeight (see page 6473)

62.1.12.2.14 TAdapterEditColumn.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 6473)

ImageWidth (see page 6474)

DisplayType (see page 6473)

62.1.12.2.15 TAdapterEditColumn.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

62.1.12.2.16 TAdapterEditColumn.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

62.1.12.2.17 TAdapterEditColumn.Style Property

Adds style attributes to the generated display control.

Pascal

```
property Style;
```

C++

```
__property Style;
```

Description

Use Style to specify display characteristics that affect the generated control. The Style option consists of one or more attribute descriptions such as

font-family: arial; font-weight: bold; font-size: 10px

These attributes are added to the HTML tag that generates the control as the value of a STYLE option.

Note: To add style attributes to the control's label, use the CaptionAttributes property instead.

See Also

Custom

StyleRule

CaptionAttributes

Setting Web Item Properties

62.1.12.2.18 TAdapterEditColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated display control.

Pascal

```
property StyleRule;
```

C++

```
__property StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated display control. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style property instead.

Note: To apply a style to the control's label, use the CaptionAttributes property instead.

See Also

Style

Styles

Styles (see page 1157)

StylesFile

StylesFile (see page 1158)

CaptionAttributes

62.1.12.2.19 TAdapterEditColumn.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth.

See Also

DisplayWidth (see page 6473)

62.1.12.2.20 TAdapterEditColumn.VAlign Property

Specifies the vertical alignment of the generated display control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign specifies the control's vertical alignment relative to the containing HTML table cell.

See Also

Align (see page 6362)

62.1.12.2.21 TAdapterEditColumn.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

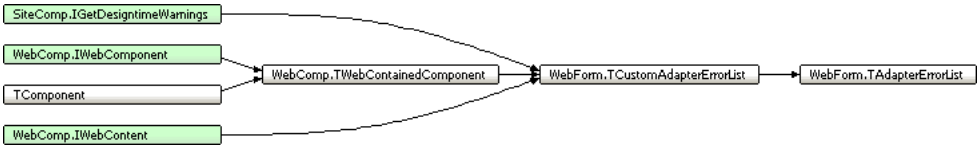
Description

Use ViewMode to specify the viewing state of the field.

62.1.13 TAdapterErrorList Class

TAdapterErrorList is the error list display component.

Class Hierarchy



Pascal

```
TAdapterErrorList = class(TCustomAdapterErrorList);
```

C++

```
class TAdapterErrorList : public TCustomAdapterErrorList;
```

File

WebForm

Description

Use TAdapterErrorList to display errors relating to a specific adapter component.

TAdapterErrorList is a minimal implementation of TCustomAdapterErrorList (see page 6478). To extend its features, create your own concrete descendant of TCustomAdapterErrorList (see page 6478).

Note: Do not confuse TAdapterErrorList with TAdapterErrorsList.

See Also

TAdapterErrorsList (see page 5259)

62.1.13.1 TAdapterErrorList Members

The following tables list the members exposed by TAdapterErrorList.

Protected Methods

	Name	Description
	Content (see page 6480)	Returns the HTML that the Web item generates.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDesigntimeWarnings (see page 6481)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	ImplContent (see page 6481)	This is ImplContent, a member of class TCustomAdapterErrorList.
	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
	Notification (see page 6482)	Forwards notification messages to all owned components.
	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
	Content (see page 6104)	Returns the HTML that the Web item generates.
	Create (see page 6039)	Creates and initializes an instance of TWebContainedComponent (see page 6038).
	Create (see page 6480)	Creates an instance of TCustomAdapterErrorList (see page 6478).
	Destroy (see page 6040)	Frees an instance of TWebContainedComponent (see page 6038).
	Destroy (see page 6480)	Frees the memory associated with an instance of TCustomAdapterErrorList (see page 6478).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.
	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.

Published Properties

	Name	Description
	Adapter (see page 6398)	Specifies a related adapter.

62.1.13.2 TAdapterErrorList Properties

The properties of the TAdapterErrorList class are listed here.

Published Properties

	Name	Description
	Adapter (see page 6398)	Specifies a related adapter.

62.1.13.2.1 TAdapterErrorList.Adapter Property

Specifies a related adapter.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

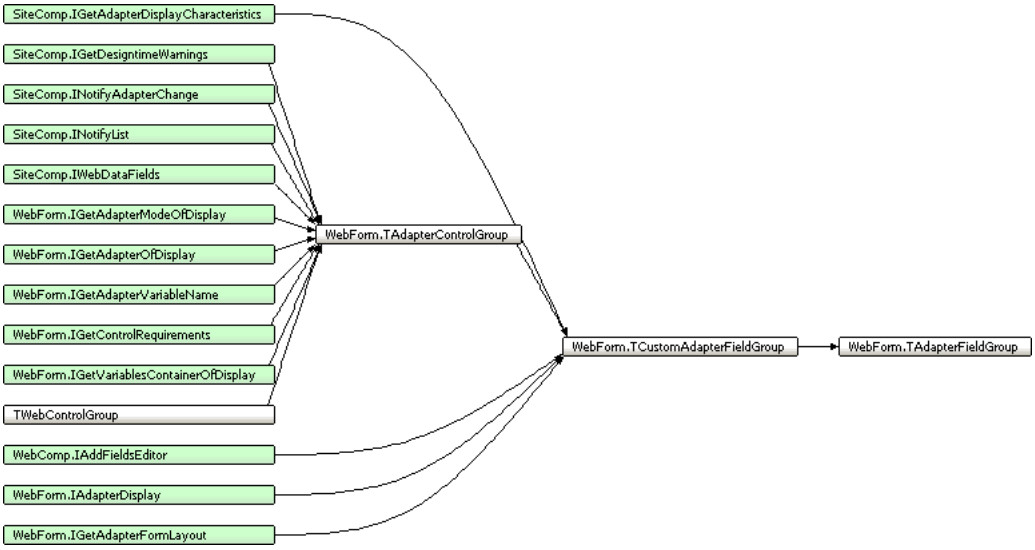
Description

Adapter specifies the adapter component for which this component reports errors.

62.1.14 TAdapterFieldGroup Class

TAdapterFieldGroup is a container for Adapter (see page 6402) display field components such as TAdapterDisplayField (see page 6371). TAdapterFieldGroup labels and fields in an HTML form.

Class Hierarchy



Pascal

```
TAdapterFieldGroup = class(TCustomAdapterFieldGroup);
```

C++

```
class TAdapterFieldGroup : public TCustomAdapterFieldGroup;
```

File

WebForm

Description

Add TAdapterFieldGroup as a child of TAdapterForm (see page 6404).

TAdapterFieldGroup is associated with an Adapter (see page 6402). All of its child display field components will use the same Adapter (see page 6402).

If no display fields have been explicitly added to the TAdapterFieldGroup, display fields associated with all of the Adapter (see page 6402) fields will be added by default. The properties of default display fields cannot be modified.

Modifying the AdapterMode (see page 6402) property of TAdapterFieldGroup may change the default display type or input type of Adapter (see page 6402) fields. This may cause the display field children of TAdapterFieldGroup to change appearance.

Use the properties and methods of TAdapterFieldGroup to:

Specify the associated Adapter (see page 6402).

Specify when the Adapter (see page 6402) Mode.

Specify whether default fields should be added.

See Also

TAdapterDisplayField (see page 6371)

TAdapterForm (see page 6404)

62.1.14.1 TAdapterFieldGroup Members

The following tables list the members exposed by TAdapterFieldGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6345)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6345)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6346)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6346)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6346)	Returns the value of the Adapter (see page 6352) property.
	GetAdapterDisplayCharacteristics (see page 6486)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 6486)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAdapterModeOfAdapter (see page 6347)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6347)	Returns the value of the AdapterMode (see page 6353) property.
	GetAdapterVariableName (see page 6348)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6486)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 6487)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 6487)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetControlRequirements (see page 6348)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6348)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6349)	Fills a string list with the names of all of the associated adapter's fields.
	GetLayoutAttributes (see page 6487)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 6487)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
	GetVisibleFields (see page 6349)	Returns the value of the VisibleFields (see page 6354) property.
	ImplCanAddClass (see page 6488)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6488)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6350)	Provides the underlying implementation for the GetControlRequirements (see page 6348) method.
	IsFieldInUse (see page 6350)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6350)	Updates the TAdapterControlGroup (see page 6341) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6351)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6351)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6352)	Sets the value of the Adapter (see page 6352) property.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	Create (see page 6343)	Creates and initializes an instance of TAdapterControlGroup (see page 6341).

	Create (see page 6485)	Creates and initializes an instance of TCustomAdapterFieldGroup (see page 6483).
	Destroy (see page 6344)	Frees an instance of TAdapterControlGroup (see page 6341).
	Destroy (see page 6485)	Frees an instance of TCustomAdapterFieldGroup (see page 6483).
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6513)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 1019)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 6511)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 6512)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6512)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.

Public Properties

	Name	Description
	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.
	VisibleFields (see page 6354)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

Published Properties

	Name	Description
	Adapter (see page 6402)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6402)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6403)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 6403)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6404)	Adds style attributes to the generated button group.
	StyleRule (see page 6404)	Applies a style in the page producer's style sheet to the button group.

62.1.14.2 TAdapterFieldGroup Properties

The properties of the TAdapterFieldGroup class are listed here.

Published Properties

	Name	Description
	Adapter (see page 6402)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6402)	Specifies the mode of the adapter that the controls in the group support.

	AddDefaultFields (see page 6403)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 6403)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6404)	Adds style attributes to the generated button group.
	StyleRule (see page 6404)	Applies a style in the page producer's style sheet to the button group.

62.1.14.2.1 TAdapterFieldGroup.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 5549)

GetFieldsList (see page 6349)

AdapterMode (see page 6353)

WebFieldControls

62.1.14.2.2 TAdapterFieldGroup.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert, Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode

Adapter (see page 6352)

WebFieldControls

62.1.14.2.3 TAdapterFieldGroup.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 6354)

WebFieldControls

Adapter (see page 6352)

62.1.14.2.4 TAdapterFieldGroup.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button group. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Style (see page 6489)

62.1.14.2.5 TAdapterFieldGroup.Style Property

Adds style attributes to the generated button group.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button group. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 6489)

StyleRule (see page 6489)

62.1.14.2.6 TAdapterFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the button group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button group. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style property.

See Also

Style (see page 6489)

Styles (see page 1157)

StylesFile (see page 1158)

62.1.15 TAdapterForm Class

TAdapterForm generates an HTML form displaying Adapter fields.

Class Hierarchy

Pascal

```
TAdapterForm = class(TCustomAdapterForm);
```

C++

```
class TAdapterForm : public TCustomAdapterForm;
```

File

WebForm

Description

TAdapter form is a basic implementation of TCustomAdapterForm (see page 6490). Add TAdapterForm to a TAdapterPageProducer to display Adapter fields users can view and/or edit, and Adapter actions that users can execute.

Add child components to TAdapterForm to display Adapter fields and Adapter actions. Child components include TAdapterGrid (see page 6419), TAdapterFieldGroup (see page 6399), and TAdapterCommandGroup (see page 6337). Add TAdapterErrors to TAdapterForm to display errors that are generated by an Adapter.

Use the properties and methods of TAdapterForm to:

Customize the appearance of the HTML form.

Indicate the HTTP method used to submit the HTML form.

See Also

TAdapterPageProducer (see page 1136)

TAdapterGrid (see page 6419)

TAdapterFieldGroup (see page 6399)

TAdapterCommandGroup (see page 6337)

TAdapterErrorList (see page 6397)

62.1.15.1 TAdapterForm Members

The following tables list the members exposed by TAdapterForm.

Protected Methods

	Name	Description
	GetAdapterFormLayout (see page 6492)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetFormRequirements (see page 6492)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 6492)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 6493)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 6493)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6493)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6494)	Provides the underlying implementation for the Content method.

Public Methods

	Name	Description
	Create (see page 6491)	Creates and initializes an instance of TCustomAdapterForm (see page 6490).
	Destroy (see page 6491)	Frees an instance of TCustomAdapterForm (see page 6490).
	GetAdapterFormLayout (see page 6511)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetFormRequirements (see page 6517)	This is GetFormRequirements, a member of class IGetFormRequirements.

Published Properties

	Name	Description
	Custom (see page 6406)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6406)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6407)	Adds style attributes to the generated form.
	StyleRule (see page 6407)	Applies a style in the page producer's style sheet to the form.

62.1.15.2 TAdapterForm Properties

The properties of the TAdapterForm class are listed here.

Published Properties

	Name	Description
	Custom (see page 6406)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6406)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6407)	Adds style attributes to the generated form.
	StyleRule (see page 6407)	Applies a style in the page producer's style sheet to the form.

62.1.15.2.1 TAdapterForm.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the <table> tag that the adapter form generates to manage the layout of its Web items. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

ALIGN=TOP TABINDEX=2

See Also

Style (see page 6495)

62.1.15.2.2 TAdapterForm.FormMethod Property

Specifies the method that the adapter form adds to its generated <form> tag.

Pascal

```
property FormMethod: TFormMethod;
```

C++

```
__property TFormMethod FormMethod;
```

Description

Set FormMethod to specify the METHOD header for HTTP messages that the form submits. This value gives rise to the MethodType property of the HTTP request message.

See Also

MethodType

62.1.15.2.3 TAdapterForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the <table> tag that the adapter form generates to manage the layout of its Web items. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated <table> tag as the value of a STYLE option.

See Also

Custom (see page 6495)

StyleRule (see page 6496)

62.1.15.2.4 TAdapterForm.StyleRule Property

Applies a style in the page producer's style sheet to the form.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the adapter page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the form's Style property.

See Also

Style (see page 6495)

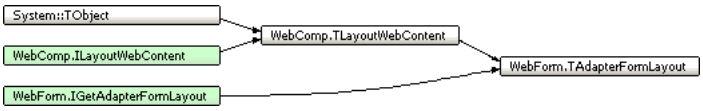
Styles (see page 1157)

StylesFile (see page 1158)

62.1.16 TAdapterFormLayout Class

TAdapterFormLayout lets child Web items adhere to layout rules imposed by their parent.

Class Hierarchy



Pascal

```
TAdapterFormLayout = class(TLayoutWebContent, IGetAdapterFormLayout);
```

C++

```
class TAdapterFormLayout : public TLayoutWebContent, public IGetAdapterFormLayout;
```

File

WebForm

Description

TAdapterFormLayout is the default implementation of the ILayoutWebContent (see page 6084) interface for WebSnap applications. The Web items that belong to an adapter page producer (TAdapterPageProducer) pass a TAdapterFormLayout instance to their child Web items as the ParentLayout parameter of their Content method. This object provides helper functions that the child items use to fit their generated content into the parent object's HTML structure.

Use the TAdapterFormLayout properties to configure how the ILayoutWebContent (see page 6084) methods convert HTML that child Web items generate into a final form.

See Also

TLayout (see page 6112)

IWebContent (see page 6103)

ILayoutWebContent (see page 6084)

TAdapterPageProducer (see page 1136)

62.1.16.1 TAdapterFormLayout Members

The following tables list the members exposed by TAdapterFormLayout.

Protected Methods

	Name	Description
	_AddRef (see page 6017)	Responds when the caller acquires a reference to this object's interface.
	_Release (see page 6017)	Responds when the caller releases a reference to this object's interface.
	EndButtons (see page 6411)	This is EndButtons, a member of class TAdapterFormLayout.
	EndField (see page 6411)	This is EndField, a member of class TAdapterFormLayout.
	EndFields (see page 6411)	This is EndFields, a member of class TAdapterFormLayout.
	EndRow (see page 6412)	This is EndRow, a member of class TAdapterFormLayout.
	EndTable (see page 6412)	This is EndTable, a member of class TAdapterFormLayout.

	GetAdapterFormLayout (see page 6412)	This is GetAdapterFormLayout, a member of class TAdapterFormLayout.
	GetChildTableIndent (see page 6413)	This is GetChildTableIndent, a member of class TAdapterFormLayout.
	GetFieldIndent (see page 6413)	This is GetFieldIndent, a member of class TAdapterFormLayout.
	GetTableIndent (see page 6413)	This is GetTableIndent, a member of class TAdapterFormLayout.
V	ImplLayoutButton (see page 6413)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 6414)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 6414)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 6415)	Provides the underlying implementation for the LayoutTable method.
	LayoutButton (see page 6020)	Converts the HTML declaring a button into a form the associated Web item can use.
	LayoutField (see page 6020)	Converts the HTML declaring a field into a form the associated Web item can use.
	LayoutLabelAndField (see page 6020)	Converts the HTML declaring a labeled field into a form the associated Web item can use.
	LayoutTable (see page 6021)	Converts the HTML declaring a table into a form the associated Web item can use.
	NextColumn (see page 6415)	This is NextColumn, a member of class TAdapterFormLayout.
V	QueryInterface (see page 6021)	Returns a reference to a specified interface if the object supports that interface.
	StartButtons (see page 6415)	This is StartButtons, a member of class TAdapterFormLayout.
	StartFields (see page 6416)	This is StartFields, a member of class TAdapterFormLayout.
	StartRow (see page 6416)	This is StartRow, a member of class TAdapterFormLayout.
	StartTable (see page 6416)	This is StartTable, a member of class TAdapterFormLayout.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6410)	Creates a new TLayoutWebContent object.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	EndLayout (see page 6412)	Returns the string of HTML to insert after laying out all child items.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	LayoutButton (see page 6085)	Converts the HTML declaring a button into a form the Web item's parent can use.
	LayoutField (see page 6085)	Converts the HTML declaring a field into a form the Web item's parent can use.

	LayoutLabelAndField (see page 6086)	Converts the HTML declaring a labeled field into a form the Web item's parent can use.
	LayoutTable (see page 6086)	Converts the HTML declaring a table into a form the Web item's parent can use.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BreakButtons (see page 6417)	Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.
R	ChildTableIndent (see page 6417)	Specifies a string that appears before the <table> tags that child Web items generate.
	ColumnCount (see page 6417)	Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.
R	FieldIndent (see page 6418)	Specifies a string that appears before the HTML tags (other than <table> tags) that child Web items generate.
R	ParentLayout (see page 6022)	Provides access to the layout object of the Web item that is the parent of the Web item to which this object belongs.
	TableHeader (see page 6418)	Specifies the HTML to generate at the beginning of a table that lays out child elements.
R	TableIndent (see page 6419)	Specifies a string that appears before the <table> tags of the HTML table that contains child Web items.

62.1.16.2 TAdapterFormLayout.Create Constructor

Creates a new TLayoutWebContent object.

Pascal

```
constructor Create(AParentLayout: TLayout);
```

C++

```
__fastcall TAdapterFormLayout(TLayout AParentLayout);
```

Description

Call Create to instantiate a TLayoutWebContent descendant to use when fetching the content produced by child Web items.

AParentLayout is the layout object of the parent Web item (the grandparent of the child Web items that call this object's methods). AParentLayout becomes the value of the ParentLayout property.

See Also

ParentLayout (see page 6022)

62.1.16.3 TAdapterFormLayout Methods

The methods of the TAdapterFormLayout class are listed here.

Protected Methods

	Name	Description
	EndButtons (see page 6411)	This is EndButtons, a member of class TAdapterFormLayout.
	EndField (see page 6411)	This is EndField, a member of class TAdapterFormLayout.

	EndFields (see page 6411)	This is EndFields, a member of class TAdapterFormLayout.
	EndRow (see page 6412)	This is EndRow, a member of class TAdapterFormLayout.
	EndTable (see page 6412)	This is EndTable, a member of class TAdapterFormLayout.
	GetAdapterFormLayout (see page 6412)	This is GetAdapterFormLayout, a member of class TAdapterFormLayout.
	GetChildTableIndent (see page 6413)	This is GetChildTableIndent, a member of class TAdapterFormLayout.
	GetFieldIndent (see page 6413)	This is GetFieldIndent, a member of class TAdapterFormLayout.
	GetTableIndent (see page 6413)	This is GetTableIndent, a member of class TAdapterFormLayout.
V	ImplLayoutButton (see page 6413)	Provides the underlying implementation for the LayoutButton method.
V	ImplLayoutField (see page 6414)	Provides the underlying implementation for the LayoutField method.
V	ImplLayoutLabelAndField (see page 6414)	Provides the underlying implementation for the LayoutLabelAndField method.
V	ImplLayoutTable (see page 6415)	Provides the underlying implementation for the LayoutTable method.
	NextColumn (see page 6415)	This is NextColumn, a member of class TAdapterFormLayout.
	StartButtons (see page 6415)	This is StartButtons, a member of class TAdapterFormLayout.
	StartFields (see page 6416)	This is StartFields, a member of class TAdapterFormLayout.
	StartRow (see page 6416)	This is StartRow, a member of class TAdapterFormLayout.
	StartTable (see page 6416)	This is StartTable, a member of class TAdapterFormLayout.

Public Methods

	Name	Description
	EndLayout (see page 6412)	Returns the string of HTML to insert after laying out all child items.

62.1.16.3.1 TAdapterFormLayout.EndButtons Method

Pascal

```
function EndButtons: string;
```

C++

```
__fastcall UnicodeString EndButtons();
```

Description

This is EndButtons, a member of class TAdapterFormLayout.

62.1.16.3.2 TAdapterFormLayout.EndField Method

Pascal

```
function EndField: string;
```

C++

```
__fastcall UnicodeString EndField();
```

Description

This is EndField, a member of class TAdapterFormLayout.

62.1.16.3.3 TAdapterFormLayout.EndFields Method

Pascal

```
function EndFields: string;
```

C++

```
__fastcall UnicodeString EndFields();
```

Description

This is EndFields, a member of class TAdapterFormLayout.

62.1.16.3.4 TAdapterFormLayout.EndLayout Method

Returns the string of HTML to insert after laying out all child items.

Pascal

```
function EndLayout: string;
```

C++

```
__fastcall UnicodeString EndLayout();
```

Description

EndLayout generates the tags that must be inserted after all the content that child Web items generate. TAdapterFormLayout (see page 6408) keeps track of the ILayoutWebContent (see page 6084) method calls that are made so that it can know what tags need to be closed.

See Also

ILayoutWebContent (see page 6084)

62.1.16.3.5 TAdapterFormLayout.EndRow Method

Pascal

```
function EndRow: string;
```

C++

```
__fastcall UnicodeString EndRow();
```

Description

This is EndRow, a member of class TAdapterFormLayout.

62.1.16.3.6 TAdapterFormLayout.EndTable Method

Pascal

```
function EndTable: string;
```

C++

```
__fastcall UnicodeString EndTable();
```

Description

This is EndTable, a member of class TAdapterFormLayout.

62.1.16.3.7 TAdapterFormLayout.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TAdapterFormLayout.

62.1.16.3.8 TAdapterFormLayout.GetChildTableIndent Method

Pascal

```
function GetChildTableIndent: string;
```

C++

```
__fastcall UnicodeString GetChildTableIndent();
```

Description

This is GetChildTableIndent, a member of class TAdapterFormLayout.

62.1.16.3.9 TAdapterFormLayout.GetFieldIndent Method

Pascal

```
function GetFieldIndent: string;
```

C++

```
__fastcall UnicodeString GetFieldIndent();
```

Description

This is GetFieldIndent, a member of class TAdapterFormLayout.

62.1.16.3.10 TAdapterFormLayout.GetTableIndent Method

Pascal

```
function GetTableIndent: string;
```

C++

```
__fastcall UnicodeString GetTableIndent();
```

Description

This is GetTableIndent, a member of class TAdapterFormLayout.

62.1.16.3.11 TAdapterFormLayout.ImplLayoutButton Method

Provides the underlying implementation for the LayoutButton method.

Pascal

```
function ImplLayoutButton(const HTMLButton: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutButton(const UnicodeString HTMLButton,
TLayoutAttributes Attributes);
```

Description

The LayoutButton method calls ImplLayoutButton to convert the HTML that declares a button (typically an <INPUT> tag) into the form that the associated Web item requires.

HTMLButton is the HTML that declares a button. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the button.

LayoutButton returns the HTML for the button in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutButton is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutButton method.

See Also

LayoutButton (see page 6020)

62.1.16.3.12 TAdapterFormLayout.ImplLayoutField Method

Provides the underlying implementation for the LayoutField method.

Pascal

```
function ImplLayoutField(const HTMLField: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutField(const UnicodeString HTMLField,
TLayoutAttributes Attributes);
```

Description

The LayoutField method calls ImplLayoutField to convert the HTML that declares a field (an input control that represents a data field or a named value) into the form that the associated Web item requires.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field.

LayoutField returns the HTML for the field in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutField method.

See Also

LayoutField (see page 6020)

62.1.16.3.13 TAdapterFormLayout.ImplLayoutLabelAndField Method

Provides the underlying implementation for the LayoutLabelAndField method.

Pascal

```
function ImplLayoutLabelAndField(const HTMLLabel: string; const HTMLField: string; Attributes:
TLayoutAttributes): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutLabelAndField(const UnicodeString HTMLLabel, const
UnicodeString HTMLField, TLayoutAttributes Attributes);
```

Description

The LayoutLabelAndField method calls ImplLayoutLabelAndField to convert the HTML that declares a labeled field (an input control that represents a data field or a named value along with its label) into the form that the associated Web item requires.

HTMLLabel is the HTML that declares the label. This is produced by the child Web item.

HTMLField is the HTML that declares the field. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the field and its label.

LayoutLabelAndField returns the HTML for the field and label in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutLabelAndField is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutLabelAndField method.

See Also

LayoutLabelAndField (see page 6020)

62.1.16.3.14 TAdapterFormLayout.ImplLayoutTable Method

Provides the underlying implementation for the LayoutTable method.

Pascal

```
function ImplLayoutTable(const HTMLTable: string; Attributes: TLayoutAttributes): string;
override;
```

C++

```
virtual __fastcall UnicodeString ImplLayoutTable(const UnicodeString HTMLTable,
TLayoutAttributes Attributes);
```

Description

The LayoutTable method calls ImplLayoutTable to convert the HTML that declares a table (typically a structure used to layout the content produced by the child Web item's children) into the form that the associated Web item requires.

HTMLTable is the HTML that declares the table. This is produced by the child Web item.

Attributes describes any style attributes or layout considerations that should be applied to the final HTML for the table.

LayoutTable returns the HTML for the table in a form that the associated Web item can use.

In TLayoutWebContent, ImplLayoutTable is abstract or, in C++ terminology, pure virtual. Descendant classes override this method to provide an implementation for the LayoutTable method.

See Also

LayoutTable (see page 6021)

62.1.16.3.15 TAdapterFormLayout.NextColumn Method

Pascal

```
function NextColumn(SubItemCount: Integer): string;
```

C++

```
__fastcall UnicodeString NextColumn(int SubItemCount);
```

Description

This is NextColumn, a member of class TAdapterFormLayout.

62.1.16.3.16 TAdapterFormLayout.StartButtons Method

Pascal

```
function StartButtons: string;
```

C++

```
__fastcall UnicodeString StartButtons();
```

Description

This is StartButtons, a member of class TAdapterFormLayout.

62.1.16.3.17 TAdapterFormLayout.StartFields Method

Pascal

```
function StartFields(ColCount: Integer): string;
```

C++

```
__fastcall UnicodeString StartFields(int ColCount);
```

Description

This is StartFields, a member of class TAdapterFormLayout.

62.1.16.3.18 TAdapterFormLayout.StartRow Method

Pascal

```
function StartRow: string;
```

C++

```
__fastcall UnicodeString StartRow();
```

Description

This is StartRow, a member of class TAdapterFormLayout.

62.1.16.3.19 TAdapterFormLayout.StartTable Method

Pascal

```
function StartTable: string;
```

C++

```
__fastcall UnicodeString StartTable();
```

Description

This is StartTable, a member of class TAdapterFormLayout.

62.1.16.4 TAdapterFormLayout Properties

The properties of the TAdapterFormLayout class are listed here.

Public Properties

	Name	Description
	BreakButtons (see page 6417)	Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.
R	ChildTableIndent (see page 6417)	Specifies a string that appears before the <table> tags that child Web items generate.
	ColumnCount (see page 6417)	Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.
R	FieldIndent (see page 6418)	Specifies a string that appears before the HTML tags (other than <table> tags) that child Web items generate.
	TableHeader (see page 6418)	Specifies the HTML to generate at the beginning of a table that lays out child elements.
R	TableIndent (see page 6419)	Specifies a string that appears before the <table> tags of the HTML table that contains child Web items.

62.1.16.4.1 TAdapterFormLayout.BreakButtons Property

Indicates whether the associated Web item uses a grid to lay out buttons that child Web items generate.

Pascal

```
property BreakButtons: Boolean;
```

C++

```
__property Boolean BreakButtons;
```

Description

Set BreakButtons to true if the Web item that supplies this object to its child Web items uses an HTML table to position HTML buttons that those child Web items generate. Set BreakButtons to false if the Web item does not use an HTML table to position child buttons.

BreakButtons affects the whether the LayoutButton method on this object's ILayoutWebContent (see page 6084) interface adds <TD> tags to the HTML that child Web items generate for HTML buttons.

See Also

LayoutButton (see page 6085)

62.1.16.4.2 TAdapterFormLayout.ChildTableIndent Property

Specifies a string that appears before the <table> tags that child Web items generate.

Pascal

```
property ChildTableIndent: string;
```

C++

```
__property UnicodeString ChildTableIndent;
```

Description

Child Web items read ChildTableIndent from the implementation of their Content method to create the proper indentation of the HTML for a table. ChildTableIndent is a string of spaces that reflects the nesting of HTML tags for an HTML table tag that the child Web item generates.

Note: Do not confuse ChildTableIndent with the TableIndent (see page 6419) property. ChildTableIndent gives the indentation of a table that the child Web item generates. TableIndent (see page 6419) gives the indentation of a table that the associated Web item generates. (The associated Web item is the one that passes this object to the Content method of its child Web items.)

See Also

FieldIndent (see page 6418)

Content (see page 6104)

TableIndent (see page 6419)

62.1.16.4.3 TAdapterFormLayout.ColumnCount Property

Indicates the number of child elements appear in a single row when the associated Web item uses a grid to lay them out.

Pascal

```
property ColumnCount: Integer;
```

C++

```
__property int ColumnCount;
```

Description

Set ColumnCount to the number of child elements that appear in a single row when laying out child controls by rows.

ColumnCount affects where the LayoutFields or LayoutButtons method on this object's ILayoutWebContent (see page 6084) interface adds <TR> and </TR> tags to the HTML that child Web items generate.

Note: ColumnCount only affects the LayoutButtons method if the value of the BreakButtons (see page 6417) property is true.

See Also

LayoutButton (see page 6085)

LayoutField (see page 6085)

BreakButtons (see page 6417)

62.1.16.4.4 TAdapterFormLayout.FieldIndent Property

Specifies a string that appears before the HTML tags (other than <table> tags) that child Web items generate.

Pascal

```
property FieldIndent: string;
```

C++

```
__property UnicodeString FieldIndent;
```

Description

Child Web items read FieldIndent from the implementation of their Content method to create the proper indentation of the HTML for individual fields or controls. FieldIndent is a string of spaces that reflects the nesting of HTML tags for the tag that the child Web item generates.

See Also

ChildTableIndent (see page 6417)

Content (see page 6104)

TableIndent (see page 6419)

62.1.16.4.5 TAdapterFormLayout.TableHeader Property

Specifies the HTML to generate at the beginning of a table that lays out child elements.

Pascal

```
property TableHeader: string;
```

C++

```
__property UnicodeString TableHeader;
```

Description

Set TableHeader to the HTML tag that appears at the beginning of the HTML table used to control the layout of child elements. TableHeader is a string that includes the <table> tag, along with any attributes (such as alignment) that affect the layout of child items.

See Also

LayoutButton (see page 6085)

LayoutField (see page 6085)

BreakButtons (see page 6417)

62.1.16.4.6 TAdapterFormLayout.TableIndent Property

Specifies a string that appears before the <table> tags of the HTML table that contains child Web items.

Pascal

```
property TableIndent: string;
```

C++

```
__property UnicodeString TableIndent;
```

Description

TAdapterFormLayout (see page 6408) uses TableIndent internally to create the proper indentation of the HTML table tags for the table used to lay out child Web items. For example, the EndLayout (see page 6412) method uses TableIndent to properly format its </table> tag.

TableIndent is a string of spaces that reflects the nesting of HTML tags for the HTML table tag that the associated Web item generates.

Note: Do not confuse TableIndent with the ChildTableIndent (see page 6417) property. TableIndent gives the indentation of a table that the associated Web item generates. (The associated Web item is the one that passes this object to the Content method of its child Web items.) ChildTableIndent (see page 6417) gives the indentation of a table that the child Web item generates.

See Also

FieldIndent (see page 6418)

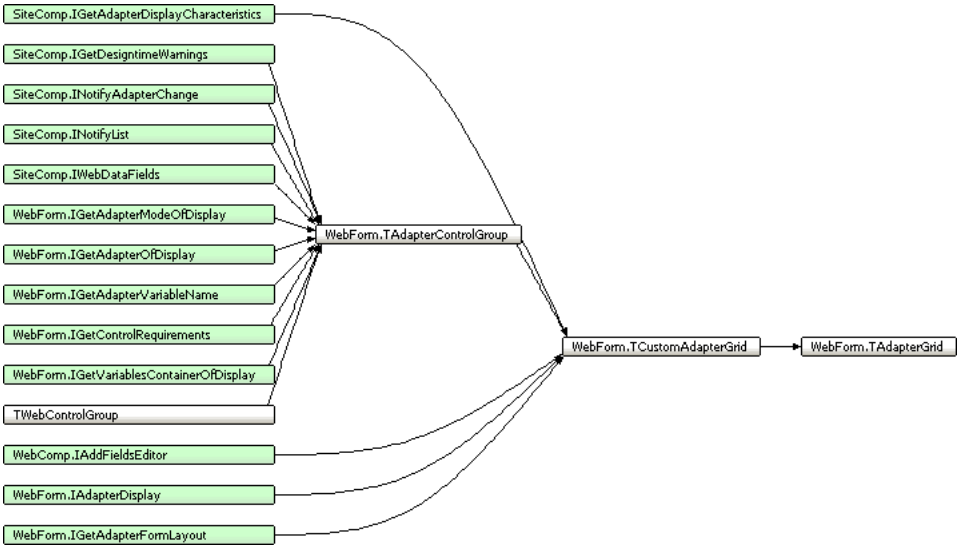
EndLayout (see page 6412)

ChildTableIndent (see page 6417)

62.1.17 TAdapterGrid Class

TAdapterGrid is a container for Adapter (see page 6422) display field components such as TAdapterDisplayColumn (see page 6364). TAdapterGrid generates an HTML table.

Class Hierarchy



Pascal

```
TAdapterGrid = class(TCustomAdapterGrid);
```

C++

```
class TAdapterGrid : public TCustomAdapterGrid;
```

File

WebForm

Description

Add TAdapterGrid as a child of TAdapterForm (see page 6404). TAdapterGrid is associated with an Adapter (see page 6422). All of it's child display field components will use the same Adapter (see page 6422).

If no display fields have been explicitly added to the TAdapterGrid, display fields associated with all of the Adapter (see page 6422) fields will be added by default. The properties of default display fields cannot be modified.

Modifying the AdapterMode (see page 6423) property of TAdapterGrid may change the default display type or input type of Adapter (see page 6422) fields. This may cause the display field children of TAdapterGrid to change appearance.

Use the properties and methods of TAdapterGrid to:

Specify the associated Adapter (see page 6422).

Specify when the Adapter (see page 6422) Mode.

Specify whether default fields should be added.

Customize the appearance of the HTML table.

See Also

TAdapterDisplayColumn (see page 6364)

TAdapterForm (see page 6404)

62.1.17.1 TAdapterGrid Members

The following tables list the members exposed by TAdapterGrid.

Protected Methods

	Name	Description
	AdapterChange (see page 6345)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6345)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6346)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6346)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6346)	Returns the value of the Adapter (see page 6352) property.
	GetAdapterDisplayCharacteristics (see page 6500)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 6500)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAdapterModeOfAdapter (see page 6347)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6347)	Returns the value of the AdapterMode (see page 6353) property.
	GetAdapterVariableName (see page 6348)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6500)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 6501)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 6501)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetControlRequirements (see page 6348)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6348)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6349)	Fills a string list with the names of all of the associated adapter's fields.
	GetNewFieldPrefix (see page 6501)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
	GetVisibleFields (see page 6349)	Returns the value of the VisibleFields (see page 6354) property.
	ImplCanAddClass (see page 6501)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6502)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6350)	Provides the underlying implementation for the GetControlRequirements (see page 6348) method.
	IsFieldInUse (see page 6350)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6350)	Updates the TAdapterControlGroup (see page 6341) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6351)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6351)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6352)	Sets the value of the Adapter (see page 6352) property.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	Create (see page 6343)	Creates and initializes an instance of TAdapterControlGroup (see page 6341).
	Create (see page 6499)	Creates and initializes an instance of TCustomAdapterGrid (see page 6496).
	Destroy (see page 6344)	Frees an instance of TAdapterControlGroup (see page 6341).
	Destroy (see page 6499)	Frees an instance of TCustomAdapterGrid (see page 6496).
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6513)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterDisplayCharacteristics (see page 1019)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAdapterFormLayout (see page 6511)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.
	GetAdapterModeOfAdapter (see page 6512)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.

	GetAdapterModeOfDisplay (see page 6512)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 6354)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

Published Properties

	Name	Description
	Adapter (see page 6422)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6423)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6423)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 6424)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6424)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6425)	Determines the basic display characteristics of the table.

62.1.17.2 TAdapterGrid Properties

The properties of the TAdapterGrid class are listed here.

Published Properties

	Name	Description
	Adapter (see page 6422)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6423)	Specifies the mode of the adapter that the controls in the group support.
	AddDefaultFields (see page 6423)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 6424)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6424)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6425)	Determines the basic display characteristics of the table.

62.1.17.2.1 TAdapterGrid.Adapter Property

Specifies the adapter that supplies values to or receives values from the controls in the group.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to link this adapter control group and all its Web items to an adapter component. Each component that the WebFieldControls property lists generates an HTML control that represents the value of one of the adapter's fields.

See Also

GetFieldsList (see page 5549)

GetFieldsList (see page 6349)

AdapterMode (see page 6353)

WebFieldControls

62.1.17.2.2 TAdapterGrid.AdapterMode Property

Specifies the mode of the adapter that the controls in the group support.

Pascal

```
property AdapterMode: string;
```

C++

```
__property UnicodeString AdapterMode;
```

Description

Some adapter components support different modes, or states. For example, dataset adapters can be in Edit, Insert, Browse, or Query mode. The mode of the adapter has an effect on the types of controls that the control group's Web items generate, and how those controls behave.

If the adapter that is assigned to the Adapter property supports different modes, you can set AdapterMode to force the controls in the group to represent the adapter fields in the specified mode. When AdapterMode is blank, the adapter control group uses the current mode of the adapter at the time when it generates its controls in the HTML form.

The possible values of AdapterMode depend on the modes that the adapter supports. If the adapter supports different modes, you can select a value for AdapterMode from a drop-down list in the Object Inspector that lists all the possible mode strings. If the adapter does not support different modes, the Object Inspector does not display a list, and the value of AdapterMode should be blank.

See Also

Mode

Adapter (see page 6352)

WebFieldControls

62.1.17.2.3 TAdapterGrid.AddDefaultFields Property

Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.

Pascal

```
property AddDefaultFields: Boolean;
```

C++

```
__property Boolean AddDefaultFields;
```

Description

Set AddDefaultFields to indicate how you want the adapter control group to respond when you do not explicitly add child Web items. You can add child Web items at design time using Adapter page editor, or by right-clicking the control group and selecting Add Fields or Add Component. You can add child Web items at runtime using the Add method of the WebFieldControls property.

When AddDefaultFields is true, if you do not add child Web items to the adapter control group, it generates a default set of child Web items to display the fields of the adapter component that the Adapter property specifies.

When AddDefaultFields is false, if you do not add child Web items to the adapter control group, it remains empty, displaying no field values.

See Also

VisibleFields (see page 6354)

WebFieldControls

Adapter (see page 6352)

62.1.17.2.4 TAdapterGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table. HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes or RowAttributes property instead.

See Also

TableAttributes (see page 6503)

RowAttributes (see page 6503)

62.1.17.2.5 TAdapterGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes or HeadingAttributes property instead.

See Also

TableAttributes (see page 6503)

HeadingAttributes (see page 6502)

62.1.17.2.6 TAdapterGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes property instead.

See Also

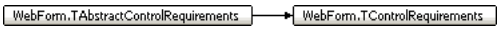
RowAttributes (see page 6503)

HeadingAttributes (see page 6502)

62.1.18 TControlRequirements Class

TControlRequirements lists the variables and functions that the controls in an adapter form require to generate their HTML.

Class Hierarchy



Pascal

```
TControlRequirements = class(TAbstractControlRequirements);
```

C++

```
class TControlRequirements : public TAbstractControlRequirements;
```

File

WebForm

Description

Adapter forms create instances of TControlRequirements to store information about variables and functions that are needed for generating an HTML form to display the controls they contain. Components that generate HTML controls to represent the fields of an adapter (TAdapterControlGroup (see page 6341) and TAdapterDataDisplay (see page 6354) descendants) use the TControlRequirements instance that the form provides to store these variable and function declarations.

Typically, applications do not need to create or use TControlRequirements instances. Component writers that create Web items for an adapter form (that is, components that generate HTML controls to reflect the fields of an adapter) use the methods of TControlRequirements when implementing the GetControlRequirements method.

See Also

TAdapterControlGroup (see page 6341)

TAdapterDataDisplay (see page 6354)

TAdapterForm (see page 6404)

62.1.18.1 TControlRequirements Members

The following tables list the members exposed by TControlRequirements.

Public Methods

	Name	Description
	Create (see page 6426)	Creates and initializes an instance of TControlRequirements (see page 6425).
V	DefineFunction (see page 6427)	Adds the declaration of a function to those that this control requirements object stores.
V	DefineVariable (see page 6428)	Adds the declaration of a variable to those that this control requirements object stores.
V	Destroy (see page 6427)	Frees an instance of TControlRequirements (see page 6425).
V	IsFunctionDefined (see page 6428)	Indicates whether a function with the specified name has already been added to the list of control requirements.
V	IsVariableDefined (see page 6429)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
V	RequiresMultipartForm (see page 6429)	Stores the fact that the generated HTML form should be multipart.

Public Properties

	Name	Description
R	NeedMultipartForm (see page 6429)	Indicates whether the form that the adapter form generates must be multipart.

62.1.18.2 TControlRequirements.Create Constructor

Creates and initializes an instance of TControlRequirements (see page 6425).

Pascal

```
constructor Create;
```

C++

```
__fastcall TControlRequirements();
```

Description

Applications should have no need to instantiate TControlRequirements (see page 6425). The adapter form creates instances of this object to assemble and store information from their Web items.

See Also

TAdapterForm (see page 6404)

62.1.18.3 TControlRequirements.Destroy Destructor

Frees an instance of TControlRequirements (see page 6425).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TControlRequirements();
```

Description

Applications should have no need to create or destroy TControlRequirements (see page 6425) instances. These objects are created and freed by the adapter form.

See Also

TAdapterForm (see page 6404)

62.1.18.4 TControlRequirements Methods

The methods of the TControlRequirements class are listed here.

Public Methods

	Name	Description
	DefineFunction (see page 6427)	Adds the declaration of a function to those that this control requirements object stores.
	DefineVariable (see page 6428)	Adds the declaration of a variable to those that this control requirements object stores.
	IsFunctionDefined (see page 6428)	Indicates whether a function with the specified name has already been added to the list of control requirements.
	IsVariableDefined (see page 6429)	Indicates whether a variable with the specified name has already been added to the list of control requirements.
	RequiresMultipartForm (see page 6429)	Stores the fact that the generated HTML form should be multipart.

62.1.18.4.1 TControlRequirements.DefineFunction Method

Adds the declaration of a function to those that this control requirements object stores.

Pascal

```
procedure DefineFunction(const AName: string; const ABody: string); override;
```

C++

```
virtual __fastcall DefineFunction(const UnicodeString AName, const UnicodeString ABody);
```

Description

DefineFunction stores the declaration of a function that has the name AName, where ABody is the text of the function body.

AName is the name of the function. This name is one of a standard set that the adapter form knows to call under specific circumstances. For example, the name can be the standard name of the function to generate an HTML button.

ABody is the body of the specified function. This value is typically customized to reflect the properties of a specific control in the

form.

See Also

IsFunctionDefined (see page 6428)

DefineVariable (see page 6428)

62.1.18.4.2 TControlRequirements.DefineVariable Method

Adds the declaration of a variable to those that this control requirements object stores.

Pascal

```
procedure DefineVariable(const AName: string; const AValue: string; AAdapter: Boolean);
override;
```

C++

```
virtual __fastcall DefineVariable(const UnicodeString AName, const UnicodeString AValue,
Boolean AAdapter);
```

Description

DefineVariable stores the name and value of a variable declaration.

AName is the name of the variable.

AValue is the value to assign to the variable.

AAdapter indicates whether this variable is the one that declares the adapter object whose field a control represents or whose set of fields a control group represents.

See Also

IsVariableDefined (see page 6429)

DefineFunction (see page 6427)

62.1.18.4.3 TControlRequirements.IsFunctionDefined Method

Indicates whether a function with the specified name has already been added to the list of control requirements.

Pascal

```
function IsFunctionDefined(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean IsFunctionDefined(const UnicodeString AName);
```

Description

IsFunctionDefined returns true if the control requirements object already lists a function with the specified name. That is, IsFunctionDefined returns true if the DefineFunction (see page 6427) method has been called to define the specified function, and false otherwise.

AName is the name of the function.

See Also

DefineFunction (see page 6427)

IsVariableDefined (see page 6429)

62.1.18.4.4 TControlRequirements.IsVariableDefined Method

Indicates whether a variable with the specified name has already been added to the list of control requirements.

Pascal

```
function IsVariableDefined(const AName: string): Boolean; override;
```

C++

```
virtual __fastcall Boolean IsVariableDefined(const UnicodeString AName);
```

Description

IsVariableDefined returns true if the control requirements object already lists a variable with the specified name. That is, IsVariableDefined returns true if the DefineVariable (see page 6428) method has been called to define a value for the specified variable, and false otherwise.

AName is the name of the variable.

See Also

DefineVariable (see page 6428)

IsFunctionDefined (see page 6428)

62.1.18.4.5 TControlRequirements.RequiresMultipartForm Method

Stores the fact that the generated HTML form should be multipart.

Pascal

```
procedure RequiresMultipartForm; override;
```

C++

```
virtual __fastcall RequiresMultipartForm();
```

Description

Web items call this method if they need to be contained in a multipart form (for example, when a field in the form is represented by a reference to another file). A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

See Also

NeedMultipartForm (see page 6429)

62.1.18.5 TControlRequirements Properties

The properties of the TControlRequirements class are listed here.

Public Properties

	Name	Description
	NeedMultipartForm (see page 6429)	Indicates whether the form that the adapter form generates must be multipart.

62.1.18.5.1 TControlRequirements.NeedMultipartForm Property

Indicates whether the form that the adapter form generates must be multipart.

Pascal

```
property NeedMultipartForm: Boolean;
```

C++

```
__property Boolean NeedMultipartForm;
```

Description

NeedMultipartForm indicates whether any control has called the RequiresMultipartForm (see page 6429) method to indicate that the form that contains it must be multipart. A multipart form's HTML declaration includes the attribute 'enctype=multipart/form-data'.

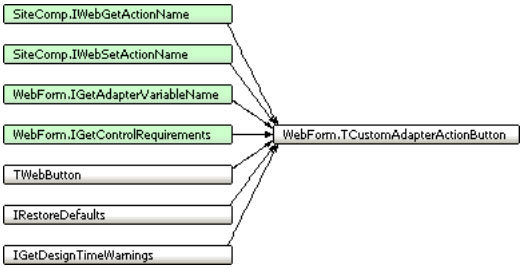
See Also

RequiresMultipartForm (see page 6429)

62.1.19 TCustomAdapterActionButton Class

TCustomAdapterActionButton is the base class for Web items that generate HTML controls for executing adapter actions.

Class Hierarchy



Pascal

```
TCustomAdapterActionButton = class(TWebButton, IWebGetActionName, IWebSetActionName,
IRestoreDefaults, IGetControlRequirements, IGetAdapterVariableName, IGetDesignTimeWarnings);
```

C++

```
class TCustomAdapterActionButton : public TWebButton, public IWebGetActionName, public
IWebSetActionName, public IRestoreDefaults, public IGetControlRequirements, public
IGetAdapterVariableName, public IGetDesignTimeWarnings;
```

File

WebForm

Description

Use TCustomAdapterActionButton as a base class when defining Web items that generate HTML controls for adapter actions. TCustomAdapterActionButton descendants are added to Web items that appear in the hierarchy of a TAdapterForm (see page 6404) object. The adapter action button generates an HTML control that appears in the HTML form. When the end user clicks that control, it sends a request message that executes the adapter action.

TCustomAdapterActionButton introduces properties to link this component to the adapter action it executes, and for defining the type and appearance of the generated control in the HTML form.

See Also

TAdapterActionButton (see page 6316)

TCustomAdapterAction (see page 5579)

TAdapterForm (see page 6404)

62.1.19.1 TCustomAdapterActionButton Members

The following tables list the members exposed by TCustomAdapterActionButton.

Protected Methods

	Name	Description
	FindAssociatedAction (see page 6433)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 6433)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 6433)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 6433)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
V	GetCaption (see page 6434)	Returns the value of the Caption property.
	GetControlRequirements (see page 6434)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 6434)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 6434)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
V	GetLayoutAttributes (see page 6435)	Returns the layout object that specifies how the generated control should look.
V	ImplContent (see page 6435)	Provides the underlying implementation for the Content method.
V	ImplGetControlRequirements (see page 6435)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
V	ImplRestoreDefaults (see page 6436)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
V	Notification (see page 6436)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 6436)	Returns the Web item to a default state.
	SetActionName (see page 6437)	This is SetActionName, a member of class TCustomAdapterActionButton.
V	SetCaption (see page 6437)	Sets the value of the Caption property.

Public Methods

	Name	Description
V	Create (see page 6431)	Creates and initializes an instance of TCustomAdapterActionButton (see page 6430).
V	Destroy (see page 6432)	Frees an instance of TCustomAdapterActionButton (see page 6430).

Public Properties

	Name	Description
	ActionName (see page 6438)	Specifies the name of the action taken by the Web button.
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6438)	Specifies the horizontal alignment of the generated HTML control.
	DisplayColumns (see page 6438)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6439)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6439)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6440)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6440)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 6441)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6441)	Specifies the width of the generated element if it is a clickable image.
	PageName (see page 6441)	Specifies the name of the page to display after successfully executing the action.
	VAlign (see page 6442)	Specifies the vertical alignment of the generated HTML control.

62.1.19.2 TCustomAdapterActionButton.Create Constructor

Creates and initializes an instance of TCustomAdapterActionButton (see page 6430).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterActionButton(TComponent * AOwner);
```

Description

Call Create to instantiate an adapter action button at runtime. Adapter action buttons added to an adapter form at design time are created automatically.

AOwner is the component, typically a Web module, that is responsible for freeing the TCustomAdapterActionButton (see page 6430) instance. This becomes the value of the Owner property.

See Also

Owner

62.1.19.3 TCustomAdapterActionButton.Destroy Destructor

Frees an instance of TCustomAdapterActionButton (see page 6430).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterActionButton();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the TCustomAdapterActionButton (see page 6430) is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

62.1.19.4 TCustomAdapterActionButton Methods

The methods of the TCustomAdapterActionButton class are listed here.

Protected Methods

	Name	Description
	FindAssociatedAction (see page 6433)	This is FindAssociatedAction, a member of class TCustomAdapterActionButton.
	GetActionName (see page 6433)	This is GetActionName, a member of class TCustomAdapterActionButton.
	GetAdapterModeOfAdapter (see page 6433)	This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.
	GetAdapterVariableName (see page 6433)	This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.
	GetCaption (see page 6434)	Returns the value of the Caption property.
	GetControlRequirements (see page 6434)	This is GetControlRequirements, a member of class TCustomAdapterActionButton.
	GetDesigntimeWarnings (see page 6434)	This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.
	GetHTMLFormVarName (see page 6434)	This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.
	GetLayoutAttributes (see page 6435)	Returns the layout object that specifies how the generated control should look.

	ImplContent (see page 6435)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6435)	This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.
	ImplRestoreDefaults (see page 6436)	This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.
	Notification (see page 6436)	Forwards notification messages to all owned components.
	RestoreDefaults (see page 6436)	Returns the Web item to a default state.
	SetActionName (see page 6437)	This is SetActionName, a member of class TCustomAdapterActionButton.
	SetCaption (see page 6437)	Sets the value of the Caption property.

62.1.19.4.1 TCustomAdapterActionButton.FindAssociatedAction Method

Pascal

```
function FindAssociatedAction: TComponent;
```

C++

```
__fastcall TComponent * FindAssociatedAction();
```

Description

This is FindAssociatedAction, a member of class TCustomAdapterActionButton.

62.1.19.4.2 TCustomAdapterActionButton.GetActionName Method

Pascal

```
function GetActionName: string;
```

C++

```
__fastcall UnicodeString GetActionName();
```

Description

This is GetActionName, a member of class TCustomAdapterActionButton.

62.1.19.4.3 TCustomAdapterActionButton.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class TCustomAdapterActionButton.

62.1.19.4.4 TCustomAdapterActionButton.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class TCustomAdapterActionButton.

62.1.19.4.5 TCustomAdapterActionButton.GetCaption Method

Returns the value of the Caption property.

Pascal

```
function GetCaption: string; override;
```

C++

```
virtual __fastcall UnicodeString GetCaption();
```

Description

GetCaption is the protected read implementation for the Caption property.

See Also

Caption

SetCaption

62.1.19.4.6 TCustomAdapterActionButton.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class TCustomAdapterActionButton.

62.1.19.4.7 TCustomAdapterActionButton.GetDesigntimeWarnings Method

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

This is GetDesigntimeWarnings, a member of class TCustomAdapterActionButton.

62.1.19.4.8 TCustomAdapterActionButton.GetHTMLFormVarName Method

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormVarName();
```

Description

This is GetHTMLFormVarName, a member of class TCustomAdapterActionButton.

62.1.19.4.9 TCustomAdapterActionButton.GetLayoutAttributes Method

Returns the layout object that specifies how the generated control should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebButton, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because the layout of buttons does not make use of layout attributes.

See Also

Content

62.1.19.4.10 TCustomAdapterActionButton.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the button control. TWebButton introduces ImplContent as an abstract or, in C++ terminology, pure virtual method. TWebButton descendants must override ImplContent to generate the definition of their particular button control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web button should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the button control.

See Also

Content

ILayoutWebContent (see page 6084)

62.1.19.4.11 TCustomAdapterActionButton.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); virtual;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterActionButton.

62.1.19.4.12 TCustomAdapterActionButton.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; virtual;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterActionButton.

62.1.19.4.13 TCustomAdapterActionButton.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

62.1.19.4.14 TCustomAdapterActionButton.RestoreDefaults Method

Returns the Web item to a default state.

Pascal

```
procedure RestoreDefaults;
```

C++

```
__fastcall RestoreDefaults();
```

Description

The Web page editor calls RestoreDefaults when the user chooses the Restore Defaults command. Web items implement this method to return property values to a default state.

62.1.19.4.15 TCustomAdapterActionButton.SetActionName Method

Pascal

```
procedure SetActionName(const AValue: string);
```

C++

```
__fastcall SetActionName(const UnicodeString AValue);
```

Description

This is SetActionName, a member of class TCustomAdapterActionButton.

62.1.19.4.16 TCustomAdapterActionButton.SetCaption Method

Sets the value of the Caption property.

Pascal

```
procedure SetCaption(const Value: string); override;
```

C++

```
virtual __fastcall SetCaption(const UnicodeString Value);
```

Description

SetCaption is the protected write implementation for the Caption property.

See Also

Caption

GetCaption

62.1.19.5 TCustomAdapterActionButton Properties

The properties of the TCustomAdapterActionButton class are listed here.

Public Properties

	Name	Description
	ActionName (see page 6438)	Specifies the name of the action taken by the Web button.
	Align (see page 6438)	Specifies the horizontal alignment of the generated HTML control.
	DisplayColumns (see page 6438)	Indicates the number of columns to use when laying out the generated HTML control.
	DisplayType (see page 6439)	Specifies the DisplayType of the generated HTML control.
	ErrorPageName (see page 6439)	Specifies the name of the page to display when an error occurs executing the action.
	HideOptions (see page 6440)	Controls when the generated HTML element appears in HTML forms.
	ImageHeight (see page 6440)	Specifies the height of the generated element if it is a clickable image.
	ImageProducer (see page 6441)	Specifies the interface that this adapter action button uses to generate images.
	ImageWidth (see page 6441)	Specifies the width of the generated element if it is a clickable image.

	PageName (see page 6441)	Specifies the name of the page to display after successfully executing the action.
	VAlign (see page 6442)	Specifies the vertical alignment of the generated HTML control.

62.1.19.5.1 TCustomAdapterActionButton.ActionName Property

Specifies the name of the action taken by the Web button.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

Use ActionName to get or set the name of the adapter action that executes in response to requests sent when the user clicks this object's generated control. ActionName is the same as the ActionName property of the adapter action object that this control executes.

See Also

Caption

ActionName (see page 5586)

62.1.19.5.2 TCustomAdapterActionButton.Align Property

Specifies the horizontal alignment of the generated HTML control.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the generated HTML control is positioned horizontally within its parent HTML element. If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each individual of the items within the grid generated to position them into rows.

Align indicates the desired value of the Align option of the HTML tag for the generated control.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6442)

DisplayColumns (see page 6438)

ItemCount (see page 5475)

62.1.19.5.3 TCustomAdapterActionButton.DisplayColumns Property

Indicates the number of columns to use when laying out the generated HTML control.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

An adapter action may represent multiple commands. When the TCustomAdapterActionButton (see page 6430) object generates controls for such an action, it must handle the layout of individual items for each of the action's commands. DisplayColumns controls how many items (controls) appear in a row before TCustomAdapterActionButton (see page 6430) starts a new row of controls.

If the corresponding adapter action does not represent multiple commands (that is, if its ItemCount property is not greater than 1), the value of DisplayColumns has no effect.

If DisplayColumns is –1, the Web browser controls the layout of the adapter action's items.

See Also

ItemCount (see page 5475)

DisplayType (see page 6439)

62.1.19.5.4 TCustomAdapterActionButton.DisplayType Property

Specifies the DisplayType of the generated HTML control.

Pascal

```
property DisplayType: TCommandHTMLElementType;
```

C++

```
__property TCommandHTMLElementType DisplayType;
```

Description

DisplayType controls the type of HTML element that this Web item generates.

When DisplayType is ctDefault (the default), TCustomAdapterActionButton (see page 6430) uses the element type that the associated adapter action suggests. Change DisplayType to generate a different type of control or set of controls.

When DisplayType is ctImage or ctEventImages, you must set the ImageProducer (see page 6441) property to a component that can provide the image or images.

See Also

ImageProducer (see page 6441)

Caption

GetDisplayStyle (see page 5457)

62.1.19.5.5 TCustomAdapterActionButton.ErrorPageName Property

Specifies the name of the page to display when an error occurs executing the action.

Pascal

```
property ErrorPageName: string;
```

C++

```
__property UnicodeString ErrorPageName;
```

Description

Use ErrorPageName to indicate the Web page that should appear if the WebSnap application encounters an error when executing the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

PageName (see page 6441)

62.1.19.5.6 TCustomAdapterActionButton.HideOptions Property

Controls when the generated HTML element appears in HTML forms.

Pascal

```
property HideOptions: TActionButtonHideOptions;
```

C++

```
__property TActionButtonHideOptions HideOptions;
```

Description

HideOptions controls when this Web item generates an HTML control for its associated action. This property is a set of flags that control when the adapter page producer suppresses the controls that this object generates. If HideOptions is an empty set, TCustomAdapterActionButton (see page 6430) always generates a control for its associated adapter action.

See Also

GetVisible (see page 5462)

WebEnabled (see page 5473)

HasExecuteAccess (see page 5463)

62.1.19.5.7 TCustomAdapterActionButton.ImageHeight Property

Specifies the height of the generated element if it is a clickable image.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

ImageHeight controls the height of the image that the user can click to execute the associated action. If ImageHeight is –1 (the default), the browser controls the height it uses to display the generated image.

ImageHeight is only used when the value of the DisplayType (see page 6439) property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6441)

ImageWidth (see page 6441)

DisplayType (see page 6439)

62.1.19.5.8 TCustomAdapterActionButton.ImageProducer Property

Specifies the interface that this adapter action button uses to generate images.

Pascal

```
property ImageProducer: IActionImageProducer;
```

C++

```
__property IActionImageProducer ImageProducer;
```

Description

Set ImageProducer when the value of the DisplayType (see page 6439) is ctImage or ctEventImages. ImageProducer is the interface of the component that supplies the image or images that appear in HTML forms as the control that this Web item generates. When the user clicks on the generated image in the HTML form, the associated action executes.

See Also

DisplayType (see page 6439)

ImageHeight (see page 6440)

ImageWidth (see page 6441)

62.1.19.5.9 TCustomAdapterActionButton.ImageWidth Property

Specifies the width of the generated element if it is a clickable image.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

ImageWidth controls the width of the image that the user can click to execute the associated action. If ImageWidth is –1 (the default), the browser controls the width it uses to display the generated image.

ImageWidth is only used when the value of the DisplayType (see page 6439) property is ctImage or ctEventImages.

See Also

ImageProducer (see page 6441)

ImageHeight (see page 6440)

DisplayType (see page 6439)

62.1.19.5.10 TCustomAdapterActionButton.PageName Property

Specifies the name of the page to display after successfully executing the action.

Pascal

```
property PageName: string;
```

C++

```
__property UnicodeString PageName;
```

Description

Use PageName to indicate the Web page that should appear if the WebSnap application successfully executes the adapter action in response to the user clicking the generated control. This is the name used to identify the Web page in server-side scripts, and appears as the pathinfo portion of the URL for that page if it is published.

See Also

ErrorPageName (see page 6439)

62.1.19.5.11 TCustomAdapterActionButton.VAlign Property

Specifies the vertical alignment of the generated HTML control.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the generated HTML control is positioned vertically within its parent HTML element. VAlign indicates the desired value of the VAlign option of the HTML tag for the generated control. . If the adapter action button is included in the column of an adapter grid, this property has no effect unless the associated action has multiple items. It affects the position of the HTML controls for each of the individual items within the grid generated to position them into rows.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 6438)

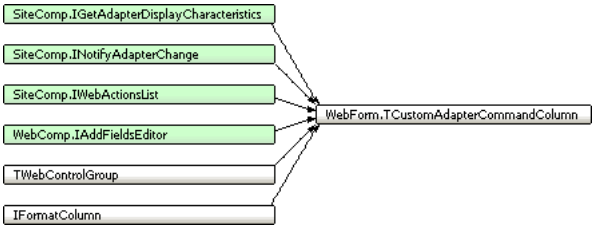
DisplayColumns (see page 6438)

ItemCount (see page 5475)

62.1.20 TCustomAdapterCommandColumn Class

TCustomAdapterCommandColumn is the base class for components that generate a column of action controls in an adapter grid.

Class Hierarchy



Pascal

```
TCustomAdapterCommandColumn = class(TWebControlGroup, IWebActionsList, IAddFieldsEditor,
IFormatColumn, IGetAdapterDisplayCharacteristics, INotifyAdapterChange);
```

C++

```
class TCustomAdapterCommandColumn : public TWebControlGroup, public IWebActionsList, public
IAddFieldsEditor, public IFormatColumn, public IGetAdapterDisplayCharacteristics, public
INotifyAdapterChange;
```

File

WebForm

Description

Use TCustomAdapterCommandColumn as a base class when creating a specialized display control that organizes the output of several adapter action buttons (TAdapterActionButton (see page 6316) objects) into a column of an adapter grid (TAdapterGrid (see page 6419)). TCustomAdapterCommandColumn generates the HTML to lay out a single column of an adapter grid. Each cell in the column can contain one or more adapter action buttons, each of which executes an adapter action. For each row in the grid, the column contains duplicates of the same set of adapter action buttons.

Use the TCustomAdapterCommandColumn properties to specify which adapter action buttons appear in the column and to customize their layout and appearance. These properties include:

The AddDefaultCommands (see page 6449), WebFieldControls, and VisibleCommands (see page 6453) properties, which specify which adapter action buttons appear in the column.

The Caption (see page 6450) and CaptionAttributes (see page 6451) properties, which determine the appearance of the label at the top of the column.

The Align (see page 6450), VAlign (see page 6453), and DisplayColumns (see page 6451) properties, which determine the layout of controls inside each cell.

The Style (see page 6452) and StyleRule (see page 6452) properties, which let you apply a style from a stylesheet to the column.

See Also

TAdapterCommandColumn (see page 6331)

TAdapterGrid (see page 6419)

TAdapterEditColumn (see page 6387)

TAdapterDisplayColumn (see page 6364)

TAdapterActionButton (see page 6316)

62.1.20.1 TCustomAdapterCommandColumn Members

The following tables list the members exposed by TCustomAdapterCommandColumn.

Protected Methods

	Name	Description
	FormatColumnData (see page 6445)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 6446)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 6446)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
V	GetLayoutAttributes (see page 6446)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6447)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6447)	Provides the underlying implementation for the Content method.
	NotifyAdapterChange (see page 6448)	Responds to changes that affect the associated adapter.

Protected Properties

	Name	Description
R	AdapterActionsListHelper (see page 6449)	Identifies the helper object that enables this component to work with the editor at design time.

Public Methods

	Name	Description
V	Create (see page 6444)	Creates and initializes an instance of TCustomAdapterCommandColumn (see page 6442).
V	Destroy (see page 6445)	Frees an instance of TCustomAdapterCommandColumn (see page 6442).
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6449)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6450)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6450)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6451)	Specifies the attributes of the Caption (see page 6450) property.
	Custom (see page 6451)	Represents a tag option and its value.
	DisplayColumns (see page 6451)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6452)	Adds style attributes to the generated caption.
	StyleRule (see page 6452)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6453)	Specifies the vertical alignment of the action control buttons within the column.
R	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
R	VisibleCommands (see page 6453)	Lists the child Web items that generate controls that appear in cells of the column.

62.1.20.2 TCustomAdapterCommandColumn.Create Constructor

Creates and initializes an instance of TCustomAdapterCommandColumn (see page 6442).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterCommandColumn(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterCommandColumn (see page 6442) at runtime. TCustomAdapterCommandColumn (see page 6442) descendants added at design time are instantiated automatically.

AOwner is the component that becomes the value of the Owner property. Typically, this is the Web module.

See Also

Owner

62.1.20.3 TCustomAdapterCommandColumn.Destroy Destructor

Frees an instance of TCustomAdapterCommandColumn (see page 6442).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterCommandColumn();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks that the TCustomAdapterCommandColumn (see page 6442) reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

62.1.20.4 TCustomAdapterCommandColumn Methods

The methods of the TCustomAdapterCommandColumn class are listed here.

Protected Methods

	Name	Description
	FormatColumnData (see page 6445)	Generates the HTML for the cells in the column that contain controls to execute actions.
	FormatColumnHeading (see page 6446)	Generates the HTML for the table header cell for this column.
	GetAdapterDisplayCharacteristics (see page 6446)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.
V	GetLayoutAttributes (see page 6446)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6447)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6447)	Provides the underlying implementation for the Content method.
	NotifyAdapterChange (see page 6448)	Responds to changes that affect the associated adapter.

62.1.20.4.1 TCustomAdapterCommandColumn.FormatColumnData Method

Generates the HTML for the cells in the column that contain controls to execute actions.

Pascal

```
function FormatColumnData(const Content: string; Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnData(const UnicodeString Content, TWebContentOptions
Options);
```

Description

The adapter grid calls FormatColumnData to update the HTML for the contents of a cell to include any column attributes that affect the layout of that content. This method adds attributes to the HTML that reflect the Align (see page 6450) and VAlign (see page 6453) properties.

Content is the HTML that represents the content of the cell. It is the value that the Content method returns.

Options contains a set of flags that limit what can be included in the generated HTML.

FormatColumnData returns the resulting HTML.

See Also

Content

TAdapterGrid (see page 6419)

Align (see page 6450)

VAlign (see page 6453)

62.1.20.4.2 TCustomAdapterCommandColumn.FormatColumnHeading Method

Generates the HTML for the table header cell for this column.

Pascal

```
function FormatColumnHeading(Options: TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatColumnHeading(TWebContentOptions Options);
```

Description

The adapter grid calls FormatColumnHeading to generate the HTML for the top cell of the column (the one that contains the caption). This method generates the HTML to reflect the Caption (see page 6450) and CaptionAttributes (see page 6451) properties.

Options contains a set of flags that limit what can be included in the generated HTML.

FormatColumnHeading returns the resulting HTML.

See Also

Caption (see page 6450)

TAdapterGrid (see page 6419)

CaptionAttributes (see page 6451)

62.1.20.4.3 TCustomAdapterCommandColumn.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterCommandColumn.

62.1.20.4.4 TCustomAdapterCommandColumn.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

GetLayoutAttributes adds returns a TLayoutAttributes (see page 6011) object that encodes the values of the Style (see page 6452), StyleRule (see page 6452), and Custom (see page 6451) properties.

See Also

Content

Style (see page 6452)

StyleRule (see page 6452)

Custom (see page 6451)

62.1.20.4.5 TCustomAdapterCommandColumn.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TCustomAdapterCommandColumn (see page 6442) descendant can add a component of type AClass to its WebFieldControls property list.

AParent is the TCustomAdapterCommandColumn (see page 6442) instance to which the component will be added.

AClass is the class of the component to be added.

In TCustomAdapterCommandColumn (see page 6442), ImplCanAddClass returns true if AClass is a descendant of TCustomAdapterActionButton (see page 6430) or TCustomLayoutGroup.

See Also

CanAddClass

WebFieldControls

TCustomAdapterActionButton (see page 6430)

TCustomLayoutGroup

62.1.20.4.6 TCustomAdapterCommandColumn.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML for the controls in one cell of the column. ImplContent generates the HTML to display the controls that the VisibleCommands (see page 6453) property lists, reflecting the values of the DisplayColumns (see page 6451), .Style (see page 6452), StyleRule (see page 6452), and Custom (see page 6451) properties.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TCustomAdapterCommandColumn (see page 6442) should format the HTML controls it generates. It implements the ILayoutWebContent (see page 6084) interface.

ImplContent returns the generated HTML.

Note: ImplContent does not include attributes for the alignment of controls inside the cell. The FormatColumnData (see page 6445) method adds this information to the HTML that ImplContent generates.

See Also

Content

FormatColumnData (see page 6445)

ILayoutWebContent (see page 6084)

VisibleCommands (see page 6453)

DisplayColumns (see page 6451)

Style (see page 6452)

StyleRule (see page 6452)

Custom (see page 6451)

62.1.20.4.7 TCustomAdapterCommandColumn.NotifyAdapterChange Method

Responds to changes that affect the associated adapter.

Pascal

```
procedure NotifyAdapterChange(Sender: TComponent);
```

C++

```
__fastcall NotifyAdapterChange(TComponent * Sender);
```

Description

NotifyAdapterChange is called automatically when the associated adapter changes. This method updates the set of default actions for that adapter to reflect the current change, and passes on the change notification to the actions that the WebFieldControls property lists.

See Also

GetDefaultWebComponents

WebFieldControls

62.1.20.5 TCustomAdapterCommandColumn Properties

The properties of the TCustomAdapterCommandColumn class are listed here.

Protected Properties

	Name	Description
	AdapterActionsListHelper (see page 6449)	Identifies the helper object that enables this component to work with the editor at design time.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6449)	Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.
	Align (see page 6450)	Specifies the horizontal alignment of the action control buttons within the column.
	Caption (see page 6450)	Specifies the text label that appears at the top of the column.
	CaptionAttributes (see page 6451)	Specifies the attributes of the Caption (see page 6450) property.
	Custom (see page 6451)	Represents a tag option and its value.
	DisplayColumns (see page 6451)	Specifies the number of adapter action buttons that can appear in a single row.
	Style (see page 6452)	Adds style attributes to the generated caption.
	StyleRule (see page 6452)	Applies a style in the page producer's style sheet to the generated column.
	VAlign (see page 6453)	Specifies the vertical alignment of the action control buttons within the column.
	VisibleCommands (see page 6453)	Lists the child Web items that generate controls that appear in cells of the column.

62.1.20.5.1 TCustomAdapterCommandColumn.AdapterActionsListHelper Property

Identifies the helper object that enables this component to work with the editor at design time.

Pascal

```
property AdapterActionsListHelper: TAdapterActionsListHelper;
```

C++

```
__property TAdapterActionsListHelper AdapterActionsListHelper;
```

Description

AdapterActionsListHelper provides read-only access to a helper object that implements design-time editing functions on behalf of this component. This helper object implements methods for locating child Web items for specified actions, for listing the child Web items of this component, and for specifying the text of menu commands that operate on this component's list of adapter action buttons.

See Also

WebFieldControls

62.1.20.5.2 TCustomAdapterCommandColumn.AddDefaultCommands Property

Indicates whether cells in the column contain controls for all of the adapter's default actions if you do not explicitly add items to the WebFieldControls property.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

Set AddDefaultCommands property to specify what should appear in the generated column when you do not explicitly add any adapter action buttons to the WebFieldControls property. When AddDefaultCommands is true (the default), the adapter command column automatically adds controls for every default action of the associated adapter if you do not explicitly set the WebFieldControls property. When AddDefaultCommands is false, the adapter command column only generates controls if you explicitly add them to the WebFieldControls property. AddDefaultCommands must be false to allow an empty column.

See Also

WebFieldControls

DefaultWebComponents (see page 5237)

VisibleCommands (see page 6453)

62.1.20.5.3 TCustomAdapterCommandColumn.Align Property

Specifies the horizontal alignment of the action control buttons within the column.

Pascal

```
property Align: THTMLAlign;
```

C++

```
__property THTMLAlign Align;
```

Description

Align controls where the child HTML control are positioned horizontally within this column. Align indicates the desired value of the Align option of the HTML tag.

The default value of Align is haDefault, which means the web browser that reads the HTML document horizontally positions the control wherever it wishes.

See Also

VAlign (see page 6453)

DisplayColumns (see page 6451)

62.1.20.5.4 TCustomAdapterCommandColumn.Caption Property

Specifies the text label that appears at the top of the column.

Pascal

```
property Caption: string;
```

C++

```
__property UnicodeString Caption;
```

Description

Use Caption to get or set the caption for the column in the adapter grid. The caption is the text label that appears at the top of the column.

To change the appearance or layout of the caption, use the CaptionAttribute property.

See Also

CaptionAttributes (see page 6451)

62.1.20.5.5 TCustomAdapterCommandColumn.CaptionAttributes Property

Specifies the attributes of the Caption (see page 6450) property.

Pascal

```
property CaptionAttributes: TCaptionAttributes;
```

C++

```
__property TCaptionAttributes CaptionAttributes;
```

Description

Use CaptionAttributes to customize the appearance of the column caption.

See Also

Caption (see page 6450)

62.1.20.5.6 TCustomAdapterCommandColumn.Custom Property

Represents a tag option and its value.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to add one or more options to the tag that surrounds the column. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces.

TCustomAdapterCommandColumn (see page 6442) inserts the Custom string into the HTML tag when it generates its content.

See Also

Style (see page 6452)

StyleRule (see page 6452)

62.1.20.5.7 TCustomAdapterCommandColumn.DisplayColumns Property

Specifies the number of adapter action buttons that can appear in a single row.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Set DisplayColumns to organize the layout of adapter action buttons when the WebFieldControls property lists more than one

adapter action button. TCustomAdapterCommandColumn (see page 6442) places the adapter action buttons by rows in the cells of the adapter grid column. DisplayColumns limits the number of controls that can appear in a single row within each column cell. If the WebFieldControls property lists more adapter action buttons than DisplayColumns, then the cell contains multiple rows of controls.

See Also

WebFieldControls

62.1.20.5.8 TCustomAdapterCommandColumn.Style Property

Adds style attributes to the generated caption.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to specify display characteristics that affect the generated column. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the HTML tag that generates the column as the value of a STYLE option.

Note: Style affects the contents of the column cells. To specify display characteristics of the column's caption, use the CaptionAttributes (see page 6451) property instead.

See Also

Custom (see page 6451)

StyleRule (see page 6452)

CaptionAttributes (see page 6451)

62.1.20.5.9 TCustomAdapterCommandColumn.StyleRule Property

Applies a style in the page producer's style sheet to the generated column.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated column. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

To apply a style definition that is not included in the style sheet, use the Style (see page 6452) property instead.

See Also

Custom (see page 6451)

Style (see page 6452)

Styles (see page 1157)

StylesFile (see page 1158)

62.1.20.5.10 TCustomAdapterCommandColumn.VAlign Property

Specifies the vertical alignment of the action control buttons within the column.

Pascal

```
property VAlign: THTMLVAlign;
```

C++

```
__property THTMLVAlign VAlign;
```

Description

VAlign controls where the child HTML control are positioned vertically within the cells of this column. VAlign indicates the desired value of the VAlign option of the HTML tag.

The default value of VAlign is haVDefault, which means the web browser that reads the HTML document vertically positions the control wherever it wishes.

See Also

Align (see page 6450)

DisplayColumns (see page 6451)

62.1.20.5.11 TCustomAdapterCommandColumn.VisibleCommands Property

Lists the child Web items that generate controls that appear in cells of the column.

Pascal

```
property VisibleCommands: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleCommands;
```

Description

VisibleCommands is the list of child Web items that generate HTML controls for the cells of the adapter command column. Each element in this list is an adapter action button (TAdapterActionButton (see page 6316) object).

If you have explicitly added child Web items to this component, VisibleCommands is the same as the value of the WebFieldControls property. Otherwise, if the AddDefaultCommands (see page 6449) property is true, VisibleCommands lists adapter action buttons for the default actions of the associated adapter. If AddDefaultCommands (see page 6449) is false, VisibleCommands is an empty list.

See Also

WebFieldControls

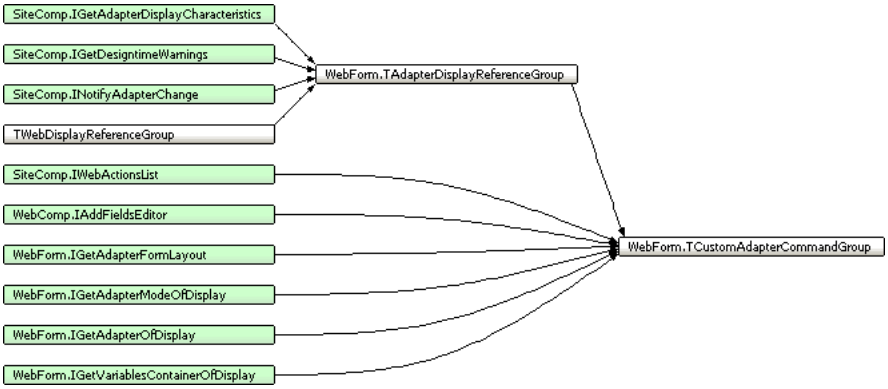
AddDefaultCommands (see page 6449)

TAdapterActionButton (see page 6316)

62.1.21 TCustomAdapterCommandGroup Class

TCustomAdapterCommandGroup is the base class for TAdapterCommandGroup (see page 6337).

Class Hierarchy



Pascal

```
TCustomAdapterCommandGroup = class(TAdapterDisplayReferenceGroup, IWebActionsList,
IGetAdapterOfDisplay, IGetVariablesContainerOfDisplay, IAddFieldsEditor,
IGetAdapterModeOfDisplay, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterCommandGroup : public TAdapterDisplayReferenceGroup, public
IWebActionsList, public IGetAdapterOfDisplay, public IGetVariablesContainerOfDisplay, public
IAddFieldsEditor, public IGetAdapterModeOfDisplay, public IGetAdapterFormLayout;
```

File

WebForm

Description

Derive from TCustomAdapterCommandGroup to define a component that generates a group of HTML buttons or other HTML elements that execute adapter actions.

See Also

TAdapterDisplayReferenceGroup (see page 6381)

TAdapterCommandGroup (see page 6337)

TAdapterForm (see page 6404)

TAdapterFieldGroup (see page 6399)

TAdapterGrid (see page 6419)

TAdapterActionButton (see page 6316)

62.1.21.1 TCustomAdapterCommandGroup Members

The following tables list the members exposed by TCustomAdapterCommandGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6383)	Responds to changes in the adapter on which the controls in this group operate.
	DisplayComponentChange (see page 6384)	Responds when the DisplayComponent (see page 6386) property changes.
	GetAdapter (see page 6457)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterDisplayCharacteristics (see page 6384)	Indicates the display characteristics of the HTML this component generates.
	GetAdapterFormLayout (see page 6457)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 6457)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 6457)	Returns the adapter mode defined by the TAdapterPageProducer components.
	GetDesigntimeWarnings (see page 6384)	Populates a design-time warnings object with any warnings that the reference group needs to emit.
	ImplCanAddClass (see page 6458)	Checks the inheritance of the given component.
	ImplContent (see page 6458)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 6408) component.
	NotifyAdapterChange (see page 6385)	Responds when other components notify this one of changes to the adapter.
	OnDisplayComponentChange (see page 6385)	Responds when the associated display component changes.
	SetWebDisplayComponent (see page 6386)	Sets the value of the DisplayComponent (see page 6386) property.

Protected Properties

	Name	Description
	Adapter (see page 6459)	Provides read-only access to the Adapter that this command group uses.
	AdapterActionsListHelper (see page 6459)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Public Methods

	Name	Description
	Create (see page 6456)	Creator for TCustomAdapterCommandGroup (see page 6454).
	Destroy (see page 6456)	Calls the base class version, and then frees the memory used by the TAdapterActionsListHelper (see page 6324) component.
	FindAction (see page 1077)	This is FindAction, a member of class IWebActionsList.
	GetActionsList (see page 1077)	This is GetActionsList, a member of class IWebActionsList.
	GetAdapterDisplayCharacteristics (see page 1019)	This is GetAdapterDisplayCharacteristics, a member of class IGetAdapterDisplayCharacteristics.
	GetAddAllFieldsItem (see page 6080)	This is GetAddAllFieldsItem, a member of class IAddFieldsEditor.
	GetAddFieldsDlgCaption (see page 6080)	This is GetAddFieldsDlgCaption, a member of class IAddFieldsEditor.
	GetAddFieldsItem (see page 6080)	This is GetAddFieldsItem, a member of class IAddFieldsEditor.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetNewFieldPrefix (see page 6081)	This is GetNewFieldPrefix, a member of class IAddFieldsEditor.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleActions (see page 1077)	This is GetVisibleActions, a member of class IWebActionsList.
	IsActionInUse (see page 1078)	This is IsActionInUse, a member of class IWebActionsList.
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6459)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 6459)	Provides read and write access to the custom HTML element attributes, such as VAlign="top."
	DisplayComponent (see page 6386)	Associates the group with a component that generates the display controls.
	Style (see page 6460)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6460)	Provides read and write access to the internal string that stores the CSS style rule for the control group.

	VisibleActions (see page 1078)	This is VisibleActions, a member of class IWebActionsList.
	VisibleCommands (see page 6460)	Provides read-only access to the TWebComponentList (see page 6025) that stores the list of visible components.

62.1.21.2 TCustomAdapterCommandGroup.Create Constructor

Creator for TCustomAdapterCommandGroup (see page 6454).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterCommandGroup(TComponent * AOwner);
```

Description

This constructor calls the base class version, and then creates a TAdapterActionsListHelper (see page 6324) component and sets the AddDefaultActions property to true by default.

See Also

Create (see page 6382)

AddDefaultActions (see page 6331)

62.1.21.3 TCustomAdapterCommandGroup.Destroy Destructor

Calls the base class version, and then frees the memory used by the TAdapterActionsListHelper (see page 6324) component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterCommandGroup();
```

Description

This destructor calls the base class version, and then frees the memory used by the TAdapterActionsListHelper (see page 6324) component. Do not call this destructor directly; instead, use the Free method

See Also

Destroy (see page 6382)

62.1.21.4 TCustomAdapterCommandGroup Methods

The methods of the TCustomAdapterCommandGroup class are listed here.

Protected Methods

	Name	Description
	GetAdapter (see page 6457)	This is GetAdapter, a member of class TCustomAdapterCommandGroup.
	GetAdapterFormLayout (see page 6457)	This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.
	GetAdapterModeOfAdapter (see page 6457)	Returns the adapter mode defined by the adapter component.
	GetAdapterModeOfDisplay (see page 6457)	Returns the adapter mode defined by the TAdapterPageProducer components.
	ImplCanAddClass (see page 6458)	Checks the inheritance of the given component.

	ImplContent (see page 6458)	Lays out the buttons for the command group using the TAdapterFormLayout (see page 6408) component.

62.1.21.4.1 TCustomAdapterCommandGroup.GetAdapter Method

Pascal

```
function GetAdapter: TComponent;
```

C++

```
__fastcall TComponent * GetAdapter();
```

Description

This is GetAdapter, a member of class TCustomAdapterCommandGroup.

62.1.21.4.2 TCustomAdapterCommandGroup.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterCommandGroup.

62.1.21.4.3 TCustomAdapterCommandGroup.GetAdapterModeOfAdapter Method

Returns the adapter mode defined by the adapter component.

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

The GetAdapterModeOfAdapter function returns the adapter mode defined by the adapter component. Some adapters support a mode. For example, the TDataSetAdapter supports Edit, Insert (see page 3601), and Delete (see page 3552) modes. The mode can be set by TAdapterPageProducer components or during adapter action execution. The mode defined by adapter action takes precedence over the mode defined by TAdapterPageProducer components.

62.1.21.4.4 TCustomAdapterCommandGroup.GetAdapterModeOfDisplay Method

Returns the adapter mode defined by the TAdapterPageProducer components.

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfDisplay();
```

Description

This function returns the adapter mode defined by the TAdapterPageProducer components. Some adapters support a mode. For example, the TDataSetAdapter supports Edit, Insert (see page 3601), and Delete (see page 3552) modes. The mode can be set by TAdapterPageProducer components or during adapter action execution. The mode defined by adapter action takes precedence over the mode defined by TAdapterPageProducer components.

62.1.21.4.5 TCustomAdapterCommandGroup.ImplCanAddClass Method

Checks the inheritance of the given component.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The ImplCanAddClass checks the inheritance of the given component. If it is derived from either TCustomAdapterActionButton (see page 6430) or TCustomLayoutGroup, it will return True to indicate that the class can be added to this TCustomAdapterCommandGroup (see page 6454).

62.1.21.4.6 TCustomAdapterCommandGroup.ImplContent Method

Lays out the buttons for the command group using the TAdapterFormLayout (see page 6408) component.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The ImplContent function lays out the buttons for the command group using the TAdapterFormLayout (see page 6408) component. It loops through all of the visible child components, calling them to render their HTML content.

62.1.21.5 TCustomAdapterCommandGroup Properties

The properties of the TCustomAdapterCommandGroup class are listed here.

Protected Properties

	Name	Description
	Adapter (see page 6459)	Provides read-only access to the Adapter that this command group uses.
	AdapterActionsListHelper (see page 6459)	Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Public Properties

	Name	Description
	AddDefaultCommands (see page 6459)	Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.
	Custom (see page 6459)	Provides read and write access to the custom HTML element attributes, such as VAlign="top."

	Style (see page 6460)	Provides read and write access to the internal string that stores the CSS style, such as making the background color red.
	StyleRule (see page 6460)	Provides read and write access to the internal string that stores the CSS style rule for the control group.
R	VisibleCommands (see page 6460)	Provides read-only access to the TWebComponentList (see page 6025) that stores the list of visible components.

62.1.21.5.1 TCustomAdapterCommandGroup.Adapter Property

Provides read-only access to the Adapter that this command group uses.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

The Adapter property provides read-only access to the Adapter that this command group uses.

62.1.21.5.2 TCustomAdapterCommandGroup.AdapterActionsListHelper Property

Provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

Pascal

```
property AdapterActionsListHelper: TAdapterActionsListHelper;
```

C++

```
__property TAdapterActionsListHelper AdapterActionsListHelper;
```

Description

The AdapterActionsListHelper provides read-only access to a helper object that implements design-time editing functions on behalf of this component.

62.1.21.5.3 TCustomAdapterCommandGroup.AddDefaultCommands Property

Provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

Pascal

```
property AddDefaultCommands: Boolean;
```

C++

```
__property Boolean AddDefaultCommands;
```

Description

The AddDefaultCommands property provides read and write access to the internal Boolean flag that determines if the default commands should be added automatically.

62.1.21.5.4 TCustomAdapterCommandGroup.Custom Property

Provides read and write access to the custom HTML element attributes, such as VAlign="top."

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

The Custom property provides read and write access to the custom HTML element attributes, such as VAlign="top." The content of this string will be written as it is when generating HTML.

62.1.21.5.5 TCustomAdapterCommandGroup.Style Property

Provides read and write access to the internal string that stores the CSS style, such as making the background color red.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

The Style property provides read and write access to the internal string that stores the CSS style, such as making the background color red.

62.1.21.5.6 TCustomAdapterCommandGroup.StyleRule Property

Provides read and write access to the internal string that stores the CSS style rule for the control group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

The StyleRule property provides read and write access to the internal string that stores the CSS style rule for the control group. This property is used to reference a CSS style defined in TAdapterPageProducer. ::Styles.

62.1.21.5.7 TCustomAdapterCommandGroup.VisibleCommands Property

Provides read-only access to the TWebComponentList (see page 6025) that stores the list of visible components.

Pascal

```
property VisibleCommands: TWebComponentList;
```

C++

```
__property TWebComponentList VisibleCommands;
```

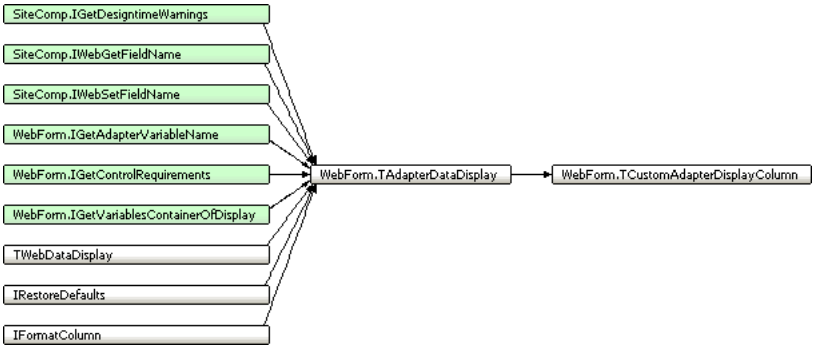
Description

The VisibleCommands property provides read-only access to the TWebComponentList (see page 6025) that stores the list of visible components.

62.1.22 TCustomAdapterDisplayColumn Class

TCustomAdapterDisplayColumn is the base class for the grid display of adapter field data.

Class Hierarchy



Pascal

```
TCustomAdapterDisplayColumn = class(TAdapterDataDisplay);
```

C++

```
class TCustomAdapterDisplayColumn : public TAdapterDataDisplay;
```

File

WebForm

Description

TCustomAdapterDisplayColumn is the base class for grid components that display adapter field data. Do not directly instantiate TCustomAdapterDisplayColumn. Use TAdapterDisplayColumn (see page 6364) or create your own concrete descendants of TCustomAdapterDisplayColumn.

See Also

IWebContent (see page 6103)

TAdapterGrid (see page 6419)

TAdapterFieldGroup (see page 6399)

TAdapterCommandColumn (see page 6331)

TAdapterCommandGroup (see page 6337)

TAdapterDisplayColumn (see page 6364)

62.1.22.1 TCustomAdapterDisplayColumn Members

The following tables list the members exposed by TCustomAdapterDisplayColumn.

Protected Methods

	Name	Description
	CanAddClass (see page 6463)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
	ControlContent (see page 6463)	Returns the HTML that defines the display control (without its caption).

	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	FormatImage (see page 6464)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 6464)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 6464)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplCanAddClass (see page 6465)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
	ImplContent (see page 6360)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6465)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
	ImplGetControlRequirements (see page 6465)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
	ImplRestoreDefaults (see page 6465)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
	SetCaption (see page 6362)	Sets the value of the Caption property.
	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
	Create (see page 6463)	Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 6461).
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	IsColumn (see page 6466)	Indicates whether the component generates the controls that appear in the column of a data grid.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6362)	Specifies the horizontal alignment of the component relative to its containing cell.
	DisplayType (see page 6466)	Sets the display type for the column.
	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 6363)	Specifies the name of the associated dataset field.
	HideOptions (see page 6363)	When field should be hidden.
	ImageHeight (see page 6466)	Specifies the height of the image in the column.
	ImageWidth (see page 6467)	Specifies the width of the image in the column.
	VAlign (see page 6363)	Specifies the vertical alignment of the generated display control.

62.1.22.2 TCustomAdapterDisplayColumn.Create Constructor

Creates and initializes an instance of TCustomAdapterDisplayColumn (see page 6461).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterDisplayColumn(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterDisplayColumn (see page 6461) at runtime. TCustomAdapterDisplayColumn (see page 6461) objects added at design time using the Web Page editor are instantiated automatically.

62.1.22.3 TCustomAdapterDisplayColumn Methods

The methods of the TCustomAdapterDisplayColumn class are listed here.

Protected Methods

	Name	Description
	CanAddClass (see page 6463)	This is CanAddClass, a member of class TCustomAdapterDisplayColumn.
V	ControlContent (see page 6463)	Returns the HTML that defines the display control (without its caption).
	FormatImage (see page 6464)	This is FormatImage, a member of class TCustomAdapterDisplayColumn.
	FormatListDisplay (see page 6464)	This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.
	FormatTextDisplay (see page 6464)	This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.
V	ImplCanAddClass (see page 6465)	This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.
V	ImplFormatColumnHeading (see page 6465)	This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.
V	ImplGetControlRequirements (see page 6465)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.
V	ImplRestoreDefaults (see page 6465)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.

Public Methods

	Name	Description
V	IsColumn (see page 6466)	Indicates whether the component generates the controls that appear in the column of a data grid.

62.1.22.3.1 TCustomAdapterDisplayColumn.CanAddClass Method

Pascal

```
function CanAddClass(AParent: TComponent; AClass: TClass): Boolean;
```

C++

```
__fastcall Boolean CanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is CanAddClass, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.2 TCustomAdapterDisplayColumn.ControlContent Method

Returns the HTML that defines the display control (without its caption).

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

ControlContent is used internally to generate the HTML for the display control. This method does not generate any HTML for the control's label, which is handled separately by the FormatCaption method.

TWebDataDisplay introduces ControlContent as an abstract (pure virtual) method. Descendant classes override this method to generate the appropriate type of display component.

Options contains a set of flags that limit what can be included in the generated HTML.

ControlContent returns a string of HTML defining the display control.

See Also

ImplContent

EventContent

FormatCaption

62.1.22.3.3 TCustomAdapterDisplayColumn.FormatImage Method

Pascal

```
function FormatImage(const VarName: string; const HTMLName: string; Options:
TWebContentOptions): string;
```

C++

```
__fastcall UnicodeString FormatImage(const UnicodeString VarName, const UnicodeString
HTMLName, TWebContentOptions Options);
```

Description

This is FormatImage, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.4 TCustomAdapterDisplayColumn.FormatListDisplay Method

Pascal

```
function FormatListDisplay(const VarName: string; const HTMLName: string): string;
```

C++

```
__fastcall UnicodeString FormatListDisplay(const UnicodeString VarName, const UnicodeString
HTMLName);
```

Description

This is FormatListDisplay, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.5 TCustomAdapterDisplayColumn.FormatTextDisplay Method

Pascal

```
function FormatTextDisplay(const VarName: string; const HTMLName: string): string;
```

C++

```
__fastcall UnicodeString FormatTextDisplay(const UnicodeString VarName, const UnicodeString HTMLName);
```

Description

This is FormatTextDisplay, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.6 TCustomAdapterDisplayColumn.ImplCanAddClass Method

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

This is ImplCanAddClass, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.7 TCustomAdapterDisplayColumn.ImplFormatColumnHeading Method

Pascal

```
function ImplFormatColumnHeading(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplFormatColumnHeading(TWebContentOptions Options);
```

Description

This is ImplFormatColumnHeading, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.8 TCustomAdapterDisplayColumn.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); override;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.9 TCustomAdapterDisplayColumn.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; override;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayColumn.

62.1.22.3.10 TCustomAdapterDisplayColumn.IsColumn Method

Indicates whether the component generates the controls that appear in the column of a data grid.

Pascal

```
class function IsColumn: Boolean; override;
```

C++

```
virtual __fastcall Boolean IsColumn();
```

Description

Call IsColumn to determine whether a TCustomAdapterDisplayColumn (see page 6461) object generates the controls that appear in a single column of a data grid. As implemented in TCustomAdapterDisplayColumn (see page 6461), IsColumn always returns true.

See Also

IsColumn

62.1.22.4 TCustomAdapterDisplayColumn Properties

The properties of the TCustomAdapterDisplayColumn class are listed here.

Public Properties

	Name	Description
	DisplayType (see page 6466)	Sets the display type for the column.
	ImageHeight (see page 6466)	Specifies the height of the image in the column.
	ImageWidth (see page 6467)	Specifies the width of the image in the column.

62.1.22.4.1 TCustomAdapterDisplayColumn.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TColumnHTMLElementType;
```

C++

```
__property TColumnHTMLElementType DisplayType;
```

Description

Set DisplayType to set the display type for the column to text or image.

See Also

ImageHeight (see page 6466)

ImageWidth (see page 6467)

62.1.22.4.2 TCustomAdapterDisplayColumn.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in pixels.

See Also

ImageWidth (see page 6467)

62.1.22.4.3 TCustomAdapterDisplayColumn.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight (see page 6466) to indicate the width of the image in pixels.

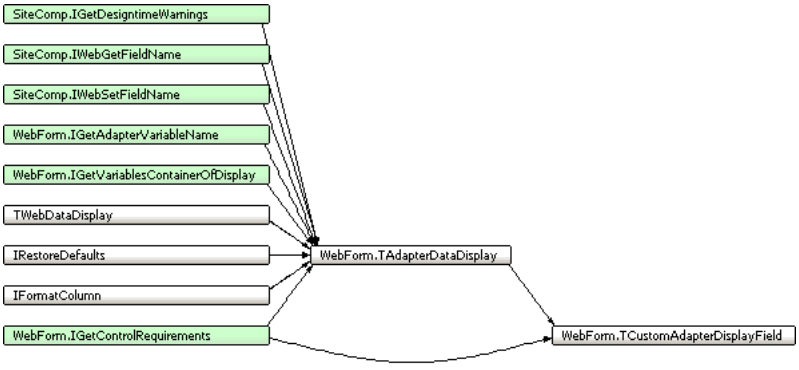
See Also

ImageHeight (see page 6466)

62.1.23 TCustomAdapterDisplayField Class

TCustomAdapterDisplayField is the base class for the free-form display and editing of adapter field data.

Class Hierarchy



Pascal

```
TCustomAdapterDisplayField = class(TAdapterDataDisplay, IGetControlRequirements);
```

C++

```
class TCustomAdapterDisplayField : public TAdapterDataDisplay, public IGetControlRequirements;
```

File

WebForm

Description

TCustomAdapterDisplayField is the base class for free-form components that display and edit adapter field data. Do not directly instantiate TCustomAdapterDisplayField. Use TAdapterDisplayField (see page 6371) or create your own concrete descendants of TCustomAdapterDisplayField.

See Also

TCustomAdapterEditColumn (see page 6475)

TAdapterDisplayField (see page 6371)

IWebContent (see page 6103)

TAdapterGrid (see page 6419)

TAdapterFieldGroup (see page 6399)

62.1.23.1 TCustomAdapterDisplayField Members

The following tables list the members exposed by TCustomAdapterDisplayField.

Protected Methods

	Name	Description
	AddAttributes (see page 6470)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6470)	Returns the HTML that defines the display control (without its caption).
V	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.
V	GetLayoutAttributes (see page 6470)	Returns the value of the LayoutAttributes property.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
V	ImplContent (see page 6471)	Provides the underlying implementation for the Content method.
V	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
V	ImplFormatColumnHeading (see page 6361)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
V	ImplGetControlRequirements (see page 6471)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
V	ImplRestoreDefaults (see page 6471)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
V	SetCaption (see page 6362)	Sets the value of the Caption property.
	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
V	Create (see page 6469)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6467).

	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6362)	Specifies the horizontal alignment of the component relative to its containing cell.
	DisplayColumns (see page 6472)	Specifies the number of columns in a display field.
	DisplayRows (see page 6472)	Specifies the number of rows in a display field.
	DisplayType (see page 6473)	Sets the display type for the column.
	DisplayWidth (see page 6473)	Specifies the width, in characters, of the generated HTML field.
R	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 6363)	Specifies the name of the associated dataset field.
	HideOptions (see page 6363)	When field should be hidden.
	ImageHeight (see page 6473)	Specifies the height of the image in the column.
	ImageWidth (see page 6474)	Specifies the width of the image in the column.
	InputType (see page 6474)	Specifies the input type for the display field.
	MaxLength (see page 6474)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6475)	Indicates the number of rows that can be selected.
	TextAreaWrap (see page 6475)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6363)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6475)	Indicates the viewing state of the adapter field.

62.1.23.2 TCustomAdapterDisplayField.Create Constructor

Creates and initializes an instance of TCustomAdapterDisplayField (see page 6467).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterDisplayField(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterDisplayField (see page 6467) at runtime. TCustomAdapterDisplayField (see page 6467) descendants added at design time using the Web Page editor are instantiated automatically.

62.1.23.3 TCustomAdapterDisplayField Methods

The methods of the TCustomAdapterDisplayField class are listed here.

Protected Methods

	Name	Description
	AddAttributes (see page 6470)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6470)	Returns the HTML that defines the display control (without its caption).
V	GetLayoutAttributes (see page 6470)	Returns the value of the LayoutAttributes property.

	ImplContent (see page 6471)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6471)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
	ImplRestoreDefaults (see page 6471)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.

62.1.23.3.1 TCustomAdapterDisplayField.AddAttributes Method

Pascal

```
procedure AddAttributes(var Attrs: string);
```

C++

```
__fastcall AddAttributes(UnicodeString Attrs);
```

Description

This is AddAttributes, a member of class TCustomAdapterDisplayField.

62.1.23.3.2 TCustomAdapterDisplayField.ControlContent Method

Returns the HTML that defines the display control (without its caption).

Pascal

```
function ControlContent(Options: TWebContentOptions): string; override;
```

C++

```
virtual __fastcall UnicodeString ControlContent(TWebContentOptions Options);
```

Description

ControlContent is used internally to generate the HTML for the display control. This method does not generate any HTML for the control's label, which is handled separately by the FormatCaption method.

TWebDataDisplay introduces ControlContent as an abstract (pure virtual) method. Descendant classes override this method to generate the appropriate type of display component.

Options contains a set of flags that limit what can be included in the generated HTML.

ControlContent returns a string of HTML defining the display control.

See Also

ImplContent

EventContent

FormatCaption

62.1.23.3.3 TCustomAdapterDisplayField.GetLayoutAttributes Method

Returns the value of the LayoutAttributes property.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the

Content method) to take into account layout properties of the child component.

GetLayoutAttributes returns the value of the LayoutAttributes property.

See Also

Content

LayoutAttributes

62.1.23.3.4 TCustomAdapterDisplayField.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content

Caption

ILayoutWebContent (see page 6084)

62.1.23.3.5 TCustomAdapterDisplayField.ImplGetControlRequirements Method

Pascal

```
procedure ImplGetControlRequirements(ARequirements: TAbstractControlRequirements); override;
```

C++

```
virtual __fastcall ImplGetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.

62.1.23.3.6 TCustomAdapterDisplayField.ImplRestoreDefaults Method

Pascal

```
procedure ImplRestoreDefaults; override;
```

C++

```
virtual __fastcall ImplRestoreDefaults();
```

Description

This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.

62.1.23.4 TCustomAdapterDisplayField Properties

The properties of the TCustomAdapterDisplayField class are listed here.

Public Properties

	Name	Description
	DisplayColumns (see page 6472)	Specifies the number of columns in a display field.
	DisplayRows (see page 6472)	Specifies the number of rows in a display field.
	DisplayType (see page 6473)	Sets the display type for the column.
	DisplayWidth (see page 6473)	Specifies the width, in characters, of the generated HTML field.
	ImageHeight (see page 6473)	Specifies the height of the image in the column.
	ImageWidth (see page 6474)	Specifies the width of the image in the column.
	InputType (see page 6474)	Specifies the input type for the display field.
	MaxLength (see page 6474)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6475)	Indicates the number of rows that can be selected.
	TextAreaWrap (see page 6475)	Specifies how the display field treats lines that are longer than the display width.
	ViewMode (see page 6475)	Indicates the viewing state of the adapter field.

62.1.23.4.1 TCustomAdapterDisplayField.DisplayColumns Property

Specifies the number of columns in a display field.

Pascal

```
property DisplayColumns: Integer;
```

C++

```
__property int DisplayColumns;
```

Description

Use DisplayColumns to control the number of columns being displayed.

See Also

DisplayRows (see page 6472)

62.1.23.4.2 TCustomAdapterDisplayField.DisplayRows Property

Specifies the number of rows in a display field.

Pascal

```
property DisplayRows: Integer;
```

C++

```
__property int DisplayRows;
```

Description

Use DisplayColumns (see page 6472) to control the number of rows being displayed.

See Also

DisplayColumns (see page 6472)

62.1.23.4.3 TCustomAdapterDisplayField.DisplayType Property

Sets the display type for the column.

Pascal

```
property DisplayType: TDisplayFieldHTMLElementType;
```

C++

```
__property TDisplayFieldHTMLElementType DisplayType;
```

Description

Set DisplayType to indicate the type of HTML control that should be used to display this field's value when it is read-only.

See Also

ImageHeight (see page 6473)

ImageWidth (see page 6474)

InputType (see page 6474)

62.1.23.4.4 TCustomAdapterDisplayField.DisplayWidth Property

Specifies the width, in characters, of the generated HTML field.

Pascal

```
property DisplayWidth: Integer;
```

C++

```
__property int DisplayWidth;
```

Description

Use DisplayWidth to control the width of the HTML field that displays record status information. By default, DisplayWidth is 1 because the TCustomAdapterDisplayField (see page 6467) control values are one character (M for modified or I for inserted).

62.1.23.4.5 TCustomAdapterDisplayField.ImageHeight Property

Specifies the height of the image in the column.

Pascal

```
property ImageHeight: Integer;
```

C++

```
__property int ImageHeight;
```

Description

Set ImageHeight to indicate the height of the image in the column.

See Also

DisplayWidth (see page 6473)

ImageWidth (see page 6474)

62.1.23.4.6 TCustomAdapterDisplayField.ImageWidth Property

Specifies the width of the image in the column.

Pascal

```
property ImageWidth: Integer;
```

C++

```
__property int ImageWidth;
```

Description

Set ImageHeight (see page 6473) to indicate the width of the image in the column.

See Also

ImageHeight (see page 6473)

62.1.23.4.7 TCustomAdapterDisplayField.InputType Property

Specifies the input type for the display field.

Pascal

```
property InputType: TInputFieldHTMLElementType;
```

C++

```
__property TInputFieldHTMLElementType InputType;
```

Description

Set InputType to indicate the type of control that should be generated to let users assign values to the adapter field.

See Also

ImageHeight (see page 6473)

ImageWidth (see page 6474)

DisplayType (see page 6473)

62.1.23.4.8 TCustomAdapterDisplayField.MaxLength Property

Specifies the maximum number of characters the user can enter into the display field.

Pascal

```
property MaxLength: Integer;
```

C++

```
__property int MaxLength;
```

Description

Use MaxLength to limit the number of characters that can be entered into the display field. A value of 0 indicates that there is no application-defined limit on the length.

Note: Setting MaxLength will not truncate the existing text, it merely prevents the user from adding more text after reaching the limit of MaxLength characters.

62.1.23.4.9 TCustomAdapterDisplayField.SelectRows Property

Indicates the number of rows that can be selected.

Pascal

```
property SelectRows: Integer;
```

C++

```
__property int SelectRows;
```

Description

Use SelectRows to indicate the number of rows that can be selected.

62.1.23.4.10 TCustomAdapterDisplayField.TextAreaWrap Property

Specifies how the display field treats lines that are longer than the display width.

Pascal

```
property TextAreaWrap: TTextAreaWrap;
```

C++

```
__property TTextAreaWrap TextAreaWrap;
```

Description

Use TextAreaWrap to specify how the generated display field formats lines that are longer than the number of characters specified by DisplayWidth (see page 6473).

See Also

DisplayWidth (see page 6473)

62.1.23.4.11 TCustomAdapterDisplayField.ViewMode Property

Indicates the viewing state of the adapter field.

Pascal

```
property ViewMode: TDisplayFieldViewMode;
```

C++

```
__property TDisplayFieldViewMode ViewMode;
```

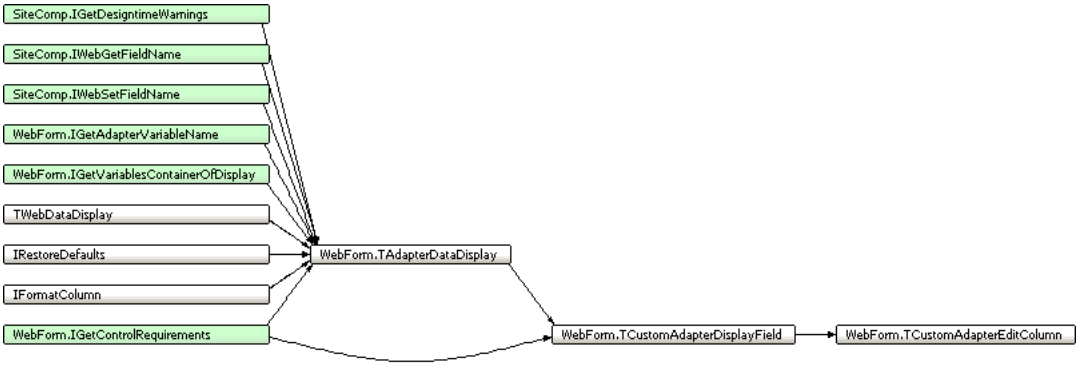
Description

Use ViewMode to specify the viewing state of the field.

62.1.24 TCustomAdapterEditColumn Class

TCustomAdapterEditColumn is the base class for components that display adapter fields as form fields in grids.

Class Hierarchy



Pascal

```
TCustomAdapterEditColumn = class(TCustomAdapterDisplayField);
```

C++

```
class TCustomAdapterEditColumn : public TCustomAdapterDisplayField;
```

File

WebForm

Description

TCustomAdapterEditColumn has the same functionality as TCustomAdapterDisplayField (see page 6467), but is designed to work with grids. Do not directly instantiate TCustomAdapterEditColumn. Use TAdapterEditColumn (see page 6387) or create your own concrete descendants of TCustomAdapterEditColumn.

See Also

TAdapterEditColumn (see page 6387)

62.1.24.1 TCustomAdapterEditColumn Members

The following tables list the members exposed by TCustomAdapterEditColumn.

Protected Methods

	Name	Description
	AddAttributes (see page 6470)	This is AddAttributes, a member of class TCustomAdapterDisplayField.
V	ControlContent (see page 6470)	Returns the HTML that defines the display control (without its caption).
V	FieldConditions (see page 6357)	This is FieldConditions, a member of class TAdapterDataDisplay.
	FindAssociatedField (see page 6357)	This is FindAssociatedField, a member of class TAdapterDataDisplay.
V	FormatCaption (see page 6357)	Returns the HTML for the display control's caption.
	FormatColumnData (see page 6358)	This is FormatColumnData, a member of class TAdapterDataDisplay.
	FormatColumnHeading (see page 6358)	This is FormatColumnHeading, a member of class TAdapterDataDisplay.
	GetAdapterModeOfAdapter (see page 6358)	This is GetAdapterModeOfAdapter, a member of class TAdapterDataDisplay.
	GetAdapterVariableName (see page 6358)	This is GetAdapterVariableName, a member of class TAdapterDataDisplay.
V	GetCaption (see page 6358)	Returns the value of the Caption property.
	GetControlRequirements (see page 6359)	This is GetControlRequirements, a member of class TAdapterDataDisplay.
	GetDesigntimeWarnings (see page 6359)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 6359)	This is GetFieldName, a member of class TAdapterDataDisplay.

	GetLayoutAttributes (see page 6470)	Returns the value of the LayoutAttributes property.
	GetVariablesContainer (see page 6360)	This is GetVariablesContainer, a member of class TAdapterDataDisplay.
	ImplContent (see page 6478)	Provides the underlying implementation for the Content method.
	ImplFormatColumnData (see page 6360)	This is ImplFormatColumnData, a member of class TAdapterDataDisplay.
	ImplFormatColumnHeading (see page 6361)	This is ImplFormatColumnHeading, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6361)	This is ImplGetControlRequirements, a member of class TAdapterDataDisplay.
	ImplGetControlRequirements (see page 6471)	This is ImplGetControlRequirements, a member of class TCustomAdapterDisplayField.
	ImplRestoreDefaults (see page 6361)	This is ImplRestoreDefaults, a member of class TAdapterDataDisplay.
	ImplRestoreDefaults (see page 6471)	This is ImplRestoreDefaults, a member of class TCustomAdapterDisplayField.
	RestoreDefaults (see page 6361)	Returns the Web item to a default state.
	SetCaption (see page 6362)	Sets the value of the Caption property.
	SetFieldName (see page 6362)	This is SetFieldName, a member of class TAdapterDataDisplay.

Public Methods

	Name	Description
	Create (see page 6356)	Creates an instance of TAdapterDataDisplay (see page 6354)
	Create (see page 6469)	Creates and initializes an instance of TCustomAdapterDisplayField (see page 6467).
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldName (see page 1090)	This is GetFieldName, a member of class IWebGetFieldName.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	SetFieldName (see page 1116)	This is SetFieldName, a member of class IWebSetFieldName.

Public Properties

	Name	Description
	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	Align (see page 6362)	Specifies the horizontal alignment of the component relative to its containing cell.
	DisplayColumns (see page 6472)	Specifies the number of columns in a display field.
	DisplayRows (see page 6472)	Specifies the number of rows in a display field.
	DisplayType (see page 6473)	Sets the display type for the column.
	DisplayWidth (see page 6473)	Specifies the width, in characters, of the generated HTML field.
	FieldName (see page 1091)	This is FieldName, a member of class IWebGetFieldName.
	FieldName (see page 6363)	Specifies the name of the associated dataset field.
	HideOptions (see page 6363)	When field should be hidden.
	ImageHeight (see page 6473)	Specifies the height of the image in the column.
	ImageWidth (see page 6474)	Specifies the width of the image in the column.
	InputType (see page 6474)	Specifies the input type for the display field.
	MaxLength (see page 6474)	Specifies the maximum number of characters the user can enter into the display field.
	SelectRows (see page 6475)	Indicates the number of rows that can be selected.
	TextAreaWrap (see page 6475)	Specifies how the display field treats lines that are longer than the display width.
	VAlign (see page 6363)	Specifies the vertical alignment of the generated display control.
	ViewMode (see page 6475)	Indicates the viewing state of the adapter field.

62.1.24.2 TCustomAdapterEditColumn Methods

The methods of the TCustomAdapterEditColumn class are listed here.

Protected Methods

	Name	Description
	ImplContent (see page 6478)	Provides the underlying implementation for the Content method.

62.1.24.2.1 TCustomAdapterEditColumn.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the HTML that defines the display control and its label. ImplContent formats the caption and combines this HTML with the results of the ControlContent method. ControlContent is an abstract (pure virtual) method in TWebDataDisplay. TWebDataDisplay descendants override it to generate the definition of their particular display control.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebDataDisplay should format the HTML control it generates. It implements the ILayoutWebContent interface.

ImplContent returns a string of HTML defining the display control.

See Also

Content

Caption

ILayoutWebContent (see page 6084)

62.1.25 TCustomAdapterErrorList Class

TCustomAdapterErrorList is the base class adapter for components that display error lists.

Class Hierarchy

Pascal

```
TCustomAdapterErrorList = class(TWebContainedComponent, IWebContent, IGetDesigntimeWarnings);
```

C++

```
class TCustomAdapterErrorList : public TWebContainedComponent, public IWebContent, public
IGetDesigntimeWarnings;
```

File

WebForm

Description

TCustomAdapterErrorList is the base class for adapter components that display error lists. Do not instantiate TCustomAdapterErrorList directly. Use TAdapterErrorList (see page 6397), or create your own concrete descendant class.

62.1.25.1 TCustomAdapterErrorList Members

The following tables list the members exposed by TCustomAdapterErrorList.

Protected Methods

	Name	Description
	Content (see page 6480)	Returns the HTML that the Web item generates.
	GetContainerList (see page 6041)	Returns the TWebComponentList (see page 6025) object that the Web parent uses to manage this and its other Web items.
	GetDesigntimeWarnings (see page 6481)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
V	ImplContent (see page 6481)	This is ImplContent, a member of class TCustomAdapterErrorList.
V	IsDefaultField (see page 6042)	Indicates whether this component represents a Web item that is used by default when the parent's set of Web items is not explicitly assigned.
V	Notification (see page 6482)	Forwards notification messages to all owned components.
V	ParentChanged (see page 6042)	Responds when the WebParent (see page 6044) property changes.
V	ReadState (see page 6043)	Reads the component's published properties, stored data, and owned components from a reader object.
V	SetParentComponent (see page 6043)	Sets the WebParent (see page 6044) property to be the same as this component's parent.
	SetWebParent (see page 6043)	Sets the value of the WebParent (see page 6044) property.

Protected Properties

	Name	Description
R	WebContainedComponentHelper (see page 6044)	Provides access to the helper object that implements the IWebComponent (see page 6093) interface for this contained component.
R	WebParent (see page 6044)	Indicates the component that uses this TWebContainedComponent (see page 6038) as one of its Web items.

Public Methods

	Name	Description
V	Create (see page 6480)	Creates an instance of TCustomAdapterErrorList (see page 6478).
V	Destroy (see page 6480)	Frees the memory associated with an instance of TCustomAdapterErrorList (see page 6478).
	GetContainer (see page 6094)	Returns the value of the Container (see page 6096) property.
	GetIndex (see page 6094)	Returns the value of the Index (see page 6097) property.
V	GetParentComponent (see page 6041)	Returns the object that uses this component to generate HTML.
V	HasParent (see page 6041)	Indicates whether this component appears in another component's list of Web items.
	SetComponentList (see page 6095)	Sets the value of the ComponentList (see page 6096) property.
	SetContainer (see page 6095)	Sets the value of the Container (see page 6096) property.
	SetIndex (see page 6095)	Sets the value of the Index (see page 6097) property.

Public Properties

	Name	Description
	Adapter (see page 6482)	Specifies a related adapter.
W	ComponentList (see page 6096)	Indicates the component list that contains the Web item.
	Container (see page 6096)	Specifies the component that uses content generated by this Web item.

	Index (see page 6097)	Indicates where the Web item appears in the Web component list's array of Web items.

62.1.25.2 TCustomAdapterErrorList.Create Constructor

Creates an instance of TCustomAdapterErrorList (see page 6478).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterErrorList(TComponent * AOwner);
```

Description

Call Create to create and initialize an instance of TCustomAdapterErrorList (see page 6478).

See Also

Create

62.1.25.3 TCustomAdapterErrorList.Destroy Destructor

Frees the memory associated with an instance of TCustomAdapterErrorList (see page 6478).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterErrorList();
```

Description

Call Destroy to destroy an instance of TCustomAdapterErrorList (see page 6478) and free the memory associated with it.

62.1.25.4 TCustomAdapterErrorList Methods

The methods of the TCustomAdapterErrorList class are listed here.

Protected Methods

	Name	Description
	Content (see page 6480)	Returns the HTML that the Web item generates.
	GetDesigntimeWarnings (see page 6481)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
V	ImplContent (see page 6481)	This is ImplContent, a member of class TCustomAdapterErrorList.
V	Notification (see page 6482)	Forwards notification messages to all owned components.

62.1.25.4.1 TCustomAdapterErrorList.Content Method

Returns the HTML that the Web item generates.

Pascal

```
function Content(Options: TWebContentOptions; Layout: TLayout): string;
```

C++

```
__fastcall UnicodeString Content(TWebContentOptions Options, TLayout Layout);
```

Description

Call Content to access the HTML produced by the Web item.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the Web item should lay out its constituent parts. This object supports the ILayoutWebContent interface, which the Web item uses to fit its own content into the parent's HTML structures.

Content returns a string of HTML produced by the Web item.

See Also

ILayoutWebContent (see page 6084)

62.1.25.4.2 TCustomAdapterErrorList.GetDesigntimeWarnings Method

Populates a design-time warnings object with any warnings that the dataset values list needs to emit.

Pascal

```
procedure GetDesigntimeWarnings(AWarnings: TAbstractDesigntimeWarnings);
```

C++

```
__fastcall GetDesigntimeWarnings(TAbstractDesigntimeWarnings AWarnings);
```

Description

GetDesigntimeWarnings is called automatically at design time when the adapter page producer is assembling the current version of its HTML form. This gives the dataset values list an opportunity to add messages to the version of the HTML form that appears at design time, describing problems that you can rectify.

AWarnings is the design-time warnings object that assembles design-time warning messages that appear in the HTML form.

If the DataSet property has not been set, GetDesigntimeWarnings adds a message that indicates this omission. If DataSet is set, but the dataset is not active, GetDesigntimeWarnings adds a message that indicates this problem. Finally, if the ValueField property is not set, GetDesigntimeWarnings adds a message indicating this omission.

See Also

DataSet

Active

ValueField

62.1.25.4.3 TCustomAdapterErrorList.ImplContent Method

Pascal

```
function ImplContent(Options: TWebContentOptions; Layout: TLayout): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout Layout);
```

Description

This is ImplContent, a member of class TCustomAdapterErrorList.

62.1.25.4.4 TCustomAdapterErrorList.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

62.1.25.5 TCustomAdapterErrorList Properties

The properties of the TCustomAdapterErrorList class are listed here.

Public Properties

	Name	Description
	Adapter (see page 6482)	Specifies a related adapter.

62.1.25.5.1 TCustomAdapterErrorList.Adapter Property

Specifies a related adapter.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

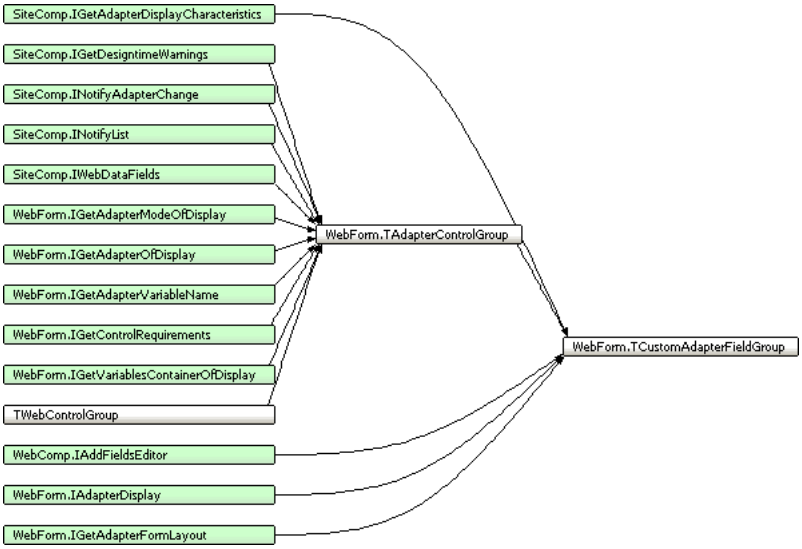
Description

Adapter specifies the adapter component for which this component reports errors.

62.1.26 TCustomAdapterFieldGroup Class

TCustomAdapterFieldGroup manages a collection of Web Items that generate a group of HTML controls.

Class Hierarchy



Pascal

```
TCustomAdapterFieldGroup = class(TAdapterControlGroup, IAdapterDisplay,
IGetAdapterDisplayCharacteristics, IAddFieldsEditor, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterFieldGroup : public TAdapterControlGroup, public IAdapterDisplay, public
IGetAdapterDisplayCharacteristics, public IAddFieldsEditor, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterFieldGroup as a base class for components that generate organized portions of the HTML document containing one or more related HTML controls. TCustomAdapterFieldGroup introduces the properties and methods for managing a collection of Web items that generate the HTML controls that make up the group.

In order to let other Web items use the HTML that defines the organized collection of controls, TCustomAdapterFieldGroup implements the IWebContent (see page 6103) interface.

Do not create instances of TCustomAdapterFieldGroup. TCustomAdapterFieldGroup manages the Web items that generate HTML defining controls, but does not combine the generated content into any content of its own. Instead, use a descendant of TCustomAdapterFieldGroup such as TAdapterFieldGroup (see page 6399).

See Also

TAdapterGrid (see page 6419)

TAdapterActionButton (see page 6316)

IWebContent (see page 6103)

62.1.26.1 TCustomAdapterFieldGroup Members

The following tables list the members exposed by TCustomAdapterFieldGroup.

Protected Methods

	Name	Description
	AdapterChange (see page 6345)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6345)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6346)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6346)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6346)	Returns the value of the Adapter (see page 6352) property.
	GetAdapterDisplayCharacteristics (see page 6486)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 6486)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAdapterModeOfAdapter (see page 6347)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6347)	Returns the value of the AdapterMode (see page 6353) property.
	GetAdapterVariableName (see page 6348)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6486)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 6487)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 6487)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetControlRequirements (see page 6348)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6348)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6349)	Fills a string list with the names of all of the associated adapter's fields.
	GetLayoutAttributes (see page 6487)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 6487)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
	GetVisibleFields (see page 6349)	Returns the value of the VisibleFields (see page 6354) property.
	ImplCanAddClass (see page 6488)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6488)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6350)	Provides the underlying implementation for the GetControlRequirements (see page 6348) method.
	IsFieldInUse (see page 6350)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6350)	Updates the TAdapterControlGroup (see page 6341) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6351)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6351)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6352)	Sets the value of the Adapter (see page 6352) property.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	Create (see page 6485)	Creates and initializes an instance of TCustomAdapterFieldGroup (see page 6483).
	Destroy (see page 6485)	Frees an instance of TCustomAdapterFieldGroup (see page 6483).
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6513)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterModeOfAdapter (see page 6512)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6512)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.

Public Properties

	Name	Description
	Adapter (see page 6352)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6353)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 6353)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	Custom (see page 6489)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6489)	Adds style attributes to the generated button group.
	StyleRule (see page 6489)	Applies a style in the page producer's style sheet to the button group.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 6354)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

62.1.26.2 TCustomAdapterFieldGroup.Create Constructor

Creates and initializes an instance of TCustomAdapterFieldGroup (see page 6483).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterFieldGroup(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterFieldGroup (see page 6483) at runtime. TCustomAdapterFieldGroup (see page 6483) objects added at design time using the Web Page editor are instantiated automatically.

62.1.26.3 TCustomAdapterFieldGroup.Destroy Destructor

Frees an instance of TCustomAdapterFieldGroup (see page 6483).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterFieldGroup();
```

Description

Destroy removes the Web control group from its parent's list of components that generate HTML before freeing its own resources.

62.1.26.4 TCustomAdapterFieldGroup Methods

The methods of the TCustomAdapterFieldGroup class are listed here.

Protected Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 6486)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.
	GetAdapterFormLayout (see page 6486)	This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.
	GetAddAllFieldsItem (see page 6486)	This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsDlgCaption (see page 6487)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.
	GetAddFieldsItem (see page 6487)	This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.
V	GetLayoutAttributes (see page 6487)	Returns an object that specifies how the generated group of controls should look.
	GetNewFieldPrefix (see page 6487)	This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.
V	ImplCanAddClass (see page 6488)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6488)	Provides the underlying implementation for the Content method.

62.1.26.4.1 TCustomAdapterFieldGroup.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterFieldGroup.

62.1.26.4.2 TCustomAdapterFieldGroup.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterFieldGroup.

62.1.26.4.3 TCustomAdapterFieldGroup.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TCustomAdapterFieldGroup.

62.1.26.4.4 TCustomAdapterFieldGroup.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TCustomAdapterFieldGroup.

62.1.26.4.5 TCustomAdapterFieldGroup.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TCustomAdapterFieldGroup.

62.1.26.4.6 TCustomAdapterFieldGroup.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content

62.1.26.4.7 TCustomAdapterFieldGroup.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TCustomAdapterFieldGroup.

62.1.26.4.8 TCustomAdapterFieldGroup.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass

WebFieldControls

62.1.26.4.9 TCustomAdapterFieldGroup.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content

AddElements

ILayoutWebContent (see page 6084)

62.1.26.5 TCustomAdapterFieldGroup Properties

The properties of the TCustomAdapterFieldGroup class are listed here.

Public Properties

	Name	Description
	Custom (see page 6489)	Represents a tag option and its value in the generated INPUT tag.
	Style (see page 6489)	Adds style attributes to the generated button group.
	StyleRule (see page 6489)	Applies a style in the page producer's style sheet to the button group.

62.1.26.5.1 TCustomAdapterFieldGroup.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the INPUT tag generated by the Web button group. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

```
ALIGN=TOP TABINDEX=2
```

See Also

Style (see page 6489)

62.1.26.5.2 TCustomAdapterFieldGroup.Style Property

Adds style attributes to the generated button group.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the INPUT tag generated by the Web button group. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated INPUT tag as the value of a STYLE option.

See Also

Custom (see page 6489)

StyleRule (see page 6489)

62.1.26.5.3 TCustomAdapterFieldGroup.StyleRule Property

Applies a style in the page producer's style sheet to the button group.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated button group. The value of StyleRule specifies one of the user-defined style names defined in the page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the button's Style (see page 6489) property.

See Also

Style (see page 6489)

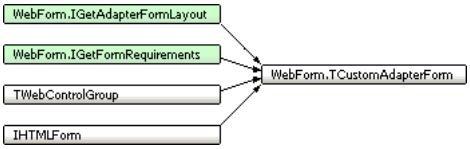
Styles (see page 1157)

StylesFile (see page 1158)

62.1.27 TCustomAdapterForm Class

TCustomAdapterForm is the base class for components that generate an HTML form for an adapter page producer.

Class Hierarchy



Pascal

```
TCustomAdapterForm = class(TWebControlGroup, IHTMLForm, IGetFormRequirements,
IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterForm : public TWebControlGroup, public IHTMLForm, public
IGetFormRequirements, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterForm as a base class when defining a form that can contain adapter fields and actions. TCustomAdapterForm is an abstract class, and should not be instantiated directly. Use TAdapterForm (see page 6404), which is a minimal implementation of TCustomAdapterForm, or create your own implementation.

See Also

TDataNavigator

TQueryButtons

TDataGrid

TFieldGroup

TQueryFieldGroup

IWebContent (see page 6103)

62.1.27.1 TCustomAdapterForm Members

The following tables list the members exposed by TCustomAdapterForm.

Protected Methods

	Name	Description
	GetAdapterFormLayout (see page 6492)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetFormRequirements (see page 6492)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 6492)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 6493)	Returns the name of the javascript variable that refers to the generated form.
	GetLayoutAttributes (see page 6493)	Returns an object that specifies how the generated group of controls should look.
	ImplCanAddClass (see page 6493)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6494)	Provides the underlying implementation for the Content method.

Public Methods

	Name	Description
	Create (see page 6491)	Creates and initializes an instance of TCustomAdapterForm (see page 6490).
	Destroy (see page 6491)	Frees an instance of TCustomAdapterForm (see page 6490).

Public Properties

	Name	Description
	Custom (see page 6495)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6495)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6495)	Adds style attributes to the generated form.
	StyleRule (see page 6496)	Applies a style in the page producer's style sheet to the form.

62.1.27.2 TCustomAdapterForm.Create Constructor

Creates and initializes an instance of TCustomAdapterForm (see page 6490).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterForm(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterForm (see page 6490) at runtime. TCustomAdapterForm (see page 6490) objects added at design time using the Web Page editor are instantiated automatically.

62.1.27.3 TCustomAdapterForm.Destroy Destructor

Frees an instance of TCustomAdapterForm (see page 6490).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterForm();
```

Description

Do not call Destroy directly. Instead, use the Free method, which checks for that the object is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

62.1.27.4 TCustomAdapterForm Methods

The methods of the TCustomAdapterForm class are listed here.

Protected Methods

	Name	Description
	GetAdapterFormLayout (see page 6492)	This is GetAdapterFormLayout, a member of class TCustomAdapterForm.
	GetFormRequirements (see page 6492)	This is GetFormRequirements, a member of class TCustomAdapterForm.
	GetHTMLFormName (see page 6492)	Returns the name of the generated HTML form.
	GetHTMLFormVarName (see page 6493)	Returns the name of the javascript variable that refers to the generated form.
V	GetLayoutAttributes (see page 6493)	Returns an object that specifies how the generated group of controls should look.
V	ImplCanAddClass (see page 6493)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6494)	Provides the underlying implementation for the Content method.

62.1.27.4.1 TCustomAdapterForm.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterForm.

62.1.27.4.2 TCustomAdapterForm.GetFormRequirements Method

Pascal

```
procedure GetFormRequirements(ARequirements: TAbstractFormRequirements);
```

C++

```
__fastcall GetFormRequirements(TAbstractFormRequirements ARequirements);
```

Description

This is GetFormRequirements, a member of class TCustomAdapterForm.

62.1.27.4.3 TCustomAdapterForm.GetHTMLFormName Method

Returns the name of the generated HTML form.

Pascal

```
function GetHTMLFormName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormName();
```

Description

GetHTMLFormName is the read access method for the HTMLFormName property.

See Also

HTMLFormName

62.1.27.4.4 TCustomAdapterForm.GetHTMLFormVarName Method

Returns the name of the javascript variable that refers to the generated form.

Pascal

```
function GetHTMLFormVarName: string;
```

C++

```
__fastcall UnicodeString GetHTMLFormVarName();
```

Description

GetHTMLFormVarName is the read access method for the HTMLFormVarName property.

See Also

HTMLFormVarName

62.1.27.4.5 TCustomAdapterForm.GetLayoutAttributes Method

Returns an object that specifies how the generated group of controls should look.

Pascal

```
function GetLayoutAttributes: TLayoutAttributes; override;
```

C++

```
virtual __fastcall TLayoutAttributes GetLayoutAttributes();
```

Description

GetLayoutAttributes is used internally by Web items to allow the layout object supplied by a parent (as a parameter to the Content method) to take into account layout properties of the child component.

As implemented by TWebControlGroup, GetLayoutAttributes returns nil (Delphi) or NULL (C++), because TWebControlGroup descendants not make use of layout attributes.

See Also

Content

62.1.27.4.6 TCustomAdapterForm.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass

WebFieldControls

62.1.27.4.7 TCustomAdapterForm.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content

AddElements

ILayoutWebContent (see page 6084)

62.1.27.5 TCustomAdapterForm Properties

The properties of the TCustomAdapterForm class are listed here.

Public Properties

	Name	Description
	Custom (see page 6495)	Represents a tag option and its value in the generated INPUT tag.
	FormMethod (see page 6495)	Specifies the method that the adapter form adds to its generated <form> tag.
	Style (see page 6495)	Adds style attributes to the generated form.
	StyleRule (see page 6496)	Applies a style in the page producer's style sheet to the form.

62.1.27.5.1 TCustomAdapterForm.Custom Property

Represents a tag option and its value in the generated INPUT tag.

Pascal

```
property Custom: string;
```

C++

```
__property UnicodeString Custom;
```

Description

Use Custom to represent one or more options that qualify the <table> tag that the adapter form generates to manage the layout of its Web items. Each option and its value has the form

```
OPTIONNAME=Value
```

The options specified by Custom should be separated by spaces. For example, to add an ALIGN option that aligns the caption and a TABINDEX tag to control tab order, set Custom to

ALIGN=TOP TABINDEX=2

See Also

Style (see page 6495)

62.1.27.5.2 TCustomAdapterForm.FormMethod Property

Specifies the method that the adapter form adds to its generated <form> tag.

Pascal

```
property FormMethod: TFormMethod;
```

C++

```
__property TFormMethod FormMethod;
```

Description

Set FormMethod to specify the METHOD header for HTTP messages that the form submits. This value gives rise to the MethodType property of the HTTP request message.

See Also

MethodType

62.1.27.5.3 TCustomAdapterForm.Style Property

Adds style attributes to the generated form.

Pascal

```
property Style: string;
```

C++

```
__property UnicodeString Style;
```

Description

Use Style to add a STYLE option that qualifies the <table> tag that the adapter form generates to manage the layout of its Web items. The Style option consists of one or more attribute descriptions such as

```
font-family: arial; font-weight: bold; font-size: 10px
```

These attributes are added to the generated <table> tag as the value of a STYLE option.

See Also

Custom (see page 6495)

StyleRule (see page 6496)

62.1.27.5.4 TCustomAdapterForm.StyleRule Property

Applies a style in the page producer's style sheet to the form.

Pascal

```
property StyleRule: string;
```

C++

```
__property UnicodeString StyleRule;
```

Description

Use StyleRule to assign one of the user-defined styles in the HTML document's style sheet to the generated form. The value of StyleRule specifies one of the user-defined style names defined in the adapter page producer's Styles or StylesFile property.

Note: To apply a style definition that is not included in the style sheet, use the form's Style (see page 6495) property.

See Also

Style (see page 6495)

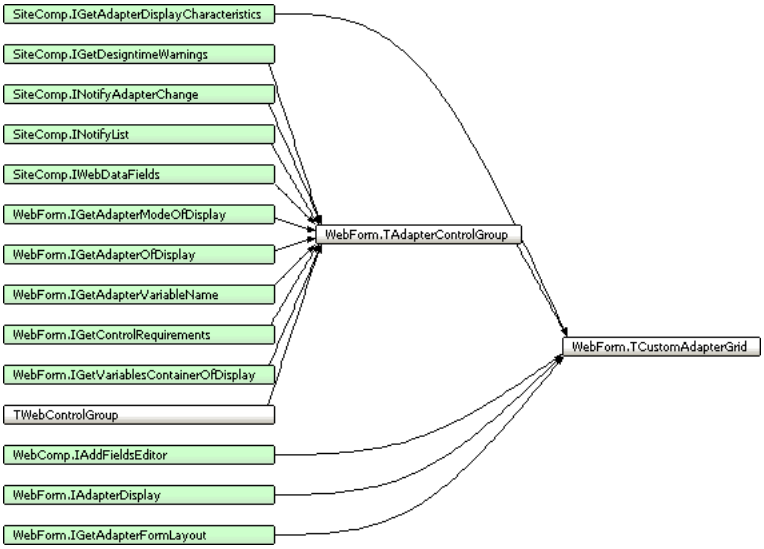
Styles (see page 1157)

StylesFile (see page 1158)

62.1.28 TCustomAdapterGrid Class

TCustomAdapterGrid is the base class for Web items that generate an HTML table to display a datasetin an XML data packet or web application.

Class Hierarchy



Pascal

```
TCustomAdapterGrid = class(TAdapterControlGroup, IAdapterDisplay,
IGetAdapterDisplayCharacteristics, IAddFieldsEditor, IGetAdapterFormLayout);
```

C++

```
class TCustomAdapterGrid : public TAdapterControlGroup, public IAdapterDisplay, public
IGetAdapterDisplayCharacteristics, public IAddFieldsEditor, public IGetAdapterFormLayout;
```

File

WebForm

Description

Use TCustomAdapterGrid as a base class for Web items that generate an HTML table to display multiple records of data supplied by an adapter component. All of the adapter grid's child Web items display field components that use the same adapter, which is specified by a property of the adapter grid.

If no display fields are explicitly added to the adapter grid, it displays all of the adapter's display fields by default. The properties of default display fields cannot be modified.

TCustomAdapterGrid introduces properties to

Specify the associated Adapter (see page 6352)

Specify whether default fields should be added

Customize the appearance of the HTML table.

Do not create instances of TCustomAdapterGrid. Instead, use a descendant of TCustomAdapterGrid such as TAdapterGrid (see page 6419).

See Also

TAdapterCommandColumn (see page 6331)

TAdapterGrid (see page 6419)

TAdapterFieldGroup (see page 6399)

IWebContent (see page 6103)

62.1.28.1 TCustomAdapterGrid Members

The following tables list the members exposed by TCustomAdapterGrid.

Protected Methods

	Name	Description
	AdapterChange (see page 6345)	Responds when the adapter associated with this control group changes.
	AddNotify (see page 6345)	Registers interest so that the adapter control group notifies an object of changes to its adapter.
	CreateDefaultFields (see page 6346)	Generates a list of the default adapter display field components that generate controls for the default fields of the control group's adapter.
	FindField (see page 6346)	Returns the child Web item that represents the value of a specified adapter field.
	GetAdapter (see page 6346)	Returns the value of the Adapter (see page 6352) property.
	GetAdapterDisplayCharacteristics (see page 6500)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 6500)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAdapterModeOfAdapter (see page 6347)	Returns the current mode of the control group's adapter.
	GetAdapterModeOfDisplay (see page 6347)	Returns the value of the AdapterMode (see page 6353) property.
	GetAdapterVariableName (see page 6348)	Returns the variable name that refers to the control group's adapter in server-side script.
	GetAddAllFieldsItem (see page 6500)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 6501)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 6501)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetControlRequirements (see page 6348)	Populates a control requirements object with declarations of the variables and functions this control group requires for generating its HTML.
	GetDesigntimeWarnings (see page 6348)	Populates a design-time warnings object with any warnings that the adapter control group needs to emit.
	GetFieldsList (see page 6349)	Fills a string list with the names of all of the associated adapter's fields.
	GetNewFieldPrefix (see page 6501)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
	GetVisibleFields (see page 6349)	Returns the value of the VisibleFields (see page 6354) property.
	ImplCanAddClass (see page 6501)	Provides the underlying implementation for the CanAddClass method.
	ImplContent (see page 6502)	Provides the underlying implementation for the Content method.
	ImplGetControlRequirements (see page 6350)	Provides the underlying implementation for the GetControlRequirements (see page 6348) method.
	IsFieldInUse (see page 6350)	Indicates whether one of the adapter control group's child Web items represents a specified adapter field.
	Notification (see page 6350)	Updates the TAdapterControlGroup (see page 6341) object when components are added to or removed from the application.
	NotifyAdapterChange (see page 6351)	Responds when the control group's adapter notifies it of a change.
	RemoveNotify (see page 6351)	Unregisters interest in changes to the control group's adapter.
	SetAdapter (see page 6352)	Sets the value of the Adapter (see page 6352) property.

Public Methods

	Name	Description
	AddNotify (see page 1059)	Registers an interest in change notifications.
	Create (see page 6499)	Creates and initializes an instance of TCustomAdapterGrid (see page 6496).
	Destroy (see page 6499)	Frees an instance of TCustomAdapterGrid (see page 6496).
	FindField (see page 1082)	This is FindField, a member of class IWebDataFields.
	GetAdapter (see page 6513)	This is GetAdapter, a member of class IGetAdapterOfDisplay.
	GetAdapterModeOfAdapter (see page 6512)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6512)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.
	GetDesigntimeWarnings (see page 1029)	Populates a design-time warnings object with any warnings that the dataset values list needs to emit.
	GetFieldsList (see page 1082)	This is GetFieldsList, a member of class IWebDataFields.
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.
	GetVisibleFields (see page 1083)	This is GetVisibleFields, a member of class IWebDataFields.
	IsFieldInUse (see page 1083)	This is IsFieldInUse, a member of class IWebDataFields.
	NotifyAdapterChange (see page 1058)	Responds when the adapter changes.
	RemoveNotify (see page 1059)	Unregisters interest in change notifications.

Public Properties

	Name	Description
	Adapter (see page 6352)	Specifies the adapter that supplies values to or receives values from the controls in the group.
	AdapterMode (see page 6353)	Specifies the mode of the adapter that the controls in the group support.
R	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.
	AddDefaultFields (see page 6353)	Specifies whether the adapter control group generates controls for the adapter's default fields when you do not explicitly add Web items.
	HeadingAttributes (see page 6502)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6503)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6503)	Determines the basic display characteristics of the table.
R	VisibleFields (see page 1083)	This is VisibleFields, a member of class IWebDataFields.
R	VisibleFields (see page 6354)	Lists the child Web items that generate the visible HTML controls this adapter control group arranges.

62.1.28.2 TCustomAdapterGrid.Create Constructor

Creates and initializes an instance of TCustomAdapterGrid (see page 6496).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomAdapterGrid(TComponent * AOwner);
```

Description

Call Create to instantiate TCustomAdapterGrid (see page 6496) at runtime. TCustomAdapterGrid (see page 6496) objects added at design time using the Web Page editor are instantiated automatically.

62.1.28.3 TCustomAdapterGrid.Destroy Destructor

Frees an instance of TCustomAdapterGrid (see page 6496).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomAdapterGrid();
```

Description

Destroy removes the Web control group from its parent's list of components that generate HTML before freeing its own resources.

62.1.28.4 TCustomAdapterGrid Methods

The methods of the TCustomAdapterGrid class are listed here.

Protected Methods

	Name	Description
	GetAdapterDisplayCharacteristics (see page 6500)	This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.
	GetAdapterFormLayout (see page 6500)	This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.
	GetAddAllFieldsItem (see page 6500)	This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.
	GetAddFieldsDlgCaption (see page 6501)	This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.
	GetAddFieldsItem (see page 6501)	This is GetAddFieldsItem, a member of class TCustomAdapterGrid.
	GetNewFieldPrefix (see page 6501)	This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.
V	ImplCanAddClass (see page 6501)	Provides the underlying implementation for the CanAddClass method.
V	ImplContent (see page 6502)	Provides the underlying implementation for the Content method.

62.1.28.4.1 TCustomAdapterGrid.GetAdapterDisplayCharacteristics Method

Pascal

```
function GetAdapterDisplayCharacteristics: TAdapterDisplayCharacteristics;
```

C++

```
__fastcall TAdapterDisplayCharacteristics GetAdapterDisplayCharacteristics();
```

Description

This is GetAdapterDisplayCharacteristics, a member of class TCustomAdapterGrid.

62.1.28.4.2 TCustomAdapterGrid.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class TCustomAdapterGrid.

62.1.28.4.3 TCustomAdapterGrid.GetAddAllFieldsItem Method

Pascal

```
function GetAddAllFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddAllFieldsItem();
```

Description

This is GetAddAllFieldsItem, a member of class TCustomAdapterGrid.

62.1.28.4.4 TCustomAdapterGrid.GetAddFieldsDlgCaption Method

Pascal

```
function GetAddFieldsDlgCaption: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsDlgCaption();
```

Description

This is GetAddFieldsDlgCaption, a member of class TCustomAdapterGrid.

62.1.28.4.5 TCustomAdapterGrid.GetAddFieldsItem Method

Pascal

```
function GetAddFieldsItem: string;
```

C++

```
__fastcall UnicodeString GetAddFieldsItem();
```

Description

This is GetAddFieldsItem, a member of class TCustomAdapterGrid.

62.1.28.4.6 TCustomAdapterGrid.GetNewFieldPrefix Method

Pascal

```
function GetNewFieldPrefix: string;
```

C++

```
__fastcall UnicodeString GetNewFieldPrefix();
```

Description

This is GetNewFieldPrefix, a member of class TCustomAdapterGrid.

62.1.28.4.7 TCustomAdapterGrid.ImplCanAddClass Method

Provides the underlying implementation for the CanAddClass method.

Pascal

```
function ImplCanAddClass(AParent: TComponent; AClass: TClass): Boolean; override;
```

C++

```
virtual __fastcall Boolean ImplCanAddClass(TComponent * AParent, TClass AClass);
```

Description

The CanAddClass method calls ImpleCanAddClass to determine whether the TWebControlGroup descendant can add a component of type AClass to its WebComponents property list. AParent is the TWebControlGroup instance to which the component will be added.

In TWebControlGroup, ImplCanAddClass always returns true. TWebControlGroup descendants override ImplCanAddClass to indicate what types of Web Items they can use when generating their HTML content.

See Also

CanAddClass

WebFieldControls

62.1.28.4.8 TCustomAdapterGrid.ImplContent Method

Provides the underlying implementation for the Content method.

Pascal

```
function ImplContent(Options: TWebContentOptions; ParentLayout: TLayout): string; override;
```

C++

```
virtual __fastcall UnicodeString ImplContent(TWebContentOptions Options, TLayout ParentLayout);
```

Description

The Content method calls ImplContent to generate the group of HTML controls. As implemented in TWebControlGroup, ImplContent returns an empty string. TWebControlGroup descendants override it to generate HTML for the group of controls.

Options contains a set of flags that limit what can be included in the generated HTML.

ParentLayout specifies a helper object that the caller supplies for determining how the TWebControlGroup should format the HTML controls it generates. It implements the ILayoutWebContent interface.

ImplContent returns the generated HTML.

See Also

Content

AddElements

ILayoutWebContent (see page 6084)

62.1.28.5 TCustomAdapterGrid Properties

The properties of the TCustomAdapterGrid class are listed here.

Public Properties

	Name	Description
	HeadingAttributes (see page 6502)	Determines the display characteristics of the table's header.
	RowAttributes (see page 6503)	Determines the display characteristics of the table's body.
	TableAttributes (see page 6503)	Determines the basic display characteristics of the table.

62.1.28.5.1 TCustomAdapterGrid.HeadingAttributes Property

Determines the display characteristics of the table's header.

Pascal

```
property HeadingAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes HeadingAttributes;
```

Description

Use HeadingAttributes to specify any display attributes that apply only to the heading of the generated HTML table.

HeadingAttributes determines the options that are added to the <TR> tag that defines the header of the grid. These include the color of the header, the alignment of the caption for each column, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the rows of the grid, use the TableAttributes (see page 6503) or RowAttributes (see page 6503) property instead.

See Also

TableAttributes (see page 6503)

RowAttributes (see page 6503)

62.1.28.5.2 TCustomAdapterGrid.RowAttributes Property

Determines the display characteristics of the table's body.

Pascal

```
property RowAttributes: TGridRowAttributes;
```

C++

```
__property TGridRowAttributes RowAttributes;
```

Description

Use RowAttributes to specify any display attributes that apply only to the rows of the generated HTML table. RowAttributes determines the options that are added to the <TR> tag for each row in the body of the grid. These include the color of the rows, the alignment of data within each cell, and any custom options or style rules.

Note: To influence the display attributes of the grid as a whole or only the header of the grid, use the TableAttributes (see page 6503) or HeadingAttributes (see page 6502) property instead.

See Also

TableAttributes (see page 6503)

HeadingAttributes (see page 6502)

62.1.28.5.3 TCustomAdapterGrid.TableAttributes Property

Determines the basic display characteristics of the table.

Pascal

```
property TableAttributes: TGridAttributes;
```

C++

```
__property TGridAttributes TableAttributes;
```

Description

Use TableAttributes to specify values for the most common options on the HTML <TABLE> tag. TableAttributes lets you specify the alignment of the table in its HTML document, its color and border size, the offset of data within each table cell, as well as any custom options or style rules.

Note: To influence the display attributes of only the header of the grid, or only the rows in its body, use the TableAttributes or HeadingAttributes (see page 6502) property instead.

See Also

RowAttributes (see page 6503)

HeadingAttributes (see page 6502)

62.1.29 TDesigntimeWarnings Class

Class Hierarchy

Pascal

```
TDesigntimeWarnings = class(TAbstractDesigntimeWarnings);
```

C++

```
class TDesigntimeWarnings : public TAbstractDesigntimeWarnings;
```

File

WebForm

Description

This is class WebForm.TDesigntimeWarnings.

62.1.29.1 TDesigntimeWarnings Members

The following tables list the members exposed by TDesigntimeWarnings.

Public Methods

	Name	Description
	AddObject (see page 6505)	Requests any design time warnings from a specified object.
	AddString (see page 6505)	Adds a warning string to the set of warnings that appear at design time.
	Create (see page 6504)	This is Create, a member of class TDesigntimeWarnings.
	Destroy (see page 6504)	This is Destroy, a member of class TDesigntimeWarnings.

Public Properties

	Name	Description
	Strings (see page 6506)	This is Strings, a member of class TDesigntimeWarnings.

62.1.29.2 TDesigntimeWarnings.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDesigntimeWarnings();
```

Description

This is Create, a member of class TDesigntimeWarnings.

62.1.29.3 TDesigntimeWarnings.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDesigntimeWarnings();
```

Description

This is Destroy, a member of class TDesigntimeWarnings.

62.1.29.4 TDesigntimeWarnings Methods

The methods of the TDesigntimeWarnings class are listed here.

Public Methods

	Name	Description
	AddObject (see page 6505)	Requests any design time warnings from a specified object.
	AddString (see page 6505)	Adds a warning string to the set of warnings that appear at design time.

62.1.29.4.1 TDesigntimeWarnings.AddObject Method

Requests any design time warnings from a specified object.

Pascal

```
procedure AddObject(AObject: TObject); override;
```

C++

```
virtual __fastcall AddObject(TObject * AObject);
```

Description

AddObject collects the design time warnings from the specified object by calling its GetDesigntimeWarnings method (if it has one). If the object has already been added by a previous call to AddObject, AddObject detects this and does nothing.

AObject is the Web item from which to request design-time warnings.

See Also

GetDesigntimeWarnings (see page 6384)

AddString (see page 964)

62.1.29.4.2 TDesigntimeWarnings.AddString Method

Adds a warning string to the set of warnings that appear at design time.

Pascal

```
procedure AddString(AWarning: string); override;
```

C++

```
virtual __fastcall AddString(UnicodeString AWarning);
```

Description

AddString adds a string to the design-time warnings object. The adapter page producer displays all such warning strings in the HTML form that appears in the HTML view of the editor.

AWarning is a warning string that describes a problem that the developer can rectify (such as an invalid property value or missing component).

See Also

TAdapterPageProducer (see page 1136)

AddObject (see page 964)

62.1.29.5 TDesigntimeWarnings Properties

The properties of the TDesigntimeWarnings class are listed here.

Public Properties

	Name	Description
	Strings (see page 6506)	This is Strings, a member of class TDesigntimeWarnings.

62.1.29.5.1 TDesigntimeWarnings.Strings Property

Pascal

```
property Strings: TStrings;
```

C++

```
__property TStrings Strings;
```

Description

This is Strings, a member of class TDesigntimeWarnings.

62.1.30 TGeneratedFunctions Class

Class Hierarchy

Pascal

```
TGeneratedFunctions = class;
```

C++

```
class TGeneratedFunctions;
```

File

WebForm

Description

This is class WebForm.TGeneratedFunctions.

62.1.30.1 TGeneratedFunctions Members

The following tables list the members exposed by TGeneratedFunctions.

Public Methods

	Name	Description
	Add (see page 6507)	This is Add, a member of class TGeneratedFunctions.

	Create (see page 6507)	This is Create, a member of class TGeneratedFunctions.
V	Destroy (see page 6507)	This is Destroy, a member of class TGeneratedFunctions.
	Exists (see page 6508)	This is Exists, a member of class TGeneratedFunctions.

Public Properties

	Name	Description
R	ItemCount (see page 6508)	This is ItemCount, a member of class TGeneratedFunctions.
R	Names (see page 6508)	This is Names, a member of class TGeneratedFunctions.
R	Script (see page 6508)	This is Script, a member of class TGeneratedFunctions.

62.1.30.2 TGeneratedFunctions.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TGeneratedFunctions();
```

Description

This is Create, a member of class TGeneratedFunctions.

62.1.30.3 TGeneratedFunctions.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TGeneratedFunctions();
```

Description

This is Destroy, a member of class TGeneratedFunctions.

62.1.30.4 TGeneratedFunctions Methods

The methods of the TGeneratedFunctions class are listed here.

Public Methods

	Name	Description
	Add (see page 6507)	This is Add, a member of class TGeneratedFunctions.
	Exists (see page 6508)	This is Exists, a member of class TGeneratedFunctions.

62.1.30.4.1 TGeneratedFunctions.Add Method

Pascal

```
procedure Add(const AName: string; const AScript: string);
```

C++

```
__fastcall Add(const UnicodeString AName, const UnicodeString AScript);
```

Description

This is Add, a member of class TGeneratedFunctions.

62.1.30.4.2 TGeneratedFunctions.Exists Method

Pascal

```
function Exists(const AName: string): Boolean;
```

C++

```
__fastcall Boolean Exists(const UnicodeString AName);
```

Description

This is Exists, a member of class TGeneratedFunctions.

62.1.30.5 TGeneratedFunctions Properties

The properties of the TGeneratedFunctions class are listed here.

Public Properties

	Name	Description
	ItemCount (see page 6508)	This is ItemCount, a member of class TGeneratedFunctions.
	Names (see page 6508)	This is Names, a member of class TGeneratedFunctions.
	Script (see page 6508)	This is Script, a member of class TGeneratedFunctions.

62.1.30.5.1 TGeneratedFunctions.ItemCount Property

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

This is ItemCount, a member of class TGeneratedFunctions.

62.1.30.5.2 TGeneratedFunctions.Names Property

Pascal

```
property Names [I: Integer]: string;
```

C++

```
__property UnicodeString Names[int I];
```

Description

This is Names, a member of class TGeneratedFunctions.

62.1.30.5.3 TGeneratedFunctions.Script Property

Pascal

```
property Script [I: Integer]: string;
```

C++

```
__property UnicodeString Script[int I];
```

Description

This is Script, a member of class TGeneratedFunctions.

62.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IAdapterDisplay (see page 6510)	This is class WebForm.IAdapterDisplay.
	IGetAdapterFormLayout (see page 6510)	This is class WebForm.IGetAdapterFormLayout.
	IGetAdapterModeOfDisplay (see page 6511)	This is class WebForm.IGetAdapterModeOfDisplay.
	IGetAdapterOfDisplay (see page 6512)	This is class WebForm.IGetAdapterOfDisplay.
	IGetAdapterVariableName (see page 6513)	This is class WebForm.IGetAdapterVariableName.
	IGetControlRequirements (see page 6515)	This is class WebForm.IGetControlRequirements.
	IGetFormRequirements (see page 6516)	This is class WebForm.IGetFormRequirements.
	IGetVariablesContainerOfDisplay (see page 6517)	This is class WebForm.IGetVariablesContainerOfDisplay.

62.2.1 IAdapterDisplay Interface

Class Hierarchy

WebForm.IAdapterDisplay

Pascal

```
IAdapterDisplay = interface;
```

C++

```
__interface IAdapterDisplay;
```

File

WebForm

Description

This is class WebForm.IAdapterDisplay.

62.2.2 IGetAdapterFormLayout Interface

Class Hierarchy

WebForm.IGetAdapterFormLayout

Pascal

```
IGetAdapterFormLayout = interface;
```

C++

```
__interface IGetAdapterFormLayout;
```

File

WebForm

Description

This is class WebForm.IGetAdapterFormLayout.

62.2.2.1 IGetAdapterFormLayout Members

The following tables list the members exposed by IGetAdapterFormLayout.

Public Methods

	Name	Description
	GetAdapterFormLayout (see page 6511)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

62.2.2.2 IGetAdapterFormLayout Methods

The methods of the IGetAdapterFormLayout class are listed here.

Public Methods

	Name	Description
	GetAdapterFormLayout (see page 6511)	This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

62.2.2.2.1 IGetAdapterFormLayout.GetAdapterFormLayout Method

Pascal

```
function GetAdapterFormLayout: TAdapterFormLayout;
```

C++

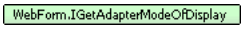
```
__fastcall TAdapterFormLayout GetAdapterFormLayout();
```

Description

This is GetAdapterFormLayout, a member of class IGetAdapterFormLayout.

62.2.3 IGetAdapterModeOfDisplay Interface

Class Hierarchy



Pascal

```
IGetAdapterModeOfDisplay = interface;
```

C++

```
__interface IGetAdapterModeOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetAdapterModeOfDisplay.

62.2.3.1 IGetAdapterModeOfDisplay Members

The following tables list the members exposed by IGetAdapterModeOfDisplay.

Public Methods

	Name	Description
	GetAdapterModeOfAdapter (see page 6512)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6512)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

62.2.3.2 IGetAdapterModeOfDisplay Methods

The methods of the IGetAdapterModeOfDisplay class are listed here.

Public Methods

	Name	Description
	GetAdapterModeOfAdapter (see page 6512)	This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.
	GetAdapterModeOfDisplay (see page 6512)	This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

62.2.3.2.1 IGetAdapterModeOfDisplay.GetAdapterModeOfAdapter Method

Pascal

```
function GetAdapterModeOfAdapter: string;
```

C++

```
__fastcall UnicodeString GetAdapterModeOfAdapter();
```

Description

This is GetAdapterModeOfAdapter, a member of class IGetAdapterModeOfDisplay.

62.2.3.2.2 IGetAdapterModeOfDisplay.GetAdapterModeOfDisplay Method

Pascal

```
function GetAdapterModeOfDisplay: string;
```

C++

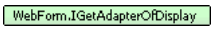
```
__fastcall UnicodeString GetAdapterModeOfDisplay();
```

Description

This is GetAdapterModeOfDisplay, a member of class IGetAdapterModeOfDisplay.

62.2.4 IGetAdapterOfDisplay Interface

Class Hierarchy



Pascal

```
IGetAdapterOfDisplay = interface;
```

C++

```
__interface IGetAdapterOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetAdapterOfDisplay.

62.2.4.1 IGetAdapterOfDisplay Members

The following tables list the members exposed by IGetAdapterOfDisplay.

Public Methods

	Name	Description
	GetAdapter (see page 6513)	This is GetAdapter, a member of class IGetAdapterOfDisplay.

62.2.4.2 IGetAdapterOfDisplay Methods

The methods of the IGetAdapterOfDisplay class are listed here.

Public Methods

	Name	Description
	GetAdapter (see page 6513)	This is GetAdapter, a member of class IGetAdapterOfDisplay.

62.2.4.2.1 IGetAdapterOfDisplay.GetAdapter Method

Pascal

```
function GetAdapter: TComponent;
```

C++

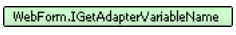
```
__fastcall TComponent * GetAdapter();
```

Description

This is GetAdapter, a member of class IGetAdapterOfDisplay.

62.2.5 IGetAdapterVariableName Interface

Class Hierarchy



Pascal

```
IGetAdapterVariableName = interface;
```

C++

```
__interface IGetAdapterVariableName;
```

File

WebForm

Description

This is class WebForm.IGetAdapterVariableName.

62.2.5.1 IGetAdapterVariableName Members

The following tables list the members exposed by IGetAdapterVariableName.

Public Methods

	Name	Description
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

Public Properties

	Name	Description
	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

62.2.5.2 IGetAdapterVariableName Methods

The methods of the IGetAdapterVariableName class are listed here.

Public Methods

	Name	Description
	GetAdapterVariableName (see page 6514)	This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

62.2.5.2.1 IGetAdapterVariableName.GetAdapterVariableName Method

Pascal

```
function GetAdapterVariableName: string;
```

C++

```
__fastcall UnicodeString GetAdapterVariableName();
```

Description

This is GetAdapterVariableName, a member of class IGetAdapterVariableName.

62.2.5.3 IGetAdapterVariableName Properties

The properties of the IGetAdapterVariableName class are listed here.

Public Properties

	Name	Description
	AdapterVariableName (see page 6515)	This is AdapterVariableName, a member of class IGetAdapterVariableName.

62.2.5.3.1 IGetAdapterVariableName.AdapterVariableName Property

Pascal

```
property AdapterVariableName: string;
```

C++

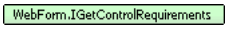
```
__property UnicodeString AdapterVariableName;
```

Description

This is AdapterVariableName, a member of class IGetAdapterVariableName.

62.2.6 IGetControlRequirements Interface

Class Hierarchy



Pascal

```
IGetControlRequirements = interface;
```

C++

```
__interface IGetControlRequirements;
```

File

WebForm

Description

This is class WebForm.IGetControlRequirements.

62.2.6.1 IGetControlRequirements Members

The following tables list the members exposed by IGetControlRequirements.

Public Methods

	Name	Description
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.

62.2.6.2 IGetControlRequirements Methods

The methods of the IGetControlRequirements class are listed here.

Public Methods

	Name	Description
	GetControlRequirements (see page 6516)	This is GetControlRequirements, a member of class IGetControlRequirements.

62.2.6.2.1 IGetControlRequirements.GetControlRequirements Method

Pascal

```
procedure GetControlRequirements(ARequirements: TAbstractControlRequirements);
```

C++

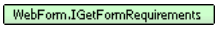
```
__fastcall GetControlRequirements(TAbstractControlRequirements ARequirements);
```

Description

This is GetControlRequirements, a member of class IGetControlRequirements.

62.2.7 IGetFormRequirements Interface

Class Hierarchy



Pascal

```
IGetFormRequirements = interface;
```

C++

```
__interface IGetFormRequirements;
```

File

WebForm

Description

This is class WebForm.IGetFormRequirements.

62.2.7.1 IGetFormRequirements Members

The following tables list the members exposed by IGetFormRequirements.

Public Methods

	Name	Description
	GetFormRequirements (see page 6517)	This is GetFormRequirements, a member of class IGetFormRequirements.

62.2.7.2 IGetFormRequirements Methods

The methods of the IGetFormRequirements class are listed here.

Public Methods

	Name	Description
	GetFormRequirements (see page 6517)	This is GetFormRequirements, a member of class IGetFormRequirements.

62.2.7.2.1 IGetFormRequirements.GetFormRequirements Method

Pascal

```
procedure GetFormRequirements(ARequirements: TAbstractFormRequirements);
```

C++

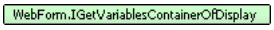
```
__fastcall GetFormRequirements(TAbstractFormRequirements ARequirements);
```

Description

This is GetFormRequirements, a member of class IGetFormRequirements.

62.2.8 IGetVariablesContainerOfDisplay Interface

Class Hierarchy



Pascal

```
IGetVariablesContainerOfDisplay = interface;
```

C++

```
__interface IGetVariablesContainerOfDisplay;
```

File

WebForm

Description

This is class WebForm.IGetVariablesContainerOfDisplay.

62.2.8.1 IGetVariablesContainerOfDisplay Members

The following tables list the members exposed by IGetVariablesContainerOfDisplay.

Public Methods

	Name	Description
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

62.2.8.2 IGetVariablesContainerOfDisplay Methods

The methods of the IGetVariablesContainerOfDisplay class are listed here.

Public Methods

	Name	Description
	GetVariablesContainer (see page 6518)	This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

62.2.8.2.1 IGetVariablesContainerOfDisplay.GetVariablesContainer Method

Pascal

```
function GetVariablesContainer: TComponent;
```

C++

```
__fastcall TComponent * GetVariablesContainer();
```

Description

This is GetVariablesContainer, a member of class IGetVariablesContainerOfDisplay.

62.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CalcDisplayFieldHTMLElementType (see page 6519)	This is function WebForm.CalcDisplayFieldHTMLElementType.
	CalcDisplayFieldViewMode (see page 6519)	This is function WebForm.CalcDisplayFieldViewMode.
	FindVariablesContainer (see page 6520)	This is function WebForm.FindVariablesContainer.
	FindVariablesContainerInParent (see page 6520)	This is function WebForm.FindVariablesContainerInParent.
	GetAdapterModeOfAdapterInParent (see page 6520)	This is function WebForm.GetAdapterModeOfAdapterInParent.
	MakeAdapterHTMLName (see page 6521)	This is function WebForm.MakeAdapterHTMLName.
	MakeAdapterVariableName (see page 6521)	This is function WebForm.MakeAdapterVariableName.

62.3.1 WebForm.CalcDisplayFieldHTMLElementType Function

Pascal

```
function CalcDisplayFieldHTMLElementType(AField: TComponent; const AAdapterMode: string;
ADisplayType: TDisplayFieldHTMLElementType; var AType: TDisplayFieldHTMLElementType): Boolean;
```

C++

```
Boolean CalcDisplayFieldHTMLElementType(TComponent * AField, const UnicodeString AAdapterMode,
TDisplayFieldHTMLElementType ADisplayType, TDisplayFieldHTMLElementType AType);
```

File

WebForm

Description

This is function WebForm.CalcDisplayFieldHTMLElementType.

62.3.2 WebForm.CalcDisplayFieldViewMode Function

Pascal

```
function CalcDisplayFieldViewMode(AField: TComponent; const AAdapterMode: string; AViewMode:
TDisplayFieldViewMode; var AMode: TDisplayFieldViewMode): Boolean;
```

C++

```
Boolean CalcDisplayFieldViewMode(TComponent * AField, const UnicodeString AAdapterMode,
TDisplayFieldViewMode AViewMode, TDisplayFieldViewMode AMode);
```

File

WebForm

Description

This is function WebForm.CalcDisplayFieldViewMode.

62.3.3 WebForm.FindVariablesContainer Function

Pascal

```
function FindVariablesContainer(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindVariablesContainer(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.FindVariablesContainer.

62.3.4 WebForm.FindVariablesContainerInParent Function

Pascal

```
function FindVariablesContainerInParent(AComponent: TComponent): TComponent;
```

C++

```
TComponent * FindVariablesContainerInParent(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.FindVariablesContainerInParent.

62.3.5 WebForm.GetAdapterModeOfAdapterInParent Function

Pascal

```
function GetAdapterModeOfAdapterInParent(AComponent: TComponent): string;
```

C++

```
UnicodeString GetAdapterModeOfAdapterInParent(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.GetAdapterModeOfAdapterInParent.

62.3.6 WebForm.MakeAdapterHTMLName Function

Pascal

```
function MakeAdapterHTMLName(AVarName: string): string;
```

C++

```
UnicodeString MakeAdapterHTMLName(UnicodeString AVarName);
```

File

WebForm

Description

This is function WebForm.MakeAdapterHTMLName.

62.3.7 WebForm.MakeAdapterVariableName Function

Pascal

```
function MakeAdapterVariableName(AComponent: TComponent): string;
```

C++

```
UnicodeString MakeAdapterVariableName(TComponent * AComponent);
```

File

WebForm

Description

This is function WebForm.MakeAdapterVariableName.

62.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TActionButtonHideOption (see page 6522)	TActionButtonHideOptions (see page 6528) indicates when a control to represent an adapter action should not be generated.
	TColumnHTMLElementType (see page 6523)	This is record WebForm.TColumnHTMLElementType.
	TCommandHTMLElementType (see page 6523)	TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.
	TDisplayFieldHideOption (see page 6524)	TDisplayFieldHideOptions (see page 6528) indicates when a control to represent a dataset field should not be generated.
	TDisplayFieldHTMLElementType (see page 6525)	TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TDisplayFieldViewMode (see page 6525)	TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.
	TInputFieldHTMLElementType (see page 6526)	TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.
	TLayoutState (see page 6527)	This is record WebForm.TLayoutState.

62.4.1 WebForm.TActionButtonHideOption Enumeration

TActionButtonHideOptions (see page 6528) indicates when a control to represent an adapter action should not be generated.

Pascal

```
TActionButtonHideOption = (
  bhoHideOnNoExecuteAccess,
  bhoHideOnActionNotVisible,
  bhoHideOnDisabledAction,
  bhoHideAlways
);
```

C++

```
enum TActionButtonHideOption {
  bhoHideOnNoExecuteAccess,
  bhoHideOnActionNotVisible,
  bhoHideOnDisabledAction,
  bhoHideAlways
};
```

File

WebForm

Description

TActionButtonHideOptions (see page 6528) is a set of TActionButtonHideOption values. It can include zero or more of the following values:

Value	Meaning
bhoHideOnNoExecuteAccess	Suppress the HTML control when the user does not have rights to execute the associated action, as indicated by the action's HasExecuteAccess method. When this flag is set, server-side script can suppress the control by setting the action's CanExecute property to false.

bhoHideOnActionNotVisible	Suppress the HTML control when the action returns false from its GetVisible method. When this flag is set, server-side script can suppress the control by setting the action's Visible property to false. This flag is set by default.
bhoHideOnDisabledAction	Suppress the HTML control when the action returns false from its WebEnabled method. When this flag is set, server-side script can suppress the control by setting the action's Enabled property to false.
bhoHideAlways	Always suppress the HTML control.

62.4.2 WebForm.TColumnHTMLElementType Enumeration

Pascal

```
TColumnHTMLElementType = (
  coltDefault,
  coltText,
  coltImage,
  coltList
);
```

C++

```
enum TColumnHTMLElementType {
  coltDefault,
  coltText,
  coltImage,
  coltList
};
```

File

WebForm

Description

This is record WebForm.TColumnHTMLElementType.

62.4.3 WebForm.TCommandHTMLElementType Enumeration

TCommandHTMLElementType identifies an HTML control type that can be used execute an adapter action.

Pascal

```
TCommandHTMLElementType = (
  ctDefault,
  ctButton,
  ctImage,
  ctAnchor,
  ctEventImages
);
```

C++

```
enum TCommandHTMLElementType {
  ctDefault,
  ctButton,
  ctImage,
  ctAnchor,
```

```
  ctEventImages
};
```

File

WebForm

Description

Each element type indicates a different HTML element. TCommandHTMLElementType defines HTML elements that the user can click to execute a command.

The following table lists the possible values:

Value	Meaning
ctDefault	Generate the element type that the adapter action object suggests.
ctButton	Generate a push button with a text label.
ctImage	Generate an image that the user can click.
ctAnchor	Generate a hypertext link.
ctEventImages	Generate an element that displays a different image, depending on the users action. That is, the image changes dynamically depending on user events such as whether the mouse is over the control, whether the user is clicking the control, and so on.

See Also

DisplayType (see page 6439)

62.4.4 WebForm.TDisplayFieldHideOption Enumeration

TDisplayFieldHideOptions (see page 6528) indicates when a control to represent a dataset field should not be generated.

Pascal

```
TDisplayFieldHideOption = (
  hoHideOnNoInputAccess,
  hoHideOnNoDisplayAccess,
  hoHideOnFieldNotVisible,
  hoHideAlways
);
```

C++

```
enum TDisplayFieldHideOption {
  hoHideOnNoInputAccess,
  hoHideOnNoDisplayAccess,
  hoHideOnFieldNotVisible,
  hoHideAlways
};
```

File

WebForm

Description

TDisplayFieldHideOptions (see page 6528) is a set of TDisplayFieldHideOption values. It can include zero or more of the following values:

Value	Meaning
hoHideOnNoInputAccess	Do not generate a control if the user does not have permission to modify this field.
hoHideOnNoDisplayAccess	Do not generate a control if the user does not have permission to view this field.
hoHideOnFieldNotVisable	Do not generate a control if the field is marked as hidden.
hoHideAlways	Never generate a control for the field.

62.4.5 WebForm.TDisplayFieldHTMLElementType Enumeration

TDisplayFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.

Pascal

```
TDisplayFieldHTMLElementType = (
  dftDefault,
  dftText,
  dftImage,
  dftList
);
```

C++

```
enum TDisplayFieldHTMLElementType {
  dftDefault,
  dftText,
  dftImage,
  dftList
};
```

File

WebForm

Description

Each element type indicates a different HTML element. TDisplayFieldHTMLElementType defines read-only HTML elements.

The following table lists the possible values:

Value	Meaning
dftDefault	Use the default display type of the component that generates the HTML control
dftText	Display as a text string.
dftImage	Display as an image.
dftList	Display as a list of values.

62.4.6 WebForm.TDisplayFieldViewMode Enumeration

TDisplayFieldViewMode indicates the mode in which an adapter field's value is displayed.

Pascal

```
TDisplayFieldViewMode = (
  vmDefault,
  vmInput,
  vmDisplay,
  vmToggleOnAccess
);
```

C++

```
enum TDisplayFieldViewMode {
  vmDefault,
  vmInput,
  vmDisplay,
  vmToggleOnAccess
};
```

File

WebForm

Description

The following table lists the possible values for the TDisplayFieldViewMode type and the meaning corresponding to each:

Value	Meaning
vmDefault	Use the default mode associated with the adapter.
vmInput	Generate an input control for the field.
vmDisplay	Generate a read-only control for the field.
vmToggleOnAccess	Generate a control that toggles between read-only and input mode when the user tries to access it.

62.4.7 WebForm.TInputFieldHTMLElementType Enumeration

TInputFieldHTMLElementType identifies an HTML type that can be used to display the value(s) associated with an adapter field.

Pascal

```
TInputFieldHTMLElementType = (
  iftDefault,
  iftTextInput,
  iftPasswordInput,
  iftSelect,
  iftSelectMultiple,
  iftRadio,
  iftCheckBox,
  iftTextArea,
  iftFile
);
```

C++

```
enum TInputFieldHTMLElementType {
  iftDefault,
  iftTextInput,
  iftPasswordInput,
  iftSelect,
  iftSelectMultiple,
  iftRadio,
  iftCheckBox,
  iftTextArea,
```

```
  iftFile
};
```

File

WebForm

Description

TInputFieldHTMLElementType defines the set of possible HTML controls that can be used to input an adapter field value. The following table lists the possible values for the TInputFieldHTMLElementType type and the meaning corresponding to each flag:

Flag	Input (see page 3827) field type
iftCheckBox	The input field is a check box.
iftDefault	The input field is the default type.
iftFile	The input field is a file.
iftPasswordInput	The input field is a password prompt.
iftRadio	The input field is a radio button.
iftSelect	The input field is selectable. iftSelect is only used with iftCheckBox and iftRadio.
iftSelectMultiple	More than one input field may be selected at a time. iftSelectMultiple is only used with iftCheckBox and iftRadio.
iftTextArea	The field is a text area.
iftTextInput	The input field is a text input box.

62.4.8 WebForm.TLayoutState Enumeration

Pascal

```
TLayoutState = (
  lsUnknown,
  lsFields,
  lsButtons
);
```

C++

```
enum TLayoutState {
  lsUnknown,
  lsFields,
  lsButtons
};
```

File

WebForm

Description

This is record WebForm.TLayoutState.

62.5 Types

The following table lists types in this documentation.

Types

Name	Description
TActionButtonHideOptions (see page 6528)	This is type WebForm.TActionButtonHideOptions.
TAdapterDataDisplayClass (see page 6528)	This is type WebForm.TAdapterDataDisplayClass.
TDisplayFieldHideOptions (see page 6528)	This is type WebForm.TDisplayFieldHideOptions.

62.5.1 WebForm.TActionButtonHideOptions Type

Pascal

```
TActionButtonHideOptions = set of TActionButtonHideOption;
```

C++

```
TActionButtonHideOption TActionButtonHideOptions;
```

File

WebForm

Description

This is type WebForm.TActionButtonHideOptions.

62.5.2 WebForm.TAdapterDataDisplayClass Type

Pascal

```
TAdapterDataDisplayClass = class of TAdapterDataDisplay;
```

C++

```
TAdapterDataDisplay TAdapterDataDisplayClass;
```

File

WebForm

Description

This is type WebForm.TAdapterDataDisplayClass.

62.5.3 WebForm.TDisplayFieldHideOptions Type

Pascal

```
TDisplayFieldHideOptions = set of TDisplayFieldHideOption;
```

C++

```
TDisplayFieldHideOption TDisplayFieldHideOptions;
```

File

WebForm

Description

This is type WebForm.TDisplayFieldHideOptions.

62.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sDisplayText (see page 6530)	This is constant WebForm.sDisplayText.
sFieldText (see page 6530)	This is constant WebForm.sFieldText.
sFormGet (see page 6531)	This is constant WebForm.sFormGet.
sFormPost (see page 6531)	This is constant WebForm.sFormPost.
sMultipartForm (see page 6531)	This is constant WebForm.sMultipartForm.

62.6.1 WebForm.sDisplayText Constant

Pascal

```
sDisplayText = '<%%= %0:s.DisplayText %%>';
```

C++

```
sDisplayText = '<%%= %0:s.DisplayText %%>';
```

File

WebForm

Description

This is constant WebForm.sDisplayText.

62.6.2 WebForm.sFieldText Constant

Pascal

```
sFieldText = '<%%= %0:s.EditText %%>';
```

C++

```
sFieldText = '<%%= %0:s.EditText %%>';
```

File

WebForm

Description

This is constant WebForm.sFieldText.

62.6.3 WebForm.sFormGet Constant

Pascal

```
sFormGet = 'get';
```

C++

```
sFormGet = 'get';
```

File

WebForm

Description

This is constant WebForm.sFormGet.

62.6.4 WebForm.sFormPost Constant

Pascal

```
sFormPost = 'post';
```

C++

```
sFormPost = 'post';
```

File

WebForm

Description

This is constant WebForm.sFormPost.

62.6.5 WebForm.sMultipartForm Constant

Pascal

```
sMultipartForm = 'multipart/form-data';
```

C++

```
sMultipartForm = 'multipart/form-data';
```

File

WebForm

Description

This is constant WebForm.sMultipartForm.

63 WebModu Namespace

This is namespace WebModu.

Topics

Name	Description
Classes (see page 6535)	The following table lists classes in this documentation.
Functions (see page 6612)	The following table lists functions in this documentation.
Types (see page 6614)	The following table lists types in this documentation.

Classes

	Name	Description
	EWebModuleException (see page 6535)	EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.
	TAdapterActionName (see page 6536)	TAdapterActionName describes an adapter action by name.
	TCustomWebAppDataModule (see page 6539)	TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.
	TCustomWebAppPageModule (see page 6544)	TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.
	TCustomWebDataModule (see page 6550)	TCustomWebDataModule is the base class for all Web modules in a WebSnap application.
	TCustomWebPageModule (see page 6556)	TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.
	TSiteModuleHelper (see page 6565)	TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.
	TSitePageModuleHelper (see page 6574)	TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.
	TWebAppDataModule (see page 6585)	TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.
	TWebAppPageModule (see page 6588)	TWebAppPageModule is used as the WebSnap application module when the application module generates a page.
	TWebAppPageModuleHelper (see page 6594)	This is class WebModu.TWebAppPageModuleHelper.
	TWebDataModule (see page 6597)	TWebDataModule is a container for components in a WebSnap application.
	TWebNotifyContainer (see page 6599)	This is class WebModu.TWebNotifyContainer.
	TWebPageModule (see page 6603)	TWebPageModule is a WebSnap module that generates a page.
	TWebPageModuleHelper (see page 6609)	This is class WebModu.TWebPageModuleHelper.

Functions

	Name	Description
	FindNamedModuleVariable (see page 6612)	This is function WebModu.FindNamedModuleVariable.
	GetModuleVariable (see page 6612)	This is function WebModu.GetModuleVariable.
	GetModuleVariableCount (see page 6612)	This is function WebModu.GetModuleVariableCount.

Types

Name	Description
TGetParamsEvent (see page 6614)	This is type WebModu.TGetParamsEvent.

63.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWebModuleException (see page 6535)	EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.
	TAdapterActionName (see page 6536)	TAdapterActionName describes an adapter action by name.
	TCustomWebAppDataModule (see page 6539)	TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.
	TCustomWebAppPageModule (see page 6544)	TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.
	TCustomWebDataModule (see page 6550)	TCustomWebDataModule is the base class for all Web modules in a WebSnap application.
	TCustomWebPageModule (see page 6556)	TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.
	TSiteModuleHelper (see page 6565)	TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.
	TSitePageModuleHelper (see page 6574)	TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.
	TWebAppDataModule (see page 6585)	TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.
	TWebAppPageModule (see page 6588)	TWebAppPageModule is used as the WebSnap application module when the application module generates a page.
	TWebAppPageModuleHelper (see page 6594)	This is class WebModu.TWebAppPageModuleHelper.
	TWebDataModule (see page 6597)	TWebDataModule is a container for components in a WebSnap application.
	TWebNotifyContainer (see page 6599)	This is class WebModu.TWebNotifyContainer.
	TWebPageModule (see page 6603)	TWebPageModule is a WebSnap module that generates a page.
	TWebPageModuleHelper (see page 6609)	This is class WebModu.TWebPageModuleHelper.

63.1.1 EWebModuleException Class

EWebModuleException is the exception that is raised when a WebSnap application tries to use a Web module to perform a task it does not support.

Class Hierarchy

Pascal

```
EWebModuleException = class(EWebBrokerException);
```

C++

```
class EWebModuleException : public EWebBrokerException;
```

File

WebModu

Description

EWebModuleException is the exception that is raised when a server-side script in a WebSnap application tries to do direct a Web module to perform a task it does not know how to handle. For example, Web modules raise this exception when a script

tries to access a variable in a Web module that does not support the use of variables.

See Also

TWebDataModule (see page 6597)

TWebAppDataModule (see page 6585)

TWebPageModule (see page 6603)

TWebAppPageModule (see page 6588)

63.1.2 TAdapterActionName Class

TAdapterActionName describes an adapter action by name.

Class Hierarchy

Pascal

```
TAdapterActionName = class(TPersistent);
```

C++

```
class TAdapterActionName : public TPersistent;
```

File

WebModu

Description

TAdapterAction represents a TAdapterAction object. It is used when the action is referenced by name rather than directly as a component. TAdapterAction publishes properties that specify the name of the action and the parent adapter component to which it belongs. The read-only Action (see page 6538) property provides access to the action component with the specified name that belongs to the specified adapter.

See Also

TAdapter (see page 5190)

TAdapterAction (see page 5201)

63.1.2.1 TAdapterActionName Members

The following tables list the members exposed by TAdapterActionName.

Protected Methods

	Name	Description
	AssignTo (see page 6537)	Copies the properties of an object to a destination object.

Public Properties

	Name	Description
	Action (see page 6538)	Indicates the action that this object specifies.

Published Methods

	Name	Description
	Create (see page 6537)	Creates and initializes an instance of TAdapterActionName (see page 6536).

Published Properties

	Name	Description
	ActionName (see page 6539)	Specifies the name of the action this object represents.
	Adapter (see page 6539)	Specifies the adapter to which the action belongs.

63.1.2.2 TAdapterActionName.Create Constructor

Creates and initializes an instance of TAdapterActionName (see page 6536).

Pascal

```
constructor Create(AModule: TComponent);
```

C++

```
__fastcall TAdapterActionName(TComponent * AModule);
```

Description

Call Create to instantiate TAdapterActionName (see page 6536).

AModule is the Web module instance that should be notified if the adapter specified by the Adapter (see page 6539) property is deleted. The TAdapterActionName (see page 6536) constructor calls this Adapter (see page 6539)'s FreeNotification method so that the adapter will notify the Web module if it (and its actions) are freed.

See Also

FreeNotification

Adapter (see page 6539)

63.1.2.3 TAdapterActionName Methods

The methods of the TAdapterActionName class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 6537)	Copies the properties of an object to a destination object.

63.1.2.3.1 TAdapterActionName.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should

be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3919)

63.1.2.4 TAdapterActionName Properties

The properties of the TAdapterActionName class are listed here.

Public Properties

	Name	Description
	Action (see page 6538)	Indicates the action that this object specifies.

Published Properties

	Name	Description
	ActionName (see page 6539)	Specifies the name of the action this object represents.
	Adapter (see page 6539)	Specifies the adapter to which the action belongs.

63.1.2.4.1 TAdapterActionName.Action Property

Indicates the action that this object specifies.

Pascal

```
property Action: TComponent;
```

C++

```
__property TComponent * Action;
```

Description

Read (see page 3634) Action to access the adapter action component specified by the Adapter (see page 6539) and ActionName (see page 6539) properties. That is, Action is the action with the name specified by ActionName (see page 6539) that belongs to the adapter specified by Adapter (see page 6539).

See Also

ActionName (see page 6539)

Adapter (see page 6539)

TAdapterAction (see page 5201)

63.1.2.4.2 TAdapterActionName.ActionName Property

Specifies the name of the action this object represents.

Pascal

```
property ActionName: string;
```

C++

```
__property UnicodeString ActionName;
```

Description

Set ActionName to the name of the desired action. This is the name of one of the actions that belongs to the adapter that the Adapter (see page 6539) property specifies.

See Also

Action (see page 6538)

Adapter (see page 6539)

Actions (see page 5572)

63.1.2.4.3 TAdapterActionName.Adapter Property

Specifies the adapter to which the action belongs.

Pascal

```
property Adapter: TComponent;
```

C++

```
__property TComponent * Adapter;
```

Description

Set Adapter to the adapter component to which the named action belongs. The TAdapterActionName (see page 6536) object represents the action that this adapter returns when you call its FindAction method, passing in the value of the ActionName (see page 6539) property.

See Also

Action (see page 6538)

ActionName (see page 6539)

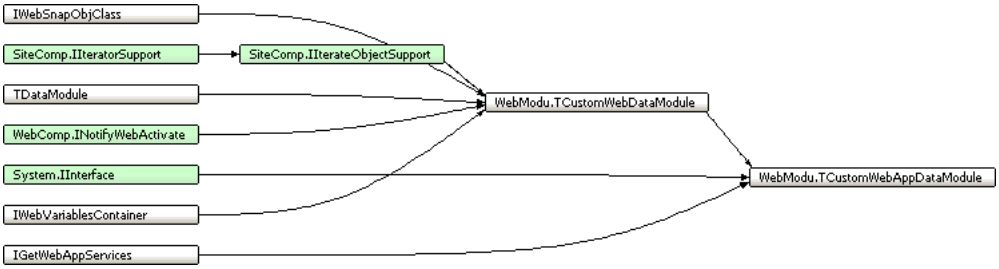
Actions (see page 5572)

FindAction (see page 5543)

63.1.3 TCustomWebAppDataModule Class

TCustomWebAppDataModule is the base class from which to derive new WebSnap application data modules.

Class Hierarchy



Pascal

```
TCustomWebAppDataModule = class(TCustomWebDataModule, IGetWebAppServices, IInterface);
```

C++

```
class TCustomWebAppDataModule : public TCustomWebDataModule, public IGetWebAppServices, public
IInterface;
```

File

WebModu

Description

Use TCustomWebAppDataModule as a base class for defining data modules that function as the main Web module in a WebSnap application. As the application module, TCustomWebAppDataModule provides access to the IWebAppServices interface that allows the WebSnap application to create a Web context for incoming requests and to forward requests to a dispatcher for handling.

If you want an application module that also has an associated HTML page, then derive from TCustomWebAppPageModule (see page 6544) instead.

See Also

TWebAppDataModule (see page 6585)

63.1.3.1 TCustomWebAppDataModule Members

The following tables list the members exposed by TCustomWebAppDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetWebAppServices (see page 6542)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6543)	Responds to notifications that components are about to be inserted or removed.
	QueryInterface (see page 6543)	Returns a reference to a specified interface if the module supports that interface.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.

Public Events

	Name	Description
	OnActivate (see page 6556)	Occurs when the Web module is activated.
	OnDeactivate (see page 6556)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	Create (see page 6541)	Creates a new TCustomWebAppDataModule (see page 6539) instance.
V	CreateNew (see page 6541)	Creates a data module without using the associated form file to initialize it.
V	Destroy (see page 6542)	Frees a Web application data module instance.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	AppServices (see page 6544)	Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 6539) object.
R	Request (see page 6554)	Provides access to the current request object.
R	Response (see page 6554)	Provides access to the current response object.
R	Session (see page 6555)	Provides access to the current Web session.

63.1.3.2 TCustomWebAppDataModule.Create Constructor

Creates a new TCustomWebAppDataModule (see page 6539) instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebAppDataModule(TComponent * AOwner);
```

Description

Use this constructor to create a new TCustomWebAppDataModule (see page 6539). WebSnap modules are registered with a factory that is responsible for creating modules as needed. The module's source unit defines a function that returns an instance of the module.

AOwner is the component that becomes the value of the Owner property.

63.1.3.3 TCustomWebAppDataModule.CreateNew Constructor

Creates a data module without using the associated form file to initialize it.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebAppDataModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6541) to create a data module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

63.1.3.4 TCustomWebAppDataModule.Destroy Destructor

Frees a Web application data module instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebAppDataModule();
```

Description

```
destructor Destroy; override;
```

Destroy is the destructor for the object instance. It clears the AppServices (see page 6544) property and calls the inherited destructor. The AppServices (see page 6544) interface is cleared to prevent design time problems that could otherwise occur because the object that implements AppServices (see page 6544) may also be owned by a Web page module.

63.1.3.5 TCustomWebAppDataModule Methods

The methods of the TCustomWebAppDataModule class are listed here.

Protected Methods

	Name	Description
	GetWebAppServices (see page 6542)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	Notification (see page 6543)	Responds to notifications that components are about to be inserted or removed.
V	QueryInterface (see page 6543)	Returns a reference to a specified interface if the module supports that interface.

63.1.3.5.1 TCustomWebAppDataModule.GetWebAppServices Method

Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

GetWebAppServices returns the IWebAppServices interface that is stored in this TCustomWebDataModule (see page 6550). This is the same as the value of the AppServices (see page 6544) property.

See Also

AppServices (see page 6544)

63.1.3.5.2 TCustomWebAppDataModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebAppDataModule (see page 6539) uses this method to check whether the component that implements AppServices (see page 6544) is about to be removed, and if so, sets the value of the AppServices (see page 6544) property to nil (Delphi) or NULL (C++).

See Also

AppServices (see page 6544)

63.1.3.5.3 TCustomWebAppDataModule.QueryInterface Method

Returns a reference to a specified interface if the module supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the Web application module supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the Web application module does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 3503)

63.1.3.6 TCustomWebAppDataModule Properties

The properties of the TCustomWebAppDataModule class are listed here.

Public Properties

	Name	Description
	AppServices (see page 6544)	Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 6539) object.

63.1.3.6.1 TCustomWebAppDataModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppDataModule (see page 6539) object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents (see page 6243) object.

See Also

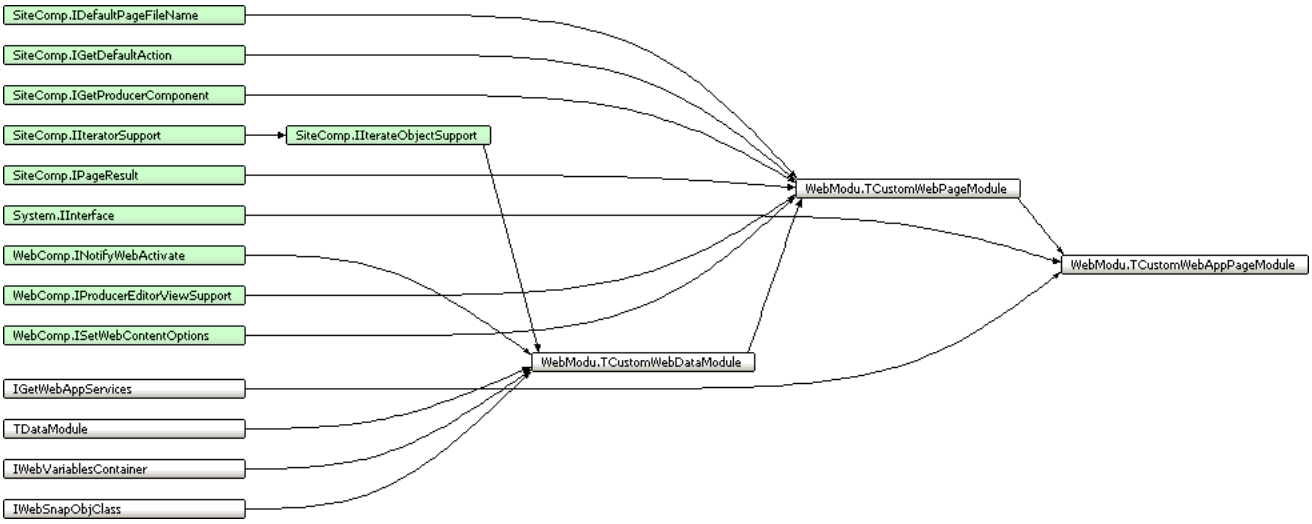
TWebAppComponents (see page 6243)

WebContext (see page 6003)

63.1.4 TCustomWebAppPageModule Class

TCustomWebAppPageModule is the base class for WebSnap application data modules that can produce Web pages.

Class Hierarchy



Pascal

```
TCustomWebAppPageModule = class(TCustomWebPageModule, IGetWebAppServices, IInterface);
```

C++

```
class TCustomWebAppPageModule : public TCustomWebPageModule, public IGetWebAppServices, public
IInterface;
```

File

WebModu

Description

Use TCustomWebAppPageModule as a base class for defining data modules that function as the main Web module in a WebSnap application and that have an associated Web page. As the application module, TCustomWebAppPageModule provides access to the IWebAppServices interface that allows the WebSnap application to create a Web context for incoming requests and to forward requests to a dispatcher for handling. As a Web page module, TCustomWebAppPageModule generates an associated Web page. This Web page is returned by default when incoming request messages do not specify what Web page they want.

See Also

IWebAppServices

TWebAppPageModule (see page 6588)

WebContext (see page 6003)

63.1.4.1 TCustomWebAppPageModule Members

The following tables list the members exposed by TCustomWebAppPageModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSiteModuleHelper (see page 6560)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSitePageModuleHelper (see page 6561)	Returns the value of the SitePageModuleHelper (see page 6562) property.
	GetWebAppServices (see page 6548)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6548)	Responds to notifications that components are about to be inserted or removed.
	QueryInterface (see page 6548)	Returns a reference to a specified interface if the module supports that interface.
	SetAppServices (see page 6549)	Sets the value of the AppServices (see page 6549) property.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.
	SitePageModuleHelper (see page 6562)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnActivate (see page 6556)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6563)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 6563)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6564)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 6564)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6556)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	Create (see page 6546)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6544).
V	CreateNew (see page 6547)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6544) without using an associated form file.
V	Destroy (see page 6547)	Frees an instance of TCustomWebAppPageModule (see page 6544).
	DispatchPage (see page 1065)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 1028)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 1035)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1065)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RedirectToPage (see page 1066)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 1015)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	AppServices (see page 6549)	Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 6544) object.
R	DefaultAction (see page 1028)	This is DefaultAction, a member of class IGetDefaultAction.
	DefaultAction (see page 6561)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6562)	Indicates the interface of the page producer that generates the content of the Web page.
R	Request (see page 6554)	Provides access to the current request object.
R	Response (see page 6554)	Provides access to the current response object.
R	Session (see page 6555)	Provides access to the current Web session.

63.1.4.2 TCustomWebAppPageModule.Create Constructor

Creates and initializes an instance of TCustomWebAppPageModule (see page 6544).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebAppPageModule(TComponent * AOwner);
```

Description

Do not call the TCustomWebAppPageModule (see page 6544) constructor. Instead, use a TWebAppPageModuleFactory

object to create Web module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 6297)

Owner

63.1.4.3 TCustomWebAppPageModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebAppPageModule (see page 6544) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebAppPageModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6546) to create a data module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 6297)

Owner

63.1.4.4 TCustomWebAppPageModule.Destroy Destructor

Frees an instance of TCustomWebAppPageModule (see page 6544).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebAppPageModule();
```

Description

Destroy is the destructor for the object instance. It clears the AppServices (see page 6549) property and calls the inherited destructor. The AppServices (see page 6549) interface is cleared to prevent design time problems that could otherwise occur because the object that implements AppServices (see page 6549) may also be owned by a Web page module.

See Also

AppServices (see page 6549)

System::TObject::Free (see page 3372)

63.1.4.5 TCustomWebAppPageModule Methods

The methods of the TCustomWebAppPageModule class are listed here.

Protected Methods

	Name	Description
	GetWebAppServices (see page 6548)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
V	Notification (see page 6548)	Responds to notifications that components are about to be inserted or removed.
V	QueryInterface (see page 6548)	Returns a reference to a specified interface if the module supports that interface.
	SetAppServices (see page 6549)	Sets the value of the AppServices (see page 6549) property.

63.1.4.5.1 TCustomWebAppPageModule.GetWebAppServices Method

Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.

Pascal

```
function GetWebAppServices: IWebAppServices;
```

C++

```
__fastcall IWebAppServices GetWebAppServices();
```

Description

GetWebAppServices returns the IWebAppServices interface that is stored in this Web application module. This is the same as the value of the AppServices (see page 6549) property.

See Also

AppServices (see page 6549)

63.1.4.5.2 TCustomWebAppPageModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebAppPageModule (see page 6544) uses this method to check whether the component that implements AppServices (see page 6549) is about to be removed, and if so, sets the value of the AppServices (see page 6549) property to nil (Delphi) or NULL (C++).

See Also

AppServices (see page 6549)

63.1.4.5.3 TCustomWebAppPageModule.QueryInterface Method

Returns a reference to a specified interface if the module supports that interface.

Pascal

```
function QueryInterface(const IID: TGUID; out Obj): HResult; override;
```

C++

```
virtual __fastcall HRESULT QueryInterface(const TGUID IID,  Obj);
```

Description

QueryInterface checks whether the Web application module supports the interface specified by IID. If so, QueryInterface

increments the reference count.

sets the Obj parameter so that it points to an instance of the specified interface.

returns 0 to indicate success.

If the Web application module does not support the interface, QueryInterface returns a nonzero error code such as E_NoInterface.

See Also

_AddRef (see page 3503)

63.1.4.5.4 TCustomWebAppPageModule.SetAppServices Method

Sets the value of the AppServices (see page 6549) property.

Pascal

```
procedure SetAppServices(const Value: IWebAppServices);
```

C++

```
__fastcall SetAppServices(const IWebAppServices Value);
```

Description

SetAppServices is the protected write implemenation of the AppServices (see page 6549) property.

See Also

AppServices (see page 6549)

63.1.4.6 TCustomWebAppPageModule Properties

The properties of the TCustomWebAppPageModule class are listed here.

Public Properties

	Name	Description
	AppServices (see page 6549)	Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 6544) object.

63.1.4.6.1 TCustomWebAppPageModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppPageModule (see page 6544) object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents (see page 6243) object.

See Also

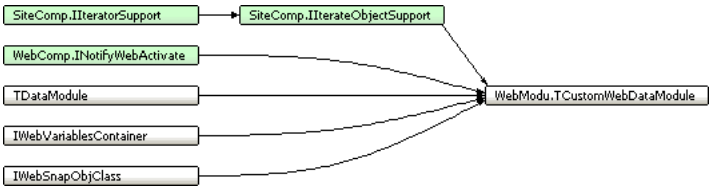
TWebAppComponents (see page 6243)

WebContext (see page 6003)

63.1.5 TCustomWebDataModule Class

TCustomWebDataModule is the base class for all Web modules in a WebSnap application.

Class Hierarchy



Pascal

```
TCustomWebDataModule = class(TDataModule, IWebVariablesContainer, INotifyWebActivate,
IIterateObjectSupport, IWebSnapObjClass);
```

C++

```
class TCustomWebDataModule : public TDataModule, public IWebVariablesContainer, public
INotifyWebActivate, public IIterateObjectSupport, public IWebSnapObjClass;
```

File

WebModu

Description

TCustomWebDataModule is the common base class for Web modules that can be used in a WebSnap application. This class introduces properties for the current HTTP request and response objects, a property to hold the current session object, and events that let you respond when the Web module is activated and deactivated. It also introduces support for interacting with server-side script and managing the WebSnap components that are placed into the Web module.

Do not create instances of TCustomWebDataModule. It is intended only as a base class. Instead, use the Web module wizard or the WebSnap application wizard to add a TCustomWebDataModule descendant to your WebSnap application. Descendant classes include TWebAppDataModule (see page 6585), TWebAppPageModule (see page 6588), TWebPageModule (see page 6603), and TWebDataModule (see page 6597).

See Also

TWebAppDataModule (see page 6585)

TWebAppPageModule (see page 6588)

TWebPageModule (see page 6603)

TWebDataModule (see page 6597)

63.1.5.1 TCustomWebDataModule Members

The following tables list the members exposed by TCustomWebDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6553)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.

Public Events

	Name	Description
	OnActivate (see page 6556)	Occurs when the Web module is activated.
	OnDeactivate (see page 6556)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	Create (see page 6551)	Creates and initializes an instance of TCustomWebDataModule (see page 6550).
	CreateNew (see page 6552)	Creates and initializes an instance of TCustomWebDataModule (see page 6550) without using an associated form file.
	Destroy (see page 6552)	Frees an instance of TCustomWebDataModule (see page 6550).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	Request (see page 6554)	Provides access to the current request object.
	Response (see page 6554)	Provides access to the current response object.
	Session (see page 6555)	Provides access to the current Web session.

63.1.5.2 TCustomWebDataModule.Create Constructor

Creates and initializes an instance of TCustomWebDataModule (see page 6550).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebDataModule(TComponent * AOwner);
```

Description

Do not call the TCustomWebDataModule (see page 6550) constructor. Instead, use a Web module factory object to create Web module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 6297)

TWebAppDataModuleFactory (see page 6295)

TWebDataModuleFactory (see page 6299)

TWebPageModuleFactory (see page 6303)

Owner

63.1.5.3 TCustomWebDataModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebDataModule (see page 6550) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebDataModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6551) to create a Web module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 6297)

Owner

63.1.5.4 TCustomWebDataModule.Destroy Destructor

Frees an instance of TCustomWebDataModule (see page 6550).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebDataModule();
```

Description

Typically, WebSnap applications do not explicitly free Web module instances. Instead, the FinishContext method of the Web application services object frees or deactivates Web modules automatically when the application has finished processing an HTTP request.

See Also

FinishContext

System::TObject::Free (see page 3372)

63.1.5.5 TCustomWebDataModule Methods

The methods of the TCustomWebDataModule class are listed here.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6553)	Responds to notifications that components are about to be inserted or removed.

63.1.5.5.1 TCustomWebDataModule.GetSiteModuleHelper Method

Returns the value of the SiteModuleHelper (see page 6555) property.

Pascal

```
function GetSiteModuleHelper: TSiteModuleHelper; virtual;
```

C++

```
virtual __fastcall TSiteModuleHelper GetSiteModuleHelper();
```

Description

GetSiteModuleHelper is the protected read implementation of the SiteModuleHelper (see page 6555) property.

See Also

SiteModuleHelper (see page 6555)

63.1.5.5.2 TCustomWebDataModule.GetWebSnapObjClass Method

Returns the class type for this adapter field.

Pascal

```
function GetWebSnapObjClass: TObjectScripterClass; virtual;
```

C++

```
virtual __fastcall TObjectScripterClass GetWebSnapObjClass();
```

Description

This function returns the class type for this adapter field, which is used by the script engine to create a script object. GetWebSnapObjClass tells the script engine what type of object to create.

63.1.5.5.3 TCustomWebDataModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebDataModule (

see page 6550) overrides this method to pass the notification on to the object that is the value of the SiteModuleHelper (see page 6555) property.

See Also

SiteModuleHelper (see page 6555)

63.1.5.6 TCustomWebDataModule Properties

The properties of the TCustomWebDataModule class are listed here.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.

Public Properties

	Name	Description
	Request (see page 6554)	Provides access to the current request object.
	Response (see page 6554)	Provides access to the current response object.
	Session (see page 6555)	Provides access to the current Web session.

63.1.5.6.1 TCustomWebDataModule.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3634) Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 6554)

63.1.5.6.2 TCustomWebDataModule.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3634) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 6554) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 6554)

63.1.5.6.3 TCustomWebDataModule.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Read (see page 3634) Session to obtain information from or add information to the current Web session. All requests are assigned a Web session, although you can only use the session to store information about the current user if the WebSnap application includes a TSessionsService component. Web sessions persist over multiple calls from the same end user until they are terminated (for example, when the user logs out or when a timeout period elapses).

See Also

TSessionsService (see page 6756)

63.1.5.6.4 TCustomWebDataModule.SiteModuleHelper Property

References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.

Pascal

```
property SiteModuleHelper: TSiteModuleHelper;
```

C++

```
__property TSiteModuleHelper SiteModuleHelper;
```

Description

Applications can't use the protected SiteModuleHelper property. TCustomWebDataModule (see page 6550) uses this property internally to delegate tasks involved in managing the WebSnap components it contains and to generate the OnActivate (see page 6556) and OnDeactivate (see page 6556) events when the Web module is activated and deactivated.

See Also

OnActivate (see page 6556)

OnDeactivate (see page 6556)

63.1.5.7 TCustomWebDataModule Events

The events of the TCustomWebDataModule class are listed here.

Public Events

	Name	Description
	OnActivate (see page 6556)	Occurs when the Web module is activated.
	OnDeactivate (see page 6556)	Occurs when the Web module is deactivated.

63.1.5.7.1 TCustomWebDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write (see page 3707) an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6556)

CreateMode

63.1.5.7.2 TCustomWebDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write (see page 3707) an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

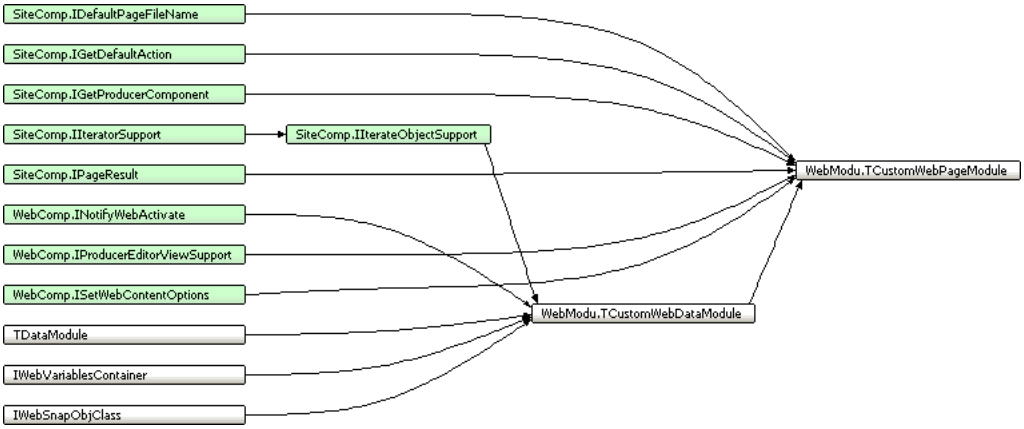
See Also

OnActivate (see page 6556)

63.1.6 TCustomWebPageModule Class

TCustomWebPageModule is the base class for WebSnap modules that generate a Web page.

Class Hierarchy



Pascal

```
TCustomWebPageModule = class(TCustomWebDataModule, IDefaultPageFileName, IPageResult,
ISetWebContentOptions, IGetDefaultAction, IGetProducerComponent, IProducerEditorViewSupport);
```

C++

```
class TCustomWebPageModule : public TCustomWebDataModule, public IDefaultPageFileName, public
IPageResult, public ISetWebContentOptions, public IGetDefaultAction, public
IGetProducerComponent, public IProducerEditorViewSupport;
```

File

WebModu

Description

TCustomWebPageModule is the common base class for Web modules that generate a Web page for a WebSnap application. This class introduces two new properties:

PageProducer (see page 6562), which is a page producer component that generates the Web page associated with this module.

DefaultAction (see page 6561), which is an adapter action associated with the module's Web page. Typically, the action sets up parameters that appear in HTTP requests for the Web page.

TCustomWebPageModule also introduces events that occur when the WebSnap application calls on the Web page module to generate its page or when it generates a redirect message to redirect a request to the associated page.

Do not create instances of TCustomWebPageModule. It is intended only as a base class. Instead, use the Web module wizard or the WebSnap application wizard to add a TCustomWebPageModule descendant to your WebSnap application. Descendant classes include TWebAppPageModule (see page 6588) and TWebPageModule (see page 6603).

See Also

TWebPageModule (see page 6603)

TWebAppPageModule (see page 6588)

63.1.6.1 TCustomWebPageModule Members

The following tables list the members exposed by TCustomWebPageModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6560)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSitePageModuleHelper (see page 6561)	Returns the value of the SitePageModuleHelper (see page 6562) property.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6561)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.
	SitePageModuleHelper (see page 6562)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnActivate (see page 6556)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6563)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 6563)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6564)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 6564)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6556)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	Create (see page 6559)	Creates and initializes an instance of TCustomWebPageModule (see page 6556).
	CreateNew (see page 6559)	Creates and initializes an instance of TCustomWebPageModule (see page 6556) without using an associated form file.
	Destroy (see page 6560)	Frees an instance of TCustomWebPageModule (see page 6556).
	DispatchPage (see page 1065)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 1028)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 1035)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1065)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RedirectToPage (see page 1066)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 1015)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	DefaultAction (see page 6561)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6562)	Indicates the interface of the page producer that generates the content of the Web page.
R	Request (see page 6554)	Provides access to the current request object.
R	Response (see page 6554)	Provides access to the current response object.
R	Session (see page 6555)	Provides access to the current Web session.

63.1.6.2 TCustomWebPageModule.Create Constructor

Creates and initializes an instance of TCustomWebPageModule (see page 6556).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebPageModule(TComponent * AOwner);
```

Description

Do not call the TCustomWebPageModule (see page 6556) constructor. Instead, use a Web page module factory object to create Web page module instances.

AOwner is the component that becomes the value of the Owner property.

See Also

TWebAppPageModuleFactory (see page 6297)

TWebPageModuleFactory (see page 6303)

Owner

63.1.6.3 TCustomWebPageModule.CreateNew Constructor

Creates and initializes an instance of TCustomWebPageModule (see page 6556) without using an associated form file.

Pascal

```
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0); override;
```

C++

```
virtual __fastcall TCustomWebPageModule(TComponent * AOwner, int Dummy = 0);
```

Description

Use CreateNew instead of Create (see page 6559) to create a Web page module without using the associated form file to initialize it.

AOwner is the component that becomes the value of the Owner property.

Dummy is not used.

See Also

TWebAppPageModuleFactory (see page 6297)

Owner

63.1.6.4 TCustomWebPageModule.Destroy Destructor

Frees an instance of TCustomWebPageModule (see page 6556).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebPageModule();
```

Description

Typically, WebSnap applications do not explicitly free Web page module instances. Instead, the FinishContext method of the Web application services object frees or deactivates Web modules automatically when the application has finished processing an HTTP request.

See Also

FinishContext

System::TObject::Free (see page 3372)

63.1.6.5 TCustomWebPageModule Methods

The methods of the TCustomWebPageModule class are listed here.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6560)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSitePageModuleHelper (see page 6561)	Returns the value of the SitePageModuleHelper (see page 6562) property.
	Notification (see page 6561)	Responds to notifications that components are about to be inserted or removed.

63.1.6.5.1 TCustomWebPageModule.GetSiteModuleHelper Method

Returns the value of the SiteModuleHelper (see page 6555) property.

Pascal

```
function GetSiteModuleHelper: TSiteModuleHelper; override;
```

C++

```
virtual __fastcall TSiteModuleHelper GetSiteModuleHelper();
```

Description

GetSiteModuleHelper is the protected read implementation of the SiteModuleHelper (see page 6555) property. TCustomWebPageModule (see page 6556) overrides the inherited method to return the value of the SitePageModuleHelper (see page 6562) property.

See Also

SiteModuleHelper (see page 6555)

SitePageModuleHelper (see page 6562)

63.1.6.5.2 TCustomWebPageModule.GetSitePageModuleHelper Method

Returns the value of the SitePageModuleHelper (see page 6562) property.

Pascal

```
function GetSitePageModuleHelper: TSitePageModuleHelper; virtual;
```

C++

```
virtual __fastcall TSitePageModuleHelper GetSitePageModuleHelper();
```

Description

GetSitePageModuleHelper is the protected read implementation of the SitePageModuleHelper (see page 6562) property.

See Also

SitePageModuleHelper (see page 6562)

63.1.6.5.3 TCustomWebPageModule.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TCustomWebPageModule (see page 6556) overrides this method to update the PageProducer (see page 6562) and DefaultAction (see page 6561) properties as appropriate.

See Also

PageProducer (see page 6562)

DefaultAction (see page 6561)

63.1.6.6 TCustomWebPageModule Properties

The properties of the TCustomWebPageModule class are listed here.

Protected Properties

	Name	Description
	SitePageModuleHelper (see page 6562)	References the helper object that assists in assembling the module's Web page.

Public Properties

	Name	Description
	DefaultAction (see page 6561)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6562)	Indicates the interface of the page producer that generates the content of the Web page.

63.1.6.6.1 TCustomWebPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TAdapterActionName;
```

C++

```
__property TAdapterActionName DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6562)

63.1.6.6.2 TCustomWebPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 1136)

TDataSetPageProducer

TXSLPageProducer (see page 7893)

TPageProducer

TDataSetTableProducer

TQueryTableProducer

TInetXPageProducer

63.1.6.6.3 TCustomWebPageModule.SitePageModuleHelper Property

References the helper object that assists in assembling the module's Web page.

Pascal

```
property SitePageModuleHelper: TSitePageModuleHelper;
```

C++

```
__property TSitePageModuleHelper SitePageModuleHelper;
```

Description

Applications can't use the protected SitePageModuleHelper property. TCustomWebPageModule (see page 6556) uses this property internally to delegate tasks involved in assembling the content of its Web page and in interacting with editor views at design time.

63.1.6.7 TCustomWebPageModule Events

The events of the TCustomWebPageModule class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6563)	Occurs after the page module generates its Web page.
	OnAfterRedirectToPage (see page 6563)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6564)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnBeforeRedirectToPage (see page 6564)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

63.1.6.7.1 TCustomWebPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write (see page 3707) an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 6564)

PageProducer (see page 6562)

Response (see page 6554)

63.1.6.7.2 TCustomWebPageModule.OnAfterRedirectToPage Event

Occurs after the Web page module assembles a response message that represents a redirection to its Web page.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

Write (see page 3707) an OnAfterRedirectToPage event handler to take any final action before returns a response message that redirects the user to this Web page. For example, the OnAfterRedirectToPage event handler can make adjustments to the headers of the response message.

This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost) to this page.

See Also

OnBeforeRedirectToPage (see page 6564)

OnAfterDispatchPage (see page 6563)

RedirectOptions (see page 5476)

Response (see page 6554)

63.1.6.7.3 TCustomWebPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write (see page 3707) an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request (see page 6554) and Response (see page 6554) properties.

See Also

OnAfterDispatchPage (see page 6563)

Request (see page 6554)

Response (see page 6554)

PageProducer (see page 6562)

63.1.6.7.4 TCustomWebPageModule.OnBeforeRedirectToPage Event

Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

Write (see page 3707) an OnBeforeRedirectToPage event handler to perform any tasks necessary for redirecting a request to the associated Web page. For example, the OnBeforeRedirectToPage event handler can add parameters to the Params parameter so that they appear in the request that results from the redirection. This event only occurs if an adapter in the WebSnap application executes an adapter action whose options include a redirection (roRedirectHTTPGet or roRedirectHTTPPost).

If the event handler sets Handled to true, the Web page module assumes it has handled the redirection and performs no further processing. Otherwise, after the event handler exits, the Web page module sets up the response message to encode the redirection parameters and calls SendRedirect.

See Also

OnAfterRedirectToPage (see page 6563)

OnBeforeDispatchPage (see page 6564)

SendRedirect

Request (see page 6554)

Response (see page 6554)

63.1.7 TSiteModuleHelper Class

TSiteModuleHelper manages common tasks for Web modules in a WebSnap application.

Class Hierarchy

Pascal

```
TSiteModuleHelper = class(TObject);
```

C++

```
class TSiteModuleHelper : public TObject;
```

File

WebModu

Description

Web modules create and use an instance of TSiteModuleHelper to keep track of WebSnap objects contained in the module and to generate OnActivate (see page 6573) and OnDeactivate (see page 6573) events when the Web module is activated and deactivated.

You should not need to work directly with TSiteModuleHelper instances. The Web module creates and uses this object.

See Also

TCustomWebDataModule (see page 6550)

63.1.7.1 TSiteModuleHelper Members

The following tables list the members exposed by TSiteModuleHelper.

Protected Methods

	Name	Description
	EndIterator (see page 6568)	This is EndIterator, a member of class TSiteModuleHelper.
	FindNamedVariable (see page 6568)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6569)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6569)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6569)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetVariable (see page 6569)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6569)	This is GetVariableCount, a member of class TSiteModuleHelper.
	NextIteration (see page 6570)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6570)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6570)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	StartIterator (see page 6571)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
	Module (see page 6571)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6573)	Occurs when the Web module is activated.
	OnDeactivate (see page 6573)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6567)	Creates and initializes an instance of TSiteModuleHelper (see page 6565).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 6567)	Frees an instance of TSiteModuleHelper (see page 6565).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.

	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6570)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Request (see page 6571)	Provides access to the current request object.
R	Response (see page 6572)	Provides access to the current response object.
R	Session (see page 6572)	Provides access to the current Web session.
R	WebModuleContext (see page 6572)	Keeps track of name/value pairs that store information for the Web module.

63.1.7.2 TSiteModuleHelper.Create Constructor

Creates and initializes an instance of TSiteModuleHelper (see page 6565).

Pascal

```
constructor Create(AModule: TComponent); virtual;
```

C++

```
virtual __fastcall TSiteModuleHelper(TComponent * AModule);
```

Description

Web modules call the TSiteModuleHelper (see page 6565) constructor from their own constructors.

AModule is the Web module that is using the TSiteModuleHelper (see page 6565) object to keep track of its components and activation events. The TSiteModuleHelper (see page 6565) object becomes the value of the module's SiteModuleHelper property.

See Also

SiteModuleHelper (see page 6555)

63.1.7.3 TSiteModuleHelper.Destroy Destructor

Frees an instance of TSiteModuleHelper (see page 6565).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSiteModuleHelper();
```

Description

Do not free TSiteModuleHelper (see page 6565) instances. This object is freed by the Web module that uses it.

See Also

System::TObject::Free (see page 3372)

63.1.7.4 TSiteModuleHelper Methods

The methods of the TSiteModuleHelper class are listed here.

Protected Methods

	Name	Description
	EndIterator (see page 6568)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 6568)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6569)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6569)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6569)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetVariable (see page 6569)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6569)	This is GetVariableCount, a member of class TSiteModuleHelper.
	NextIteration (see page 6570)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6570)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6570)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	StartIterator (see page 6571)	This is StartIterator, a member of class TSiteModuleHelper.

Public Methods

	Name	Description
	Notification (see page 6570)	Responds to notifications that components are about to be inserted or removed.

63.1.7.4.1 TSiteModuleHelper.EndIterator Method

Pascal

```
procedure EndIterator(OwnerData: Pointer);
```

C++

```
__fastcall EndIterator(void * OwnerData);
```

Description

This is EndIterator, a member of class TSiteModuleHelper.

63.1.7.4.2 TSiteModuleHelper.FindNamedVariable Method

Pascal

```
function FindNamedVariable(const AName: string): TComponent; virtual;
```

C++

```
virtual __fastcall TComponent * FindNamedVariable(const UnicodeString AName);
```

Description

This is FindNamedVariable, a member of class TSiteModuleHelper.

63.1.7.4.3 TSiteModuleHelper.FindVariable Method

Pascal

```
function FindVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariable(const UnicodeString AName);
```

Description

This is FindVariable, a member of class TSiteModuleHelper.

63.1.7.4.4 TSiteModuleHelper.FindVariableContainer Method

Pascal

```
function FindVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindVariableContainer(const UnicodeString AName);
```

Description

This is FindVariableContainer, a member of class TSiteModuleHelper.

63.1.7.4.5 TSiteModuleHelper.GetCurrentObject Method

Pascal

```
function GetCurrentObject(OwnerData: Pointer): TObject;
```

C++

```
__fastcall TObject * GetCurrentObject(void * OwnerData);
```

Description

This is GetCurrentObject, a member of class TSiteModuleHelper.

63.1.7.4.6 TSiteModuleHelper.GetVariable Method

Pascal

```
function GetVariable(I: Integer): TComponent;
```

C++

```
__fastcall TComponent * GetVariable(int I);
```

Description

This is GetVariable, a member of class TSiteModuleHelper.

63.1.7.4.7 TSiteModuleHelper.GetVariableCount Method

Pascal

```
function GetVariableCount: Integer;
```

C++

```
__fastcall int GetVariableCount();
```

Description

This is GetVariableCount, a member of class TSiteModuleHelper.

63.1.7.4.8 TSiteModuleHelper.NextIteration Method

Pascal

```
function NextIteration(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean NextIteration(void * OwnerData);
```

Description

This is NextIteration, a member of class TSiteModuleHelper.

63.1.7.4.9 TSiteModuleHelper.Notification Method

Responds to notifications that components are about to be inserted or removed.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation);
```

C++

```
__fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Notification is called automatically when components are added to or removed from the application. TSiteModuleHelper (see page 6565) overrides this method to ensure that any WebSnap objects added to the associated module receive notifications when the module is activated or deactivated.

63.1.7.4.10 TSiteModuleHelper.NotifyActivate Method

Pascal

```
procedure NotifyActivate;
```

C++

```
__fastcall NotifyActivate();
```

Description

This is NotifyActivate, a member of class TSiteModuleHelper.

63.1.7.4.11 TSiteModuleHelper.NotifyDeactivate Method

Pascal

```
procedure NotifyDeactivate;
```

C++

```
__fastcall NotifyDeactivate();
```

Description

This is NotifyDeactivate, a member of class TSiteModuleHelper.

63.1.7.4.12 TSiteModuleHelper.StartIterator Method

Pascal

```
function StartIterator(var OwnerData: Pointer): Boolean;
```

C++

```
__fastcall Boolean StartIterator(void * OwnerData);
```

Description

This is StartIterator, a member of class TSiteModuleHelper.

63.1.7.5 TSiteModuleHelper Properties

The properties of the TSiteModuleHelper class are listed here.

Protected Properties

	Name	Description
	Module (see page 6571)	This is Module, a member of class TSiteModuleHelper.

Public Properties

	Name	Description
	Request (see page 6571)	Provides access to the current request object.
	Response (see page 6572)	Provides access to the current response object.
	Session (see page 6572)	Provides access to the current Web session.
	WebModuleContext (see page 6572)	Keeps track of name/value pairs that store information for the Web module.

63.1.7.5.1 TSiteModuleHelper.Module Property

Pascal

```
property Module: TComponent;
```

C++

```
__property TComponent * Module;
```

Description

This is Module, a member of class TSiteModuleHelper.

63.1.7.5.2 TSiteModuleHelper.Request Property

Provides access to the current request object.

Pascal

```
property Request: TWebRequest;
```

C++

```
__property TWebRequest Request;
```

Description

Read (see page 3634) Request to obtain information about the current HTTP request message. The request object provides access to the headers of the request message, as well as the content of the message.

See Also

Response (see page 6572)

63.1.7.5.3 TSiteModuleHelper.Response Property

Provides access to the current response object.

Pascal

```
property Response: TWebResponse;
```

C++

```
__property TWebResponse Response;
```

Description

Read (see page 3634) Response to obtain information about or add to the response message that the application builds as a reply to the message that the Request (see page 6571) property specifies. The response object provides access to the headers of the response message, as well as the content of the message.

See Also

Request (see page 6571)

63.1.7.5.4 TSiteModuleHelper.Session Property

Provides access to the current Web session.

Pascal

```
property Session: TAbstractWebSession;
```

C++

```
__property TAbstractWebSession Session;
```

Description

Read (see page 3634) Session to obtain information from or add information to the current Web session. All requests are assigned a Web session, although you can only use the session to store information about the current user if the WebSnap application includes a TSessionsService component. Web sessions persist over multiple calls from the same end user until they are terminated (for example, when the user logs out or when a time-out period elapses).

See Also

TSessionsService (see page 6756)

63.1.7.5.5 TSiteModuleHelper.WebModuleContext Property

Keeps track of name/value pairs that store information for the Web module.

Pascal

```
property WebModuleContext: TWebModuleContext;
```

C++

```
__property TWebModuleContext WebModuleContext;
```

Description

Use WebModuleContext to access the TWebModuleContext object. The TWebModuleContext object maintains a list of named variables that the components in the Web module can use.

See Also

WebModuleContext

63.1.7.6 TSiteModuleHelper Events

The events of the TSiteModuleHelper class are listed here.

Public Events

	Name	Description
	OnActivate (see page 6573)	Occurs when the Web module is activated.
	OnDeactivate (see page 6573)	Occurs when the Web module is deactivated.

63.1.7.6.1 TSiteModuleHelper.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

OnActivate uses the same event handler as that of the Web module's OnActivate event. The TSiteModuleHelper (see page 6565) object generates this event when the Web module is first activated.

See Also

OnDeactivate (see page 6573)

OnActivate (see page 6556)

63.1.7.6.2 TSiteModuleHelper.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

OnDeactivate uses the same event handler as that of the Web module's OnDeactivate event. The TSiteModuleHelper (see page 6565) object generates this event when the Web module is deactivated.

See Also

OnActivate (see page 6573)

OnDeactivate (see page 6556)

63.1.8 TSitePageModuleHelper Class

TSitePageModuleHelper manages common tasks for Web page modules in a WebSnap application.

Class Hierarchy

Pascal

```
TSitePageModuleHelper = class(TSiteModuleHelper);
```

C++

```
class TSitePageModuleHelper : public TSiteModuleHelper;
```

File

WebModu

Description

Web page modules create and use an instance of TSitePageModuleHelper to assist in the following tasks:

Assembling the content of the associated Web page.

Interacting with editor views at design time.

Keeping track of WebSnap objects contained in the module.

Generating the events that occur when the Web module is activated or deactivated, and when it dispatches or redirects a request for a Web page.

You should not need to work directly with TSitePageModuleHelper instances. The Web page module creates and uses this object.

See Also

TCustomWebPageModule (see page 6556)

TCustomWebDataModule (see page 6550)

63.1.8.1 TSitePageModuleHelper Members

The following tables list the members exposed by TSitePageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 6577)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 6568)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 6568)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6569)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6569)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6569)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 6578)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 6578)	This is GetPageProducer, a member of class TSitePageModuleHelper.
	GetProducerComponent (see page 6578)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6578)	This is GetRedirectParams, a member of class TSitePageModuleHelper.

	GetTemplateFileType (see page 6579)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6579)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 6569)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6569)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 6579)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6579)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6579)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6580)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6580)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 6580)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6580)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 6581)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 6581)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 6581)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 6581)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6582)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 6570)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6570)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6570)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 6582)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6582)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6582)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 6571)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 6571)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6573)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6583)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6584)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6584)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6584)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6573)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
V	Create (see page 6576)	Creates and initializes an instance of TSitePageModuleHelper (see page 6574).
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6577)	Frees an instance of TSitePageModuleHelper (see page 6574).

	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6570)	Responds to notifications that components are about to be inserted or removed.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultAction (see page 6583)	Represents the adapter action associated with the Web page module.
R	Request (see page 6571)	Provides access to the current request object.
R	Response (see page 6572)	Provides access to the current response object.
R	Session (see page 6572)	Provides access to the current Web session.
R	WebModuleContext (see page 6572)	Keeps track of name/value pairs that store information for the Web module.

63.1.8.2 TSitePageModuleHelper.Create Constructor

Creates and initializes an instance of TSitePageModuleHelper (see page 6574).

Pascal

```
constructor Create(AModule: TComponent); override;
```

C++

```
virtual __fastcall TSitePageModuleHelper(TComponent * AModule);
```

Description

Web page modules call the TSitePageModuleHelper (see page 6574) constructor from their own constructors.

AModule is the Web page module that is using the TSitePageModuleHelper (see page 6574) object to generate events and help assemble its Web content. The TSitePageModuleHelper (see page 6574) object becomes the value of the module's SitePageModuleHelper property.

See Also

SitePageModuleHelper (see page 6562)

63.1.8.3 TSitePageModuleHelper.Destroy Destructor

Frees an instance of TSitePageModuleHelper (see page 6574).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TSitePageModuleHelper();
```

Description

Do not free TSitePageModuleHelper (see page 6574) instances. This object is freed by the Web page module that uses it.

See Also

System::TObject::Free (see page 3372)

63.1.8.4 TSitePageModuleHelper Methods

The methods of the TSitePageModuleHelper class are listed here.

Protected Methods

	Name	Description
	DispatchPage (see page 6577)	This is DispatchPage, a member of class TSitePageModuleHelper.
	GetDefaultAction (see page 6578)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 6578)	This is GetPageProducer, a member of class TSitePageModuleHelper.
	GetProducerComponent (see page 6578)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6578)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 6579)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6579)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetXMLData (see page 6579)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6579)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6579)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6580)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6580)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 6580)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6580)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 6581)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 6581)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 6581)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 6581)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6582)	This is IncludePage, a member of class TSitePageModuleHelper.
	RedirectToPage (see page 6582)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6582)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6582)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.

63.1.8.4.1 TSitePageModuleHelper.DispatchPage Method

Pascal

```
function DispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean DispatchPage(const UnicodeString APageName, TWebResponse AWebResponse);
```

Description

This is DispatchPage, a member of class TSitePageModuleHelper.

63.1.8.4.2 TSitePageModuleHelper.GetDefaultAction Method

Pascal

```
function GetDefaultAction: TComponent;
```

C++

```
__fastcall TComponent * GetDefaultAction();
```

Description

This is GetDefaultAction, a member of class TSitePageModuleHelper.

63.1.8.4.3 TSitePageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; virtual;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

Description

This is GetPageProducer, a member of class TSitePageModuleHelper.

63.1.8.4.4 TSitePageModuleHelper.GetProducerComponent Method

Pascal

```
function GetProducerComponent: TComponent;
```

C++

```
__fastcall TComponent * GetProducerComponent();
```

Description

This is GetProducerComponent, a member of class TSitePageModuleHelper.

63.1.8.4.5 TSitePageModuleHelper.GetRedirectParams Method

Pascal

```
procedure GetRedirectParams(const APageName: string; AParams: TStrings);
```

C++

```
__fastcall GetRedirectParams(const UnicodeString APageName, TStrings AParams);
```

Description

This is GetRedirectParams, a member of class TSitePageModuleHelper.

63.1.8.4.6 TSitePageModuleHelper.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class TSitePageModuleHelper.

63.1.8.4.7 TSitePageModuleHelper.GetTemplateStream Method

Pascal

```
function GetTemplateStream(const APage: string; out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetTemplateStream(const UnicodeString APage, Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TSitePageModuleHelper.

63.1.8.4.8 TSitePageModuleHelper.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class TSitePageModuleHelper.

63.1.8.4.9 TSitePageModuleHelper.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class TSitePageModuleHelper.

63.1.8.4.10 TSitePageModuleHelper.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.

63.1.8.4.11 TSitePageModuleHelper.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class TSitePageModuleHelper.

63.1.8.4.12 TSitePageModuleHelper.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class TSitePageModuleHelper.

63.1.8.4.13 TSitePageModuleHelper.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class TSitePageModuleHelper.

63.1.8.4.14 TSitePageModuleHelper.HTMLFromTemplate Method

Pascal

```
function HTMLFromTemplate(AStream: TStream): string;
```

C++

```
__fastcall UnicodeString HTMLFromTemplate(TStream AStream);
```

Description

This is HTMLFromTemplate, a member of class TSitePageModuleHelper.

63.1.8.4.15 TSitePageModuleHelper.ImplDispatchPage Method

Pascal

```
function ImplDispatchPage(const APageName: string; AWebResponse: TWebResponse): Boolean;
virtual;
```

C++

```
virtual __fastcall Boolean ImplDispatchPage(const UnicodeString APageName, TWebResponse
AWebResponse);
```

Description

This is ImplDispatchPage, a member of class TSitePageModuleHelper.

63.1.8.4.16 TSitePageModuleHelper.ImplIncludePage Method

Pascal

```
function ImplIncludePage(const APageName: string; var AOwned: Boolean): TStream; virtual;
```

C++

```
virtual __fastcall TStream ImplIncludePage(const UnicodeString APageName, Boolean AOwned);
```

Description

This is ImplIncludePage, a member of class TSitePageModuleHelper.

63.1.8.4.17 TSitePageModuleHelper.ImplRedirectToPage Method

Pascal

```
function ImplRedirectToPage(const APageName: string; AParams: TStrings; AResponse:
TWebResponse): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean ImplRedirectToPage(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse);
```

Description

This is ImplRedirectToPage, a member of class TSitePageModuleHelper.

63.1.8.4.18 TSitePageModuleHelper.ImplSetWebContentOptions Method

Pascal

```
procedure ImplSetWebContentOptions(AOptions: TWebContentOptions); virtual;
```

C++

```
virtual __fastcall ImplSetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.

63.1.8.4.19 TSitePageModuleHelper.IncludePage Method

Pascal

```
function IncludePage(const APageName: string; var AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream IncludePage(const UnicodeString APageName, Boolean AOwned);
```

Description

This is IncludePage, a member of class TSitePageModuleHelper.

63.1.8.4.20 TSitePageModuleHelper.RedirectToPage Method

Pascal

```
function RedirectToPage(const APageName: string; AParams: TStrings; AResponse: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean RedirectToPage(const UnicodeString APageName, TStrings AParams,
TWebResponse AResponse);
```

Description

This is RedirectToPage, a member of class TSitePageModuleHelper.

63.1.8.4.21 TSitePageModuleHelper.SetDefaultPageFileName Method

Pascal

```
procedure SetDefaultPageFileName(const AValue: string);
```

C++

```
__fastcall SetDefaultPageFileName(const UnicodeString AValue);
```

Description

This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.

63.1.8.4.22 TSitePageModuleHelper.SetWebContentOptions Method

Pascal

```
procedure SetWebContentOptions(AOptions: TWebContentOptions);
```

C++

```
__fastcall SetWebContentOptions(TWebContentOptions AOptions);
```

Description

This is SetWebContentOptions, a member of class TSitePageModuleHelper.

63.1.8.5 TSitePageModuleHelper Properties

The properties of the TSitePageModuleHelper class are listed here.

Public Properties

	Name	Description
	DefaultAction (see page 6583)	Represents the adapter action associated with the Web page module.

63.1.8.5.1 TSitePageModuleHelper.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TAdapterActionName;
```

C++

```
__property TAdapterActionName DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6562)

63.1.8.6 TSitePageModuleHelper Events

The events of the TSitePageModuleHelper class are listed here.

Public Events

	Name	Description
	OnAfterDispatchPage (see page 6583)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6584)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6584)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6584)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

63.1.8.6.1 TSitePageModuleHelper.OnAfterDispatchPage Event

Occurs after the Web page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

OnAfterDispatchPage uses the same event handler as that of the Web page module's OnAfterDispatchPage event. The TSitePageModuleHelper (see page 6574) object generates this event after the Web page module writes the content for its Web page to the response object.

See Also

OnDeactivate (see page 6573)

OnAfterDispatchPage (see page 6563)

63.1.8.6.2 TSitePageModuleHelper.OnAfterRedirectToPage Event

Occurs after the Web page module assembles a response message that represents a redirection to its Web page.

Pascal

```
property OnAfterRedirectToPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterRedirectToPage;
```

Description

OnAfterRedirectToPage uses the same event handler as that of the Web page module's OnAfterRedirectToPage event. The TSitePageModuleHelper (see page 6574) object generates this event after the Web page module sends a response redirecting the caller to its Web page.

See Also

OnBeforeRedirectToPage (see page 6584)

OnAfterRedirectToPage (see page 6563)

63.1.8.6.3 TSitePageModuleHelper.OnBeforeDispatchPage Event

Occurs before the Web module generates the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

OnBeforeDispatchPage uses the same event handler as that of the Web page module's OnBeforeDispatchPage event. The TSitePageModuleHelper (see page 6574) object generates this event before the Web page module generates the content for its Web page.

See Also

OnAfterDispatchPage (see page 6583)

OnBeforeDispatchPage (see page 6564)

63.1.8.6.4 TSitePageModuleHelper.OnBeforeRedirectToPage Event

Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Pascal

```
property OnBeforeRedirectToPage: TDispatchPageParamsHandledEvent;
```

C++

```
__property TDispatchPageParamsHandledEvent OnBeforeRedirectToPage;
```

Description

OnBeforeRedirectToPage uses the same event handler as that of the Web page module's OnBeforeRedirectToPage event. The TSitePageModuleHelper (see page 6574) object generates this event before the Web page module assembles and sends a response redirecting the caller to its Web page.

See Also

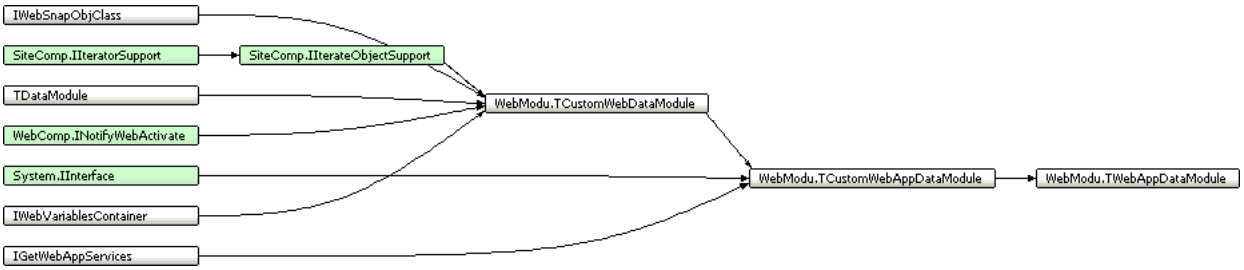
OnAfterRedirectToPage (see page 6584)

OnBeforeRedirectToPage (see page 6564)

63.1.9 TWebAppDataModule Class

TWebAppDataModule is used as the WebSnap application module when there is no need to generate a page.

Class Hierarchy



Pascal

```
TWebAppDataModule = class(TCustomWebAppDataModule);
```

C++

```
class TWebAppDataModule : public TCustomWebAppDataModule;
```

File

WebModu

Description

TWebAppDataModule is a container for WebSnap system components such as dispatchers, global adapters, user list, and session components. The WebSnap Application Wizard automatically adds system components to the TWebAppDataModule when it creates a TWebAppDataModule. System (see page 3159) components may be added or removed later. TWebAppComponents (see page 6243) is the component in a TWebAppDataModule that references all of the system components in use by the WebSnap application.

See Also

Web Modules

63.1.9.1 TWebAppDataModule Members

The following tables list the members exposed by TWebAppDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetWebAppServices (see page 6542)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6543)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6553)	Responds to notifications that components are about to be inserted or removed.
	QueryInterface (see page 6543)	Returns a reference to a specified interface if the module supports that interface.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	Create (see page 6541)	Creates a new TCustomWebAppDataModule (see page 6539) instance.
	Create (see page 6551)	Creates and initializes an instance of TCustomWebDataModule (see page 6550).
	CreateNew (see page 6541)	Creates a data module without using the associated form file to initialize it.
	CreateNew (see page 6552)	Creates and initializes an instance of TCustomWebDataModule (see page 6550) without using an associated form file.
	Destroy (see page 6542)	Frees a Web application data module instance.
	Destroy (see page 6552)	Frees an instance of TCustomWebDataModule (see page 6550).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	Request (see page 6554)	Provides access to the current request object.
	Response (see page 6554)	Provides access to the current response object.
	Session (see page 6555)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6587)	Occurs when the Web module is activated.
	OnDeactivate (see page 6588)	Occurs when the Web module is deactivated.

Published Properties

	Name	Description
	AppServices (see page 6587)	Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

63.1.9.2 TWebAppDataModule Properties

The properties of the TWebAppDataModule class are listed here.

Published Properties

	Name	Description
	AppServices (see page 6587)	Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

63.1.9.2.1 TWebAppDataModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppDataModule object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 6243)

WebContext (see page 6003)

63.1.9.3 TWebAppDataModule Events

The events of the TWebAppDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6587)	Occurs when the Web module is activated.
	OnDeactivate (see page 6588)	Occurs when the Web module is deactivated.

63.1.9.3.1 TWebAppDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6556)

CreateMode

63.1.9.3.2 TWebAppDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

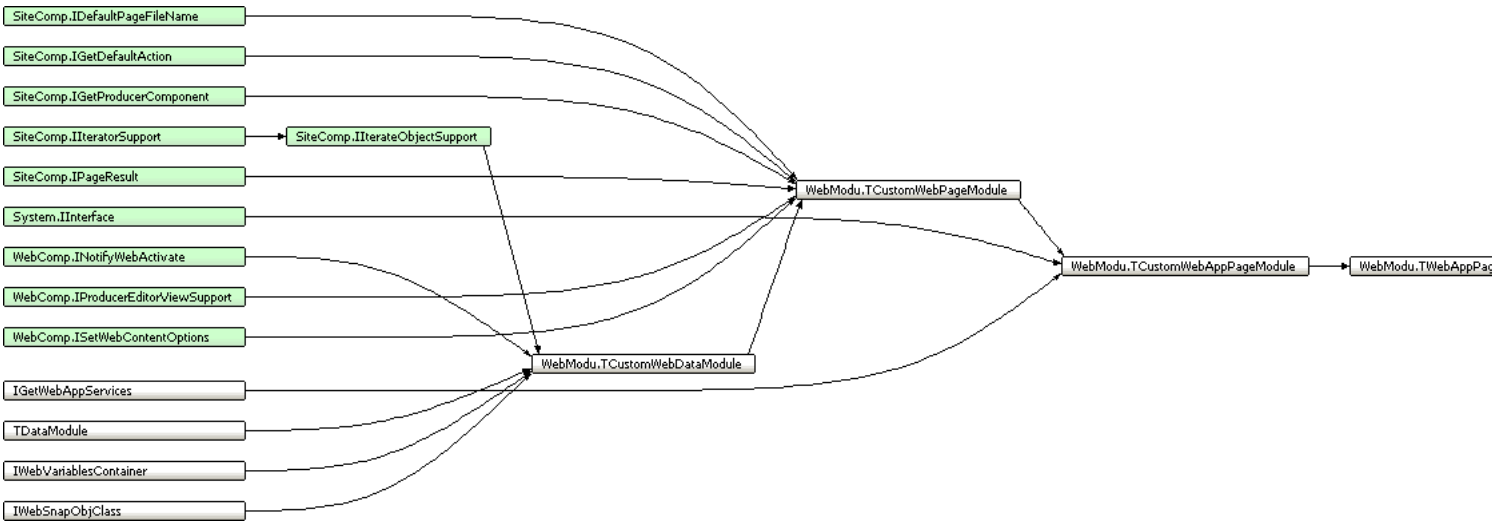
See Also

OnActivate (see page 6556)

63.1.10 TWebAppPageModule Class

TWebAppPageModule is used as the WebSnap application module when the application module generates a page.

Class Hierarchy



Pascal

```
TWebAppPageModule = class(TCustomWebAppPageModule);
```

C++

```
class TWebAppPageModule : public TCustomWebAppPageModule;
```

File

WebModu

Description

Like TWebAppDataModule (see page 6585), TWebAppPageModule is a container for WebSnap system components such as dispatchers, global adapters, user list, and session components. The WebSnap Application Wizard automatically adds system components to the TWebAppPageModule when it creates a TWebAppPageModule. System (see page 3159) components may be added or removed later. TWebAppComponents (see page 6243) is the component in a TWebAppPageModule that references all of the system components in use by the WebSnap application.

The WebAppPageModule is different from TWebAppDataModule (see page 6585) because it generates a page. The PageProducer (see page 6592) property references the producer component responsible for generating content. This property must not be nil (Delphi) or NULL (C++) or no content will be generated. The WebSnap Application Wizard automatically adds a page producer component when it creates a new TWebAppPageModule.

See Also

Web Modules

63.1.10.1 TWebAppPageModule Members

The following tables list the members exposed by TWebAppPageModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSiteModuleHelper (see page 6560)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSitePageModuleHelper (see page 6561)	Returns the value of the SitePageModuleHelper (see page 6562) property.
	GetWebAppServices (see page 6548)	Returns the IWebAppServices interface of the component that centralizes the dispatching of requests.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6548)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6553)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6561)	Responds to notifications that components are about to be inserted or removed.
	QueryInterface (see page 6548)	Returns a reference to a specified interface if the module supports that interface.
	SetAppServices (see page 6549)	Sets the value of the AppServices (see page 6549) property.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.
	SitePageModuleHelper (see page 6562)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 6563)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeRedirectToPage (see page 6564)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	Create (see page 6546)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6544).

	Create (see page 6551)	Creates and initializes an instance of TCustomWebDataModule (see page 6550).
	Create (see page 6559)	Creates and initializes an instance of TCustomWebPageModule (see page 6556).
	CreateNew (see page 6547)	Creates and initializes an instance of TCustomWebAppPageModule (see page 6544) without using an associated form file.
	CreateNew (see page 6552)	Creates and initializes an instance of TCustomWebDataModule (see page 6550) without using an associated form file.
	CreateNew (see page 6559)	Creates and initializes an instance of TCustomWebPageModule (see page 6556) without using an associated form file.
	Destroy (see page 6547)	Frees an instance of TCustomWebAppPageModule (see page 6544).
	Destroy (see page 6552)	Frees an instance of TCustomWebDataModule (see page 6550).
	Destroy (see page 6560)	Frees an instance of TCustomWebPageModule (see page 6556).
	DispatchPage (see page 1065)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 1028)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 1035)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1065)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RedirectToPage (see page 1066)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 1015)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	Request (see page 6554)	Provides access to the current request object.
	Response (see page 6554)	Provides access to the current response object.
	Session (see page 6555)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6592)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6593)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6593)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6594)	Occurs when the Web module is deactivated.

Published Properties

	Name	Description
	AppServices (see page 6591)	Provides read and write access to the application services stored in this TCustomWebAppPageModule object.
	DefaultAction (see page 6591)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6592)	Indicates the interface of the page producer that generates the content of the Web page.

63.1.10.2 TWebAppPageModule Properties

The properties of the TWebAppPageModule class are listed here.

Published Properties

	Name	Description
	AppServices (see page 6591)	Provides read and write access to the application services stored in this TCustomWebAppPageModule object.
	DefaultAction (see page 6591)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6592)	Indicates the interface of the page producer that generates the content of the Web page.

63.1.10.2.1 TWebAppPageModule.AppServices Property

Provides read and write access to the application services stored in this TCustomWebAppPageModule object.

Pascal

```
property AppServices: IWebAppServices;
```

C++

```
__property IWebAppServices AppServices;
```

Description

AppServices provides access to the application services implemented by a component in this Web application module. The application services component sets up the Web context for each incoming request message, passes incoming request objects on to a dispatcher for handling, and provides the final backstop for exceptions that arise while processing request messages.

Typically, this property references the interface of a TWebAppComponents object.

See Also

TWebAppComponents (see page 6243)

WebContext (see page 6003)

63.1.10.2.2 TWebAppPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TComponent;
```

C++

```
__property TAdapterActionName DefaultAction;__property TComponent * DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6562)

63.1.10.2.3 TWebAppPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 1136)

TDataSetPageProducer

TXSLPageProducer (see page 7893)

TPageProducer

TDataSetTableProducer

TQueryTableProducer

TInetXPageProducer

63.1.10.3 TWebAppPageModule Events

The events of the TWebAppPageModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6592)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6593)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6593)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6594)	Occurs when the Web module is deactivated.

63.1.10.3.1 TWebAppPageModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the

CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6556)

CreateMode

63.1.10.3.2 TWebAppPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 6564)

PageProducer (see page 6562)

Response (see page 6554)

63.1.10.3.3 TWebAppPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request and Response properties.

See Also

OnAfterDispatchPage (see page 6563)

Request (see page 6554)

Response (see page 6554)

PageProducer (see page 6562)

63.1.10.3.4 TWebAppPageModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 6556)

63.1.11 TWebAppPageModuleHelper Class

Class Hierarchy

Pascal

```
TWebAppPageModuleHelper = class(TSitePageModuleHelper);
```

C++

```
class TWebAppPageModuleHelper : public TSitePageModuleHelper;
```

File

WebModu

Description

This is class WebModu.TWebAppPageModuleHelper.

63.1.11.1 TWebAppPageModuleHelper Members

The following tables list the members exposed by TWebAppPageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 6577)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 6568)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 6568)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6569)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6569)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6569)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 6578)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 6597)	This is GetPageProducer, a member of class TWebAppPageModuleHelper.
	GetProducerComponent (see page 6578)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6578)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 6579)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6579)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 6569)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6569)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 6579)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6579)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6579)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6580)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6580)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 6580)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6580)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 6581)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.
V	ImplIncludePage (see page 6581)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
V	ImplRedirectToPage (see page 6581)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
V	ImplSetWebContentOptions (see page 6581)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6582)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 6570)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6570)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6570)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 6582)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6582)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6582)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 6571)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
R	Module (see page 6571)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6573)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6583)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6584)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6584)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6584)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6573)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.

	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 6567)	Creates and initializes an instance of TSiteModuleHelper (see page 6565).
	Create (see page 6576)	Creates and initializes an instance of TSitePageModuleHelper (see page 6574).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 6567)	Frees an instance of TSiteModuleHelper (see page 6565).
	Destroy (see page 6577)	Frees an instance of TSitePageModuleHelper (see page 6574).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6570)	Responds to notifications that components are about to be inserted or removed.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultAction (see page 6583)	Represents the adapter action associated with the Web page module.
R	Request (see page 6571)	Provides access to the current request object.
R	Response (see page 6572)	Provides access to the current response object.
R	Session (see page 6572)	Provides access to the current Web session.
R	WebModuleContext (see page 6572)	Keeps track of name/value pairs that store information for the Web module.

63.1.11.2 TWebAppPageModuleHelper Methods

The methods of the TWebAppPageModuleHelper class are listed here.

Protected Methods

	Name	Description
	GetPageProducer (see page 6597)	This is GetPageProducer, a member of class TWebAppPageModuleHelper.

63.1.11.2.1 TWebAppPageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; override;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

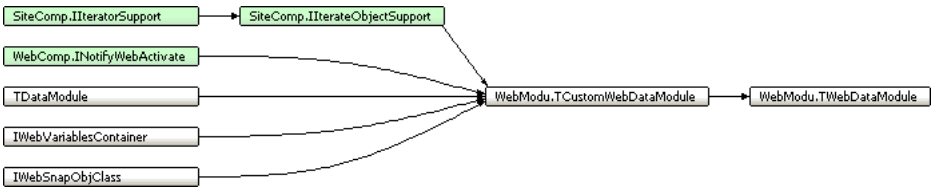
Description

This is GetPageProducer, a member of class TWebAppPageModuleHelper.

63.1.12 TWebDataModule Class

TWebDataModule is a container for components in a WebSnap application.

Class Hierarchy



Pascal

```
TWebDataModule = class(TCustomWebDataModule);
```

C++

```
class TWebDataModule : public TCustomWebDataModule;
```

File

WebModu

Description

Use TWebDataModule as a container for components that may be used by other modules in your WebSnap application. TWebDataModule is compatible with WebSnap applications. The initialization section of TWebDataModule registers a factory that is used by the WebSnap application to create the data module.

See Also

Web Modules

63.1.12.1 TWebDataModule Members

The following tables list the members exposed by TWebDataModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6553)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.

Public Methods

	Name	Description
	Create (see page 6551)	Creates and initializes an instance of TCustomWebDataModule (see page 6550).
	CreateNew (see page 6552)	Creates and initializes an instance of TCustomWebDataModule (see page 6550) without using an associated form file.
	Destroy (see page 6552)	Frees an instance of TCustomWebDataModule (see page 6550).
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	Request (see page 6554)	Provides access to the current request object.
	Response (see page 6554)	Provides access to the current response object.
	Session (see page 6555)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6598)	Occurs when the Web module is activated.
	OnDeactivate (see page 6599)	Occurs when the Web module is deactivated.

63.1.12.2 TWebDataModule Events

The events of the TWebDataModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6598)	Occurs when the Web module is activated.
	OnDeactivate (see page 6599)	Occurs when the Web module is deactivated.

63.1.12.2.1 TWebDataModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6556)

CreateMode

63.1.12.2.2 TWebDataModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 6556)

63.1.13 TWebNotifyContainer Class

Class Hierarchy

Pascal

```
TWebNotifyContainer = class(TObject);
```

C++

```
class TWebNotifyContainer : public TObject;
```

File

WebModu

Description

This is class WebModu.TWebNotifyContainer.

63.1.13.1 TWebNotifyContainer Members

The following tables list the members exposed by TWebNotifyContainer.

Protected Fields

	Name	Description
	FList (see page 6602)	This is FList, a member of class TWebNotifyContainer.

Protected Methods

	Name	Description
	GetCount (see page 6602)	This is GetCount, a member of class TWebNotifyContainer.

Public Methods

	Name	Description
	Add (see page 6602)	This is Add, a member of class TWebNotifyContainer.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6601)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 6601)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NotifyWebActivate (see page 6603)	This is NotifyWebActivate, a member of class TWebNotifyContainer.
	NotifyWebDeactivate (see page 6603)	This is NotifyWebDeactivate, a member of class TWebNotifyContainer.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

63.1.13.2 TWebNotifyContainer.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebNotifyContainer();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

63.1.13.3 TWebNotifyContainer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebNotifyContainer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

63.1.13.4 TWebNotifyContainer Fields

The fields of the TWebNotifyContainer class are listed here.

Protected Fields

	Name	Description
	FList (see page 6602)	This is FList, a member of class TWebNotifyContainer.

63.1.13.4.1 TWebNotifyContainer.FList Field

Pascal

```
FList: TComponentList;
```

C++

```
TComponentList FList;
```

Description

This is FList, a member of class TWebNotifyContainer.

63.1.13.5 TWebNotifyContainer Methods

The methods of the TWebNotifyContainer class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 6602)	This is GetCount, a member of class TWebNotifyContainer.

Public Methods

	Name	Description
	Add (see page 6602)	This is Add, a member of class TWebNotifyContainer.
	NotifyWebActivate (see page 6603)	This is NotifyWebActivate, a member of class TWebNotifyContainer.
	NotifyWebDeactivate (see page 6603)	This is NotifyWebDeactivate, a member of class TWebNotifyContainer.

63.1.13.5.1 TWebNotifyContainer.Add Method

Pascal

```
procedure Add(AComponent: TComponent);
```

C++

```
__fastcall Add(TComponent * AComponent);
```

Description

This is Add, a member of class TWebNotifyContainer.

63.1.13.5.2 TWebNotifyContainer.GetCount Method

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

This is GetCount, a member of class TWebNotifyContainer.

63.1.13.5.3 TWebNotifyContainer.NotifyWebActivate Method

Pascal

```
procedure NotifyWebActivate;
```

C++

```
__fastcall NotifyWebActivate();
```

Description

This is NotifyWebActivate, a member of class TWebNotifyContainer.

63.1.13.5.4 TWebNotifyContainer.NotifyWebDeactivate Method

Pascal

```
procedure NotifyWebDeactivate;
```

C++

```
__fastcall NotifyWebDeactivate();
```

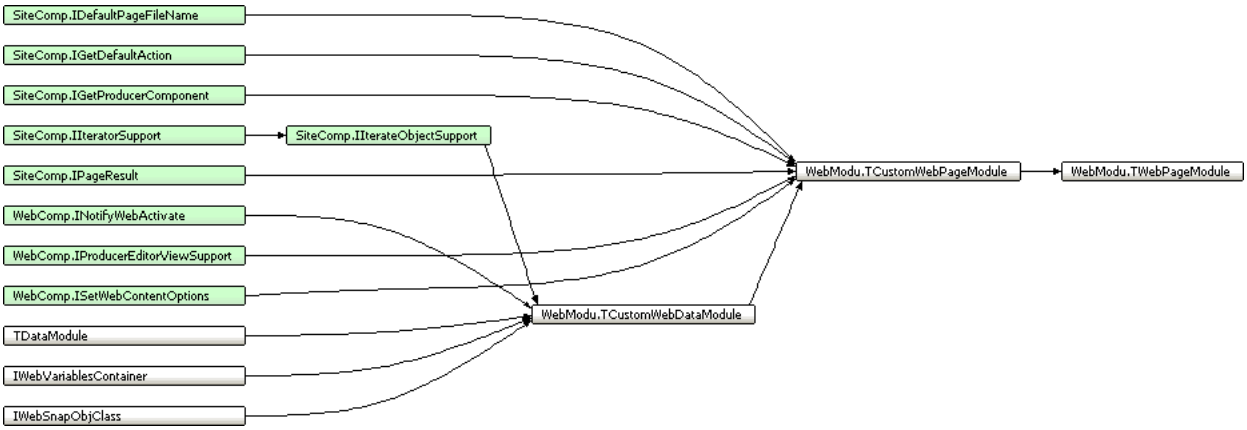
Description

This is NotifyWebDeactivate, a member of class TWebNotifyContainer.

63.1.14 TWebPageModule Class

TWebPageModule is a WebSnap module that generates a page.

Class Hierarchy



Pascal

```
TWebPageModule = class(TCustomWebPageModule);
```

C++

```
class TWebPageModule : public TCustomWebPageModule;
```

File

WebModu

Description

Use TWebPageModule in a WebSnap application to hold all the components needed for generating a specific Web page. When you use the WebSnap page module wizard to create a TWebPageModule descendant, the wizard adds a page producer to the module. The page producer generates the content of the Web page and adds code to the unit it creates (in Delphi, the code is added to the initialization section; in C++, it is added to a #pragma startup). The added code registers the TWebPageModule descendant with a factory object that the WebSnap application can use to create the Web page module.

The page producer that the wizard adds to the Web page module becomes the value of the PageProducer (see page 6606) property. If the PageProducer (see page 6606) property is nil (Delphi) or NULL (C++), the Web page module can't generate the content of a Web page. Add additional components to the Web page module that assist the page producer in creating its content. Other modules in the WebSnap application can access any components you add to the Web page module, but typically Web page module components are intended only for internal use.

See Also

TWebAppDataModule (see page 6585)

TWebAppPageModule (see page 6588)

TWebDataModule (see page 6597)

TWebPageModuleFactory (see page 6303)

Web Modules

63.1.14.1 TWebPageModule Members

The following tables list the members exposed by TWebPageModule.

Protected Methods

	Name	Description
	GetSiteModuleHelper (see page 6553)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSiteModuleHelper (see page 6560)	Returns the value of the SiteModuleHelper (see page 6555) property.
	GetSitePageModuleHelper (see page 6561)	Returns the value of the SitePageModuleHelper (see page 6562) property.
	GetWebSnapObjClass (see page 6553)	Returns the class type for this adapter field.
	Notification (see page 6553)	Responds to notifications that components are about to be inserted or removed.
	Notification (see page 6561)	Responds to notifications that components are about to be inserted or removed.

Protected Properties

	Name	Description
	SiteModuleHelper (see page 6555)	References the helper object that keeps track of WebSnap objects in the module and generates OnActivate (see page 6556) and OnDeactivate (see page 6556) events.
	SitePageModuleHelper (see page 6562)	References the helper object that assists in assembling the module's Web page.

Public Events

	Name	Description
	OnAfterRedirectToPage (see page 6563)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.

	OnBeforeRedirectToPage (see page 6564)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.

Public Methods

	Name	Description
	Create (see page 6551)	Creates and initializes an instance of TCustomWebDataModule (see page 6550).
	Create (see page 6559)	Creates and initializes an instance of TCustomWebPageModule (see page 6556).
	CreateNew (see page 6552)	Creates and initializes an instance of TCustomWebDataModule (see page 6550) without using an associated form file.
	CreateNew (see page 6559)	Creates and initializes an instance of TCustomWebPageModule (see page 6556) without using an associated form file.
	Destroy (see page 6552)	Frees an instance of TCustomWebDataModule (see page 6550).
	Destroy (see page 6560)	Frees an instance of TCustomWebPageModule (see page 6556).
	DispatchPage (see page 1065)	This is DispatchPage, a member of class IPageResult.
	EndIterator (see page 1056)	Signals the end of the iterating operation.
	GetCurrentObject (see page 1052)	This is GetCurrentObject, a member of class IIterateObjectSupport.
	GetDefaultAction (see page 1028)	This is GetDefaultAction, a member of class IGetDefaultAction.
	GetProducerComponent (see page 1035)	This is GetProducerComponent, a member of class IGetProducerComponent.
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IncludePage (see page 1065)	This is IncludePage, a member of class IPageResult.
	NextIteration (see page 1056)	Advances to the objects next item.
	NotifyActivate (see page 6089)	This is NotifyActivate, a member of class INotifyWebActivate.
	NotifyDeactivate (see page 6089)	This is NotifyDeactivate, a member of class INotifyWebActivate.
	RedirectToPage (see page 1066)	This is RedirectToPage, a member of class IPageResult.
	SetDefaultPageFileName (see page 1015)	This is SetDefaultPageFileName, a member of class IDefaultPageFileName.
	SetWebContentOptions (see page 6092)	This is SetWebContentOptions, a member of class ISetWebContentOptions.
	StartIterator (see page 1056)	Positions the object on its first item and starts the iteration process.

Public Properties

	Name	Description
	Request (see page 6554)	Provides access to the current request object.
	Response (see page 6554)	Provides access to the current response object.
	Session (see page 6555)	Provides access to the current Web session.

Published Events

	Name	Description
	OnActivate (see page 6607)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6607)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6608)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6608)	Occurs when the Web module is deactivated.

Published Properties

	Name	Description
	DefaultAction (see page 6606)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6606)	Indicates the interface of the page producer that generates the content of the Web page.

63.1.14.2 TWebPageModule Properties

The properties of the TWebPageModule class are listed here.

Published Properties

	Name	Description
	DefaultAction (see page 6606)	Represents the adapter action associated with the Web page module.
	PageProducer (see page 6606)	Indicates the interface of the page producer that generates the content of the Web page.

63.1.14.2.1 TWebPageModule.DefaultAction Property

Represents the adapter action associated with the Web page module.

Pascal

```
property DefaultAction: TComponent;
```

C++

```
__property TAdapterActionName DefaultAction;__property TComponent * DefaultAction;
```

Description

Use DefaultAction to associate an adapter action with the Web page module. Server-side script can reference this action by calling on the default action of the Web page.

Typically, the default action sets up parameters that appear in HTTP requests for the Web page. By adding a default action, the parameters can then appear in requests for the Web page, which the page producer in the Web page module uses in generating its page. In addition, this action can be called to set up parameters in redirections to this Web page.

See Also

PageProducer (see page 6562)

63.1.14.2.2 TWebPageModule.PageProducer Property

Indicates the interface of the page producer that generates the content of the Web page.

Pascal

```
property PageProducer: IProduceContent;
```

C++

```
__property IProduceContent PageProducer;
```

Description

PageProducer is the interface that supplies the content of the Web page this module creates. It is the interface of a page producer component that is included in the Web page module. If PageProducer is nil (Delphi) or NULL (C++), the Web page module can't produce a Web page.

See Also

TAdapterPageProducer (see page 1136)

TDataSetPageProducer

TXSLPageProducer (see page 7893)

TPageProducer

TDataSetTableProducer

TQueryTableProducer

TInetXPageProducer

63.1.14.3 TWebPageModule Events

The events of the TWebPageModule class are listed here.

Published Events

	Name	Description
	OnActivate (see page 6607)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6607)	Occurs after the page module generates its Web page.
	OnBeforeDispatchPage (see page 6608)	Occurs before the Web module calls on the page producer to generate the content of the Web page.
	OnDeactivate (see page 6608)	Occurs when the Web module is deactivated.

63.1.14.3.1 TWebPageModule.OnActivate Event

Occurs when the Web module is activated.

Pascal

```
property OnActivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnActivate;
```

Description

Write an OnActivate event handler to perform any initializations when the Web module is first activated. The Web application module is the first module activated, and it is activated for every HTTP request. Other Web modules are activated based on the CreateMode of their corresponding factory objects, and whether they are needed to process the current request message.

See Also

OnDeactivate (see page 6556)

CreateMode

63.1.14.3.2 TWebPageModule.OnAfterDispatchPage Event

Occurs after the page module generates its Web page.

Pascal

```
property OnAfterDispatchPage: TDispatchPageEvent;
```

C++

```
__property TDispatchPageEvent OnAfterDispatchPage;
```

Description

Write an OnAfterDispatchPage event handler to take any final action after the page producer in the Web page module has generated its Web page. For example, the OnAfterDispatchPage event handler can make adjustments to the headers of the response message.

When OnAfterDispatchPage occurs, the Web page module has already added the content to the response message, but has not yet sent the response. This event does not occur if the BeforeDispatchPage event handler handles the request, bypassing the

automatic use of the module's page producer.

Note: If the Web page module does not include a page producer, the OnAfterDispatchPage event still occurs, but the response message has no content.

See Also

OnBeforeDispatchPage (see page 6564)

PageProducer (see page 6562)

Response (see page 6554)

63.1.14.3.3 TWebPageModule.OnBeforeDispatchPage Event

Occurs before the Web module calls on the page producer to generate the content of the Web page.

Pascal

```
property OnBeforeDispatchPage: TDispatchPageHandledEvent;
```

C++

```
__property TDispatchPageHandledEvent OnBeforeDispatchPage;
```

Description

Write an OnBeforeDispatchPage event handler to process an HTTP request message before the Web page module performs the default processing.

This event can provide an alternate mechanism for handling request messages, and then set the Handled parameter to true to prevent the Web page module from fetching the template file for the page and calling on the page producer to generate the page content.

If the event handler does not handle the request (for example, if it is merely initializing values that the page producer uses), it can leave the Handled parameter as false and the Web page module will continue with the default processing.

Tip: To access the Web request and response objects from within an OnBeforeDispatchPage event handler, use the Request and Response properties.

See Also

OnAfterDispatchPage (see page 6563)

Request (see page 6554)

Response (see page 6554)

PageProducer (see page 6562)

63.1.14.3.4 TWebPageModule.OnDeactivate Event

Occurs when the Web module is deactivated.

Pascal

```
property OnDeactivate: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnDeactivate;
```

Description

Write an OnDeactivate event handler to perform any final cleanup when the Web module is deactivated. Web modules are

deactivated after the HTTP request has been processed, just before the Web context is freed.

See Also

OnActivate (see page 6556)

63.1.15 TWebPageModuleHelper Class

Class Hierarchy

Pascal

```
TWebPageModuleHelper = class(TSitePageModuleHelper);
```

C++

```
class TWebPageModuleHelper : public TSitePageModuleHelper;
```

File

WebModu

Description

This is class WebModu.TWebPageModuleHelper.

63.1.15.1 TWebPageModuleHelper Members

The following tables list the members exposed by TWebPageModuleHelper.

Protected Methods

	Name	Description
	DispatchPage (see page 6577)	This is DispatchPage, a member of class TSitePageModuleHelper.
	EndIterator (see page 6568)	This is EndIterator, a member of class TSiteModuleHelper.
V	FindNamedVariable (see page 6568)	This is FindNamedVariable, a member of class TSiteModuleHelper.
	FindVariable (see page 6569)	This is FindVariable, a member of class TSiteModuleHelper.
	FindVariableContainer (see page 6569)	This is FindVariableContainer, a member of class TSiteModuleHelper.
	GetCurrentObject (see page 6569)	This is GetCurrentObject, a member of class TSiteModuleHelper.
	GetDefaultAction (see page 6578)	This is GetDefaultAction, a member of class TSitePageModuleHelper.
V	GetPageProducer (see page 6611)	This is GetPageProducer, a member of class TWebPageModuleHelper.
	GetProducerComponent (see page 6578)	This is GetProducerComponent, a member of class TSitePageModuleHelper.
	GetRedirectParams (see page 6578)	This is GetRedirectParams, a member of class TSitePageModuleHelper.
	GetTemplateFileType (see page 6579)	This is GetTemplateFileType, a member of class TSitePageModuleHelper.
	GetTemplateStream (see page 6579)	This is GetTemplateStream, a member of class TSitePageModuleHelper.
	GetVariable (see page 6569)	This is GetVariable, a member of class TSiteModuleHelper.
	GetVariableCount (see page 6569)	This is GetVariableCount, a member of class TSiteModuleHelper.
	GetXMLData (see page 6579)	This is GetXMLData, a member of class TSitePageModuleHelper.
	GetXSLData (see page 6579)	This is GetXSLData, a member of class TSitePageModuleHelper.
	HasHTMLBrowserView (see page 6579)	This is HasHTMLBrowserView, a member of class TSitePageModuleHelper.
	HasScriptView (see page 6580)	This is HasScriptView, a member of class TSitePageModuleHelper.
	HasXMLBrowserView (see page 6580)	This is HasXMLBrowserView, a member of class TSitePageModuleHelper.
	HasXSLBrowserView (see page 6580)	This is HasXSLBrowserView, a member of class TSitePageModuleHelper.
	HTMLFromTemplate (see page 6580)	This is HTMLFromTemplate, a member of class TSitePageModuleHelper.
V	ImplDispatchPage (see page 6581)	This is ImplDispatchPage, a member of class TSitePageModuleHelper.

	ImplIncludePage (see page 6581)	This is ImplIncludePage, a member of class TSitePageModuleHelper.
	ImplRedirectToPage (see page 6581)	This is ImplRedirectToPage, a member of class TSitePageModuleHelper.
	ImplSetWebContentOptions (see page 6581)	This is ImplSetWebContentOptions, a member of class TSitePageModuleHelper.
	IncludePage (see page 6582)	This is IncludePage, a member of class TSitePageModuleHelper.
	NextIteration (see page 6570)	This is NextIteration, a member of class TSiteModuleHelper.
	NotifyActivate (see page 6570)	This is NotifyActivate, a member of class TSiteModuleHelper.
	NotifyDeactivate (see page 6570)	This is NotifyDeactivate, a member of class TSiteModuleHelper.
	RedirectToPage (see page 6582)	This is RedirectToPage, a member of class TSitePageModuleHelper.
	SetDefaultPageFileName (see page 6582)	This is SetDefaultPageFileName, a member of class TSitePageModuleHelper.
	SetWebContentOptions (see page 6582)	This is SetWebContentOptions, a member of class TSitePageModuleHelper.
	StartIterator (see page 6571)	This is StartIterator, a member of class TSiteModuleHelper.

Protected Properties

	Name	Description
	Module (see page 6571)	This is Module, a member of class TSiteModuleHelper.

Public Events

	Name	Description
	OnActivate (see page 6573)	Occurs when the Web module is activated.
	OnAfterDispatchPage (see page 6583)	Occurs after the Web page module generates its Web page.
	OnAfterRedirectToPage (see page 6584)	Occurs after the Web page module assembles a response message that represents a redirection to its Web page.
	OnBeforeDispatchPage (see page 6584)	Occurs before the Web module generates the content of the Web page.
	OnBeforeRedirectToPage (see page 6584)	Occurs before the Web page module assembles a response that redirects the caller to its Web page.
	OnDeactivate (see page 6573)	Occurs when the Web module is deactivated.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 6567)	Creates and initializes an instance of TSiteModuleHelper (see page 6565).
	Create (see page 6576)	Creates and initializes an instance of TSitePageModuleHelper (see page 6574).
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 6567)	Frees an instance of TSiteModuleHelper (see page 6565).
	Destroy (see page 6577)	Frees an instance of TSitePageModuleHelper (see page 6574).
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Notification (see page 6570)	Responds to notifications that components are about to be inserted or removed.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	DefaultAction (see page 6583)	Represents the adapter action associated with the Web page module.
R	Request (see page 6571)	Provides access to the current request object.
R	Response (see page 6572)	Provides access to the current response object.
R	Session (see page 6572)	Provides access to the current Web session.
R	WebModuleContext (see page 6572)	Keeps track of name/value pairs that store information for the Web module.

63.1.15.2 TWebPageModuleHelper Methods

The methods of the TWebPageModuleHelper class are listed here.

Protected Methods

	Name	Description
V	GetPageProducer (see page 6611)	This is GetPageProducer, a member of class TWebPageModuleHelper.

63.1.15.2.1 TWebPageModuleHelper.GetPageProducer Method

Pascal

```
function GetPageProducer: IProduceContent; override;
```

C++

```
virtual __fastcall IProduceContent GetPageProducer();
```

Description

This is GetPageProducer, a member of class TWebPageModuleHelper.

63.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FindNamedModuleVariable (see page 6612)	This is function WebModu.FindNamedModuleVariable.
	GetModuleVariable (see page 6612)	This is function WebModu.GetModuleVariable.
	GetModuleVariableCount (see page 6612)	This is function WebModu.GetModuleVariableCount.

63.2.1 WebModu.FindNamedModuleVariable Function

Pascal

```
function FindNamedModuleVariable(AModule: TComponent; const AName: string): TComponent;
```

C++

```
TComponent * FindNamedModuleVariable(TComponent * AModule, const UnicodeString AName);
```

File

WebModu

Description

This is function WebModu.FindNamedModuleVariable.

63.2.2 WebModu.GetModuleVariable Function

Pascal

```
function GetModuleVariable(AModule: TComponent; Index: Integer): TComponent;
```

C++

```
TComponent * GetModuleVariable(TComponent * AModule, int Index);
```

File

WebModu

Description

This is function WebModu.GetModuleVariable.

63.2.3 WebModu.GetModuleVariableCount Function

Pascal

```
function GetModuleVariableCount(AModule: TComponent): Integer;
```

C++

```
int GetModuleVariableCount(TComponent * AModule);
```

File

WebModu

Description

This is function WebModu.GetModuleVariableCount.

63.3 Types

The following table lists types in this documentation.

Types

Name	Description
TGetParamsEvent (see page 6614)	This is type WebModu.TGetParamsEvent.

63.3.1 WebModu.TGetParamsEvent Type

Pascal

```
TGetParamsEvent = procedure (Sender: TObject; Params: TStrings);
```

C++

```
(Sender: TObject; Params: TStrings) ( TGetParamsEvent)();
```

File

WebModu

Description

This is type WebModu.TGetParamsEvent.

64 WebNode Namespace

This is namespace WebNode.

Topics

Name	Description
Interfaces (see page 6616)	The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWebNode (see page 6616)	IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

64.1 Interfaces

The following table lists interfaces in this documentation.

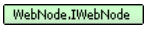
Interfaces

	Name	Description
	IWebNode (see page 6616)	IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

64.1.1 IWebNode Interface

IWebNode is the interface for executing invokable interface calls from the client of a Web Service.

Class Hierarchy



Pascal

```
IWebNode = interface;
```

C++

```
__interface IWebNode;
```

File

WebNode

Description

TRIO objects use IWebNode to execute method calls on an invokable interface. IWebNode takes an encoded method call, executes it, and returns the encoded results. Typically, objects that implement IWebNode use a specific transport protocol to forward the encoded method call to a remote server. For example, the THTTPReqResp class implements this interface for making interface calls to a remote Web Service application using HTTP.

When implementing the IWebNode interface, you can use another IWebNode implementation to delegate part of the method execution. For example, you can write a class that compresses or encrypts the encoded method call, passes it to another IWebNode implementation that expects the compressed or encrypted string, and then decompresses or decrypts the results returned by the other IWebNode implementation.

Note: C++ method declarations that use IWebNode use the _di_IWebNode type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< >  _di_IWebNode;
```

See Also

THTTPReqResp (see page 1276)

TRIO

64.1.1.1 IWebNode Members

The following tables list the members exposed by IWebNode.

Public Methods

	Name	Description
	BeforeExecute (see page 6617)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 6618)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 6619)	Returns the value of the MimeBoundary (see page 6619) property.
	SetMimeBoundary (see page 6619)	Sets the value of the MimeBoundary (see page 6619) property.

Public Properties

	Name	Description
	MimeBoundary (see page 6619)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

64.1.1.2 IWebNode Methods

The methods of the IWebNode class are listed here.

Public Methods

	Name	Description
	BeforeExecute (see page 6617)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 6618)	This is the overview for the Execute method overload.
	GetMimeBoundary (see page 6619)	Returns the value of the MimeBoundary (see page 6619) property.
	SetMimeBoundary (see page 6619)	Sets the value of the MimeBoundary (see page 6619) property.

64.1.1.2.1 IWebNode.BeforeExecute Method

Performs any transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMD: TIntfMetaData; const MethMD: TIntfMethEntry;
MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMD, const TIntfMethEntry MethMD, int
MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute handles transport-specific tasks such as dynamically querying a WSDL document or setting up serialization options before a call is made to the Execute (see page 6618) method. Remote interfaced objects (TRio) call this method after their OnBeforeExecute event, but before delegating the execution of the method to the Execute (see page 6618) method.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method.

See Also

Execute (see page 6618)

OnBeforeExecute

64.1.1.2.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	IWebNode.Execute (String, TStream) (see page 6618)	Executes an encoded method call and returns the encoded results in a stream.
	IWebNode.Execute (TStream) (see page 6618)	Executes an encoded method call and returns the encoded results in a stream.
	IWebNode.Execute (TStream, TStream) (see page 6618)	Executes an encoded method call and returns the encoded results in a stream.

64.1.1.2.2.1 IWebNode.Execute Method (String, TStream)

Executes an encoded method call and returns the encoded results in a stream.

Pascal

```
procedure Execute(const DataMsg: String; Response: TStream); overload; deprecated;
procedure Execute(const Request: TStream; Response: TStream); overload;
function Execute(const Request: TStream): TStream; overload;
```

C++

```
__fastcall Execute(const UnicodeString DataMsg, TStream Response);
__fastcall Execute(const TStream Request, TStream Response);
__fastcall TStream Execute(const TStream Request);
```

Description

Execute interprets an encoded method call, executes it, and returns the encoded results using a stream. Typically, this method works in conjunction with an object that implements the IOPConvert interface, which marshals a method call into the DataMsg or Request string and unmarshals the return value.

DataMsg is an encoded method call. It represents the results of marshaling a method call on an invokable interface into a transportable string. The use of this syntax is not recommended.

Request is a stream from which to read the same string that is the value of the DataMsg parameter. This syntax is preferred because the stream need not store this value in memory (for example, it can be a file stream).

Response is a stream object that receives the encoded results that arise from executing the method call.

When using the third syntax, Execute returns a stream from which to read the encoded results from executing the method call. In this case, the caller is responsible for freeing the stream.

See Also

IOPConvert

BeforeExecute (see page 6617)

64.1.1.2.3 IWebNode.GetMimeBoundary Method

Returns the value of the MimeBoundary (see page 6619) property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary (see page 6619) property.

See Also

MimeBoundary (see page 6619)

64.1.1.2.4 IWebNode.SetMimeBoundary Method

Sets the value of the MimeBoundary (see page 6619) property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary (see page 6619) property.

Value is the new value for MimeBoundary (see page 6619).

See Also

MimeBoundary (see page 6619)

64.1.1.3 IWebNode Properties

The properties of the IWebNode class are listed here.

Public Properties

	Name	Description
	MimeBoundary (see page 6619)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

64.1.1.3.1 IWebNode.MimeBoundary Property

Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Pascal

```
property MimeBoundary: string;
```

C++

```
__property UnicodeString MimeBoundary;
```

Description

When the method call is included in a multipart form, MimeBoundary is the value of the MIME boundary header that uniquely identifies the part of the form that contains the method call. If the method call is not included in a multipart form, MimeBoundary is an empty string.

See Also

Execute (see page 6618)

65 WebReq Namespace

This is namespace WebReq.

Topics

Name	Description
Classes (see page 6622)	The following table lists classes in this documentation.
Functions (see page 6642)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6643)	The following table lists structs, records, enums in this documentation.
Variables (see page 6644)	The following table lists variables in this documentation.

Classes

	Name	Description
	TWebModuleFactoryList (see page 6622)	
	TWebModuleList (see page 6626)	TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TWebRequestHandler (see page 6633)	TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

Functions

	Name	Description
	WebRequestHandler (see page 6642)	Returns a TWebRequestHandler (see page 6633) object.

Structs, Records, Enums

	Name	Description
	TRequestNotification (see page 6643)	This is record WebReq.TRequestNotification.

Variables

Name	Description
WebRequestHandlerProc (see page 6644)	This is variable WebReq.WebRequestHandlerProc.

65.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWebModuleFactoryList (see page 6622)	
	TWebModuleList (see page 6626)	TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.
	TWebRequestHandler (see page 6633)	TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the application receives HTTP request messages.

65.1.1 TWebModuleFactoryList Class

Class Hierarchy

Pascal

```
TWebModuleFactoryList = class(TObject);
```

C++

```
class TWebModuleFactoryList : public TObject;
```

File

WebReq

65.1.1.1 TWebModuleFactoryList Members

The following tables list the members exposed by TWebModuleFactoryList.

Protected Methods

	Name	Description
	GetItem (see page 6625)	This is GetItem, a member of class TWebModuleFactoryList.

Public Methods

	Name	Description
	AddFactory (see page 6624)	Adds a factory to the web module factory list.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.

	Create (see page 6623)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6624)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AppModuleFactory (see page 6625)	
R	ItemCount (see page 6625)	Lists the number of items in the web module factory list.
R	Items (see page 6626)	

65.1.1.2 TWebModuleFactoryList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebModuleFactoryList();
```

Description

Do not create instances of Create. Instead, construct descendants of Create using the new keyword, passing any arguments for the descendant object's constructor.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

new

65.1.1.3 TWebModuleFactoryList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebModuleFactoryList();
```

Description

Do not call ~Destroy directly. Instead, use the delete keyword when destroying VCL objects.

~Destroy deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

Warning: Never destroy a component from one of its own event handlers or from the event handler of a component it owns or contains. For example, don't destroy a button, or the form that owns the button, in the button's OnClick event handler.

To destroy a form, call its Release (see page 3641) method, which destroys the form and releases the memory allocated for it after all its event handlers and those of the components it contains have executed.

See Also

delete

65.1.1.4 TWebModuleFactoryList Methods

The methods of the TWebModuleFactoryList class are listed here.

Protected Methods

	Name	Description
	GetItem (see page 6625)	This is GetItem, a member of class TWebModuleFactoryList.

Public Methods

	Name	Description
	AddFactory (see page 6624)	Adds a factory to the web module factory list.

65.1.1.4.1 TWebModuleFactoryList.AddFactory Method

Adds a factory to the web module factory list.

Pascal

```
procedure AddFactory(AFactory: TAbstractWebModuleFactory);
```

C++

```
__fastcall AddFactory(TAbstractWebModuleFactory AFactory);
```

Description

Use the AddFactory method to add a factory to the web module factory list (more here, obviously).

See Also

AppModuleFactory (see page 6625)

65.1.1.4.2 TWebModuleFactoryList.GetItem Method

Pascal

```
function GetItem(I: Integer): TAbstractWebModuleFactory;
```

C++

```
__fastcall TAbstractWebModuleFactory GetItem(int I);
```

Description

This is GetItem, a member of class TWebModuleFactoryList.

65.1.1.5 TWebModuleFactoryList Properties

The properties of the TWebModuleFactoryList class are listed here.

Public Properties

	Name	Description
	AppModuleFactory (see page 6625)	
	ItemCount (see page 6625)	Lists the number of items in the web module factory list.
	Items (see page 6626)	

65.1.1.5.1 TWebModuleFactoryList.AppModuleFactory Property

Pascal

```
property AppModuleFactory: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory AppModuleFactory;
```

See Also

AddFactory (see page 6624)

65.1.1.5.2 TWebModuleFactoryList.ItemCount Property

Lists the number of items in the web module factory list.

Pascal

```
property ItemCount: Integer;
```

C++

```
__property int ItemCount;
```

Description

Use the ItemCount property to list the number of items in the web module factory list.

See Also

Items (see page 6626)

65.1.1.5.3 TWebModuleFactoryList.Items Property

Pascal

```
property Items [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory Items[int I];
```

See Also

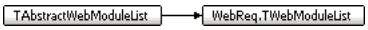
AddFactory (see page 6624)

ItemCount (see page 6625)

65.1.2 TWebModuleList Class

TWebModuleList lists the WebModules in a WebSnap application and provides access to their factory objects.

Class Hierarchy



Pascal

```
TWebModuleList = class(TAbstractWebModuleList);
```

C++

```
class TWebModuleList : public TAbstractWebModuleList;
```

File

WebReq

Description

TWebModuleList contains every Web module that has been instantiated in order to serve the current request. In addition, it lists the factory objects for all Web modules in the application, including any that are not instantiated.

When a WebSnap application handles multiple requests concurrently, it includes multiple Web modules lists, each on a separate thread and each associated with a different request.

The Web context, which can be accessed using the global WebContext (see page 6003) function, provides access to the TWebModuleList object that is associated with the current request. Use the Web context's WebModules property to obtain a reference to the Web module list.

See Also

TWebModuleFactoryList (see page 6622)

TAbstractWebModuleFactory

WebContext (see page 6003)

TAbstractWebContext (see page 5960)

65.1.2.1 TWebModuleList Members

The following tables list the members exposed by TWebModuleList.

Protected Methods

	Name	Description
	GetFactory (see page 6632)	This is GetFactory, a member of class TWebModuleList.
	GetFactoryCount (see page 6632)	
	GetItem (see page 6632)	This is GetItem, a member of class TWebModuleList.
	GetItemCount (see page 6632)	
	GetOnModuleAdded (see page 6633)	
	SetOnModuleAdded (see page 6633)	

Protected Properties

	Name	Description
	Factories (see page 6633)	This is Factories, a member of class TWebModuleList.

Public Methods

	Name	Description
	AddModule (see page 6628)	Adds a module to the Items property array.
	AddModuleClass (see page 6629)	Creates a new Web module, given its class, and adds it to the Items property array.
	AddModuleName (see page 6629)	Creates a new Web module, given its name, and adds it to the Items property array.
	AutoCreateModules (see page 6630)	Instantiates the Web modules that are flagged to handle all incoming requests.
	AutoDestroyModules (see page 6630)	Frees all Web modules that are flagged as inappropriate for caching between requests.
	Create (see page 6627)	Creates and initializes a TWebModuleList (see page 6626) object.
	Destroy (see page 6628)	Frees a TWebModuleList (see page 6626) instance.
	FindModuleClass (see page 6631)	Locates a Web module given its class reference.
	FindModuleName (see page 6631)	Locates a Web module given its name.

65.1.2.2 TWebModuleList.Create Constructor

Creates and initializes a TWebModuleList (see page 6626) object.

Pascal

```
constructor Create(const AFactories: TWebModuleFactoryList);
```

C++

```
__fastcall TWebModuleList(const TWebModuleFactoryList AFactories);
```

Description

There is no need to call the TWebModuleList (see page 6626) constructor. Web module lists are created automatically for each incoming HTTP request.

AFactories is the list of Web module factories in the WebSnap application. TWebModuleList (see page 6626) uses this list to implement the Factory property.

See Also

Factory

65.1.2.3 TWebModuleList.Destroy Destructor

Frees a TWebModuleList (see page 6626) instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebModuleList();
```

Description

Do not free TWebModuleList (see page 6626) objects. Web module lists are freed automatically when the WebSnap application finishes processing the HTTP request.

65.1.2.4 TWebModuleList Methods

The methods of the TWebModuleList class are listed here.

Protected Methods

	Name	Description
	GetFactory (see page 6632)	This is GetFactory, a member of class TWebModuleList.
	GetFactoryCount (see page 6632)	
	GetItem (see page 6632)	This is GetItem, a member of class TWebModuleList.
	GetItemCount (see page 6632)	
	GetOnModuleAdded (see page 6633)	
	SetOnModuleAdded (see page 6633)	

Public Methods

	Name	Description
	AddModule (see page 6628)	Adds a module to the Items property array.
	AddModuleClass (see page 6629)	Creates a new Web module, given its class, and adds it to the Items property array.
	AddModuleName (see page 6629)	Creates a new Web module, given its name, and adds it to the Items property array.
	AutoCreateModules (see page 6630)	Instantiates the Web modules that are flagged to handle all incoming requests.
	AutoDestroyModules (see page 6630)	Frees all Web modules that are flagged as inappropriate for caching between requests.
	FindModuleClass (see page 6631)	Locates a Web module given its class reference.
	FindModuleName (see page 6631)	Locates a Web module given its name.

65.1.2.4.1 TWebModuleList.AddModule Method

Adds a module to the Items property array.

Pascal

```
procedure AddModule(AComponent: TComponent);
```

C++

```
__fastcall AddModule(TComponent * AComponent);
```

Description

Call AddModule to add a newly instantiated Web module to the Items property array.

AComponent is a Web module instance that is not already listed in the Items property array. It must be one of the objects created by a Web module factory from the Factory property array.

See Also

AddModuleName (see page 6629)

AddModuleClass (see page 6629)

Items

Factory

65.1.2.4.2 TWebModuleList.AddModuleClass Method

Creates a new Web module, given its class, and adds it to the Items property array.

Pascal

```
function AddModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * AddModuleClass(TComponentClass AClass);
```

Description

AddModuleClass searches for a Web module factory that can create a Web module of the specified class. If it finds a match, it instantiates the Web module and adds it to the Items property array.

AClass is a class reference to the Web module you want to instantiate. It should not match the class type of any Web module already listed in the Items property array.

AddModuleClass returns the new Web module instance. If the Factory property array does not include a Web module factory that can create a Web module of the specified class, AddModuleClass returns nil (Delphi) or NULL (C++).

See Also

AddModuleName (see page 6629)

AddModule (see page 6628)

Items

Factory

FindModuleClass (see page 6631)

65.1.2.4.3 TWebModuleList.AddModuleName Method

Creates a new Web module, given its name, and adds it to the Items property array.

Pascal

```
function AddModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * AddModuleName(const UnicodeString AName);
```

Description

AddModuleName searches for a Web module factory that can create a Web module of the specified name. If it finds a match, it instantiates the Web module and adds it to the Items property array.

AName is the name of the Web module you want to instantiate. It should not match the name of any Web module already listed

in the Items property array.

AddModuleClass (see page 6629) returns the new Web module instance. If the Factory property array does not include a Web module factory that can create a Web module of the specified name, AddModuleClass (see page 6629) returns nil (Delphi) or NULL (C++).

See Also

AddModuleClass (see page 6629)

AddModule (see page 6628)

Items

Factory

FindModuleName (see page 6631)

65.1.2.4.4 TWebModuleList.AutoCreateModules Method

Instantiates the Web modules that are flagged to handle all incoming requests.

Pascal

```
procedure AutoCreateModules;
```

C++

```
__fastcall AutoCreateModules();
```

Description

TWebRequestHandler (see page 6633) calls AutoCreateModules to instantiate all of the Web modules that should be used to handle every incoming request object.

AutoCreateModules searches the module factory list for factory objects that have a CreateMode property with the crAlways flag. For each such factory object, AutoCreateModules creates the corresponding Web module if it does not already exist and adds it to the Items property array.

See Also

CreateMode

AutoDestroyModules (see page 6630)

Items

Factory

TWebRequestHandler (see page 6633)

65.1.2.4.5 TWebModuleList.AutoDestroyModules Method

Frees all Web modules that are flagged as inappropriate for caching between requests.

Pascal

```
procedure AutoDestroyModules;
```

C++

```
__fastcall AutoDestroyModules();
```

Description

TWebRequestHandler (see page 6633) calls AutoDestroyModules when it finishes processing an HTTP request message.

This allows the Web module list to free those Web modules that should not be cached for reuse with another request message.

AutoDestroyModules searches the module factory list for factory objects that have a CacheMode property with the caDestroy flag. For each such factory object, AutoDestroyModules frees the Web module if it exists and removes it from Items property array.

See Also

CacheMode

AutoCreateModules (see page 6630)

Items

TWebRequestHandler (see page 6633)

Factory

65.1.2.4.6 TWebModuleList.FindModuleClass Method

Locates a Web module given its class reference.

Pascal

```
function FindModuleClass(AClass: TComponentClass): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleClass(TComponentClass AClass);
```

Description

Call FindModuleClass to locate the Web module instance of a given class type. FindModuleClass searches the Items property array for a Web module instance of the specified class.

AClass is a class reference to the class type of the desired Web module.

FindModuleClass returns the requested Web module instance if it exists. If there is no Web module of the specified class, FindModuleClass returns nil (Delphi) or NULL (C++).

See Also

FindModuleName (see page 6631)

Items

65.1.2.4.7 TWebModuleList.FindModuleName Method

Locates a Web module given its name.

Pascal

```
function FindModuleName(const AName: string): TComponent; override;
```

C++

```
virtual __fastcall TComponent * FindModuleName(const UnicodeString AName);
```

Description

Call FindModuleName to locate the Web module instance of a given name. FindModuleClass (see page 6631) searches the Items property array for a Web module instance of the specified name.

AName is the name of the desired Web module.

FindModuleName returns the requested Web module instance if it exists. If there is no Web module with the specified name,

FindModuleClass (see page 6631) returns nil (Delphi) or NULL (C++).

See Also

FindModuleClass (see page 6631)

Items

65.1.2.4.8 TWebModuleList.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractWebModuleFactory; override;
```

C++

```
virtual __fastcall TAbstractWebModuleFactory GetFactory(int I);
```

Description

This is GetFactory, a member of class TWebModuleList.

65.1.2.4.9 TWebModuleList.GetFactoryCount Method

Pascal

```
function GetFactoryCount: Integer; override;
```

C++

```
virtual __fastcall int GetFactoryCount();
```

See Also

GetFactory

FactoryCount

65.1.2.4.10 TWebModuleList.GetItem Method

Pascal

```
function GetItem(I: Integer): TComponent; override;
```

C++

```
virtual __fastcall TComponent * GetItem(int I);
```

Description

This is GetItem, a member of class TWebModuleList.

65.1.2.4.11 TWebModuleList.GetItemCount Method

Pascal

```
function GetItemCount: Integer; override;
```

C++

```
virtual __fastcall int GetItemCount();
```

See Also

GetItem

ItemCount

65.1.2.4.12 TWebModuleList.GetOnModuleAdded Method

Pascal

```
function GetOnModuleAdded: TModuleAddedProc; override;
```

C++

```
virtual __fastcall TModuleAddedProc GetOnModuleAdded();
```

See Also

OnModuleAdded

SetOnModuleAdded

65.1.2.4.13 TWebModuleList.SetOnModuleAdded Method

Pascal

```
procedure SetOnModuleAdded(AProc: TModuleAddedProc); override;
```

C++

```
virtual __fastcall SetOnModuleAdded(TModuleAddedProc AProc);
```

See Also

GetOnModuleAdded

OnModuleAdded

65.1.2.5 TWebModuleList Properties

The properties of the TWebModuleList class are listed here.

Protected Properties

	Name	Description
	Factories (see page 6633)	This is Factories, a member of class TWebModuleList.

65.1.2.5.1 TWebModuleList.Factories Property

Pascal

```
property Factories: TWebModuleFactoryList;
```

C++

```
__property TWebModuleFactoryList Factories;
```

Description

This is Factories, a member of class TWebModuleList.

65.1.3 TWebRequestHandler Class

TWebRequestHandler maintains the Web modules in an application and creates request and response objects when the

application receives HTTP request messages.

Class Hierarchy

Pascal

```
TWebRequestHandler = class(TComponent);
```

C++

```
class TWebRequestHandler : public TComponent;
```

File

WebReq

Description

TWebRequestHandler keeps a pool of active Web modules. In response to a request from the application, TWebRequestHandler creates a request object and assigns it to one of the active Web modules.

Do not create instances of TWebRequestHandler. Use TWebRequestHandler as a base class when declaring nonvisual components that can appear on the component palette and be used in the form designer. Properties and methods of TWebRequestHandler provide basic behavior that descendant classes inherit as well as behavior that components can override to customize their behavior.

See Also

TWebApplication (see page 5936)

TWebModule

65.1.3.1 TWebRequestHandler Members

The following tables list the members exposed by TWebRequestHandler.

Protected Methods

	Name	Description
	ActivateWebModules (see page 6636)	Activates a Web module for the application and returns it.
	DeactivateWebModules (see page 6636)	Deactivates the Web module of the application.
	HandleRequest (see page 6637)	Responds to an HTTP request message received by the server.

Public Methods

	Name	Description
	AddWebModuleFactory (see page 6636)	Adds a Web module factory to the request handler.
V	Create (see page 6635)	Creates an instance of TWebRequestHandler (see page 6633).
V	Destroy (see page 6635)	Destroys the TWebRequestHandler (see page 6633) instance and frees its memory.
	HandleException (see page 6637)	Responds to exceptions related to TWebRequestHandler (see page 6633) objects.

Public Properties

	Name	Description
R	ActiveCount (see page 6638)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 6638)	Determines whether the Web modules used by the application are cached for re-use.
R	InactiveCount (see page 6639)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 6640)	Sets or returns the MaxConnections used by the socket.

	WebModuleClass (see page 6640)	Contains a reference to a Web module.
R	WebModuleFactory (see page 6640)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 6641)	The number of Web module factories in use by the request handler.

65.1.3.2 TWebRequestHandler.Create Constructor

Creates an instance of TWebRequestHandler (see page 6633).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWebRequestHandler(TComponent * AOwner);
```

Description

Call Create to instantiate a TWebRequestHandler (see page 6633) object.

See Also

Destroy (see page 6635)

65.1.3.3 TWebRequestHandler.Destroy Destructor

Destroys the TWebRequestHandler (see page 6633) instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebRequestHandler();
```

Description

Destroy cleans up references in the global fix-up list for form linking. It then calls the inherited destructor.

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

System::TObject::Destroy (see page 3356)

65.1.3.4 TWebRequestHandler Methods

The methods of the TWebRequestHandler class are listed here.

Protected Methods

	Name	Description
	ActivateWebModules (see page 6636)	Activates a Web module for the application and returns it.
	DeactivateWebModules (see page 6636)	Deactivates the Web module of the application.
	HandleRequest (see page 6637)	Responds to an HTTP request message received by the server.

Public Methods

	Name	Description
	AddWebModuleFactory (see page 6636)	Adds a Web module factory to the request handler.
	HandleException (see page 6637)	Responds to exceptions related to TWebRequestHandler (see page 6633) objects.

65.1.3.4.1 TWebRequestHandler.ActivateWebModules Method

Activates a Web module for the application and returns it.

Pascal

```
function ActivateWebModules: TWebModuleList;
```

C++

```
__fastcall TWebModuleList ActivateWebModules();
```

Description

ActivateWebModules is called internally when the application receives an HTTP request message. If the InactiveCount (see page 6639) property is greater than zero, a cached Web module is reactivated. Otherwise, a new instance of the Web module is created.

Before ActivateWebModules is called, the CreateForm method must be called to register the data module.

Note: In cross-platform applications, a Web application object contains only a single Web module. Usually, that Web module is a TWebModule object, but it may also be an ordinary data module that contains a TWebDispatcher object. If no Web module is associated with the Web application, ActivateWebModule raises an exception. If the ActiveCount (see page 6638) property is already at MaxConnections (see page 6640) when ActivateWebModules is called, ActivateWebModules raises an exception.

See Also

CreateForm (see page 5938)

ActiveCount (see page 6638)

InactiveCount (see page 6639)

MaxConnections (see page 6640)

65.1.3.4.2 TWebRequestHandler.AddWebModuleFactory Method

Adds a Web module factory to the request handler.

Pascal

```
procedure AddWebModuleFactory(AFactory: TAbstractWebModuleFactory);
```

C++

```
__fastcall AddWebModuleFactory(TAbstractWebModuleFactory AFactory);
```

Description

AddWebModuleFactory is used to add a Web module factory to the Web request handler. Web module factories are for internal use, and should not require any attention from developers.

65.1.3.4.3 TWebRequestHandler.DeactivateWebModules Method

Deactivates the Web module of the application.

Pascal

```
procedure DeactivateWebModules(WebModules: TWebModuleList);
```

C++

```
__fastcall DeactivateWebModules(TWebModuleList WebModules);
```

Description

Do not call DeactivateWebModules directly. The Web application calls DeactivateWebModules internally after handling an HTTP request message. When CacheConnections (see page 6638) is true, the Web module remains inactive until the next request message is received. When CacheConnections (see page 6638) is false, the runtime instance of the Web module is destroyed.

See Also

ActivateWebModules (see page 6636)

CacheConnections (see page 6638)

HandleRequest (see page 6637)

InactiveCount (see page 6639)

65.1.3.4.4 TWebRequestHandler.HandleException Method

Responds to exceptions related to TWebRequestHandler (see page 6633) objects.

Pascal

```
procedure HandleException(Sender: TObject);
```

C++

```
__fastcall HandleException(TObject * Sender);
```

Description

When exceptions occur in TWebRequestHandler (see page 6633) and descendant objects, HandleException will attempt to deal with it. If it cannot do so, it displays an error message to the user.

65.1.3.4.5 TWebRequestHandler.HandleRequest Method

Responds to an HTTP request message received by the server.

Pascal

```
function HandleRequest(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean HandleRequest(TWebRequest Request, TWebResponse Response);
```

Description

When a descendant of TWebApplication receives an HTTP request message, it creates a TWebRequest object to represent the message, and a TWebResponse message to match the request. It then calls HandleRequest, passing those objects as the Request and Response parameters.

HandleRequest activates the Web module associated with the Web application. It then passes the request and response objects to the dispatcher, so that it can dispatch the request to the appropriate action items. When the dispatcher returns, HandleRequest checks whether the response was handled but not sent, and if not, sends it. Finally, HandleRequest deactivates the data module.

HandleRequest returns a Boolean value indicating whether the request message was handled and sent.

See Also

ActivateWebModules (see page 6636)

DeactivateWebModules (see page 6636)

65.1.3.5 TWebRequestHandler Properties

The properties of the TWebRequestHandler class are listed here.

Public Properties

	Name	Description
R	ActiveCount (see page 6638)	Indicates the number of active connections currently being serviced by the application.
	CacheConnections (see page 6638)	Determines whether the Web modules used by the application are cached for re-use.
R	InactiveCount (see page 6639)	Indicates the number of Web modules cached for later reuse.
	MaxConnections (see page 6640)	Sets or returns the MaxConnections used by the socket.
	WebModuleClass (see page 6640)	Contains a reference to a Web module.
R	WebModuleFactory (see page 6640)	A list of Web module factory objects.
R	WebModuleFactoryCount (see page 6641)	The number of Web module factories in use by the request handler.

65.1.3.5.1 TWebRequestHandler.ActiveCount Property

Indicates the number of active connections currently being serviced by the application.

Pascal

```
property ActiveCount: Integer;
```

C++

```
__property int ActiveCount;
```

Description

Use ActiveCount to monitor the number of HTTP request messages the application is servicing at any one time. Over time, this value provides an indication of the optimal value for the MaxConnections (see page 6640) property.

Each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it generates a new instance of the Web module. After the response is returned to the Web client, the Web module is either destroyed or deactivated and cached for later reuse. Each active Web module resides in a separate execution thread.

ActiveCount is the number of Web modules currently in use. InactiveCount (see page 6639) is the number of deactivated Web modules cached for later reuse. MaxConnections (see page 6640) is the maximum number of Web modules that can be active at any time. ActiveCount can never exceed MaxConnections (see page 6640).

Note: ActiveCount is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message.

See Also

InactiveCount (see page 6639)

65.1.3.5.2 TWebRequestHandler.CacheConnections Property

Determines whether the Web modules used by the application are cached for re-use.

Pascal

```
property CacheConnections: Boolean;
```

C++

```
__property Boolean CacheConnections;
```

Description

Each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it spawns a thread to handle the request and creates a new Web module that contains instances of all the objects used in the Web application. To improve performance, these Web modules can be saved in a cache when the thread associated with the HTTP request message finishes executing.

Set CacheConnections to true to have the Web application save inactive Web modules in a cache. Caching the Web modules speeds performance as new instances of the Web module and its contents do not need to be created every time a request message is received. Set CacheConnections to false to have the Web application create a new Web module for every request thread. Recreating the Web module ensures that changes made to the module or its contents when servicing the last request do not cause unexpected behavior the next time the module is activated.

CacheConnections is only relevant to Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) Web applications.

See Also

MaxConnections (see page 6640)

65.1.3.5.3 TWebRequestHandler.InactiveCount Property

Indicates the number of Web modules cached for later reuse.

Pascal

```
property InactiveCount: Integer;
```

C++

```
__property int InactiveCount;
```

Description

Use InactiveCount to monitor the number of unused Web modules. Over time, this value provides an indication of the optimal value for the MaxConnections (see page 6640) property.

When CacheConnections (see page 6638) is true, each time an Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only) application receives an HTTP request message, it reactivates one of the inactive Web modules. The application generates a new instance of the Web module if InactiveCount is 0. After the response is returned to the Web client, the Web module is deactivated again, and counted by the InactiveCount property.

When CacheConnections (see page 6638) is false, a new Web module is created for each HTTP request message and destroyed when the response is returned. InactiveCount is always 0 when CacheConnections (see page 6638) is false.

Note: InactiveCount is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message, so there is no need to cache unused data modules.

See Also

ActiveCount (see page 6638)

65.1.3.5.4 TWebRequestHandler.MaxConnections Property

Sets or returns the MaxConnections used by the socket.

Pascal

```
property MaxConnections: Integer;
```

C++

```
__property int MaxConnections;
```

Description

Set MaxConnections to adjust the number of HTTP request messages the application can handle simultaneously. Use the ActiveCount (see page 6638) and InactiveCount (see page 6639) properties to get a sense of the frequency and duration of request messages. Then set MaxConnections so that it is high enough to avoid exceptions when the application runs out of Web modules, but low enough not to degrade performance too severely by running too many simultaneous threads.

If a request message is received when ActiveCount (see page 6638) is already at MaxConnections, the application raises an exception.

Note: InactiveCount (see page 6639) is only meaningful for Apache (cross-platform), ISAPI (Windows only), or NSAPI (Windows only)

applications. Each CGI application handles only a single request message, so there is no need to cache unused data modules.

See Also

CacheConnections (see page 6638)

65.1.3.5.5 TWebRequestHandler.WebModuleClass Property

Contains a reference to a Web module.

Pascal

```
property WebModuleClass: TComponentClass;
```

C++

```
__property TComponentClass WebModuleClass;
```

Description

WebModuleClass is a reference to a Web module, and is for internal use only. When ActivateWebModules (see page 6636) is called, it is used in that method to help generate infrastructure for a Web module.

See Also

ActivateWebModules (see page 6636)

65.1.3.5.6 TWebRequestHandler.WebModuleFactory Property

A list of Web module factory objects.

Pascal

```
property WebModuleFactory [I: Integer]: TAbstractWebModuleFactory;
```

C++

```
__property TAbstractWebModuleFactory WebModuleFactory[int I];
```

Description

This is a list of Web module factory objects, which are intended for internal use.

See Also

WebModuleFactoryCount (see page 6641)

65.1.3.5.7 TWebRequestHandler.WebModuleFactoryCount Property

The number of Web module factories in use by the request handler.

Pascal

```
property WebModuleFactoryCount: Integer;
```

C++

```
__property int WebModuleFactoryCount;
```

Description

WebModuleFactoryCount is the number of Web module factories in use by the request handler. This property is intended for internal use.

See Also

WebModuleFactory (see page 6640)

65.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WebRequestHandler (see page 6642)	Returns a TWebRequestHandler (see page 6633) object.

65.2.1 WebReq.WebRequestHandler Function

Returns a TWebRequestHandler (see page 6633) object.

Pascal

```
function WebRequestHandler: TWebRequestHandler;
```

C++

```
TWebRequestHandler WebRequestHandler();
```

File

WebReq

Description

Use the WebRequestHandler function to obtain a reference to the global TWebRequestHandler (see page 6633) object. This object manages all the Web modules in the application, and creates Web request and response objects when the application receives HTTP request messages.

See Also

TWebApplication (see page 5936)

TWebModule

65.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TRequestNotification (see page 6643)	This is record WebReq.TRequestNotification.

65.3.1 WebReq.TRequestNotification Enumeration

Pascal

```
TRequestNotification = (
  rnActivateModule,
  rnDeactivateModule,
  rnCreateModule,
  rnFreeModule,
  rnStartRequest,
  rnFinishRequest
);
```

C++

```
enum TRequestNotification {
  rnActivateModule,
  rnDeactivateModule,
  rnCreateModule,
  rnFreeModule,
  rnStartRequest,
  rnFinishRequest
};
```

File

WebReq

Description

This is record WebReq.TRequestNotification.

65.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
WebRequestHandlerProc (see page 6644)	This is variable WebReq.WebRequestHandlerProc.

65.4.1 WebReq.WebRequestHandlerProc Variable

Pascal

```
WebRequestHandlerProc: function: TWebRequestHandler = nil;
```

C++

```
function: TWebRequestHandler WebRequestHandlerProc = nil;
```

File

WebReq

Description

This is variable WebReq.WebRequestHandlerProc.

66 WebScript Namespace

This is namespace WebScript.

Topics

Name	Description
Classes (see page 6647)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6720)	The following table lists structs, records, enums in this documentation.
Types (see page 6722)	The following table lists types in this documentation.
Constants (see page 6723)	The following table lists constants in this documentation.

Classes

	Name	Description
	ERecursiveIncludeFile (see page 6647)	This is class WebScript.ERecursiveIncludeFile.
	EScriptEvalError (see page 6648)	This is class WebScript.EScriptEvalError.
	TAbstractScriptEngine (see page 6648)	This is class WebScript.TAbstractScriptEngine.
	TAbstractScriptObjectFactories (see page 6652)	This is class WebScript.TAbstractScriptObjectFactories.
	TAbstractScriptObjectFactory (see page 6655)	This is class WebScript.TAbstractScriptObjectFactory.
	TCurrentWebPageInfo (see page 6657)	This is class WebScript.TCurrentWebPageInfo.
	THTMLItem (see page 6662)	This is class WebScript.THTMLItem.
	THTMLItems (see page 6665)	This is class WebScript.THTMLItems.
	TLineNumberMap (see page 6668)	This is class WebScript.TLineNumberMap.
	TScriptCodeItem (see page 6671)	This is class WebScript.TScriptCodeItem.
	TScriptEnginesList (see page 6673)	This is class WebScript.TScriptEnginesList.
	TScriptError (see page 6677)	This is class WebScript.TScriptError.
	TScriptErrors (see page 6681)	This is class WebScript.TScriptErrors.
	TScriptFile (see page 6683)	This is class WebScript.TScriptFile.
	TScriptGlobalObjects (see page 6687)	This is class WebScript.TScriptGlobalObjects.
	TScriptIncludeFile (see page 6692)	This is class WebScript.TScriptIncludeFile.
	TScriptIncludeFiles (see page 6693)	This is class WebScript.TScriptIncludeFiles.
	TScriptIncludeItem (see page 6696)	This is class WebScript.TScriptIncludeItem.
	TScriptItem (see page 6698)	This is class WebScript.TScriptItem.
	TScriptItems (see page 6700)	This is class WebScript.TScriptItems.
	TScriptObjectFactories (see page 6702)	This is class WebScript.TScriptObjectFactories.
	TScriptProducer (see page 6707)	This is class WebScript.TScriptProducer.
	TVariableLookup (see page 6716)	This is class WebScript.TVariableLookup.

Constants

Name	Description
GlobalScriptVariableNames (see page 6723)	This is constant WebScript.GlobalScriptVariableNames.

Structs, Records, Enums

	Name	Description
	TGlobalScriptVariables (see page 6720)	This is record WebScript.TGlobalScriptVariables.
	TIncludeType (see page 6721)	This is record WebScript.TIncludeType.
	TLineNumbers (see page 6721)	This is record WebScript.TLineNumbers.

Types

Name	Description
CreateComponentStubFactory (see page 6722)	This is type WebScript.CreateComponentStubFactory.
PLineNumbers (see page 6722)	This is type WebScript.PLineNumbers.

66.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	ERecursiveIncludeFile (see page 6647)	This is class WebScript.ERecursiveIncludeFile.
	EScriptEvalError (see page 6648)	This is class WebScript.EScriptEvalError.
	TAbstractScriptEngine (see page 6648)	This is class WebScript.TAbstractScriptEngine.
	TAbstractScriptObjectFactories (see page 6652)	This is class WebScript.TAbstractScriptObjectFactories.
	TAbstractScriptObjectFactory (see page 6655)	This is class WebScript.TAbstractScriptObjectFactory.
	TCurrentWebPageInfo (see page 6657)	This is class WebScript.TCurrentWebPageInfo.
	THTMLItem (see page 6662)	This is class WebScript.THTMLItem.
	THTMLItems (see page 6665)	This is class WebScript.THTMLItems.
	TLineNumberMap (see page 6668)	This is class WebScript.TLineNumberMap.
	TScriptCodeItem (see page 6671)	This is class WebScript.TScriptCodeItem.
	TScriptEnginesList (see page 6673)	This is class WebScript.TScriptEnginesList.
	TScriptError (see page 6677)	This is class WebScript.TScriptError.
	TScriptErrors (see page 6681)	This is class WebScript.TScriptErrors.
	TScriptFile (see page 6683)	This is class WebScript.TScriptFile.
	TScriptGlobalObjects (see page 6687)	This is class WebScript.TScriptGlobalObjects.
	TScriptIncludeFile (see page 6692)	This is class WebScript.TScriptIncludeFile.
	TScriptIncludeFiles (see page 6693)	This is class WebScript.TScriptIncludeFiles.
	TScriptIncludeItem (see page 6696)	This is class WebScript.TScriptIncludeItem.
	TScriptItem (see page 6698)	This is class WebScript.TScriptItem.
	TScriptItems (see page 6700)	This is class WebScript.TScriptItems.
	TScriptObjectFactories (see page 6702)	This is class WebScript.TScriptObjectFactories.
	TScriptProducer (see page 6707)	This is class WebScript.TScriptProducer.
	TVariableLookup (see page 6716)	This is class WebScript.TVariableLookup.

66.1.1 ERecursiveIncludeFile Class

Class Hierarchy

Pascal

```
ERecursiveIncludeFile = class(EScriptError);
```

C++

```
class ERecursiveIncludeFile : public EScriptError;
```

File

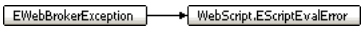
WebScript

Description

This is class WebScript.ERecursiveIncludeFile.

66.1.2 EScriptEvalError Class

Class Hierarchy



Pascal

```
EScriptEvalError = class(EWebBrokerException);
```

C++

```
class EScriptEvalError : public EWebBrokerException;
```

File

WebScript

Description

This is class WebScript.EScriptEvalError.

66.1.3 TAbstractScriptEngine Class

Class Hierarchy

Pascal

```
TAbstractScriptEngine = class(TObject);
```

C++

```
class TAbstractScriptEngine : public TObject;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptEngine.

66.1.3.1 TAbstractScriptEngine Members

The following tables list the members exposed by TAbstractScriptEngine.

Public Methods

	Name	Description
	AddObjectFactory (see page 6650)	This is AddObjectFactory, a member of class TAbstractScriptEngine.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.

	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6649)	This is Create, a member of class TAbstractScriptEngine.
	CreateComponentStub (see page 6651)	This is CreateComponentStub, a member of class TAbstractScriptEngine.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6649)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
A	ListLanguageNames (see page 6651)	This is ListLanguageNames, a member of class TAbstractScriptEngine.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
A	SupportsLanguage (see page 6651)	This is SupportsLanguage, a member of class TAbstractScriptEngine.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Name (see page 6651)	This is Name, a member of class TAbstractScriptEngine.
R	ScriptObjectFactories (see page 6652)	This is ScriptObjectFactories, a member of class TAbstractScriptEngine.
	ScriptProducerClass (see page 6652)	This is ScriptProducerClass, a member of class TAbstractScriptEngine.

66.1.3.2 TAbstractScriptEngine.Create Constructor

Pascal

```
constructor Create(const AName: string);
```

C++

```
__fastcall TAbstractScriptEngine(const UnicodeString AName);
```

Description

This is Create, a member of class TAbstractScriptEngine.

66.1.3.3 TAbstractScriptEngine.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TAbstractScriptEngine();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

66.1.3.4 TAbstractScriptEngine Methods

The methods of the TAbstractScriptEngine class are listed here.

Public Methods

	Name	Description
	AddObjectFactory (see page 6650)	This is AddObjectFactory, a member of class TAbstractScriptEngine.
	CreateComponentStub (see page 6651)	This is CreateComponentStub, a member of class TAbstractScriptEngine.
	ListLanguageNames (see page 6651)	This is ListLanguageNames, a member of class TAbstractScriptEngine.
	SupportsLanguage (see page 6651)	This is SupportsLanguage, a member of class TAbstractScriptEngine.

66.1.3.4.1 TAbstractScriptEngine.AddObjectFactory Method

Pascal

```
procedure AddObjectFactory(AFactory: TAbstractScriptObjectFactory);
```

C++

```
__fastcall AddObjectFactory(TAbstractScriptObjectFactory AFactory);
```

Description

This is AddObjectFactory, a member of class TAbstractScriptEngine.

66.1.3.4.2 TAbstractScriptEngine.CreateComponentStub Method

Pascal

```
function CreateComponentStub(AObject: TObject): IInterface;
```

C++

```
__fastcall IInterface CreateComponentStub(TObject * AObject);
```

Description

This is CreateComponentStub, a member of class TAbstractScriptEngine.

66.1.3.4.3 TAbstractScriptEngine.ListLanguageNames Method

Pascal

```
procedure ListLanguageNames(AList: TStrings); virtual; abstract;
```

C++

```
virtual __fastcall ListLanguageNames(TStrings AList) = 0;
```

Description

This is ListLanguageNames, a member of class TAbstractScriptEngine.

66.1.3.4.4 TAbstractScriptEngine.SupportsLanguage Method

Pascal

```
function SupportsLanguage(const AName: string): Boolean; virtual; abstract;
```

C++

```
virtual __fastcall Boolean SupportsLanguage(const UnicodeString AName) = 0;
```

Description

This is SupportsLanguage, a member of class TAbstractScriptEngine.

66.1.3.5 TAbstractScriptEngine Properties

The properties of the TAbstractScriptEngine class are listed here.

Public Properties

	Name	Description
	Name (see page 6651)	This is Name, a member of class TAbstractScriptEngine.
	ScriptObjectFactories (see page 6652)	This is ScriptObjectFactories, a member of class TAbstractScriptEngine.
	ScriptProducerClass (see page 6652)	This is ScriptProducerClass, a member of class TAbstractScriptEngine.

66.1.3.5.1 TAbstractScriptEngine.Name Property

Pascal

```
property Name: string;
```

C++

```
__property UnicodeString Name;
```

Description

This is Name, a member of class TAbstractScriptEngine.

66.1.3.5.2 TAbstractScriptEngine.ScriptObjectFactories Property

Pascal

```
property ScriptObjectFactories: TAbstractScriptObjectFactories;
```

C++

```
__property TAbstractScriptObjectFactories ScriptObjectFactories;
```

Description

This is ScriptObjectFactories, a member of class TAbstractScriptEngine.

66.1.3.5.3 TAbstractScriptEngine.ScriptProducerClass Property

Pascal

```
property ScriptProducerClass: TScriptProducerClass;
```

C++

```
__property TScriptProducerClass ScriptProducerClass;
```

Description

This is ScriptProducerClass, a member of class TAbstractScriptEngine.

66.1.4 TAbstractScriptObjectFactories Class

Class Hierarchy

Pascal

```
TAbstractScriptObjectFactories = class(TObject);
```

C++

```
class TAbstractScriptObjectFactories : public TObject;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptObjectFactories.

66.1.4.1 TAbstractScriptObjectFactories Members

The following tables list the members exposed by TAbstractScriptObjectFactories.

Protected Methods

	Name	Description
	GetCount (see page 6654)	This is GetCount, a member of class TAbstractScriptObjectFactories.
	GetFactory (see page 6654)	This is GetFactory, a member of class TAbstractScriptObjectFactories.
	GetObjectList (see page 6654)	This is GetObjectList, a member of class TAbstractScriptObjectFactories.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Count (see page 6655)	This is Count, a member of class TAbstractScriptObjectFactories.
	Factories (see page 6655)	This is Factories, a member of class TAbstractScriptObjectFactories.
	ObjectList (see page 6655)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

66.1.4.2 TAbstractScriptObjectFactories Methods

The methods of the TAbstractScriptObjectFactories class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 6654)	This is GetCount, a member of class TAbstractScriptObjectFactories.
	GetFactory (see page 6654)	This is GetFactory, a member of class TAbstractScriptObjectFactories.
	GetObjectList (see page 6654)	This is GetObjectList, a member of class TAbstractScriptObjectFactories.

66.1.4.2.1 TAbstractScriptObjectFactories.GetCount Method

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

This is GetCount, a member of class TAbstractScriptObjectFactories.

66.1.4.2.2 TAbstractScriptObjectFactories.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractScriptObjectFactory; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptObjectFactory GetFactory(int I) = 0;
```

Description

This is GetFactory, a member of class TAbstractScriptObjectFactories.

66.1.4.2.3 TAbstractScriptObjectFactories.GetObjectList Method

Pascal

```
function GetObjectList: TObjectList; virtual; abstract;
```

C++

```
virtual __fastcall TObjectList GetObjectList() = 0;
```

Description

This is GetObjectList, a member of class TAbstractScriptObjectFactories.

66.1.4.3 TAbstractScriptObjectFactories Properties

The properties of the TAbstractScriptObjectFactories class are listed here.

Public Properties

	Name	Description
	Count (see page 6655)	This is Count, a member of class TAbstractScriptObjectFactories.
	Factories (see page 6655)	This is Factories, a member of class TAbstractScriptObjectFactories.
	ObjectList (see page 6655)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

66.1.4.3.1 TAbstractScriptObjectFactories.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TAbstractScriptObjectFactories.

66.1.4.3.2 TAbstractScriptObjectFactories.Factories Property

Pascal

```
property Factories [I: Integer]: TAbstractScriptObjectFactory;
```

C++

```
__property TAbstractScriptObjectFactory Factories[int I];
```

Description

This is Factories, a member of class TAbstractScriptObjectFactories.

66.1.4.3.3 TAbstractScriptObjectFactories.ObjectList Property

Pascal

```
property ObjectList: TObjectList;
```

C++

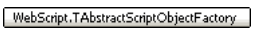
```
__property TObjectList ObjectList;
```

Description

This is ObjectList, a member of class TAbstractScriptObjectFactories.

66.1.5 TAbstractScriptObjectFactory Class

Class Hierarchy



Pascal

```
TAbstractScriptObjectFactory = class;
```

C++

```
class TAbstractScriptObjectFactory;
```

File

WebScript

Description

This is class WebScript.TAbstractScriptObjectFactory.

66.1.5.1 TAbstractScriptObjectFactory Members

The following tables list the members exposed by TAbstractScriptObjectFactory.

Protected Methods

	Name	Description
	AddGlobalObjects (see page 6656)	This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.
	CreateComponentObject (see page 6656)	This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.
	CreateProducerObject (see page 6657)	This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.
	GetScriptEngine (see page 6657)	This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

Public Methods

	Name	Description
	CreateGlobalObject (see page 6657)	This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

66.1.5.2 TAbstractScriptObjectFactory Methods

The methods of the TAbstractScriptObjectFactory class are listed here.

Protected Methods

	Name	Description
	AddGlobalObjects (see page 6656)	This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.
	CreateComponentObject (see page 6656)	This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.
	CreateProducerObject (see page 6657)	This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.
	GetScriptEngine (see page 6657)	This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

Public Methods

	Name	Description
	CreateGlobalObject (see page 6657)	This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

66.1.5.2.1 TAbstractScriptObjectFactory.AddGlobalObjects Method

Pascal

```
procedure AddGlobalObjects(AScriptGlobalObjects: TScriptGlobalObjects); virtual;
```

C++

```
virtual __fastcall AddGlobalObjects(TScriptGlobalObjects AScriptGlobalObjects);
```

Description

This is AddGlobalObjects, a member of class TAbstractScriptObjectFactory.

66.1.5.2.2 TAbstractScriptObjectFactory.CreateComponentObject Method

Pascal

```
function CreateComponentObject(AComponent: TObject; AClassKey: TClass): IInterface; virtual;
abstract;
```

C++

```
virtual __fastcall IInterface CreateComponentObject(TObject * AComponent, TClass AClassKey) =
0;
```

Description

This is CreateComponentObject, a member of class TAbstractScriptObjectFactory.

66.1.5.2.3 TAbstractScriptObjectFactory.CreateGlobalObject Method

Pascal

```
function CreateGlobalObject(AVariable: TGlobalScriptVariables; AContext: IScriptContext):
IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateGlobalObject(TGlobalScriptVariables AVariable,
IScriptContext AContext) = 0;
```

Description

This is CreateGlobalObject, a member of class TAbstractScriptObjectFactory.

66.1.5.2.4 TAbstractScriptObjectFactory.CreateProducerObject Method

Pascal

```
function CreateProducerObject(AVariable: TGlobalScriptVariables; AProducer: IScriptProducer):
IInterface; virtual; abstract;
```

C++

```
virtual __fastcall IInterface CreateProducerObject(TGlobalScriptVariables AVariable,
IScriptProducer AProducer) = 0;
```

Description

This is CreateProducerObject, a member of class TAbstractScriptObjectFactory.

66.1.5.2.5 TAbstractScriptObjectFactory.GetScriptEngine Method

Pascal

```
function GetScriptEngine: TAbstractScriptEngine; virtual; abstract;
```

C++

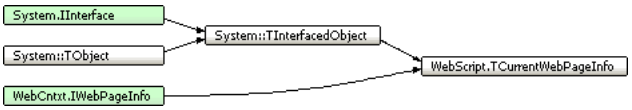
```
virtual __fastcall TAbstractScriptEngine GetScriptEngine() = 0;
```

Description

This is GetScriptEngine, a member of class TAbstractScriptObjectFactory.

66.1.6 TCurrentWebPageInfo Class

Class Hierarchy



Pascal

```
TCurrentWebPageInfo = class(TInterfacedObject, IWebPageInfo);
```

C++

```
class TCurrentWebPageInfo : public TInterfacedObject, public IWebPageInfo;
```

File

WebScript

Description

This is class WebScript.TCurrentWebPageInfo.

66.1.6.1 TCurrentWebPageInfo Members

The following tables list the members exposed by TCurrentWebPageInfo.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetHasViewAccess (see page 6660)	Returns the value of the HasViewAccess property.
	GetLoginRequired (see page 6660)	Returns the value of the LoginRequired property.
	GetPageDescription (see page 6660)	Returns the value of the PageDescription property.
	GetPageHREF (see page 6661)	Returns the value of the PageHREF property.
	GetPageIsPublished (see page 6661)	Returns the value of the PageIsPublished property.
	GetPageName (see page 6661)	Returns the value of the PageName property.
	GetPageTitle (see page 6662)	Returns the value of the PageTitle property.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	WebPageInfo (see page 6662)	This is WebPageInfo, a member of class TCurrentWebPageInfo.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6659)	This is Create, a member of class TCurrentWebPageInfo.

	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	HasViewAccess (see page 6000)	Indicates whether the current user has access rights to view the Web page.
	LoginRequired (see page 6000)	Indicates whether callers must log in before accessing this Web page.
	PageDescription (see page 6000)	Describes the Web page.
	PageHREF (see page 6001)	Identifies the HREF to use for references to the associated Web page.
	PageIsPublished (see page 6001)	Indicates whether the WebSnap application automatically displays the Web page in response to request messages where the pathinfo portion of the target URL matches PageName (see page 6002).
	PageName (see page 6002)	Denotes the logical name for the Web page.
	PageTitle (see page 6002)	Indicates the title that appears on the Web page.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

66.1.6.2 TCurrentWebPageInfo.Create Constructor

Pascal

```
constructor Create(AContext: IScriptContext);
```

C++

```
__fastcall TCurrentWebPageInfo(IScriptContext AContext);
```

Description

This is Create, a member of class TCurrentWebPageInfo.

66.1.6.3 TCurrentWebPageInfo Methods

The methods of the TCurrentWebPageInfo class are listed here.

Protected Methods

	Name	Description
	GetHasViewAccess (see page 6660)	Returns the value of the HasViewAccess property.
	GetLoginRequired (see page 6660)	Returns the value of the LoginRequired property.
	GetPageDescription (see page 6660)	Returns the value of the PageDescription property.
	GetPageHREF (see page 6661)	Returns the value of the PageHREF property.
	GetPageIsPublished (see page 6661)	Returns the value of the PageIsPublished property.
	GetPageName (see page 6661)	Returns the value of the PageName property.
	GetPageTitle (see page 6662)	Returns the value of the PageTitle property.
	WebPageInfo (see page 6662)	This is WebPageInfo, a member of class TCurrentWebPageInfo.

66.1.6.3.1 TCurrentWebPageInfo.GetHasViewAccess Method

Returns the value of the HasViewAccess property.

Pascal

```
function GetHasViewAccess: Boolean;
```

C++

```
__fastcall Boolean GetHasViewAccess();
```

Description

GetHasViewAccess is the read implementation of the HasViewAccess property.

See Also

HasViewAccess (see page 6000)

66.1.6.3.2 TCurrentWebPageInfo.GetLoginRequired Method

Returns the value of the LoginRequired property.

Pascal

```
function GetLoginRequired: Boolean;
```

C++

```
__fastcall Boolean GetLoginRequired();
```

Description

GetLoginRequired is the read implementation of the LoginRequired property.

See Also

LoginRequired (see page 6000)

66.1.6.3.3 TCurrentWebPageInfo.GetPageDescription Method

Returns the value of the PageDescription property.

Pascal

```
function GetPageDescription: string;
```

C++

```
__fastcall UnicodeString GetPageDescription();
```

Description

GetPageDescription is the read implementation of the PageDescription property.

See Also

PageDescription (see page 6000)

66.1.6.3.4 TCurrentWebPageInfo.GetPageHREF Method

Returns the value of the PageHREF property.

Pascal

```
function GetPageHREF: string;
```

C++

```
__fastcall UnicodeString GetPageHREF();
```

Description

GetPageHREF is the read implementation of the PageHREF property.

See Also

PageHREF (see page 6001)

66.1.6.3.5 TCurrentWebPageInfo.GetPageIsPublished Method

Returns the value of the PageIsPublished property.

Pascal

```
function GetPageIsPublished: Boolean;
```

C++

```
__fastcall Boolean GetPageIsPublished();
```

Description

GetPageIsPublished is the read implementation of the PageIsPublished property.

See Also

PageIsPublished (see page 6001)

66.1.6.3.6 TCurrentWebPageInfo.GetPageName Method

Returns the value of the PageName property.

Pascal

```
function GetPageName: string;
```

C++

```
__fastcall UnicodeString GetPageName();
```

Description

GetPageName is the read implementation of the PageName property.

See Also

PageName (see page 6002)

66.1.6.3.7 TCurrentWebPageInfo.GetPageTitle Method

Returns the value of the PageTitle property.

Pascal

```
function GetPageTitle: string;
```

C++

```
__fastcall UnicodeString GetPageTitle();
```

Description

GetPageTitle is the read implementation of the PageTitle property.

See Also

PageTitle (see page 6002)

66.1.6.3.8 TCurrentWebPageInfo.WebPageInfo Method

Pascal

```
function WebPageInfo: IWebPageInfo;
```

C++

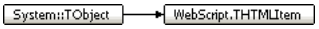
```
__fastcall IWebPageInfo WebPageInfo();
```

Description

This is WebPageInfo, a member of class TCurrentWebPageInfo.

66.1.7 THTMLItem Class

Class Hierarchy



Pascal

```
THTMLItem = class(TObject);
```

C++

```
class THTMLItem : public TObject;
```

File

WebScript

Description

This is class WebScript.THTMLItem.

66.1.7.1 THTMLItem Members

The following tables list the members exposed by THTMLItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	AssignString (see page 6664)	This is AssignString, a member of class THTMLItem.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6664)	This is Create, a member of class THTMLItem.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	HTML (see page 6664)	This is HTML, a member of class THTMLItem.
	Ident (see page 6665)	This is Ident, a member of class THTMLItem.
	ItemLength (see page 6665)	This is ItemLength, a member of class THTMLItem.
	TagToken (see page 6665)	This is TagToken, a member of class THTMLItem.

66.1.7.2 THTMLItem.Create Constructor

Pascal

```
constructor Create(AItems: THTMLItems);
```

C++

```
__fastcall THTMLItem(THTMLItems AItems);
```

Description

This is Create, a member of class THTMLItem.

66.1.7.3 THTMLItem Methods

The methods of the THTMLItem class are listed here.

Public Methods

	Name	Description
	AssignString (see page 6664)	This is AssignString, a member of class THTMLItem.

66.1.7.3.1 THTMLItem.AssignString Method

Pascal

```
procedure AssignString(AOffset: Integer; AHTML: PAnsiChar; ALength: Integer; AEncoding:
TEncoding);
```

C++

```
__fastcall AssignString(int AOffset, PAnsiChar AHTML, int ALength, TEncoding AEncoding);
```

Description

This is AssignString, a member of class THTMLItem.

66.1.7.4 THTMLItem Properties

The properties of the THTMLItem class are listed here.

Public Properties

	Name	Description
R	HTML (see page 6664)	This is HTML, a member of class THTMLItem.
R	Ident (see page 6665)	This is Ident, a member of class THTMLItem.
R	ItemLength (see page 6665)	This is ItemLength, a member of class THTMLItem.
	TagToken (see page 6665)	This is TagToken, a member of class THTMLItem.

66.1.7.4.1 THTMLItem.HTML Property

Pascal

```
property HTML: string;
```

C++

```
__property UnicodeString HTML;
```

Description

This is HTML, a member of class THTMLItem.

66.1.7.4.2 THTMLItem.Ident Property

Pascal

```
property Ident: string;
```

C++

```
__property UnicodeString Ident;
```

Description

This is Ident, a member of class THTMLItem.

66.1.7.4.3 THTMLItem.ItemLength Property

Pascal

```
property ItemLength: Integer;
```

C++

```
__property int ItemLength;
```

Description

This is ItemLength, a member of class THTMLItem.

66.1.7.4.4 THTMLItem.TagToken Property

Pascal

```
property TagToken: Boolean;
```

C++

```
__property Boolean TagToken;
```

Description

This is TagToken, a member of class THTMLItem.

66.1.8 THTMLItems Class

Class Hierarchy

Pascal

```
THTMLItems = class(TObjectList);
```

C++

```
class THTMLItems : public TObjectList;
```

File

WebScript

Description

This is class WebScript.THTMLItems.

66.1.8.1 THTMLItems Members

The following tables list the members exposed by THTMLItems.

Public Methods

	Name	Description
	Create (see page 6666)	This is Create, a member of class THTMLItems.
V	Destroy (see page 6666)	Destroys an instance of TList.
	GetHTMLItem (see page 6667)	This is GetHTMLItem, a member of class THTMLItems.
	MakeIdentifier (see page 6667)	This is MakeIdentifier, a member of class THTMLItems.

Public Properties

	Name	Description
R	Items (see page 6667)	Lists the object references.

66.1.8.2 THTMLItems.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall THTMLItems(TScriptProducer AProducer);
```

Description

This is Create, a member of class THTMLItems.

66.1.8.3 THTMLItems.Destroy Destructor

Destroys an instance of TList.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~THTMLItems();
```

Description

Do not call Destroy directly. Instead, call Free. Free verifies that the TList reference is not nil, and only then calls Destroy.

Destroy frees the memory used to store the list of items.

Note: Destroy does not free the memory pointed to by the elements of the list.

See Also

System::TObject::Free (see page 3372)

66.1.8.4 THTMLItems Methods

The methods of the THTMLItems class are listed here.

Public Methods

	Name	Description
	GetHTMLItem (see page 6667)	This is GetHTMLItem, a member of class THTMLItems.
	MakeIdentifier (see page 6667)	This is MakeIdentifier, a member of class THTMLItems.

66.1.8.4.1 THTMLItems.GetHTMLItem Method

Pascal

```
function GetHTMLItem(I: Integer): THTMLItem;
```

C++

```
__fastcall THTMLItem GetHTMLItem(int I);
```

Description

This is GetHTMLItem, a member of class THTMLItems.

66.1.8.4.2 THTMLItems.MakeIdentifier Method

Pascal

```
function MakeIdentifier(AItem: THTMLItem): string;
```

C++

```
__fastcall UnicodeString MakeIdentifier(THTMLItem AItem);
```

Description

This is MakeIdentifier, a member of class THTMLItems.

66.1.8.5 THTMLItems Properties

The properties of the THTMLItems class are listed here.

Public Properties

	Name	Description
R	Items (see page 6667)	Lists the object references.

66.1.8.5.1 THTMLItems.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: THTMLItem;
```

C++

```
__property THTMLItem Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects

Add

Insert

Delete

Remove

List

Extract

Count

Clear

Pack

66.1.9 TLineNumberMap Class

Class Hierarchy

Pascal

```
TLineNumberMap = class(TObject);
```

C++

```
class TLineNumberMap : public TObject;
```

File

WebScript

Description

This is class WebScript.TLineNumberMap.

66.1.9.1 TLineNumberMap Members

The following tables list the members exposed by TLineNumberMap.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	DestLineToSourceLine (see page 6670)	This is DestLineToSourceLine, a member of class TLineNumberMap.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	New (see page 6670)	This is New, a member of class TLineNumberMap.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Count (see page 6670)	This is Count, a member of class TLineNumberMap.
	Items (see page 6671)	This is Items, a member of class TLineNumberMap.

66.1.9.2 TLineNumberMap Methods

The methods of the TLineNumberMap class are listed here.

Public Methods

	Name	Description
	DestLineToSourceLine (see page 6670)	This is DestLineToSourceLine, a member of class TLineNumberMap.
	New (see page 6670)	This is New, a member of class TLineNumberMap.

66.1.9.2.1 TLineNumberMap.DestLineToSourceLine Method

Pascal

```
function DestLineToSourceLine(ALine: Integer; var AFileName: string): Integer;
```

C++

```
__fastcall int DestLineToSourceLine(int ALine, UnicodeString AFileName);
```

Description

This is DestLineToSourceLine, a member of class TLineNumberMap.

66.1.9.2.2 TLineNumberMap.New Method

Pascal

```
function New: PLineNumbers;
```

C++

```
__fastcall PLineNumbers New();
```

Description

This is New, a member of class TLineNumberMap.

66.1.9.3 TLineNumberMap Properties

The properties of the TLineNumberMap class are listed here.

Public Properties

	Name	Description
R	Count (see page 6670)	This is Count, a member of class TLineNumberMap.
R	Items (see page 6671)	This is Items, a member of class TLineNumberMap.

66.1.9.3.1 TLineNumberMap.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TLineNumberMap.

66.1.9.3.2 TLineNumberMap.Items Property

Pascal

```
property Items [I: Integer]: PLineNumbers;
```

C++

```
__property PLineNumbers Items[int I];
```

Description

This is Items, a member of class TLineNumberMap.

66.1.10 TScriptCodeItem Class

Class Hierarchy

Pascal

```
TScriptCodeItem = class(TScriptItem);
```

C++

```
class TScriptCodeItem : public TScriptItem;
```

File

WebScript

Description

This is class WebScript.TScriptCodeItem.

66.1.10.1 TScriptCodeItem Members

The following tables list the members exposed by TScriptCodeItem.

Protected Methods

	Name	Description
	GetScript (see page 6673)	This is GetScript, a member of class TScriptCodeItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	AssignString (see page 6672)	This is AssignString, a member of class TScriptCodeItem.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.

	Create (see page 6699)	This is Create, a member of class TScriptItem.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Script (see page 6700)	This is Script, a member of class TScriptItem.

66.1.10.2 TScriptCodeItem Methods

The methods of the TScriptCodeItem class are listed here.

Protected Methods

	Name	Description
V	GetScript (see page 6673)	This is GetScript, a member of class TScriptCodeItem.

Public Methods

	Name	Description
	AssignString (see page 6672)	This is AssignString, a member of class TScriptCodeItem.

66.1.10.2.1 TScriptCodeItem.AssignString Method

Pascal

```
procedure AssignString(const AValue: string);
```

C++

```
__fastcall AssignString(const UnicodeString AValue);
```

Description

This is AssignString, a member of class TScriptCodeItem.

66.1.10.2.2 TScriptCodeItem.GetScript Method

Pascal

```
function GetScript: string; override;
```

C++

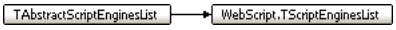
```
virtual __fastcall UnicodeString GetScript();
```

Description

This is GetScript, a member of class TScriptCodeItem.

66.1.11 TScriptEnginesList Class

Class Hierarchy



Pascal

```
TScriptEnginesList = class(TAbstractScriptEnginesList);
```

C++

```
class TScriptEnginesList : public TAbstractScriptEnginesList;
```

File

WebScript

Description

This is class WebScript.TScriptEnginesList.

66.1.11.1 TScriptEnginesList Members

The following tables list the members exposed by TScriptEnginesList.

Public Methods

	Name	Description
	AddEngine (see page 6675)	This is AddEngine, a member of class TScriptEnginesList.
	AddObjectFactory (see page 6675)	This is AddObjectFactory, a member of class TScriptEnginesList.
	Create (see page 6674)	Constructs an object and initializes its data before the object is first used.
V	Destroy (see page 6674)	Disposes of an object instance.
	Find (see page 6675)	This is Find, a member of class TScriptEnginesList.
V	FindScriptProducerClass (see page 6676)	This is FindScriptProducerClass, a member of class TScriptEnginesList.
	ListLanguageNames (see page 6676)	This is ListLanguageNames, a member of class TScriptEnginesList.
	RemoveEngine (see page 6676)	This is RemoveEngine, a member of class TScriptEnginesList.

Public Properties

	Name	Description
R	Count (see page 6676)	This is Count, a member of class TScriptEnginesList.
R	Items (see page 6677)	This is Items, a member of class TScriptEnginesList.

66.1.11.2 TScriptEnginesList.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptEnginesList();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

66.1.11.3 TScriptEnginesList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptEnginesList();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

66.1.11.4 TScriptEnginesList Methods

The methods of the TScriptEnginesList class are listed here.

Public Methods

	Name	Description
	AddEngine (see page 6675)	This is AddEngine, a member of class TScriptEnginesList.
	AddObjectFactory (see page 6675)	This is AddObjectFactory, a member of class TScriptEnginesList.
	Find (see page 6675)	This is Find, a member of class TScriptEnginesList.
	FindScriptProducerClass (see page 6676)	This is FindScriptProducerClass, a member of class TScriptEnginesList.
	ListLanguageNames (see page 6676)	This is ListLanguageNames, a member of class TScriptEnginesList.
	RemoveEngine (see page 6676)	This is RemoveEngine, a member of class TScriptEnginesList.

66.1.11.4.1 TScriptEnginesList.AddEngine Method

Pascal

```
procedure AddEngine(AEngine: TAbstractScriptEngine);
```

C++

```
__fastcall AddEngine(TAbstractScriptEngine AEngine);
```

Description

This is AddEngine, a member of class TScriptEnginesList.

66.1.11.4.2 TScriptEnginesList.AddObjectFactory Method

Pascal

```
procedure AddObjectFactory(AFactory: TAbstractScriptObjectFactory; const AEngineName: string);
```

C++

```
__fastcall AddObjectFactory(TAbstractScriptObjectFactory AFactory, const UnicodeString
AEngineName);
```

Description

This is AddObjectFactory, a member of class TScriptEnginesList.

66.1.11.4.3 TScriptEnginesList.Find Method

Pascal

```
function Find(const AName: string): TAbstractScriptEngine;
```

C++

```
__fastcall TAbstractScriptEngine Find(const UnicodeString AName);
```

Description

This is Find, a member of class TScriptEnginesList.

66.1.11.4.4 TScriptEnginesList.FindScriptProducerClass Method

Pascal

```
function FindScriptProducerClass(const ALanguageName: string): TScriptProducerClass; override;
```

C++

```
virtual __fastcall TScriptProducerClass FindScriptProducerClass(const UnicodeString
ALanguageName);
```

Description

This is FindScriptProducerClass, a member of class TScriptEnginesList.

66.1.11.4.5 TScriptEnginesList.ListLanguageNames Method

Pascal

```
procedure ListLanguageNames(AList: TStrings);
```

C++

```
__fastcall ListLanguageNames(TStrings AList);
```

Description

This is ListLanguageNames, a member of class TScriptEnginesList.

66.1.11.4.6 TScriptEnginesList.RemoveEngine Method

Pascal

```
procedure RemoveEngine(AEngine: TAbstractScriptEngine);
```

C++

```
__fastcall RemoveEngine(TAbstractScriptEngine AEngine);
```

Description

This is RemoveEngine, a member of class TScriptEnginesList.

66.1.11.5 TScriptEnginesList Properties

The properties of the TScriptEnginesList class are listed here.

Public Properties

	Name	Description
	Count (see page 6676)	This is Count, a member of class TScriptEnginesList.
	Items (see page 6677)	This is Items, a member of class TScriptEnginesList.

66.1.11.5.1 TScriptEnginesList.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TScriptEnginesList.

66.1.11.5.2 TScriptEnginesList.Items Property

Pascal

```
property Items [I: Integer]: TAbstractScriptEngine;
```

C++

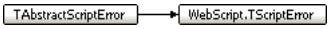
```
__property TAbstractScriptEngine Items[int I];
```

Description

This is Items, a member of class TScriptEnginesList.

66.1.12 TScriptError Class

Class Hierarchy



Pascal

```
TScriptError = class(TAbstractScriptError);
```

C++

```
class TScriptError : public TAbstractScriptError;
```

File

WebScript

Description

This is class WebScript.TScriptError.

66.1.12.1 TScriptError Members

The following tables list the members exposed by TScriptError.

Protected Fields

	Name	Description
	FCharPos (see page 6678)	This is FCharPos, a member of class TScriptError.
	FDescription (see page 6678)	This is FDescription, a member of class TScriptError.
	FFileName (see page 6679)	This is FFileName, a member of class TScriptError.
	FLine (see page 6679)	This is FLine, a member of class TScriptError.
	FSourceLine (see page 6679)	This is FSourceLine, a member of class TScriptError.

Protected Methods

	Name	Description
	GetCharPos (see page 6679)	This is GetCharPos, a member of class TScriptError.
	GetDescription (see page 6680)	This is GetDescription, a member of class TScriptError.
	GetFileName (see page 6680)	This is GetFileName, a member of class TScriptError.
	GetLine (see page 6680)	This is GetLine, a member of class TScriptError.

	GetSourceLine (see page 6680)	This is GetSourceLine, a member of class TScriptError.
	SetDescription (see page 6681)	This is SetDescription, a member of class TScriptError.

Public Methods

	Name	Description
	Create (see page 6678)	This is Create, a member of class TScriptError.

66.1.12.2 TScriptError.Create Constructor

Pascal

```
constructor Create(E: Exception); overload;
```

C++

```
__fastcall TScriptError(Exception E);
```

Description

This is Create, a member of class TScriptError.

66.1.12.3 TScriptError Fields

The fields of the TScriptError class are listed here.

Protected Fields

	Name	Description
	FCharPos (see page 6678)	This is FCharPos, a member of class TScriptError.
	FDescription (see page 6678)	This is FDescription, a member of class TScriptError.
	FFileName (see page 6679)	This is FFileName, a member of class TScriptError.
	FLine (see page 6679)	This is FLine, a member of class TScriptError.
	FSourceLine (see page 6679)	This is FSourceLine, a member of class TScriptError.

66.1.12.3.1 TScriptError.FCharPos Field

Pascal

```
FCharPos: Integer;
```

C++

```
int FCharPos;
```

Description

This is FCharPos, a member of class TScriptError.

66.1.12.3.2 TScriptError.FDescription Field

Pascal

```
FDescription: string;
```

C++

```
UnicodeString FDescription;
```

Description

This is FDescription, a member of class TScriptError.

66.1.12.3.3 TScriptError.FFileName Field

Pascal

```
FFileName: string;
```

C++

```
UnicodeString FFileName;
```

Description

This is FFileName, a member of class TScriptError.

66.1.12.3.4 TScriptError.FLine Field

Pascal

```
FLine: Integer;
```

C++

```
int FLine;
```

Description

This is FLine, a member of class TScriptError.

66.1.12.3.5 TScriptError.FSourceLine Field

Pascal

```
FSourceLine: string;
```

C++

```
UnicodeString FSourceLine;
```

Description

This is FSourceLine, a member of class TScriptError.

66.1.12.4 TScriptError Methods

The methods of the TScriptError class are listed here.

Protected Methods

	Name	Description
	GetCharPos (see page 6679)	This is GetCharPos, a member of class TScriptError.
	GetDescription (see page 6680)	This is GetDescription, a member of class TScriptError.
	GetFileName (see page 6680)	This is GetFileName, a member of class TScriptError.
	GetLine (see page 6680)	This is GetLine, a member of class TScriptError.
	GetSourceLine (see page 6680)	This is GetSourceLine, a member of class TScriptError.
	SetDescription (see page 6681)	This is SetDescription, a member of class TScriptError.

66.1.12.4.1 TScriptError.GetCharPos Method

Pascal

```
function GetCharPos: Integer; override;
```

C++

```
virtual __fastcall int GetCharPos();
```

Description

This is GetCharPos, a member of class TScriptError.

66.1.12.4.2 TScriptError.GetDescription Method

Pascal

```
function GetDescription: string; override;
```

C++

```
virtual __fastcall UnicodeString GetDescription();
```

Description

This is GetDescription, a member of class TScriptError.

66.1.12.4.3 TScriptError.GetFileName Method

Pascal

```
function GetFileName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetFileName();
```

Description

This is GetFileName, a member of class TScriptError.

66.1.12.4.4 TScriptError.GetLine Method

Pascal

```
function GetLine: Integer; override;
```

C++

```
virtual __fastcall int GetLine();
```

Description

This is GetLine, a member of class TScriptError.

66.1.12.4.5 TScriptError.GetSourceLine Method

Pascal

```
function GetSourceLine: string; override;
```

C++

```
virtual __fastcall UnicodeString GetSourceLine();
```

Description

This is GetSourceLine, a member of class TScriptError.

66.1.12.4.6 TScriptError.SetDescription Method

Pascal

```
procedure SetDescription(const AValue: string); override;
```

C++

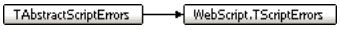
```
virtual __fastcall SetDescription(const UnicodeString AValue);
```

Description

This is SetDescription, a member of class TScriptError.

66.1.13 TScriptErrors Class

Class Hierarchy



Pascal

```
TScriptErrors = class(TAbstractScriptErrors);
```

C++

```
class TScriptErrors : public TAbstractScriptErrors;
```

File

WebScript

Description

This is class WebScript.TScriptErrors.

66.1.13.1 TScriptErrors Members

The following tables list the members exposed by TScriptErrors.

Protected Methods

	Name	Description
	GetError (see page 6683)	This is GetError, a member of class TScriptErrors.
	GetErrorCount (see page 6683)	This is GetErrorCount, a member of class TScriptErrors.

Public Methods

	Name	Description
	Add (see page 6683)	This is Add, a member of class TScriptErrors.
	Create (see page 6681)	Constructs an object and initializes its data before the object is first used.
	Destroy (see page 6682)	Disposes of an object instance.

66.1.13.2 TScriptErrors.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptErrors();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

66.1.13.3 TScriptErrors.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptErrors();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

66.1.13.4 TScriptErrors Methods

The methods of the TScriptErrors class are listed here.

Protected Methods

	Name	Description
	GetError (see page 6683)	This is GetError, a member of class TScriptErrors.
	GetErrorCount (see page 6683)	This is GetErrorCount, a member of class TScriptErrors.

Public Methods

	Name	Description
	Add (see page 6683)	This is Add, a member of class TScriptErrors.

66.1.13.4.1 TScriptErrors.Add Method

Pascal

```
procedure Add(const AError: TAbstractScriptError); override;
```

C++

```
virtual __fastcall Add(const TAbstractScriptError AError);
```

Description

This is Add, a member of class TScriptErrors.

66.1.13.4.2 TScriptErrors.GetError Method

Pascal

```
function GetError(I: Integer): TAbstractScriptError; override;
```

C++

```
virtual __fastcall TAbstractScriptError GetError(int I);
```

Description

This is GetError, a member of class TScriptErrors.

66.1.13.4.3 TScriptErrors.GetErrorCount Method

Pascal

```
function GetErrorCount: Integer; override;
```

C++

```
virtual __fastcall int GetErrorCount();
```

Description

This is GetErrorCount, a member of class TScriptErrors.

66.1.14 TScriptFile Class

Class Hierarchy

Pascal

```
TScriptFile = class;
```

C++

```
class TScriptFile;
```

File

WebScript

Description

This is class WebScript.TScriptFile.

66.1.14.1 TScriptFile Members

The following tables list the members exposed by TScriptFile.

Public Fields

	Name	Description
	FLineNumberMap (see page 6685)	This is FLineNumberMap, a member of class TScriptFile.

Public Methods

	Name	Description
	Create (see page 6684)	This is Create, a member of class TScriptFile.
V	Destroy (see page 6684)	This is Destroy, a member of class TScriptFile.
	ParseStream (see page 6685)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 6685)	This is ParseString, a member of class TScriptFile.

Public Properties

	Name	Description
R	ApplicationName (see page 6686)	This is ApplicationName, a member of class TScriptFile.
R	LongName (see page 6686)	This is LongName, a member of class TScriptFile.
R	Script (see page 6686)	This is Script, a member of class TScriptFile.
R	ShortName (see page 6687)	This is ShortName, a member of class TScriptFile.
R	Source (see page 6687)	This is Source, a member of class TScriptFile.

66.1.14.2 TScriptFile.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptFile(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptFile.

66.1.14.3 TScriptFile.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptFile();
```

Description

This is Destroy, a member of class TScriptFile.

66.1.14.4 TScriptFile Fields

The fields of the TScriptFile class are listed here.

Public Fields

	Name	Description
	FLineNumberMap (see page 6685)	This is FLineNumberMap, a member of class TScriptFile.

66.1.14.4.1 TScriptFile.FLineNumberMap Field

Pascal

```
FLineNumberMap: TLineNumberMap;
```

C++

```
TLineNumberMap FLineNumberMap;
```

Description

This is FLineNumberMap, a member of class TScriptFile.

66.1.14.5 TScriptFile Methods

The methods of the TScriptFile class are listed here.

Public Methods

	Name	Description
	ParseStream (see page 6685)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 6685)	This is ParseString, a member of class TScriptFile.

66.1.14.5.1 TScriptFile.ParseStream Method

Pascal

```
procedure ParseStream(ASource: TStream; AOwned: Boolean);
```

C++

```
__fastcall ParseStream(TStream ASource, Boolean AOwned);
```

Description

This is ParseStream, a member of class TScriptFile.

66.1.14.5.2 TScriptFile.ParseString Method

Pascal

```
procedure ParseString(const ASource: string);
```

C++

```
__fastcall ParseString(const UnicodeString ASource);
```

Description

This is ParseString, a member of class TScriptFile.

66.1.14.6 TScriptFile Properties

The properties of the TScriptFile class are listed here.

Public Properties

	Name	Description
	ApplicationName (see page 6686)	This is ApplicationName, a member of class TScriptFile.
	LongName (see page 6686)	This is LongName, a member of class TScriptFile.
	Script (see page 6686)	This is Script, a member of class TScriptFile.
	ShortName (see page 6687)	This is ShortName, a member of class TScriptFile.
	Source (see page 6687)	This is Source, a member of class TScriptFile.

66.1.14.6.1 TScriptFile.ApplicationName Property

Pascal

```
property ApplicationName: string;
```

C++

```
__property UnicodeString ApplicationName;
```

Description

This is ApplicationName, a member of class TScriptFile.

66.1.14.6.2 TScriptFile.LongName Property

Pascal

```
property LongName: string;
```

C++

```
__property UnicodeString LongName;
```

Description

This is LongName, a member of class TScriptFile.

66.1.14.6.3 TScriptFile.Script Property

Pascal

```
property Script: string;
```

C++

```
__property UnicodeString Script;
```

Description

This is Script, a member of class TScriptFile.

66.1.14.6.4 TScriptFile.ShortName Property

Pascal

```
property ShortName: string;
```

C++

```
__property UnicodeString ShortName;
```

Description

This is ShortName, a member of class TScriptFile.

66.1.14.6.5 TScriptFile.Source Property

Pascal

```
property Source: string;
```

C++

```
__property UnicodeString Source;
```

Description

This is Source, a member of class TScriptFile.

66.1.15 TScriptGlobalObjects Class

Class Hierarchy

Pascal

```
TScriptGlobalObjects = class(TObject);
```

C++

```
class TScriptGlobalObjects : public TObject;
```

File

WebScript

Description

This is class WebScript.TScriptGlobalObjects.

66.1.15.1 TScriptGlobalObjects Members

The following tables list the members exposed by TScriptGlobalObjects.

Protected Methods

	Name	Description
	GetScriptEngine (see page 6690)	This is GetScriptEngine, a member of class TScriptGlobalObjects.

Public Methods

	Name	Description
	AddNamedIntf (see page 6690)	This is AddNamedIntf, a member of class TScriptGlobalObjects.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6688)	This is Create, a member of class TScriptGlobalObjects.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6689)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindNamedItemIntf (see page 6690)	This is FindNamedItemIntf, a member of class TScriptGlobalObjects.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetNamedItemCount (see page 6690)	This is GetNamedItemCount, a member of class TScriptGlobalObjects.
	GetNamedItemName (see page 6690)	This is GetNamedItemName, a member of class TScriptGlobalObjects.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	NamedItemCount (see page 6691)	This is NamedItemCount, a member of class TScriptGlobalObjects.
R	NamedItemName (see page 6691)	This is NamedItemName, a member of class TScriptGlobalObjects.
R	Producer (see page 6691)	This is Producer, a member of class TScriptGlobalObjects.
R	ScriptEngine (see page 6692)	This is ScriptEngine, a member of class TScriptGlobalObjects.

66.1.15.2 TScriptGlobalObjects.Create Constructor

Pascal

```
constructor Create(AProducer: IScriptProducer);
```

C++

```
__fastcall TScriptGlobalObjects(IScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptGlobalObjects.

66.1.15.3 TScriptGlobalObjects.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptGlobalObjects();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

66.1.15.4 TScriptGlobalObjects Methods

The methods of the TScriptGlobalObjects class are listed here.

Protected Methods

	Name	Description
	GetScriptEngine (see page 6690)	This is GetScriptEngine, a member of class TScriptGlobalObjects.

Public Methods

	Name	Description
	AddNamedIntf (see page 6690)	This is AddNamedIntf, a member of class TScriptGlobalObjects.
	FindNamedItemIntf (see page 6690)	This is FindNamedItemIntf, a member of class TScriptGlobalObjects.
	GetNamedItemCount (see page 6690)	This is GetNamedItemCount, a member of class TScriptGlobalObjects.
	GetNamedItemName (see page 6690)	This is GetNamedItemName, a member of class TScriptGlobalObjects.

66.1.15.4.1 TScriptGlobalObjects.AddNamedIntf Method

Pascal

```
procedure AddNamedIntf(const AName: string; AIntf: IUnknown);
```

C++

```
__fastcall AddNamedIntf(const UnicodeString AName, IUnknown AIntf);
```

Description

This is AddNamedIntf, a member of class TScriptGlobalObjects.

66.1.15.4.2 TScriptGlobalObjects.FindNamedItemIntf Method

Pascal

```
function FindNamedItemIntf(const AName: string): IUnknown;
```

C++

```
__fastcall IUnknown FindNamedItemIntf(const UnicodeString AName);
```

Description

This is FindNamedItemIntf, a member of class TScriptGlobalObjects.

66.1.15.4.3 TScriptGlobalObjects.GetNamedItemCount Method

Pascal

```
function GetNamedItemCount: Integer;
```

C++

```
__fastcall int GetNamedItemCount();
```

Description

This is GetNamedItemCount, a member of class TScriptGlobalObjects.

66.1.15.4.4 TScriptGlobalObjects.GetNamedItemName Method

Pascal

```
function GetNamedItemName(I: Integer): string;
```

C++

```
__fastcall UnicodeString GetNamedItemName(int I);
```

Description

This is GetNamedItemName, a member of class TScriptGlobalObjects.

66.1.15.4.5 TScriptGlobalObjects.GetScriptEngine Method

Pascal

```
function GetScriptEngine: TAbstractScriptEngine; virtual; abstract;
```

C++

```
virtual __fastcall TAbstractScriptEngine GetScriptEngine() = 0;
```

Description

This is GetScriptEngine, a member of class TScriptGlobalObjects.

66.1.15.5 TScriptGlobalObjects Properties

The properties of the TScriptGlobalObjects class are listed here.

Public Properties

	Name	Description
	NamedItemCount (see page 6691)	This is NamedItemCount, a member of class TScriptGlobalObjects.
	NamedItemName (see page 6691)	This is NamedItemName, a member of class TScriptGlobalObjects.
	Producer (see page 6691)	This is Producer, a member of class TScriptGlobalObjects.
	ScriptEngine (see page 6692)	This is ScriptEngine, a member of class TScriptGlobalObjects.

66.1.15.5.1 TScriptGlobalObjects.NamedItemCount Property

Pascal

```
property NamedItemCount: Integer;
```

C++

```
__property int NamedItemCount;
```

Description

This is NamedItemCount, a member of class TScriptGlobalObjects.

66.1.15.5.2 TScriptGlobalObjects.NamedItemName Property

Pascal

```
property NamedItemName [I: Integer]: string;
```

C++

```
__property UnicodeString NamedItemName[int I];
```

Description

This is NamedItemName, a member of class TScriptGlobalObjects.

66.1.15.5.3 TScriptGlobalObjects.Producer Property

Pascal

```
property Producer: IScriptProducer;
```

C++

```
__property IScriptProducer Producer;
```

Description

This is Producer, a member of class TScriptGlobalObjects.

66.1.15.5.4 TScriptGlobalObjects.ScriptEngine Property

Pascal

```
property ScriptEngine: TAbstractScriptEngine;
```

C++

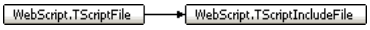
```
__property TAbstractScriptEngine ScriptEngine;
```

Description

This is ScriptEngine, a member of class TScriptGlobalObjects.

66.1.16 TScriptIncludeFile Class

Class Hierarchy



Pascal

```
TScriptIncludeFile = class(TScriptFile);
```

C++

```
class TScriptIncludeFile : public TScriptFile;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeFile.

66.1.16.1 TScriptIncludeFile Members

The following tables list the members exposed by TScriptIncludeFile.

Public Fields

	Name	Description
	FLineNumberMap (see page 6685)	This is FLineNumberMap, a member of class TScriptFile.

Public Methods

	Name	Description
	Create (see page 6693)	This is Create, a member of class TScriptIncludeFile.
V	Destroy (see page 6684)	This is Destroy, a member of class TScriptFile.
	LoadFile (see page 6693)	This is LoadFile, a member of class TScriptIncludeFile.
	ParseStream (see page 6685)	This is ParseStream, a member of class TScriptFile.
	ParseString (see page 6685)	This is ParseString, a member of class TScriptFile.

Public Properties

	Name	Description
R	ApplicationName (see page 6686)	This is ApplicationName, a member of class TScriptFile.
R	LongName (see page 6686)	This is LongName, a member of class TScriptFile.
R	Script (see page 6686)	This is Script, a member of class TScriptFile.

	ShortName (see page 6687)	This is ShortName, a member of class TScriptFile.
	Source (see page 6687)	This is Source, a member of class TScriptFile.

66.1.16.2 TScriptIncludeFile.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer; AIncludeType: TIncludeType; const AFileName:
string);
```

C++

```
__fastcall TScriptIncludeFile(TScriptProducer AProducer, TIncludeType AIncludeType, const
UnicodeString AFileName);
```

Description

This is Create, a member of class TScriptIncludeFile.

66.1.16.3 TScriptIncludeFile Methods

The methods of the TScriptIncludeFile class are listed here.

Public Methods

	Name	Description
	LoadFile (see page 6693)	This is LoadFile, a member of class TScriptIncludeFile.

66.1.16.3.1 TScriptIncludeFile.LoadFile Method

Pascal

```
procedure LoadFile;
```

C++

```
__fastcall LoadFile();
```

Description

This is LoadFile, a member of class TScriptIncludeFile.

66.1.17 TScriptIncludeFiles Class

Class Hierarchy

Pascal

```
TScriptIncludeFiles = class(TObjectList);
```

C++

```
class TScriptIncludeFiles : public TObjectList;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeFiles.

66.1.17.1 TScriptIncludeFiles Members

The following tables list the members exposed by TScriptIncludeFiles.

Public Methods

	Name	Description
	Create (see page 6694)	This is Create, a member of class TScriptIncludeFiles.
	GetScriptIncludeFile (see page 6694)	This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

Public Properties

	Name	Description
	Items (see page 6695)	Lists the object references.

66.1.17.2 TScriptIncludeFiles.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptIncludeFiles(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptIncludeFiles.

66.1.17.3 TScriptIncludeFiles Methods

The methods of the TScriptIncludeFiles class are listed here.

Public Methods

	Name	Description
	GetScriptIncludeFile (see page 6694)	This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

66.1.17.3.1 TScriptIncludeFiles.GetScriptIncludeFile Method

Pascal

```
function GetScriptIncludeFile(I: Integer): TScriptIncludeFile;
```

C++

```
__fastcall TScriptIncludeFile GetScriptIncludeFile(int I);
```

Description

This is GetScriptIncludeFile, a member of class TScriptIncludeFiles.

66.1.17.4 TScriptIncludeFiles Properties

The properties of the TScriptIncludeFiles class are listed here.

Public Properties

	Name	Description
	Items (see page 6695)	Lists the object references.

66.1.17.4.1 TScriptIncludeFiles.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: TScriptIncludeFile;
```

C++

```
__property TScriptIncludeFile Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects

Add

Insert

Delete

Remove

List

Extract

Count

Clear

Pack

66.1.18 TScriptIncludeItem Class

Class Hierarchy

Pascal

```
TScriptIncludeItem = class(TScriptItem);
```

C++

```
class TScriptIncludeItem : public TScriptItem;
```

File

WebScript

Description

This is class WebScript.TScriptIncludeItem.

66.1.18.1 TScriptIncludeItem Members

The following tables list the members exposed by TScriptIncludeItem.

Protected Methods

	Name	Description
	GetScript (see page 6697)	This is GetScript, a member of class TScriptIncludeItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6697)	This is Create, a member of class TScriptIncludeItem.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Script (see page 6700)	This is Script, a member of class TScriptItem.

66.1.18.2 TScriptIncludeItem.Create Constructor

Pascal

```
constructor Create(AItems: TScriptItems; AIncludeFile: TScriptIncludeFile);
```

C++

```
__fastcall TScriptIncludeItem(TScriptItems AItems, TScriptIncludeFile AIncludeFile);
```

Description

This is Create, a member of class TScriptIncludeItem.

66.1.18.3 TScriptIncludeItem Methods

The methods of the TScriptIncludeItem class are listed here.

Protected Methods

	Name	Description
V	GetScript (see page 6697)	This is GetScript, a member of class TScriptIncludeItem.

66.1.18.3.1 TScriptIncludeItem.GetScript Method

Pascal

```
function GetScript: string; override;
```

C++

```
virtual __fastcall UnicodeString GetScript();
```

Description

This is GetScript, a member of class TScriptIncludeItem.

66.1.19 TScriptItem Class

Class Hierarchy

Pascal

```
TScriptItem = class(TObject);
```

C++

```
class TScriptItem : public TObject;
```

File

WebScript

Description

This is class WebScript.TScriptItem.

66.1.19.1 TScriptItem Members

The following tables list the members exposed by TScriptItem.

Protected Methods

	Name	Description
	GetScript (see page 6699)	This is GetScript, a member of class TScriptItem.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6699)	This is Create, a member of class TScriptItem.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.

	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	Script (see page 6700)	This is Script, a member of class TScriptItem.

66.1.19.2 TScriptItem.Create Constructor

Pascal

```
constructor Create(AItems: TScriptItems);
```

C++

```
__fastcall TScriptItem(TScriptItems AItems);
```

Description

This is Create, a member of class TScriptItem.

66.1.19.3 TScriptItem Methods

The methods of the TScriptItem class are listed here.

Protected Methods

	Name	Description
V	GetScript (see page 6699)	This is GetScript, a member of class TScriptItem.

66.1.19.3.1 TScriptItem.GetScript Method

Pascal

```
function GetScript: string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetScript();
```

Description

This is GetScript, a member of class TScriptItem.

66.1.19.4 TScriptItem Properties

The properties of the TScriptItem class are listed here.

Public Properties

	Name	Description
	Script (see page 6700)	This is Script, a member of class TScriptItem.

66.1.19.4.1 TScriptItem.Script Property

Pascal

```
property Script: string;
```

C++

```
__property UnicodeString Script;
```

Description

This is Script, a member of class TScriptItem.

66.1.20 TScriptItems Class

Class Hierarchy

Pascal

```
TScriptItems = class(TObjectList);
```

C++

```
class TScriptItems : public TObjectList;
```

File

WebScript

Description

This is class WebScript.TScriptItems.

66.1.20.1 TScriptItems Members

The following tables list the members exposed by TScriptItems.

Public Methods

	Name	Description
	Create (see page 6701)	This is Create, a member of class TScriptItems.
	GetScriptItem (see page 6701)	This is GetScriptItem, a member of class TScriptItems.

Public Properties

	Name	Description
	Items (see page 6701)	Lists the object references.

66.1.20.2 TScriptItems.Create Constructor

Pascal

```
constructor Create(AProducer: TScriptProducer);
```

C++

```
__fastcall TScriptItems(TScriptProducer AProducer);
```

Description

This is Create, a member of class TScriptItems.

66.1.20.3 TScriptItems Methods

The methods of the TScriptItems class are listed here.

Public Methods

	Name	Description
	GetScriptItem (see page 6701)	This is GetScriptItem, a member of class TScriptItems.

66.1.20.3.1 TScriptItems.GetScriptItem Method

Pascal

```
function GetScriptItem(I: Integer): TScriptItem;
```

C++

```
__fastcall TScriptItem GetScriptItem(int I);
```

Description

This is GetScriptItem, a member of class TScriptItems.

66.1.20.4 TScriptItems Properties

The properties of the TScriptItems class are listed here.

Public Properties

	Name	Description
R	Items (see page 6701)	Lists the object references.

66.1.20.4.1 TScriptItems.Items Property

Lists the object references.

Pascal

```
property Items [Index: Integer]: TScriptItem;
```

C++

```
__property TScriptItem Items[int Index];
```

Description

Use Items to access objects in the list. Items is a zero-based array: The first object is indexed as 0, the second object is indexed

as 1, and so forth. You can read or change the value at a specific index, or use Items with the Count property to iterate through the list.

If OwnsObjects is true, reassigning an Items index frees the object that previously occupied that position in the list.

Items can contain nil (Delphi) or NULL (C++) references. To remove nil (Delphi) or NULL (C++) references and reduce the size of the array, call the Pack method.

Note: In Delphi, Items is the default property of TObjectList. This means you can omit the property name. Thus, the following two lines of code are both acceptable and do the same thing:

Note: ObjectList1.Items[i] := Form1.FindChildControl(ListBox1.Items[i]);

Note: ObjectList1[i] := Form1.FindChildControl(ListBox1.Items[i]);

See Also

OwnsObjects

Add

Insert

Delete

Remove

List

Extract

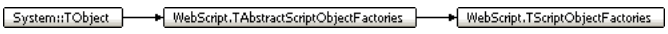
Count

Clear

Pack

66.1.21 TScriptObjectFactories Class

Class Hierarchy



Pascal

```
TScriptObjectFactories = class(TAbstractScriptObjectFactories);
```

C++

```
class TScriptObjectFactories : public TAbstractScriptObjectFactories;
```

File

WebScript

Description

This is class WebScript.TScriptObjectFactories.

66.1.21.1 TScriptObjectFactories Members

The following tables list the members exposed by TScriptObjectFactories.

Protected Methods

	Name	Description
	GetCount (see page 6705)	This is GetCount, a member of class TScriptObjectFactories.
	GetFactory (see page 6705)	This is GetFactory, a member of class TScriptObjectFactories.
	GetObjectList (see page 6706)	This is GetObjectList, a member of class TScriptObjectFactories.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6704)	Constructs an object and initializes its data before the object is first used.
	CreateGlobalObject (see page 6705)	This is CreateGlobalObject, a member of class TScriptObjectFactories.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 6704)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Count (see page 6706)	This is Count, a member of class TScriptObjectFactories.
	Factories (see page 6706)	This is Factories, a member of class TScriptObjectFactories.
	ObjectList (see page 6655)	This is ObjectList, a member of class TAbstractScriptObjectFactories.

66.1.21.2 TScriptObjectFactories.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TScriptObjectFactories();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

66.1.21.3 TScriptObjectFactories.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptObjectFactories();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

66.1.21.4 TScriptObjectFactories Methods

The methods of the TScriptObjectFactories class are listed here.

Protected Methods

	Name	Description
	GetCount (see page 6705)	This is GetCount, a member of class TScriptObjectFactories.
	GetFactory (see page 6705)	This is GetFactory, a member of class TScriptObjectFactories.
	GetObjectList (see page 6706)	This is GetObjectList, a member of class TScriptObjectFactories.

Public Methods

	Name	Description
	CreateGlobalObject (see page 6705)	This is CreateGlobalObject, a member of class TScriptObjectFactories.

66.1.21.4.1 TScriptObjectFactories.CreateGlobalObject Method

Pascal

```
function CreateGlobalObject(AVariable: TGlobalScriptVariables; AContext: IScriptContext):
IInterface;
```

C++

```
__fastcall IInterface CreateGlobalObject(TGlobalScriptVariables AVariable, IScriptContext
AContext);
```

Description

This is CreateGlobalObject, a member of class TScriptObjectFactories.

66.1.21.4.2 TScriptObjectFactories.GetCount Method

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

This is GetCount, a member of class TScriptObjectFactories.

66.1.21.4.3 TScriptObjectFactories.GetFactory Method

Pascal

```
function GetFactory(I: Integer): TAbstractScriptObjectFactory; override;
```

C++

```
virtual __fastcall TAbstractScriptObjectFactory GetFactory(int I);
```

Description

This is GetFactory, a member of class TScriptObjectFactories.

66.1.21.4.4 TScriptObjectFactories.GetObjectList Method

Pascal

```
function GetObjectList: TObjectList; override;
```

C++

```
virtual __fastcall TObjectList GetObjectList();
```

Description

This is GetObjectList, a member of class TScriptObjectFactories.

66.1.21.5 TScriptObjectFactories Properties

The properties of the TScriptObjectFactories class are listed here.

Public Properties

	Name	Description
	Count (see page 6706)	This is Count, a member of class TScriptObjectFactories.
	Factories (see page 6706)	This is Factories, a member of class TScriptObjectFactories.

66.1.21.5.1 TScriptObjectFactories.Count Property

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

This is Count, a member of class TScriptObjectFactories.

66.1.21.5.2 TScriptObjectFactories.Factories Property

Pascal

```
property Factories [I: Integer]: TAbstractScriptObjectFactory;
```

C++

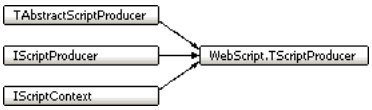
```
__property TAbstractScriptObjectFactory Factories[int I];
```

Description

This is Factories, a member of class TScriptObjectFactories.

66.1.22 TScriptProducer Class

Class Hierarchy



Pascal

```
TScriptProducer = class(TAbstractScriptProducer, IScriptProducer, IScriptContext);
```

C++

```
class TScriptProducer : public TAbstractScriptProducer, public IScriptProducer, public
IScriptContext;
```

File

WebScript

Description

This is class WebScript.TScriptProducer.

66.1.22.1 TScriptProducer Members

The following tables list the members exposed by TScriptProducer.

Protected Fields

	Name	Description
	FErrors (see page 6709)	This is FErrors, a member of class TScriptProducer.
	FScriptEngine (see page 6709)	This is FScriptEngine, a member of class TScriptProducer.
	FScriptFile (see page 6709)	This is FScriptFile, a member of class TScriptProducer.

Protected Methods

	Name	Description
	Evaluate (see page 6710)	This is Evaluate, a member of class TScriptProducer.
A	EvaluateExpression (see page 6710)	This is EvaluateExpression, a member of class TScriptProducer.
	GetContent (see page 6710)	This is GetContent, a member of class TScriptProducer.
	GetErrors (see page 6711)	This is GetErrors, a member of class TScriptProducer.
	GetHTMLBlock (see page 6711)	This is GetHTMLBlock, a member of class TScriptProducer.
	GetHTMLBlockCount (see page 6711)	This is GetHTMLBlockCount, a member of class TScriptProducer.
	GetWebModuleContext (see page 6711)	This is GetWebModuleContext, a member of class TScriptProducer.
	HandleParserError (see page 6712)	This is HandleParserError, a member of class TScriptProducer.
	HandleScriptError (see page 6712)	This is HandleScriptError, a member of class TScriptProducer.
A	ImplHandleScriptError (see page 6712)	This is ImplHandleScriptError, a member of class TScriptProducer.
	ParseStream (see page 6712)	This is ParseStream, a member of class TScriptProducer.
	ParseString (see page 6712)	This is ParseString, a member of class TScriptProducer.
	ReplaceTags (see page 6713)	This is ReplaceTags, a member of class TScriptProducer.
	SetContent (see page 6713)	This is SetContent, a member of class TScriptProducer.
	Write (see page 6713)	This is the overview for the Write method overload.
	WriteItem (see page 6714)	This is WriteItem, a member of class TScriptProducer.

Protected Properties

	Name	Description
	Component (see page 6714)	This is Component, a member of class TScriptProducer.
	Content (see page 6715)	This is Content, a member of class TScriptProducer.
	Designing (see page 6715)	This is Designing, a member of class TScriptProducer.
	Errors (see page 6715)	This is Errors, a member of class TScriptProducer.
	Script (see page 6715)	This is Script, a member of class TScriptProducer.

Public Methods

	Name	Description
	Create (see page 6708)	This is Create, a member of class TScriptProducer.
	Destroy (see page 6708)	Disposes of an object instance.

66.1.22.2 TScriptProducer.Create Constructor

Pascal

```
constructor Create(AWebModuleContext: TWebModuleContext; AStripParamQuotes: Boolean;
AHandleTag: THandleTagProc; const AScriptEngine: string; ALocateFileService:
ILocateFileService); override;
```

C++

```
virtual __fastcall TScriptProducer(TWebModuleContext AWebModuleContext, Boolean
AStripParamQuotes, THandleTagProc AHandleTag, const UnicodeString AScriptEngine,
ILocateFileService ALocateFileService);
```

Description

This is Create, a member of class TScriptProducer.

66.1.22.3 TScriptProducer.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TScriptProducer();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the

object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

66.1.22.4 TScriptProducer Fields

The fields of the TScriptProducer class are listed here.

Protected Fields

	Name	Description
	FErrors (see page 6709)	This is FErrors, a member of class TScriptProducer.
	FScriptEngine (see page 6709)	This is FScriptEngine, a member of class TScriptProducer.
	FScriptFile (see page 6709)	This is FScriptFile, a member of class TScriptProducer.

66.1.22.4.1 TScriptProducer.FErrors Field

Pascal

```
FErrors: TAbstractScriptErrors;
```

C++

```
TAbstractScriptErrors FErrors;
```

Description

This is FErrors, a member of class TScriptProducer.

66.1.22.4.2 TScriptProducer.FScriptEngine Field

Pascal

```
FScriptEngine: string;
```

C++

```
UnicodeString FScriptEngine;
```

Description

This is FScriptEngine, a member of class TScriptProducer.

66.1.22.4.3 TScriptProducer.FScriptFile Field

Pascal

```
FScriptFile: TScriptFile;
```

C++

```
TScriptFile FScriptFile;
```

Description

This is FScriptFile, a member of class TScriptProducer.

66.1.22.5 TScriptProducer Methods

The methods of the TScriptProducer class are listed here.

Protected Methods

	Name	Description
	Evaluate (see page 6710)	This is Evaluate, a member of class TScriptProducer.
A	EvaluateExpression (see page 6710)	This is EvaluateExpression, a member of class TScriptProducer.
	GetContent (see page 6710)	This is GetContent, a member of class TScriptProducer.
	GetErrors (see page 6711)	This is GetErrors, a member of class TScriptProducer.
	GetHTMLBlock (see page 6711)	This is GetHTMLBlock, a member of class TScriptProducer.
	GetHTMLBlockCount (see page 6711)	This is GetHTMLBlockCount, a member of class TScriptProducer.
	GetWebModuleContext (see page 6711)	This is GetWebModuleContext, a member of class TScriptProducer.
	HandleParserError (see page 6712)	This is HandleParserError, a member of class TScriptProducer.
	HandleScriptError (see page 6712)	This is HandleScriptError, a member of class TScriptProducer.
A	ImplHandleScriptError (see page 6712)	This is ImplHandleScriptError, a member of class TScriptProducer.
	ParseStream (see page 6712)	This is ParseStream, a member of class TScriptProducer.
	ParseString (see page 6712)	This is ParseString, a member of class TScriptProducer.
	ReplaceTags (see page 6713)	This is ReplaceTags, a member of class TScriptProducer.
	SetContent (see page 6713)	This is SetContent, a member of class TScriptProducer.
	Write (see page 6713)	This is the overview for the Write method overload.
	WriteItem (see page 6714)	This is WriteItem, a member of class TScriptProducer.

66.1.22.5.1 TScriptProducer.Evaluate Method

Pascal

```
function Evaluate: string;
```

C++

```
__fastcall UnicodeString Evaluate();
```

Description

This is Evaluate, a member of class TScriptProducer.

66.1.22.5.2 TScriptProducer.EvaluateExpression Method

Pascal

```
function EvaluateExpression(AScriptFile: TScriptFile): string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString EvaluateExpression(TScriptFile AScriptFile) = 0;
```

Description

This is EvaluateExpression, a member of class TScriptProducer.

66.1.22.5.3 TScriptProducer.GetContent Method

Pascal

```
function GetContent: string;
```

C++

```
__fastcall UnicodeString GetContent();
```

Description

This is GetContent, a member of class TScriptProducer.

66.1.22.5.4 TScriptProducer.GetErrors Method

Pascal

```
function GetErrors: TAbstractScriptErrors;
```

C++

```
__fastcall TAbstractScriptErrors GetErrors();
```

Description

This is GetErrors, a member of class TScriptProducer.

66.1.22.5.5 TScriptProducer.GetHTMLBlock Method

Pascal

```
function GetHTMLBlock(I: Integer): string;
```

C++

```
__fastcall UnicodeString GetHTMLBlock(int I);
```

Description

This is GetHTMLBlock, a member of class TScriptProducer.

66.1.22.5.6 TScriptProducer.GetHTMLBlockCount Method

Pascal

```
function GetHTMLBlockCount: Integer;
```

C++

```
__fastcall int GetHTMLBlockCount();
```

Description

This is GetHTMLBlockCount, a member of class TScriptProducer.

66.1.22.5.7 TScriptProducer.GetWebModuleContext Method

Pascal

```
function GetWebModuleContext: TWebModuleContext;
```

C++

```
__fastcall TWebModuleContext GetWebModuleContext();
```

Description

This is GetWebModuleContext, a member of class TScriptProducer.

66.1.22.5.8 TScriptProducer.HandleParserError Method

Pascal

```
procedure HandleParserError(E: Exception);
```

C++

```
__fastcall HandleParserError(Exception E);
```

Description

This is HandleParserError, a member of class TScriptProducer.

66.1.22.5.9 TScriptProducer.HandleScriptError Method

Pascal

```
function HandleScriptError(const ScriptError: IUnknown): HRESULT;
```

C++

```
__fastcall HRESULT HandleScriptError(const IUnknown ScriptError);
```

Description

This is HandleScriptError, a member of class TScriptProducer.

66.1.22.5.10 TScriptProducer.ImplHandleScriptError Method

Pascal

```
function ImplHandleScriptError(const ScriptError: IUnknown): HRESULT; virtual; abstract;
```

C++

```
virtual __fastcall HRESULT ImplHandleScriptError(const IUnknown ScriptError) = 0;
```

Description

This is ImplHandleScriptError, a member of class TScriptProducer.

66.1.22.5.11 TScriptProducer.ParseStream Method

Pascal

```
procedure ParseStream(Stream: TStream; Owned: Boolean);
```

C++

```
__fastcall ParseStream(TStream Stream, Boolean Owned);
```

Description

This is ParseStream, a member of class TScriptProducer.

66.1.22.5.12 TScriptProducer.ParseString Method

Pascal

```
procedure ParseString(const S: string);
```

C++

```
__fastcall ParseString(const UnicodeString S);
```

Description

This is ParseString, a member of class TScriptProducer.

66.1.22.5.13 TScriptProducer.ReplaceTags Method

Pascal

```
function ReplaceTags(const AValue: string): string;
```

C++

```
__fastcall UnicodeString ReplaceTags(const UnicodeString AValue);
```

Description

This is ReplaceTags, a member of class TScriptProducer.

66.1.22.5.14 TScriptProducer.SetContent Method

Pascal

```
procedure SetContent(const Value: string);
```

C++

```
__fastcall SetContent(const UnicodeString Value);
```

Description

This is SetContent, a member of class TScriptProducer.

66.1.22.5.15 Write Method

This is the overview for the Write method overload.

Overload List

	Name	Description
	TScriptProducer.Write (PAnsiChar) (see page 6713)	This is Write, a member of class TScriptProducer.
	TScriptProducer.Write (PWideChar, Integer) (see page 6714)	This is Write, a member of class TScriptProducer.
	TScriptProducer.Write (string) (see page 6714)	This is Write, a member of class TScriptProducer.

66.1.22.5.15.1 TScriptProducer.Write Method (PAnsiChar)

Pascal

```
procedure Write(Value: PAnsiChar); overload;
```

C++

```
__fastcall Write(PAnsiChar Value);
```

Description

This is Write, a member of class TScriptProducer.

66.1.22.5.15.2 TScriptProducer.Write Method (PWideChar, Integer)

Pascal

```
procedure Write(const AValue: PWideChar; ALength: Integer); overload;
```

C++

```
__fastcall Write(const PWideChar AValue, int ALength);
```

Description

This is Write, a member of class TScriptProducer.

66.1.22.5.15.3 TScriptProducer.Write Method (string)

Pascal

```
procedure Write(const Value: string); overload;
```

C++

```
__fastcall Write(const UnicodeString Value);
```

Description

This is Write, a member of class TScriptProducer.

66.1.22.5.16 TScriptProducer.WriteItem Method

Pascal

```
procedure WriteItem(Index: Integer);
```

C++

```
__fastcall WriteItem(int Index);
```

Description

This is WriteItem, a member of class TScriptProducer.

66.1.22.6 TScriptProducer Properties

The properties of the TScriptProducer class are listed here.

Protected Properties

	Name	Description
	Component (see page 6714)	This is Component, a member of class TScriptProducer.
	Content (see page 6715)	This is Content, a member of class TScriptProducer.
	Designing (see page 6715)	This is Designing, a member of class TScriptProducer.
	Errors (see page 6715)	This is Errors, a member of class TScriptProducer.
	Script (see page 6715)	This is Script, a member of class TScriptProducer.

66.1.22.6.1 TScriptProducer.Component Property

Pascal

```
property Component: TComponent;
```

C++

```
__property TComponent * Component;
```

Description

This is Component, a member of class TScriptProducer.

66.1.22.6.2 TScriptProducer.Content Property

Pascal

```
property Content: string;
```

C++

```
__property UnicodeString Content;
```

Description

This is Content, a member of class TScriptProducer.

66.1.22.6.3 TScriptProducer.Designing Property

Pascal

```
property Designing: Boolean;
```

C++

```
__property Boolean Designing;
```

Description

This is Designing, a member of class TScriptProducer.

66.1.22.6.4 TScriptProducer.Errors Property

Pascal

```
property Errors: TAbstractScriptErrors;
```

C++

```
__property TAbstractScriptErrors Errors;
```

Description

This is Errors, a member of class TScriptProducer.

66.1.22.6.5 TScriptProducer.Script Property

Pascal

```
property Script: string;
```

C++

```
__property UnicodeString Script;
```

Description

This is Script, a member of class TScriptProducer.

66.1.23 TVariableLookup Class

Class Hierarchy

Pascal

```
TVariableLookup = class(TObject);
```

C++

```
class TVariableLookup : public TObject;
```

File

WebScript

Description

This is class WebScript.TVariableLookup.

66.1.23.1 TVariableLookup Members

The following tables list the members exposed by TVariableLookup.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6717)	This is Create, a member of class TVariableLookup.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindGlobalVariable (see page 6717)	This is FindGlobalVariable, a member of class TVariableLookup.
	FindGlobalVariableContainer (see page 6718)	This is FindGlobalVariableContainer, a member of class TVariableLookup.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.

	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	LookupObject (see page 6718)	This is LookupObject, a member of class TVariableLookup.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Ident (see page 6718)	This is Ident, a member of class TVariableLookup.
R	Obj (see page 6718)	This is Obj, a member of class TVariableLookup.

66.1.23.2 TVariableLookup.Create Constructor

Pascal

```
constructor Create(const AIdent: string; AWebModuleContext: TWebModuleContext);
```

C++

```
__fastcall TVariableLookup(const UnicodeString AIdent, TWebModuleContext AWebModuleContext);
```

Description

This is Create, a member of class TVariableLookup.

66.1.23.3 TVariableLookup Methods

The methods of the TVariableLookup class are listed here.

Public Methods

	Name	Description
	FindGlobalVariable (see page 6717)	This is FindGlobalVariable, a member of class TVariableLookup.
	FindGlobalVariableContainer (see page 6718)	This is FindGlobalVariableContainer, a member of class TVariableLookup.
	LookupObject (see page 6718)	This is LookupObject, a member of class TVariableLookup.

66.1.23.3.1 TVariableLookup.FindGlobalVariable Method

Pascal

```
function FindGlobalVariable(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindGlobalVariable(const UnicodeString AName);
```

Description

This is FindGlobalVariable, a member of class TVariableLookup.

66.1.23.3.2 TVariableLookup.FindGlobalVariableContainer Method

Pascal

```
function FindGlobalVariableContainer(const AName: string): TComponent;
```

C++

```
__fastcall TComponent * FindGlobalVariableContainer(const UnicodeString AName);
```

Description

This is FindGlobalVariableContainer, a member of class TVariableLookup.

66.1.23.3.3 TVariableLookup.LookupObject Method

Pascal

```
function LookupObject: Boolean;
```

C++

```
__fastcall Boolean LookupObject();
```

Description

This is LookupObject, a member of class TVariableLookup.

66.1.23.4 TVariableLookup Properties

The properties of the TVariableLookup class are listed here.

Public Properties

	Name	Description
	Ident (see page 6718)	This is Ident, a member of class TVariableLookup.
R	Obj (see page 6718)	This is Obj, a member of class TVariableLookup.

66.1.23.4.1 TVariableLookup.Ident Property

Pascal

```
property Ident: string;
```

C++

```
__property UnicodeString Ident;
```

Description

This is Ident, a member of class TVariableLookup.

66.1.23.4.2 TVariableLookup.Obj Property

Pascal

```
property Obj: TObject;
```

C++

```
__property TObject * Obj;
```

Description

This is Obj, a member of class TVariableLookup.

66.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TGlobalScriptVariables (see page 6720)	This is record WebScript.TGlobalScriptVariables.
	TIncludeType (see page 6721)	This is record WebScript.TIncludeType.

Records

	Name	Description
	TLineNumbers (see page 6721)	This is record WebScript.TLineNumbers.

66.2.1 WebScript.TGlobalScriptVariables Enumeration

Pascal

```
TGlobalScriptVariables = (
  svPage,
  svPages,
  svModules,
  svApplication,
  svSession,
  svEndUser,
  svResponse,
  svRequest,
  svHTMLItems,
  svProducer
);
```

C++

```
enum TGlobalScriptVariables {
  svPage,
  svPages,
  svModules,
  svApplication,
  svSession,
  svEndUser,
  svResponse,
  svRequest,
  svHTMLItems,
  svProducer
};
```

File

WebScript

Description

This is record WebScript.TGlobalScriptVariables.

66.2.2 WebScript.TIncludeType Enumeration

Pascal

```
TIncludeType = (
  incFile,
  incPage
);
```

C++

```
enum TIncludeType {
  incFile,
  incPage
};
```

File

WebScript

Description

This is record WebScript.TIncludeType.

66.2.3 WebScript.TLineNumbers Record

Pascal

```
TLineNumbers = record
  SourceStartLine: Integer;
  SourceLineCount: Integer;
  DestStartLine: Integer;
  DestLineCount: Integer;
  ScriptIncludeFile: TScriptIncludeFile;
end;
```

C++

```
struct TLineNumbers {
  int SourceStartLine;
  int SourceLineCount;
  int DestStartLine;
  int DestLineCount;
  TScriptIncludeFile ScriptIncludeFile;
};
```

File

WebScript

Description

This is record WebScript.TLineNumbers.

66.3 Types

The following table lists types in this documentation.

Types

Name	Description
CreateComponentStubFactory (see page 6722)	This is type WebScript.CreateComponentStubFactory.
PLineNumbers (see page 6722)	This is type WebScript.PLineNumbers.

66.3.1 WebScript.CreateComponentStubFactory Type

Pascal

```
CreateComponentStubFactory = function (AComponent: TObject; AComponentClass: TClass):
IInterface;
```

C++

```
(AComponent: TObject; AComponentClass: TClass): IInterface ( CreateComponentStubFactory)();
```

File

WebScript

Description

This is type WebScript.CreateComponentStubFactory.

66.3.2 WebScript.PLineNumbers Type

Pascal

```
PLineNumbers = ^TLineNumbers;
```

C++

```
^TLineNumbers PLineNumbers;
```

File

WebScript

Description

This is type WebScript.PLineNumbers.

66.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
GlobalScriptVariableNames (see page 6723)	This is constant WebScript.GlobalScriptVariableNames.

66.4.1 WebScript.GlobalScriptVariableNames Constant

Pascal

```
GlobalScriptVariableNames: array[TGlobalScriptVariables] of PChar = ('Page', 'Pages',
'Modules', 'Application', 'Session', 'EndUser', 'Response', 'Request', 'HTML_', 'Producer');
```

C++

```
array[TGlobalScriptVariables] of PChar GlobalScriptVariableNames = ('Page', 'Pages',
'Modules', 'Application', 'Session', 'EndUser', 'Response', 'Request', 'HTML_', 'Producer');
```

File

WebScript

Description

This is constant WebScript.GlobalScriptVariableNames.

67 WebServExp Namespace

This is namespace WebServExp.

Topics

Name	Description
Classes (see page 6727)	The following table lists classes in this documentation.
Interfaces (see page 6735)	The following table lists interfaces in this documentation.
Functions (see page 6738)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6740)	The following table lists structs, records, enums in this documentation.
Types (see page 6742)	The following table lists types in this documentation.

Classes

	Name	Description
	TWebServExp (see page 6727)	This is class WebServExp.TWebServExp.

Functions

	Name	Description
	GetAliasBaseTypeInfo (see page 6738)	This is function WebServExp.GetAliasBaseTypeInfo.
	GetBindingType (see page 6738)	This is function WebServExp.GetBindingType.
	IsBaseClassTypeInfo (see page 6739)	This is function WebServExp.IsBaseClassTypeInfo.

Interfaces

	Name	Description
	IWebServExp (see page 6735)	IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.
	IWebServExpAccess (see page 6737)	This is class WebServExp.IWebServExpAccess.

Structs, Records, Enums

	Name	Description
	ArgumentType (see page 6740)	This is record WebServExp.ArgumentType.
	MessageType (see page 6740)	This is record WebServExp.MessageType.
	TSchemaType (see page 6741)	This is record WebServExp.TSchemaType.

Types

Name	Description
TAfterPublishingWSDLEvent (see page 6742)	TAfterPublishingWSDLEvent is the type of the OnAfterPublishingWSDL event of TWSDLHTMLPublish.

TBeforePublishingTypesEvent (see page 6742)	TBeforePublishingTypesEvent is the type of the OnBeforePublishingTypes event of TWSDLHTMLPublish.
TPublishingTypeEvent (see page 6743)	TPublishingTypeEvent is the type of the OnPublishingType event of TWSDLHTMLPublish.
TSchemaTypeArray (see page 6744)	This is type WebServExp.TSchemaTypeArray.

67.1 Classes

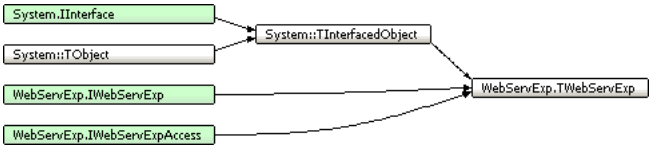
The following table lists classes in this documentation.

Classes

	Name	Description
	TWebServExp (see page 6727)	This is class WebServExp.TWebServExp.

67.1.1 TWebServExp Class

Class Hierarchy



Pascal

```
TWebServExp = class(TInterfacedObject, IWebServExp, IWebServExpAccess);
```

C++

```
class TWebServExp : public TInterfacedObject, public IWebServExp, public IWebServExpAccess;
```

File

WebServExp

Description

This is class WebServExp.TWebServExp.

67.1.1.1 TWebServExp Members

The following tables list the members exposed by TWebServExp.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6729)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6729)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindOrAddSchema (see page 6736)	Ensures that a schema is eventually generated for a specified type.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GenerateXMLSchema (see page 6730)	Adds schema information to represent a specified type.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetWebServExp (see page 6731)	This is GetWebServExp, a member of class TWebServExp.
	GetWSDLForInterface (see page 6731)	This is GetWSDLForInterface, a member of class TWebServExp.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	ImportLocations (see page 6732)	This is ImportLocations, a member of class TWebServExp.
	ImportNames (see page 6732)	This is ImportNames, a member of class TWebServExp.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	TargetNameSpace (see page 6732)	This is TargetNameSpace, a member of class TWebServExp.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 6733)	This is OnAfterPublishingWSDL, a member of class TWebServExp.
	OnBeforePublishingTypes (see page 6733)	This is OnBeforePublishingTypes, a member of class TWebServExp.
	OnPublishingType (see page 6733)	This is OnPublishingType, a member of class TWebServExp.

Published Properties

	Name	Description
	ArrayAsComplexContent (see page 6731)	This is ArrayAsComplexContent, a member of class TWebServExp.
	BindingType (see page 6732)	This is BindingType, a member of class TWebServExp.
	ServiceAddress (see page 6732)	This is ServiceAddress, a member of class TWebServExp.
	WSDLElements (see page 6733)	This is WSDLElements, a member of class TWebServExp.

67.1.1.2 TWebServExp.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebServExp();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

67.1.1.3 TWebServExp.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebServExp();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and

call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

67.1.1.4 TWebServExp Methods

The methods of the TWebServExp class are listed here.

Public Methods

	Name	Description
	GenerateXMLSchema (see page 6730)	Adds schema information to represent a specified type.
	GetWebServExp (see page 6731)	This is GetWebServExp, a member of class TWebServExp.
	GetWSDLForInterface (see page 6731)	This is GetWSDLForInterface, a member of class TWebServExp.

67.1.1.4.1 TWebServExp.GenerateXMLSchema Method

Adds schema information to represent a specified type.

Pascal

```
procedure GenerateXMLSchema(SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo; const
ParentInfo: PTypeinfo; Namespace: InvString);
```

C++

```
__fastcall GenerateXMLSchema(IXMLSchemaDef SchemaDef, const PTypeinfo ATypeInfo, const
PTypeinfo ParentInfo, InvString Namespace);
```

Description

GenerateXMLSchema adds information to an emerging schema to represent information about a specified type.

SchemaDef is the interface for the emerging schema. GenerateXMLSchema uses the properties and methods of this interface to add schema information for the specified type.

ATypeInfo is the runtime type information (RTT) for the type to add.

ParentInfo is the RTTI for the type from which the type specified by ATypeInfo is derived. If ATypeInfo does not describe a descendant type, ParentInfo is nil (Delphi) or NULL (C++).

Namespace is the namespace URI in which the schema definition is defined.

See Also

FindOrAddSchema (see page 6736)

67.1.1.4.2 TWebServExp.GetWebServExp Method

Pascal

```
function GetWebServExp: TWebServExp;
```

C++

```
__fastcall TWebServExp GetWebServExp();
```

Description

This is GetWebServExp, a member of class TWebServExp.

67.1.1.4.3 TWebServExp.GetWSDLForInterface Method

Pascal

```
procedure GetWSDLForInterface(const IntfTypeInfo: Pointer; WSDLDoc: IWSDLDocument; PortNames:
array of InvString; Locations: array of InvString);
```

C++

```
__fastcall GetWSDLForInterface(const void * IntfTypeInfo, IWSDLDocument WSDLDoc, array of
InvString PortNames, array of InvString Locations);
```

Description

This is GetWSDLForInterface, a member of class TWebServExp.

67.1.1.5 TWebServExp Properties

The properties of the TWebServExp class are listed here.

Public Properties

	Name	Description
	ImportLocations (see page 6732)	This is ImportLocations, a member of class TWebServExp.
	ImportNames (see page 6732)	This is ImportNames, a member of class TWebServExp.
	TargetNameSpace (see page 6732)	This is TargetNameSpace, a member of class TWebServExp.

Published Properties

	Name	Description
	ArrayAsComplexContent (see page 6731)	This is ArrayAsComplexContent, a member of class TWebServExp.
	BindingType (see page 6732)	This is BindingType, a member of class TWebServExp.
	ServiceAddress (see page 6732)	This is ServiceAddress, a member of class TWebServExp.
	WSDLElements (see page 6733)	This is WSDLElements, a member of class TWebServExp.

67.1.1.5.1 TWebServExp.ArrayAsComplexContent Property

Pascal

```
property ArrayAsComplexContent: Boolean;
```

C++

```
__property Boolean ArrayAsComplexContent;
```

Description

This is ArrayAsComplexContent, a member of class TWebServExp.

67.1.1.5.2 TWebServExp.BindingType Property

Pascal

```
property BindingType: TWebServiceBindingType;
```

C++

```
__property TWebServiceBindingType BindingType;
```

Description

This is BindingType, a member of class TWebServExp.

67.1.1.5.3 TWebServExp.ImportLocations Property

Pascal

```
property ImportLocations [const Index: Integer]: InvString;
```

C++

```
__property InvString ImportLocations[int const Index];
```

Description

This is ImportLocations, a member of class TWebServExp.

67.1.1.5.4 TWebServExp.ImportNames Property

Pascal

```
property ImportNames [const Index: Integer]: InvString;
```

C++

```
__property InvString ImportNames[int const Index];
```

Description

This is ImportNames, a member of class TWebServExp.

67.1.1.5.5 TWebServExp.ServiceAddress Property

Pascal

```
property ServiceAddress: InvString;
```

C++

```
__property InvString ServiceAddress;
```

Description

This is ServiceAddress, a member of class TWebServExp.

67.1.1.5.6 TWebServExp.TargetNameSpace Property

Pascal

```
property TargetNameSpace: InvString;
```

C++

```
__property InvString TargetNameSpace;
```

Description

This is TargetNameSpace, a member of class TWebServExp.

67.1.1.5.7 TWebServExp.WSDLElements Property

Pascal

```
property WSDLElements: TWSDLElements;
```

C++

```
__property TWSDLElements WSDLElements;
```

Description

This is WSDLElements, a member of class TWebServExp.

67.1.1.6 TWebServExp Events

The events of the TWebServExp class are listed here.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 6733)	This is OnAfterPublishingWSDL, a member of class TWebServExp.
	OnBeforePublishingTypes (see page 6733)	This is OnBeforePublishingTypes, a member of class TWebServExp.
	OnPublishingType (see page 6733)	This is OnPublishingType, a member of class TWebServExp.

67.1.1.6.1 TWebServExp.OnAfterPublishingWSDL Event

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

This is OnAfterPublishingWSDL, a member of class TWebServExp.

67.1.1.6.2 TWebServExp.OnBeforePublishingTypes Event

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

This is OnBeforePublishingTypes, a member of class TWebServExp.

67.1.1.6.3 TWebServExp.OnPublishingType Event

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

This is OnPublishingType, a member of class TWebServExp.

67.2 Interfaces

The following table lists interfaces in this documentation.

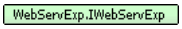
Interfaces

	Name	Description
	IWebServExp (see page 6735)	IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.
	IWebServExpAccess (see page 6737)	This is class WebServExp.IWebServExpAccess.

67.2.1 IWebServExp Interface

IWebServExp is the interface for classes that generate schema definitions for a WSDL publisher.

Class Hierarchy



Pascal

```
IWebServExp = interface;
```

C++

```
__interface IWebServExp;
```

File

WebServExp

Description

IWebServExp is an interface for generating the schema definitions that represent types in a WSDL document.

Note: C++ method declarations that use IWebServExp use the _di_IWebServExp type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWebServExp interface:

Note: typedef System::DelphiInterface (see page 3256)< IWebServExp > _di_IWebServExp;

See Also

TWSDLHTMLPublish (see page 7280)

67.2.1.1 IWebServExp Members

The following tables list the members exposed by IWebServExp.

Public Methods

	Name	Description
	FindOrAddSchema (see page 6736)	Ensures that a schema is eventually generated for a specified type.
	GenerateXMLSchema (see page 6736)	Adds schema information to represent a specified type.

67.2.1.2 IWebServExp Methods

The methods of the IWebServExp class are listed here.

Public Methods

	Name	Description
	FindOrAddSchema (see page 6736)	Ensures that a schema is eventually generated for a specified type.
	GenerateXMLSchema (see page 6736)	Adds schema information to represent a specified type.

67.2.1.2.1 IWebServExp.FindOrAddSchema Method

Ensures that a schema is eventually generated for a specified type.

Pascal

```
function FindOrAddSchema(const ATypeInfo: PTypeinfo; const TnsURI: string): Boolean;
```

C++

```
__fastcall Boolean FindOrAddSchema(const PTypeinfo ATypeInfo, const UnicodeString TnsURI);
```

Description

Call FindOrAddSchema to indicate that a schema should be generated for a given type.

ATypeInfo is the runtime type information (RTTI) for the type.

TnsURI is the namespace URI in which the schema for the type should be defined.

FindOrAddSchema returns true if there is already a schema for the specified type. It returns false if the type has not previously been added.

See Also

GenerateXMLSchema (see page 6736)

67.2.1.2.2 IWebServExp.GenerateXMLSchema Method

Adds schema information to represent a specified type.

Pascal

```
procedure GenerateXMLSchema(SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo; const
ParentInfo: PTypeinfo; Namespace: InvString);
```

C++

```
__fastcall GenerateXMLSchema(IXMLSchemaDef SchemaDef, const PTypeinfo ATypeInfo, const
PTypeinfo ParentInfo, InvString Namespace);
```

Description

GenerateXMLSchema adds information to an emerging schema to represent information about a specified type.

SchemaDef is the interface for the emerging schema. GenerateXMLSchema uses the properties and methods of this interface to add schema information for the specified type.

ATypeInfo is the runtime type information (RTT) for the type to add.

ParentInfo is the RTTI for the type from which the type specified by ATypeInfo is derived. If ATypeInfo does not describe a descendant type, ParentInfo is nil (Delphi) or NULL (C++).

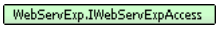
Namespace is the namespace URI in which the schema definition is defined.

See Also

FindOrAddSchema (see page 6736)

67.2.2 IWebServExpAccess Interface

Class Hierarchy



Pascal

```
IWebServExpAccess = interface;
```

C++

```
__interface IWebServExpAccess;
```

File

WebServExp

Description

This is class WebServExp.IWebServExpAccess.

67.2.2.1 IWebServExpAccess Members

The following tables list the members exposed by IWebServExpAccess.

Public Methods

	Name	Description
	GetWebServExp (see page 6737)	This is GetWebServExp, a member of class IWebServExpAccess.

67.2.2.2 IWebServExpAccess Methods

The methods of the IWebServExpAccess class are listed here.

Public Methods

	Name	Description
	GetWebServExp (see page 6737)	This is GetWebServExp, a member of class IWebServExpAccess.

67.2.2.2.1 IWebServExpAccess.GetWebServExp Method

Pascal

```
function GetWebServExp: TWebServExp;
```

C++

```
__fastcall TWebServExp GetWebServExp();
```

Description

This is GetWebServExp, a member of class IWebServExpAccess.

67.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetAliasBaseTypeInfo (see page 6738)	This is function WebServExp.GetAliasBaseTypeInfo.
	GetBindingType (see page 6738)	This is function WebServExp.GetBindingType.
	IsBaseClassTypeInfo (see page 6739)	This is function WebServExp.IsBaseClassTypeInfo.

67.3.1 WebServExp.GetAliasBaseTypeInfo Function

Pascal

```
function GetAliasBaseTypeInfo(const ParamType: TTypeKind): PTypeInfo;
```

C++

```
PTypeInfo GetAliasBaseTypeInfo(const TTypeKind ParamType);
```

File

WebServExp

Description

This is function WebServExp.GetAliasBaseTypeInfo.

67.3.2 WebServExp.GetBindingType Function

Pascal

```
function GetBindingType(const MethEntry: TIntfMethEntry; Input: Boolean):
TWebServiceBindingType; overload;
```

C++

```
TWebServiceBindingType GetBindingType(const TIntfMethEntry MethEntry, Boolean Input);
```

File

WebServExp

Description

This is function WebServExp.GetBindingType.

67.3.3 WebServExp.IsBaseClassTypeInfo Function

Pascal

```
function IsBaseClassTypeInfo(const ATypeInfo: PTypeInfo): Boolean;
```

C++

```
Boolean IsBaseClassTypeInfo(const PTypeInfo ATypeInfo);
```

File

WebServExp

Description

This is function WebServExp.IsBaseClassTypeInfo.

67.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	ArgumentType (see page 6740)	This is record WebServExp.ArgumentType.
	MessageType (see page 6740)	This is record WebServExp.MessageType.

Records

	Name	Description
	TSchemaType (see page 6741)	This is record WebServExp.TSchemaType.

67.4.1 WebServExp.ArgumentType Enumeration

Pascal

```
ArgumentType = (
  argIn,
  argOut,
  argInOut,
  argReturn
);
```

C++

```
enum ArgumentType {
  argIn,
  argOut,
  argInOut,
  argReturn
};
```

File

WebServExp

Description

This is record WebServExp.ArgumentType.

67.4.2 WebServExp.MessageType Enumeration

Pascal

```
MessageType = (
  mtInput,
  mtOutput,
  mtHeaderInput,
  mtHeaderOutput,
  mtFault
);
```

C++

```
enum MessageType {
  mtInput,
  mtOutput,
  mtHeaderInput,
  mtHeaderOutput,
  mtFault
};
```

File

WebServExp

Description

This is record WebServExp.MessageType.

67.4.3 WebServExp.TSchemaType Record

Pascal

```
TSchemaType = record
  TypeName: InvString;
  NameSpace: InvString;
  TypeInfo: PTypeinfo;
  NSPrefix: InvString;
  XSGenerated: Boolean;
end;
```

C++

```
struct TSchemaType {
  InvString TypeName;
  InvString NameSpace;
  PTypeinfo TypeInfo;
  InvString NSPrefix;
  Boolean XSGenerated;
};
```

File

WebServExp

Description

This is record WebServExp.TSchemaType.

67.5 Types

The following table lists types in this documentation.

Types

Name	Description
TAfterPublishingWSDLEvent (see page 6742)	TAfterPublishingWSDLEvent is the type of the OnAfterPublishingWSDL event of TWSDLHTMLPublish.
TBeforePublishingTypesEvent (see page 6742)	TBeforePublishingTypesEvent is the type of the OnBeforePublishingTypes event of TWSDLHTMLPublish.
TPublishingTypeEvent (see page 6743)	TPublishingTypeEvent is the type of the OnPublishingType event of TWSDLHTMLPublish.
TSchemaTypeArray (see page 6744)	This is type WebServExp.TSchemaTypeArray.

67.5.1 WebServExp.TAfterPublishingWSDLEvent Type

TAfterPublishingWSDLEvent is the type of the OnAfterPublishingWSDL event of TWSDLHTMLPublish.

Pascal

```
TAfterPublishingWSDLEvent = procedure (const WSDLDoc: IWSDLDocument) of object;
```

C++

```
(const WSDLDoc: IWSDLDocument) ( TAfterPublishingWSDLEvent)();
```

File

WebServExp

Description

TAfterPublishingWSDLEvent is the type of the OnAfterPublishingWSDL event handler, which is called after the publisher generates the WSDL describing a Web Service application.

WSDLDoc is an interface to the generated WSDL document. You can use this interface to make any final modifications to the generated document.

See Also

Procedural types

Creating events

67.5.2 WebServExp.TBeforePublishingTypesEvent Type

TBeforePublishingTypesEvent is the type of the OnBeforePublishingTypes event of TWSDLHTMLPublish.

Pascal

```
TBeforePublishingTypesEvent = procedure (const WebServ: IWebServExp) of object;
```

C++

```
(const WebServ: IWebServExp) ( TBeforePublishingTypesEvent)();
```

File

WebServExp

Description

TBeforePublishingTypesEvent is the type of the OnBeforePublishingTypes event handler, which is called before adding information about types to the WSDL document.

WebServ is an interface for adding type information to the WSDL document.

See Also

Procedural types

Creating events

67.5.3 WebServExp.TPublishingTypeEvent Type

TPublishingTypeEvent is the type of the OnPublishingType event of TWSDLHTMLPublish.

Pascal

```
TPublishingTypeEvent = procedure (const WebServ: IWebServExp; const SchemaDef: IXMLSchemaDef;
const ATypeInfo: PTypeinfo; Namespace: InvString) of object;
```

C++

```
(const WebServ: IWebServExp; const SchemaDef: IXMLSchemaDef; const ATypeInfo: PTypeinfo;
Namespace: InvString) ( TPublishingTypeEvent)();
```

File

WebServExp

Description

TPublishingTypeEvent is the type of the OnPublishingType event handler, which is called before adding information about a given type to a WSDL document.

WebServ is an interface for adding type information to the WSDL document.

SchemaDef is the interface that represents the current schema information in the WSDL document. Its properties and methods can be used to add additional schema information about the current type.

ATypeInfo is the runtime type information (RTTI) for the type to add.

Namespace is the namespace URI in which the generated schema definitions are defined.

See Also

Procedural types

Creating events

67.5.4 WebServExp.TSchemaTypeArray Type

Pascal

```
TSchemaTypeArray = array of TSchemaType;
```

C++

```
array of TSchemaType TSchemaTypeArray;
```

File

WebServExp

Description

This is type WebServExp.TSchemaTypeArray.

68 WebSess Namespace

This is namespace WebSess.

Topics

Name	Description
Classes (see page 6746)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6773)	The following table lists structs, records, enums in this documentation.
Types (see page 6774)	The following table lists types in this documentation.

Classes

	Name	Description
	TCustomSessionsService (see page 6746)	TCustomSessionsService stores end user data that is needed for a short period of time.
	TSessionsService (see page 6756)	TSessionsService stores end user data that is needed for a short period of time.
	TWebSessionIDsImpl (see page 6759)	This is class WebSess.TWebSessionIDsImpl.
	TWebSessionImpl (see page 6763)	This is class WebSess.TWebSessionImpl.

Structs, Records, Enums

	Name	Description
	TEndSessionReason (see page 6773)	TEndSessionReason indicates the reason a session ended.

Types

Name	Description
TEndSessionEvent (see page 6774)	TEndSessionEvent is the type for event handlers that are called when a session terminates.
TStartSessionEvent (see page 6774)	TStartSessionEvent is the type for event handlers that are called when a session starts up.

68.1 Classes

The following table lists classes in this documentation.

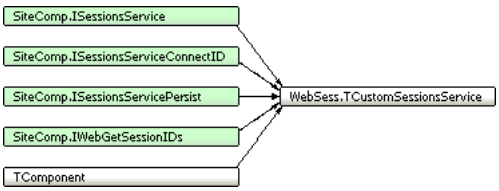
Classes

	Name	Description
	TCustomSessionsService (see page 6746)	TCustomSessionsService stores end user data that is needed for a short period of time.
	TSessionsService (see page 6756)	TSessionsService stores end user data that is needed for a short period of time.
	TWebSessionIDsImpl (see page 6759)	This is class WebSess.TWebSessionIDsImpl.
	TWebSessionImpl (see page 6763)	This is class WebSess.TWebSessionImpl.

68.1.1 TCustomSessionsService Class

TCustomSessionsService stores end user data that is needed for a short period of time.

Class Hierarchy



Pascal

```
TCustomSessionsService = class(TComponent, ISessionsService, ISessionsServiceConnectID,
IWebGetSessionIDs, ISessionsServicePersist);
```

C++

```
class TCustomSessionsService : public TComponent, public ISessionsService, public
ISessionsServiceConnectID, public IWebGetSessionIDs, public ISessionsServicePersist;
```

File

WebSess

Description

TCustomSessionsService is the base class for classes that manage the sessions in a WebSnap application. When a WebSnap application contains a TCustomSessionsService descendant, then each HTTP request is associated with a session. TCustomSessionsService can manage multiple sessions on behalf of different users, each on a separate thread of execution. Only one session is available when handling a request. This is called the current session. The current session is available as the Session property of the global Web context, which is accessed using the global WebContext function.

TCustomSessionsService stores session IDs and name/value pairs for all sessions in memory. This information must remain in memory between requests. Thus, Web application types that do not remain in memory between requests, such as CGI, do not support TCustomSessionsService.

TCustomSessionsService implements the ISessionsService (see page 1066) interface.

See Also

TSessionsService (see page 6756)

TEndUserSessionAdapter (see page 5792)

TAbstractWebContext (see page 5960)

TWebSession (see page 6251)

WebContext (see page 6003)

ISessionsService (see page 1066)

68.1.1.1 TCustomSessionsService Members

The following tables list the members exposed by TCustomSessionsService.

Protected Methods

	Name	Description
	CheckStatusChange (see page 6748)	Checks whether any sessions have started or terminated.
	Clear (see page 6749)	Deletes all sessions from memory.
	Connect (see page 6749)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 6750)	Obtains the interface for an active Web session, given its Session ID.
	DoEndSession (see page 6750)	Triggers the OnEndSession (see page 6755) event when appropriate.
	DoStartSession (see page 6750)	Triggers the OnStartSession (see page 6755) event when appropriate.
	GetSessionIDs (see page 6751)	Returns a list of the session IDs for all sessions currently in memory.
	LoadFromFile (see page 6751)	Loads the list of sessions from a file.
	LoadFromStream (see page 6751)	Loads the list of sessions from a stream.
	NewSession (see page 6752)	Starts a new session.
	NotifyStatusChange (see page 6752)	Generates an OnEndSession (see page 6755) or OnStartSession (see page 6755) event, as approrpriate.
	SaveToFile (see page 6753)	Saves the list of sessions to a file.
	SaveToStream (see page 6753)	Saves the list of sessions to a stream.

Public Events

	Name	Description
	OnEndSession (see page 6755)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6755)	Occurs when the sessions service detects that a new session has started.

Public Methods

	Name	Description
	Create (see page 6747)	Creates and initializes an instance of TCustomSessionsService (see page 6746).

Public Properties

	Name	Description
	DefaultTimeout (see page 6754)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6754)	Indicates the maximum number of sessions the TCustomSessionsService (see page 6746) can keep in memory.

68.1.1.2 TCustomSessionsService.Create Constructor

Creates and initializes an instance of TCustomSessionsService (see page 6746).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomSessionsService(TComponent * AOwner);
```

Description

Call Create to instantiate a sessions service at runtime. Typically, this is unnecessary, because a WebSnap application needs at most a single sessions service component, and in most cases that is added at design time.

AOwner is the component that becomes the value of the TCustomSessionsService (see page 6746) object's Owner property. This should be the Web application module.

See Also

Owner

68.1.1.3 TCustomSessionsService Methods

The methods of the TCustomSessionsService class are listed here.

Protected Methods

	Name	Description
	CheckStatusChange (see page 6748)	Checks whether any sessions have started or terminated.
	Clear (see page 6749)	Deletes all sessions from memory.
	Connect (see page 6749)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 6750)	Obtains the interface for an active Web session, given its Session ID.
V	DoEndSession (see page 6750)	Triggers the OnEndSession (see page 6755) event when appropriate.
V	DoStartSession (see page 6750)	Triggers the OnStartSession (see page 6755) event when appropriate.
	GetSessionIDs (see page 6751)	Returns a list of the session IDs for all sessions currently in memory.
	LoadFromFile (see page 6751)	Loads the list of sessions from a file.
	LoadFromStream (see page 6751)	Loads the list of sessions from a stream.
	NewSession (see page 6752)	Starts a new session.
	NotifyStatusChange (see page 6752)	Generates an OnEndSession (see page 6755) or OnStartSession (see page 6755) event, as approrpriate.
	SaveToFile (see page 6753)	Saves the list of sessions to a file.
	SaveToStream (see page 6753)	Saves the list of sessions to a stream.

68.1.1.3.1 TCustomSessionsService.CheckStatusChange Method

Checks whether any sessions have started or terminated.

Pascal

```
function CheckStatusChange: Boolean;
```

C++

```
__fastcall Boolean CheckStatusChange();
```

Description

CheckStatusChange is called automatically when the application creates a new Web context or initializes a new Web module. That is, this method is called at least once every time the application receives a new HTTP request message.

If CheckStatusChange detects that a session has started or terminated since the last call, it returns true. Otherwise, it returns false.

See Also

NotifyStatusChange (see page 6752)

TWebContext (see page 5976)

68.1.1.3.2 TCustomSessionsService.Clear Method

Deletes all sessions from memory.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the sessions service of all the sessions it has stored in memory.

Clear does not generate an OnEndSession (see page 6755) event for the sessions it deletes when it frees their memory, it simply removes them from memory. To shut down all sessions in a more orderly fashion, iterate through all the active sessions by calling ConnectActive (see page 6750) for every session ID that the GetSessionIDs (see page 6751) method returns, and terminate each one. Then call NotifyStatusChange (see page 6752) to generate the OnEndSession (see page 6755) events for every terminated session. At that point, calling Clear does not prevent the application from receiving an OnEndSession (see page 6755) event for any session that the sessions service manages.

See Also

NewSession (see page 6752)

GetSessionIDs (see page 6751)

ConnectActive (see page 6750)

NotifyStatusChange (see page 6752)

68.1.1.3.3 TCustomSessionsService.Connect Method

Obtains the interface for a Web session, given its Session ID.

Pascal

```
function Connect(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession Connect(TSessionID AID);
```

Description

Call Connect to access the session that has a specified Session ID. Unlike the ConnectActive (see page 6750) method, Connect may return a session that has expired or been terminated but that has not yet been freed from memory.

AID is the session ID of the session to locate.

Connect returns the IWebSession (see page 1102) interface for the located session. This interface has a single method, Terminate, which lets you terminate the session. If the sessions service does not have a session with the specified session ID, Connect returns nil (Delphi) or NULL (C++).

Note: Calling Connect does not count as accessing a session. That is, if you call Connect for any session, it does not affect the time when that session times out.

See Also

ConnectActive (see page 6750)

GetSessionIDs (see page 6751)

68.1.1.3.4 TCustomSessionsService.ConnectActive Method

Obtains the interface for an active Web session, given its Session ID.

Pascal

```
function ConnectActive(AID: TSessionID): IWebSession;
```

C++

```
__fastcall IWebSession ConnectActive(TSessionID AID);
```

Description

Call ConnectActive to access the session that has a specified Session ID. Unlike the Connect (see page 6749) method, ConnectActive never returns a session that has expired or been terminated.

AID is the session ID of the session to locate.

ConnectActive returns the IWebSession (see page 1102) interface for the located session. This interface has a single method, Terminate, which lets you terminate the session. If the sessions service does not have an active session with the specified session ID, ConnectActive returns nil (Delphi) or NULL (C++).

Note: Calling ConnectActive counts as accessing a session. That is, if you call ConnectActive for any session, the session will not time out until the entire time-out period elapses after the call to ConnectActive.

See Also

Connect (see page 6749)

GetSessionIDs (see page 6751)

68.1.1.3.5 TCustomSessionsService.DoEndSession Method

Triggers the OnEndSession (see page 6755) event when appropriate.

Pascal

```
procedure DoEndSession(ASession: TAbstractWebSession; AReason: TEndSessionReason); virtual;
```

C++

```
virtual __fastcall DoEndSession(TAbstractWebSession ASession, TEndSessionReason AReason);
```

Description

Use DoEndSession to trigger an OnEndSession (see page 6755) event, if it is appropriate to do so.

See Also

OnEndSession (see page 6755)

68.1.1.3.6 TCustomSessionsService.DoStartSession Method

Triggers the OnStartSession (see page 6755) event when appropriate.

Pascal

```
procedure DoStartSession(ASession: TAbstractWebSession); virtual;
```

C++

```
virtual __fastcall DoStartSession(TAbstractWebSession ASession);
```

Description

Use DoStartSession to trigger an OnStartSession (see page 6755) event, if it is appropriate to do so.

See Also

OnStartSession (see page 6755)

68.1.1.3.7 TCustomSessionsService.GetSessionIDs Method

Returns a list of the session IDs for all sessions currently in memory.

Pascal

```
function GetSessionIDs: IWebSessionIDs;
```

C++

```
__fastcall IWebSessionIDs GetSessionIDs();
```

Description

Call GetSessionIDs to obtain a list of the session IDs for all the sessions that the sessions service is storing. GetSessionIDs returns an IWebSessionIDs (see page 1109) interface, which lists the session IDs.

See Also

Connect (see page 6749)

68.1.1.3.8 TCustomSessionsService.LoadFromFile Method

Loads the list of sessions from a file.

Pascal

```
procedure LoadFromFile(const Filename: string);
```

C++

```
__fastcall LoadFromFile(const UnicodeString Filename);
```

Description

Call LoadFromFile to replace the current list of sessions with a set of sessions stored in a file. The file must be previously created by a call to the SaveToFile (see page 6753) method.

Filename is the name of the file that contains the saved list of sessions.

See Also

SaveToFile (see page 6753)

LoadFromStream (see page 6751)

68.1.1.3.9 TCustomSessionsService.LoadFromStream Method

Loads the list of sessions from a stream.

Pascal

```
procedure LoadFromStream(S: TStream);
```

C++

```
__fastcall LoadFromStream(TStream S);
```

Description

Call LoadFromStream to replace the current list of sessions with a set of sessions read from a stream. The sessions must be previously saved by a call to the SaveToStream (see page 6753) method.

S is the stream from which to read the list of sessions.

See Also

SaveToStream (see page 6753)

LoadFromFile (see page 6751)

68.1.1.3.10 TCustomSessionsService.NewSession Method

Starts a new session.

Pascal

```
function NewSession: IWebSession;
```

C++

```
__fastcall IWebSession NewSession();
```

Description

Call NewSession to generate a new session. NewSession starts a new session and returns an interface that can be used to terminate that session.

If the sessions service already has MaxSessions (see page 6754) sessions, NewSession raises an ESessionsException (see page 374) exception.

Note: The OnStartSession (see page 6755) event does not occur immediately when you call NewSession to start a new session. It occurs on the next call to NotifyStatusChange (see page 6752).

Tip: To reconnect to an existing session, call the ConnectActive (see page 6750) method instead.

See Also

MaxSessions (see page 6754)

OnStartSession (see page 6755)

ConnectActive (see page 6750)

NotifyStatusChange (see page 6752)

68.1.1.3.11 TCustomSessionsService.NotifyStatusChange Method

Generates an OnEndSession (see page 6755) or OnStartSession (see page 6755) event, as approrpriate.

Pascal

```
procedure NotifyStatusChange;
```

C++

```
__fastcall NotifyStatusChange();
```

Description

Call NotifyStatusChange to generate the OnStartSession (see page 6755) event for any new sessions that have been created since the last call and to generate the OnEndSession (see page 6755) event for any sessions that terminated since the last call. For any terminated sessions, NotifyStatusChange frees the memory allocated to the session after the OnEndSession (see page 6755) event handler exits.

NotifyStatusChange is called automatically every time the WebSnap application initializes a new Web module.

See Also

OnEndSession (see page 6755)

OnStartSession (see page 6755)

CheckStatusChange (see page 6748)

68.1.1.3.12 TCustomSessionsService.SaveToFile Method

Saves the list of sessions to a file.

Pascal

```
procedure SaveToFile(const Filename: string);
```

C++

```
__fastcall SaveToFile(const UnicodeString Filename);
```

Description

Call SaveToFile to save the current list of sessions to a file. SaveToFile saves all of the sessions, including their session IDs, name/value lists, time-out period, and status information about when they were last accessed.

The file can be read back into a sessions service by calling the LoadFromFile (see page 6751) method.

Filename is the name of the file to which the list of sessions is saved.

See Also

SaveToStream (see page 6753)

LoadFromFile (see page 6751)

68.1.1.3.13 TCustomSessionsService.SaveToStream Method

Saves the list of sessions to a stream.

Pascal

```
procedure SaveToStream(S: TStream);
```

C++

```
__fastcall SaveToStream(TStream S);
```

Description

Call SaveToStream to write the current list of sessions to a stream. SaveToStream saves all of the sessions, including their session IDs, name/value lists, time-out period, and status information about when they were last accessed.

The sessions can be read back into a sessions service by calling the LoadFromStream (see page 6751) method.

S is the stream to which the list of sessions is written.

See Also

LoadFromStream (see page 6751)

SaveToFile (see page 6753)

68.1.1.4 TCustomSessionsService Properties

The properties of the TCustomSessionsService class are listed here.

Public Properties

	Name	Description
	DefaultTimeout (see page 6754)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6754)	Indicates the maximum number of sessions the TCustomSessionsService (see page 6746) can keep in memory.

68.1.1.4.1 TCustomSessionsService.DefaultTimeout Property

Specifies the number of minutes a session can persist without being accessed.

Pascal

```
property DefaultTimeout: Integer;
```

C++

```
__property int DefaultTimeout;
```

Description

Each session that the sessions service manages has a time-out period. When a session is not accessed within that time-out period, the sessions service automatically terminates the session, discarding its session ID and name/value pairs.

Set DefaultTimeout to specify the time-out period, in minutes, for the sessions in the WebSnap application. Individual sessions can override this value by setting their TimeoutMinutes property.

See Also

TimeoutMinutes

68.1.1.4.2 TCustomSessionsService.MaxSessions Property

Indicates the maximum number of sessions the TCustomSessionsService (see page 6746) can keep in memory.

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

Set MaxSessions to limit the number of sessions that the sessions service stores in memory. If MaxSessions is –1, then there is no limit to the number of sessions, and the application can continue creating sessions until it runs out of memory. When MaxSessions is greater than or equal to zero, any attempt to create a new session when the sessions service has MaxSessions already stored in memory causes an ESessionsException (see page 374) exception. The Exception (see page 3949) occurs before the OnStartSession (see page 6755) event.

Note: When the application terminates sessions, it frees up memory for new sessions. Thus, after reaching the limit imposed by MaxSessions, the application can add subsequent sessions after explicitly terminating a session or waiting for one to time out.

See Also

ESessionsException (see page 374)

OnStartSession (see page 6755)

Terminate (see page 6255)

68.1.1.5 TCustomSessionsService Events

The events of the TCustomSessionsService class are listed here.

Public Events

	Name	Description
	OnEndSession (see page 6755)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6755)	Occurs when the sessions service detects that a new session has started.

68.1.1.5.1 TCustomSessionsService.OnEndSession Event

Occurs when the sessions service detects that a session has terminated.

Pascal

```
property OnEndSession: TEndSessionEvent;
```

C++

```
__property TEndSessionEvent OnEndSession;
```

Description

Write (see page 3707) an OnEndSession event handler to take specific action when a session terminates. For example, an OnEndSession event handler could store information from sessions that time out in a more persistent form such as a database.

See Also

OnStartSession (see page 6755)

Terminate (see page 6255)

TimeoutMinutes

68.1.1.5.2 TCustomSessionsService.OnStartSession Event

Occurs when the sessions service detects that a new session has started.

Pascal

```
property OnStartSession: TStartSessionEvent;
```

C++

```
__property TStartSessionEvent OnStartSession;
```

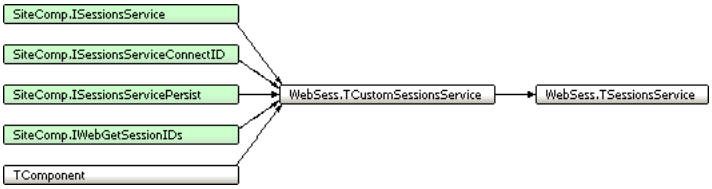
Description

Write (see page 3707) an OnStartSession event handler to take specific action when a session starts. For example, you can use this event to customize the time-out period for new sessions or initialize them by adding name/value pairs.

68.1.2 TSessionsService Class

TSessionsService stores end user data that is needed for a short period of time.

Class Hierarchy



Pascal

```
TSessionsService = class(TCustomSessionsService);
```

C++

```
class TSessionsService : public TCustomSessionsService;
```

File

WebSess

Description

Add TSessionsService to a WebSnap application when you want to store temporary end user data in a WebSnap session. For example, TSessionsService can keep track of all users that are currently logged in and automatically log a user out after a period of inactivity.

When a WebSnap application contains a TSessionsService component, each HTTP request is associated with a session. The WebSnap application can manage multiple sessions on behalf of different users, each on a separate thread of execution. When handling a request, only one session is available, which is called the current session. The current session is available as the Session property of the global Web context, which is accessed using the global WebContext function.

TSessionsService stores session IDs and name/value pairs for all sessions in memory. This information must remain in memory between requests. Web application types that do not remain in memory between requests, such as CGI, do not support TSessionsService. When testing Web App Debugger executables, be sure to start the executables before browsing so that the executable remains running between requests.

Use TEndUserSessionsAdapter and TSessionsAdapter together to keep track of end user name and login state.

See Also

TEndUserSessionAdapter (see page 5792)

TAbstractWebContext (see page 5960)

TWebSession (see page 6251)

WebContext (see page 6003)

Adding Login Support

Using the Sessions Service

68.1.2.1 TSessionsService Members

The following tables list the members exposed by TSessionsService.

Protected Methods

	Name	Description
	CheckStatusChange (see page 6748)	Checks whether any sessions have started or terminated.
	Clear (see page 6749)	Deletes all sessions from memory.
	Connect (see page 6749)	Obtains the interface for a Web session, given its Session ID.
	ConnectActive (see page 6750)	Obtains the interface for an active Web session, given its Session ID.
	DoEndSession (see page 6750)	Triggers the OnEndSession (see page 6755) event when appropriate.
	DoStartSession (see page 6750)	Triggers the OnStartSession (see page 6755) event when appropriate.
	GetSessionIDs (see page 6751)	Returns a list of the session IDs for all sessions currently in memory.
	LoadFromFile (see page 6751)	Loads the list of sessions from a file.
	LoadFromStream (see page 6751)	Loads the list of sessions from a stream.
	NewSession (see page 6752)	Starts a new session.
	NotifyStatusChange (see page 6752)	Generates an OnEndSession (see page 6755) or OnStartSession (see page 6755) event, as apprорpriate.
	SaveToFile (see page 6753)	Saves the list of sessions to a file.
	SaveToStream (see page 6753)	Saves the list of sessions to a stream.

Public Methods

	Name	Description
	CheckStatusChange (see page 1067)	Checks whether any sessions have started or terminated.
	Clear (see page 1067)	Deletes all sessions from memory.
	Connect (see page 1069)	This is Connect, a member of class ISessionsServiceConnectID.
	ConnectActive (see page 1069)	This is ConnectActive, a member of class ISessionsServiceConnectID.
	Create (see page 6747)	Creates and initializes an instance of TCustomSessionsService (see page 6746).
	GetSessionIDs (see page 1097)	This is GetSessionIDs, a member of class IWebGetSessionIDs.
	LoadFromFile (see page 1070)	This is LoadFromFile, a member of class ISessionsServicePersist.
	LoadFromStream (see page 1071)	This is LoadFromStream, a member of class ISessionsServicePersist.
	NewSession (see page 1068)	Starts a new session.
	NotifyStatusChange (see page 1068)	Generates an OnEndSession or OnStartSession event, as apprорpriate.
	SaveToFile (see page 1071)	This is SaveToFile, a member of class ISessionsServicePersist.
	SaveToStream (see page 1071)	This is SaveToStream, a member of class ISessionsServicePersist.

Published Events

	Name	Description
	OnEndSession (see page 6759)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6759)	Occurs when the sessions service detects that a new session has started.

Published Properties

	Name	Description
	DefaultTimeout (see page 6758)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6758)	Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

68.1.2.2 TSessionsService Properties

The properties of the TSessionsService class are listed here.

Published Properties

	Name	Description
	DefaultTimeout (see page 6758)	Specifies the number of minutes a session can persist without being accessed.
	MaxSessions (see page 6758)	Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

68.1.2.2.1 TSessionsService.DefaultTimeout Property

Specifies the number of minutes a session can persist without being accessed.

Pascal

```
property DefaultTimeout: Integer;
```

C++

```
__property int DefaultTimeout;
```

Description

Each session that the sessions service manages has a time-out period. When a session is not accessed within that time-out period, the sessions service automatically terminates the session, discarding its session ID and name/value pairs.

Set DefaultTimeout to specify the time-out period, in minutes, for the sessions in the WebSnap application. Individual sessions can override this value by setting their TimeoutMinutes property.

See Also

TimeoutMinutes

68.1.2.2.2 TSessionsService.MaxSessions Property

Indicates the maximum number of sessions the TCustomSessionsService can keep in memory.

Pascal

```
property MaxSessions: Integer;
```

C++

```
__property int MaxSessions;
```

Description

Set MaxSessions to limit the number of sessions that the sessions service stores in memory. If MaxSessions is –1, then there is no limit to the number of sessions, and the application can continue creating sessions until it runs out of memory. When MaxSessions is greater than or equal to zero, any attempt to create a new session when the sessions service has MaxSessions already stored in memory causes an ESessionsException exception. The Exception occurs before the OnStartSession event.

Note: When the application terminates sessions, it frees up memory for new sessions. Thus, after reaching the limit imposed by MaxSessions, the application can add subsequent sessions after explicitly terminating a session or waiting for one to time out.

See Also

ESessionsException (see page 374)

OnStartSession (see page 6755)

Terminate (see page 6255)

68.1.2.3 TSessionsService Events

The events of the TSessionsService class are listed here.

Published Events

	Name	Description
	OnEndSession (see page 6759)	Occurs when the sessions service detects that a session has terminated.
	OnStartSession (see page 6759)	Occurs when the sessions service detects that a new session has started.

68.1.2.3.1 TSessionsService.OnEndSession Event

Occurs when the sessions service detects that a session has terminated.

Pascal

```
property OnEndSession: TEndSessionEvent;
```

C++

```
__property TEndSessionEvent OnEndSession;
```

Description

Write an OnEndSession event handler to take specific action when a session terminates. For example, an OnEndSession event handler could store information from sessions that time out in a more persistent form such as a database.

See Also

OnStartSession (see page 6755)

Terminate (see page 6255)

TimeoutMinutes

68.1.2.3.2 TSessionsService.OnStartSession Event

Occurs when the sessions service detects that a new session has started.

Pascal

```
property OnStartSession: TStartSessionEvent;
```

C++

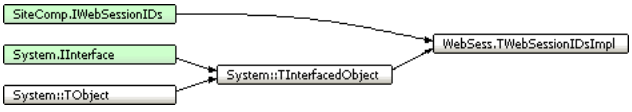
```
__property TStartSessionEvent OnStartSession;
```

Description

Write an OnStartSession event handler to take specific action when a session starts. For example, you can use this event to customize the time-out period for new sessions or initialize them by adding name/value pairs.

68.1.3 TWebSessionIDsImpl Class

Class Hierarchy



Pascal

```
TWebSessionIDsImpl = class(TInterfacedObject, IWebSessionIDs);
```

C++

```
class TWebSessionIDsImpl : public TInterfacedObject, public IWebSessionIDs;
```

File

WebSess

Description

This is class WebSess.TWebSessionIDsImpl.

68.1.3.1 TWebSessionIDsImpl Members

The following tables list the members exposed by TWebSessionIDsImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetID (see page 6762)	Returns the value of the IDs property.
	GetIDCount (see page 6763)	Returns the value of the IDCount property.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6761)	Constructs an object and initializes its data before the object is first used.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6761)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.

	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	IDCount (see page 1111)	Indicates the number of session IDs (see page 1111) available.
R	IDs (see page 1111)	Lists the session IDs for all the sessions stored by the sessions service.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

68.1.3.2 TWebSessionIDsImpl.Create Constructor

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create;
```

C++

```
__fastcall TWebSessionIDsImpl();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

68.1.3.3 TWebSessionIDsImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebSessionIDsImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

68.1.3.4 TWebSessionIDsImpl Methods

The methods of the TWebSessionIDsImpl class are listed here.

Protected Methods

	Name	Description
	GetID (see page 6762)	Returns the value of the IDs property.
	GetIDCount (see page 6763)	Returns the value of the IDCount property.

68.1.3.4.1 TWebSessionIDsImpl.GetID Method

Returns the value of the IDs property.

Pascal

```
function GetID(I: Integer): TSessionID;
```

C++

```
__fastcall TSessionID GetID(int I);
```

Description

GetID is the read implementation of the IDs property.

I is the index of the session ID to return.

See Also

IDs (see page 1111)

68.1.3.4.2 TWebSessionIDsImpl.GetIDCount Method

Returns the value of the IDCount property.

Pascal

```
function GetIDCount: Integer;
```

C++

```
__fastcall int GetIDCount();
```

Description

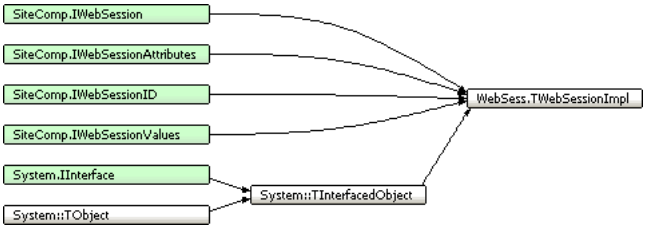
GetIDCount is the read implementation of the IDCount property.

See Also

IDCount (see page 1111)

68.1.4 TWebSessionImpl Class

Class Hierarchy



Pascal

```
TWebSessionImpl = class(TInterfacedObject, IWebSession, IWebSessionID, IWebSessionValues,
IWebSessionAttributes);
```

C++

```
class TWebSessionImpl : public TInterfacedObject, public IWebSession, public IWebSessionID,
public IWebSessionValues, public IWebSessionAttributes;
```

File

WebSess

Description

This is class WebSess.TWebSessionImpl.

68.1.4.1 TWebSessionImpl Members

The following tables list the members exposed by TWebSessionImpl.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	GetExpirationTime (see page 6766)	This is GetExpirationTime, a member of class TWebSessionImpl.
	GetExpired (see page 6767)	This is GetExpired, a member of class TWebSessionImpl.
	GetID (see page 6767)	This is GetID, a member of class TWebSessionImpl.
	GetIsActive (see page 6767)	This is GetIsActive, a member of class TWebSessionImpl.
	GetItemCount (see page 6767)	This is GetItemCount, a member of class TWebSessionImpl.
	GetItems (see page 6767)	This is GetItems, a member of class TWebSessionImpl.
	GetStartTime (see page 6768)	This is GetStartTime, a member of class TWebSessionImpl.
	GetTerminated (see page 6768)	This is GetTerminated, a member of class TWebSessionImpl.
	GetTimeoutMinutes (see page 6768)	This is GetTimeoutMinutes, a member of class TWebSessionImpl.
	GetTouchTime (see page 6768)	This is GetTouchTime, a member of class TWebSessionImpl.
	GetValue (see page 6769)	This is GetValue, a member of class TWebSessionImpl.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	SetTimeoutMinutes (see page 6769)	This is SetTimeoutMinutes, a member of class TWebSessionImpl.
	SetValue (see page 6769)	This is SetValue, a member of class TWebSessionImpl.
	Terminate (see page 6769)	Terminates the session.

Protected Properties

	Name	Description
R	ExpirationTime (see page 6770)	This is ExpirationTime, a member of class TWebSessionImpl.
R	Expired (see page 6770)	This is Expired, a member of class TWebSessionImpl.
R	ID (see page 6770)	This is ID, a member of class TWebSessionImpl.
R	StartTime (see page 6771)	This is StartTime, a member of class TWebSessionImpl.
R	Terminated (see page 6771)	This is Terminated, a member of class TWebSessionImpl.
R	TimeoutMinutes (see page 6771)	This is TimeoutMinutes, a member of class TWebSessionImpl.
R	TouchTime (see page 6771)	This is TouchTime, a member of class TWebSessionImpl.
	Values (see page 6771)	This is Values, a member of class TWebSessionImpl.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 6765)	This is Create, a member of class TWebSessionImpl.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 6765)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	IsActive (see page 1107)	This is IsActive, a member of class IWebSessionAttributes.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

68.1.4.2 TWebSessionImpl.Create Constructor

Pascal

```
constructor Create(AID: TSessionID);
```

C++

```
__fastcall TWebSessionImpl(TSessionID AID);
```

Description

This is Create, a member of class TWebSessionImpl.

68.1.4.3 TWebSessionImpl.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebSessionImpl();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

68.1.4.4 TWebSessionImpl Methods

The methods of the TWebSessionImpl class are listed here.

Protected Methods

	Name	Description
	GetExpirationTime (see page 6766)	This is GetExpirationTime, a member of class TWebSessionImpl.
	GetExpired (see page 6767)	This is GetExpired, a member of class TWebSessionImpl.
	GetID (see page 6767)	This is GetID, a member of class TWebSessionImpl.
	GetIsActive (see page 6767)	This is GetIsActive, a member of class TWebSessionImpl.
	GetItemCount (see page 6767)	This is GetItemCount, a member of class TWebSessionImpl.
	GetItems (see page 6767)	This is GetItems, a member of class TWebSessionImpl.
	GetStartTime (see page 6768)	This is GetStartTime, a member of class TWebSessionImpl.
	GetTerminated (see page 6768)	This is GetTerminated, a member of class TWebSessionImpl.
	GetTimeoutMinutes (see page 6768)	This is GetTimeoutMinutes, a member of class TWebSessionImpl.
	GetTouchTime (see page 6768)	This is GetTouchTime, a member of class TWebSessionImpl.
	GetValue (see page 6769)	This is GetValue, a member of class TWebSessionImpl.
	SetTimeoutMinutes (see page 6769)	This is SetTimeoutMinutes, a member of class TWebSessionImpl.
	SetValue (see page 6769)	This is SetValue, a member of class TWebSessionImpl.
	Terminate (see page 6769)	Terminates the session.

68.1.4.4.1 TWebSessionImpl.GetExpirationTime Method

Pascal

```
function GetExpirationTime: TDateTime;
```

C++

```
__fastcall TDateTime GetExpirationTime();
```

Description

This is GetExpirationTime, a member of class TWebSessionImpl.

68.1.4.4.2 TWebSessionImpl.GetExpired Method

Pascal

```
function GetExpired: Boolean;
```

C++

```
__fastcall Boolean GetExpired();
```

Description

This is GetExpired, a member of class TWebSessionImpl.

68.1.4.4.3 TWebSessionImpl.GetID Method

Pascal

```
function GetID: TSessionID;
```

C++

```
__fastcall TSessionID GetID();
```

Description

This is GetID, a member of class TWebSessionImpl.

68.1.4.4.4 TWebSessionImpl.GetIsActive Method

Pascal

```
function GetIsActive: Boolean;
```

C++

```
__fastcall Boolean GetIsActive();
```

Description

This is GetIsActive, a member of class TWebSessionImpl.

68.1.4.4.5 TWebSessionImpl.GetItemCount Method

Pascal

```
function GetItemCount: Integer;
```

C++

```
__fastcall int GetItemCount();
```

Description

This is GetItemCount, a member of class TWebSessionImpl.

68.1.4.4.6 TWebSessionImpl.GetItems Method

Pascal

```
procedure GetItems(var AItems: TAbstractNamedVariants);
```

C++

```
__fastcall GetItems(TAbstractNamedVariants AItems);
```

Description

This is GetItems, a member of class TWebSessionImpl.

68.1.4.4.7 TWebSessionImpl.GetStartTime Method

Pascal

```
function GetStartTime: TDateTime;
```

C++

```
__fastcall TDateTime GetStartTime();
```

Description

This is GetStartTime, a member of class TWebSessionImpl.

68.1.4.4.8 TWebSessionImpl.GetTerminated Method

Pascal

```
function GetTerminated: Boolean;
```

C++

```
__fastcall Boolean GetTerminated();
```

Description

This is GetTerminated, a member of class TWebSessionImpl.

68.1.4.4.9 TWebSessionImpl.GetTimeoutMinutes Method

Pascal

```
function GetTimeoutMinutes: Integer;
```

C++

```
__fastcall int GetTimeoutMinutes();
```

Description

This is GetTimeoutMinutes, a member of class TWebSessionImpl.

68.1.4.4.10 TWebSessionImpl.GetTouchTime Method

Pascal

```
function GetTouchTime: TDateTime;
```

C++

```
__fastcall TDateTime GetTouchTime();
```

Description

This is GetTouchTime, a member of class TWebSessionImpl.

68.1.4.4.11 TWebSessionImpl.GetValue Method

Pascal

```
function GetValue(const AName: string): Variant;
```

C++

```
__fastcall Variant GetValue(const UnicodeString AName);
```

Description

This is GetValue, a member of class TWebSessionImpl.

68.1.4.4.12 TWebSessionImpl.SetTimeoutMinutes Method

Pascal

```
procedure SetTimeoutMinutes(AValue: Integer);
```

C++

```
__fastcall SetTimeoutMinutes(int AValue);
```

Description

This is SetTimeoutMinutes, a member of class TWebSessionImpl.

68.1.4.4.13 TWebSessionImpl.SetValue Method

Pascal

```
procedure SetValue(const AName: string; const AValue: Variant);
```

C++

```
__fastcall SetValue(const UnicodeString AName, const Variant AValue);
```

Description

This is SetValue, a member of class TWebSessionImpl.

68.1.4.4.14 TWebSessionImpl.Terminate Method

Terminates the session.

Pascal

```
procedure Terminate;
```

C++

```
__fastcall Terminate();
```

Description

Call Terminate to terminate a session. Once a session terminates, the application can no longer access its Values.

Sessions are terminated automatically when the application does not receive any requests for the session for TimeOutMinutes minutes.

See Also

TimeoutMinutes

Values

68.1.4.5 TWebSessionImpl Properties

The properties of the TWebSessionImpl class are listed here.

Protected Properties

	Name	Description
R	ExpirationTime (see page 6770)	This is ExpirationTime, a member of class TWebSessionImpl.
R	Expired (see page 6770)	This is Expired, a member of class TWebSessionImpl.
R	ID (see page 6770)	This is ID, a member of class TWebSessionImpl.
R	StartTime (see page 6771)	This is StartTime, a member of class TWebSessionImpl.
R	Terminated (see page 6771)	This is Terminated, a member of class TWebSessionImpl.
R	TimeoutMinutes (see page 6771)	This is TimeoutMinutes, a member of class TWebSessionImpl.
R	TouchTime (see page 6771)	This is TouchTime, a member of class TWebSessionImpl.
	Values (see page 6771)	This is Values, a member of class TWebSessionImpl.

68.1.4.5.1 TWebSessionImpl.ExpirationTime Property

Pascal

```
property ExpirationTime: TDateTime;
```

C++

```
__property TDateTime ExpirationTime;
```

Description

This is ExpirationTime, a member of class TWebSessionImpl.

68.1.4.5.2 TWebSessionImpl.Expired Property

Pascal

```
property Expired: Boolean;
```

C++

```
__property Boolean Expired;
```

Description

This is Expired, a member of class TWebSessionImpl.

68.1.4.5.3 TWebSessionImpl.ID Property

Pascal

```
property ID: TSessionID;
```

C++

```
__property TSessionID ID;
```

Description

This is ID, a member of class TWebSessionImpl.

68.1.4.5.4 TWebSessionImpl.StartTime Property

Pascal

```
property StartTime: TDateTime;
```

C++

```
__property TDateTime StartTime;
```

Description

This is StartTime, a member of class TWebSessionImpl.

68.1.4.5.5 TWebSessionImpl.Terminated Property

Pascal

```
property Terminated: Boolean;
```

C++

```
__property Boolean Terminated;
```

Description

This is Terminated, a member of class TWebSessionImpl.

68.1.4.5.6 TWebSessionImpl.TimeoutMinutes Property

Pascal

```
property TimeoutMinutes: Integer;
```

C++

```
__property int TimeoutMinutes;
```

Description

This is TimeoutMinutes, a member of class TWebSessionImpl.

68.1.4.5.7 TWebSessionImpl.TouchTime Property

Pascal

```
property TouchTime: TDateTime;
```

C++

```
__property TDateTime TouchTime;
```

Description

This is TouchTime, a member of class TWebSessionImpl.

68.1.4.5.8 TWebSessionImpl.Values Property

Pascal

```
property Values [const AName: string]: Variant;
```

C++

```
__property Variant Values[UnicodeString const AName];
```

Description

This is Values, a member of class TWebSessionImpl.

68.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEndSessionReason (see page 6773)	TEndSessionReason indicates the reason a session ended.

68.2.1 WebSess.TEndSessionReason Enumeration

TEndSessionReason indicates the reason a session ended.

Pascal

```
TEndSessionReason = (
  esTimeout,
  esTerminate
);
```

C++

```
enum TEndSessionReason {
  esTimeout,
  esTerminate
};
```

File

WebSess

Description

The following table gives the possible values:

Value	Meaning
esTimeout	The session's time-out period elapsed without the application accessing the session.
esTerminate	The application called the session's Terminate method. This can be an explicit call in application call, or an implicit call when the user logs out in an application that uses TEndUserSessionsAdpater.

68.3 Types

The following table lists types in this documentation.

Types

Name	Description
TEndSessionEvent (see page 6774)	TEndSessionEvent is the type for event handlers that are called when a session terminates.
TStartSessionEvent (see page 6774)	TStartSessionEvent is the type for event handlers that are called when a session starts up.

68.3.1 WebSess.TEndSessionEvent Type

TEndSessionEvent is the type for event handlers that are called when a session terminates.

Pascal

```
TEndSessionEvent = procedure (ASender: TObject; ASession: TAbstractWebSession; AReason:
TEndSessionReason) of object;
```

C++

```
(ASender: TObject; ASession: TAbstractWebSession; AReason: TEndSessionReason) (
TEndSessionEvent)();
```

File

WebSess

Description

ASender is the sessions service component that manages the sessions in the WebSnap application.

ASession is the session that just terminated. The event handler can read its session ID and name/value pairs before they are deleted from memory.

AReason indicates the reason that the session was terminated.

See Also

Procedural types

Creating events

68.3.2 WebSess.TStartSessionEvent Type

TStartSessionEvent is the type for event handlers that are called when a session starts up.

Pascal

```
TStartSessionEvent = procedure (ASender: TObject; ASession: TAbstractWebSession) of object;
```

C++

```
(ASender: TObject; ASession: TAbstractWebSession) ( TStartSessionEvent)();
```

File

WebSess

Description

ASender is the sessions service component that manages the sessions in the WebSnap application.

ASession is the session object for the new session. Note that the object for a session may change over the lifetime of the session. The name/value pairs, session ID, and time-out period persist over changes in the session object.

See Also

Procedural types

Creating events

69 WebUsers Namespace

This is namespace WebUsers.

Topics

Name	Description
Classes (see page 6779)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 6801)	The following table lists structs, records, enums in this documentation.
Types (see page 6802)	The following table lists types in this documentation.

Classes

	Name	Description
	EUserIDNotFoundException (see page 6779)	EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.
	EValidateUserException (see page 6779)	EValidateUserException is raised when a Web user list fails to validate a user.
	EWebUsersListException (see page 6780)	EWebUsersListException is the base class for exceptions that TWebUserList (see page 6796) raises.
	TCustomWebUserList (see page 6781)	The TCustomWebUserList component uses a TWebUserItems (see page 6792) object to store the account information about a collection of users.
	TWebUserItem (see page 6787)	TWebUserItem stores the user identification, password, and access rights for a single user.
	TWebUserItems (see page 6792)	TWebUserItems stores multiple sets of user information.
	TWebUserList (see page 6796)	TWebUserList validates login and checks access rights for a particular user.

Structs, Records, Enums

	Name	Description
	TValidateUserError (see page 6801)	TValidateUserError indicates the type of problem that was encountered when trying to validate a user.

Types

Name	Description
TCheckAccessRightsEvent (see page 6802)	TCheckAccessRightsEvent is the type for event handlers that check the current user's access rights against a rights string.
TCheckAccessRightsHandledEvent (see page 6802)	TCheckAccessRightsHandledEvent is the type for event handlers that can potentially replace the built-in checking of a user's rights against an access string.
TValidateUserErrorEvent (see page 6803)	TValidateUserErrorEvent is the type for event handlers that compensate for errors that occur when trying to validate a user that is logging in.
TValidateUserEvent (see page 6804)	TValidateUserEvent is the type for event handlers that respond immediately after a user is validated.

TValidateUserHandledEvent (see page 6804)	TValidateUserHandledEvent is the type for event handlers that can validate a user that has just logged in.

69.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EUserIDNotFoundException (see page 6779)	EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.
	EValidateUserException (see page 6779)	EValidateUserException is raised when a Web user list fails to validate a user.
	EWebUsersListException (see page 6780)	EWebUsersListException is the base class for exceptions that TWebUserList (see page 6796) raises.
	TCustomWebUserList (see page 6781)	The TCustomWebUserList component uses a TWebUserItems (see page 6792) object to store the account information about a collection of users.
	TWebUserItem (see page 6787)	TWebUserItem stores the user identification, password, and access rights for a single user.
	TWebUserItems (see page 6792)	TWebUserItems stores multiple sets of user information.
	TWebUserList (see page 6796)	TWebUserList validates login and checks access rights for a particular user.

69.1.1 EUserIDNotFoundException Class

EUserIDNotFoundException is raised when a Web user list attempts to locate information about a user that is not in the list.

Class Hierarchy

Pascal

```
EUserIDNotFoundException = class(EWebUsersListException);
```

C++

```
class EUserIDNotFoundException : public EWebUsersListException;
```

File

WebUsers

Description

When a Web user list (TWebUserList (see page 6796)) attempts to look up information about a user and fails to find an entry for that user, it generates an OnUserIDNotFound event. If that event handler does not handle the situation, then the Web user list raises an EUserIDNotFoundException exception.

See Also

TWebUserList (see page 6796)

TCustomWebUserList (see page 6781)

69.1.2 EValidateUserException Class

EValidateUserException is raised when a Web user list fails to validate a user.

Class Hierarchy

Pascal

```
EValidateUserException = class(EWebUsersListException);
```

C++

```
class EValidateUserException : public EWebUsersListException;
```

File

WebUsers

Description

When a WebSnap application that contains a Web user list (TWebUserList (see page 6796)) tries to log a user in, the Web user list raises an EValidateUserException if it can't validate the supplied user ID and password and its OnValidateUserError event handler does not handle the situation.

The Web user list may fail to validate a user ID and password for any of the following reasons:

The WebSnap application failed to supply a password or user ID when trying to validate the user.

The supplied user ID does not refer to a user in the list.

The supplied password is not correct for the specified user.

See Also

TWebUserList (see page 6796)

TCustomWebUserList (see page 6781)

69.1.3 EWebUsersListException Class

EWebUsersListException is the base class for exceptions that TWebUserList (see page 6796) raises.

Class Hierarchy

Pascal

```
EWebUsersListException = class(EWebBrokerException);
```

C++

```
class EWebUsersListException : public EWebBrokerException;
```

File

WebUsers

Description

EWebUsersListException is the common base class for all exceptions that represent errors which TWebUserList (see page 6796) encounters. Its descendants include

EUserIDNotFoundException (see page 6779), which is raised when a WebSnap application tries to use the Web user list to look up information about a user that is not listed.

EValidateUserException (see page 6779), which is raised when the WebSnap application tries to log in a user and the Web

user list fails to validate the supplied user id and password.

See Also

TWebUserList (see page 6796)

EUserIDNotFoundException (see page 6779)

EValidateUserException (see page 6779)

69.1.4 TCustomWebUserList Class

The TCustomWebUserList component uses a TWebUserItems (see page 6792) object to store the account information about a collection of users.

Class Hierarchy

Pascal

```
TCustomWebUserList = class(TComponent, IWebUserList);
```

C++

```
class TCustomWebUserList : public TComponent, public IWebUserList;
```

File

WebUsers

Description

The TCustomWebUserList component uses a TWebUserItems (see page 6792) object to store the account information about a collection of users. It also provides a way to save and load the user list from a file. Several events are triggered at stages that occur when a user is logging in that allow checks to be performed during the user validation process, including the verification of user rights.

See Also

TWebUserItems (see page 6792)

TWebUserList (see page 6796)

69.1.4.1 TCustomWebUserList Members

The following tables list the members exposed by TCustomWebUserList.

Protected Methods

	Name	Description
	CheckAccessRights (see page 6783)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 6784)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6781) object.

Public Events

	Name	Description
	OnAfterCheckAccessRights (see page 6785)	Triggered at the start of the CheckAccessRights (see page 6783) protected function.

	OnAfterValidateUser (see page 6786)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6786)	Triggered at the end of the CheckAccessRights (see page 6783) protected function.
	OnBeforeValidateUser (see page 6786)	Triggered at the beginning of the ValidateUser (see page 6784) function.
	OnUserIDNotFound (see page 6787)	Triggered if the UserID is not found in the TCustomWebUserList (see page 6781) object.
	OnValidateUserError (see page 6787)	Triggered if an error occurs during the validation process.

Public Methods

	Name	Description
V	Create (see page 6782)	Creates a newTCustomWebUserList object.
V	Destroy (see page 6782)	Frees the TCustomWebUserList (see page 6781) object.
	LoadFromFile (see page 6783)	Loads the data from the file with the given filename.
	SaveToFile (see page 6784)	This procedure saves the collection of the user's account information to a file with the given filename.

Public Properties

	Name	Description
	UserItems (see page 6784)	Stores the account information about a collection of users.
R	Users (see page 6785)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6787).

69.1.4.2 TCustomWebUserList.Create Constructor

Creates a newTCustomWebUserList object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TCustomWebUserList(TComponent * AOwner);
```

Description

Creates a newTCustomWebUserList object and initializes the TWebUserItems (see page 6792) object that will store the user's account information.

69.1.4.3 TCustomWebUserList.Destroy Destructor

Frees the TCustomWebUserList (see page 6781) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TCustomWebUserList();
```

Description

Frees the TCustomWebUserList (see page 6781) object initialized by the constructor.

69.1.4.4 TCustomWebUserList Methods

The methods of the TCustomWebUserList class are listed here.

Protected Methods

	Name	Description
	CheckAccessRights (see page 6783)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 6784)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6781) object.

Public Methods

	Name	Description
	LoadFromFile (see page 6783)	Loads the data from the file with the given filename.
	SaveToFile (see page 6784)	This procedure saves the collection of the user's account information to a file with the given filename.

69.1.4.4.1 TCustomWebUserList.CheckAccessRights Method

Checks whether a specified user has a given access rights string.

Pascal

```
function CheckAccessRights(AUserID: Variant; ARights: string): Boolean;
```

C++

```
__fastcall Boolean CheckAccessRights(Variant AUserID, UnicodeString ARights);
```

Description

Call CheckAccessRights to determine whether the current end user is authorized to view Web pages or adapters whose access rights string is Rights. The end user must be logged in before CheckAccessRights can determine the user's access rights.

UserID is the user ID of the user to check.

Rights is the access rights string that CheckAccessRights must look for. Typically, this is the value of the ViewRights property on a Web page information object that describes the page the user wants to access or on an adapter that the user wants to call.

See Also

HasViewRights (see page 5510)

HasModifyRights (see page 5509)

HasExecuteRights (see page 5509)

ViewAccess

ViewAccess (see page 5574)

69.1.4.4.2 TCustomWebUserList.LoadFromFile Method

Loads the data from the file with the given filename.

Pascal

```
procedure LoadFromFile(FileName: string);
```

C++

```
__fastcall LoadFromFile(UnicodeString FileName);
```

Description

Loads the data from the file with the given filename into the TCustomWebUserList (see page 6781) object.

69.1.4.4.3 TCustomWebUserList.SaveToFile Method

This procedure saves the collection of the user's account information to a file with the given filename.

Pascal

```
procedure SaveToFile(FileName: string);
```

C++

```
__fastcall SaveToFile(UnicodeString FileName);
```

Description

Saves the collection of the user's account information to a file with the given filename. This file can later be reloaded by calling LoadFromFile (see page 6783).

69.1.4.4.4 TCustomWebUserList.ValidateUser Method

Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6781) object.

Pascal

```
function ValidateUser(Strings: TStrings): Variant;
```

C++

```
__fastcall Variant ValidateUser(TStrings Strings);
```

Description

This function verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6781) object. Before the validation begins, it triggers the OnBeforeValidateUser (see page 6786) event. If an error occurs during the validation check, the OnValidateUserError (see page 6787) event will be triggered and a description of the error is sent, which could be either a missing or invalid password, or a missing or unknown user name. After the validation is complete, this function triggers the OnAfterValidateUser (see page 6786) event.

69.1.4.5 TCustomWebUserList Properties

The properties of the TCustomWebUserList class are listed here.

Public Properties

	Name	Description
	UserItems (see page 6784)	Stores the account information about a collection of users.
R	Users (see page 6785)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6787).

69.1.4.5.1 TCustomWebUserList.UserItems Property

Stores the account information about a collection of users.

Pascal

```
property UserItems: TWebUserItems;
```

C++

```
__property TWebUserItems UserItems;
```

Description

Use this object to store the account information about a collection of users. This property provides read and write access to the stored collection.

See Also

Users (see page 6785)

69.1.4.5.2 TCustomWebUserList.Users Property

Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6787).

Pascal

```
property Users [I: Integer]: TWebUserItem;
```

C++

```
__property TWebUserItem Users[int I];
```

Description

Use this property to provide read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6787). An individual TWebUserItem (see page 6787) object can be indexed from the array using the standard bracketed array notation.

See Also

UserItems (see page 6784)

69.1.4.6 TCustomWebUserList Events

The events of the TCustomWebUserList class are listed here.

Public Events

	Name	Description
	OnAfterCheckAccessRights (see page 6785)	Triggered at the start of the CheckAccessRights (see page 6783) protected function.
	OnAfterValidateUser (see page 6786)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6786)	Triggered at the end of the CheckAccessRights (see page 6783) protected function.
	OnBeforeValidateUser (see page 6786)	Triggered at the beginning of the ValidateUser (see page 6784) function.
	OnUserIDNotFound (see page 6787)	Triggered if the UserID is not found in the TCustomWebUserList (see page 6781) object.
	OnValidateUserError (see page 6787)	Triggered if an error occurs during the validation process.

69.1.4.6.1 TCustomWebUserList.OnAfterCheckAccessRights Event

Triggered at the start of the CheckAccessRights (see page 6783) protected function.

Pascal

```
property OnAfterCheckAccessRights: TCheckAccessRightsEvent;
```

C++

```
__property TCheckAccessRightsEvent OnAfterCheckAccessRights;
```

Description

This event is triggered at the start of the CheckAccessRights (see page 6783) protected function. This event can be used if

any additional checking or initializing is desired before the access rights are checked.

See Also

OnBeforeCheckAccessRights (see page 6786)

69.1.4.6.2 TCustomWebUserList.OnAfterValidateUser Event

Occurs after a user has been validated.

Pascal

```
property OnAfterValidateUser: TValidateUserEvent;
```

C++

```
__property TValidateUserEvent OnAfterValidateUser;
```

Description

This event is triggered at the end of the ValidateUser (see page 6784) function. It can be used if any additional processing needs to be done after a user's name and password are verified.

See Also

OnBeforeValidateUser (see page 6786)

ValidateUser (see page 6784)

69.1.4.6.3 TCustomWebUserList.OnBeforeCheckAccessRights Event

Triggered at the end of the CheckAccessRights (see page 6783) protected function.

Pascal

```
property OnBeforeCheckAccessRights: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnBeforeCheckAccessRights;
```

Description

This event is triggered at the end of the CheckAccessRights (see page 6783) protected function. It can be used to perform your own custom checking instead of the default checking by the Web user list.

See Also

OnAfterCheckAccessRights (see page 6785)

69.1.4.6.4 TCustomWebUserList.OnBeforeValidateUser Event

Triggered at the beginning of the ValidateUser (see page 6784) function.

Pascal

```
property OnBeforeValidateUser: TValidateUserHandledEvent;
```

C++

```
__property TValidateUserHandledEvent OnBeforeValidateUser;
```

Description

This event is triggered at the beginning of the ValidateUser (see page 6784) function. It can be used to identify a user given a name and password and supply a user ID for the user, once identified.

See Also

OnAfterValidateUser (see page 6786)

69.1.4.6.5 TCustomWebUserList.OnUserIDNotFound Event

Triggered if the UserID is not found in the TCustomWebUserList (see page 6781) object.

Pascal

```
property OnUserIDNotFound: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnUserIDNotFound;
```

Description

This event is triggered if the UserID is not found in the TCustomWebUserList (see page 6781) object. The CheckAccessRights (see page 6783) protected function triggers this function if the UserID is not found while checking the access rights.

69.1.4.6.6 TCustomWebUserList.OnValidateUserError Event

Triggered if an error occurs during the validation process.

Pascal

```
property OnValidateUserError: TValidateUserErrorEvent;
```

C++

```
__property TValidateUserErrorEvent OnValidateUserError;
```

Description

This event is triggered if an error occurs during the validation process. The ValidateUser (see page 6784) function calls this event with a description of the error. Examples of error descriptions include a missing or unknown user name or a missing or invalid password. Any additional processing that needs to be completed in the case of one of these error conditions can be added to this event handler.

See Also

ValidateUser (see page 6784)

69.1.5 TWebUserItem Class

TWebUserItem stores the user identification, password, and access rights for a single user.

Class Hierarchy

Pascal

```
TWebUserItem = class(TCollectionItem);
```

C++

```
class TWebUserItem : public TCollectionItem;
```

File

WebUsers

Description

TWebUserItem stores the user ID and password along with the associated access rights for a single user. Methods are provided to set the access rights as well as methods to retrieve the access rights in the form of a single delimited string, or as a collection in the form of a TString.

See Also

TWebUserList (see page 6796)

TWebUserItems (see page 6792)

69.1.5.1 TWebUserItem Members

The following tables list the members exposed by TWebUserItem.

Public Methods

	Name	Description
	Assign (see page 6789)	Copies the UserName (see page 6791), Password (see page 6791), and AccessRights (see page 6790) from the given TWebUserItem (see page 6787).
	CheckRights (see page 6790)	Checks the user's stored access rights against the access rights passed in.
	Create (see page 6788)	Creates an instance of Create.
	Destroy (see page 6789)	Destroys an instance of TWebUserItem (see page 6787).

Public Properties

	Name	Description
	AccessRightsStrings (see page 6791)	Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.
	UserID (see page 6791)	Provides read-only access to the user ID.

Published Properties

	Name	Description
	AccessRights (see page 6790)	Stores the user rights as a comma, semicolon, or space delimited string.
	Password (see page 6791)	Stores the user's password as a string.
	UserName (see page 6791)	Stores the user's name.

69.1.5.2 TWebUserItem.Create Constructor

Creates an instance of Create.

Pascal

```
constructor Create(Collection: TCollection); override;
```

C++

```
virtual __fastcall TWebUserItem(TCollection Collection);
```

Description

Do not call Create. It is called automatically when the Create object is created.

69.1.5.3 TWebUserItem.Destroy Destructor

Destroys an instance of TWebUserItem (see page 6787).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWebUserItem();
```

Description

The Destroy destructor frees all objects owned by the TWebUserItem (see page 6787) object. Never call the destructor explicitly; it is called automatically when the TWebUserItem (see page 6787) object is destroyed (for example, when the object falls out of the scope of the program).

See Also

delete

69.1.5.4 TWebUserItem Methods

The methods of the TWebUserItem class are listed here.

Public Methods

	Name	Description
	Assign (see page 6789)	Copies the UserName (see page 6791), Password (see page 6791), and AccessRights (see page 6790) from the given TWebUserItem (see page 6787).
	CheckRights (see page 6790)	Checks the user's stored access rights against the access rights passed in.

69.1.5.4.1 TWebUserItem.Assign Method

Copies the UserName (see page 6791), Password (see page 6791), and AccessRights (see page 6790) from the given TWebUserItem (see page 6787).

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

The Assign procedure copies the UserName (see page 6791), Password (see page 6791), and AccessRights (see page 6790) from the given TWebUserItem (see page 6787).

See Also

UserName (see page 6791)

Password (see page 6791)

AccessRights (see page 6790)

69.1.5.4.2 TWebUserItem.CheckRights Method

Checks the user's stored access rights against the access rights passed in.

Pascal

```
function CheckRights(const ARights: string): Boolean;
```

C++

```
__fastcall Boolean CheckRights(const UnicodeString ARights);
```

Description

This function checks the user's stored access rights against the access rights passed in. If the user has a matching right, then the function returns true or it will return false if the rights do not match.

See Also

AccessRights (see page 6790)

69.1.5.5 TWebUserItem Properties

The properties of the TWebUserItem class are listed here.

Public Properties

	Name	Description
R	AccessRightsStrings (see page 6791)	Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.
R	UserID (see page 6791)	Provides read-only access to the user ID.

Published Properties

	Name	Description
	AccessRights (see page 6790)	Stores the user rights as a comma, semicolon, or space delimited string.
	Password (see page 6791)	Stores the user's password as a string.
	UserName (see page 6791)	Stores the user's name.

69.1.5.5.1 TWebUserItem.AccessRights Property

Stores the user rights as a comma, semicolon, or space delimited string.

Pascal

```
property AccessRights: string;
```

C++

```
__property UnicodeString AccessRights;
```

Description

The AccessRights property stores the user rights as a comma, semicolon, or space delimited string. This variable can be read or written directly or set as a single string using SetAccessRights. The individual rights can be obtained as a collection in the form of a TString by calling the GetAccessRightsStrings function.

See Also

CheckRights (see page 6790)

69.1.5.5.2 TWebUserItem.AccessRightsStrings Property

Stores the access rights for the user as a collection of individual strings in the form of a TStrings object.

Pascal

```
property AccessRightsStrings: TStrings;
```

C++

```
__property TStrings AccessRightsStrings;
```

Description

The AccessRightsStrings property stores the access rights for the user as a collection of individual strings in the form of a TStrings object.

See Also

AccessRights (see page 6790)

TStrings

69.1.5.5.3 TWebUserItem.Password Property

Stores the user's password as a string.

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

The Password property stores the user's password as a string.

69.1.5.5.4 TWebUserItem.UserID Property

Provides read-only access to the user ID.

Pascal

```
property UserID: Variant;
```

C++

```
__property Variant UserID;
```

Description

The UserID property provides read-only access to the user ID.

See Also

System::Variant (see page 3421)

69.1.5.5.5 TWebUserItem.UserName Property

Stores the user's name.

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

The UserName property stores the user's name. This property can be read or written directly.

69.1.6 TWebUserItems Class

TWebUserItems stores multiple sets of user information.

Class Hierarchy

Pascal

```
TWebUserItems = class(TCollection);
```

C++

```
class TWebUserItems : public TCollection;
```

File

WebUsers

Description

The TWebUserItems stores multiple TWebUserItem (see page 6787) modules as an array. It also provides methods to access a particular user's information by giving the user name or user ID.

69.1.6.1 TWebUserItems Members

The following tables list the members exposed by TWebUserItems.

Protected Methods

	Name	Description
	GetAttr (see page 6794)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 6794)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 6795)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 6795)	Returns the owner of an object.

Public Methods

	Name	Description
	Create (see page 6793)	Constructor used for the TWebUserItems (see page 6792) object.
	FindUserID (see page 6793)	Returns the user ID, password, and access rights information for the user.
	FindUserName (see page 6793)	Returns the user ID, password, and access rights information for the user.

Public Properties

	Name	Description
	Items (see page 6796)	Allows public read and write access to individual TWebUserItem (see page 6787) records as if they were stored in an array.

69.1.6.2 TWebUserItems.Create Constructor

Constructor used for the TWebUserItems (see page 6792) object.

Pascal

```
constructor Create(AOwner: TComponent; ItemClass: TCollectionItemClass);
```

C++

```
__fastcall TWebUserItems(TComponent * AOwner, TCollectionItemClass ItemClass);
```

Description

The constructor used for the TWebUserItems (see page 6792) object. This method creates a new TWebUserItem (see page 6787) and sets the owner.

69.1.6.3 TWebUserItems Methods

The methods of the TWebUserItems class are listed here.

Protected Methods

	Name	Description
	GetAttr (see page 6794)	Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.
	GetAttrCount (see page 6794)	Returns the number of custom attributes associated with items in the collection.
	GetItemAttr (see page 6795)	Returns the value of a custom attribute assigned to one of the collection's items.
	GetOwner (see page 6795)	Returns the owner of an object.

Public Methods

	Name	Description
	FindUserID (see page 6793)	Returns the user ID, password, and access rights information for the user.
	FindUserName (see page 6793)	Returns the user ID, password, and access rights information for the user.

69.1.6.3.1 TWebUserItems.FindUserID Method

Returns the user ID, password, and access rights information for the user.

Pascal

```
function FindUserID(AUserID: Variant): TWebUserItem;
```

C++

```
__fastcall TWebUserItem FindUserID(Variant AUserID);
```

Description

This method returns the user ID, password, and access rights information for the user with the given user name in the form of a TWebUserItem (see page 6787) object.

69.1.6.3.2 TWebUserItems.FindUserName Method

Returns the user ID, password, and access rights information for the user.

Pascal

```
function FindUserName(AUserName: string): TWebUserItem;
```

C++

```
__fastcall TWebUserItem FindUserName(UnicodeString AUserName);
```

Description

This method returns the user ID, password, and access rights information for the user with the given user ID in the form of a TWebUserItem (see page 6787) object.

69.1.6.3.3 TWebUserItems.GetAttr Method

Returns the name of a custom attribute that can be retrieved using the GetItemAttr method.

Pascal

```
function GetAttr(Index: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetAttr(int Index);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. The GetAttr method returns the name of an attribute.

Index identifies the attribute whose name is requested. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

As implemented by TCollection, GetAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttrCount

GetItemAttr

69.1.6.3.4 TWebUserItems.GetAttrCount Method

Returns the number of custom attributes associated with items in the collection.

Pascal

```
function GetAttrCount: Integer; override;
```

C++

```
virtual __fastcall int GetAttrCount();
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetAttrCount returns the number of distinct attributes assigned to each item in the collection.

As implemented in TCollection, GetAttrCount always returns 0, because TCollection defines no custom attributes.

See Also

GetAttr

GetItemAttr

69.1.6.3.5 TWebUserItems.GetItemAttr Method

Returns the value of a custom attribute assigned to one of the collection's items.

Pascal

```
function GetItemAttr(Index: Integer; ItemIndex: Integer): string; override;
```

C++

```
virtual __fastcall UnicodeString GetItemAttr(int Index, int ItemIndex);
```

Description

TCollection descendants can associate user-defined attributes with the items in the collection. Each attribute has a name and, for each item in the collection, a value that is a string. GetItemAttr returns the value of one of these attributes for a specified item in the collection.

Index identifies which attribute's value is desired. This is a value between 0 and n-1, where n is the value returned by GetAttrCount.

ItemIndex identifies the item whose attribute value is desired. This is an index into the Items property array.

As implemented in TCollection, GetItemAttr always returns an empty string, because TCollection defines no custom attributes.

See Also

GetAttr

GetAttrCount

69.1.6.3.6 TWebUserItems.GetOwner Method

Returns the owner of an object.

Pascal

```
function GetOwner: TPersistent; override;
```

C++

```
virtual __fastcall TPersistent * GetOwner();
```

Description

GetOwner is used by the GetNamePath method to find the owner of a persistent object. GetNamePath and GetOwner are introduced in TPersistent so descendants such as collections can appear in the Object Inspector. As implemented in TPersistent, GetOwner returns nil (Delphi) or NULL (C++).

For TCollection, GetOwner returns the owner of the collection. For collection items, it returns the collection object into which the collection item has been inserted. For TComponent it is the value of the Owner property.

See Also

GetNamePath

TCollection

TComponent

Owner

69.1.6.4 TWebUserItems Properties

The properties of the TWebUserItems class are listed here.

Public Properties

	Name	Description
	Items (see page 6796)	Allows public read and write access to individual TWebUserItem (see page 6787) records as if they were stored in an array.

69.1.6.4.1 TWebUserItems.Items Property

Allows public read and write access to individual TWebUserItem (see page 6787) records as if they were stored in an array.

Pascal

```
property Items [Index: Integer]: TWebUserItem;
```

C++

```
__property TWebUserItem Items[int Index];
```

Description

The Items property allows public read and write access to individual TWebUserItem (see page 6787) records as if they were stored in an array. It also provides methods to access a particular user's information by giving the user name or user ID.

69.1.7 TWebUserList Class

TWebUserList validates login and checks access rights for a particular user.

Class Hierarchy

Pascal

```
TWebUserList = class(TCustomWebUserList);
```

C++

```
class TWebUserList : public TCustomWebUserList;
```

File

WebUsers

Description

Use TWebUserList, which contains a list of user names, passwords, and access rights, to validate login and to check a user's access rights.

The TWebUserList component provides events that are triggered during the initial stages when a user is logging in, before and after the user is validated, before and after the user's access rights are checked, in the case of an error, and when detecting an invalid user.

See Also

TWebUserItems (see page 6792)

TWebUserItem (see page 6787)

Adding Login Support

69.1.7.1 TWebUserList Members

The following tables list the members exposed by TWebUserList.

Protected Methods

	Name	Description
	CheckAccessRights (see page 6783)	Checks whether a specified user has a given access rights string.
	ValidateUser (see page 6784)	Verifies that the given user name and password matches one of the entries stored in the TCustomWebUserList (see page 6781) object.

Public Methods

	Name	Description
	CheckAccessRights (see page 1117)	Checks whether a specified user has a given access rights string.
V	Create (see page 6782)	Creates a newTCustomWebUserList object.
V	Destroy (see page 6782)	Frees the TCustomWebUserList (see page 6781) object.
	LoadFromFile (see page 6783)	Loads the data from the file with the given filename.
	SaveToFile (see page 6784)	This procedure saves the collection of the user's account information to a file with the given filename.
	ValidateUser (see page 1117)	Validates a user login.

Public Properties

	Name	Description
R	Users (see page 6785)	Provides read-only access to an individual user's account data in the form of an array of TWebUserItem (see page 6787).

Published Events

	Name	Description
	OnAfterCheckAccessRights (see page 6798)	Triggered at the start of the CheckAccessRights protected function.
	OnAfterValidateUser (see page 6798)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6799)	Triggered at the end of the CheckAccessRights protected function.
	OnBeforeValidateUser (see page 6799)	Triggered at the beginning of the ValidateUser function.
	OnUserIDNotFound (see page 6799)	Triggered if the UserID is not found in the TCustomWebUserList object.
	OnValidateUserError (see page 6800)	Triggered if an error occurs during the validation process.

Published Properties

	Name	Description
	UserItems (see page 6797)	Stores the account information about a collection of users.

69.1.7.2 TWebUserList Properties

The properties of the TWebUserList class are listed here.

Published Properties

	Name	Description
	UserItems (see page 6797)	Stores the account information about a collection of users.

69.1.7.2.1 TWebUserList.UserItems Property

Stores the account information about a collection of users.

Pascal

```
property UserItems: TWebUserItems;
```

C++

```
__property TWebUserItems UserItems;
```

Description

Use this object to store the account information about a collection of users. This property provides read and write access to the stored collection.

See Also

Users (see page 6785)

69.1.7.3 TWebUserList Events

The events of the TWebUserList class are listed here.

Published Events

	Name	Description
	OnAfterCheckAccessRights (see page 6798)	Triggered at the start of the CheckAccessRights protected function.
	OnAfterValidateUser (see page 6798)	Occurs after a user has been validated.
	OnBeforeCheckAccessRights (see page 6799)	Triggered at the end of the CheckAccessRights protected function.
	OnBeforeValidateUser (see page 6799)	Triggered at the beginning of the ValidateUser function.
	OnUserIDNotFound (see page 6799)	Triggered if the UserID is not found in the TCustomWebUserList object.
	OnValidateUserError (see page 6800)	Triggered if an error occurs during the validation process.

69.1.7.3.1 TWebUserList.OnAfterCheckAccessRights Event

Triggered at the start of the CheckAccessRights protected function.

Pascal

```
property OnAfterCheckAccessRights: TCheckAccessRightsEvent;
```

C++

```
__property TCheckAccessRightsEvent OnAfterCheckAccessRights;
```

Description

This event is triggered at the start of the CheckAccessRights protected function. This event can be used if any additional checking or initializing is desired before the access rights are checked.

See Also

OnBeforeCheckAccessRights (see page 6786)

69.1.7.3.2 TWebUserList.OnAfterValidateUser Event

Occurs after a user has been validated.

Pascal

```
property OnAfterValidateUser: TValidateUserEvent;
```

C++

```
__property TValidateUserEvent OnAfterValidateUser;
```

Description

This event is triggered at the end of the ValidateUser function. It can be used if any additional processing needs to be done after a user's name and password are verified.

See Also

OnBeforeValidateUser (see page 6786)

ValidateUser (see page 6784)

69.1.7.3.3 TWebUserList.OnBeforeCheckAccessRights Event

Triggered at the end of the CheckAccessRights protected function.

Pascal

```
property OnBeforeCheckAccessRights: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnBeforeCheckAccessRights;
```

Description

This event is triggered at the end of the CheckAccessRights protected function. It can be used to perform your own custom checking instead of the default checking by the Web user list.

See Also

OnAfterCheckAccessRights (see page 6785)

69.1.7.3.4 TWebUserList.OnBeforeValidateUser Event

Triggered at the beginning of the ValidateUser function.

Pascal

```
property OnBeforeValidateUser: TValidateUserHandledEvent;
```

C++

```
__property TValidateUserHandledEvent OnBeforeValidateUser;
```

Description

This event is triggered at the beginning of the ValidateUser function. It can be used to identify a user given a name and password and supply a user ID for the user, once identified.

See Also

OnAfterValidateUser (see page 6786)

69.1.7.3.5 TWebUserList.OnUserIDNotFound Event

Triggered if the UserID is not found in the TCustomWebUserList object.

Pascal

```
property OnUserIDNotFound: TCheckAccessRightsHandledEvent;
```

C++

```
__property TCheckAccessRightsHandledEvent OnUserIDNotFound;
```

Description

This event is triggered if the UserID is not found in the TCustomWebUserList object. The CheckAccessRights protected function triggers this function if the UserID is not found while checking the access rights.

69.1.7.3.6 TWebUserList.OnValidateUserError Event

Triggered if an error occurs during the validation process.

Pascal

```
property OnValidateUserError: TValidateUserErrorEvent;
```

C++

```
__property TValidateUserErrorEvent OnValidateUserError;
```

Description

This event is triggered if an error occurs during the validation process. The ValidateUser function calls this event with a description of the error. Examples of error descriptions include a missing or unknown user name or a missing or invalid password. Any additional processing that needs to be completed in the case of one of these error conditions can be added to this event handler.

See Also

ValidateUser (see page 6784)

69.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TValidateUserError (see page 6801)	TValidateUserError indicates the type of problem that was encountered when trying to validate a user.

69.2.1 WebUsers.TValidateUserError Enumeration

TValidateUserError indicates the type of problem that was encountered when trying to validate a user.

Pascal

```
TValidateUserError = (
  vuBlankPassword,
  vuBlankUserName,
  vuUnknownUserName,
  vuUnknownPassword
);
```

C++

```
enum TValidateUserError {
  vuBlankPassword,
  vuBlankUserName,
  vuUnknownUserName,
  vuUnknownPassword
};
```

File

WebUsers

Description

The following table lists the possible values:

Value	Meaning
vuBlankPassword	The user did not supply a password, or supplied an empty string.
vuBlankUserName	The user did not supply a user name, or supplied an empty string.
vuUnknownUserName	The user supplied a user name that the Web user list could not identify.
vuUnknownPassword	The user supplied a password that the Web user list could not find associated with the specified user.

69.3 Types

The following table lists types in this documentation.

Types

Name	Description
TCheckAccessRightsEvent (see page 6802)	TCheckAccessRightsEvent is the type for event handlers that check the current user's access rights against a rights string.
TCheckAccessRightsHandledEvent (see page 6802)	TCheckAccessRightsHandledEvent is the type for event handlers that can potentially replace the built-in checking of a user's rights against an access string.
TValidateUserErrorEvent (see page 6803)	TValidateUserErrorEvent is the type for event handlers that compensate for errors that occur when trying to validate a user that is logging in.
TValidateUserEvent (see page 6804)	TValidateUserEvent is the type for event handlers that respond immediately after a user is validated.
TValidateUserHandledEvent (see page 6804)	TValidateUserHandledEvent is the type for event handlers that can validate a user that has just logged in.

69.3.1 WebUsers.TCheckAccessRightsEvent Type

TCheckAccessRightsEvent is the type for event handlers that check the current user's access rights against a rights string.

Pascal

```
TCheckAccessRightsEvent = procedure (UserID: Variant; Rights: string; var HasRight: Boolean)
of object;
```

C++

```
(UserID: Variant; Rights: string; var HasRight: Boolean) ( TCheckAccessRightsEvent)();
```

File

WebUsers

Description

UserID is the user ID of the user whose rights should be checked.

Rights is a string that represents the rights the user should have to be granted access. Typically, this value comes from the ViewRights property of a Web page information object that describes the page that the user wants to access or of an adapter that the user wants to call.

HasRight returns true if the event handler grants the user access rights, false if the event handler denies access.

See Also

Procedural types

Creating events

69.3.2 WebUsers.TCheckAccessRightsHandledEvent Type

TCheckAccessRightsHandledEvent is the type for event handlers that can potentially replace the built-in checking of a user's

rights against an access string.

Pascal

```
TCheckAccessRightsHandledEvent = procedure (UserID: Variant; Rights: string; var HasRight:
Boolean; var Handled: Boolean) of object;
```

C++

```
(UserID: Variant; Rights: string; var HasRight: Boolean; var Handled: Boolean) (
TCheckAccessRightsHandledEvent)();
```

File

WebUsers

Description

UserID is the user ID of the user whose rights should be checked.

Rights is a string that represents the rights the user should have to be granted access. Typically, this value comes from the ViewRights property of a Web page information object that describes the page that the user wants to access or of an adapter that the user wants to call.

HasRight returns true if the event handler grants the user access rights, false if the event handler denies access.

Handled returns true if the event handler has checked the end user's rights, false if the HasRight parameter has no meaning on exit. If Handled returns false, the Web user list continues with its default processing, such as checking of the end user's rights or denying access. If Handled returns true, the Web user list skips this step.

See Also

Procedural types

Creating events

69.3.3 WebUsers.TValidateUserErrorEvent Type

TValidateUserErrorEvent is the type for event handlers that compensate for errors that occur when trying to validate a user that is logging in.

Pascal

```
TValidateUserErrorEvent = procedure (Error: TValidateUserError; Strings: TStrings; var UserID:
Variant; var Handled: Boolean) of object;
```

C++

```
(Error: TValidateUserError; Strings: TStrings; var UserID: Variant; var Handled: Boolean) (
TValidateUserErrorEvent)();
```

File

WebUsers

Description

Error (see page 3560) indicates what type of problem was encountered while trying to validate the user.

Strings is a string list that contains information the user supplied while logging in. At a minimum, this includes two strings, of the form UserName=Name and Password=Pword, where Name and Pword are strings that the user supplied. If the application uses a custom login form, additional strings may be included as well.

UserID returns an ID that the event handler supplies to uniquely identify the user that just logged in.

Handled returns true if the event handler compensated for the error and identified the user given the parameters in Strings and supplied an ID using the UserID parameter. Error (see page 3560) returns false if the event handler could not compensate for the error.

See Also

Procedural types

Creating events

69.3.4 WebUsers.TValidateUserEvent Type

TValidateUserEvent is the type for event handlers that respond immediately after a user is validated.

Pascal

```
TValidateUserEvent = procedure (Strings: TStrings; var UserID: Variant) of object;
```

C++

```
(Strings: TStrings; var UserID: Variant) ( TValidateUserEvent)();
```

File

WebUsers

Description

Strings is a string list that contains information the user supplied while logging in. At a minimum, this includes two strings, of the form UserName=Name and Password=Pword, where Name and Pword are strings that the user supplied. If the application uses a custom login form, additional strings may be included as well.

UserID is the ID that the Web user list assigned to the user based on the supplied name and password.

See Also

Procedural types

Creating events

69.3.5 WebUsers.TValidateUserHandledEvent Type

TValidateUserHandledEvent is the type for event handlers that can validate a user that has just logged in.

Pascal

```
TValidateUserHandledEvent = procedure (Strings: TStrings; var UserID: Variant; var Handled:
Boolean) of object;
```

C++

```
(Strings: TStrings; var UserID: Variant; var Handled: Boolean) ( TValidateUserHandledEvent)();
```

File

WebUsers

Description

Strings is a string list that contains information the user supplied while logging in. At a minimum, this includes two strings, of the form UserName=Name and Password=Pword, where Name and Pword are strings that the user supplied. If the application uses

a custom login form, additional strings may be included as well.

UserID returns an ID that the event handler supplies to uniquely identify the user that just logged in.

Handled returns true if the event handler identified the user given the parameters in Strings and supplied an ID using the UserID parameter. Handled returns false if the event handler did not return a user ID for the current user. If Handled returns false, the Web user list proceeds with its own attempt to identify the user and assign a user ID.

See Also

Procedural types

Creating events

70 WideStrings Namespace

This is namespace WideStrings.

Topics

Name	Description
Classes (see page 6808)	The following table lists classes in this documentation.
Interfaces (see page 6859)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 6861)	The following table lists structs, records, enums in this documentation.
Types (see page 6862)	The following table lists types in this documentation.

Classes

	Name	Description
	TWideStringList (see page 6808)	TWideStringList maintains a list of strings.
	TWideStrings (see page 6825)	TWideStrings is the base class for objects that represent a list of Unicode strings.
	TWideStringsEnumerator (see page 6856)	Enumerates through the strings of a TWideStrings (see page 6825) object.

Interfaces

	Name	Description
	IWideStringsAdapter (see page 6859)	This is class WideStrings.IWideStringsAdapter.

Structs, Records, Enums

	Name	Description
	TWideStringItem (see page 6861)	TWideStringItem defines a TWideStringList (see page 6808) entry.

Types

Name	Description
PWideStringItem (see page 6862)	PWideStringItem is a TWideStringItem (see page 6861) pointer.
PWideStringItemList (see page 6862)	PWideStringItemList is a TWideStringItemList (see page 6863) pointer.
TWideStringItemList (see page 6863)	TWideStringItemList defines an array of TWideStringItems.
TWideStringListSortCompare (see page 6863)	TWideStringListSortCompare is a type of callback function called used internally by the TWideStringList (see page 6808) class.

70.1 Classes

The following table lists classes in this documentation.

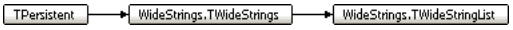
Classes

	Name	Description
	TWideStringList (see page 6808)	TWideStringList maintains a list of strings.
	TWideStrings (see page 6825)	TWideStrings is the base class for objects that represent a list of Unicode strings.
	TWideStringsEnumerator (see page 6856)	Enumerates through the strings of a TWideStrings (see page 6825) object.

70.1.1 TWideStringList Class

TWideStringList maintains a list of strings.

Class Hierarchy



Pascal

```
TWideStringList = class(TWideStrings);
```

C++

```
class TWideStringList : public TWideStrings;
```

File

WideStrings

Description

Use a string list object to store and manipulate a list of strings. TWideStringList implements the abstract properties and methods introduced by TWideStrings (see page 6825), and introduces new properties, events, and methods to:

Sort (see page 6821) the strings in the list;

Prohibit duplicate strings in sorted lists;

Respond to changes in the contents of the list;

Control whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.

See Also

TCollection

TList

Adding Graphical Objects to a String List

Adding Images to a String List

List Controls

Working with Lists

70.1.1.1 TWideStringList Members

The following tables list the members exposed by TWideStringList.

Protected Methods

	Name	Description
	AssignTo (see page 6832)	Copies the properties of an object to a destination object.
	Changed (see page 6813)	This is Changed, a member of class TWideStringList.
	Changing (see page 6814)	This is Changing, a member of class TWideStringList.
	CompareStrings (see page 6814)	Compares two strings.
	DefineProperties (see page 6834)	Reads and writes the Strings (see page 6853) property as if it were published.
	Error (see page 6836)	This is the overview for the Error method overload.
	ExtractName (see page 6837)	Returns the name portion of a string that is a name value pair.
	Get (see page 6817)	Returns a string given its index.
	GetCapacity (see page 6817)	Returns the currently allocated size of the list of strings.
	GetCount (see page 6817)	Returns the number of strings in the list
	GetObject (see page 6818)	Returns the object associated with the string at a specified index.
	GetTextStr (see page 6839)	Returns the value of the Text (see page 6854) property.
	InsertItem (see page 6819)	This is InsertItem, a member of class TWideStringList.
	Put (see page 6820)	Changes the value of the string with a specified index.
	PutObject (see page 6820)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6820)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 6846)	Sets the Text (see page 6854) property.
	SetUpdateState (see page 6821)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 6855)	Indicates the number of calls to BeginUpdate (see page 6832) that have not been matched by a call to EndUpdate (see page 6835).

Public Events

	Name	Description
	OnChange (see page 6824)	Occurs immediately after the list of strings changes.
	OnChanging (see page 6825)	Occurs immediately before the list of strings changes.

Public Methods

	Name	Description
	Add (see page 6812)	Adds a new string to the list.
	AddObject (see page 6813)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 6830)	This is the overview for the AddStrings method overload.
	Append (see page 6831)	Adds a string to the list.
	Assign (see page 6831)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeginUpdate (see page 6832)	Enables the TWideStrings (see page 6825) object to track when the list of strings is changing.
	Clear (see page 6814)	Deletes all the strings from the list.
	Create (see page 6810)	This is the overview for the Create constructor overload.
	CustomSort (see page 6815)	Sorts the strings in the list in a customized order.
	Delete (see page 6815)	Removes the string specified by the Index parameter.
	Destroy (see page 6811)	Destroys an instance of TWideStringList (see page 6808).
	EndUpdate (see page 6835)	Enables the TWideStrings (see page 6825) object to keep track of when the list of strings has finished changing.
	Equals (see page 6835)	Compares the list of strings to the list from another TWideStrings (see page 6825) object and returns true if the two lists match.
	Exchange (see page 6816)	Swaps the position of two strings in the list.

	Find (see page 6816)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	GetEnumerator (see page 6838)	Returns a TWideStrings (see page 6825) enumerator.
	GetText (see page 6839)	Allocates a text buffer and fills it with the value of the Text (see page 6854) property.
	IndexOf (see page 6818)	Returns the position of a string in the list.
	IndexOfName (see page 6840)	Returns the position of the first name-value pair with the specified name.
	IndexOfObject (see page 6841)	Returns the index of the first string in the list associated with a given object.
	Insert (see page 6819)	Inserts a string to the list at the position specified by `Index`.
	InsertObject (see page 6819)	Inserts a string into the list at the specified position, and associates it with an object.
	LoadFromFile (see page 6842)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 6842)	Fills the list with lines of text read from a stream.
	Move (see page 6843)	Changes the position of a string in the list.
	SaveToFile (see page 6844)	Saves the strings in the list to the specified file.
	SaveToStream (see page 6845)	Writes the value of the Text (see page 6854) property to a stream object.
	SetText (see page 6845)	Sets the Text (see page 6854) property.
	Sort (see page 6821)	Sorts the strings in the list in ascending order.

Public Properties

	Name	Description
	Capacity (see page 6847)	Indicates the number of strings the TWideStrings (see page 6825) object can hold.
	CaseSensitive (see page 6822)	Controls whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	CommaText (see page 6848)	Lists the strings in the TWideStrings (see page 6825) object in system data format (SDF).
R	Count (see page 6849)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 6849)	Represents all the strings in the TWideStrings (see page 6825) object as a single delimited string.
	Delimiter (see page 6850)	Specifies the delimiter used by the DelimitedText (see page 6849) property.
	Duplicates (see page 6822)	Specifies whether duplicate strings can be added to sorted lists.
	LineBreak (see page 6850)	Defines the line break character(s).
R	Names (see page 6850)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 6851)	Indicates the character used to separate names from values.
	Objects (see page 6851)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 6853) property.
	OwnsObjects (see page 6823)	This is OwnsObjects, a member of class TWideStringList.
	QuoteChar (see page 6852)	Specifies the quote character used by the DelimitedText (see page 6849) property.
	Sorted (see page 6823)	Specifies whether the strings in the list should be automatically sorted.
	StrictDelimiter (see page 6852)	Determines how the Delimiter (see page 6850) property is used.
	Strings (see page 6853)	References the strings in the list by their positions.
	StringsAdapter (see page 6854)	Implements an IWideStringsAdapter (see page 6859) interface for the TWideStrings (see page 6825) object.
	Text (see page 6854)	Lists the strings in the TWideStrings (see page 6825) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 6855)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 6856)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

70.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TWideStringList.Create () (see page 6811)	Constructs an object and initializes its data before the object is first used.
	TWideStringList.Create (Boolean) (see page 6811)	This is Create, a member of class TWideStringList.

70.1.1.2.1 TWideStringList.Create Constructor ()

Constructs an object and initializes its data before the object is first used.

Pascal

```
constructor Create; overload;
```

C++

```
__fastcall TWideStringList();
```

Description

System::TObject::Create constructs an object. The purpose, size, and behavior of objects differ greatly. The System::TObject::Create constructor defined by System::TObject allocates memory but does not initialize data.

Descendant objects usually define a constructor that creates the particular kind of object and initializes its data.

Note: If an exception escapes from a constructor, the object's destructor is called to clean up the failed instance.

See Also

System::TObject::Destroy (see page 3356)

System::TObject::Free (see page 3372)

70.1.1.2.2 TWideStringList.Create Constructor (Boolean)

Pascal

```
constructor Create(OwnsObjects: Boolean); overload;
```

C++

```
__fastcall TWideStringList(Boolean OwnsObjects);
```

Description

This is Create, a member of class TWideStringList.

70.1.1.3 TWideStringList.Destroy Destructor

Destroys an instance of TWideStringList (see page 6808).

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWideStringList();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TWideStringList (see page 6808) reference is not nil and only then calls Destroy.

Destroy frees the memory allocated to hold the list of strings and object references before calling the inherited destructor.

See Also

System::TObject::Free (see page 3372)

70.1.1.4 TWideStringList Methods

The methods of the TWideStringList class are listed here.

Protected Methods

	Name	Description
	Changed (see page 6813)	This is Changed, a member of class TWideStringList.
	Changing (see page 6814)	This is Changing, a member of class TWideStringList.
	CompareStrings (see page 6814)	Compares two strings.
	Get (see page 6817)	Returns a string given its index.
	GetCapacity (see page 6817)	Returns the currently allocated size of the list of strings.
	GetCount (see page 6817)	Returns the number of strings in the list
	GetObject (see page 6818)	Returns the object associated with the string at a specified index.
	InsertItem (see page 6819)	This is InsertItem, a member of class TWideStringList.
	Put (see page 6820)	Changes the value of the string with a specified index.
	PutObject (see page 6820)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6820)	Changes the amount of memory allocated to hold strings in the list.
	SetUpdateState (see page 6821)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 6812)	Adds a new string to the list.
	AddObject (see page 6813)	Adds a string to the list, and associates an object with the string.
	Clear (see page 6814)	Deletes all the strings from the list.
	CustomSort (see page 6815)	Sorts the strings in the list in a customized order.
	Delete (see page 6815)	Removes the string specified by the Index parameter.
	Exchange (see page 6816)	Swaps the position of two strings in the list.
	Find (see page 6816)	Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.
	IndexOf (see page 6818)	Returns the position of a string in the list.
	Insert (see page 6819)	Inserts a string to the list at the position specified by `Index`.
	InsertObject (see page 6819)	Inserts a string into the list at the specified position, and associates it with an object.
	Sort (see page 6821)	Sorts the strings in the list in ascending order.

70.1.1.4.1 TWideStringList.Add Method

Adds a new string to the list.

Pascal

```
function Add(const S: WideString): Integer; override;
```

C++

```
virtual __fastcall int Add(const BSTR S);
```

Description

Call Add to add the string S to the list. If the list is sorted, S is added to the appropriate position in the sort order. If the list is not sorted, S is added to the end of the list. Add returns the position of the item in the list, where the first item in the list has a value of 0.

Note: For sorted lists, Add will raise an EListError exception if the string S already appears in the list and Duplicates (see page 6822) is set to dupError. If Duplicates (see page 6822) is set to dupIgnore, trying to add a duplicate string causes Add to return the index of the existing entry.

See Also

AddStrings (see page 6830)

Append (see page 6831)

Delete (see page 6815)

Duplicates (see page 6822)

Insert (see page 6819)

Sorted (see page 6823)

70.1.1.4.2 TWideStringList.AddObject Method

Adds a string to the list, and associates an object with the string.

Pascal

```
function AddObject(const S: WideString; AObject: TObject): Integer; override;
```

C++

```
virtual __fastcall int AddObject(const BSTR S, TObject * AObject);
```

Description

Call AddObject to add a string and its associated object to the list. AddObject returns the index of the new string and object.

Note: The TWideStringList (see page 6808) object does not own the objects you add this way. Objects (see page 6851) added to the TWideStringList (see page 6808) object still exist even if the TWideStringList (see page 6808) instance is destroyed. They must be explicitly destroyed by the application.

Note: For sorted lists, AddObject raises an EListError exception if the string S already appears in the list and Duplicates (see page 6822) is set to dupError. If Duplicates (see page 6822) is set to dupIgnore, trying to add a duplicate string causes AddObject to return the index of the existing entry.

See Also

Add (see page 6812)

AddStrings (see page 6830)

70.1.1.4.3 TWideStringList.Changed Method

Pascal

```
procedure Changed; virtual;
```

C++

```
virtual __fastcall Changed();
```

Description

This is Changed, a member of class TWideStringList.

70.1.1.4.4 TWideStringList.Changing Method

Pascal

```
procedure Changing; virtual;
```

C++

```
virtual __fastcall Changing();
```

Description

This is Changing, a member of class TWideStringList.

70.1.1.4.5 TWideStringList.Clear Method

Deletes all the strings from the list.

Pascal

```
procedure Clear; override;
```

C++

```
virtual __fastcall Clear();
```

Description

Call clear to empty the list of strings. All references to associated objects are also removed. However, the objects themselves are not freed.

See Also

Delete (see page 6815)

Objects (see page 6851)

Strings (see page 6853)

70.1.1.4.6 TWideStringList.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: WideString; const S2: WideString): Integer; override;
```

C++

```
virtual __fastcall int CompareStrings(const BSTR S1, const BSTR S2);
```

Description

TWideStrings uses CompareStrings internally to compare the values of strings that appear in the list. For example, the IndexOf and IndexOfName methods use CompareStrings to compare a specified string with the strings in the list.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 == S2, and a value greater than 0 if S1 > S2.

As implemented in TWideStrings, CompareStrings uses the global WideCompareText function, which compares strings case insensitively. Some descendant classes override this method to change the way strings are compared (for example, to introduce case sensitivity).

See Also

WideCompareText (see page 4313)

IndexOf (see page 6840)

IndexOfName (see page 6840)

70.1.1.4.7 TWideStringList.CustomSort Method

Sorts the strings in the list in a customized order.

Pascal

```
procedure CustomSort(Compare: TWideStringListSortCompare); virtual;
```

C++

```
virtual __fastcall CustomSort(TWideStringListSortCompare Compare);
```

Description

Use CustomSort to sort the strings in the list, where the sort order is defined by the Compare parameter.

Supply a value for the Compare function that compares two strings in the string list. The List parameter provides access to the string list, while the Index1 and Index2 parameters identify the strings to be compared.

Do not pass nil (Delphi) or NULL (C++) as the value of the Compare parameter.

Note: You must explicitly call the CustomSort method. Setting the Sorted (see page 6823) property only sorts strings using ANSI (Windows) or UTF-8 (Linux) order, as implemented in the Sort (see page 6821) method.

See Also

Sort (see page 6821)

Strings (see page 6853)

Sorted (see page 6823)

70.1.1.4.8 TWideStringList.Delete Method

Removes the string specified by the Index parameter.

Pascal

```
procedure Delete(Index: Integer); override;
```

C++

```
virtual __fastcall Delete(int Index);
```

Description

Call Delete to remove a single string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

Add (see page 6812)

AddObject (see page 6813)

Clear (see page 6814)

IndexOf (see page 6818)

InsertObject (see page 6819)

Objects (see page 6851)

Strings (see page 6853)

70.1.1.4.9 TWideStringList.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); override;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the index of the object as well.

Warning: Do not call Exchange on a sorted list except to swap two identical strings with different associated objects. Exchange does not check whether the list is sorted, and can destroy the sort order of a sorted list.

See Also

Move (see page 6843)

Objects (see page 6851)

Sorted (see page 6823)

Strings (see page 6853)

70.1.1.4.10 TWideStringList.Find Method

Locates the index for a string in a sorted list and indicates whether a string with that value already exists in the list.

Pascal

```
function Find(const S: WideString; var Index: Integer): Boolean; virtual;
```

C++

```
virtual __fastcall Boolean Find(const BSTR S, int Index);
```

Description

Use Find to obtain the index in a sorted list where the string S should be added. If the string S, or a string that differs from S only in case when CaseSensitive (see page 6822) is false, already exists in the list, Find returns true. If the list does not contain a string that matches S, Find returns false. The index where S should go is returned in the Index parameter. The value of Index is zero-based, where the first string has the index 0, the second string has the index 1, and so on.

Note: Only use Find with sorted lists. For unsorted lists, use the IndexOf (see page 6818) method instead.

See Also

Duplicates (see page 6822)

IndexOf (see page 6818)

Sorted (see page 6823)

Strings (see page 6853)

70.1.1.4.11 TWideStringList.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): WideString; override;
```

C++

```
virtual __fastcall BSTR Get(int Index);
```

Description

Get is the protected read implementation of the Strings property.

In TWideStrings Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings (see page 6853)

70.1.1.4.12 TWideStringList.GetCapacity Method

Returns the currently allocated size of the list of strings.

Pascal

```
function GetCapacity: Integer; override;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity property. In TWideStrings, GetCapacity returns the value of the Count property. Descendants of TWideStrings can override this property to let a string list allocate memory for entries that have not been added to the list.

See Also

Capacity (see page 6847)

Count (see page 6849)

70.1.1.4.13 TWideStringList.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; override;
```

C++

```
virtual __fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count property.

In TWideStrings GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count (see page 6849)

70.1.1.4.14 TWideStringList.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; override;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects property.

Index is the index of the string with which the object is associated.

In TWideStrings, GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects (see page 6851)

70.1.1.4.15 TWideStringList.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: WideString): Integer; override;
```

C++

```
virtual __fastcall int IndexOf(const BSTR S);
```

Description

Call IndexOf to obtain the position of the first occurrence of a string that matches S. A string matches S if it is identical to S or, if CaseSensitive (see page 6822) is false, if it differs only in case.

IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string does not have a match in the string list, IndexOf returns -1.

See Also

Find (see page 6816)

IndexOfName (see page 6840)

IndexOfObject (see page 6841)

Strings (see page 6853)

70.1.1.4.16 TWideStringList.Insert Method

Inserts a string to the list at the position specified by `Index`.

Pascal

```
procedure Insert(Index: Integer; const S: WideString); override;
```

C++

```
virtual __fastcall Insert(int Index, const BSTR S);
```

Description

Call Insert to add the string S to the list at the position specified by Index. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

If the string has an associated object, use the InsertObject (see page 6819) method instead.

Note: If the list is sorted, calling Insert or InsertObject (see page 6819) will raise an EListError exception. Use Add (see page 6812) or AddObject (see page 6813) with sorted lists.

See Also

Add (see page 6812)

AddObject (see page 6813)

EListError

InsertObject (see page 6819)

Sorted (see page 6823)

Strings (see page 6853)

70.1.1.4.17 TWideStringList.InsertItem Method

Pascal

```
procedure InsertItem(Index: Integer; const S: WideString; AObject: TObject); virtual;
```

C++

```
virtual __fastcall InsertItem(int Index, const BSTR S, TObject * AObject);
```

Description

This is InsertItem, a member of class TWideStringList.

70.1.1.4.18 TWideStringList.InsertObject Method

Inserts a string into the list at the specified position, and associates it with an object.

Pascal

```
procedure InsertObject(Index: Integer; const S: WideString; AObject: TObject); override;
```

C++

```
virtual __fastcall InsertObject(int Index, const BSTR S, TObject * AObject);
```

Description

Call InsertObject to insert the string S into the list at the position identified by Index, and associate it with the object AObject. If Index is 0, the string is inserted at the beginning of the list. If Index is 1, the string is put in the second position of the list, and so on.

See Also

Insert (see page 6819)

Objects (see page 6851)

70.1.1.4.19 TWideStringList.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: WideString); override;
```

C++

```
virtual __fastcall Put(int Index, const BSTR S);
```

Description

Put is the protected write implementation of the Strings property.

Put changes the value of the string with the index specified by `Index` to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 6853)

Objects (see page 6851)

70.1.1.4.20 TWideStringList.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); override;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects property.

As implemented in TWideStrings, PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects (see page 6851)

70.1.1.4.21 TWideStringList.SetCapacity Method

Changes the amount of memory allocated to hold strings in the list.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); override;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected read implementation of the Capacity property.

Integer is the number of strings the list can hold after the capacity has changed.

In TWideStrings, SetCapacity does nothing. Descendant classes override this method to change the number of strings that the list can hold. For descendant classes that implement SetCapacity, assigning a value smaller than Count removes strings from the end of the list. Assigning a value greater than Count allocates space for more strings to be added.

See Also

Capacity (see page 6847)

Count (see page 6849)

70.1.1.4.22 TWideStringList.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); override;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate method is first called and the TWideStrings object is not already in the middle of an update, TWideStrings calls SetUpdateState internally, with Updating set to true. When the EndUpdate method is called and it cancels out the last unmatched call to BeginUpdate, TWideStrings calls SetUpdateState internally, with Updating set to false.

As implemented in TWideStrings, SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate (see page 6832)

EndUpdate (see page 6835)

70.1.1.4.23 TWideStringList.Sort Method

Sorts the strings in the list in ascending order.

Pascal

```
procedure Sort; virtual;
```

C++

```
virtual __fastcall Sort();
```

Description

Call Sort to sort the strings in a list that has the Sorted (see page 6823) property set to false. String lists with the Sorted (see page 6823) property set to true are automatically sorted.

Note: Sort uses WideCompareStr (see page 4313) to sort the strings when CaseSensitive (see page 6822) is true and WideCompareText (see page 4313) when CaseSensitive (see page 6822) is false. To provide your own comparison operator instead, use the CustomSort (see page 6815) method.

See Also

CustomSort (see page 6815)

Sorted (see page 6823)

Strings (see page 6853)

WideCompareStr (see page 4313)

WideCompareText (see page 4313)

70.1.1.5 TWideStringList Properties

The properties of the TWideStringList class are listed here.

Public Properties

	Name	Description
	CaseSensitive (see page 6822)	Controls whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.
	Duplicates (see page 6822)	Specifies whether duplicate strings can be added to sorted lists.
	OwnsObjects (see page 6823)	This is OwnsObjects, a member of class TWideStringList.
	Sorted (see page 6823)	Specifies whether the strings in the list should be automatically sorted.

70.1.1.5.1 TWideStringList.CaseSensitive Property

Controls whether strings are located, sorted, and identified as duplicates in a case-sensitive or case-insensitive manner.

Pascal

```
property CaseSensitive: Boolean;
```

C++

```
__property Boolean CaseSensitive;
```

Description

Use CaseSensitive to indicate whether strings in the list should be compared in a case-sensitive or case-insensitive manner. Set CaseSensitive to true to make the string list locate, check for duplicates, and sort its strings in a case-sensitive manner. Set CaseSensitive to false to make the string list perform these operations case-insensitively.

See Also

CustomSort (see page 6815)

Sort (see page 6821)

70.1.1.5.2 TWideStringList.Duplicates Property

Specifies whether duplicate strings can be added to sorted lists.

Pascal

```
property Duplicates: TDuplicates;
```

C++

```
__property TDuplicates Duplicates;
```

Description

Set Duplicates to specify what should happen when an attempt is made to add a duplicate string to a sorted list. The CaseSensitive (see page 6822) property controls whether two strings are considered duplicates if they are identical except for differences in case.

The value of Duplicates should be one of the following.

Value	Meaning
dupIgnore	Ignore attempts to add duplicate strings to the list.
dupError	raise an EStringListError exception when an attempt is made to add duplicate strings to the sorted list.
dupAccept	Permit duplicate strings in the sorted list.

Set Duplicates before adding any strings to the list. Setting Duplicates to dupIgnore or dupError does nothing about duplicate strings that are already in the list.

Note: Duplicates does nothing if the list is not sorted.

See Also

Add (see page 6812)

EStringListError

Sorted (see page 6823)

CaseSensitive (see page 6822)

70.1.1.5.3 TWideStringList.OwnsObjects Property

Pascal

```
property OwnsObjects: Boolean;
```

C++

```
__property Boolean OwnsObjects;
```

Description

This is OwnsObjects, a member of class TWideStringList.

70.1.1.5.4 TWideStringList.Sorted Property

Specifies whether the strings in the list should be automatically sorted.

Pascal

```
property Sorted: Boolean;
```

C++

```
__property Boolean Sorted;
```

Description

Set Sorted to true to cause the strings in the list to be automatically sorted in ascending order. Set Sorted to false to allow strings to remain where they are inserted. When Sorted is false, the strings in the list can be put in ascending order at any time by calling the Sort (see page 6821) method.

When Sorted is true, do not use Insert (see page 6819) to add strings to the list. Instead, use Add (see page 6812), which will insert the new strings in the appropriate position. When Sorted is false, use Insert (see page 6819) to add strings to an arbitrary position in the list, or Add (see page 6812) to add strings to the end of the list.

Note: The CaseSensitive (see page 6822) property controls whether the strings in the list are sorted based on a case-sensitive or case-insensitive comparison. The sort order takes into account the locale of the system on which the application is running.

See Also

Add (see page 6812)

Duplicates (see page 6822)

Insert (see page 6819)

Sort (see page 6821)

WideCompareStr (see page 4313)

CaseSensitive (see page 6822)

70.1.1.6 TWideStringList Events

The events of the TWideStringList class are listed here.

Public Events

	Name	Description
	OnChange (see page 6824)	Occurs immediately after the list of strings changes.
	OnChanging (see page 6825)	Occurs immediately before the list of strings changes.

70.1.1.6.1 TWideStringList.OnChange Event

Occurs immediately after the list of strings changes.

Pascal

```
property OnChange: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChange;
```

Description

Write (see page 3707) an OnChange event handler to respond to changes in the list of strings. For example, if the string list is associated with a control, the OnChange event handler could tell the control to repaint itself whenever the content of the list changes.

Whenever strings in the list are added, deleted, moved, or modified, the following events take place:

1. An OnChanging (see page 6825) event occurs before the change.
2. The strings are added, deleted, moved, or modified.

3. An OnChange event occurs.

   Note: OnChange occurs for every change made to the list, regardless of whether the application calls BeginUpdate (see page 6832) and EndUpdate (see page 6835) around a series of changes.

See Also

BeginUpdate (see page 6832)

EndUpdate (see page 6835)

OnChanging (see page 6825)

70.1.1.6.2 TWideStringList.OnChanging Event

Occurs immediately before the list of strings changes.

Pascal

```
property OnChanging: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnChanging;
```

Description

Write (see page 3707) an OnChanging event handler to prepare for changes in the list of strings. For example, if the string list is associated with a control, the OnChanging event handler could tell the control to disable repaints until the OnChange (see page 6824) event when the list has finished changing.

Whenever strings in the list are added, deleted, moved, or modified, the following events take place:

1. An OnChanging event occurs.
2. The strings are added, deleted, moved, or modified.
3. An OnChange (see page 6824) event occurs after the changes are complete.

   Note: OnChanging occurs for every change made to the list, regardless of whether the application calls BeginUpdate (see page 6832) and EndUpdate (see page 6835) around a series of changes.

See Also

BeginUpdate (see page 6832)

EndUpdate (see page 6835)

OnChange (see page 6824)

70.1.2 TWideStrings Class

TWideStrings is the base class for objects that represent a list of Unicode strings.

Class Hierarchy

Pascal

```
TWideStrings = class(TPersistent);
```

C++

```
class TWideStrings : public TPersistent;
```

File

WideStrings

Description

Derive a class from TWideStrings to store and manipulate a list of strings. TWideStrings contains abstract or, in C++ terminology, pure virtual methods and should not be directly instantiated.

Descendants of TWideStrings can represent several individual strings, such as the individual lines that appear in a list box. Some objects use descendants of TWideStrings to represent one long body of text so that it can be manipulated in smaller chunks.

TWideStrings introduces many properties and methods to:

- Add (see page 6829) or delete strings at specified positions in the list.
- Rearrange the strings in the list.
- Access the string at a particular location.
- Read (see page 3634) the strings from or write the strings to a file or stream.
- Associate an object with each string in the list.
- Store and retrieve strings as name-value pairs.

For an explanation of name-value pairs, refer to the NameValueSeparator (see page 6851) property.

See Also

TWideStringList (see page 6808)

TStrings

TStringList

TList

THashedStringList

TWideStringsEnumerator (see page 6856)

70.1.2.1 TWideStrings Members

The following tables list the members exposed by TWideStrings.

Protected Methods

	Name	Description
	AssignTo (see page 6832)	Copies the properties of an object to a destination object.
	CompareStrings (see page 6833)	Compares two strings.
	DefineProperties (see page 6834)	Reads and writes the Strings (see page 6853) property as if it were published.
	Error (see page 6836)	This is the overview for the Error method overload.
	ExtractName (see page 6837)	Returns the name portion of a string that is a name value pair.
	Get (see page 6837)	Returns a string given its index.
	GetCapacity (see page 6837)	Returns the currently allocated size of the list of strings.

	GetCount (see page 6838)	Returns the number of strings in the list
	GetObject (see page 6838)	Returns the object associated with the string at a specified index.
	GetTextStr (see page 6839)	Returns the value of the Text (see page 6854) property.
	Put (see page 6843)	Changes the value of the string with a specified index.
	PutObject (see page 6844)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6845)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 6846)	Sets the Text (see page 6854) property.
	SetUpdateState (see page 6846)	Performs internal adjustments before or after a series of updates.

Protected Properties

	Name	Description
	UpdateCount (see page 6855)	Indicates the number of calls to BeginUpdate (see page 6832) that have not been matched by a call to EndUpdate (see page 6835).

Public Methods

	Name	Description
	Add (see page 6829)	Adds a string at the end of the list.
	AddObject (see page 6830)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 6830)	This is the overview for the AddStrings method overload.
	Append (see page 6831)	Adds a string to the list.
	Assign (see page 6831)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeginUpdate (see page 6832)	Enables the TWideStrings (see page 6825) object to track when the list of strings is changing.
	Clear (see page 6833)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Delete (see page 6834)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified Unicode string from the list.
	Destroy (see page 6828)	Destroys the TPersistent instance and frees its memory.
	EndUpdate (see page 6835)	Enables the TWideStrings (see page 6825) object to keep track of when the list of strings has finished changing.
	Equals (see page 6835)	Compares the list of strings to the list from another TWideStrings (see page 6825) object and returns true if the two lists match.
	Exchange (see page 6836)	Swaps the position of two strings in the list.
	GetEnumerator (see page 6838)	Returns a TWideStrings (see page 6825) enumerator.
	GetText (see page 6839)	Allocates a text buffer and fills it with the value of the Text (see page 6854) property.
	IndexOf (see page 6840)	Returns the position of a string in the list.
	IndexOfName (see page 6840)	Returns the position of the first name-value pair with the specified name.
	IndexOfObject (see page 6841)	Returns the index of the first string in the list associated with a given object.
	Insert (see page 6841)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
	InsertObject (see page 6842)	Inserts a string into the list at the specified position, and associates it with an object.
	LoadFromFile (see page 6842)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 6842)	Fills the list with lines of text read from a stream.
	Move (see page 6843)	Changes the position of a string in the list.
	SaveToFile (see page 6844)	Saves the strings in the list to the specified file.
	SaveToStream (see page 6845)	Writes the value of the Text (see page 6854) property to a stream object.
	SetText (see page 6845)	Sets the Text (see page 6854) property.

Public Properties

	Name	Description
	Capacity (see page 6847)	Indicates the number of strings the TWideStrings (see page 6825) object can hold.
	CommaText (see page 6848)	Lists the strings in the TWideStrings (see page 6825) object in system data format (SDF).
	Count (see page 6849)	Introduces an abstract property to represent the number of strings in the list.

	DelimitedText (see page 6849)	Represents all the strings in the TWideStrings (see page 6825) object as a single delimited string.
	Delimiter (see page 6850)	Specifies the delimiter used by the DelimitedText (see page 6849) property.
	LineBreak (see page 6850)	Defines the line break character(s).
R	Names (see page 6850)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 6851)	Indicates the character used to separate names from values.
	Objects (see page 6851)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 6853) property.
	QuoteChar (see page 6852)	Specifies the quote character used by the DelimitedText (see page 6849) property.
	StrictDelimiter (see page 6852)	Determines how the Delimiter (see page 6850) property is used.
	Strings (see page 6853)	References the strings in the list by their positions.
	StringsAdapter (see page 6854)	Implements an IWideStringsAdapter (see page 6859) interface for the TWideStrings (see page 6825) object.
	Text (see page 6854)	Lists the strings in the TWideStrings (see page 6825) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 6855)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 6856)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

70.1.2.2 TWideStrings.Destroy Destructor

Destroys the TPersistent instance and frees its memory.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWideStrings();
```

Description

Do not call Destroy directly. Call Free instead. Free checks that the object reference is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

System::TObject::Destroy (see page 3356)

70.1.2.3 TWideStrings Methods

The methods of the TWideStrings class are listed here.

Protected Methods

	Name	Description
V	AssignTo (see page 6832)	Copies the properties of an object to a destination object.
V	CompareStrings (see page 6833)	Compares two strings.
V	DefineProperties (see page 6834)	Reads and writes the Strings (see page 6853) property as if it were published.
	Error (see page 6836)	This is the overview for the Error method overload.
	ExtractName (see page 6837)	Returns the name portion of a string that is a name value pair.
A	Get (see page 6837)	Returns a string given its index.
V	GetCapacity (see page 6837)	Returns the currently allocated size of the list of strings.
A	GetCount (see page 6838)	Returns the number of strings in the list
V	GetObject (see page 6838)	Returns the object associated with the string at a specified index.
V	GetTextStr (see page 6839)	Returns the value of the Text (see page 6854) property.

	Put (see page 6843)	Changes the value of the string with a specified index.
	PutObject (see page 6844)	Changes the object associated with the string at a specified index.
	SetCapacity (see page 6845)	Changes the amount of memory allocated to hold strings in the list.
	SetTextStr (see page 6846)	Sets the Text (see page 6854) property.
	SetUpdateState (see page 6846)	Performs internal adjustments before or after a series of updates.

Public Methods

	Name	Description
	Add (see page 6829)	Adds a string at the end of the list.
	AddObject (see page 6830)	Adds a string to the list, and associates an object with the string.
	AddStrings (see page 6830)	This is the overview for the AddStrings method overload.
	Append (see page 6831)	Adds a string to the list.
	Assign (see page 6831)	Sets the strings in the list, and possibly associated objects, from a source object.
	BeginUpdate (see page 6832)	Enables the TWideStrings (see page 6825) object to track when the list of strings is changing.
	Clear (see page 6833)	Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.
	Delete (see page 6834)	Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified Unicode string from the list.
	EndUpdate (see page 6835)	Enables the TWideStrings (see page 6825) object to keep track of when the list of strings has finished changing.
	Equals (see page 6835)	Compares the list of strings to the list from another TWideStrings (see page 6825) object and returns true if the two lists match.
	Exchange (see page 6836)	Swaps the position of two strings in the list.
	GetEnumerator (see page 6838)	Returns a TWideStrings (see page 6825) enumerator.
	GetText (see page 6839)	Allocates a text buffer and fills it with the value of the Text (see page 6854) property.
	IndexOf (see page 6840)	Returns the position of a string in the list.
	IndexOfName (see page 6840)	Returns the position of the first name-value pair with the specified name.
	IndexOfObject (see page 6841)	Returns the index of the first string in the list associated with a given object.
	Insert (see page 6841)	Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.
	InsertObject (see page 6842)	Inserts a string into the list at the specified position, and associates it with an object.
	LoadFromFile (see page 6842)	Fills the list with the lines of text in a specified file.
	LoadFromStream (see page 6842)	Fills the list with lines of text read from a stream.
	Move (see page 6843)	Changes the position of a string in the list.
	SaveToFile (see page 6844)	Saves the strings in the list to the specified file.
	SaveToStream (see page 6845)	Writes the value of the Text (see page 6854) property to a stream object.
	SetText (see page 6845)	Sets the Text (see page 6854) property.

70.1.2.3.1 TWideStrings.Add Method

Adds a string at the end of the list.

Pascal

```
function Add(const S: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int Add(const BSTR S);
```

Description

Call Add to add a string to the end of the list. Add returns the index of the new string.

See Also

AddStrings (see page 6830)

Append (see page 6831)

Count (see page 6849)

IndexOf (see page 6840)

Insert (see page 6841)

70.1.2.3.2 TWideStrings.AddObject Method

Adds a string to the list, and associates an object with the string.

Pascal

```
function AddObject(const S: WideString; AObject: TObject): Integer; virtual;
```

C++

```
virtual __fastcall int AddObject(const BSTR S, TObject * AObject);
```

Description

Call AddObject to add a string and its associated object to the list. AddObject returns the index of the new string and object.

Note: The TWideStrings (see page 6825) object does not own the objects you add this way. Objects (see page 6851) added to the TWideStrings (see page 6825) object still exist even if the TWideStrings (see page 6825) instance is destroyed. They must be explicitly destroyed by the application.

See Also

Add (see page 6829)

AddStrings (see page 6830)

Count (see page 6849)

IndexOfObject (see page 6841)

Objects (see page 6851)

70.1.2.3.3 AddStrings Method

This is the overview for the AddStrings method overload.

Overload List

	Name	Description
	TWideStrings.AddStrings (TStrings) (see page 6830)	Adds a group of strings to the list.
	TWideStrings.AddStrings (TWideStrings) (see page 6830)	Adds a group of strings to the list.

70.1.2.3.3.1 TWideStrings.AddStrings Method (TStrings)

Adds a group of strings to the list.

Pascal

```
procedure AddStrings(Strings: TStrings); virtual; overload;
procedure AddStrings(Strings: TWideStrings); virtual; overload;
```

C++

```
virtual __fastcall AddStrings(TStrings Strings);
virtual __fastcall AddStrings(TWideStrings Strings);
```

Description

Call AddStrings to add the strings from another TWideStrings (see page 6825) object to the list. If both the source and destination TWideStrings (see page 6825) objects support objects associated with their strings, references to the associated objects will be added as well.

See Also

Add (see page 6829)

AddObject (see page 6830)

Assign (see page 6831)

Count (see page 6849)

Objects (see page 6851)

70.1.2.3.4 TWideStrings.Append Method

Adds a string to the list.

Pascal

```
procedure Append(const S: WideString);
```

C++

```
__fastcall Append(const BSTR S);
```

Description

Append is the same as the Add (see page 6829) method, except that it does not return a value. Use Append when there is no need to know the index of the string after it has been added, or with descendants of TWideStrings (see page 6825) for which the index returned is not meaningful.

For example, the TWideStrings (see page 6825) descendant used by memo objects uses an index to determine where to insert a string, but the inserted string does not necessarily end up as a single string in the list. Part of the inserted text may become part of the previous string, and part may be broken off into a subsequent string. The index returned by Add (see page 6829) is not meaningful in this case.

See Also

Add (see page 6829)

70.1.2.3.5 TWideStrings.Assign Method

Sets the strings in the list, and possibly associated objects, from a source object.

Pascal

```
procedure Assign(Source: TPersistent); override;
```

C++

```
virtual __fastcall Assign(TPersistent * Source);
```

Description

Use Assign to set the value of the TWideStrings (see page 6825) object from another object. If Source is of type TWideStrings

(see page 6825), the list is set to the list of the source TWideStrings (see page 6825) object, and if associated objects are supported, any associated objects are copied from the Source as well.

If Source is not of type TWideStrings (see page 6825), the inherited Assign will set the value of the list from any object that supports TWideStrings (see page 6825) in its AssignTo (see page 6832) method.

See Also

AddStrings (see page 6830)

Objects (see page 6851)

Strings (see page 6853)

AssignTo

70.1.2.3.6 TWideStrings.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3919)

70.1.2.3.7 TWideStrings.BeginUpdate Method

Enables the TWideStrings (see page 6825) object to track when the list of strings is changing.

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

BeginUpdate is called automatically by any property or method that changes the list of strings. Once the changes are complete, the property or method calls EndUpdate (see page 6835). Call BeginUpdate before directly modifying the strings in the list, and EndUpdate (see page 6835) after. When implementing properties or methods that change the list in descendants of TWideStrings (see page 6825), call BeginUpdate before the changes are made, and EndUpdate (see page 6835) when the changes are complete.

TWideStrings (see page 6825) simply keeps track of when the list of strings is being changed. Some descendants of TWideStrings (see page 6825) use this information to perform certain actions, such as telling a control to repaint, when updates are complete.

See Also

EndUpdate (see page 6835)

UpdateCount (see page 6855)

70.1.2.3.8 TWideStrings.Clear Method

Introduces an abstract (Delphi) or pure virtual (C++) method to empty the list and any associated objects.

Pascal

```
procedure Clear; virtual; abstract;
```

C++

```
virtual __fastcall Clear() = 0;
```

Description

Descendants of TWideStrings (see page 6825) implement a Clear method to delete all the strings in the list, and to remove any references to associated objects.

See Also

Delete (see page 6834)

Objects (see page 6851)

Strings (see page 6853)

70.1.2.3.9 TWideStrings.CompareStrings Method

Compares two strings.

Pascal

```
function CompareStrings(const S1: WideString; const S2: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int CompareStrings(const BSTR S1, const BSTR S2);
```

Description

TWideStrings (see page 6825) uses CompareStrings internally to compare the values of strings that appear in the list. For example, the IndexOf (see page 6840) and IndexOfName (see page 6840) methods use CompareStrings to compare a specified string with the strings in the list.

S1 and S2 are the strings to compare.

CompareStrings returns a value less than 0 if S1 < S2, 0 if S1 == S2, and a value greater than 0 if S1 > S2.

As implemented in TWideStrings (see page 6825), CompareStrings uses the global WideCompareText (see page 4313) function, which compares strings case insensitively. Some descendant classes override this method to change the way strings are compared (for example, to introduce case sensitivity).

See Also

WideCompareText (see page 4313)

IndexOf (see page 6840)

IndexOfName (see page 6840)

70.1.2.3.10 TWideStrings.DefineProperties Method

Reads and writes the Strings (see page 6853) property as if it were published.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TWideStrings (see page 6825) overrides DefineProperties so that the strings in the list can be loaded and saved with a form file as if the Strings (see page 6853) property were published.

See Also

DefineProperties

Strings (see page 6853)

70.1.2.3.11 TWideStrings.Delete Method

Introduces an abstract (Delphi) or pure virtual (C++) method to delete a specified Unicode string from the list.

Pascal

```
procedure Delete(Index: Integer); virtual; abstract;
```

C++

```
virtual __fastcall Delete(int Index) = 0;
```

Description

Descendants of TWideStrings (see page 6825) implement a Delete method to remove a specified string from the list. If an object is associated with the string, the reference to the object is removed as well. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

See Also

AddObject (see page 6830)

Clear (see page 6833)

IndexOf (see page 6840)

InsertObject (see page 6842)

Objects (see page 6851)

Strings (see page 6853)

70.1.2.3.12 TWideStrings.EndUpdate Method

Enables the TWideStrings (see page 6825) object to keep track of when the list of strings has finished changing.

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

EndUpdate is called automatically by any property or method that changes the list of strings. Call BeginUpdate (see page 6832) before directly modifying the strings in the list, and EndUpdate after. When implementing properties or methods that change the list in descendants of TWideStrings (see page 6825), call BeginUpdate (see page 6832) before the changes are made, and EndUpdate when the changes are complete.

TWideStrings (see page 6825) simply keeps track of when the list of strings is being changed. Some descendants of TWideStrings (see page 6825) use this information to perform certain actions, such as telling a control to repaint, when updates are complete.

See Also

BeginUpdate (see page 6832)

UpdateCount (see page 6855)

70.1.2.3.13 TWideStrings.Equals Method

Compares the list of strings to the list from another TWideStrings (see page 6825) object and returns true if the two lists match.

Pascal

```
function Equals(Strings: TWideStrings): Boolean; reintroduce;
```

C++

```
__fastcall Boolean Equals(TWideStrings Strings);
```

Description

Call Equals to compare the lists in two TWideStrings (see page 6825) objects. Equals compares only the strings, not any references to associated objects. Equals returns true if the lists for both TWideStrings (see page 6825) objects have the same number of strings and the strings in each list match. Equals returns false if the lists are different in length, if they contain different strings, or if the order of the strings in the two lists differ.

Note: The Equals method uses an overridden Get (see page 6837) method to perform the compare, thus it does not provide a context-insensitive option.

See Also

Count (see page 6849)

Strings (see page 6853)

CompareStrings (see page 6833)

70.1.2.3.14 Error Method

This is the overview for the Error method overload.

Overload List

	Name	Description
	TWideStrings.Error (PResStringRec, Integer) (see page 6836)	Raises an EStringListError exception.
	TWideStrings.Error (WideString, Integer) (see page 6836)	Raises an EStringListError exception.

70.1.2.3.14.1 TWideStrings.Error Method (PResStringRec, Integer)

Raises an EStringListError exception.

Pascal

```
procedure Error(const Msg: WideString; Data: Integer); overload;
procedure Error(Msg: PResStringRec; Data: Integer); overload;
```

C++

```
__fastcall Error(const BSTR Msg, int Data);
__fastcall Error(PResStringRec Msg, int Data);
```

Description

TWideStrings (see page 6825) calls Error internally to raise an EStringListError exception when it encounters a problem.

Msg specifies the string with a single format specifier for an integer, that appears in the exception message box. It can be either a string, or a pointer to a record structure that indicates the module and resource identifier for a string.

Data is an integer value that is inserted into Msg.

See Also

EStringListError

70.1.2.3.15 TWideStrings.Exchange Method

Swaps the position of two strings in the list.

Pascal

```
procedure Exchange(Index1: Integer; Index2: Integer); virtual;
```

C++

```
virtual __fastcall Exchange(int Index1, int Index2);
```

Description

Call Exchange to rearrange the strings in the list. The strings are specified by their index values in the Index1 and Index2 parameters. Indexes are zero-based, so the first string in the list has an index value of 0, the second has an index value of 1, and so on.

If either string has an associated object, Exchange changes the position of the object as well.

See Also

Move (see page 6843)

Objects (see page 6851)

Strings (see page 6853)

70.1.2.3.16 TWideStrings.ExtractName Method

Returns the name portion of a string that is a name value pair.

Pascal

```
function ExtractName(const S: WideString): WideString;
```

C++

```
__fastcall BSTR ExtractName(const BSTR S);
```

Description

TWideStrings (see page 6825) calls ExtractName internally to parse strings that are name-value pairs and return the name portion.

S is the string to parse.

If S is a name-value pair, ExtractName returns the name portion. Otherwise, ExtractName returns an empty string.

See Also

Names (see page 6850)

70.1.2.3.17 TWideStrings.Get Method

Returns a string given its index.

Pascal

```
function Get(Index: Integer): WideString; virtual; abstract;
```

C++

```
virtual __fastcall BSTR Get(int Index) = 0;
```

Description

Get is the protected read implementation of the Strings (see page 6853) property.

In TWideStrings (see page 6825) Get is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the string with the specified index.

See Also

Strings (see page 6853)

70.1.2.3.18 TWideStrings.GetCapacity Method

Returns the currently allocated size of the list of strings.

Pascal

```
function GetCapacity: Integer; virtual;
```

C++

```
virtual __fastcall int GetCapacity();
```

Description

GetCapacity is the protected read implementation of the Capacity (see page 6847) property. In TWideStrings (see page 6825), GetCapacity returns the value of the Count (see page 6849) property. Descendants of TWideStrings (see page 6825) can override this property to let a string list allocate memory for entries that have not been added to the list.

See Also

Capacity (see page 6847)

Count (see page 6849)

70.1.2.3.19 TWideStrings.GetCount Method

Returns the number of strings in the list

Pascal

```
function GetCount: Integer; virtual; abstract;
```

C++

```
virtual __fastcall int GetCount() = 0;
```

Description

GetCount is the protected read implementation of the Count (see page 6849) property.

In TWideStrings (see page 6825) GetCount is abstract or, in C++ terminology, pure virtual, meaning it has no implementation. Descendant classes must override this method to return the number of strings that have been added to the list.

See Also

Count (see page 6849)

70.1.2.3.20 TWideStrings.GetEnumerator Method

Returns a TWideStrings (see page 6825) enumerator.

Pascal

```
function GetEnumerator: TWideStringsEnumerator;
```

C++

```
__fastcall TWideStringsEnumerator GetEnumerator();
```

Description

GetEnumerator returns a TWideStringsEnumerator (see page 6856) reference, which enumerates the strings in a TWideStrings (see page 6825) object.

To process all these strings, call the TWideStringsEnumerator (see page 6856) GetCurrent method within a While MoveNext do loop.

See Also

TWideStringsEnumerator (see page 6856)

70.1.2.3.21 TWideStrings.GetObject Method

Returns the object associated with the string at a specified index.

Pascal

```
function GetObject(Index: Integer): TObject; virtual;
```

C++

```
virtual __fastcall TObject * GetObject(int Index);
```

Description

GetObject is the protected read implementation of the Objects (see page 6851) property.

Index is the index of the string with which the object is associated.

In TWideStrings (see page 6825), GetObject always returns nil (Delphi) or NULL (C++). This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override GetObject to return the specified object.

See Also

Objects (see page 6851)

70.1.2.3.22 TWideStrings.GetText Method

Allocates a text buffer and fills it with the value of the Text (see page 6854) property.

Pascal

```
function GetText: PwideChar; virtual;
```

C++

```
virtual __fastcall PwideChar GetText();
```

Description

Call GetText to obtain a dynamically allocated character buffer containing all of the strings in the list. Individual strings are separated by a carriage return and (on Windows) line feed. The caller is responsible for freeing the returned value using the StrDispose (see page 4247) procedure.

See Also

SetText (see page 6845)

Strings (see page 6853)

Text (see page 6854)

StrDispose (see page 4247)

70.1.2.3.23 TWideStrings.GetTextStr Method

Returns the value of the Text (see page 6854) property.

Pascal

```
function GetTextStr: WideString; virtual;
```

C++

```
virtual __fastcall BSTR GetTextStr();
```

Description

GetTextStr is the protected read implementation of the Text (see page 6854) property. It returns a string that lists all the strings in the list, with individual strings separated by a carriage return and (on Windows) line feed.

See Also

GetText (see page 6839)

Text (see page 6854)

70.1.2.3.24 TWideStrings.IndexOf Method

Returns the position of a string in the list.

Pascal

```
function IndexOf(const S: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOf(const BSTR S);
```

Description

Call IndexOf to obtain the position of the first occurrence of the string S, or of a string that differs from S only by case. IndexOf returns the 0-based index of the string. Thus, if S matches the first string in the list, IndexOf returns 0, if S is the second string, IndexOf returns 1, and so on. If the string is not in the string list, IndexOf returns -1.

Note: If the string appears in the list more than once, IndexOf returns the position of the first occurrence.

See Also

IndexOfName (see page 6840)

IndexOfObject (see page 6841)

Strings (see page 6853)

70.1.2.3.25 TWideStrings.IndexOfName Method

Returns the position of the first name-value pair with the specified name.

Pascal

```
function IndexOfName(const Name: WideString): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfName(const BSTR Name);
```

Description

Call IndexOfName to locate the first occurrence of a name-value pair where the name part is equal to the `Name` parameter or differs only in case. IndexOfName returns the 0-based index of the string. If no string in the list has the indicated name, IndexOfName returns -1.

Note: If there is more than one name-value pair with a name portion matching the `Name` parameter, IndexOfName returns the position of the first such string.

See Also

IndexOf (see page 6840)

Names (see page 6850)

Values (see page 6856)

70.1.2.3.26 TWideStrings.IndexOfObject Method

Returns the index of the first string in the list associated with a given object.

Pascal

```
function IndexOfObject(AObject: TObject): Integer; virtual;
```

C++

```
virtual __fastcall int IndexOfObject(TObject * AObject);
```

Description

Call IndexOfObject to locate the first string in the list associated with the object AObject. Specify the object you want to locate as the value of the AObject parameter. IndexOfObject returns the 0-based index of the string and object. If the object is not associated with any of the strings, IndexOfObject returns -1.

See Also

IndexOf (see page 6840)

Objects (see page 6851)

70.1.2.3.27 TWideStrings.Insert Method

Introduces abstract (Delphi) or pure virtual (C++) method to insert a string at a specified position.

Pascal

```
procedure Insert(Index: Integer; const S: WideString); virtual; abstract;
```

C++

```
virtual __fastcall Insert(int Index, const BSTR S) = 0;
```

Description

Descendants of TWideStrings (see page 6825) implement an Insert method to add the string S to the list at the position specified by `Index`. If `Index` is 0, the string is inserted at the beginning of the list. If `Index` is 1, the string is put in the second position of the list, and so on.

All methods that add strings to the list use the Insert method to add the string.

If the string has an associated object, use the InsertObject (see page 6842) method instead.

See Also

Add (see page 6829)

AddObject (see page 6830)

AddStrings (see page 6830)

Append (see page 6831)

BeginUpdate (see page 6832)

EndUpdate (see page 6835)

InsertObject (see page 6842)

Strings (see page 6853)

70.1.2.3.28 TWideStrings.InsertObject Method

Inserts a string into the list at the specified position, and associates it with an object.

Pascal

```
procedure InsertObject(Index: Integer; const S: WideString; AObject: TObject); virtual;
```

C++

```
virtual __fastcall InsertObject(int Index, const BSTR S, TObject * AObject);
```

Description

Call InsertObject to insert the string S into the list at the position identified by `Index`, and associate it with the object AObject. If `Index` is 0, the string is inserted at the beginning of the list. If `Index` is 1, the string is put in the second position of the list, and so on.

See Also

Insert (see page 6841)

Objects (see page 6851)

70.1.2.3.29 TWideStrings.LoadFromFile Method

Fills the list with the lines of text in a specified file.

Pascal

```
procedure LoadFromFile(const FileName: WideString); virtual;
```

C++

```
virtual __fastcall LoadFromFile(const BSTR FileName);
```

Description

Call LoadFromFile to fill the list of the TWideStrings (see page 6825) object from the file specified by FileName. LoadFromFile first clears any strings already in the list. Then, each line in the file, as indicated by carriage return or linefeed characters, is appended as a string in the list.

Note: LoadFromFile uses the Add (see page 6829) method to add the strings that are read from the file.

See Also

Add (see page 6829)

LoadFromStream (see page 6842)

SaveToFile (see page 6844)

Strings (see page 6853)

Text (see page 6854)

70.1.2.3.30 TWideStrings.LoadFromStream Method

Fills the list with lines of text read from a stream.

Pascal

```
procedure LoadFromStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall LoadFromStream(TStream Stream);
```

Description

Call LoadFromStream to fill the list of the TWideStrings (see page 6825) object from the stream specified by Stream. The text read from the stream is parsed into strings separated by carriage return or linefeed characters. Thus, LoadFromStream reads the value of the Text (see page 6854) property.

If the stream is a file stream, LoadFromStream does the same thing as LoadFromFile (see page 6842), except the application must create and destroy the file stream.

See Also

Add (see page 6829)

LoadFromFile (see page 6842)

SaveToStream (see page 6845)

Text (see page 6854)

70.1.2.3.31 TWideStrings.Move Method

Changes the position of a string in the list.

Pascal

```
procedure Move(CurIndex: Integer; NewIndex: Integer); virtual;
```

C++

```
virtual __fastcall Move(int CurIndex, int NewIndex);
```

Description

Use Move to move the string at position CurIndex so that it occupies the position NewIndex. The positions are specified as 0-based indexes. For example, the following lines of code move the string in the first position to the last position.

```
MyStringsObject.Move(0, MyStringsObject.Count)
MyStringsObject->Move(0, MyStringsObject->Count - 1);
```

If the string has an associated object, the object remains associated with the string in its new position.

See Also

Count (see page 6849)

Objects (see page 6851)

Strings (see page 6853)

70.1.2.3.32 TWideStrings.Put Method

Changes the value of the string with a specified index.

Pascal

```
procedure Put(Index: Integer; const S: WideString); virtual;
```

C++

```
virtual __fastcall Put(int Index, const BSTR S);
```

Description

Put is the protected write implementation of the Strings (see page 6853) property.

Put changes the value of the string with the index specified by `Index` to S. Put does not change the object at the specified position. That is, any object associated with the previous string becomes associated with the new string.

See Also

Strings (see page 6853)

Objects (see page 6851)

70.1.2.3.33 TWideStrings.PutObject Method

Changes the object associated with the string at a specified index.

Pascal

```
procedure PutObject(Index: Integer; AObject: TObject); virtual;
```

C++

```
virtual __fastcall PutObject(int Index, TObject * AObject);
```

Description

PutObject is the protected write implementation of the Objects (see page 6851) property.

As implemented in TWideStrings (see page 6825), PutObject does nothing. This provides a default implementation for descendants that do not support associating objects with the strings in the list. Descendants that support this feature override PutObject to change the specified object.

See Also

Objects (see page 6851)

70.1.2.3.34 TWideStrings.SaveToFile Method

Saves the strings in the list to the specified file.

Pascal

```
procedure SaveToFile(const FileName: WideString); virtual;
```

C++

```
virtual __fastcall SaveToFile(const BSTR FileName);
```

Description

Call SaveToFile to save the strings in the list to the file specified by FileName. Each string in the list is written to a separate line in the file.

Note: In Linux, SaveToFile saves the strings as they appear in the list. If the last string does not have an end-of-line marker, the resulting file may be considered incomplete by some editors.

See Also

LoadFromFile (see page 6842)

SaveToStream (see page 6845)

Strings (see page 6853)

Text (see page 6854)

70.1.2.3.35 TWideStrings.SaveToStream Method

Writes the value of the Text (see page 6854) property to a stream object.

Pascal

```
procedure SaveToStream(Stream: TStream); virtual;
```

C++

```
virtual __fastcall SaveToStream(TStream Stream);
```

Description

Call SaveToStream to save the strings in the list to the stream specified by the Stream parameter. SaveToStream writes the strings delimited by carriage return, line feed pairs. If the stream is a file stream, SaveToStream does the same thing as SaveToFile (see page 6844), except the application must create and destroy the file stream.

See Also

LoadFromStream (see page 6842)

SaveToFile (see page 6844)

Strings (see page 6853)

Text (see page 6854)

70.1.2.3.36 TWideStrings.SetCapacity Method

Changes the amount of memory allocated to hold strings in the list.

Pascal

```
procedure SetCapacity(NewCapacity: Integer); virtual;
```

C++

```
virtual __fastcall SetCapacity(int NewCapacity);
```

Description

SetCapacity is the protected read implementation of the Capacity (see page 6847) property.

Integer is the number of strings the list can hold after the capacity has changed.

In TWideStrings (see page 6825), SetCapacity does nothing. Descendant classes override this method to change the number of strings that the list can hold. For descendant classes that implement SetCapacity, assigning a value smaller than Count (see page 6849) removes strings from the end of the list. Assigning a value greater than Count (see page 6849) allocates space for more strings to be added.

See Also

Capacity (see page 6847)

Count (see page 6849)

70.1.2.3.37 TWideStrings.SetText Method

Sets the Text (see page 6854) property.

Pascal

```
procedure SetText(Text: PwideChar); virtual;
```

C++

```
virtual __fastcall SetText(PwideChar Text);
```

Description

Call SetText to replace the list with the strings specified by the Text (see page 6854) parameter. SetText adds strings one at a time to the list, using the carriage returns or linefeed characters in Text (see page 6854) as delimiters indicating when to add a new string.

See Also

SetTextStr (see page 6846)

Text (see page 6854)

70.1.2.3.38 TWideStrings.SetTextStr Method

Sets the Text (see page 6854) property.

Pascal

```
procedure SetTextStr(const Value: WideString); virtual;
```

C++

```
virtual __fastcall SetTextStr(const BSTR Value);
```

Description

GetTextStr (see page 6839) is the protected write implementation of the Text (see page 6854) property. It replaces the list with the strings specified by the Value parameter. SetTextStr adds strings one at a time to the list, using the carriage returns or linefeed characters in Value as delimiters indicating when to add a new string.

See Also

SetText (see page 6845)

Text (see page 6854)

70.1.2.3.39 TWideStrings.SetUpdateState Method

Performs internal adjustments before or after a series of updates.

Pascal

```
procedure SetUpdateState(Updating: Boolean); virtual;
```

C++

```
virtual __fastcall SetUpdateState(Boolean Updating);
```

Description

SetUpdateState is called at the beginning or end of a series of updates. When the BeginUpdate (see page 6832) method is first called and the TWideStrings (see page 6825) object is not already in the middle of an update, TWideStrings (see page 6825) calls SetUpdateState internally, with Updating set to true. When the EndUpdate (see page 6835) method is called and it cancels out the last unmatched call to BeginUpdate (see page 6832), TWideStrings (see page 6825) calls SetUpdateState internally, with Updating set to false.

As implemented in TWideStrings (see page 6825), SetUpdateState does nothing. Descendant classes can override this method to optimize the response to updates.

See Also

BeginUpdate (see page 6832)

EndUpdate (see page 6835)

70.1.2.4 TWideStrings Properties

The properties of the TWideStrings class are listed here.

Protected Properties

	Name	Description
R	UpdateCount (see page 6855)	Indicates the number of calls to BeginUpdate (see page 6832) that have not been matched by a call to EndUpdate (see page 6835).

Public Properties

	Name	Description
	Capacity (see page 6847)	Indicates the number of strings the TWideStrings (see page 6825) object can hold.
	CommaText (see page 6848)	Lists the strings in the TWideStrings (see page 6825) object in system data format (SDF).
R	Count (see page 6849)	Introduces an abstract property to represent the number of strings in the list.
	DelimitedText (see page 6849)	Represents all the strings in the TWideStrings (see page 6825) object as a single delimited string.
	Delimiter (see page 6850)	Specifies the delimiter used by the DelimitedText (see page 6849) property.
	LineBreak (see page 6850)	Defines the line break character(s).
R	Names (see page 6850)	Indicates the name part of strings that are name-value pairs.
	NameValueSeparator (see page 6851)	Indicates the character used to separate names from values.
	Objects (see page 6851)	Represents a set of objects that are associated one with each of the strings in the Strings (see page 6853) property.
	QuoteChar (see page 6852)	Specifies the quote character used by the DelimitedText (see page 6849) property.
	StrictDelimiter (see page 6852)	Determines how the Delimiter (see page 6850) property is used.
	Strings (see page 6853)	References the strings in the list by their positions.
	StringsAdapter (see page 6854)	Implements an IWideStringsAdapter (see page 6859) interface for the TWideStrings (see page 6825) object.
	Text (see page 6854)	Lists the strings in the TWideStrings (see page 6825) object as a single string with the individual strings delimited by carriage returns and line feeds.
	ValueFromIndex (see page 6855)	Represents the value part of a string with a given index, on strings that are name-value pairs.
	Values (see page 6856)	Represents the value part of a string associated with a given name, on strings that are name-value pairs.

70.1.2.4.1 TWideStrings.Capacity Property

Indicates the number of strings the TWideStrings (see page 6825) object can hold.

Pascal

```
property Capacity: Integer;
```

C++

```
__property int Capacity;
```

Description

Read (see page 3634) Capacity to determine the currently allocated size of the string list. For the TWideStrings (see page 6825) object, reading Capacity returns the Count (see page 6849) property, and setting Capacity does nothing. Descendants of TWideStrings (see page 6825) can override this property to allow a string list to allocate memory for entries that have not

been added to the list.

See Also

Count (see page 6849)

70.1.2.4.2 TWideStrings.CommaText Property

Lists the strings in the TWideStrings (see page 6825) object in system data format (SDF).

Pascal

```
property CommaText: WideString;
```

C++

```
__property BSTR CommaText;
```

Description

Use CommaText to get or set all the strings in the TWideStrings (see page 6825) object in a single comma-delimited string.

When retrieving CommaText, any string in the list that include spaces, commas or quotes will be contained in double quotes, and any double quotes in a string will be repeated. For example, if the list contains the following strings:

```
Stri,ng 1
Stri"ng 2
String 3
String4
```

CommaText will return:

```
"Stri,ng 1","Stri""ng 2","String 3",String4
```

When assigning CommaText, the value is parsed as SDF formatted text. For SDF format, strings are separated by commas or spaces, and optionally enclosed in double quotes. Double quote marks that are part of the string are repeated to distinguish them from the quotes that surround the string. Spaces and commas that are not contained within double quote marks are delimiters. Two commas next to each other will indicate an empty string, but spaces that appear next to another delimiter are ignored. For example, suppose CommaText is set to:

```
"Stri,ng 1", "Stri""ng 2" , String 3,String4
```

The list will then contain:

```
Stri,ng 1
Stri"ng 2
String
3
String4
```

Note: CommaText is the same as the DelimitedText (see page 6849) property with a delimiter of ',' and a quote character of "".

Including a trailing comma in the source string causes a blank item to be included in the string list. For example, if CommaText is set to

```
"String1, String 2, String 3,"
```

the string list will contain

```
String1
String2
String3
<Blank>
```

See Also

Add (see page 6829)

Strings (see page 6853)

Text (see page 6854)

DelimitedText (see page 6849)

70.1.2.4.3 TWideStrings.Count Property

Introduces an abstract property to represent the number of strings in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Descendants of TWideStrings (see page 6825) implement a Count property to indicate the number of strings in the list.

Use the Count property when iterating over all the strings in the list, or when trying to locate the position of a string relative to the last string in the list.

See Also

IndexOf (see page 6840)

Strings (see page 6853)

70.1.2.4.4 TWideStrings.DelimitedText Property

Represents all the strings in the TWideStrings (see page 6825) object as a single delimited string.

Pascal

```
property DelimitedText: WideString;
```

C++

```
__property BSTR DelimitedText;
```

Description

Use DelimitedText to get or set all the strings in the TWideStrings (see page 6825) object in a single string.

When retrieving DelimitedText, the resulting value delimits individual strings in two ways: each string is surrounded (before and after) by the quote character specified by the QuoteChar (see page 6852) property. In addition, individual strings are separated by the character specified by the Delimiter (see page 6850) property.

When assigning DelimitedText, individual strings must be separated using QuoteChar (see page 6852) at both ends, using Delimiter (see page 6850) as a separator, or using both these methods.

If StrictDelimiter (see page 6852) is set to false, the space character is also interpreted as a delimiter, regardless of the value of Delimiter (see page 6850). This is not true when the space character occurs between quotation marks.

Note: CommaText (see page 6848) is the same as the DelimitedText property when Delimiter (see page 6850) is ',' and QuoteChar (see page 6852) is '"'.

See Also

CommaText (see page 6848)

Delimiter (see page 6850)

Text (see page 6854)

QuoteChar (see page 6852)

StrictDelimiter (see page 6852)

70.1.2.4.5 TWideStrings.Delimiter Property

Specifies the delimiter used by the DelimitedText (see page 6849) property.

Pascal

```
property Delimiter: WideChar;
```

C++

```
__property WideChar Delimiter;
```

Description

Use Delimiter to get or set the delimiter used by the DelimitedText (see page 6849) property. DelimitedText (see page 6849) represents all of the strings in the TWideStrings (see page 6825) object as a single string, with individual strings separated by the character that is the value of Delimiter.

See Also

DelimitedText (see page 6849)

QuoteChar (see page 6852)

70.1.2.4.6 TWideStrings.LineBreak Property

Defines the line break character(s).

Pascal

```
property LineBreak: WideString;
```

C++

```
__property BSTR LineBreak;
```

Description

LineBreak can be read and set. It is used internally in TWideStrings (see page 6825) to terminate strings in multi-string output operations.

For example, the GetText (see page 6839) method returns a long string containing all TWideStrings (see page 6825) strings, each of which is terminated by the LineBreak value.

Note: In Windows, the default LineBreak value is a carriage return and line feed combination (#13#10), whereas on Linux, it is just a carriage return (#13).

See Also

GetText (see page 6839)

70.1.2.4.7 TWideStrings.Names Property

Indicates the name part of strings that are name-value pairs.

Pascal

```
property Names [Index: Integer]: WideString;
```

C++

```
__property BSTR Names[int Index];
```

Description

When the list of strings for the TWideStrings (see page 6825) object includes strings that are name-value pairs, read Names to access the name part of a string. Names is the name part of the string at `Index`, where 0 is the first string, 1 is the second string, and so on. If the string is not a name-value pair, Names contains an empty string.

See Also

NameValueSeparator (see page 6851)

IndexOfName (see page 6840)

Strings (see page 6853)

Values (see page 6856)

70.1.2.4.8 TWideStrings.NameValueSeparator Property

Indicates the character used to separate names from values.

Pascal

```
property NameValueSeparator: WideChar;
```

C++

```
__property WideChar NameValueSeparator;
```

Description

Strings (see page 6853) that contain the NameValueSeparator character are considered name-value pairs. NameValueSeparator defaults to the equal sign (=). TWideStrings (see page 6825) defines various methods for accessing names and values and for searching for specific names.

Strings (see page 6853) that are name-value pairs consist of a name part, the separator character, and a value part. Any spaces around the separator character are part of the name or value. This convention corresponds to the format used in many initialization files. For example:

```
DisplayGrid=1
SnapToGrid=1
GridSizeX=8
GridSizeY=8
```

See Also

ValueFromIndex (see page 6855)

Names (see page 6850)

IndexOfName (see page 6840)

Strings (see page 6853)

Values (see page 6856)

70.1.2.4.9 TWideStrings.Objects Property

Represents a set of objects that are associated one with each of the strings in the Strings (see page 6853) property.

Pascal

```
property Objects [Index: Integer]: TObject;
```

C++

```
__property TObject * Objects[int Index];
```

Description

Setting the Objects property for TWideStrings (see page 6825) does nothing. Reading the Objects property for TWideStrings (see page 6825) returns nil (Delphi) or NULL (C++). Descendant classes can associate objects with the strings in the set by implementing the Objects property.

Use the Objects property of a descendant of TWideStrings (see page 6825) to get or set the object associated with the string at the position indicated by `Index`. Index gives the position of the string associated with the object, where 0 is the first string, 1 is the second string, and so on. If a descendant of TWideStrings (see page 6825) does not support the Objects property, reading this property returns nil (Delphi) or NULL (C++).

Note: The TWideStrings (see page 6825) object does not own the objects in the Objects array. Objects added to the Objects array still exist even if the TWideStrings (see page 6825) object is destroyed. They must be explicitly destroyed by the application.

See Also

AddObject (see page 6830)

IndexOfObject (see page 6841)

InsertObject (see page 6842)

Strings (see page 6853)

70.1.2.4.10 TWideStrings.QuoteChar Property

Specifies the quote character used by the DelimitedText (see page 6849) property.

Pascal

```
property QuoteChar: WideChar;
```

C++

```
__property WideChar QuoteChar;
```

Description

Use QuoteChar to get or set the quote character that is used to enclose individual strings in the DelimitedText (see page 6849) property.

See Also

Delimiter (see page 6850)

DelimitedText (see page 6849)

CommaText (see page 6848)

70.1.2.4.11 TWideStrings.StrictDelimiter Property

Determines how the Delimiter (see page 6850) property is used.

Pascal

```
property StrictDelimiter: Boolean;
```

C++

```
__property Boolean StrictDelimiter;
```

Description

Use this property to specify whether the Delimiter (see page 6850) is the only value used within the DelimitedText (see page 6849) property. If set to True, individual strings in DelimitedText (see page 6849) are separated only by the character that is the value of Delimiter (see page 6850). If set to False, individual strings in DelimitedText (see page 6849) can be separated by a space, a non-printable character, or the character that is the value of Delimiter (see page 6850).

70.1.2.4.12 TWideStrings.Strings Property

References the strings in the list by their positions.

Pascal

```
property Strings [Index: Integer]: WideString;
```

C++

```
__property BSTR Strings[int Index];
```

Description

Descendants of TWideStrings (see page 6825) must implement an accessor function for the Strings property to return the string at the position indicated by `Index`. `Index` gives the position of the string, where 0 is the first string, 1 is the second string, and so on.

Use the Strings property to get or set the string at a particular position.

To store a string as a name-value pair, assign Strings a value that includes the NameValueSeparator (see page 6851) character. The name and value will then be accessible separately using the Names (see page 6850) and Values (see page 6856) properties.

Note: In Delphi, Strings is the default property of TWideStrings (see page 6825) objects. The Strings identifier can be omitted when accessing the Strings property of a descendant of TWideStrings (see page 6825). For example, the following two lines of code are both acceptable and do the same thing:

```
MyStrings.Strings[0] := 'This is the first string';
        MyStrings[0] := 'This is the first string';
```

See Also

ValueFromIndex (see page 6855)

NameValueSeparator (see page 6851)

Names (see page 6850)

Values (see page 6856)

CommaText (see page 6848)

IndexOf (see page 6840)

Objects (see page 6851)

Text (see page 6854)

70.1.2.4.13 TWideStrings.StringsAdapter Property

Implements an IWideStringsAdapter (see page 6859) interface for the TWideStrings (see page 6825) object.

Pascal

```
property StringsAdapter: IWideStringsAdapter;
```

C++

```
__property IWideStringsAdapter StringsAdapter;
```

Description

Do not use the StringsAdapter property. StringsAdapter is used internally.

See Also

TStringsAdapter

70.1.2.4.14 TWideStrings.Text Property

Lists the strings in the TWideStrings (see page 6825) object as a single string with the individual strings delimited by carriage returns and line feeds.

Pascal

```
property Text: WideString;
```

C++

```
__property BSTR Text;
```

Description

Use Text to get or set all the strings in the TWideStrings (see page 6825) object in a single string delimited by carriage return, line feed pairs.

When reading Text, the strings in the list will be separated by carriage return and (on Windows) line feed. If any of the strings in the list contain a carriage return (and line feed), the resulting value of Text will appear to contain more strings than is indicated by the Count (see page 6849) property.

When setting Text, the value will be parsed by separating into substrings whenever a carriage return or linefeed is encountered. (The two do not need to form pairs).

If the strings in the list contain carriage return or linefeed characters, a less ambiguous format for the strings is available through the CommaText (see page 6848) or DelimitedText (see page 6849) property.

See Also

Add (see page 6829)

CommaText (see page 6848)

Count (see page 6849)

Strings (see page 6853)

GetText (see page 6839)

SetText (see page 6845)

DelimitedText (see page 6849)

70.1.2.4.15 TWideStrings.UpdateCount Property

Indicates the number of calls to BeginUpdate (see page 6832) that have not been matched by a call to EndUpdate (see page 6835).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

TWideStrings (see page 6825) uses UpdateCount to keep track of calls to the BeginUpdate (see page 6832) and EndUpdate (see page 6835) methods. Every time a call is made to BeginUpdate (see page 6832), TWideStrings (see page 6825) increments the value of UpdateCount. Every call to EndUpdate (see page 6835) causes TWideStrings (see page 6825) to decrement UpdateCount.

When UpdateCount changes from 0 to 1, TWideStrings (see page 6825) calls the SetUpdateState (see page 6846) method with a parameter of true. When UpdateCount changes from 1 to 0, TStrings calls the SetUpdateState (see page 6846) method with a parameter of false. This allows descendant classes to perform optimizations when handling multiple updates.

See Also

BeginUpdate (see page 6832)

EndUpdate (see page 6835)

SetUpdateState (see page 6846)

70.1.2.4.16 TWideStrings.ValueFromIndex Property

Represents the value part of a string with a given index, on strings that are name-value pairs.

Pascal

```
property ValueFromIndex [Index: Integer]: WideString;
```

C++

```
__property BSTR ValueFromIndex[int Index];
```

Description

When the list of strings for the TWideStrings (see page 6825) object includes strings that are name-value pairs, use ValueFromIndex to get or set the value part of a string associated with an index.

For more information on name-value pairs, refer to the NameValueSeparator (see page 6851) property.

See Also

NameValueSeparator (see page 6851)

IndexOfName (see page 6840)

Strings (see page 6853)

Values (see page 6856)

70.1.2.4.17 TWideStrings.Values Property

Represents the value part of a string associated with a given name, on strings that are name-value pairs.

Pascal

```
property Values [const Name: WideString]: WideString;
```

C++

```
__property BSTR Values[BSTR const Name];
```

Description

When the list of strings for the TWideStrings (see page 6825) object includes strings that are name-value pairs, use Values to get or set the value part of a string associated with a specific name part.

For more information on name-value pairs, refer to the NameValueSeparator (see page 6851) property.

Note: The `Name` index is case-insensitive. That is, Values is the value part for the first occurrence of `Name` or an equivalent string that differs only in case.

See Also

ValueFromIndex (see page 6855)

NameValueSeparator (see page 6851)

IndexOfName (see page 6840)

Names (see page 6850)

Strings (see page 6853)

70.1.3 TWideStringsEnumerator Class

Enumerates through the strings of a TWideStrings (see page 6825) object.

Class Hierarchy

Pascal

```
TWideStringsEnumerator = class;
```

C++

```
class TWideStringsEnumerator;
```

File

WideStrings

Description

TWideStringsEnumerator provides an enumerator for TWideStrings (see page 6825) objects.

70.1.3.1 TWideStringsEnumerator Members

The following tables list the members exposed by TWideStringsEnumerator.

Public Methods

	Name	Description
	Create (see page 6857)	Creates a TWideStrings (see page 6825) enumerator.
	GetCurrent (see page 6857)	Gets the current string.
	MoveNext (see page 6857)	Moves the enumerator to the next string in the TWideStrings (see page 6825) collection.

Public Properties

	Name	Description
R	Current (see page 6858)	The current string.

70.1.3.2 TWideStringsEnumerator.Create Constructor

Creates a TWideStrings (see page 6825) enumerator.

Pascal

```
constructor Create(AStrings: TWideStrings);
```

C++

```
__fastcall TWideStringsEnumerator(TWideStrings AStrings);
```

Description

Creates an instance of TWideStringsEnumerator (see page 6856).

70.1.3.3 TWideStringsEnumerator Methods

The methods of the TWideStringsEnumerator class are listed here.

Public Methods

	Name	Description
	GetCurrent (see page 6857)	Gets the current string.
	MoveNext (see page 6857)	Moves the enumerator to the next string in the TWideStrings (see page 6825) collection.

70.1.3.3.1 TWideStringsEnumerator.GetCurrent Method

Gets the current string.

Pascal

```
function GetCurrent: WideString;
```

C++

```
__fastcall BSTR GetCurrent();
```

Description

Use GetCurrent to get the current string in the TWideStrings (see page 6825) object being enumerated.

70.1.3.3.2 TWideStringsEnumerator.MoveNext Method

Moves the enumerator to the next string in the TWideStrings (see page 6825) collection.

Pascal

```
function MoveNext: Boolean;
```

C++

```
__fastcall Boolean MoveNext();
```

Description

Use MoveNext to move the TWideStringsEnumerator (see page 6856) to the next string in the TWideStrings (see page 6825) object being enumerated.

70.1.3.4 TWideStringsEnumerator Properties

The properties of the TWideStringsEnumerator class are listed here.

Public Properties

	Name	Description
	Current (see page 6858)	The current string.

70.1.3.4.1 TWideStringsEnumerator.Current Property

The current string.

Pascal

```
property Current: WideString;
```

C++

```
__property BSTR Current;
```

Description

Current contains the current string in the TWideStrings (see page 6825) object being enumerated.

70.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWideStringsAdapter (see page 6859)	This is class WideStrings.IWideStringsAdapter.

70.2.1 IWideStringsAdapter Interface

Class Hierarchy

WideStrings.IWideStringsAdapter

Pascal

```
IWideStringsAdapter = interface;
```

C++

```
__interface IWideStringsAdapter;
```

File

WideStrings

Description

This is class WideStrings.IWideStringsAdapter.

70.2.1.1 IWideStringsAdapter Members

The following tables list the members exposed by IWideStringsAdapter.

Public Methods

	Name	Description
	ReferenceStrings (see page 6860)	This is ReferenceStrings, a member of class IWideStringsAdapter.
	ReleaseStrings (see page 6860)	This is ReleaseStrings, a member of class IWideStringsAdapter.

70.2.1.2 IWideStringsAdapter Methods

The methods of the IWideStringsAdapter class are listed here.

Public Methods

	Name	Description
	ReferenceStrings (see page 6860)	This is ReferenceStrings, a member of class IWideStringsAdapter.
	ReleaseStrings (see page 6860)	This is ReleaseStrings, a member of class IWideStringsAdapter.

70.2.1.2.1 IWideStringsAdapter.ReferenceStrings Method

Pascal

```
procedure ReferenceStrings(S: TWideStrings);
```

C++

```
__fastcall ReferenceStrings(TWideStrings S);
```

Description

This is ReferenceStrings, a member of class IWideStringsAdapter.

70.2.1.2.2 IWideStringsAdapter.ReleaseStrings Method

Pascal

```
procedure ReleaseStrings;
```

C++

```
__fastcall ReleaseStrings();
```

Description

This is ReleaseStrings, a member of class IWideStringsAdapter.

70.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Records

	Name	Description
	TWideStringItem (see page 6861)	TWideStringItem defines a TWideStringList (see page 6808) entry.

70.3.1 WideStrings.TWideStringItem Record

TWideStringItem defines a TWideStringList (see page 6808) entry.

Pascal

```
TWideStringItem = record
  FString: WideString;
  FObject: TObject;
end;
```

C++

```
struct TWideStringItem {
  BSTR FString;
  TObject * FObject;
};
```

File

WideStrings

Description

The TWideStringItem type defines a data structure used internally in Delphi.

It is used by TWideStringList (see page 6808) to hold a list entry.

FString defines an Unicode string for a list entry.

FObject points to an optional `TObject (see page 3354)` associated with the string.

See Also

TWideStringList (see page 6808)

70.4 Types

The following table lists types in this documentation.

Types

Name	Description
PWideStringItem (see page 6862)	PWideStringItem is a TWideStringItem (see page 6861) pointer.
PWideStringItemList (see page 6862)	PWideStringItemList is a TWideStringItemList (see page 6863) pointer.
TWideStringItemList (see page 6863)	TWideStringItemList defines an array of TWideStringItems.
TWideStringListSortCompare (see page 6863)	TWideStringListSortCompare is a type of callback function called used internally by the TWideStringList (see page 6808) class.

70.4.1 WideStrings.PWideStringItem Type

PWideStringItem is a TWideStringItem (see page 6861) pointer.

Pascal

```
PWideStringItem = ^TWideStringItem;
```

C++

```
^TWideStringItem PWideStringItem;
```

File

WideStrings

Description

PWideStringItem is used internally by TWideStringList (see page 6808) to point to a TWideStringItem (see page 6861) value.

TWideStringItem (see page 6861) comprises :

FString defines an Unicode string for a list entry.

FObject points to an optional System::TObject (see page 3354) associated with the string.

See Also

TWideStringItem (see page 6861)

TWideStringList (see page 6808)

70.4.2 WideStrings.PWideStringItemList Type

PWideStringItemList is a TWideStringItemList (see page 6863) pointer.

Pascal

```
PWideStringItemList = ^TWideStringItemList;
```

C++

```
^TWideStringItemList PWideStringItemList;
```

File

WideStrings

Description

PWideStringItemList is used internally by TWideStringList (see page 6808) to refer to its internal array of TWideStringItem (see page 6861) values. This array holds the list of strings.

TWideStringItem (see page 6861) comprises :

FString defines an Unicode string for a list entry.

FObject points to an optional System::TObject (see page 3354) associated with the string.

See Also

TWideStringItem (see page 6861)

TWideStringList (see page 6808)

70.4.3 WideStrings.TWideStringItemList Type

TWideStringItemList defines an array of TWideStringItems.

Pascal

```
TWideStringItemList = array[0..MaxListSize] of TWideStringItem;
```

C++

```
array[0..MaxListSize] of TWideStringItem TWideStringItemList;
```

File

WideStrings

Description

TWideStringItemList is used internally by TWideStringList (see page 6808) to refer to its internal array of TWideStringItem (see page 6861) values. This array holds the list of strings.

TWideStringItem (see page 6861) comprises :

FString defines an Unicode string for a list entry.

FObject points to an optional System::TObject (see page 3354) associated with the string.

See Also

TWideStringItem (see page 6861)

TWideStringList (see page 6808)

70.4.4 WideStrings.TWideStringListSortCompare Type

TWideStringListSortCompare is a type of callback function called used internally by the TWideStringList (see page 6808) class.

Pascal

```
TWideStringListSortCompare = function (List: TWideStringList; Index1, Index2: Integer):
```

```
Integer;
```

C++

```
(List: TWideStringList; Index1, Index2: Integer): Integer ( TWideStringListSortCompare)();
```

File

WideStrings

Description

You define a TWideStringListSortCompare function when you need to use the TWideStringList (see page 6808) CustomSort method. The procedure defined by TWideStringListSortCompare is invoked by CustomSort to sort the TWideStringList (see page 6808) elements into sequence.

CustomSort is used instead of Sort when the sort sequence required differs from the default alphanumeric sequence.

List is the list of Unicode strings.

Index1 and **Index2** are indexes into the list. When these are passed to the TListSortCompare function, the CustomSort method is asking which order they should be in.

You return 0 if the entry referred to by **Index1** equals the entry referred to by **Index2**

You return less than 0 if the entry referred to by **Index1** is less than the entry referred to by **Index2**

You return a number above 0 if the entry referred to by **Index1** is greater than the entry referred to by **Index2**

Note: One of the principle advantages of the CustomSort method is that a TWideStringList (see page 6808) may be sorted according to the objects optionally attachable to each string.

See Also

CustomSort (see page 6815)

TWideStringList (see page 6808)

71 WideStrUtils Namespace

This is namespace WideStrUtils.

Topics

Name	Description
Functions (see page 6867)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 6889)	The following table lists structs, records, enums in this documentation.
Types (see page 6890)	The following table lists types in this documentation.
Constants (see page 6891)	The following table lists constants in this documentation.

Constants

Name	Description
sUTF8BOMString (see page 6891)	Represents the UTF-8 byte-order mark.

Functions

	Name	Description
	AnsiToUtf8Ex (see page 6868)	Converts a string encoded in ANSI to UTF-8 with a given code page.
	ConvertStreamFromAnsiToUTF8 (see page 6868)	Converts a stream from ANSI to UTF-8 encoding.
	ConvertStreamFromUTF8ToAnsi (see page 6869)	Converts a stream from UTF-8 to ANSI encoding.
	DetectUTF8Encoding (see page 6869)	Detects the encoding of a given WideString (see page 3467).
	HasExtendCharacter (see page 6870)	Checks whether the string contains non-ASCII characters.
	HasUTF8BOM (see page 6870)	Detects if a string or a stream contains the UTF-8 byte-order mark.
	HasUTF8BOM (see page 6870)	Detects if a string or a stream contains the UTF-8 byte-order mark.
	inOpArray (see page 6871)	Checks if a WideChar is contained in a given array.
	inOpSet (see page 6871)	Checks if a WideChar is contained in a given set.
	IsUTF8LeadByte (see page 6871)	Checks whether a character is a valid UTF-8 lead byte.
	IsUTF8String (see page 6872)	Detects if a given string is UTF-8 encoded.
	IsUTF8TrailByte (see page 6872)	Checks whether a character is a valid UTF-8 trail byte.
	LoadResWideString (see page 6873)	Loads a WideString (see page 3467) resource.
	LoadWideStr (see page 6873)	Loads a WideString (see page 3467) resource.
	UTF8CharLength (see page 6874)	Returns the number of bytes that follow a lead UTF-8 byte.
	UTF8CharSize (see page 6874)	Returns the number of bytes that follow a lead UTF-8 byte.
	UTF8LowerCase (see page 6875)	Returns the UTF-8 string converted to lower case.
	Utf8ToAnsiEx (see page 6875)	Converts a string encoded in UTF-8 to ANSI with a given code page.
	UTF8UpperCase (see page 6876)	Returns the UTF-8 string converted to uppercase.

	WideAdjustLineBreaks (see page 6876)	Adjusts line break characters in a WideString (see page 3467) to reflect Windows or Linux style.
	WideDequotedStr (see page 6877)	Converts a quoted WideString (see page 3467) into an unquoted WideString (see page 3467).
	WideExtractQuotedStr (see page 6877)	Converts a quoted PWideChar into an unquoted WideString (see page 3467).
	WideLastChar (see page 6878)	Gets last character in a WideString (see page 3467).
	WideQuotedStr (see page 6878)	Returns the quoted version of a WideString (see page 3467).
	WideReplaceStr (see page 6879)	Replaces all occurrences of a substring within a string, using case-sensitive search.
	WideReplaceText (see page 6880)	Replaces all occurrences of a substring in a WideString (see page 3467).
	WideStringReplace (see page 6880)	Replaces occurrences of a substring within a string.
	WStrAlloc (see page 6881)	Allocates a buffer for a PWideChar and returns a pointer to its first character.
	WStrBufSize (see page 6881)	Returns the maximum number of characters that can be stored in a PWideChar allocated by WStrAlloc (see page 6881).
	WStrCat (see page 6882)	Appends a PWideChar to another PWideChar.
	WStrComp (see page 6882)	Compares two PWideChar values with case sensitivity.
	WStrCopy (see page 6883)	Copies a PWideChar to another PWideChar.
	WStrDispose (see page 6883)	Disposes of a PWideChar.
	WStrEnd (see page 6884)	Returns a pointer to the end of a PWideChar.
	WStrLCopy (see page 6884)	Copies up to a specified maximum number of characters from a PWideChar.
	WStrLen (see page 6885)	Returns number of characters in a PWideChar excluding the null terminator.
	WStrMove (see page 6885)	Copies a specified number of characters from a PWideChar to another PWideChar.
	WStrNew (see page 6886)	Allocates space on and copies a PWideChar to the heap, returning a new PWideChar.
	WStrPCopy (see page 6886)	Copies a WideString (see page 3467) to a PWideChar.
	WStrPLCopy (see page 6887)	Copies characters from a WideString (see page 3467) into a PWideChar.
	WStrPos (see page 6887)	Returns a pointer to the first occurrence of a PWideChar in another PWideChar.
	WStrScan (see page 6888)	Returns a pointer to the first occurrence of a specified character in a PWideChar.

Structs, Records, Enums

	Name	Description
	TEncodeType (see page 6889)	Defines a list of encodings.

Types

Name	Description
CharSet (see page 6890)	Represents a set of allowed characters.

71.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	AnsiToUtf8Ex (see page 6868)	Converts a string encoded in ANSI to UTF-8 with a given code page.
	ConvertStreamFromAnsiToUTF8 (see page 6868)	Converts a stream from ANSI to UTF-8 encoding.
	ConvertStreamFromUTF8ToAnsi (see page 6869)	Converts a stream from UTF-8 to ANSI encoding.
	DetectUTF8Encoding (see page 6869)	Detects the encoding of a given WideString (see page 3467).
	HasExtendCharacter (see page 6870)	Checks whether the string contains non-ASCII characters.
	HasUTF8BOM (see page 6870)	Detects if a string or a stream contains the UTF-8 byte-order mark.
	HasUTF8BOM (see page 6870)	Detects if a string or a stream contains the UTF-8 byte-order mark.
	inOpArray (see page 6871)	Checks if a WideChar is contained in a given array.
	inOpSet (see page 6871)	Checks if a WideChar is contained in a given set.
	IsUTF8LeadByte (see page 6871)	Checks whether a character is a valid UTF-8 lead byte.
	IsUTF8String (see page 6872)	Detects if a given string is UTF-8 encoded.
	IsUTF8TrailByte (see page 6872)	Checks whether a character is a valid UTF-8 trail byte.
	LoadResWideString (see page 6873)	Loads a WideString (see page 3467) resource.
	LoadWideStr (see page 6873)	Loads a WideString (see page 3467) resource.
	UTF8CharLength (see page 6874)	Returns the number of bytes that follow a lead UTF-8 byte.
	UTF8CharSize (see page 6874)	Returns the number of bytes that follow a lead UTF-8 byte.
	UTF8LowerCase (see page 6875)	Returns the UTF-8 string converted to lower case.
	Utf8ToAnsiEx (see page 6875)	Converts a string encoded in UTF-8 to ANSI with a given code page.
	UTF8UpperCase (see page 6876)	Returns the UTF-8 string converted to uppercase.
	WideAdjustLineBreaks (see page 6876)	Adjusts line break characters in a WideString (see page 3467) to reflect Windows or Linux style.
	WideDequotedStr (see page 6877)	Converts a quoted WideString (see page 3467) into an unquoted WideString (see page 3467).
	WideExtractQuotedStr (see page 6877)	Converts a quoted PWideChar into an unquoted WideString (see page 3467).
	WideLastChar (see page 6878)	Gets last character in a WideString (see page 3467).
	WideQuotedStr (see page 6878)	Returns the quoted version of a WideString (see page 3467).
	WideReplaceStr (see page 6879)	Replaces all occurrences of a substring within a string, using case-sensitive search.
	WideReplaceText (see page 6880)	Replaces all occurrences of a substring in a WideString (see page 3467).
	WideStringReplace (see page 6880)	Replaces occurrences of a substring within a string.
	WStrAlloc (see page 6881)	Allocates a buffer for a PWideChar and returns a pointer to its first character.
	WStrBufSize (see page 6881)	Returns the maximum number of characters that can be stored in a PWideChar allocated by WStrAlloc (see page 6881).
	WStrCat (see page 6882)	Appends a PWideChar to another PWideChar.
	WStrComp (see page 6882)	Compares two PWideChar values with case sensitivity.
	WStrCopy (see page 6883)	Copies a PWideChar to another PWideChar.
	WStrDispose (see page 6883)	Disposes of a PWideChar.
	WStrEnd (see page 6884)	Returns a pointer to the end of a PWideChar.
	WStrLCopy (see page 6884)	Copies up to a specified maximum number of characters from a PWideChar.
	WStrLen (see page 6885)	Returns number of characters in a PWideChar excluding the null terminator.
	WStrMove (see page 6885)	Copies a specified number of characters from a PWideChar to another PWideChar.
	WStrNew (see page 6886)	Allocates space on and copies a PWideChar to the heap, returning a new PWideChar.
	WStrPCopy (see page 6886)	Copies a WideString (see page 3467) to a PWideChar.
	WStrPLCopy (see page 6887)	Copies characters from a WideString (see page 3467) into a PWideChar.

	WStrPos (see page 6887)	Returns a pointer to the first occurrence of a PWideChar in another PWideChar.
	WStrScan (see page 6888)	Returns a pointer to the first occurrence of a specified character in a PWideChar.

71.1.1 WideStrUtils.AnsiToUtf8Ex Function

Converts a string encoded in ANSI to UTF-8 with a given code page.

Pascal

```
function AnsiToUtf8Ex(const S: AnsiString; const cp: integer): UTF8String;
```

C++

```
UTF8String AnsiToUtf8Ex(const AnsiString S, const int cp);
```

File

WideStrUtils

Description

AnsiToUtf8Ex converts the string specified by `S` to UTF-8. The code page used by the input **AnsiString** is specified in the `cp` parameter.

If the specified code page is `CP_UTF8` then a direct assignment is made; otherwise a conversion is performed.

See Also

Utf8ToAnsiEx (see page 6875)

AnsiToUtf8 (see page 3517)

Utf8ToAnsi (see page 3698)

71.1.2 WideStrUtils.ConvertStreamFromAnsiToUTF8 Function

Converts a stream from ANSI to UTF-8 encoding.

Pascal

```
procedure ConvertStreamFromAnsiToUTF8(Src: TStream; Dst: TStream; cp: integer = CP_ACP);
```

C++

```
ConvertStreamFromAnsiToUTF8(TStream Src, TStream Dst, int cp = CP_ACP);
```

File

WideStrUtils

Description

ConvertStreamFromAnsiToUTF8 converts an input stream specified by `Src` to an output stream specified by `Dst`. The function assumes that the input stream contains text encoded in ANSI and converts it to UTF-8. Optionally, the code page of the input ANSI stream is specified.

If the specified code page is `CP_UTF8` then a direct copy is made; otherwise a conversion is performed.

See Also

ConvertStreamFromUTF8ToAnsi (see page 6869)

71.1.3 WideStrUtils.ConvertStreamFromUTF8ToAnsi Function

Converts a stream from UTF-8 to ANSI encoding.

Pascal

```
procedure ConvertStreamFromUTF8ToAnsi(Src: TStream; Dst: TStream; cp: integer = CP_ACP);
```

C++

```
ConvertStreamFromUTF8ToAnsi(TStream Src, TStream Dst, int cp = CP_ACP);
```

File

WideStrUtils

Description

ConvertStreamFromUTF8ToAnsi converts an input stream specified by `Src` to an output stream specified by `Dst`. The function assumes that the input stream contains text encoded in UTF-8 and converts it to ANSI. Optionally, the code page of the output ANSI stream is specified.

If the specified code page is `CP_UTF8` then a direct copy is made; otherwise a conversion is performed.

71.1.4 WideStrUtils.DetectUTF8Encoding Function

Detects the encoding of a given WideString (see page 3467).

Pascal

```
function DetectUTF8Encoding(const s: RawByteString): TEncodeType;
```

C++

```
TEncodeType DetectUTF8Encoding(const RawByteString s);
```

File

WideStrUtils

Description

Call DetectUTF8Encoding to detect if a given WideString (see page 3467) is UTF-8, ASCII, or ANSI encoded.

Note: DetectUTF8Encoding does not guarantee that the detected encoding is correct. DetectUTF8Encoding attempts to identify the encoding by checking the characters present in the string.

See Also

TEncodeType (see page 6889)

HasExtendCharacter (see page 6870)

IsUTF8String (see page 6872)

71.1.5 WideStrUtils.HasExtendCharacter Function

Checks whether the string contains non-ASCII characters.

Pascal

```
function HasExtendCharacter(const s: RawByteString): Boolean;
```

C++

```
Boolean HasExtendCharacter(const RawByteString s);
```

File

WideStrUtils

Description

Call HasExtendCharacter to detect if a given string contains non-ASCII characters. All characters over `128` (`$FF`) are considered to be non-ASCII.

See Also

DetectUTF8Encoding (see page 6869)

HasUTF8BOM (see page 6870)

71.1.6 WideStrUtils.HasUTF8BOM Function

Detects if a string or a stream contains the UTF-8 byte-order mark.

Pascal

```
function HasUTF8BOM(S: TStream): boolean; overload;
function HasUTF8BOM(const S: RawByteString): boolean; overload;
```

C++

```
boolean HasUTF8BOM(TStream S);
boolean HasUTF8BOM(const RawByteString S);
```

File

WideStrUtils

Description

Call HasUTF8BOM to detect if a given string or stream contains the byte-order mark.

See Also

sUTF8BOMString (see page 6891)

DetectUTF8Encoding (see page 6869)

HasExtendCharacter (see page 6870)

71.1.7 WideStrUtils.inOpArray Function

Checks if a WideChar is contained in a given array.

Pascal

```
function inOpArray(W: WideChar; sets: array of WideChar): boolean;
```

C++

```
boolean inOpArray(WideChar W, array of WideChar sets);
```

File

WideStrUtils

Description

Call inOpArray to check whether a WideChar identified by `W` parameter is present in the `sets` array.

See Also

CharSet (see page 6890)

inOpSet (see page 6871)

71.1.8 WideStrUtils.inOpSet Function

Checks if a WideChar is contained in a given set.

Pascal

```
function inOpSet(W: WideChar; sets: CharSet): boolean;
```

C++

```
boolean inOpSet(WideChar W, CharSet sets);
```

File

WideStrUtils

Description

Call inOpSet to check whether a WideChar identified by `W` parameter is present in the set of characters identified by `sets` parameter.

If the ordinal value of `W` is less than 255, the **in** operator is used on the input set; otherwise the function returns false.

See Also

CharSet (see page 6890)

inOpArray (see page 6871)

71.1.9 WideStrUtils.IsUTF8LeadByte Function

Checks whether a character is a valid UTF-8 lead byte.

Pascal

```
function IsUTF8LeadByte(Lead: AnsiChar): Boolean;
```

C++

```
Boolean IsUTF8LeadByte(AnsiChar Lead);
```

File

WideStrUtils

Description

Call IsUTF8LeadByte to check whether a character is a valid UTF-8 lead byte.

All bytes between `0 .. 127 ($7F)` and `194 ($C2) .. 253 ($FD)` are valid UTF-8 lead bytes.

See Also

IsUTF8TrailByte (see page 6872)

HasExtendCharacter (see page 6870)

71.1.10 WideStrUtils.IsUTF8String Function

Detects if a given string is UTF-8 encoded.

Pascal

```
function IsUTF8String(const s: RawByteString): Boolean;
```

C++

```
Boolean IsUTF8String(const RawByteString s);
```

File

WideStrUtils

Description

Call IsUTF8String to detect if a given string is UTF-8 encoded.

Note: IsUTF8String does not guarantee that the string is UTF-8 encoded. IsUTF8String attempts to determine the encoding by checking the characters present in the string.

See Also

DetectUTF8Encoding (see page 6869)

HasExtendCharacter (see page 6870)

71.1.11 WideStrUtils.IsUTF8TrailByte Function

Checks whether a character is a valid UTF-8 trail byte.

Pascal

```
function IsUTF8TrailByte(Lead: AnsiChar): Boolean;
```

C++

```
Boolean IsUTF8TrailByte(AnsiChar Lead);
```

File

WideStrUtils

Description

Call IsUTF8TrailByte to check whether a character is a valid UTF-8 trail byte.

All bytes between `128 ($80) .. 191 ($BF)` are valid UTF-8 trail bytes.

See Also

IsUTF8LeadByte (see page 6871)

HasExtendCharacter (see page 6870)

71.1.12 WideStrUtils.LoadResWideString Function

Loads a WideString (see page 3467) resource.

Pascal

```
function LoadResWideString(ResStringRec: PResStringRec): WideString;
```

C++

```
BSTR LoadResWideString(PResStringRec ResStringRec);
```

File

WideStrUtils

Description

LoadResWideString loads the WideString (see page 3467) resource given by `ResStringRec` from an application module or shared library. If the resource does not exist, LoadResWideString returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

LoadWideStr (see page 6873)

LoadStr (see page 4224)

LoadResString (see page 3610)

71.1.13 WideStrUtils.LoadWideStr Function

Loads a WideString (see page 3467) resource.

Pascal

```
function LoadWideStr(Ident: Integer): WideString;
```

C++

```
BSTR LoadWideStr(int Ident);
```

File

WideStrUtils

Description

LoadWideStr loads the WideString (see page 3467) resource given by `Ident` from an application module or shared library. If the resource does not exist, LoadWideStr returns an empty string.

Moving display strings into string resources makes an application easier to localize without rewriting the program.

See Also

LoadResWideString (see page 6873)

LoadStr (see page 4224)

LoadResString (see page 3610)

71.1.14 WideStrUtils.UTF8CharLength Function

Returns the number of bytes that follow a lead UTF-8 byte.

Pascal

```
function UTF8CharLength(Lead: AnsiChar): Integer;
```

C++

```
int UTF8CharLength(AnsiChar Lead);
```

File

WideStrUtils

Description

Call UTF8CharLength to determine the number of bytes that follow a lead UTF-8 character (including the lead byte).

In UTF-8, each lead character can be followed by a variable number of other characters. This combination of UTF-8 characters is used to encode a single Unicode character.

Note: UTF8CharLength always returns 1, if the given character is not a valid UTF-8 lead byte.

See Also

IsUTF8LeadByte (see page 6871)

UTF8CharSize (see page 6874)

71.1.15 WideStrUtils.UTF8CharSize Function

Returns the number of bytes that follow a lead UTF-8 byte.

Pascal

```
function UTF8CharSize(Lead: AnsiChar): Integer;
```

C++

```
int UTF8CharSize(AnsiChar Lead);
```

File

WideStrUtils

Description

Call UTF8CharSize to determine the number of bytes that follow a lead UTF-8 character (including the lead byte).

In UTF-8, each lead character can be followed by a variable number of other characters. This combination of UTF-8 characters is used to encode a single Unicode character.

Note: UTF8CharSize always returns 0, if the given character is not a valid UTF-8 lead byte.

See Also

IsUTF8LeadByte (see page 6871)

UTF8CharLength (see page 6874)

71.1.16 WideStrUtils.UTF8LowerCase Function

Returns the UTF-8 string converted to lower case.

Pascal

```
function UTF8LowerCase(const S: UTF8string): UTF8string;
```

C++

```
UTF8string UTF8LowerCase(const UTF8string S);
```

File

WideStrUtils

Description

UTF8LowerCase converts all characters in the given **UTF8String (see page 3804)** to lower case. The conversion is not locale specific.

See Also

UTF8UpperCase (see page 6876)

AnsiLowerCase

LowerCase

WideLowerCase (see page 4316)

71.1.17 WideStrUtils.Utf8ToAnsiEx Function

Converts a string encoded in UTF-8 to ANSI with a given code page.

Pascal

```
function Utf8ToAnsiEx(const S: UTF8String; const cp: integer): AnsiString;
```

C++

```
AnsiString Utf8ToAnsiEx(const UTF8String S, const int cp);
```

File

WideStrUtils

Description

Utf8ToAnsiEx converts the string specified by S to ANSI. The code page of the output string is specified by the `cp` parameter.

If the specified code page is `CP_UTF8` then a direct assignment is made; otherwise a conversion is performed.

See Also

AnsiToUtf8Ex (see page 6868)

AnsiToUtf8 (see page 3517)

Utf8ToAnsi (see page 3698)

71.1.18 WideStrUtils.UTF8UpperCase Function

Returns the UTF-8 string converted to uppercase.

Pascal

```
function UTF8UpperCase(const S: UTF8string): UTF8string;
```

C++

```
UTF8string UTF8UpperCase(const UTF8string S);
```

File

WideStrUtils

Description

UTF8UpperCase converts all characters in the given **UTF8String (see page 3804)** to uppercase. The conversion is not locale specific.

See Also

UTF8LowerCase (see page 6875)

AnsiUpperCase

UpperCase

WideUpperCase (see page 4318)

71.1.19 WideStrUtils.WideAdjustLineBreaks Function

Adjusts line break characters in a WideString (see page 3467) to reflect Windows or Linux style.

Pascal

```
function WideAdjustLineBreaks(const S: WideString; Style: TTextLineBreakStyle = tlbsCRLF):
WideString;
```

C++

```
BSTR WideAdjustLineBreaks(const BSTR S, TTextLineBreakStyle Style = tlbsCRLF);
```

File

WideStrUtils

Description

WideAdjustLineBreaks returns a text string with its line breaks adjusted to fit a specified style.

`S` is the WideString (see page 3467) to convert.

`Style` indicates whether the line breaks should all be Windows-style breaks with a carriage return and line feed (**tlbsCRLF**), or Linux-style breaks with a single line feed character (**tlbsLF**). If `Style` is omitted, the style for the compile target is used.

See Also

TrimLeft (see page 4301)

TrimRight (see page 4302)

71.1.20 WideStrUtils.WideDequotedStr Function

Converts a quoted WideString (see page 3467) into an unquoted WideString (see page 3467).

Pascal

```
function WideDequotedStr(const S: WideString; AQuote: WideChar): WideString;
```

C++

```
BSTR WideDequotedStr(const BSTR S, WideChar AQuote);
```

File

WideStrUtils

Description

WideDequotedStr removes the quotation marks from the beginning and end of a quoted string, and reduces pairs of quotation marks within the string to a single quotation mark. The `AQuote` parameter defines what character to use as a quotation mark. If the `S` parameter does not begin and end with the quotation mark, WideDequotedStr returns `S` unchanged.

See Also

WideQuotedStr (see page 6878)

WideExtractQuotedStr (see page 6877)

71.1.21 WideStrUtils.WideExtractQuotedStr Function

Converts a quoted PWideChar into an unquoted WideString (see page 3467).

Pascal

```
function WideExtractQuotedStr(var Src: PWideChar; Quote: WideChar): WideString;
```

C++

```
BSTR WideExtractQuotedStr(PWideChar Src, WideChar Quote);
```

File

WideStrUtils

Description

WideExtractQuotedStr removes the quotation marks from the beginning and end of a quoted string, and reduces pairs of quotation marks within the string to a single quotation mark. The `Quote` parameter defines what character to use as a quotation mark. If the first character in `Src` is not the value of the `Quote` parameter, WideExtractQuotedStr returns an empty string.

The function copies characters from `Src` to the result string until the second solitary quotation mark or the first null character in `Src`. The `Src` parameter is updated to point to the first character following the quoted string. If `Src` does not contain a matching end quotation mark, the `Src` parameter is updated to point to the terminating null character.

See Also

WideQuotedStr (see page 6878)

71.1.22 WideStrUtils.WideLastChar Function

Gets last character in a WideString (see page 3467).

Pascal

```
function WideLastChar(const S: WideString): PWideChar;
```

C++

```
PWideChar WideLastChar(const BSTR S);
```

File

WideStrUtils

Description

WideLastChar returns a pointer to the last character in the **WideString (see page 3467)**.

See Also

WideCompareStr (see page 4313)

WideCompareText (see page 4313)

71.1.23 WideStrUtils.WideQuotedStr Function

Returns the quoted version of a WideString (see page 3467).

Pascal

```
function WideQuotedStr(const S: WideString; Quote: WideChar): WideString;
```

C++

```
BSTR WideQuotedStr(const BSTR S, WideChar Quote);
```

File

WideStrUtils

Description

Use WideQuotedStr to convert a string (S) to a quoted string, using the provided `Quote` character. A `Quote` character is inserted at the beginning and end of `S`, and each `Quote` character in the string is doubled.

See Also

WideExtractQuotedStr (see page 6877)

71.1.24 WideStrUtils.WideReplaceStr Function

Replaces all occurrences of a substring within a string, using case-sensitive search.

Pascal

```
function WideReplaceStr(const AText: WideString; const AFromText: WideString; const AToText:
WideString): WideString;
```

C++

```
BSTR WideReplaceStr(const BSTR AText, const BSTR AFromText, const BSTR AToText);
```

File

WideStrUtils

Description

Use WideReplaceStr to replace all occurrences of the substring `AFromText` with the string `AToText`, within the string `AText`. WideReplaceStr returns the string obtained after replacement.

Use WideReplaceText (see page 6880) to do a case-insensitive replacement.

Note: Both the parameters and the return value are of type `WideString` (see page 3467). To do the replacement in a `UnicodeString` (see page 3381) context, use the ReplaceStr function. Also, to do the replacement in an `AnsiString` context, use the ReplaceStr function.

Note: Recursive replacement of substrings is not supported. This means that if the substitution of `AToText` results in a new match for `AFromText`, that match is not replaced.

The WideStringReplace (see page 6880) function is a more general replacement routine, giving case-sensitivity options, as well as the option to decide whether to replace all occurrences or the first occurrence of the substring in the string.

See Also

ReplaceStr (see page 3044)

ReplaceStr

WStrPos (see page 6887)

WideReplaceText (see page 6880)

WideStringReplace (see page 6880)

71.1.25 WideStrUtils.WideReplaceText Function

Replaces all occurrences of a substring in a WideString (see page 3467).

Pascal

```
function WideReplaceText(const AText: WideString; const AFromText: WideString; const AToText:
WideString): WideString;
```

C++

```
BSTR WideReplaceText(const BSTR AText, const BSTR AFromText, const BSTR AToText);
```

File

WideStrUtils

Description

WideReplaceText replaces all occurrences of a substring in a **WideString (see page 3467)** with another substring without case sensitivity.

Use WideReplaceStr (see page 6879) to make this replacement with case sensitivity.

Note: Recursion substring replacement is not supported.

See Also

WideReplaceStr (see page 6879)

WideStringReplace (see page 6880)

71.1.26 WideStrUtils.WideStringReplace Function

Replaces occurrences of a substring within a string.

Pascal

```
function WideStringReplace(const S: Widestring; const OldPattern: Widestring; const
NewPattern: Widestring; Flags: TReplaceFlags): Widestring;
```

C++

```
BSTR WideStringReplace(const BSTR S, const BSTR OldPattern, const BSTR NewPattern,
TReplaceFlags Flags);
```

File

WideStrUtils

Description

WideStringReplace replaces occurrences of the substring specified by `OldPattern` with the substring specified by `NewPattern` in the string `S`.

`Flags` is a TReplaceFlags (see page 4343) type parameter. If `rfIgnoreCase` is set, the replacement is case-sensitive; otherwise case is ignored. If `rfReplaceAll` is on, all occurrences of `OldPattern` are replaced; if not, only the first occurrence is replaced.

Note: The parameters `S`, `OldPattern`, `NewPattern`, and the return value are of type `WideString` (see page 3467). To do the replacement in a `UnicodeString` (see page 3381) context, use the StringReplace (see page 4253)

function. Also, to do the replacement in an `AnsiString` context, use the StringReplace (see page 4253) function.

Note: Recursive replacement of substrings is not supported. This means that if the substitution of `OldPattern` results in a new match for `NewPattern`, that match is not replaced.

To replace all occurrences of the substring within the string, you may also use the WideReplaceStr (see page 6879) function to do a case-sensitive search, or WideReplaceText (see page 6880) to do a case-insensitive search.

See Also

TReplaceFlags (see page 4343)

StringReplace (see page 4253)

StringReplace

WideReplaceStr (see page 6879)

WideReplaceText (see page 6880)

WStrPos (see page 6887)

71.1.27 WideStrUtils.WStrAlloc Function

Allocates a buffer for a PWideChar and returns a pointer to its first character.

Pascal

```
function WStrAlloc(Size: Cardinal): PWideChar;
```

C++

```
PWideChar WStrAlloc(unsigned Size);
```

File

WideStrUtils

Description

WStrAlloc allocates a buffer for a PWideChar with a maximum length of `Size - 1` (1 byte must be reserved for the termination character). The result points to the location where the first character of the string is to be stored. A 32-bit number giving the total amount of memory allocated is stored in the four bytes preceding the first character; it is equal to `Size * 2 + 4`. If space for a string is allocated with WStrAlloc, it should be deallocated via WStrDispose (see page 6883).

See Also

WStrDispose (see page 6883)

WStrNew (see page 6886)

WStrBufSize (see page 6881)

71.1.28 WideStrUtils.WStrBufSize Function

Returns the maximum number of characters that can be stored in a PWideChar allocated by WStrAlloc (see page 6881).

Pascal

```
function WStrBufSize(const Str: PWideChar): Cardinal;
```

C++

```
unsigned WStrBufSize(const PWideChar Str);
```

File

WideStrUtils

Description

WStrBufSize returns the number of characters in `Str` (see page 3676), including the byte for the termination character.

Note: If `Str` (see page 3676) does not point to a string buffer allocated by WStrAlloc (see page 6881) or WStrNew (see page 6886), no error message is returned, and the result is unpredictable.

See Also

WStrAlloc (see page 6881)

WStrNew (see page 6886)

71.1.29 WideStrUtils.WStrCat Function

Appends a PWideChar to another PWideChar.

Pascal

```
function WStrCat(Dest: PWideChar; const Source: PWideChar): PWideChar;
```

C++

```
PWideChar WStrCat(PWideChar Dest, const PWideChar Source);
```

File

WideStrUtils

Description

Use WStrCat to concatenate `Source` to the end of `Dest`. WStrCat does not perform any length check. The destination buffer must have room for at least `WStrLen` (see page 6885)`(Dest) + WStrLen` (see page 6885)`(Source) + 1` characters.

See Also

WStrLen (see page 6885)

71.1.30 WideStrUtils.WStrComp Function

Compares two PWideChar values with case sensitivity.

Pascal

```
function WStrComp(const Str1: PWideChar; const Str2: PWideChar): Integer;
```

C++

```
int WStrComp(const PWideChar Str1, const PWideChar Str2);
```

File

WideStrUtils

Description

Call WStrComp to compare two PWideChar strings, with case sensitivity. The return value is indicated in the following table.

Return value	Condition
<0	`Str1` sorts before `Str2`
=0	`Str1` is the same as `Str2`
>0	`Str1` sorts after `Str2`

See Also

WideCompareStr (see page 4313)

WideSameStr (see page 4316)

WideSameText (see page 4316)

71.1.31 WideStrUtils.WStrCopy Function

Copies a PWideChar to another PWideChar.

Pascal

```
function WStrCopy(Dest: PWideChar; const Source: PWideChar): PWideChar;
```

C++

```
PWideChar WStrCopy(PWideChar Dest, const PWideChar Source);
```

File

WideStrUtils

Description

Use WStrCopy to copy `Source` to `Dest`. WStrCopy returns `Dest`.

WStrCopy does not perform any length check. The destination buffer must have room for at least `WStrLen (see page 6885)(Source) + 1` characters.

For length checking, use the WStrLCopy (see page 6884) function.

See Also

WStrLCopy (see page 6884)

WStrPCopy (see page 6886)

WStrPLCopy (see page 6887)

71.1.32 WideStrUtils.WStrDispose Function

Disposes of a PWideChar.

Pascal

```
procedure WStrDispose(Str: PWideChar);
```

C++

```
WStrDispose(PWideChar Str);
```

File

WideStrUtils

Description

WStrDispose is provided for backward compatibility. WStrDispose disposes of a PWideChar that was previously allocated with WStrAlloc (see page 6881) or WStrNew (see page 6886).

If `Str (see page 3676)` is **nil** (Delphi) or **NULL** (C++), WStrDispose does nothing.

See Also

WStrAlloc (see page 6881)

WStrNew (see page 6886)

71.1.33 WideStrUtils.WStrEnd Function

Returns a pointer to the end of a PWideChar.

Pascal

```
function WStrEnd(const Str: PWideChar): PWideChar;
```

C++

```
PWideChar WStrEnd(const PWideChar Str);
```

File

WideStrUtils

Description

WStrEnd returns a pointer to the null character at the end of `Str (see page 3676)`.

See Also

WStrLen (see page 6885)

71.1.34 WideStrUtils.WStrLCopy Function

Copies up to a specified maximum number of characters from a PWideChar.

Pascal

```
function WStrLCopy(Dest: PWideChar; const Source: PWideChar; MaxLen: Cardinal): PWideChar;
```

C++

```
PWideChar WStrLCopy(PWideChar Dest, const PWideChar Source, unsigned MaxLen);
```

File

WideStrUtils

Description

WStrLCopy copies at most `MaxLen` characters from `Source` to `Dest`, then adds a null terminator to `Dest` and returns it. The **SizeOf (see page 3670)** standard function (Delphi) or the **sizeof** operator (C++) can be used to determine the `MaxLen` parameter. Usually, `MaxLen` equals `((SizeOf (see page 3670)(Dest) div 2) - 1`.

Note: WStrLCopy copies a null terminator as well as `MaxLen` characters. Be sure that `Dest` is large enough to accommodate the null terminator at the end.

See Also

SizeOf (see page 3670)

WStrCopy (see page 6883)

WStrLen (see page 6885)

71.1.35 WideStrUtils.WStrLen Function

Returns number of characters in a PWideChar excluding the null terminator.

Pascal

```
function WStrLen(const Str: PWideChar): Cardinal;
```

C++

```
unsigned WStrLen(const PWideChar Str);
```

File

WideStrUtils

Description

WStrLen returns the number of characters in `Str (see page 3676)`, not counting the null terminator.

See Also

WStrEnd (see page 6884)

71.1.36 WideStrUtils.WStrMove Function

Copies a specified number of characters from a PWideChar to another PWideChar.

Pascal

```
function WStrMove(Dest: PWideChar; const Source: PWideChar; Count: Cardinal): PWideChar;
```

C++

```
PWideChar WStrMove(PWideChar Dest, const PWideChar Source, unsigned Count);
```

File

WideStrUtils

Description

WStrMove copies exactly `Count` characters from `Source` to `Dest` and returns `Dest`. `Source` and `Dest` can overlap.

See Also

Move (see page 3617)

71.1.37 WideStrUtils.WStrNew Function

Allocates space on and copies a PWideChar to the heap, returning a new PWideChar.

Pascal

```
function WStrNew(const Str: PWideChar): PWideChar;
```

C++

```
PWideChar WStrNew(const PWideChar Str);
```

File

WideStrUtils

Description

WStrNew allocates a copy of `Str (see page 3676)` on the heap.

If `Str (see page 3676)` is **nil** (Delphi) or **NULL** (C++) or points to an empty string, WStrNew returns a pointer to a new empty PWideChar.

Otherwise, WStrNew makes a duplicate of `Str (see page 3676)`, obtaining space with a call to WStrAlloc (see page 6881), and returns a pointer to the duplicated PWideChar.

The allocated space is the length of `Str (see page 3676) * 2 + 5` bytes.

See Also

GetMem (see page 3584)

WStrDispose (see page 6883)

WStrAlloc (see page 6881)

71.1.38 WideStrUtils.WStrPCopy Function

Copies a WideString (see page 3467) to a PWideChar.

Pascal

```
function WStrPCopy(Dest: PWideChar; const Source: WideString): PWideChar;
```

C++

```
PWideChar WStrPCopy(PWideChar Dest, const BSTR Source);
```

File

WideStrUtils

Description

WStrPCopy copies `Source` into a PWideChar `Dest`. It returns a pointer to `Dest`.

WStrPCopy does not perform any length checking.

The destination buffer must have room for at least `Length (` see page 3607`)(Source) + 1` characters.

See Also

WStrCopy (see page 6883)

71.1.39 WideStrUtils.WStrPLCopy Function

Copies characters from a WideString (see page 3467) into a PWideChar.

Pascal

```
function WStrPLCopy(Dest: PWideChar; const Source: WideString; MaxLen: Cardinal): PWideChar;
```

C++

```
PWideChar WStrPLCopy(PWideChar Dest, const BSTR Source, unsigned MaxLen);
```

File

WideStrUtils

Description

WStrPLCopy copies up to a maximum of `MaxLen` characters from `Source` into the PWideChar `Dest`. WStrPLCopy returns a pointer to `Dest`.

Note: `Dest` should point to a location with room for `MaxLen + 1` characters, including the null terminator.

See Also

WStrPCopy (see page 6886)

WStrLCopy (see page 6884)

71.1.40 WideStrUtils.WStrPos Function

Returns a pointer to the first occurrence of a PWideChar in another PWideChar.

Pascal

```
function WStrPos(const Str1: PWideChar; const Str2: PWideChar): PWideChar;
```

C++

```
PWideChar WStrPos(const PWideChar Str1, const PWideChar Str2);
```

File

WideStrUtils

Description

WStrPos returns a pointer to the first occurrence of `Str2` in `Str1`. If `Str2` does not occur in `Str1`, WStrPos returns **nil** (Delphi) or **NULL** (C++).

See Also

AnsiStrPos (see page 4060)

StrPos (see page 4272)

Pos (see page 3627)

71.1.41 WideStrUtils.WStrScan Function

Returns a pointer to the first occurrence of a specified character in a PWideChar.

Pascal

```
function WStrScan(Str: PWideChar; Chr: WideChar): PWideChar;
```

C++

```
PWideChar WStrScan(PWideChar Str, WideChar Chr);
```

File

WideStrUtils

Description

WStrScan returns a pointer to the first occurrence of `Chr` (see page 3536) in `Str` (see page 3676). If `Chr` (see page 3536) does not occur in `Str` (see page 3676), WStrScan returns **nil** (Delphi) or **NULL** (C++). The null terminator is considered to be part of the string.

See Also

AnsiStrScan (see page 4061)

StrScan (see page 4275)

71.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TEncodeType (see page 6889)	Defines a list of encodings.

71.2.1 WideStrUtils.TEncodeType Enumeration

Defines a list of encodings.

Pascal

```
TEncodeType = (
  etUSASCII,
  etUTF8,
  etANSI
);
```

C++

```
enum TEncodeType {
  etUSASCII,
  etUTF8,
  etANSI
};
```

File

WideStrUtils

Description

TEncodeType defines a list of the most used encodings. There are three encodings defined:

Value	Meaning
etUSASCII	The standard USA ASCII encoding.
etUTF8	The UTF-8 encoding.
etANSI	The standard ANSI encoding.

See Also

DetectUTF8Encoding (see page 6869)

TEncoding (see page 3969)

CharSet (see page 6890)

71.3 Types

The following table lists types in this documentation.

Types

Name	Description
CharSet (see page 6890)	Represents a set of allowed characters.

71.3.1 WideStrUtils.CharSet Type

Represents a set of allowed characters.

Pascal

```
CharSet = set of ansichar;
```

C++

```
ansichar CharSet;
```

File

WideStrUtils

Description

CharSet is used to identify the allowed characters in a character set. Several routines in WideStrUtils (see page 6865) unit require such a parameter.

See Also

TEncoding (see page 3969)

inOpArray (see page 6871)

inOpSet (see page 6871)

71.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sUTF8BOMString (see page 6891)	Represents the UTF-8 byte-order mark.

71.4.1 WideStrUtils.sUTF8BOMString Constant

Represents the UTF-8 byte-order mark.

Pascal

```
sUTF8BOMString: array[1..3] of AnsiChar = (AnsiChar(#$EF), AnsiChar(#$BB), AnsiChar(#$BF));
```

C++

```
array[1..3] of AnsiChar sUTF8BOMString = (AnsiChar(#$EF), AnsiChar(#$BB), AnsiChar(#$BF));
```

File

WideStrUtils

Description

sUTF8BOMString is a 3 byte array that represents the UTF-8 byte-order mark. The byte-order mark is used to detect if a given text is UTF-8 encoded.

To manually check if a given text is UTF-8 encoded, check whether the first 3 bytes match the ones defined by sUTF8BOMString constant.

See Also

HasUTF8BOM (see page 6870)

IsUTF8String (see page 6872)

72 WinHelpViewer Namespace

This is namespace WinHelpViewer.

Topics

Name	Description
Interfaces (see page 6894)	The following table lists interfaces in this documentation.
Variables (see page 6897)	The following table lists variables in this documentation.

Interfaces

	Name	Description
	IWinHelpTester (see page 6894)	This is class WinHelpViewer.IWinHelpTester.

Variables

Name	Description
ViewerName (see page 6897)	This is variable WinHelpViewer.ViewerName.
WinHelpTester (see page 6897)	This is variable WinHelpViewer.WinHelpTester.

72.1 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWinHelpTester (see page 6894)	This is class WinHelpViewer.IWinHelpTester.

72.1.1 IWinHelpTester Interface

Class Hierarchy

Pascal

```
IWinHelpTester = interface(IInterface);
```

C++

```
__interface IWinHelpTester : public IInterface;
```

File

WinHelpViewer

Description

This is class WinHelpViewer.IWinHelpTester.

72.1.1.1 IWinHelpTester Members

The following tables list the members exposed by IWinHelpTester.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	CanShowALink (see page 6895)	This is CanShowALink, a member of class IWinHelpTester.
	CanShowContext (see page 6895)	This is CanShowContext, a member of class IWinHelpTester.
	CanShowTopic (see page 6895)	This is CanShowTopic, a member of class IWinHelpTester.
	GetDefaultHelpFile (see page 6895)	This is GetDefaultHelpFile, a member of class IWinHelpTester.
	GetHelpPath (see page 6896)	This is GetHelpPath, a member of class IWinHelpTester.
	GetHelpStrings (see page 6896)	This is GetHelpStrings, a member of class IWinHelpTester.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

72.1.1.2 IWinHelpTester Methods

The methods of the IWinHelpTester class are listed here.

Public Methods

	Name	Description
	CanShowALink (see page 6895)	This is CanShowALink, a member of class IWinHelpTester.
	CanShowContext (see page 6895)	This is CanShowContext, a member of class IWinHelpTester.
	CanShowTopic (see page 6895)	This is CanShowTopic, a member of class IWinHelpTester.
	GetDefaultHelpFile (see page 6895)	This is GetDefaultHelpFile, a member of class IWinHelpTester.
	GetHelpPath (see page 6896)	This is GetHelpPath, a member of class IWinHelpTester.
	GetHelpStrings (see page 6896)	This is GetHelpStrings, a member of class IWinHelpTester.

72.1.1.2.1 IWinHelpTester.CanShowALink Method

Pascal

```
function CanShowALink(const ALink: String; const FileName: String): Boolean;
```

C++

```
__fastcall Boolean CanShowALink(const UnicodeString ALink, const UnicodeString FileName);
```

Description

This is CanShowALink, a member of class IWinHelpTester.

72.1.1.2.2 IWinHelpTester.CanShowContext Method

Pascal

```
function CanShowContext(const Context: Integer; const FileName: String): Boolean;
```

C++

```
__fastcall Boolean CanShowContext(const int Context, const UnicodeString FileName);
```

Description

This is CanShowContext, a member of class IWinHelpTester.

72.1.1.2.3 IWinHelpTester.CanShowTopic Method

Pascal

```
function CanShowTopic(const Topic: String; const FileName: String): Boolean;
```

C++

```
__fastcall Boolean CanShowTopic(const UnicodeString Topic, const UnicodeString FileName);
```

Description

This is CanShowTopic, a member of class IWinHelpTester.

72.1.1.2.4 IWinHelpTester.GetDefaultHelpFile Method

Pascal

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
function GetDefaultHelpFile: String;
```

C++

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
__fastcall UnicodeString GetDefaultHelpFile();
```

Description

This is GetDefaultHelpFile, a member of class IWinHelpTester.

72.1.1.2.5 IWinHelpTester.GetHelpPath Method

Pascal

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
function GetHelpPath: String;
```

C++

```
[FileIOPermission(SecurityAction.Demand, Unrestricted=True)]
__fastcall UnicodeString GetHelpPath();
```

Description

This is GetHelpPath, a member of class IWinHelpTester.

72.1.1.2.6 IWinHelpTester.GetHelpStrings Method

Pascal

```
function GetHelpStrings(const ALink: String): TStringList;
```

C++

```
__fastcall TStringList GetHelpStrings(const UnicodeString ALink);
```

Description

This is GetHelpStrings, a member of class IWinHelpTester.

72.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ViewerName (see page 6897)	This is variable WinHelpViewer.ViewerName.
WinHelpTester (see page 6897)	This is variable WinHelpViewer.WinHelpTester.

72.2.1 WinHelpViewer.ViewerName Variable

Pascal

```
ViewerName: String;
```

C++

```
UnicodeString ViewerName;
```

File

WinHelpViewer

Description

This is variable WinHelpViewer.ViewerName.

72.2.2 WinHelpViewer.WinHelpTester Variable

Pascal

```
WinHelpTester: IWinHelpTester;
```

C++

```
IWinHelpTester WinHelpTester;
```

File

WinHelpViewer

Description

This is variable WinHelpViewer.WinHelpTester.

73 WSDLBind Namespace

This is namespace WSDLBind.

Topics

Name	Description
Classes (see page 6902)	The following table lists classes in this documentation.
Interfaces (see page 7078)	The following table lists interfaces in this documentation.
Functions (see page 7171)	The following table lists functions in this documentation.
Constants (see page 7173)	The following table lists constants in this documentation.

Classes

	Name	Description
	TBinding (see page 6902)	This is class WSDLBind.TBinding.
	TBindingOperation (see page 6909)	This is class WSDLBind.TBindingOperation.
	TBindingOperations (see page 6917)	This is class WSDLBind.TBindingOperations.
	TBindings (see page 6923)	This is class WSDLBind.TBindings.
	TDefinition (see page 6930)	This is class WSDLBind.TDefinition.
	TDocumentation (see page 6938)	This is class WSDLBind.TDocumentation.
	TDocumentationCollection (see page 6939)	This is class WSDLBind.TDocumentationCollection.
	TDocumented (see page 6945)	This is class WSDLBind.TDocumented.
	TFault (see page 6951)	This is class WSDLBind.TFault.
	TFaults (see page 6952)	This is class WSDLBind.TFaults.
	TImport (see page 6958)	This is class WSDLBind.TImport.
	TImports (see page 6965)	This is class WSDLBind.TImports.
	TMessage (see page 6971)	This is class WSDLBind.TMessage.
	TMessages (see page 6978)	This is class WSDLBind.TMessages.
	TOperation (see page 6984)	This is class WSDLBind.TOperation.
	TOperations (see page 6991)	This is class WSDLBind.TOperations.
	TParam (see page 6998)	This is class WSDLBind.TParam.
	TPart (see page 7005)	This is class WSDLBind.TPart.
	TParts (see page 7012)	This is class WSDLBind.TParts.
	TPort (see page 7018)	This is class WSDLBind.TPort.
	TPorts (see page 7025)	This is class WSDLBind.TPorts.
	TPortType (see page 7031)	This is class WSDLBind.TPortType.
	TPortTypes (see page 7038)	This is class WSDLBind.TPortTypes.
	TService (see page 7044)	This is class WSDLBind.TService.

	TServices (see page 7051)	This is class WSDLBind.TServices.
	TTypes (see page 7057)	This is class WSDLBind.TTypes.
	TWSDLDocument (see page 7063)	TWSDLDocument represents a WSDL document.
	TXMLSchemaDefs (see page 7071)	This is class WSDLBind.TXMLSchemaDefs.

Constants

Name	Description
SBinding (see page 7173)	This is constant WSDLBind.SBinding.
SDefinitions (see page 7173)	This is constant WSDLBind.SDefinitions.
SFault (see page 7174)	This is constant WSDLBind.SFault.
SImport (see page 7174)	This is constant WSDLBind.SImport.
SInput (see page 7174)	This is constant WSDLBind.SInput.
SLocation (see page 7175)	This is constant WSDLBind.SLocation.
SMessage (see page 7175)	This is constant WSDLBind.SMessage.
SNamespace (see page 7175)	This is constant WSDLBind.SNamespace.
SOperation (see page 7176)	This is constant WSDLBind.SOperation.
SOutput (see page 7176)	This is constant WSDLBind.SOutput.
SPart (see page 7176)	This is constant WSDLBind.SPart.
SPort (see page 7177)	This is constant WSDLBind.SPort.
SPortType (see page 7177)	This is constant WSDLBind.SPortType.
SService (see page 7177)	This is constant WSDLBind.SService.
SType (see page 7178)	This is constant WSDLBind.SType.
STypes (see page 7178)	This is constant WSDLBind.STypes.

Functions

	Name	Description
	GetDefinition (see page 7171)	This is function WSDLBind.GetDefinition.
	LoadWSDLDoc (see page 7171)	This is function WSDLBind.LoadWSDLDoc.
	NewWSDLDoc (see page 7171)	This is function WSDLBind.NewWSDLDoc.

Interfaces

	Name	Description
	IBinding (see page 7078)	This is class WSDLBind.IBinding.
	IBindingInput (see page 7082)	This is class WSDLBind.IBindingInput.
	IBindingOperation (see page 7083)	This is class WSDLBind.IBindingOperation.
	IBindingOperations (see page 7087)	This is class WSDLBind.IBindingOperations.
	IBindingOutput (see page 7090)	This is class WSDLBind.IBindingOutput.
	IBindings (see page 7091)	This is class WSDLBind.IBindings.
	IDefinition (see page 7094)	This is class WSDLBind.IDefinition.
	IDocumentation (see page 7100)	This is class WSDLBind.IDocumentation.
	IDocumentationCollection (see page 7101)	This is class WSDLBind.IDocumentationCollection.
	IDocumented (see page 7104)	This is class WSDLBind.IDocumented.
	IFault (see page 7106)	This is class WSDLBind.IFault.
	IFaults (see page 7107)	This is class WSDLBind.IFaults.
	IImport (see page 7110)	This is class WSDLBind.IImport.
	IImports (see page 7113)	This is class WSDLBind.IImports.
	IMessage (see page 7116)	This is class WSDLBind.IMessage.
	IMessages (see page 7120)	This is class WSDLBind.IMessages.
	IOperation (see page 7123)	This is class WSDLBind.IOperation.
	IOperations (see page 7127)	This is class WSDLBind.IOperations.
	IParam (see page 7131)	This is class WSDLBind.IParam.
	IPart (see page 7134)	This is class WSDLBind.IPart.
	IParts (see page 7139)	This is class WSDLBind.IParts.
	IPort (see page 7142)	This is class WSDLBind.IPort.
	IPorts (see page 7145)	This is class WSDLBind.IPorts.

	IPortType (see page 7148)	This is class WSDLBind.IPortType.
	IPortTypes (see page 7152)	This is class WSDLBind.IPortTypes.
	IService (see page 7155)	This is class WSDLBind.IService.
	IServices (see page 7158)	This is class WSDLBind.IServices.
	ITypes (see page 7161)	This is class WSDLBind.ITypes.
	IWSDLDocument (see page 7164)	IWSDLDocument provides low-level access to the definitions in a WSDL document.
	IXMLSchemaDefs (see page 7168)	This is class WSDLBind.IXMLSchemaDefs.

73.1 Classes

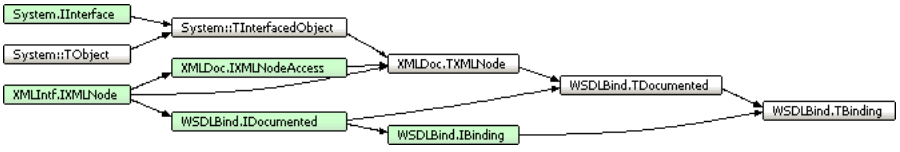
The following table lists classes in this documentation.

Classes

	Name	Description
	TBinding (see page 6902)	This is class WSDLBind.TBinding.
	TBindingOperation (see page 6909)	This is class WSDLBind.TBindingOperation.
	TBindingOperations (see page 6917)	This is class WSDLBind.TBindingOperations.
	TBindings (see page 6923)	This is class WSDLBind.TBindings.
	TDefinition (see page 6930)	This is class WSDLBind.TDefinition.
	TDocumentation (see page 6938)	This is class WSDLBind.TDocumentation.
	TDocumentationCollection (see page 6939)	This is class WSDLBind.TDocumentationCollection.
	TDocumented (see page 6945)	This is class WSDLBind.TDocumented.
	TFault (see page 6951)	This is class WSDLBind.TFault.
	TFaults (see page 6952)	This is class WSDLBind.TFaults.
	TImport (see page 6958)	This is class WSDLBind.TImport.
	TImports (see page 6965)	This is class WSDLBind.TImports.
	TMessage (see page 6971)	This is class WSDLBind.TMessage.
	TMessages (see page 6978)	This is class WSDLBind.TMessages.
	TOperation (see page 6984)	This is class WSDLBind.TOperation.
	TOperations (see page 6991)	This is class WSDLBind.TOperations.
	TParam (see page 6998)	This is class WSDLBind.TParam.
	TPart (see page 7005)	This is class WSDLBind.TPart.
	TParts (see page 7012)	This is class WSDLBind.TParts.
	TPort (see page 7018)	This is class WSDLBind.TPort.
	TPorts (see page 7025)	This is class WSDLBind.TPorts.
	TPortType (see page 7031)	This is class WSDLBind.TPortType.
	TPortTypes (see page 7038)	This is class WSDLBind.TPortTypes.
	TService (see page 7044)	This is class WSDLBind.TService.
	TServices (see page 7051)	This is class WSDLBind.TServices.
	TTypes (see page 7057)	This is class WSDLBind.TTypes.
	TWSDLDocument (see page 7063)	TWSDLDocument represents a WSDL document.
	TXMLSchemaDefs (see page 7071)	This is class WSDLBind.TXMLSchemaDefs.

73.1.1 TBinding Class

Class Hierarchy



Pascal

```
TBinding = class(TDocumented, IBinding);
```

C++

```
class TBinding : public TDocumented, public IBinding;
```

File

WSDLBind

Description

This is class WSDLBind.TBinding.

73.1.1.1 TBinding Members

The following tables list the members exposed by TBinding.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.

	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindingOperations (see page 6908)	This is GetBindingOperations, a member of class TBinding.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 6908)	This is GetName, a member of class TBinding.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetType_ (see page 6909)	This is GetType_, a member of class TBinding.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 6909)	This is SetName, a member of class TBinding.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	SetType_ (see page 6909)	This is SetType_, a member of class TBinding.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 6908)	Responds after the last constructor executes.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.

	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.

	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	BindingOperations (see page 7082)	This is BindingOperations, a member of class IBinding.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7082)	This is Name, a member of class IBinding.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
	Type_ (see page 7082)	This is Type_, a member of class IBinding.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.1.2 TBinding Methods

The methods of the TBinding class are listed here.

Protected Methods

	Name	Description
	GetBindingOperations (see page 6908)	This is GetBindingOperations, a member of class TBinding.
	GetName (see page 6908)	This is GetName, a member of class TBinding.
	GetType_ (see page 6909)	This is GetType_, a member of class TBinding.
	SetName (see page 6909)	This is SetName, a member of class TBinding.
	SetType_ (see page 6909)	This is SetType_, a member of class TBinding.

Public Methods

	Name	Description
	AfterConstruction (see page 6908)	Responds after the last constructor executes.

73.1.1.2.1 TBinding.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.1.2.2 TBinding.GetBindingOperations Method

Pascal

```
function GetBindingOperations: IBindingOperations;
```

C++

```
__fastcall IBindingOperations GetBindingOperations();
```

Description

This is GetBindingOperations, a member of class TBinding.

73.1.1.2.3 TBinding.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TBinding.

73.1.1.2.4 TBinding.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class TBinding.

73.1.1.2.5 TBinding.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TBinding.

73.1.1.2.6 TBinding.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

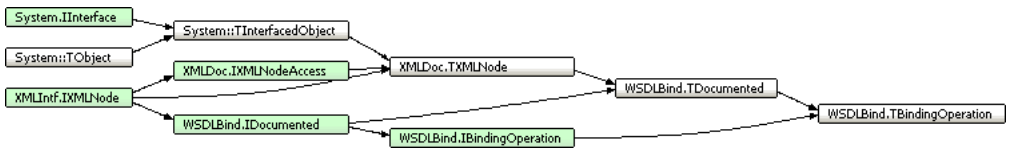
```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class TBinding.

73.1.2 TBindingOperation Class

Class Hierarchy



Pascal

```
TBindingOperation = class(TDocumented, IBindingOperation);
```

C++

```
class TBindingOperation : public TDocumented, public IBindingOperation;
```

File

WSDLBind

Description

This is class WSDLBind.TBindingOperation.

73.1.2.1 TBindingOperation Members

The following tables list the members exposed by TBindingOperation.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.

	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetFaults (see page 6915)	This is GetFaults, a member of class TBindingOperation.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetInput (see page 6916)	This is GetInput, a member of class TBindingOperation.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 6916)	This is GetName, a member of class TBindingOperation.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOutput (see page 6916)	This is GetOutput, a member of class TBindingOperation.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.

	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 6916)	This is SetName, a member of class TBindingOperation.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 6915)	Responds after the last constructor executes.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.

	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.

	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	Faults (see page 7086)	This is Faults, a member of class IBindingOperation.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	Input (see page 7086)	This is Input, a member of class IBindingOperation.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7087)	This is Name, a member of class IBindingOperation.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Output (see page 7087)	This is Output, a member of class IBindingOperation.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.2.2 TBindingOperation Methods

The methods of the TBindingOperation class are listed here.

Protected Methods

	Name	Description
	GetFaults (see page 6915)	This is GetFaults, a member of class TBindingOperation.
	GetInput (see page 6916)	This is GetInput, a member of class TBindingOperation.
	GetName (see page 6916)	This is GetName, a member of class TBindingOperation.
	GetOutput (see page 6916)	This is GetOutput, a member of class TBindingOperation.
	SetName (see page 6916)	This is SetName, a member of class TBindingOperation.

Public Methods

	Name	Description
	AfterConstruction (see page 6915)	Responds after the last constructor executes.

73.1.2.2.1 TBindingOperation.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.2.2.2 TBindingOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class TBindingOperation.

73.1.2.2.3 TBindingOperation.GetInput Method

Pascal

```
function GetInput: IBindingInput;
```

C++

```
__fastcall IBindingInput GetInput();
```

Description

This is GetInput, a member of class TBindingOperation.

73.1.2.2.4 TBindingOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TBindingOperation.

73.1.2.2.5 TBindingOperation.GetOutput Method

Pascal

```
function GetOutput: IBindingOutput;
```

C++

```
__fastcall IBindingOutput GetOutput();
```

Description

This is GetOutput, a member of class TBindingOperation.

73.1.2.2.6 TBindingOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

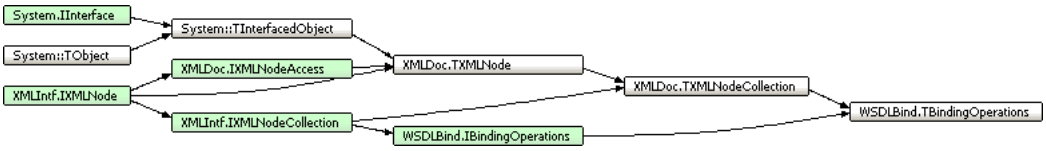
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TBindingOperation.

73.1.3 TBindingOperations Class

Class Hierarchy



Pascal

```
TBindingOperations = class(TXMLNodeCollection, IBindingOperations);
```

C++

```
class TBindingOperations : public TXMLNodeCollection, public IBindingOperations;
```

File

WSDLBind

Description

This is class WSDLBind.TBindingOperations.

73.1.3.1 TBindingOperations Members

The following tables list the members exposed by TBindingOperations.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 6922)	This is Add, a member of class TBindingOperations.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindingOperation (see page 6923)	This is GetBindingOperation, a member of class TBindingOperations.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.

	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 6923)	This is Insert, a member of class TBindingOperations.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.

R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.

	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	BindingOperation (see page 7090)	This is BindingOperation, a member of class IBindingOperations.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.3.2 TBindingOperations Methods

The methods of the TBindingOperations class are listed here.

Protected Methods

	Name	Description
	Add (see page 6922)	This is Add, a member of class TBindingOperations.
	GetBindingOperation (see page 6923)	This is GetBindingOperation, a member of class TBindingOperations.
	Insert (see page 6923)	This is Insert, a member of class TBindingOperations.

73.1.3.2.1 TBindingOperations.Add Method

Pascal

```
function Add(const Name: WideString): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Add(const BSTR Name);
```

Description

This is Add, a member of class TBindingOperations.

73.1.3.2.2 TBindingOperations.GetBindingOperation Method

Pascal

```
function GetBindingOperation(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperation(int Index);
```

Description

This is GetBindingOperation, a member of class TBindingOperations.

73.1.3.2.3 TBindingOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IBindingOperation;
```

C++

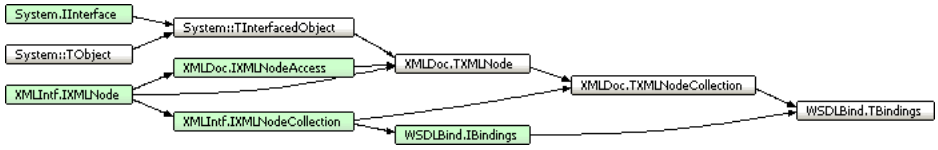
```
__fastcall IBindingOperation Insert(int Index);
```

Description

This is Insert, a member of class TBindingOperations.

73.1.4 TBindings Class

Class Hierarchy



Pascal

```
TBindings = class(TXMLNodeCollection, IBindings);
```

C++

```
class TBindings : public TXMLNodeCollection, public IBindings;
```

File

WSDLBind

Description

This is class WSDLBind.TBindings.

73.1.4.1 TBindings Members

The following tables list the members exposed by TBindings.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 6929)	This is Add, a member of class TBindings.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.

	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBinding (see page 6929)	This is GetBinding, a member of class TBindings.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 6929)	This is Insert, a member of class TBindings.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.

	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
	Count (see page 7457)	Indicates the number of repeating child nodes.
	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.

	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.

	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	Binding (see page 7094)	This is Binding, a member of class IBindings.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.

	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.4.2 TBindings Methods

The methods of the TBindings class are listed here.

Protected Methods

	Name	Description
	Add (see page 6929)	This is Add, a member of class TBindings.
	GetBinding (see page 6929)	This is GetBinding, a member of class TBindings.
	Insert (see page 6929)	This is Insert, a member of class TBindings.

73.1.4.2.1 TBindings.Add Method

Pascal

```
function Add(const Name: WideString; const AType: WideString): IBinding;
```

C++

```
__fastcall IBinding Add(const BSTR Name, const BSTR AType);
```

Description

This is Add, a member of class TBindings.

73.1.4.2.2 TBindings.GetBinding Method

Pascal

```
function GetBinding(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding GetBinding(int Index);
```

Description

This is GetBinding, a member of class TBindings.

73.1.4.2.3 TBindings.Insert Method

Pascal

```
function Insert(Index: Integer): IBinding;
```

C++

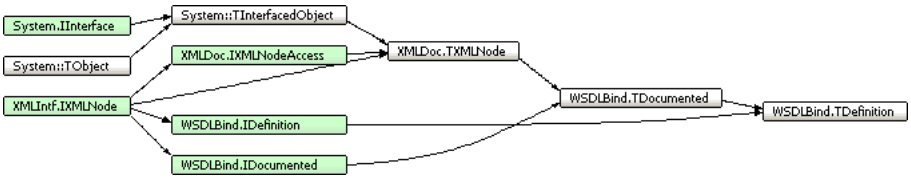
```
__fastcall IBinding Insert(int Index);
```

Description

This is Insert, a member of class TBindings.

73.1.5 TDefinition Class

Class Hierarchy



Pascal

```
TDefinition = class(TDocumented, IDefinition);
```

C++

```
class TDefinition : public TDocumented, public IDefinition;
```

File

WSDLBind

Description

This is class WSDLBind.TDefinition.

73.1.5.1 TDefinition Members

The following tables list the members exposed by TDefinition.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.

	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBindings (see page 6936)	This is GetBindings, a member of class TDefinition.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetImports (see page 6936)	This is GetImports, a member of class TDefinition.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetMessages (see page 6936)	This is GetMessages, a member of class TDefinition.
	GetName (see page 6936)	This is GetName, a member of class TDefinition.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPortTypes (see page 6937)	This is GetPortTypes, a member of class TDefinition.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetServices (see page 6937)	This is GetServices, a member of class TDefinition.
	GetTargetNamespace (see page 6937)	This is GetTargetNamespace, a member of class TDefinition.
	GetText (see page 7426)	Returns the text value of the node.
	GetTypes (see page 6937)	This is GetTypes, a member of class TDefinition.

	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 6938)	This is SetName, a member of class TDefinition.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetTargetNamespace (see page 6938)	This is SetTargetNamespace, a member of class TDefinition.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6935)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.

	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.

	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	Bindings (see page 7099)	This is Bindings, a member of class IDefinition.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.

	Imports (see page 7099)	This is Imports, a member of class IDefinition.
	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Messages (see page 7099)	This is Messages, a member of class IDefinition.
	Name (see page 7099)	This is Name, a member of class IDefinition.
	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7693)	Indicates the node name.
	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
	PortTypes (see page 7099)	This is PortTypes, a member of class IDefinition.
	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Services (see page 7100)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 7100)	This is TargetNamespace, a member of class IDefinition.
	Text (see page 7697)	Represents the Text of the node.
	Types (see page 7100)	This is Types, a member of class IDefinition.
	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.5.2 TDefinition Methods

The methods of the TDefinition class are listed here.

Protected Methods

	Name	Description
	GetBindings (see page 6936)	This is GetBindings, a member of class TDefinition.
	GetImports (see page 6936)	This is GetImports, a member of class TDefinition.
	GetMessages (see page 6936)	This is GetMessages, a member of class TDefinition.
	GetName (see page 6936)	This is GetName, a member of class TDefinition.
	GetPortTypes (see page 6937)	This is GetPortTypes, a member of class TDefinition.
	GetServices (see page 6937)	This is GetServices, a member of class TDefinition.
	GetTargetNamespace (see page 6937)	This is GetTargetNamespace, a member of class TDefinition.
	GetTypes (see page 6937)	This is GetTypes, a member of class TDefinition.
	SetName (see page 6938)	This is SetName, a member of class TDefinition.
	SetTargetNamespace (see page 6938)	This is SetTargetNamespace, a member of class TDefinition.

Public Methods

	Name	Description
	AfterConstruction (see page 6935)	Responds after the last constructor executes.

73.1.5.2.1 TDefinition.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.5.2.2 TDefinition.GetBindings Method

Pascal

```
function GetBindings: IBindings;
```

C++

```
__fastcall IBindings GetBindings();
```

Description

This is GetBindings, a member of class TDefinition.

73.1.5.2.3 TDefinition.GetImports Method

Pascal

```
function GetImports: IImports;
```

C++

```
__fastcall IImports GetImports();
```

Description

This is GetImports, a member of class TDefinition.

73.1.5.2.4 TDefinition.GetMessages Method

Pascal

```
function GetMessages: IMessages;
```

C++

```
__fastcall IMessages GetMessages();
```

Description

This is GetMessages, a member of class TDefinition.

73.1.5.2.5 TDefinition.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TDefinition.

73.1.5.2.6 TDefinition.GetPortTypes Method

Pascal

```
function GetPortTypes: IPortTypes;
```

C++

```
__fastcall IPortTypes GetPortTypes();
```

Description

This is GetPortTypes, a member of class TDefinition.

73.1.5.2.7 TDefinition.GetServices Method

Pascal

```
function GetServices: IServices;
```

C++

```
__fastcall IServices GetServices();
```

Description

This is GetServices, a member of class TDefinition.

73.1.5.2.8 TDefinition.GetTargetNamespace Method

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

This is GetTargetNamespace, a member of class TDefinition.

73.1.5.2.9 TDefinition.GetTypes Method

Pascal

```
function GetTypes: ITypes;
```

C++

```
__fastcall ITypes GetTypes();
```

Description

This is GetTypes, a member of class TDefinition.

73.1.5.2.10 TDefinition.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TDefinition.

73.1.5.2.11 TDefinition.SetTargetNamespace Method

Pascal

```
procedure SetTargetNamespace(Value: WideString);
```

C++

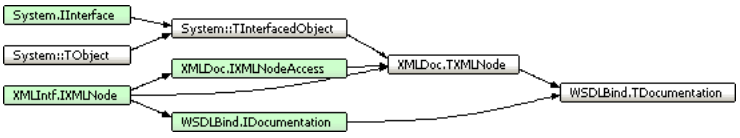
```
__fastcall SetTargetNamespace(BSTR Value);
```

Description

This is SetTargetNamespace, a member of class TDefinition.

73.1.6 TDocumentation Class

Class Hierarchy



Pascal

```
TDocumentation = class(TXMLNode, IDocumentation);
```

C++

```
class TDocumentation : public TXMLNode, public IDocumentation;
```

File

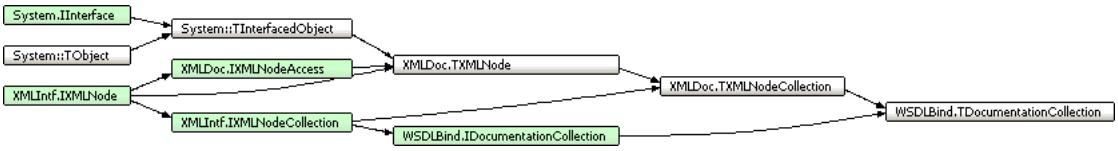
WSDLBind

Description

This is class WSDLBind.TDocumentation.

73.1.7 TDocumentationCollection Class

Class Hierarchy



Pascal

```
TDocumentationCollection = class(TXMLNodeCollection, IDocumentationCollection);
```

C++

```
class TDocumentationCollection : public TXMLNodeCollection, public IDocumentationCollection;
```

File

WSDLBind

Description

This is class WSDLBind.TDocumentationCollection.

73.1.7.1 TDocumentationCollection Members

The following tables list the members exposed by TDocumentationCollection.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 6944)	This is Add, a member of class TDocumentationCollection.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDocumentation (see page 6945)	This is GetDocumentation, a member of class TDocumentationCollection.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.

	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 6945)	This is Insert, a member of class TDocumentationCollection.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.

R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.

	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	Documentation (see page 7104)	This is Documentation, a member of class IDocumentationCollection.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.7.2 TDocumentationCollection Methods

The methods of the TDocumentationCollection class are listed here.

Protected Methods

	Name	Description
	Add (see page 6944)	This is Add, a member of class TDocumentationCollection.
	GetDocumentation (see page 6945)	This is GetDocumentation, a member of class TDocumentationCollection.
	Insert (see page 6945)	This is Insert, a member of class TDocumentationCollection.

73.1.7.2.1 TDocumentationCollection.Add Method

Pascal

```
function Add(const Text: WideString): IDocumentation;
```

C++

```
__fastcall IDocumentation Add(const BSTR Text);
```

Description

This is Add, a member of class TDocumentationCollection.

73.1.7.2.2 TDocumentationCollection.GetDocumentation Method

Pascal

```
function GetDocumentation(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation GetDocumentation(int Index);
```

Description

This is GetDocumentation, a member of class TDocumentationCollection.

73.1.7.2.3 TDocumentationCollection.Insert Method

Pascal

```
function Insert(Index: Integer): IDocumentation;
```

C++

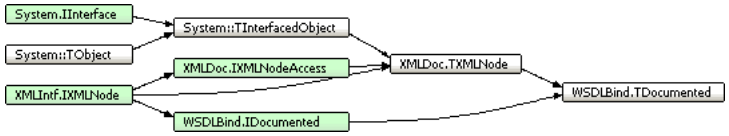
```
__fastcall IDocumentation Insert(int Index);
```

Description

This is Insert, a member of class TDocumentationCollection.

73.1.8 TDocumented Class

Class Hierarchy



Pascal

```
TDocumented = class(TXMLNode, IDocumented);
```

C++

```
class TDocumented : public TXMLNode, public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.TDocumented.

73.1.8.1 TDocumented Members

The following tables list the members exposed by TDocumented.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.

	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.

R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6950)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.

	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.

	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.8.2 TDocumented Methods

The methods of the TDocumented class are listed here.

Protected Methods

	Name	Description
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.

Public Methods

	Name	Description
V	AfterConstruction (see page 6950)	Responds after the last constructor executes.

73.1.8.2.1 TDocumented.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.8.2.2 TDocumented.GetDocumentationCollection Method

Pascal

```
function GetDocumentationCollection: IDocumentationCollection;
```

C++

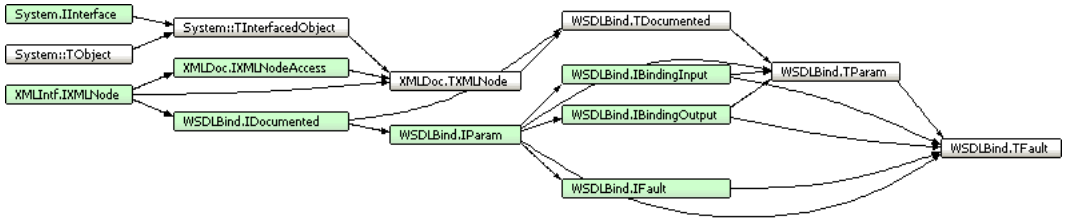
```
__fastcall IDocumentationCollection GetDocumentationCollection();
```

Description

This is GetDocumentationCollection, a member of class TDocumented.

73.1.9 TFault Class

Class Hierarchy



Pascal

```
TFault = class(TParam, IParam, IFault, IBindingInput, IBindingOutput);
```

C++

```
class TFault : public TParam, public IParam, public IFault, public IBindingInput, public
IBindingOutput;
```

File

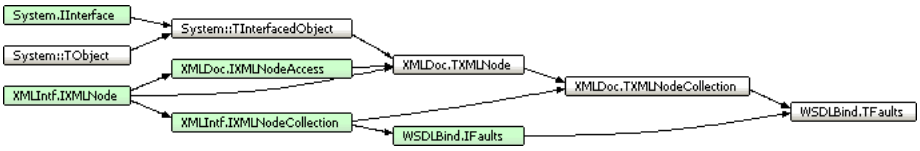
WSDLBind

Description

This is class WSDLBind.TFault.

73.1.10 TFaults Class

Class Hierarchy



Pascal

```
TFaults = class(TXMLNodeCollection, IFaults);
```

C++

```
class TFaults : public TXMLNodeCollection, public IFaults;
```

File

WSDLBind

Description

This is class WSDLBind.TFaults.

73.1.10.1 TFaults Members

The following tables list the members exposed by TFaults.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 6957)	This is Add, a member of class TFaults.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetFault (see page 6958)	This is GetFault, a member of class TFaults.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.

	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 6958)	This is Insert, a member of class TFaults.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.

R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.

	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	Fault (see page 7110)	This is Fault, a member of class IFaults.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.10.2 TFaults Methods

The methods of the TFaults class are listed here.

Protected Methods

	Name	Description
	Add (see page 6957)	This is Add, a member of class TFaults.
	GetFault (see page 6958)	This is GetFault, a member of class TFaults.
	Insert (see page 6958)	This is Insert, a member of class TFaults.

73.1.10.2.1 TFaults.Add Method

Pascal

```
function Add(const Name: WideString; const Message: WideString): IFault;
```

C++

```
__fastcall IFault Add(const BSTR Name, const BSTR Message);
```

Description

This is Add, a member of class TFaults.

73.1.10.2.2 TFaults.GetFault Method

Pascal

```
function GetFault(Index: Integer): IFault;
```

C++

```
__fastcall IFault GetFault(int Index);
```

Description

This is GetFault, a member of class TFaults.

73.1.10.2.3 TFaults.Insert Method

Pascal

```
function Insert(Index: Integer): IFault;
```

C++

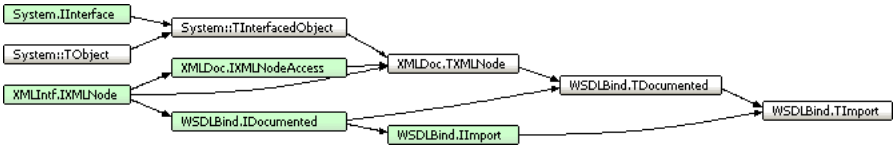
```
__fastcall IFault Insert(int Index);
```

Description

This is Insert, a member of class TFaults.

73.1.11 TImport Class

Class Hierarchy



Pascal

```
TImport = class(TDocumented, IImport);
```

C++

```
class TImport : public TDocumented, public IImport;
```

File

WSDLBind

Description

This is class WSDLBind.TImport.

73.1.11.1 TImport Members

The following tables list the members exposed by TImport.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.

	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetLocation (see page 6964)	This is GetLocation, a member of class TImport.
	GetNamespace (see page 6964)	This is GetNamespace, a member of class TImport.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetLocation (see page 6964)	This is SetLocation, a member of class TImport.
	SetNamespace (see page 6964)	This is SetNamespace, a member of class TImport.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	AfterConstruction (see page 6950)	Responds after the last constructor executes.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.

	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.

	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Location (see page 7113)	This is Location, a member of class IImport.
	Namespace (see page 7113)	This is Namespace, a member of class IImport.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.11.2 TImport Methods

The methods of the TImport class are listed here.

Protected Methods

	Name	Description
	GetLocation (see page 6964)	This is GetLocation, a member of class TImport.
	GetNamespace (see page 6964)	This is GetNamespace, a member of class TImport.
	SetLocation (see page 6964)	This is SetLocation, a member of class TImport.
	SetNamespace (see page 6964)	This is SetNamespace, a member of class TImport.

73.1.11.2.1 TImport.GetLocation Method

Pascal

```
function GetLocation: WideString;
```

C++

```
__fastcall BSTR GetLocation();
```

Description

This is GetLocation, a member of class TImport.

73.1.11.2.2 TImport.GetNamespace Method

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

This is GetNamespace, a member of class TImport.

73.1.11.2.3 TImport.SetLocation Method

Pascal

```
procedure SetLocation(Value: WideString);
```

C++

```
__fastcall SetLocation(BSTR Value);
```

Description

This is SetLocation, a member of class TImport.

73.1.11.2.4 TImport.SetNamespace Method

Pascal

```
procedure SetNamespace(Value: WideString);
```

C++

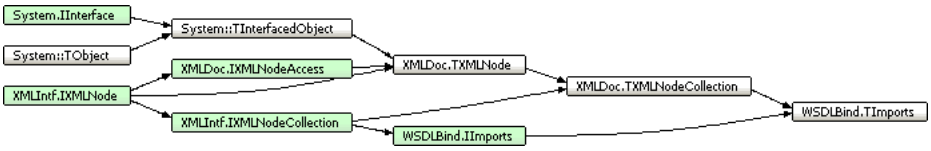
```
__fastcall SetNamespace(BSTR Value);
```

Description

This is SetNamespace, a member of class TImport.

73.1.12 TImports Class

Class Hierarchy



Pascal

```
TImports = class(TXMLNodeCollection, IImports);
```

C++

```
class TImports : public TXMLNodeCollection, public IImports;
```

File

WSDLBind

Description

This is class WSDLBind.TImports.

73.1.12.1 TImports Members

The following tables list the members exposed by TImports.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 6970)	This is Add, a member of class TImports.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetImport (see page 6971)	This is GetImport, a member of class TImports.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.

	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 6971)	This is Insert, a member of class TImports.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.

R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.

	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	Import (see page 7116)	This is Import, a member of class IImports.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.12.2 TImports Methods

The methods of the TImports class are listed here.

Protected Methods

	Name	Description
	Add (see page 6970)	This is Add, a member of class TImports.
	GetImport (see page 6971)	This is GetImport, a member of class TImports.
	Insert (see page 6971)	This is Insert, a member of class TImports.

73.1.12.2.1 TImports.Add Method

Pascal

```
function Add(const Namespace: WideString; const Location: WideString): IImport;
```

C++

```
__fastcall IImport Add(const BSTR Namespace, const BSTR Location);
```

Description

This is Add, a member of class TImports.

73.1.12.2.2 TImports.GetImport Method

Pascal

```
function GetImport(Index: Integer): IImport;
```

C++

```
__fastcall IImport GetImport(int Index);
```

Description

This is GetImport, a member of class TImports.

73.1.12.2.3 TImports.Insert Method

Pascal

```
function Insert(Index: Integer): IImport;
```

C++

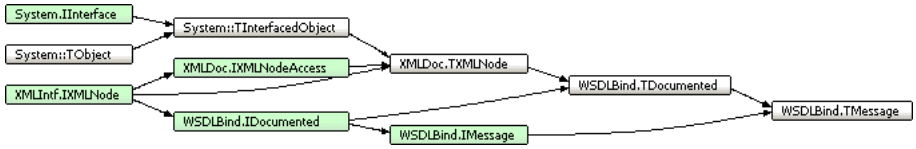
```
__fastcall IImport Insert(int Index);
```

Description

This is Insert, a member of class TImports.

73.1.13 TMessage Class

Class Hierarchy



Pascal

```
TMessage = class(TDocumented, IMessage);
```

C++

```
class TMessage : public TDocumented, public IMessage;
```

File

WSDLBind

Description

This is class WSDLBind.TMessage.

73.1.13.1 TMessage Members

The following tables list the members exposed by TMessage.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.

	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 6977)	This is GetName, a member of class TMessage.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetParts (see page 6977)	This is GetParts, a member of class TMessage.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 6977)	This is SetName, a member of class TMessage.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6977)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.

	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7119)	This is Name, a member of class IMessage.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Parts (see page 7119)	This is Parts, a member of class IMessage.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.13.2 TMessage Methods

The methods of the TMessage class are listed here.

Protected Methods

	Name	Description
	GetName (see page 6977)	This is GetName, a member of class TMessage.
	GetParts (see page 6977)	This is GetParts, a member of class TMessage.
	SetName (see page 6977)	This is SetName, a member of class TMessage.

Public Methods

	Name	Description
V	AfterConstruction (see page 6977)	Responds after the last constructor executes.

73.1.13.2.1 TMessage.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.13.2.2 TMessage.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TMessage.

73.1.13.2.3 TMessage.GetParts Method

Pascal

```
function GetParts: IParts;
```

C++

```
__fastcall IParts GetParts();
```

Description

This is GetParts, a member of class TMessage.

73.1.13.2.4 TMessage.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

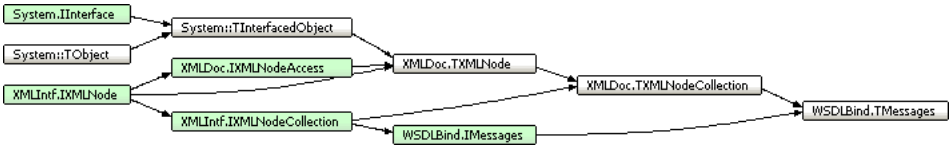
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TMessage.

73.1.14 TMessages Class

Class Hierarchy



Pascal

```
TMessages = class(TXMLNodeCollection, IMessages);
```

C++

```
class TMessages : public TXMLNodeCollection, public IMessages;
```

File

WSDLBind

Description

This is class WSDLBind.TMessages.

73.1.14.1 TMessages Members

The following tables list the members exposed by TMessages.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 6984)	This is Add, a member of class TMessages.

	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 6984)	This is GetMessage, a member of class TMessages.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.

	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 6984)	This is Insert, a member of class TMessages.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.

	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.

	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	Message (see page 7123)	This is Message, a member of class IMessages.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.14.2 TMessages Methods

The methods of the TMessages class are listed here.

Protected Methods

	Name	Description
	Add (see page 6984)	This is Add, a member of class TMessages.
	GetMessage (see page 6984)	This is GetMessage, a member of class TMessages.
	Insert (see page 6984)	This is Insert, a member of class TMessages.

73.1.14.2.1 TMessages.Add Method

Pascal

```
function Add(const Name: WideString): IMessage;
```

C++

```
__fastcall IMessage Add(const BSTR Name);
```

Description

This is Add, a member of class TMessages.

73.1.14.2.2 TMessages.GetMessage Method

Pascal

```
function GetMessage(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage GetMessage(int Index);
```

Description

This is GetMessage, a member of class TMessages.

73.1.14.2.3 TMessages.Insert Method

Pascal

```
function Insert(Index: Integer): IMessage;
```

C++

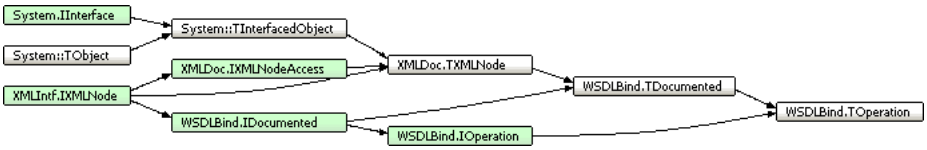
```
__fastcall IMessage Insert(int Index);
```

Description

This is Insert, a member of class TMessages.

73.1.15 TOperation Class

Class Hierarchy



Pascal

```
TOperation = class(TDocumented, IOperation);
```

C++

```
class TOperation : public TDocumented, public IOperation;
```

File

WSDLBind

Description

This is class WSDLBind.TOperation.

73.1.15.1 TOperation Members

The following tables list the members exposed by TOperation.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetFaults (see page 6990)	This is GetFaults, a member of class TOperation.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetInput (see page 6990)	This is GetInput, a member of class TOperation.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 6991)	This is GetName, a member of class TOperation.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOutput (see page 6991)	This is GetOutput, a member of class TOperation.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.

	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 6991)	This is SetName, a member of class TOperation.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 6990)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.

	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.

	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	Faults (see page 7126)	This is Faults, a member of class IOperation.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	Input (see page 7126)	This is Input, a member of class IOperation.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7127)	This is Name, a member of class IOperation.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Output (see page 7127)	This is Output, a member of class IOperation.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.15.2 TOperation Methods

The methods of the TOperation class are listed here.

Protected Methods

	Name	Description
	GetFaults (see page 6990)	This is GetFaults, a member of class TOperation.

	GetInput (see page 6990)	This is GetInput, a member of class TOperation.
	GetName (see page 6991)	This is GetName, a member of class TOperation.
	GetOutput (see page 6991)	This is GetOutput, a member of class TOperation.
	SetName (see page 6991)	This is SetName, a member of class TOperation.

Public Methods

	Name	Description
	AfterConstruction (see page 6990)	Responds after the last constructor executes.

73.1.15.2.1 TOperation.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.15.2.2 TOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class TOperation.

73.1.15.2.3 TOperation.GetInput Method

Pascal

```
function GetInput: IParam;
```

C++

```
__fastcall IParam GetInput();
```

Description

This is GetInput, a member of class TOperation.

73.1.15.2.4 TOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TOperation.

73.1.15.2.5 TOperation.GetOutput Method

Pascal

```
function GetOutput: IParam;
```

C++

```
__fastcall IParam GetOutput();
```

Description

This is GetOutput, a member of class TOperation.

73.1.15.2.6 TOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

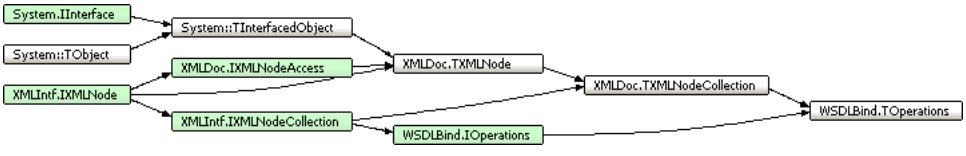
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TOperation.

73.1.16 TOperations Class

Class Hierarchy



Pascal

```
TOperations = class(TXMLNodeCollection, IOperations);
```

C++

```
class TOperations : public TXMLNodeCollection, public IOperations;
```

File

WSDLBind

Description

This is class WSDLBind.TOperations.

73.1.16.1 TOperations Members

The following tables list the members exposed by TOperations.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 6997)	This is the overview for the Add method overload.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.

	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOperation (see page 6998)	This is GetOperation, a member of class TOperations.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 6998)	This is Insert, a member of class TOperations.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.

	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	Add (see page 7129)	This is the overview for the Add method overload.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.

	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.

	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.

	Collection (see page 7690)	Indicates the collection in which this node appears.
	Count (see page 7702)	Indicates the number of repeating child nodes.
	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7693)	Indicates the node name.
	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
	Operation (see page 7130)	This is Operation, a member of class IOperations.
	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.16.2 TOperations Methods

The methods of the TOperations class are listed here.

Protected Methods

	Name	Description
	Add (see page 6997)	This is the overview for the Add method overload.
	GetOperation (see page 6998)	This is GetOperation, a member of class TOperations.
	Insert (see page 6998)	This is Insert, a member of class TOperations.

73.1.16.2.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	TOperations.Add (WideString) (see page 6997)	This is Add, a member of class TOperations.
	TOperations.Add (WideString, WideString, WideString, WideString) (see page 6998)	This is Add, a member of class TOperations.

73.1.16.2.1.1 TOperations.Add Method (WideString)

Pascal

```
function Add(const Name: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name);
```

Description

This is Add, a member of class TOperations.

73.1.16.2.1.2 TOperations.Add Method (WideString, WideString, WideString, WideString)

Pascal

```
function Add(const Name: WideString; const InputMessage: WideString; const InputName:
WideString; const OutputMessage: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name, const BSTR InputMessage, const BSTR InputName,
const BSTR OutputMessage);
```

Description

This is Add, a member of class TOperations.

73.1.16.2.2 TOperations.GetOperation Method

Pascal

```
function GetOperation(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation GetOperation(int Index);
```

Description

This is GetOperation, a member of class TOperations.

73.1.16.2.3 TOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IOperation;
```

C++

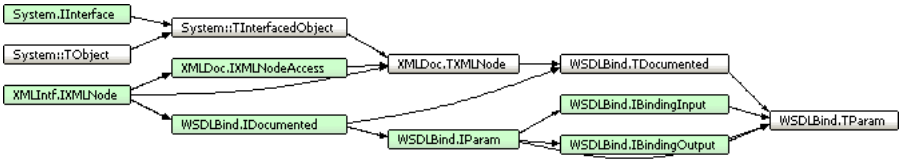
```
__fastcall IOperation Insert(int Index);
```

Description

This is Insert, a member of class TOperations.

73.1.17 TParam Class

Class Hierarchy



Pascal

```
TParam = class(TDocumented, IParam, IBindingInput, IBindingOutput);
```

C++

```
class TParam : public TDocumented, public IParam, public IBindingInput, public IBindingOutput;
```

File

WSDLBind

Description

This is class WSDLBind.TParam.

73.1.17.1 TParam Members

The following tables list the members exposed by TParam.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 7004)	This is GetMessage, a member of class TParam.
	GetName (see page 7004)	This is GetName, a member of class TParam.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetMessage (see page 7004)	This is SetMessage, a member of class TParam.

	SetName (see page 7004)	This is SetName, a member of class TParam.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 6950)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Message (see page 7134)	This is Message, a member of class IParam.
	Name (see page 7134)	This is Name, a member of class IParam.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.17.2 TParam Methods

The methods of the TParam class are listed here.

Protected Methods

	Name	Description
	GetMessage (see page 7004)	This is GetMessage, a member of class TParam.
	GetName (see page 7004)	This is GetName, a member of class TParam.

	SetMessage (see page 7004)	This is SetMessage, a member of class TParam.
	SetName (see page 7004)	This is SetName, a member of class TParam.

73.1.17.2.1 TParam.GetMessage Method

Pascal

```
function GetMessage: WideString;
```

C++

```
__fastcall BSTR GetMessage();
```

Description

This is GetMessage, a member of class TParam.

73.1.17.2.2 TParam.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TParam.

73.1.17.2.3 TParam.SetMessage Method

Pascal

```
procedure SetMessage(Value: WideString);
```

C++

```
__fastcall SetMessage(BSTR Value);
```

Description

This is SetMessage, a member of class TParam.

73.1.17.2.4 TParam.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

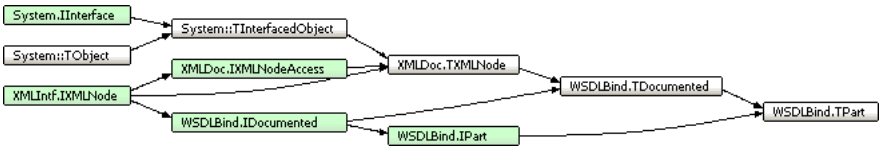
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TParam.

73.1.18 TPart Class

Class Hierarchy



Pascal

```
TPart = class(TDocumented, IPart);
```

C++

```
class TPart : public TDocumented, public IPart;
```

File

WSDLBind

Description

This is class WSDLBind.TPart.

73.1.18.1 TPart Members

The following tables list the members exposed by TPart.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.

	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetElement (see page 7010)	This is GetElement, a member of class TPart.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 7010)	This is GetName, a member of class TPart.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetType_ (see page 7011)	This is GetType_, a member of class TPart.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.

	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetElement (see page 7011)	This is SetElement, a member of class TPart.
	SetName (see page 7011)	This is SetName, a member of class TPart.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	SetType_ (see page 7011)	This is SetType_, a member of class TPart.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 6950)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.

	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.

	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
	Element (see page 7138)	This is Element, a member of class IPart.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7138)	This is Name, a member of class IPart.

	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7693)	Indicates the node name.
	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
	Type_ (see page 7138)	This is Type_, a member of class IPart.
	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.18.2 TPart Methods

The methods of the TPart class are listed here.

Protected Methods

	Name	Description
	GetElement (see page 7010)	This is GetElement, a member of class TPart.
	GetName (see page 7010)	This is GetName, a member of class TPart.
	GetType_ (see page 7011)	This is GetType_, a member of class TPart.
	SetElement (see page 7011)	This is SetElement, a member of class TPart.
	SetName (see page 7011)	This is SetName, a member of class TPart.
	SetType_ (see page 7011)	This is SetType_, a member of class TPart.

73.1.18.2.1 TPart.GetElement Method

Pascal

```
function GetElement: WideString;
```

C++

```
__fastcall BSTR GetElement();
```

Description

This is GetElement, a member of class TPart.

73.1.18.2.2 TPart.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPart.

73.1.18.2.3 TPart.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class TPart.

73.1.18.2.4 TPart.SetElement Method

Pascal

```
procedure SetElement(Value: WideString);
```

C++

```
__fastcall SetElement(BSTR Value);
```

Description

This is SetElement, a member of class TPart.

73.1.18.2.5 TPart.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPart.

73.1.18.2.6 TPart.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

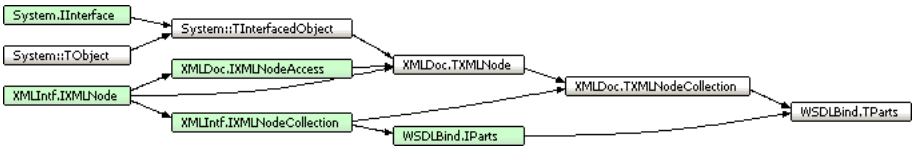
```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class TPart.

73.1.19 TParts Class

Class Hierarchy



Pascal

```
TParts = class(TXMLNodeCollection, IParts);
```

C++

```
class TParts : public TXMLNodeCollection, public IParts;
```

File

WSDLBind

Description

This is class WSDLBind.TParts.

73.1.19.1 TParts Members

The following tables list the members exposed by TParts.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 7017)	This is Add, a member of class TParts.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPart (see page 7018)	This is GetPart, a member of class TParts.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.

	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 7018)	This is Insert, a member of class TParts.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.

R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.

	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Part (see page 7142)	This is Part, a member of class IParts.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.19.2 TParts Methods

The methods of the TParts class are listed here.

Protected Methods

	Name	Description
	Add (see page 7017)	This is Add, a member of class TParts.
	GetPart (see page 7018)	This is GetPart, a member of class TParts.
	Insert (see page 7018)	This is Insert, a member of class TParts.

73.1.19.2.1 TParts.Add Method

Pascal

```
function Add(const Name: WideString; const Element: WideString; const AType: WideString = ''):
IPart;
```

C++

```
__fastcall IPart Add(const BSTR Name, const BSTR Element, const BSTR AType = '');
```

Description

This is Add, a member of class TParts.

73.1.19.2.2 TParts.GetPart Method

Pascal

```
function GetPart(Index: Integer): IPart;
```

C++

```
__fastcall IPart GetPart(int Index);
```

Description

This is GetPart, a member of class TParts.

73.1.19.2.3 TParts.Insert Method

Pascal

```
function Insert(Index: Integer): IPart;
```

C++

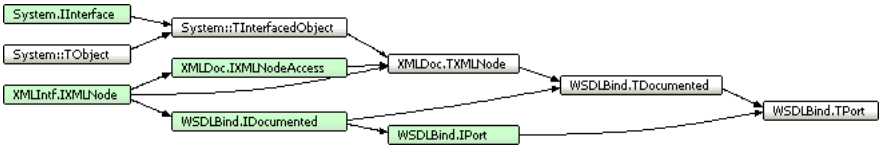
```
__fastcall IPart Insert(int Index);
```

Description

This is Insert, a member of class TParts.

73.1.20 TPort Class

Class Hierarchy



Pascal

```
TPort = class(TDocumented, IPort);
```

C++

```
class TPort : public TDocumented, public IPort;
```

File

WSDLBind

Description

This is class WSDLBind.TPort.

73.1.20.1 TPort Members

The following tables list the members exposed by TPort.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetBinding (see page 7024)	This is GetBinding, a member of class TPort.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.

	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 7024)	This is GetName, a member of class TPort.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetBinding (see page 7024)	This is SetBinding, a member of class TPort.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 7024)	This is SetName, a member of class TPort.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	AfterConstruction (see page 6950)	Responds after the last constructor executes.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.

	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.

	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
	Binding (see page 7145)	This is Binding, a member of class IPort.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7145)	This is Name, a member of class IPort.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.20.2 TPort Methods

The methods of the TPort class are listed here.

Protected Methods

	Name	Description
	GetBinding (see page 7024)	This is GetBinding, a member of class TPort.
	GetName (see page 7024)	This is GetName, a member of class TPort.
	SetBinding (see page 7024)	This is SetBinding, a member of class TPort.
	SetName (see page 7024)	This is SetName, a member of class TPort.

73.1.20.2.1 TPort.GetBinding Method

Pascal

```
function GetBinding: WideString;
```

C++

```
__fastcall BSTR GetBinding();
```

Description

This is GetBinding, a member of class TPort.

73.1.20.2.2 TPort.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPort.

73.1.20.2.3 TPort.SetBinding Method

Pascal

```
procedure SetBinding(Value: WideString);
```

C++

```
__fastcall SetBinding(BSTR Value);
```

Description

This is SetBinding, a member of class TPort.

73.1.20.2.4 TPort.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

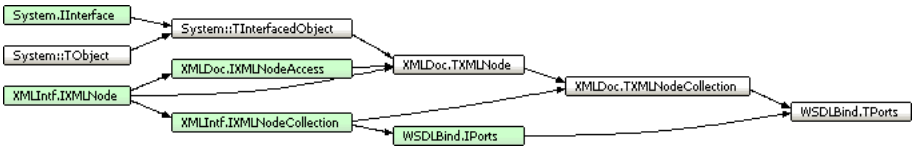
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPort.

73.1.21 TPorts Class

Class Hierarchy



Pascal

```
TPorts = class(TXMLNodeCollection, IPorts);
```

C++

```
class TPorts : public TXMLNodeCollection, public IPorts;
```

File

WSDLBind

Description

This is class WSDLBind.TPorts.

73.1.21.1 TPorts Members

The following tables list the members exposed by TPorts.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 7030)	This is Add, a member of class TPorts.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPort (see page 7031)	This is GetPort, a member of class TPorts.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.

	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 7031)	This is Insert, a member of class TPorts.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.

R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.

	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.

	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Port (see page 7148)	This is Port, a member of class IPorts.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.21.2 TPorts Methods

The methods of the TPorts class are listed here.

Protected Methods

	Name	Description
	Add (see page 7030)	This is Add, a member of class TPorts.
	GetPort (see page 7031)	This is GetPort, a member of class TPorts.
	Insert (see page 7031)	This is Insert, a member of class TPorts.

73.1.21.2.1 TPorts.Add Method

Pascal

```
function Add(const Name: WideString; const Binding: WideString): IPort;
```

C++

```
__fastcall IPort Add(const BSTR Name, const BSTR Binding);
```

Description

This is Add, a member of class TPorts.

73.1.21.2.2 TPorts.GetPort Method

Pascal

```
function GetPort(Index: Integer): IPort;
```

C++

```
__fastcall IPort GetPort(int Index);
```

Description

This is GetPort, a member of class TPorts.

73.1.21.2.3 TPorts.Insert Method

Pascal

```
function Insert(Index: Integer): IPort;
```

C++

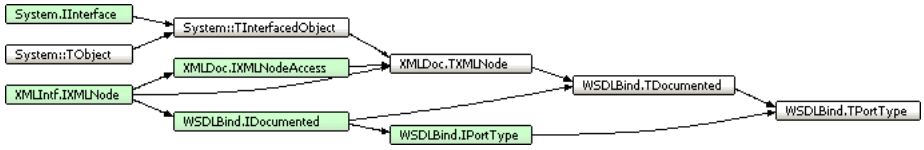
```
__fastcall IPort Insert(int Index);
```

Description

This is Insert, a member of class TPorts.

73.1.22 TPortType Class

Class Hierarchy



Pascal

```
TPortType = class(TDocumented, IPortType);
```

C++

```
class TPortType : public TDocumented, public IPortType;
```

File

WSDLBind

Description

This is class WSDLBind.TPortType.

73.1.22.1 TPortType Members

The following tables list the members exposed by TPortType.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.

	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 7037)	This is GetName, a member of class TPortType.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOperations (see page 7037)	This is GetOperations, a member of class TPortType.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 7037)	This is SetName, a member of class TPortType.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7037)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.

	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7151)	This is Name, a member of class IPortType.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Operations (see page 7151)	This is Operations, a member of class IPortType.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.22.2 TPortType Methods

The methods of the TPortType class are listed here.

Protected Methods

	Name	Description
	GetName (see page 7037)	This is GetName, a member of class TPortType.
	GetOperations (see page 7037)	This is GetOperations, a member of class TPortType.
	SetName (see page 7037)	This is SetName, a member of class TPortType.

Public Methods

	Name	Description
V	AfterConstruction (see page 7037)	Responds after the last constructor executes.

73.1.22.2.1 TPortType.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.22.2.2 TPortType.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TPortType.

73.1.22.2.3 TPortType.GetOperations Method

Pascal

```
function GetOperations: IOperations;
```

C++

```
__fastcall IOperations GetOperations();
```

Description

This is GetOperations, a member of class TPortType.

73.1.22.2.4 TPortType.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

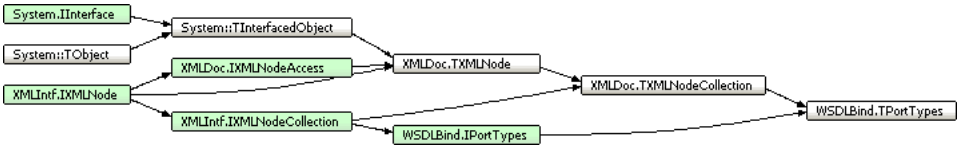
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TPortType.

73.1.23 TPortTypes Class

Class Hierarchy



Pascal

```
TPortTypes = class(TXMLNodeCollection, IPortTypes);
```

C++

```
class TPortTypes : public TXMLNodeCollection, public IPortTypes;
```

File

WSDLBind

Description

This is class WSDLBind.TPortTypes.

73.1.23.1 TPortTypes Members

The following tables list the members exposed by TPortTypes.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 7044)	This is Add, a member of class TPortTypes.

	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.

	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPortType (see page 7044)	This is GetPortType, a member of class TPortTypes.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 7044)	This is Insert, a member of class TPortTypes.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.

	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.

	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	PortType (see page 7155)	This is PortType, a member of class IPortTypes.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.23.2 TPortTypes Methods

The methods of the TPortTypes class are listed here.

Protected Methods

	Name	Description
	Add (see page 7044)	This is Add, a member of class TPortTypes.
	GetPortType (see page 7044)	This is GetPortType, a member of class TPortTypes.
	Insert (see page 7044)	This is Insert, a member of class TPortTypes.

73.1.23.2.1 TPortTypes.Add Method

Pascal

```
function Add(const Name: WideString): IPortType; overload;
```

C++

```
__fastcall IPortType Add(const BSTR Name);
```

Description

This is Add, a member of class TPortTypes.

73.1.23.2.2 TPortTypes.GetPortType Method

Pascal

```
function GetPortType(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType GetPortType(int Index);
```

Description

This is GetPortType, a member of class TPortTypes.

73.1.23.2.3 TPortTypes.Insert Method

Pascal

```
function Insert(Index: Integer): IPortType;
```

C++

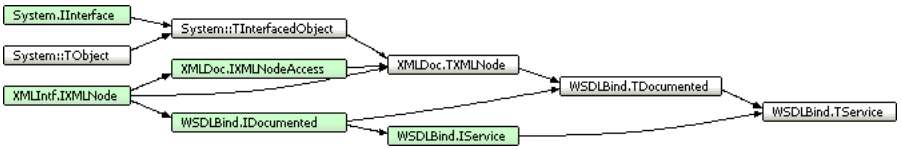
```
__fastcall IPortType Insert(int Index);
```

Description

This is Insert, a member of class TPortTypes.

73.1.24 TService Class

Class Hierarchy



Pascal

```
TService = class(TDocumented, IService);
```

C++

```
class TService : public TDocumented, public IService;
```

File

WSDLBind

Description

This is class WSDLBind.TService.

73.1.24.1 TService Members

The following tables list the members exposed by TService.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.

	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetName (see page 7050)	This is GetName, a member of class TService.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPorts (see page 7050)	This is GetPorts, a member of class TService.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetName (see page 7051)	This is SetName, a member of class TService.

	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7050)	Responds after the last constructor executes.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.

	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.

	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7158)	This is Name, a member of class IService.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Ports (see page 7158)	This is Ports, a member of class IService.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.24.2 TService Methods

The methods of the TService class are listed here.

Protected Methods

	Name	Description
	GetName (see page 7050)	This is GetName, a member of class TService.
	GetPorts (see page 7050)	This is GetPorts, a member of class TService.
	SetName (see page 7051)	This is SetName, a member of class TService.

Public Methods

	Name	Description
	AfterConstruction (see page 7050)	Responds after the last constructor executes.

73.1.24.2.1 TService.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.24.2.2 TService.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class TService.

73.1.24.2.3 TService.GetPorts Method

Pascal

```
function GetPorts: IPorts;
```

C++

```
__fastcall IPorts GetPorts();
```

Description

This is GetPorts, a member of class TService.

73.1.24.2.4 TService.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

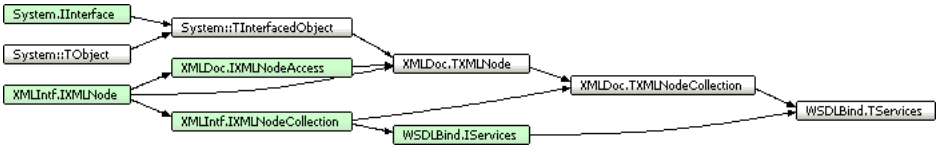
```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class TService.

73.1.25 TServices Class

Class Hierarchy



Pascal

```
TServices = class(TXMLNodeCollection, IServices);
```

C++

```
class TServices : public TXMLNodeCollection, public IServices;
```

File

WSDLBind

Description

This is class WSDLBind.TServices.

73.1.25.1 TServices Members

The following tables list the members exposed by TServices.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 7057)	This is Add, a member of class TServices.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.

	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetService (see page 7057)	This is GetService, a member of class TServices.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	Insert (see page 7057)	This is Insert, a member of class TServices.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
	Count (see page 7457)	Indicates the number of repeating child nodes.
	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	Destroy (see page 3356)	Disposes of an object instance.

	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.

	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
R	Service (see page 7161)	This is Service, a member of class IServices.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.25.2 TServices Methods

The methods of the TServices class are listed here.

Protected Methods

	Name	Description
	Add (see page 7057)	This is Add, a member of class TServices.
	GetService (see page 7057)	This is GetService, a member of class TServices.
	Insert (see page 7057)	This is Insert, a member of class TServices.

73.1.25.2.1 TServices.Add Method

Pascal

```
function Add(const Name: WideString): IService;
```

C++

```
__fastcall IService Add(const BSTR Name);
```

Description

This is Add, a member of class TServices.

73.1.25.2.2 TServices.GetService Method

Pascal

```
function GetService(Index: Integer): IService;
```

C++

```
__fastcall IService GetService(int Index);
```

Description

This is GetService, a member of class TServices.

73.1.25.2.3 TServices.Insert Method

Pascal

```
function Insert(Index: Integer): IService;
```

C++

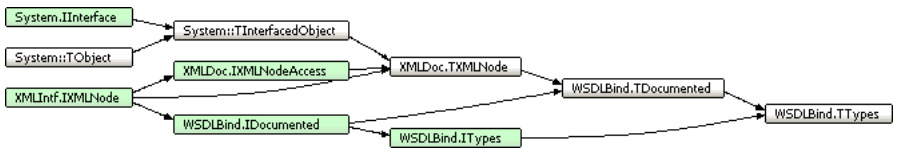
```
__fastcall IService Insert(int Index);
```

Description

This is Insert, a member of class TServices.

73.1.26 TTypes Class

Class Hierarchy



Pascal

```
TTypes = class(TDocumented, ITypes);
```

C++

```
class TTypes : public TDocumented, public ITypes;
```

File

WSDLBind

Description

This is class WSDLBind.TTypes.

73.1.26.1 TTypes Members

The following tables list the members exposed by TTypes.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.

	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDocumentationCollection (see page 6951)	This is GetDocumentationCollection, a member of class TDocumented.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetSchemaDefs (see page 7063)	This is GetSchemaDefs, a member of class TTypes.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 7063)	Responds after the last constructor executes.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.

	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
R	SchemaDefs (see page 7163)	This is SchemaDefs, a member of class ITypes.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.26.2 TTypes Methods

The methods of the TTypes class are listed here.

Protected Methods

	Name	Description
	GetSchemaDefs (see page 7063)	This is GetSchemaDefs, a member of class TTypes.

Public Methods

	Name	Description
	AfterConstruction (see page 7063)	Responds after the last constructor executes.

73.1.26.2.1 TTypes.AfterConstruction Method

Responds after the last constructor executes.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TInterfacedObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

When an instance of System::TInterfacedObject is allocated, System::TInterfacedObject::RefCount is incremented to prevent its destruction while processing the constructor. System::TInterfacedObject::AfterConstruction decrements that System::TInterfacedObject::RefCount to re-establish the correct reference count after all constructors have executed.

See Also

System::TInterfacedObject::BeforeDestruction (see page 3346)

System::TInterfacedObject::NewInstance (see page 3346)

System::TInterfacedObject::RefCount (see page 3347)

73.1.26.2.2 TTypes.GetSchemaDefs Method

Pascal

```
function GetSchemaDefs: IXMLSchemaDefs;
```

C++

```
__fastcall IXMLSchemaDefs GetSchemaDefs();
```

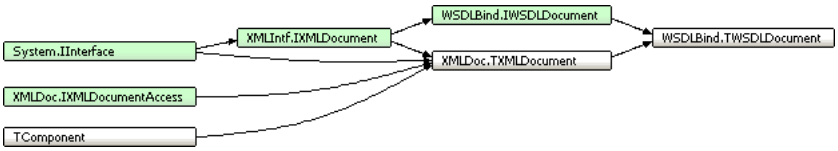
Description

This is GetSchemaDefs, a member of class TTypes.

73.1.27 TWSDLDocument Class

TWSDLDocument represents a WSDL document.

Class Hierarchy



Pascal

```
TWSDLDocument = class(TXMLDocument, IWSDLDocument);
```

C++

```
class TWSDLDocument : public TXMLDocument, public IWSDLDocument;
```

File

WSDLBind

Description

Use TWSDLDocument to represent a WSDL (Web Services Definition (see page 7070) Language) document. A WSDL document is a special form of XML document that describes the contents of a Web Service application and how you call the services it implements.

TWSDLDocument can read an existing WSDL document from a file, it can be associated with an in-memory string that is the contents of a WSDL document, or it can create a new, empty WSDL document. TWSDLDocument uses an external DOM (Document Object Model) parser to analyze the WSDL document. This parser is usually assigned by specifying the DOMVendor property. When you set the Active property to true, TWSDLDocument uses the DOM parser to parse the WSDL document so that you can examine or modify it.

When TWSDLDocument is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TWSDLDocument instance is automatically freed. When TWSDLDocument is created with an Owner, however, it behaves like any other component, and is freed by its Owner.

Tip: Although you can work directly with TWSDLDocument to access a WSDL document, it is often easier to use TWSDLItems, a TWSDLDocument descendant. TWSDLItems implements a number of methods for getting at the definitions in a WSDL document.

See Also

TWSDLItems (see page 7211)

TWSDLView (see page 7271)

73.1.27.1 TWSDLDocument Members

The following tables list the members exposed by TWSDLDocument.

Protected Methods

	Name	Description
	_AddRef (see page 7354)	Called when an application uses a component interface.
	_Release (see page 7354)	Called when an application releases a component interface.
	AssignParseOptions (see page 7356)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7356)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7356)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7356)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7358)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7359)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7359)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7359)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7360)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7360)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7360)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7361)	Returns the value of the AsyncLoadState property.

V	GetChildNodeClass (see page 7361)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7361)	Returns the value of the ChildNodes property.
	GetDefinition (see page 7069)	Returns the value of the Definition property.
	GetDocumentElement (see page 7362)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7362)	Returns the value of the Node property.
	GetDocumentObject (see page 7363)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7363)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7363)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7363)	Returns the value of the Encoding property.
	GetFileName (see page 7364)	Returns the value of the FileName property.
	GetModified (see page 7364)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7364)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7365)	Returns the value of the Options property.
	GetParseOptions (see page 7365)	Returns the value of the ParseOptions property.
	GetPrologNode (see page 7365)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7366)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7366)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7366)	Returns the value of the StandAlone property.
	GetVersion (see page 7366)	Returns the value of the Version property.
	GetXML (see page 7367)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7367)	This is InternalSetPrologValue, a member of class TXMLDocument.
V	LoadData (see page 7069)	This is LoadData, a member of class TWSDLDocument.
V	Loaded (see page 7368)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7371)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7372)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7374)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7375)	Saves the XML (see page 7392) document to an internal string list.
V	SetActive (see page 7375)	Sets the value of the Active property.
	SetDefinition (see page 7070)	Sets the value of the Definition property.
	SetDocumentElement (see page 7375)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7376)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7376)	Sets the value of the Encoding property.
	SetFileName (see page 7376)	Sets the value of the FileName property.
	SetModified (see page 7377)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7377)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7377)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7378)	Sets the value of the Options property.
	SetParseOptions (see page 7378)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7378)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7378)	Sets the value of the StandAlone property.
	SetVersion (see page 7379)	Sets the value of the Version property.
	SetXML (see page 7379)	Sets the value of the XML property.
	SetXMLStrings (see page 7379)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7380)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7380)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7383)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7383)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7385)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7386)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7391)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7354)	This is the overview for the AddChild method overload.
	AddChild (see page 7643)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7069)	Initializes the TWSDLDocument (see page 7063) object after the last constructor has executed.
	Create (see page 7351)	This is the overview for the Create constructor overload.
	CreateElement (see page 7357)	Creates a new element node that is associated with this document.
	CreateElement (see page 7644)	Creates a new element node that is associated with this document.
	CreateNode (see page 7357)	Creates a new node that is associated with this document.
	CreateNode (see page 7645)	Creates a new node that is associated with this document.
V	Destroy (see page 7352)	Disposes of a TXMLDocument (see page 7347) object.
	GeneratePrefix (see page 7360)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 7646)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7646)	Returns the value of the Active (see page 7661) property.
	GetAsyncLoadState (see page 7647)	Returns the value of the AsyncLoadState (see page 7662) property.
	GetChildNodes (see page 7647)	Returns the value of the ChildNodes (see page 7663) property.
	GetDocBinding (see page 7362)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7647)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7648)	Returns the value of the DocumentElement (see page 7663) property.
	GetDocumentNode (see page 7648)	Returns the value of the Node (see page 7665) property.
	GetDocumentObject (see page 7479)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7648)	Returns the value of the DOMDocument (see page 7663) property.
	GetDOMPersist (see page 7479)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7649)	Returns the value of the Encoding (see page 7664) property.
	GetFileName (see page 7649)	Returns the value of the FileName (see page 7664) property.
	GetModified (see page 7649)	Returns the value of the Modified (see page 7665) property.
	GetNodeIndentStr (see page 7650)	Returns the value of the NodeIndentStr (see page 7666) property.
	GetOptions (see page 7650)	Returns the value of the Options (see page 7666) property.
	GetParseOptions (see page 7650)	Returns the value of the ParseOptions (see page 7667) property.
	GetSchemaRef (see page 7650)	Returns the value of the SchemaRef (see page 7667) property.
	GetStandAlone (see page 7651)	Returns the value of the StandAlone (see page 7668) property.
	GetVersion (see page 7651)	Returns the value of the Version (see page 7668) property.
	GetXML (see page 7651)	Returns the value of the XML (see page 7668) property.
	IsEmptyDoc (see page 7367)	Indicates whether the TXMLDocument (see page 7347) instance represents an empty document.
	IsEmptyDoc (see page 7652)	Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.
	LoadFromFile (see page 7368)	Loads an XML (see page 7392) document and activates it.
	LoadFromFile (see page 7652)	Loads an XML (see page 7668) document from disk.
	LoadFromStream (see page 7369)	Loads an XML (see page 7392) document from a stream and activates it.
	LoadFromStream (see page 7653)	Loads an XML (see page 7668) document from a stream and activates it.
	LoadFromXML (see page 7370)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
V	NewInstance (see page 7370)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7371)	Updates the parsed XML (see page 7392) document to reflect external changes.
	Refresh (see page 7654)	Updates the parsed XML (see page 7668) document to reflect external changes.

	RegisterDocBinding (see page 7372)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7654)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7372)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7655)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7373)	Saves the XML (see page 7392) document to disk.
	SaveToFile (see page 7655)	Saves the XML (see page 7668) document to disk.
	SaveToStream (see page 7373)	Saves the XML (see page 7392) document to a stream.
	SaveToStream (see page 7656)	Saves the XML (see page 7668) document to a stream.
	SaveToXML (see page 7374)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7657)	Sets the value of the Active (see page 7661) property.
	SetDocumentElement (see page 7657)	Sets the value of the DocumentElement (see page 7663) property.
	SetDOMDocument (see page 7658)	Sets the value of the DOMDocument (see page 7663) property.
	SetEncoding (see page 7658)	Sets the value of the Encoding (see page 7664) property.
	SetFileName (see page 7658)	Sets the value of the FileName (see page 7664) property.
	SetNodeIndentStr (see page 7659)	Sets the value of the NodeIndentStr (see page 7666) property.
	SetOnAsyncLoad (see page 7659)	Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.
	SetOptions (see page 7659)	Sets the value of the Options (see page 7666) property.
	SetParseOptions (see page 7660)	Sets the value of the ParseOptions (see page 7667) property.
	SetStandAlone (see page 7660)	Sets the value of the StandAlone (see page 7668) property.
	SetVersion (see page 7660)	Sets the value of the Version (see page 7668) property.
	SetXML (see page 7661)	Sets the value of the XML (see page 7668) property.

Public Properties

	Name	Description
	Active (see page 7661)	Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7382)	Indicates the state of the DOM parser when it parses the XML (see page 7392) document asynchronously.
R	AsyncLoadState (see page 7662)	Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.
R	ChildNodes (see page 7383)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7663)	Lists all of the child nodes of the document.
	Definition (see page 7070)	Provides access to the definitions contained in the WSDL document.
	DocumentElement (see page 7383)	Provides access to the root node of the XML (see page 7392) document.
	DocumentElement (see page 7663)	Provides access to the root node of the XML (see page 7668) document.
R	DocumentObject (see page 7479)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7384)	Provides access to the DOM interface for the parsed XML (see page 7392) document.
	DOMDocument (see page 7663)	Provides access to the DOM interface for the parsed XML (see page 7668) document.
	DOMImplementation (see page 7385)	Provides access to the DOM interface for document-independent functions.
R	DOMPersist (see page 7480)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7386)	Indicates the character set that the XML (see page 7392) document uses to encode characters.
	Encoding (see page 7664)	Indicates the character set that the XML (see page 7668) document uses to encode characters.
	FileName (see page 7664)	Indicates the XML (see page 7668) document file, if any, that this interface represents.
R	Modified (see page 7387)	Indicates whether the XML (see page 7392) document has been modified.
R	Modified (see page 7665)	Indicates whether the XML (see page 7668) document has been modified.
R	Node (see page 7388)	Provides access to the document node for the XML (see page 7392) document.
R	Node (see page 7665)	Provides access to the document node for the XML (see page 7668) document.

	NodeIndentStr (see page 7666)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.
	NSPrefixBase (see page 7389)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7666)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7667)	Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.
R	SchemaRef (see page 7391)	Returns the name of the schema associated with this XML (see page 7392) document.
R	SchemaRef (see page 7667)	Returns the name of the schema associated with this XML (see page 7668) document.
	StandAlone (see page 7391)	Indicates whether the XML (see page 7392) document includes any external markup declarations.
	StandAlone (see page 7668)	Indicates whether the XML (see page 7668) document includes any external markup declarations.
	Version (see page 7391)	Indicates the XML (see page 7392) version that the document uses.
	Version (see page 7668)	Indicates the XML (see page 7668) version that the document uses.
	XML (see page 7668)	Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.

Published Events

	Name	Description
	AfterClose (see page 7393)	Occurs after the XML (see page 7392) document is closed.
	AfterNodeChange (see page 7393)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7394)	Occurs immediately after the XML (see page 7392) document is opened.
	BeforeClose (see page 7394)	Occurs immediately before the XML (see page 7392) document is closed.
	BeforeNodeChange (see page 7394)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7395)	Occurs immediately before the XML (see page 7392) document is opened.
	OnAsyncLoad (see page 7395)	Occurs at various stages during the asynchronous parsing of an XML (see page 7392) document.

Published Properties

	Name	Description
	Active (see page 7381)	Specifies whether the XML (see page 7392) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7386)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7392) document.
	FileName (see page 7387)	Specifies the file name for the XML (see page 7392) document associated with this TXMLDocument (see page 7347) object.
	NodeIndentStr (see page 7388)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7392) text.
	Options (see page 7389)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7390)	Specifies implementation-specific options that are followed when parsing the XML (see page 7392) document.
	XML (see page 7392)	Specifies the content of the XML document associated with this TXMLDocument (see page 7347) object.

73.1.27.2 TWSDLDocument Methods

The methods of the TWSDLDocument class are listed here.

Protected Methods

	Name	Description
	GetDefinition (see page 7069)	Returns the value of the Definition property.
V	LoadData (see page 7069)	This is LoadData, a member of class TWSDLDocument.
	SetDefinition (see page 7070)	Sets the value of the Definition property.

Public Methods

	Name	Description
	AfterConstruction (see page 7069)	Initializes the TWSDLDocument (see page 7063) object after the last constructor has executed.

73.1.27.2.1 TWSDLDocument.AfterConstruction Method

Initializes the TWSDLDocument (see page 7063) object after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

The AfterConstruction method is called automatically after the last constructor has finished executing. TWSDLDocument (see page 7063) overrides this method to register an implementation class for the document element. That class implements the IDefinition (see page 7094) interface that is the value of the Definition (see page 7070) property.

See Also

RegisterDocBinding (see page 7372)

Definition (see page 7070)

73.1.27.2.2 TWSDLDocument.GetDefinition Method

Returns the value of the Definition property.

Pascal

```
function GetDefinition: IDefinition;
```

C++

```
__fastcall IDefinition GetDefinition();
```

Description

GetDefinition is the read access method for the Definition property.

See Also

Definition (see page 7167)

73.1.27.2.3 TWSDLDocument.LoadData Method

Pascal

```
procedure LoadData; override;
```

C++

```
virtual __fastcall LoadData();
```

Description

This is LoadData, a member of class TWSDLDocument.

73.1.27.2.4 TWSDLDocument.SetDefinition Method

Sets the value of the Definition property.

Pascal

```
procedure SetDefinition(const Value: IDefinition);
```

C++

```
__fastcall SetDefinition(const IDefinition Value);
```

Description

SetDefinition is the write method for the Definition property.

See Also

Definition (see page 7167)

73.1.27.3 TWSDLDocument Properties

The properties of the TWSDLDocument class are listed here.

Public Properties

	Name	Description
	Definition (see page 7070)	Provides access to the definitions contained in the WSDL document.

73.1.27.3.1 TWSDLDocument.Definition Property

Provides access to the definitions contained in the WSDL document.

Pascal

```
property Definition: IDefinition;
```

C++

```
__property IDefinition Definition;
```

Description

Definition is the IDefinition (see page 7094) interface of the root node in the WSDL document. This interface is implemented by the node specified by DocumentElement.

You can use the IDefinition (see page 7094) interface to access the Bindings, Messages, PortTypes, Services, and Types (see page 4649) defined in the WSDL document by reading properties of the same names. Alternately, you can use the methods of TWSDLItems (a TWSDLDocument (see page 7063) descendant) to access those definitions.

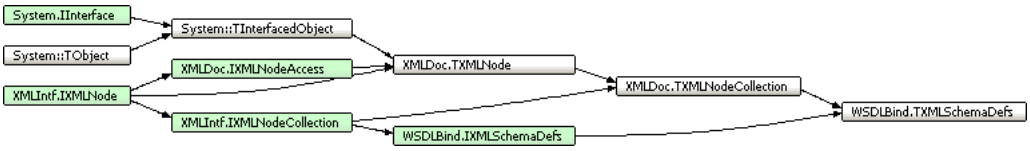
See Also

DocumentElement (see page 7383)

TWSDLItems (see page 7211)

73.1.28 TXMLSchemaDefs Class

Class Hierarchy



Pascal

```
TXMLSchemaDefs = class(TXMLNodeCollection, IXMLSchemaDefs);
```

C++

```
class TXMLSchemaDefs : public TXMLNodeCollection, public IXMLSchemaDefs;
```

File

WSDLBind

Description

This is class WSDLBind.TXMLSchemaDefs.

73.1.28.1 TXMLSchemaDefs Members

The following tables list the members exposed by TXMLSchemaDefs.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	Add (see page 7076)	This is Add, a member of class TXMLSchemaDefs.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.

	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	GetXMLSchemaDef (see page 7077)	This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.

R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.

	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.

	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
R	SchemaDef (see page 7170)	This is SchemaDef, a member of class IXMLSchemaDefs.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.1.28.2 TXMLSchemaDefs Methods

The methods of the TXMLSchemaDefs class are listed here.

Protected Methods

	Name	Description
	Add (see page 7076)	This is Add, a member of class TXMLSchemaDefs.
	GetXMLSchemaDef (see page 7077)	This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.

73.1.28.2.1 TXMLSchemaDefs.Add Method

Pascal

```
function Add(const NamespaceURI: WideString = ''; const TargetNamespace: WideString = ''):
IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef Add(const BSTR NamespaceURI = '', const BSTR TargetNamespace = '');
```

Description

This is Add, a member of class TXMLSchemaDefs.

73.1.28.2.2 TXMLSchemaDefs.GetXMLSchemaDef Method

Pascal

```
function GetXMLSchemaDef(Index: Integer): IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef GetXMLSchemaDef(int Index);
```

Description

This is GetXMLSchemaDef, a member of class TXMLSchemaDefs.

73.2 Interfaces

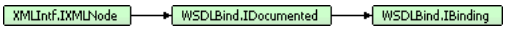
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IBinding (see page 7078)	This is class WSDLBind.IBinding.
	IBindingInput (see page 7082)	This is class WSDLBind.IBindingInput.
	IBindingOperation (see page 7083)	This is class WSDLBind.IBindingOperation.
	IBindingOperations (see page 7087)	This is class WSDLBind.IBindingOperations.
	IBindingOutput (see page 7090)	This is class WSDLBind.IBindingOutput.
	IBindings (see page 7091)	This is class WSDLBind.IBindings.
	IDefinition (see page 7094)	This is class WSDLBind.IDefinition.
	IDocumentation (see page 7100)	This is class WSDLBind.IDocumentation.
	IDocumentationCollection (see page 7101)	This is class WSDLBind.IDocumentationCollection.
	IDocumented (see page 7104)	This is class WSDLBind.IDocumented.
	IFault (see page 7106)	This is class WSDLBind.IFault.
	IFaults (see page 7107)	This is class WSDLBind.IFaults.
	IImport (see page 7110)	This is class WSDLBind.IImport.
	IImports (see page 7113)	This is class WSDLBind.IImports.
	IMessage (see page 7116)	This is class WSDLBind.IMessage.
	IMessages (see page 7120)	This is class WSDLBind.IMessages.
	IOperation (see page 7123)	This is class WSDLBind.IOperation.
	IOperations (see page 7127)	This is class WSDLBind.IOperations.
	IParam (see page 7131)	This is class WSDLBind.IParam.
	IPart (see page 7134)	This is class WSDLBind.IPart.
	IParts (see page 7139)	This is class WSDLBind.IParts.
	IPort (see page 7142)	This is class WSDLBind.IPort.
	IPorts (see page 7145)	This is class WSDLBind.IPorts.
	IPortType (see page 7148)	This is class WSDLBind.IPortType.
	IPortTypes (see page 7152)	This is class WSDLBind.IPortTypes.
	IService (see page 7155)	This is class WSDLBind.IService.
	IServices (see page 7158)	This is class WSDLBind.IServices.
	ITypes (see page 7161)	This is class WSDLBind.ITypes.
	IWSDLDocument (see page 7164)	IWSDLDocument provides low-level access to the definitions in a WSDL document.
	IXMLSchemaDefs (see page 7168)	This is class WSDLBind.IXMLSchemaDefs.

73.2.1 IBinding Interface

Class Hierarchy



Pascal

```
IBinding = interface(IDocumented);
```

C++

```
__interface IBinding : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IBinding.

73.2.1.1 IBinding Members

The following tables list the members exposed by IBinding.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetBindingOperations (see page 7080)	This is GetBindingOperations, a member of class IBinding.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7081)	This is GetName, a member of class IBinding.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetType_ (see page 7081)	This is GetType_, a member of class IBinding.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7081)	This is SetName, a member of class IBinding.

	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	SetType_ (see page 7081)	This is SetType_, a member of class IBinding.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	BindingOperations (see page 7082)	This is BindingOperations, a member of class IBinding.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7082)	This is Name, a member of class IBinding.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
	Type_ (see page 7082)	This is Type_, a member of class IBinding.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.1.2 IBinding Methods

The methods of the IBinding class are listed here.

Public Methods

	Name	Description
	GetBindingOperations (see page 7080)	This is GetBindingOperations, a member of class IBinding.
	GetName (see page 7081)	This is GetName, a member of class IBinding.
	GetType_ (see page 7081)	This is GetType_, a member of class IBinding.
	SetName (see page 7081)	This is SetName, a member of class IBinding.
	SetType_ (see page 7081)	This is SetType_, a member of class IBinding.

73.2.1.2.1 IBinding.GetBindingOperations Method

Pascal

```
function GetBindingOperations: IBindingOperations;
```

C++

```
__fastcall IBindingOperations GetBindingOperations();
```

Description

This is GetBindingOperations, a member of class IBinding.

73.2.1.2.2 IBinding.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IBinding.

73.2.1.2.3 IBinding.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class IBinding.

73.2.1.2.4 IBinding.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IBinding.

73.2.1.2.5 IBinding.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class IBinding.

73.2.1.3 IBinding Properties

The properties of the IBinding class are listed here.

Public Properties

	Name	Description
	BindingOperations (see page 7082)	This is BindingOperations, a member of class IBinding.
	Name (see page 7082)	This is Name, a member of class IBinding.
	Type_ (see page 7082)	This is Type_, a member of class IBinding.

73.2.1.3.1 IBinding.BindingOperations Property

Pascal

```
property BindingOperations: IBindingOperations;
```

C++

```
__property IBindingOperations BindingOperations;
```

Description

This is BindingOperations, a member of class IBinding.

73.2.1.3.2 IBinding.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IBinding.

73.2.1.3.3 IBinding.Type_ Property

Pascal

```
property Type_: WideString;
```

C++

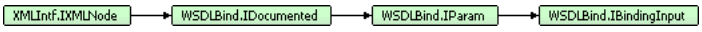
```
__property BSTR Type_;
```

Description

This is Type_, a member of class IBinding.

73.2.2 IBindingInput Interface

Class Hierarchy



Pascal

```
IBindingInput = interface(IParam);
```

C++

```
__interface IBindingInput : public IParam;
```

File

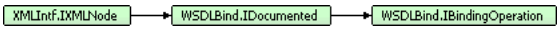
WSDLBind

Description

This is class WSDLBind.IBindingInput.

73.2.3 IBindingOperation Interface

Class Hierarchy



Pascal

```
IBindingOperation = interface(IDocumented);
```

C++

```
__interface IBindingOperation : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingOperation.

73.2.3.1 IBindingOperation Members

The following tables list the members exposed by IBindingOperation.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetFaults (see page 7085)	This is GetFaults, a member of class IBindingOperation.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetInput (see page 7085)	This is GetInput, a member of class IBindingOperation.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7085)	This is GetName, a member of class IBindingOperation.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.

	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOutput (see page 7085)	This is GetOutput, a member of class IBindingOperation.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7086)	This is SetName, a member of class IBindingOperation.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	Faults (see page 7086)	This is Faults, a member of class IBindingOperation.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	Input (see page 7086)	This is Input, a member of class IBindingOperation.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7087)	This is Name, a member of class IBindingOperation.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Output (see page 7087)	This is Output, a member of class IBindingOperation.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.3.2 IBindingOperation Methods

The methods of the IBindingOperation class are listed here.

Public Methods

	Name	Description
	GetFaults (see page 7085)	This is GetFaults, a member of class IBindingOperation.
	GetInput (see page 7085)	This is GetInput, a member of class IBindingOperation.
	GetName (see page 7085)	This is GetName, a member of class IBindingOperation.
	GetOutput (see page 7085)	This is GetOutput, a member of class IBindingOperation.
	SetName (see page 7086)	This is SetName, a member of class IBindingOperation.

73.2.3.2.1 IBindingOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class IBindingOperation.

73.2.3.2.2 IBindingOperation.GetInput Method

Pascal

```
function GetInput: IBindingInput;
```

C++

```
__fastcall IBindingInput GetInput();
```

Description

This is GetInput, a member of class IBindingOperation.

73.2.3.2.3 IBindingOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IBindingOperation.

73.2.3.2.4 IBindingOperation.GetOutput Method

Pascal

```
function GetOutput: IBindingOutput;
```

C++

```
__fastcall IBindingOutput GetOutput();
```

Description

This is GetOutput, a member of class IBindingOperation.

73.2.3.2.5 IBindingOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IBindingOperation.

73.2.3.3 IBindingOperation Properties

The properties of the IBindingOperation class are listed here.

Public Properties

	Name	Description
	Faults (see page 7086)	This is Faults, a member of class IBindingOperation.
	Input (see page 7086)	This is Input, a member of class IBindingOperation.
	Name (see page 7087)	This is Name, a member of class IBindingOperation.
	Output (see page 7087)	This is Output, a member of class IBindingOperation.

73.2.3.3.1 IBindingOperation.Faults Property

Pascal

```
property Faults: IFaults;
```

C++

```
__property IFaults Faults;
```

Description

This is Faults, a member of class IBindingOperation.

73.2.3.3.2 IBindingOperation.Input Property

Pascal

```
property Input: IBindingInput;
```

C++

```
__property IBindingInput Input;
```

Description

This is Input, a member of class IBindingOperation.

73.2.3.3.3 IBindingOperation.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IBindingOperation.

73.2.3.3.4 IBindingOperation.Output Property

Pascal

```
property Output: IBindingOutput;
```

C++

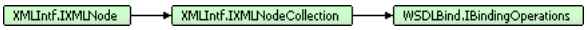
```
__property IBindingOutput Output;
```

Description

This is Output, a member of class IBindingOperation.

73.2.4 IBindingOperations Interface

Class Hierarchy



Pascal

```
IBindingOperations = interface(IXMLNodeCollection);
```

C++

```
__interface IBindingOperations : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IBindingOperations.

73.2.4.1 IBindingOperations Members

The following tables list the members exposed by IBindingOperations.

Public Methods

	Name	Description
	Add (see page 7089)	This is Add, a member of class IBindingOperations.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.

	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetBindingOperation (see page 7089)	This is GetBindingOperation, a member of class IBindingOperations.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7090)	This is Insert, a member of class IBindingOperations.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	BindingOperation (see page 7090)	This is BindingOperation, a member of class IBindingOperations.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.

R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.4.2 IBindingOperations Methods

The methods of the IBindingOperations class are listed here.

Public Methods

	Name	Description
	Add (see page 7089)	This is Add, a member of class IBindingOperations.
	GetBindingOperation (see page 7089)	This is GetBindingOperation, a member of class IBindingOperations.
	Insert (see page 7090)	This is Insert, a member of class IBindingOperations.

73.2.4.2.1 IBindingOperations.Add Method

Pascal

```
function Add(const Name: WideString): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Add(const BSTR Name);
```

Description

This is Add, a member of class IBindingOperations.

73.2.4.2.2 IBindingOperations.GetBindingOperation Method

Pascal

```
function GetBindingOperation(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperation(int Index);
```

Description

This is GetBindingOperation, a member of class IBindingOperations.

73.2.4.2.3 IBindingOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation Insert(int Index);
```

Description

This is Insert, a member of class IBindingOperations.

73.2.4.3 IBindingOperations Properties

The properties of the IBindingOperations class are listed here.

Public Properties

	Name	Description
	BindingOperation (see page 7090)	This is BindingOperation, a member of class IBindingOperations.

73.2.4.3.1 IBindingOperations.BindingOperation Property

Pascal

```
property BindingOperation [Index: Integer]: IBindingOperation;
```

C++

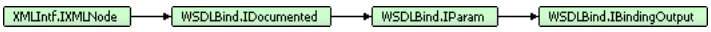
```
__property IBindingOperation BindingOperation[int Index];
```

Description

This is BindingOperation, a member of class IBindingOperations.

73.2.5 IBindingOutput Interface

Class Hierarchy



Pascal

```
IBindingOutput = interface(IParam);
```

C++

```
__interface IBindingOutput : public IParam;
```

File

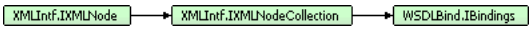
WSDLBind

Description

This is class WSDLBind.IBindingOutput.

73.2.6 IBindings Interface

Class Hierarchy



Pascal

```
IBindings = interface(IXMLNodeCollection);
```

C++

```
__interface IBindings : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IBindings.

73.2.6.1 IBindings Members

The following tables list the members exposed by IBindings.

Public Methods

	Name	Description
	Add (see page 7093)	This is Add, a member of class IBindings.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetBinding (see page 7093)	This is GetBinding, a member of class IBindings.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.

	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7093)	This is Insert, a member of class IBindings.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	Binding (see page 7094)	This is Binding, a member of class IBindings.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.6.2 IBindings Methods

The methods of the IBindings class are listed here.

Public Methods

	Name	Description
	Add (see page 7093)	This is Add, a member of class IBindings.
	GetBinding (see page 7093)	This is GetBinding, a member of class IBindings.
	Insert (see page 7093)	This is Insert, a member of class IBindings.

73.2.6.2.1 IBindings.Add Method

Pascal

```
function Add(const Name: WideString; const AType: WideString): IBinding;
```

C++

```
__fastcall IBinding Add(const BSTR Name, const BSTR AType);
```

Description

This is Add, a member of class IBindings.

73.2.6.2.2 IBindings.GetBinding Method

Pascal

```
function GetBinding(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding GetBinding(int Index);
```

Description

This is GetBinding, a member of class IBindings.

73.2.6.2.3 IBindings.Insert Method

Pascal

```
function Insert(Index: Integer): IBinding;
```

C++

```
__fastcall IBinding Insert(int Index);
```

Description

This is Insert, a member of class IBindings.

73.2.6.3 IBindings Properties

The properties of the IBindings class are listed here.

Public Properties

	Name	Description
	Binding (see page 7094)	This is Binding, a member of class IBindings.

73.2.6.3.1 IBindings.Binding Property

Pascal

```
property Binding [Index: Integer]: IBinding;
```

C++

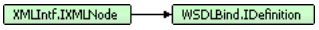
```
__property IBinding Binding[int Index];
```

Description

This is Binding, a member of class IBindings.

73.2.7 IDefinition Interface

Class Hierarchy



Pascal

```
IDefinition = interface(IXMLNode);
```

C++

```
__interface IDefinition : public IXMLNode;
```

File

WSDLBind

Description

This is class WSDLBind.IDefinition.

73.2.7.1 IDefinition Members

The following tables list the members exposed by IDefinition.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetBindings (see page 7096)	This is GetBindings, a member of class IDefinition.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetImports (see page 7096)	This is GetImports, a member of class IDefinition.

	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetMessages (see page 7097)	This is GetMessages, a member of class IDefinition.
	GetName (see page 7097)	This is GetName, a member of class IDefinition.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPortTypes (see page 7097)	This is GetPortTypes, a member of class IDefinition.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetServices (see page 7097)	This is GetServices, a member of class IDefinition.
	GetTargetNamespace (see page 7097)	This is GetTargetNamespace, a member of class IDefinition.
	GetText (see page 7681)	Returns the text value of the node.
	GetTypes (see page 7098)	This is GetTypes, a member of class IDefinition.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7098)	This is SetName, a member of class IDefinition.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetTargetNamespace (see page 7098)	This is SetTargetNamespace, a member of class IDefinition.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	Bindings (see page 7099)	This is Bindings, a member of class IDefinition.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	Imports (see page 7099)	This is Imports, a member of class IDefinition.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	Messages (see page 7099)	This is Messages, a member of class IDefinition.
	Name (see page 7099)	This is Name, a member of class IDefinition.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.

R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	PortTypes (see page 7099)	This is PortTypes, a member of class IDefinition.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	Services (see page 7100)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 7100)	This is TargetNamespace, a member of class IDefinition.
	Text (see page 7697)	Represents the Text of the node.
R	Types (see page 7100)	This is Types, a member of class IDefinition.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.7.2 IDefinition Methods

The methods of the IDefinition class are listed here.

Public Methods

	Name	Description
	GetBindings (see page 7096)	This is GetBindings, a member of class IDefinition.
	GetImports (see page 7096)	This is GetImports, a member of class IDefinition.
	GetMessages (see page 7097)	This is GetMessages, a member of class IDefinition.
	GetName (see page 7097)	This is GetName, a member of class IDefinition.
	GetPortTypes (see page 7097)	This is GetPortTypes, a member of class IDefinition.
	GetServices (see page 7097)	This is GetServices, a member of class IDefinition.
	GetTargetNamespace (see page 7097)	This is GetTargetNamespace, a member of class IDefinition.
	GetTypes (see page 7098)	This is GetTypes, a member of class IDefinition.
	SetName (see page 7098)	This is SetName, a member of class IDefinition.
	SetTargetNamespace (see page 7098)	This is SetTargetNamespace, a member of class IDefinition.

73.2.7.2.1 IDefinition.GetBindings Method

Pascal

```
function GetBindings: IBindings;
```

C++

```
__fastcall IBindings GetBindings();
```

Description

This is GetBindings, a member of class IDefinition.

73.2.7.2.2 IDefinition.GetImports Method

Pascal

```
function GetImports: IImports;
```

C++

```
__fastcall IImports GetImports();
```

Description

This is GetImports, a member of class IDefinition.

73.2.7.2.3 IDefinition.GetMessages Method

Pascal

```
function GetMessages: IMessages;
```

C++

```
__fastcall IMessages GetMessages();
```

Description

This is GetMessages, a member of class IDefinition.

73.2.7.2.4 IDefinition.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IDefinition.

73.2.7.2.5 IDefinition.GetPortTypes Method

Pascal

```
function GetPortTypes: IPortTypes;
```

C++

```
__fastcall IPortTypes GetPortTypes();
```

Description

This is GetPortTypes, a member of class IDefinition.

73.2.7.2.6 IDefinition.GetServices Method

Pascal

```
function GetServices: IServices;
```

C++

```
__fastcall IServices GetServices();
```

Description

This is GetServices, a member of class IDefinition.

73.2.7.2.7 IDefinition.GetTargetNamespace Method

Pascal

```
function GetTargetNamespace: WideString;
```

C++

```
__fastcall BSTR GetTargetNamespace();
```

Description

This is GetTargetNamespace, a member of class IDefinition.

73.2.7.2.8 IDefinition.GetTypes Method

Pascal

```
function GetTypes: ITypes;
```

C++

```
__fastcall ITypes GetTypes();
```

Description

This is GetTypes, a member of class IDefinition.

73.2.7.2.9 IDefinition.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IDefinition.

73.2.7.2.10 IDefinition.SetTargetNamespace Method

Pascal

```
procedure SetTargetNamespace(Value: WideString);
```

C++

```
__fastcall SetTargetNamespace(BSTR Value);
```

Description

This is SetTargetNamespace, a member of class IDefinition.

73.2.7.3 IDefinition Properties

The properties of the IDefinition class are listed here.

Public Properties

	Name	Description
R	Bindings (see page 7099)	This is Bindings, a member of class IDefinition.
R	Imports (see page 7099)	This is Imports, a member of class IDefinition.
R	Messages (see page 7099)	This is Messages, a member of class IDefinition.
	Name (see page 7099)	This is Name, a member of class IDefinition.
R	PortTypes (see page 7099)	This is PortTypes, a member of class IDefinition.
R	Services (see page 7100)	This is Services, a member of class IDefinition.
	TargetNamespace (see page 7100)	This is TargetNamespace, a member of class IDefinition.
R	Types (see page 7100)	This is Types, a member of class IDefinition.

73.2.7.3.1 IDefinition.Bindings Property

Pascal

```
property Bindings: IBindings;
```

C++

```
__property IBindings Bindings;
```

Description

This is Bindings, a member of class IDefinition.

73.2.7.3.2 IDefinition.Imports Property

Pascal

```
property Imports: IImports;
```

C++

```
__property IImports Imports;
```

Description

This is Imports, a member of class IDefinition.

73.2.7.3.3 IDefinition.Messages Property

Pascal

```
property Messages: IMessages;
```

C++

```
__property IMessages Messages;
```

Description

This is Messages, a member of class IDefinition.

73.2.7.3.4 IDefinition.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IDefinition.

73.2.7.3.5 IDefinition.PortTypes Property

Pascal

```
property PortTypes: IPortTypes;
```

C++

```
__property IPortTypes PortTypes;
```

Description

This is PortTypes, a member of class IDefinition.

73.2.7.3.6 IDefinition.Services Property

Pascal

```
property Services: IServices;
```

C++

```
__property IServices Services;
```

Description

This is Services, a member of class IDefinition.

73.2.7.3.7 IDefinition.TargetNamespace Property

Pascal

```
property TargetNamespace: WideString;
```

C++

```
__property BSTR TargetNamespace;
```

Description

This is TargetNamespace, a member of class IDefinition.

73.2.7.3.8 IDefinition.Types Property

Pascal

```
property Types: ITypes;
```

C++

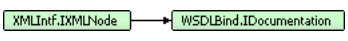
```
__property ITypes Types;
```

Description

This is Types, a member of class IDefinition.

73.2.8 IDocumentation Interface

Class Hierarchy



Pascal

```
IDocumentation = interface(IXMLNode);
```

C++

```
__interface IDocumentation : public IXMLNode;
```

File

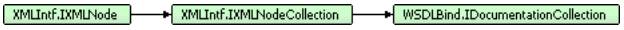
WSDLBind

Description

This is class WSDLBind.IDocumentation.

73.2.9 IDocumentationCollection Interface

Class Hierarchy



Pascal

```
IDocumentationCollection = interface(IXMLNodeCollection);
```

C++

```
__interface IDocumentationCollection : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IDocumentationCollection.

73.2.9.1 IDocumentationCollection Members

The following tables list the members exposed by IDocumentationCollection.

Public Methods

	Name	Description
	Add (see page 7103)	This is Add, a member of class IDocumentationCollection.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDocumentation (see page 7103)	This is GetDocumentation, a member of class IDocumentationCollection.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.

	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7103)	This is Insert, a member of class IDocumentationCollection.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	Documentation (see page 7104)	This is Documentation, a member of class IDocumentationCollection.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.9.2 IDocumentationCollection Methods

The methods of the IDocumentationCollection class are listed here.

Public Methods

	Name	Description
	Add (see page 7103)	This is Add, a member of class IDocumentationCollection.
	GetDocumentation (see page 7103)	This is GetDocumentation, a member of class IDocumentationCollection.
	Insert (see page 7103)	This is Insert, a member of class IDocumentationCollection.

73.2.9.2.1 IDocumentationCollection.Add Method

Pascal

```
function Add(const Text: WideString): IDocumentation;
```

C++

```
__fastcall IDocumentation Add(const BSTR Text);
```

Description

This is Add, a member of class IDocumentationCollection.

73.2.9.2.2 IDocumentationCollection.GetDocumentation Method

Pascal

```
function GetDocumentation(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation GetDocumentation(int Index);
```

Description

This is GetDocumentation, a member of class IDocumentationCollection.

73.2.9.2.3 IDocumentationCollection.Insert Method

Pascal

```
function Insert(Index: Integer): IDocumentation;
```

C++

```
__fastcall IDocumentation Insert(int Index);
```

Description

This is Insert, a member of class IDocumentationCollection.

73.2.9.3 IDocumentationCollection Properties

The properties of the IDocumentationCollection class are listed here.

Public Properties

	Name	Description
R	Documentation (see page 7104)	This is Documentation, a member of class IDocumentationCollection.

73.2.9.3.1 IDocumentationCollection.Documentation Property

Pascal

```
property Documentation [Index: Integer]: IDocumentation;
```

C++

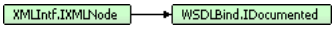
```
__property IDocumentation Documentation[int Index];
```

Description

This is Documentation, a member of class IDocumentationCollection.

73.2.10 IDocumented Interface

Class Hierarchy



Pascal

```
IDocumented = interface(IXMLNode);
```

C++

```
__interface IDocumented : public IXMLNode;
```

File

WSDLBind

Description

This is class WSDLBind.IDocumented.

73.2.10.1 IDocumented Members

The following tables list the members exposed by IDocumented.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.

	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.10.2 IDocumented Methods

The methods of the IDocumented class are listed here.

Public Methods

	Name	Description
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.

73.2.10.2.1 IDocumented.GetDocumentationCollection Method

Pascal

```
function GetDocumentationCollection: IDocumentationCollection;
```

C++

```
__fastcall IDocumentationCollection GetDocumentationCollection();
```

Description

This is GetDocumentationCollection, a member of class IDocumented.

73.2.10.3 IDocumented Properties

The properties of the IDocumented class are listed here.

Public Properties

	Name	Description
	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.

73.2.10.3.1 IDocumented.Documentation Property

Pascal

```
property Documentation: IDocumentationCollection;
```

C++

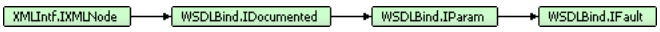
```
__property IDocumentationCollection Documentation;
```

Description

This is Documentation, a member of class IDocumented.

73.2.11 IFault Interface

Class Hierarchy



Pascal

```
IFault = interface(IParam);
```

C++

```
__interface IFault : public IParam;
```

File

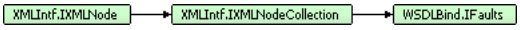
WSDLBind

Description

This is class WSDLBind.IFault.

73.2.12 IFaults Interface

Class Hierarchy



Pascal

```
IFaults = interface(IXMLNodeCollection);
```

C++

```
__interface IFaults : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IFaults.

73.2.12.1 IFaults Members

The following tables list the members exposed by IFaults.

Public Methods

	Name	Description
	Add (see page 7109)	This is Add, a member of class IFaults.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetFault (see page 7109)	This is GetFault, a member of class IFaults.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.

	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7109)	This is Insert, a member of class IFaults.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	Fault (see page 7110)	This is Fault, a member of class IFaults.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.12.2 IFaults Methods

The methods of the IFaults class are listed here.

Public Methods

	Name	Description
	Add (see page 7109)	This is Add, a member of class IFaults.
	GetFault (see page 7109)	This is GetFault, a member of class IFaults.
	Insert (see page 7109)	This is Insert, a member of class IFaults.

73.2.12.2.1 IFaults.Add Method

Pascal

```
function Add(const Name: WideString; const Message: WideString): IFault;
```

C++

```
__fastcall IFault Add(const BSTR Name, const BSTR Message);
```

Description

This is Add, a member of class IFaults.

73.2.12.2.2 IFaults.GetFault Method

Pascal

```
function GetFault(Index: Integer): IFault;
```

C++

```
__fastcall IFault GetFault(int Index);
```

Description

This is GetFault, a member of class IFaults.

73.2.12.2.3 IFaults.Insert Method

Pascal

```
function Insert(Index: Integer): IFault;
```

C++

```
__fastcall IFault Insert(int Index);
```

Description

This is Insert, a member of class IFaults.

73.2.12.3 IFaults Properties

The properties of the IFaults class are listed here.

Public Properties

	Name	Description
	Fault (see page 7110)	This is Fault, a member of class IFaults.

73.2.12.3.1 IFaults.Fault Property

Pascal

```
property Fault [Index: Integer]: IFault;
```

C++

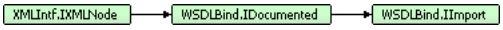
```
__property IFault Fault[int Index];
```

Description

This is Fault, a member of class IFaults.

73.2.13 IImport Interface

Class Hierarchy



Pascal

```
IImport = interface(IDocumented);
```

C++

```
__interface IImport : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IImport.

73.2.13.1 IImport Members

The following tables list the members exposed by IImport.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.

	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetLocation (see page 7112)	This is GetLocation, a member of class IImport.
	GetNamespace (see page 7112)	This is GetNamespace, a member of class IImport.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetLocation (see page 7112)	This is SetLocation, a member of class IImport.
	SetNamespace (see page 7112)	This is SetNamespace, a member of class IImport.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Location (see page 7113)	This is Location, a member of class IImport.
	Namespace (see page 7113)	This is Namespace, a member of class IImport.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.13.2 IImport Methods

The methods of the IImport class are listed here.

Public Methods

	Name	Description
	GetLocation (see page 7112)	This is GetLocation, a member of class IImport.
	GetNamespace (see page 7112)	This is GetNamespace, a member of class IImport.
	SetLocation (see page 7112)	This is SetLocation, a member of class IImport.
	SetNamespace (see page 7112)	This is SetNamespace, a member of class IImport.

73.2.13.2.1 IImport.GetLocation Method

Pascal

```
function GetLocation: WideString;
```

C++

```
__fastcall BSTR GetLocation();
```

Description

This is GetLocation, a member of class IImport.

73.2.13.2.2 IImport.GetNamespace Method

Pascal

```
function GetNamespace: WideString;
```

C++

```
__fastcall BSTR GetNamespace();
```

Description

This is GetNamespace, a member of class IImport.

73.2.13.2.3 IImport.SetLocation Method

Pascal

```
procedure SetLocation(Value: WideString);
```

C++

```
__fastcall SetLocation(BSTR Value);
```

Description

This is SetLocation, a member of class IImport.

73.2.13.2.4 IImport.SetNamespace Method

Pascal

```
procedure SetNamespace(Value: WideString);
```

C++

```
__fastcall SetNamespace(BSTR Value);
```

Description

This is SetNamespace, a member of class IImport.

73.2.13.3 IImport Properties

The properties of the IImport class are listed here.

Public Properties

	Name	Description
	Location (see page 7113)	This is Location, a member of class IImport.
	Namespace (see page 7113)	This is Namespace, a member of class IImport.

73.2.13.3.1 IImport.Location Property

Pascal

```
property Location: WideString;
```

C++

```
__property BSTR Location;
```

Description

This is Location, a member of class IImport.

73.2.13.3.2 IImport.Namespace Property

Pascal

```
property Namespace: WideString;
```

C++

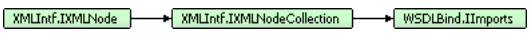
```
__property BSTR Namespace;
```

Description

This is Namespace, a member of class IImport.

73.2.14 IImports Interface

Class Hierarchy



Pascal

```
IImports = interface(IXMLNodeCollection);
```

C++

```
__interface IImports : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IImports.

73.2.14.1 IImports Members

The following tables list the members exposed by IImports.

Public Methods

	Name	Description
	Add (see page 7115)	This is Add, a member of class IImports.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetImport (see page 7116)	This is GetImport, a member of class IImports.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7116)	This is Insert, a member of class IImports.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	Import (see page 7116)	This is Import, a member of class IImports.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.14.2 IImports Methods

The methods of the IImports class are listed here.

Public Methods

	Name	Description
	Add (see page 7115)	This is Add, a member of class IImports.
	GetImport (see page 7116)	This is GetImport, a member of class IImports.
	Insert (see page 7116)	This is Insert, a member of class IImports.

73.2.14.2.1 IImports.Add Method

Pascal

```
function Add(const Namespace: WideString; const Location: WideString): IImport;
```

C++

```
__fastcall IImport Add(const BSTR Namespace, const BSTR Location);
```

Description

This is Add, a member of class IImports.

73.2.14.2.2 IImports.GetImport Method

Pascal

```
function GetImport(Index: Integer): IImport;
```

C++

```
__fastcall IImport GetImport(int Index);
```

Description

This is GetImport, a member of class IImports.

73.2.14.2.3 IImports.Insert Method

Pascal

```
function Insert(Index: Integer): IImport;
```

C++

```
__fastcall IImport Insert(int Index);
```

Description

This is Insert, a member of class IImports.

73.2.14.3 IImports Properties

The properties of the IImports class are listed here.

Public Properties

	Name	Description
	Import (see page 7116)	This is Import, a member of class IImports.

73.2.14.3.1 IImports.Import Property

Pascal

```
property Import [Index: Integer]: IImport;
```

C++

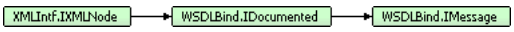
```
__property IImport Import[int Index];
```

Description

This is Import, a member of class IImports.

73.2.15 IMessage Interface

Class Hierarchy



Pascal

```
IMessage = interface(IDocumented);
```

C++

```
__interface IMessage : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IMessage.

73.2.15.1 IMessage Members

The following tables list the members exposed by IMessage.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7118)	This is GetName, a member of class IMessage.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetParts (see page 7119)	This is GetParts, a member of class IMessage.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7119)	This is SetName, a member of class IMessage.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7119)	This is Name, a member of class IMessage.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Parts (see page 7119)	This is Parts, a member of class IMessage.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.15.2 IMessage Methods

The methods of the IMessage class are listed here.

Public Methods

	Name	Description
	GetName (see page 7118)	This is GetName, a member of class IMessage.
	GetParts (see page 7119)	This is GetParts, a member of class IMessage.
	SetName (see page 7119)	This is SetName, a member of class IMessage.

73.2.15.2.1 IMessage.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IMessage.

73.2.15.2.2 IMessage.GetParts Method

Pascal

```
function GetParts: IParts;
```

C++

```
__fastcall IParts GetParts();
```

Description

This is GetParts, a member of class IMessage.

73.2.15.2.3 IMessage.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IMessage.

73.2.15.3 IMessage Properties

The properties of the IMessage class are listed here.

Public Properties

	Name	Description
	Name (see page 7119)	This is Name, a member of class IMessage.
R	Parts (see page 7119)	This is Parts, a member of class IMessage.

73.2.15.3.1 IMessage.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IMessage.

73.2.15.3.2 IMessage.Parts Property

Pascal

```
property Parts: IParts;
```

C++

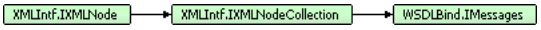
```
__property IParts Parts;
```

Description

This is Parts, a member of class IMessage.

73.2.16 IMessages Interface

Class Hierarchy



Pascal

```
IMessages = interface(IXMLNodeCollection);
```

C++

```
__interface IMessages : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IMessages.

73.2.16.1 IMessages Members

The following tables list the members exposed by IMessages.

Public Methods

	Name	Description
	Add (see page 7122)	This is Add, a member of class IMessages.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 7122)	This is GetMessage, a member of class IMessages.

	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7122)	This is Insert, a member of class IMessages.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	Message (see page 7123)	This is Message, a member of class IMessages.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.16.2 IMessages Methods

The methods of the IMessages class are listed here.

Public Methods

	Name	Description
	Add (see page 7122)	This is Add, a member of class IMessages.
	GetMessage (see page 7122)	This is GetMessage, a member of class IMessages.
	Insert (see page 7122)	This is Insert, a member of class IMessages.

73.2.16.2.1 IMessages.Add Method

Pascal

```
function Add(const Name: WideString): IMessage;
```

C++

```
__fastcall IMessage Add(const BSTR Name);
```

Description

This is Add, a member of class IMessages.

73.2.16.2.2 IMessages.GetMessage Method

Pascal

```
function GetMessage(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage GetMessage(int Index);
```

Description

This is GetMessage, a member of class IMessages.

73.2.16.2.3 IMessages.Insert Method

Pascal

```
function Insert(Index: Integer): IMessage;
```

C++

```
__fastcall IMessage Insert(int Index);
```

Description

This is Insert, a member of class IMessages.

73.2.16.3 IMessages Properties

The properties of the IMessages class are listed here.

Public Properties

	Name	Description
R	Message (see page 7123)	This is Message, a member of class IMessages.

73.2.16.3.1 IMessages.Message Property

Pascal

```
property Message [Index: Integer]: IMessage;
```

C++

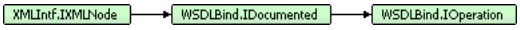
```
__property IMessage Message[int Index];
```

Description

This is Message, a member of class IMessages.

73.2.17 IOperation Interface

Class Hierarchy



Pascal

```
IOperation = interface(IDocumented);
```

C++

```
__interface IOperation : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IOperation.

73.2.17.1 IOperation Members

The following tables list the members exposed by IOperation.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetFaults (see page 7125)	This is GetFaults, a member of class IOperation.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.

	GetInput (see page 7125)	This is GetInput, a member of class IOperation.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7125)	This is GetName, a member of class IOperation.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOutput (see page 7126)	This is GetOutput, a member of class IOperation.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7126)	This is SetName, a member of class IOperation.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	Faults (see page 7126)	This is Faults, a member of class IOperation.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	Input (see page 7126)	This is Input, a member of class IOperation.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7127)	This is Name, a member of class IOperation.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Output (see page 7127)	This is Output, a member of class IOperation.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.

	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.17.2 IOperation Methods

The methods of the IOperation class are listed here.

Public Methods

	Name	Description
	GetFaults (see page 7125)	This is GetFaults, a member of class IOperation.
	GetInput (see page 7125)	This is GetInput, a member of class IOperation.
	GetName (see page 7125)	This is GetName, a member of class IOperation.
	GetOutput (see page 7126)	This is GetOutput, a member of class IOperation.
	SetName (see page 7126)	This is SetName, a member of class IOperation.

73.2.17.2.1 IOperation.GetFaults Method

Pascal

```
function GetFaults: IFaults;
```

C++

```
__fastcall IFaults GetFaults();
```

Description

This is GetFaults, a member of class IOperation.

73.2.17.2.2 IOperation.GetInput Method

Pascal

```
function GetInput: IParam;
```

C++

```
__fastcall IParam GetInput();
```

Description

This is GetInput, a member of class IOperation.

73.2.17.2.3 IOperation.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IOperation.

73.2.17.2.4 IOperation.GetOutput Method

Pascal

```
function GetOutput: IParam;
```

C++

```
__fastcall IParam GetOutput();
```

Description

This is GetOutput, a member of class IOperation.

73.2.17.2.5 IOperation.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IOperation.

73.2.17.3 IOperation Properties

The properties of the IOperation class are listed here.

Public Properties

	Name	Description
R	Faults (see page 7126)	This is Faults, a member of class IOperation.
R	Input (see page 7126)	This is Input, a member of class IOperation.
	Name (see page 7127)	This is Name, a member of class IOperation.
R	Output (see page 7127)	This is Output, a member of class IOperation.

73.2.17.3.1 IOperation.Faults Property

Pascal

```
property Faults: IFaults;
```

C++

```
__property IFaults Faults;
```

Description

This is Faults, a member of class IOperation.

73.2.17.3.2 IOperation.Input Property

Pascal

```
property Input: IParam;
```

C++

```
__property IParam Input;
```

Description

This is Input, a member of class IOperation.

73.2.17.3.3 IOperation.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IOperation.

73.2.17.3.4 IOperation.Output Property

Pascal

```
property Output: IParam;
```

C++

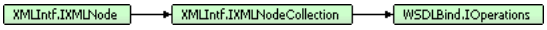
```
__property IParam Output;
```

Description

This is Output, a member of class IOperation.

73.2.18 IOperations Interface

Class Hierarchy



Pascal

```
IOperations = interface(IXMLNodeCollection);
```

C++

```
__interface IOperations : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IOperations.

73.2.18.1 IOperations Members

The following tables list the members exposed by IOperations.

Public Methods

	Name	Description
	Add (see page 7129)	This is the overview for the Add method overload.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOperation (see page 7130)	This is GetOperation, a member of class IOperations.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7130)	This is Insert, a member of class IOperations.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.

	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Operation (see page 7130)	This is Operation, a member of class IOperations.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.18.2 IOperations Methods

The methods of the IOperations class are listed here.

Public Methods

	Name	Description
	Add (see page 7129)	This is the overview for the Add method overload.
	GetOperation (see page 7130)	This is GetOperation, a member of class IOperations.
	Insert (see page 7130)	This is Insert, a member of class IOperations.

73.2.18.2.1 Add Method

This is the overview for the Add method overload.

Overload List

	Name	Description
	IOperations.Add (WideString) (see page 7129)	This is Add, a member of class IOperations.
	IOperations.Add (WideString, WideString, WideString, WideString) (see page 7130)	This is Add, a member of class IOperations.

73.2.18.2.1.1 IOperations.Add Method (WideString)

Pascal

```
function Add(const Name: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name);
```

Description

This is Add, a member of class IOperations.

73.2.18.2.1.2 IOperations.Add Method (WideString, WideString, WideString, WideString)

Pascal

```
function Add(const Name: WideString; const InputMessage: WideString; const InputName:
WideString; const OutputMessage: WideString): IOperation; overload;
```

C++

```
__fastcall IOperation Add(const BSTR Name, const BSTR InputMessage, const BSTR InputName,
const BSTR OutputMessage);
```

Description

This is Add, a member of class IOperations.

73.2.18.2.2 IOperations.GetOperation Method

Pascal

```
function GetOperation(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation GetOperation(int Index);
```

Description

This is GetOperation, a member of class IOperations.

73.2.18.2.3 IOperations.Insert Method

Pascal

```
function Insert(Index: Integer): IOperation;
```

C++

```
__fastcall IOperation Insert(int Index);
```

Description

This is Insert, a member of class IOperations.

73.2.18.3 IOperations Properties

The properties of the IOperations class are listed here.

Public Properties

	Name	Description
	Operation (see page 7130)	This is Operation, a member of class IOperations.

73.2.18.3.1 IOperations.Operation Property

Pascal

```
property Operation [Index: Integer]: IOperation;
```

C++

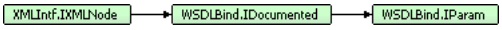
```
__property IOperation Operation[int Index];
```

Description

This is Operation, a member of class IOperations.

73.2.19 IParam Interface

Class Hierarchy



Pascal

```
IParam = interface(IDocumented);
```

C++

```
__interface IParam : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IParam.

73.2.19.1 IParam Members

The following tables list the members exposed by IParam.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetMessage (see page 7133)	This is GetMessage, a member of class IParam.
	GetName (see page 7133)	This is GetName, a member of class IParam.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.

	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetMessage (see page 7133)	This is SetMessage, a member of class IParam.
	SetName (see page 7133)	This is SetName, a member of class IParam.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Message (see page 7134)	This is Message, a member of class IParam.
	Name (see page 7134)	This is Name, a member of class IParam.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.19.2 IParam Methods

The methods of the IParam class are listed here.

Public Methods

	Name	Description
	GetMessage (see page 7133)	This is GetMessage, a member of class IParam.
	GetName (see page 7133)	This is GetName, a member of class IParam.
	SetMessage (see page 7133)	This is SetMessage, a member of class IParam.
	SetName (see page 7133)	This is SetName, a member of class IParam.

73.2.19.2.1 IParam.GetMessage Method

Pascal

```
function GetMessage: WideString;
```

C++

```
__fastcall BSTR GetMessage();
```

Description

This is GetMessage, a member of class IParam.

73.2.19.2.2 IParam.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IParam.

73.2.19.2.3 IParam.SetMessage Method

Pascal

```
procedure SetMessage(Value: WideString);
```

C++

```
__fastcall SetMessage(BSTR Value);
```

Description

This is SetMessage, a member of class IParam.

73.2.19.2.4 IParam.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IParam.

73.2.19.3 IParam Properties

The properties of the IParam class are listed here.

Public Properties

	Name	Description
	Message (see page 7134)	This is Message, a member of class IParam.
	Name (see page 7134)	This is Name, a member of class IParam.

73.2.19.3.1 IParam.Message Property

Pascal

```
property Message: WideString;
```

C++

```
__property BSTR Message;
```

Description

This is Message, a member of class IParam.

73.2.19.3.2 IParam.Name Property

Pascal

```
property Name: WideString;
```

C++

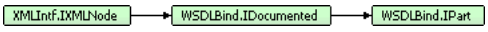
```
__property BSTR Name;
```

Description

This is Name, a member of class IParam.

73.2.20 IPart Interface

Class Hierarchy



Pascal

```
IPart = interface(IDocumented);
```

C++

```
__interface IPart : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPart.

73.2.20.1 IPart Members

The following tables list the members exposed by IPart.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetElement (see page 7136)	This is GetElement, a member of class IPart.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7137)	This is GetName, a member of class IPart.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetType_ (see page 7137)	This is GetType_, a member of class IPart.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetElement (see page 7137)	This is SetElement, a member of class IPart.
	SetName (see page 7137)	This is SetName, a member of class IPart.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	SetType_ (see page 7137)	This is SetType_, a member of class IPart.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
	Element (see page 7138)	This is Element, a member of class IPart.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7138)	This is Name, a member of class IPart.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
	Type_ (see page 7138)	This is Type_, a member of class IPart.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.20.2 IPart Methods

The methods of the IPart class are listed here.

Public Methods

	Name	Description
	GetElement (see page 7136)	This is GetElement, a member of class IPart.
	GetName (see page 7137)	This is GetName, a member of class IPart.
	GetType_ (see page 7137)	This is GetType_, a member of class IPart.
	SetElement (see page 7137)	This is SetElement, a member of class IPart.
	SetName (see page 7137)	This is SetName, a member of class IPart.
	SetType_ (see page 7137)	This is SetType_, a member of class IPart.

73.2.20.2.1 IPart.GetElement Method

Pascal

```
function GetElement: WideString;
```

C++

```
__fastcall BSTR GetElement();
```

Description

This is GetElement, a member of class IPart.

73.2.20.2.2 IPart.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPart.

73.2.20.2.3 IPart.GetType_ Method

Pascal

```
function GetType_: WideString;
```

C++

```
__fastcall BSTR GetType_();
```

Description

This is GetType_, a member of class IPart.

73.2.20.2.4 IPart.SetElement Method

Pascal

```
procedure SetElement(Value: WideString);
```

C++

```
__fastcall SetElement(BSTR Value);
```

Description

This is SetElement, a member of class IPart.

73.2.20.2.5 IPart.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPart.

73.2.20.2.6 IPart.SetType_ Method

Pascal

```
procedure SetType_(Value: WideString);
```

C++

```
__fastcall SetType_(BSTR Value);
```

Description

This is SetType_, a member of class IPart.

73.2.20.3 IPart Properties

The properties of the IPart class are listed here.

Public Properties

	Name	Description
	Element (see page 7138)	This is Element, a member of class IPart.
	Name (see page 7138)	This is Name, a member of class IPart.
	Type_ (see page 7138)	This is Type_, a member of class IPart.

73.2.20.3.1 IPart.Element Property

Pascal

```
property Element: WideString;
```

C++

```
__property BSTR Element;
```

Description

This is Element, a member of class IPart.

73.2.20.3.2 IPart.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IPart.

73.2.20.3.3 IPart.Type_ Property

Pascal

```
property Type_: WideString;
```

C++

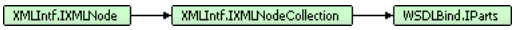
```
__property BSTR Type_;
```

Description

This is Type_, a member of class IPart.

73.2.21 IParts Interface

Class Hierarchy



Pascal

```
IParts = interface(IXMLNodeCollection);
```

C++

```
__interface IParts : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IParts.

73.2.21.1 IParts Members

The following tables list the members exposed by IParts.

Public Methods

	Name	Description
	Add (see page 7141)	This is Add, a member of class IParts.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.

	GetPart (see page 7141)	This is GetPart, a member of class IParts.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7141)	This is Insert, a member of class IParts.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Part (see page 7142)	This is Part, a member of class IParts.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.21.2 IParts Methods

The methods of the IParts class are listed here.

Public Methods

	Name	Description
	Add (see page 7141)	This is Add, a member of class IParts.
	GetPart (see page 7141)	This is GetPart, a member of class IParts.
	Insert (see page 7141)	This is Insert, a member of class IParts.

73.2.21.2.1 IParts.Add Method

Pascal

```
function Add(const Name: WideString; const Element: WideString; const AType: WideString = ''):
IPart;
```

C++

```
__fastcall IPart Add(const BSTR Name, const BSTR Element, const BSTR AType = '');
```

Description

This is Add, a member of class IParts.

73.2.21.2.2 IParts.GetPart Method

Pascal

```
function GetPart(Index: Integer): IPart;
```

C++

```
__fastcall IPart GetPart(int Index);
```

Description

This is GetPart, a member of class IParts.

73.2.21.2.3 IParts.Insert Method

Pascal

```
function Insert(Index: Integer): IPart;
```

C++

```
__fastcall IPart Insert(int Index);
```

Description

This is Insert, a member of class IParts.

73.2.21.3 IParts Properties

The properties of the IParts class are listed here.

Public Properties

	Name	Description
R	Part (see page 7142)	This is Part, a member of class IParts.

73.2.21.3.1 IParts.Part Property

Pascal

```
property Part [Index: Integer]: IPart;
```

C++

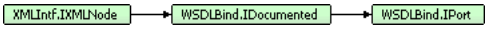
```
__property IPart Part[int Index];
```

Description

This is Part, a member of class IParts.

73.2.22 IPort Interface

Class Hierarchy



Pascal

```
IPort = interface(IDocumented);
```

C++

```
__interface IPort : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPort.

73.2.22.1 IPort Members

The following tables list the members exposed by IPort.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetBinding (see page 7144)	This is GetBinding, a member of class IPort.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.

	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7144)	This is GetName, a member of class IPort.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetBinding (see page 7144)	This is SetBinding, a member of class IPort.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7144)	This is SetName, a member of class IPort.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
	Binding (see page 7145)	This is Binding, a member of class IPort.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7145)	This is Name, a member of class IPort.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.22.2 IPort Methods

The methods of the IPort class are listed here.

Public Methods

	Name	Description
	GetBinding (see page 7144)	This is GetBinding, a member of class IPort.
	GetName (see page 7144)	This is GetName, a member of class IPort.
	SetBinding (see page 7144)	This is SetBinding, a member of class IPort.
	SetName (see page 7144)	This is SetName, a member of class IPort.

73.2.22.2.1 IPort.GetBinding Method

Pascal

```
function GetBinding: WideString;
```

C++

```
__fastcall BSTR GetBinding();
```

Description

This is GetBinding, a member of class IPort.

73.2.22.2.2 IPort.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPort.

73.2.22.2.3 IPort.SetBinding Method

Pascal

```
procedure SetBinding(Value: WideString);
```

C++

```
__fastcall SetBinding(BSTR Value);
```

Description

This is SetBinding, a member of class IPort.

73.2.22.2.4 IPort.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPort.

73.2.22.3 IPort Properties

The properties of the IPort class are listed here.

Public Properties

	Name	Description
	Binding (see page 7145)	This is Binding, a member of class IPort.
	Name (see page 7145)	This is Name, a member of class IPort.

73.2.22.3.1 IPort.Binding Property

Pascal

```
property Binding: WideString;
```

C++

```
__property BSTR Binding;
```

Description

This is Binding, a member of class IPort.

73.2.22.3.2 IPort.Name Property

Pascal

```
property Name: WideString;
```

C++

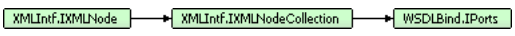
```
__property BSTR Name;
```

Description

This is Name, a member of class IPort.

73.2.23 IPorts Interface

Class Hierarchy



Pascal

```
IPorts = interface(IXMLNodeCollection);
```

C++

```
__interface IPorts : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IPorts.

73.2.23.1 IPorts Members

The following tables list the members exposed by IPorts.

Public Methods

	Name	Description
	Add (see page 7147)	This is Add, a member of class IPorts.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPort (see page 7148)	This is GetPort, a member of class IPorts.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7148)	This is Insert, a member of class IPorts.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Port (see page 7148)	This is Port, a member of class IPorts.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.23.2 IPorts Methods

The methods of the IPorts class are listed here.

Public Methods

	Name	Description
	Add (see page 7147)	This is Add, a member of class IPorts.
	GetPort (see page 7148)	This is GetPort, a member of class IPorts.
	Insert (see page 7148)	This is Insert, a member of class IPorts.

73.2.23.2.1 IPorts.Add Method

Pascal

```
function Add(const Name: WideString; const ABinding: WideString): IPort;
```

C++

```
__fastcall IPort Add(const BSTR Name, const BSTR ABinding);
```

Description

This is Add, a member of class IPorts.

73.2.23.2.2 IPorts.GetPort Method

Pascal

```
function GetPort(Index: Integer): IPort;
```

C++

```
__fastcall IPort GetPort(int Index);
```

Description

This is GetPort, a member of class IPorts.

73.2.23.2.3 IPorts.Insert Method

Pascal

```
function Insert(Index: Integer): IPort;
```

C++

```
__fastcall IPort Insert(int Index);
```

Description

This is Insert, a member of class IPorts.

73.2.23.3 IPorts Properties

The properties of the IPorts class are listed here.

Public Properties

	Name	Description
	Port (see page 7148)	This is Port, a member of class IPorts.

73.2.23.3.1 IPorts.Port Property

Pascal

```
property Port [Index: Integer]: IPort;
```

C++

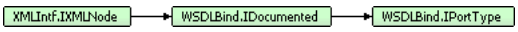
```
__property IPort Port[int Index];
```

Description

This is Port, a member of class IPorts.

73.2.24 IPortType Interface

Class Hierarchy



Pascal

```
IPortType = interface(IDocumented);
```

C++

```
__interface IPortType : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IPortType.

73.2.24.1 IPortType Members

The following tables list the members exposed by IPortType.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7150)	This is GetName, a member of class IPortType.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOperations (see page 7151)	This is GetOperations, a member of class IPortType.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7151)	This is SetName, a member of class IPortType.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7151)	This is Name, a member of class IPortType.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Operations (see page 7151)	This is Operations, a member of class IPortType.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.24.2 IPortType Methods

The methods of the IPortType class are listed here.

Public Methods

	Name	Description
	GetName (see page 7150)	This is GetName, a member of class IPortType.
	GetOperations (see page 7151)	This is GetOperations, a member of class IPortType.
	SetName (see page 7151)	This is SetName, a member of class IPortType.

73.2.24.2.1 IPortType.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IPortType.

73.2.24.2.2 IPortType.GetOperations Method

Pascal

```
function GetOperations: IOperations;
```

C++

```
__fastcall IOperations GetOperations();
```

Description

This is GetOperations, a member of class IPortType.

73.2.24.2.3 IPortType.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IPortType.

73.2.24.3 IPortType Properties

The properties of the IPortType class are listed here.

Public Properties

	Name	Description
	Name (see page 7151)	This is Name, a member of class IPortType.
R	Operations (see page 7151)	This is Operations, a member of class IPortType.

73.2.24.3.1 IPortType.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IPortType.

73.2.24.3.2 IPortType.Operations Property

Pascal

```
property Operations: IOperations;
```

C++

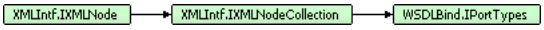
```
__property IOperations Operations;
```

Description

This is Operations, a member of class IPortType.

73.2.25 IPortTypes Interface

Class Hierarchy



Pascal

```
IPortTypes = interface(IXMLNodeCollection);
```

C++

```
__interface IPortTypes : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IPortTypes.

73.2.25.1 IPortTypes Members

The following tables list the members exposed by IPortTypes.

Public Methods

	Name	Description
	Add (see page 7154)	This is Add, a member of class IPortTypes.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.

	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPortType (see page 7154)	This is GetPortType, a member of class IPortTypes.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7154)	This is Insert, a member of class IPortTypes.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	PortType (see page 7155)	This is PortType, a member of class IPortTypes.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.25.2 IPortTypes Methods

The methods of the IPortTypes class are listed here.

Public Methods

	Name	Description
	Add (see page 7154)	This is Add, a member of class IPortTypes.
	GetPortType (see page 7154)	This is GetPortType, a member of class IPortTypes.
	Insert (see page 7154)	This is Insert, a member of class IPortTypes.

73.2.25.2.1 IPortTypes.Add Method

Pascal

```
function Add(const Name: WideString): IPortType;
```

C++

```
__fastcall IPortType Add(const BSTR Name);
```

Description

This is Add, a member of class IPortTypes.

73.2.25.2.2 IPortTypes.GetPortType Method

Pascal

```
function GetPortType(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType GetPortType(int Index);
```

Description

This is GetPortType, a member of class IPortTypes.

73.2.25.2.3 IPortTypes.Insert Method

Pascal

```
function Insert(Index: Integer): IPortType;
```

C++

```
__fastcall IPortType Insert(int Index);
```

Description

This is Insert, a member of class IPortTypes.

73.2.25.3 IPortTypes Properties

The properties of the IPortTypes class are listed here.

Public Properties

	Name	Description
R	PortType (see page 7155)	This is PortType, a member of class IPortTypes.

73.2.25.3.1 IPortTypes.PortType Property

Pascal

```
property PortType [Index: Integer]: IPortType;
```

C++

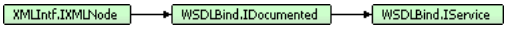
```
__property IPortType PortType[int Index];
```

Description

This is PortType, a member of class IPortTypes.

73.2.26 IService Interface

Class Hierarchy



Pascal

```
IService = interface(IDocumented);
```

C++

```
__interface IService : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.IService.

73.2.26.1 IService Members

The following tables list the members exposed by IService.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.

	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetName (see page 7157)	This is GetName, a member of class IService.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPorts (see page 7157)	This is GetPorts, a member of class IService.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetName (see page 7157)	This is SetName, a member of class IService.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	Name (see page 7158)	This is Name, a member of class IService.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Ports (see page 7158)	This is Ports, a member of class IService.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.26.2 IService Methods

The methods of the IService class are listed here.

Public Methods

	Name	Description
	GetName (see page 7157)	This is GetName, a member of class IService.
	GetPorts (see page 7157)	This is GetPorts, a member of class IService.
	SetName (see page 7157)	This is SetName, a member of class IService.

73.2.26.2.1 IService.GetName Method

Pascal

```
function GetName: WideString;
```

C++

```
__fastcall BSTR GetName();
```

Description

This is GetName, a member of class IService.

73.2.26.2.2 IService.GetPorts Method

Pascal

```
function GetPorts: IPorts;
```

C++

```
__fastcall IPorts GetPorts();
```

Description

This is GetPorts, a member of class IService.

73.2.26.2.3 IService.SetName Method

Pascal

```
procedure SetName(Value: WideString);
```

C++

```
__fastcall SetName(BSTR Value);
```

Description

This is SetName, a member of class IService.

73.2.26.3 IService Properties

The properties of the IService class are listed here.

Public Properties

	Name	Description
	Name (see page 7158)	This is Name, a member of class IService.

	Ports (see page 7158)	This is Ports, a member of class IService.

73.2.26.3.1 IService.Name Property

Pascal

```
property Name: WideString;
```

C++

```
__property BSTR Name;
```

Description

This is Name, a member of class IService.

73.2.26.3.2 IService.Ports Property

Pascal

```
property Ports: IPorts;
```

C++

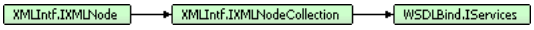
```
__property IPorts Ports;
```

Description

This is Ports, a member of class IService.

73.2.27 IServices Interface

Class Hierarchy



Pascal

```
IServices = interface(IXMLNodeCollection);
```

C++

```
__interface IServices : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IServices.

73.2.27.1 IServices Members

The following tables list the members exposed by IServices.

Public Methods

	Name	Description
	Add (see page 7160)	This is Add, a member of class IServices.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.

	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetService (see page 7160)	This is GetService, a member of class IServices.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	Insert (see page 7161)	This is Insert, a member of class IServices.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.

	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7693)	Indicates the node name.
	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Service (see page 7161)	This is Service, a member of class IServices.
	Text (see page 7697)	Represents the Text of the node.
	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.27.2 IServices Methods

The methods of the IServices class are listed here.

Public Methods

	Name	Description
	Add (see page 7160)	This is Add, a member of class IServices.
	GetService (see page 7160)	This is GetService, a member of class IServices.
	Insert (see page 7161)	This is Insert, a member of class IServices.

73.2.27.2.1 IServices.Add Method

Pascal

```
function Add(const Name: WideString): IService;
```

C++

```
__fastcall IService Add(const BSTR Name);
```

Description

This is Add, a member of class IServices.

73.2.27.2.2 IServices.GetService Method

Pascal

```
function GetService(Index: Integer): IService;
```

C++

```
__fastcall IService GetService(int Index);
```

Description

This is GetService, a member of class IServices.

73.2.27.2.3 IServices.Insert Method

Pascal

```
function Insert(Index: Integer): IService;
```

C++

```
__fastcall IService Insert(int Index);
```

Description

This is Insert, a member of class IServices.

73.2.27.3 IServices Properties

The properties of the IServices class are listed here.

Public Properties

	Name	Description
	Service (see page 7161)	This is Service, a member of class IServices.

73.2.27.3.1 IServices.Service Property

Pascal

```
property Service [Index: Integer]: IService;
```

C++

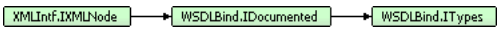
```
__property IService Service[int Index];
```

Description

This is Service, a member of class IServices.

73.2.28 ITypes Interface

Class Hierarchy



Pascal

```
ITypes = interface(IDocumented);
```

C++

```
__interface ITypes : public IDocumented;
```

File

WSDLBind

Description

This is class WSDLBind.ITypes.

73.2.28.1 ITypes Members

The following tables list the members exposed by ITypes.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDocumentationCollection (see page 7106)	This is GetDocumentationCollection, a member of class IDocumented.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetSchemaDefs (see page 7163)	This is GetSchemaDefs, a member of class ITypes.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.

	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Documentation (see page 7106)	This is Documentation, a member of class IDocumented.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	SchemaDefs (see page 7163)	This is SchemaDefs, a member of class ITypes.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.28.2 ITypes Methods

The methods of the ITypes class are listed here.

Public Methods

	Name	Description
	GetSchemaDefs (see page 7163)	This is GetSchemaDefs, a member of class ITypes.

73.2.28.2.1 ITypes.GetSchemaDefs Method

Pascal

```
function GetSchemaDefs: IXMLSchemaDefs;
```

C++

```
__fastcall IXMLSchemaDefs GetSchemaDefs();
```

Description

This is GetSchemaDefs, a member of class ITypes.

73.2.28.3 ITypes Properties

The properties of the ITypes class are listed here.

Public Properties

	Name	Description
R	SchemaDefs (see page 7163)	This is SchemaDefs, a member of class ITypes.

73.2.28.3.1 ITypes.SchemaDefs Property

Pascal

```
property SchemaDefs: IXMLSchemaDefs;
```

C++

```
__property IXMLSchemaDefs SchemaDefs;
```

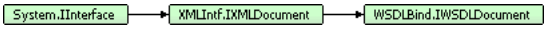
Description

This is SchemaDefs, a member of class ITypes.

73.2.29 IWSDLDocument Interface

IWSDLDocument provides low-level access to the definitions in a WSDL document.

Class Hierarchy



Pascal

```
IWSDLDocument = interface(IXMLDocument);
```

C++

```
__interface IWSDLDocument : public IXMLDocument;
```

File

WSDLBind

Description

IWSDLDocument is an interface to the TWSDLDocument (see page 7063) class. It is primarily for internal use, because it only provides access to the Definitions property, which is used for low-level access to the definitions in a WSDL document. It is generally easier to work with a TWSDLItems object. You can often obtain this from an IWSDLDocument interface by using the global Supports function (Delphi) or the Supports method (C++) to obtain an IWSDLItems interface from this one, and then obtaining a TWSDLItems object from that interface.

Note: C++ method declarations that use IWSDLDocument use the _di_IWSDLDocument type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWSDLDocument interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWSDLDocument >  _di_IWSDLDocument;
```

See Also

TWSDLDocument (see page 7063)

IWSDLItems (see page 7256)

TWSDLItems (see page 7211)

73.2.29.1 IWSDLDocument Members

The following tables list the members exposed by IWSDLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7643)	This is the overview for the AddChild method overload.

	CreateElement (see page 7644)	Creates a new element node that is associated with this document.
	CreateNode (see page 7645)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7646)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7646)	Returns the value of the Active (see page 7661) property.
	GetAsyncLoadState (see page 7647)	Returns the value of the AsyncLoadState (see page 7662) property.
	GetChildNodes (see page 7647)	Returns the value of the ChildNodes (see page 7663) property.
	GetDefinition (see page 7166)	Returns the value of the Definition (see page 7167) property.
	GetDocBinding (see page 7647)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7648)	Returns the value of the DocumentElement (see page 7663) property.
	GetDocumentNode (see page 7648)	Returns the value of the Node (see page 7665) property.
	GetDOMDocument (see page 7648)	Returns the value of the DOMDocument (see page 7663) property.
	GetEncoding (see page 7649)	Returns the value of the Encoding (see page 7664) property.
	GetFileName (see page 7649)	Returns the value of the FileName (see page 7664) property.
	GetModified (see page 7649)	Returns the value of the Modified (see page 7665) property.
	GetNodeIndentStr (see page 7650)	Returns the value of the NodeIndentStr (see page 7666) property.
	GetOptions (see page 7650)	Returns the value of the Options (see page 7666) property.
	GetParseOptions (see page 7650)	Returns the value of the ParseOptions (see page 7667) property.
	GetSchemaRef (see page 7650)	Returns the value of the SchemaRef (see page 7667) property.
	GetStandAlone (see page 7651)	Returns the value of the StandAlone (see page 7668) property.
	GetVersion (see page 7651)	Returns the value of the Version (see page 7668) property.
	GetXML (see page 7651)	Returns the value of the XML (see page 7668) property.
	IsEmptyDoc (see page 7652)	Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.
	LoadFromFile (see page 7652)	Loads an XML (see page 7668) document from disk.
	LoadFromStream (see page 7653)	Loads an XML (see page 7668) document from a stream and activates it.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7654)	Updates the parsed XML (see page 7668) document to reflect external changes.
	RegisterDocBinding (see page 7654)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7655)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7655)	Saves the XML (see page 7668) document to disk.
	SaveToStream (see page 7656)	Saves the XML (see page 7668) document to a stream.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7657)	Sets the value of the Active (see page 7661) property.
	SetDefinition (see page 7166)	Sets the value of the Definition (see page 7167) property.
	SetDocumentElement (see page 7657)	Sets the value of the DocumentElement (see page 7663) property.
	SetDOMDocument (see page 7658)	Sets the value of the DOMDocument (see page 7663) property.
	SetEncoding (see page 7658)	Sets the value of the Encoding (see page 7664) property.
	SetFileName (see page 7658)	Sets the value of the FileName (see page 7664) property.
	SetNodeIndentStr (see page 7659)	Sets the value of the NodeIndentStr (see page 7666) property.
	SetOnAsyncLoad (see page 7659)	Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.
	SetOptions (see page 7659)	Sets the value of the Options (see page 7666) property.
	SetParseOptions (see page 7660)	Sets the value of the ParseOptions (see page 7667) property.
	SetStandAlone (see page 7660)	Sets the value of the StandAlone (see page 7668) property.
	SetVersion (see page 7660)	Sets the value of the Version (see page 7668) property.
	SetXML (see page 7661)	Sets the value of the XML (see page 7668) property.

Public Properties

	Name	Description
	Active (see page 7661)	Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.

R	AsyncLoadState (see page 7662)	Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.
R	ChildNodes (see page 7663)	Lists all of the child nodes of the document.
	Definition (see page 7167)	Provides access to the definitions contained in the WSDL document.
	DocumentElement (see page 7663)	Provides access to the root node of the XML (see page 7668) document.
	DOMDocument (see page 7663)	Provides access to the DOM interface for the parsed XML (see page 7668) document.
	Encoding (see page 7664)	Indicates the character set that the XML (see page 7668) document uses to encode characters.
	FileName (see page 7664)	Indicates the XML (see page 7668) document file, if any, that this interface represents.
R	Modified (see page 7665)	Indicates whether the XML (see page 7668) document has been modified.
R	Node (see page 7665)	Provides access to the document node for the XML (see page 7668) document.
	NodeIndentStr (see page 7666)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.
	Options (see page 7666)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7667)	Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.
R	SchemaRef (see page 7667)	Returns the name of the schema associated with this XML (see page 7668) document.
	StandAlone (see page 7668)	Indicates whether the XML (see page 7668) document includes any external markup declarations.
	Version (see page 7668)	Indicates the XML (see page 7668) version that the document uses.
	XML (see page 7668)	Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.

73.2.29.2 IWSDLDocument Methods

The methods of the IWSDLDocument class are listed here.

Public Methods

	Name	Description
	GetDefinition (see page 7166)	Returns the value of the Definition (see page 7167) property.
	SetDefinition (see page 7166)	Sets the value of the Definition (see page 7167) property.

73.2.29.2.1 IWSDLDocument.GetDefinition Method

Returns the value of the Definition (see page 7167) property.

Pascal

```
function GetDefinition: IDefinition;
```

C++

```
__fastcall IDefinition GetDefinition();
```

Description

GetDefinition is the read access method for the Definition (see page 7167) property.

See Also

Definition (see page 7167)

73.2.29.2.2 IWSDLDocument.SetDefinition Method

Sets the value of the Definition (see page 7167) property.

Pascal

```
procedure SetDefinition(const Value: IDefinition);
```

C++

```
__fastcall SetDefinition(const IDefinition Value);
```

Description

SetDefinition is the write method for the Definition (see page 7167) property.

See Also

Definition (see page 7167)

73.2.29.3 IWSDLDocument Properties

The properties of the IWSDLDocument class are listed here.

Public Properties

	Name	Description
	Definition (see page 7167)	Provides access to the definitions contained in the WSDL document.

73.2.29.3.1 IWSDLDocument.Definition Property

Provides access to the definitions contained in the WSDL document.

Pascal

```
property Definition: IDefinition;
```

C++

```
__property IDefinition Definition;
```

Description

Definition is the IDefinition (see page 7094) interface of the root node in the WSDL document. This interface is implemented by the node specified by DocumentElement.

You can use the IDefinition (see page 7094) interface to access the Bindings, Messages, PortTypes, Services, and Types (see page 4649) defined in the WSDL document by reading properties of the same names. Alternately, you can use the methods of TWSDLItems. To obtain a TWSDLItems instance for this document,

1.Cast this interface to IWSDLItems using the global Supports function (C++) or the inherited Supports method (C++).

2.Use the GetWSDLItems method of that interface.

See Also

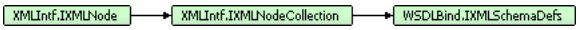
DocumentElement (see page 7663)

TWSDLItems (see page 7211)

Supports (see page 4294)

73.2.30 IXMLSchemaDefs Interface

Class Hierarchy



Pascal

```
IXMLSchemaDefs = interface(IXMLNodeCollection);
```

C++

```
__interface IXMLSchemaDefs : public IXMLNodeCollection;
```

File

WSDLBind

Description

This is class WSDLBind.IXMLSchemaDefs.

73.2.30.1 IXMLSchemaDefs Members

The following tables list the members exposed by IXMLSchemaDefs.

Public Methods

	Name	Description
	Add (see page 7170)	This is Add, a member of class IXMLSchemaDefs.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.

	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	GetXMLSchemaDef (see page 7170)	This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	SchemaDef (see page 7170)	This is SchemaDef, a member of class IXMLSchemaDefs.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

73.2.30.2 IXMLSchemaDefs Methods

The methods of the IXMLSchemaDefs class are listed here.

Public Methods

	Name	Description
	Add (see page 7170)	This is Add, a member of class IXMLSchemaDefs.

	GetXMLSchemaDef (see page 7170)	This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.

73.2.30.2.1 IXMLSchemaDefs.Add Method

Pascal

```
function Add(const NamespaceURI: WideString = ''; const TargetNamespace: WideString = ''):
IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef Add(const BSTR NamespaceURI = '', const BSTR TargetNamespace = '');
```

Description

This is Add, a member of class IXMLSchemaDefs.

73.2.30.2.2 IXMLSchemaDefs.GetXMLSchemaDef Method

Pascal

```
function GetXMLSchemaDef(Index: Integer): IXMLSchemaDef;
```

C++

```
__fastcall IXMLSchemaDef GetXMLSchemaDef(int Index);
```

Description

This is GetXMLSchemaDef, a member of class IXMLSchemaDefs.

73.2.30.3 IXMLSchemaDefs Properties

The properties of the IXMLSchemaDefs class are listed here.

Public Properties

	Name	Description
	SchemaDef (see page 7170)	This is SchemaDef, a member of class IXMLSchemaDefs.

73.2.30.3.1 IXMLSchemaDefs.SchemaDef Property

Pascal

```
property SchemaDef [Index: Integer]: IXMLSchemaDef;
```

C++

```
__property IXMLSchemaDef SchemaDef[int Index];
```

Description

This is SchemaDef, a member of class IXMLSchemaDefs.

73.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	GetDefinition (see page 7171)	This is function WSDLBind.GetDefinition.
	LoadWSDLDoc (see page 7171)	This is function WSDLBind.LoadWSDLDoc.
	NewWSDLDoc (see page 7171)	This is function WSDLBind.NewWSDLDoc.

73.3.1 WSDLBind.GetDefinition Function

Pascal

```
function GetDefinition(Doc: TXMLDocument): IDefinition;
```

C++

```
IDefinition GetDefinition(TXMLDocument Doc);
```

File

WSDLBind

Description

This is function WSDLBind.GetDefinition.

73.3.2 WSDLBind.LoadWSDLDoc Function

Pascal

```
function LoadWSDLDoc(const FileName: WideString): IWSDLDocument;
```

C++

```
IWSDLDocument LoadWSDLDoc(const BSTR FileName);
```

File

WSDLBind

Description

This is function WSDLBind.LoadWSDLDoc.

73.3.3 WSDLBind.NewWSDLDoc Function

Pascal

```
function NewWSDLDoc: IWSDLDocument;
```

C++

```
IWSDLDocument NewWSDLDoc();
```

File

WSDLBind

Description

This is function WSDLBind.NewWSDLDoc.

73.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
SBinding (see page 7173)	This is constant WSDLBind.SBinding.
SDefinitions (see page 7173)	This is constant WSDLBind.SDefinitions.
SFault (see page 7174)	This is constant WSDLBind.SFault.
SImport (see page 7174)	This is constant WSDLBind.SImport.
SInput (see page 7174)	This is constant WSDLBind.SInput.
SLocation (see page 7175)	This is constant WSDLBind.SLocation.
SMessage (see page 7175)	This is constant WSDLBind.SMessage.
SNamespace (see page 7175)	This is constant WSDLBind.SNamespace.
SOperation (see page 7176)	This is constant WSDLBind.SOperation.
SOutput (see page 7176)	This is constant WSDLBind.SOutput.
SPart (see page 7176)	This is constant WSDLBind.SPart.
SPort (see page 7177)	This is constant WSDLBind.SPort.
SPortType (see page 7177)	This is constant WSDLBind.SPortType.
SService (see page 7177)	This is constant WSDLBind.SService.
SType (see page 7178)	This is constant WSDLBind.SType.
STypes (see page 7178)	This is constant WSDLBind.STypes.

73.4.1 WSDLBind.SBinding Constant

Pascal

```
SBinding = 'binding';
```

C++

```
SBinding = 'binding';
```

File

WSDLBind

Description

This is constant WSDLBind.SBinding.

73.4.2 WSDLBind.SDefinitions Constant

Pascal

```
SDefinitions = 'definitions';
```

C++

```
SDefinitions = 'definitions';
```

File

WSDLBind

Description

This is constant WSDLBind.SDefinitions.

73.4.3 WSDLBind.SFault Constant

Pascal

```
SFault = 'fault';
```

C++

```
SFault = 'fault';
```

File

WSDLBind

Description

This is constant WSDLBind.SFault.

73.4.4 WSDLBind.SImport Constant

Pascal

```
SImport = 'import';
```

C++

```
SImport = 'import';
```

File

WSDLBind

Description

This is constant WSDLBind.SImport.

73.4.5 WSDLBind.SInput Constant

Pascal

```
SInput = 'input';
```

C++

```
SInput = 'input';
```

File

WSDLBind

Description

This is constant WSDLBind.SInput.

73.4.6 WSDLBind.SLocation Constant

Pascal

```
SLocation = 'location';
```

C++

```
SLocation = 'location';
```

File

WSDLBind

Description

This is constant WSDLBind.SLocation.

73.4.7 WSDLBind.SMessage Constant

Pascal

```
SMessage = 'message';
```

C++

```
SMessage = 'message';
```

File

WSDLBind

Description

This is constant WSDLBind.SMessage.

73.4.8 WSDLBind.SNamespace Constant

Pascal

```
SNamespace = 'namespace';
```

C++

```
SNamespace = 'namespace';
```

File

WSDLBind

Description

This is constant WSDLBind.SNamespace.

73.4.9 WSDLBind.SOperation Constant

Pascal

```
SOperation = 'operation';
```

C++

```
SOperation = 'operation';
```

File

WSDLBind

Description

This is constant WSDLBind.SOperation.

73.4.10 WSDLBind.SOutput Constant

Pascal

```
SOutput = 'output';
```

C++

```
SOutput = 'output';
```

File

WSDLBind

Description

This is constant WSDLBind.SOutput.

73.4.11 WSDLBind.SPart Constant

Pascal

```
SPart = 'part';
```

C++

```
SPart = 'part';
```

File

WSDLBind

Description

This is constant WSDLBind.SPart.

73.4.12 WSDLBind.SPort Constant

Pascal

```
SPort = 'port';
```

C++

```
SPort = 'port';
```

File

WSDLBind

Description

This is constant WSDLBind.SPort.

73.4.13 WSDLBind.SPortType Constant

Pascal

```
SPortType = 'portType';
```

C++

```
SPortType = 'portType';
```

File

WSDLBind

Description

This is constant WSDLBind.SPortType.

73.4.14 WSDLBind.SService Constant

Pascal

```
SService = 'service';
```

C++

```
SService = 'service';
```

File

WSDLBind

Description

This is constant WSDLBind.SService.

73.4.15 WSDLBind.SType Constant

Pascal

```
SType = 'type';
```

C++

```
SType = 'type';
```

File

WSDLBind

Description

This is constant WSDLBind.SType.

73.4.16 WSDLBind.STypes Constant

Pascal

```
STypes = 'types';
```

C++

```
STypes = 'types';
```

File

WSDLBind

Description

This is constant WSDLBind.STypes.

74 WSDLIntf Namespace

This is namespace WSDLIntf.

Topics

Name	Description
Classes (see page 7181)	The following table lists classes in this documentation.
Structs, Records, Enums (see page 7185)	The following table lists structs, records, enums in this documentation.
Types (see page 7187)	The following table lists types in this documentation.
Constants (see page 7188)	The following table lists constants in this documentation.

Classes

	Name	Description
	TDOMStrings (see page 7181)	This is class WSDLIntf.TDOMStrings.

Constants

Name	Description
Directives (see page 7189)	This is constant WSDLIntf.Directives.
DynArrayTypes (see page 7189)	This is constant WSDLIntf.DynArrayTypes.
Httpns (see page 7190)	This is constant WSDLIntf.Httpns.
Operators (see page 7190)	This is constant WSDLIntf.Operators.
OtherReservedWords (see page 7190)	This is constant WSDLIntf.OtherReservedWords.
ReservedWords (see page 7191)	This is constant WSDLIntf.ReservedWords.
SAddress (see page 7191)	This is constant WSDLIntf.SAddress.
SAnySimpleType (see page 7192)	This is constant WSDLIntf.SAnySimpleType.
SAnyType (see page 7192)	This is constant WSDLIntf.SAnyType.
SArray (see page 7192)	This is constant WSDLIntf.SArray.
SArrayOf (see page 7193)	This is constant WSDLIntf.SArrayOf.
SArrayType (see page 7193)	This is constant WSDLIntf.SArrayType.
SBinding (see page 7193)	This is constant WSDLIntf.SBinding.
SBody (see page 7194)	This is constant WSDLIntf.SBody.
ScalarArrayTypes (see page 7194)	This is constant WSDLIntf.ScalarArrayTypes.
SComplexType (see page 7194)	This is constant WSDLIntf.SComplexType.
SDefinitions (see page 7195)	This is constant WSDLIntf.SDefinitions.
SDynArray (see page 7195)	This is constant WSDLIntf.SDynArray.
SElement (see page 7195)	This is constant WSDLIntf.SElement.
SEncodingStyle (see page 7196)	This is constant WSDLIntf.SEncodingStyle.

SFault (see page 7196)	This is constant WSDLIntf.SFault.
SHeader (see page 7196)	This is constant WSDLIntf.SHeader.
SImport (see page 7197)	This is constant WSDLIntf.SImport.
SInOut (see page 7197)	This is constant WSDLIntf.SInOut.
SInput (see page 7197)	This is constant WSDLIntf.SInput.
SLocation (see page 7198)	This is constant WSDLIntf.SLocation.
SMessage (see page 7198)	This is constant WSDLIntf.SMessage.
SName (see page 7198)	This is constant WSDLIntf.SName.
SNameSpace (see page 7199)	This is constant WSDLIntf.SNameSpace.
SNsPrefix (see page 7199)	This is constant WSDLIntf.SNsPrefix.
Soapns (see page 7199)	This is constant WSDLIntf.Soapns.
SOperation (see page 7200)	This is constant WSDLIntf.SOperation.
SOutput (see page 7200)	This is constant WSDLIntf.SOutput.
SPart (see page 7200)	This is constant WSDLIntf.SPart.
SParts (see page 7201)	This is constant WSDLIntf.SParts.
SPort (see page 7201)	This is constant WSDLIntf.SPort.
SPortType (see page 7201)	This is constant WSDLIntf.SPortType.
SRequest (see page 7202)	This is constant WSDLIntf.SRequest.
SResponse (see page 7202)	This is constant WSDLIntf.SResponse.
SReturn (see page 7202)	This is constant WSDLIntf.SReturn.
SSchema (see page 7203)	This is constant WSDLIntf.SSchema.
SService (see page 7203)	This is constant WSDLIntf.SService.
SSoapAction (see page 7203)	This is constant WSDLIntf.SSoapAction.
SStyle (see page 7204)	This is constant WSDLIntf.SStyle.
STns (see page 7204)	This is constant WSDLIntf.STns.
STransport (see page 7204)	This is constant WSDLIntf.STransport.
SType (see page 7205)	This is constant WSDLIntf.SType.
STypes (see page 7205)	This is constant WSDLIntf.STypes.
SUnknown (see page 7205)	This is constant WSDLIntf.SUnknown.
SUse (see page 7206)	This is constant WSDLIntf.SUse.
SVerb (see page 7206)	This is constant WSDLIntf.SVerb.
tns (see page 7206)	This is constant WSDLIntf.tns.
Wsdlns (see page 7207)	This is constant WSDLIntf.Wsdlns.
xsdns (see page 7207)	This is constant WSDLIntf.xsdns.

Structs, Records, Enums

	Name	Description
	TWebServiceBindingType (see page 7185)	TWebServiceBindingType indicates the binding type of a Web Service request or response.
	TWSDLElement (see page 7186)	This is record WSDLIntf.TWSDLElement.
	TWString (see page 7186)	This is record WSDLIntf.TWString.

Types

Name	Description
TWSDLElements (see page 7187)	This is type WSDLIntf.TWSDLElements.

74.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TDOMStrings (see page 7181)	This is class WSDLIntf.TDOMStrings.

74.1.1 TDOMStrings Class

Class Hierarchy

Pascal

```
TDOMStrings = class;
```

C++

```
class TDOMStrings;
```

File

WSDLIntf

Description

This is class WSDLIntf.TDOMStrings.

74.1.1.1 TDOMStrings Members

The following tables list the members exposed by TDOMStrings.

Public Methods

	Name	Description
	Add (see page 7182)	This is Add, a member of class TDOMStrings.
	Clear (see page 7183)	This is Clear, a member of class TDOMStrings.
	Count (see page 7183)	This is Count, a member of class TDOMStrings.
	Create (see page 7182)	This is Create, a member of class TDOMStrings.
	Destroy (see page 7182)	This is Destroy, a member of class TDOMStrings.
	IndexOf (see page 7183)	This is IndexOf, a member of class TDOMStrings.
	IndexOfIgnoreCase (see page 7183)	This is IndexOfIgnoreCase, a member of class TDOMStrings.
	Insert (see page 7183)	This is Insert, a member of class TDOMStrings.

Public Properties

	Name	Description
	Strings (see page 7184)	This is Strings, a member of class TDOMStrings.

74.1.1.2 TDOMStrings.Create Constructor

Pascal

```
constructor Create;
```

C++

```
__fastcall TDOMStrings();
```

Description

This is Create, a member of class TDOMStrings.

74.1.1.3 TDOMStrings.Destroy Destructor

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TDOMStrings();
```

Description

This is Destroy, a member of class TDOMStrings.

74.1.1.4 TDOMStrings Methods

The methods of the TDOMStrings class are listed here.

Public Methods

	Name	Description
	Add (see page 7182)	This is Add, a member of class TDOMStrings.
	Clear (see page 7183)	This is Clear, a member of class TDOMStrings.
	Count (see page 7183)	This is Count, a member of class TDOMStrings.
	IndexOf (see page 7183)	This is IndexOf, a member of class TDOMStrings.
	IndexOfIgnoreCase (see page 7183)	This is IndexOfIgnoreCase, a member of class TDOMStrings.
	Insert (see page 7183)	This is Insert, a member of class TDOMStrings.

74.1.1.4.1 TDOMStrings.Add Method

Pascal

```
function Add(const S: DOMString): Integer;
```

C++

```
__fastcall int Add(const DOMString S);
```

Description

This is Add, a member of class TDOMStrings.

74.1.1.4.2 TDOMStrings.Clear Method

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

This is Clear, a member of class TDOMStrings.

74.1.1.4.3 TDOMStrings.Count Method

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

This is Count, a member of class TDOMStrings.

74.1.1.4.4 TDOMStrings.IndexOf Method

Pascal

```
function IndexOf(const S: DOMString): Integer;
```

C++

```
__fastcall int IndexOf(const DOMString S);
```

Description

This is IndexOf, a member of class TDOMStrings.

74.1.1.4.5 TDOMStrings.IndexOfIgnoreCase Method

Pascal

```
function IndexOfIgnoreCase(const S: DOMString): Integer;
```

C++

```
__fastcall int IndexOfIgnoreCase(const DOMString S);
```

Description

This is IndexOfIgnoreCase, a member of class TDOMStrings.

74.1.1.4.6 TDOMStrings.Insert Method

Pascal

```
procedure Insert(Index: Integer; const S: DOMString);
```

C++

```
__fastcall Insert(int Index, const DOMString S);
```

Description

This is Insert, a member of class TDOMStrings.

74.1.1.5 TDOMStrings Properties

The properties of the TDOMStrings class are listed here.

Public Properties

	Name	Description
	Strings (see page 7184)	This is Strings, a member of class TDOMStrings.

74.1.1.5.1 TDOMStrings.Strings Property

Pascal

```
property Strings [Index: Integer]: DOMString;
```

C++

```
__property DOMString Strings[int Index];
```

Description

This is Strings, a member of class TDOMStrings.

74.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TWebServiceBindingType (see page 7185)	TWebServiceBindingType indicates the binding type of a Web Service request or response.
	TWSDLElement (see page 7186)	This is record WSDLIntf.TWSDLElement.

Records

	Name	Description
	TWString (see page 7186)	This is record WSDLIntf.TWString.

74.2.1 WSDLIntf.TWebServiceBindingType Enumeration

TWebServiceBindingType indicates the binding type of a Web Service request or response.

Pascal

```
TWebServiceBindingType = (
  btUnknown,
  btSoap,
  btHttp,
  btMime,
  btDIME
);
```

C++

```
enum TWebServiceBindingType {
  btUnknown,
  btSoap,
  btHttp,
  btMime,
  btDIME
};
```

File

WSDLIntf

Description

The following table lists the possible values:

Value	Meaning
btUnknown	The binding type is unknown.
btSoap	The request or response is encoded as a SOAP packet.
btHttp	The request or response is an HTTP messge.
btMime	The request or response uses a MIME encoding.
btDIME	The request or response uses a DIME encoding.

74.2.2 WSDLIntf.TWSDLElement Enumeration

Pascal

```
TWSDLElement = (
  weServiceIntf,
  weServiceImpl,
  weMessage,
  wePortType,
  weBinding,
  weService,
  weTypes,
  WeImport,
  weOperation,
  wePart
);
```

C++

```
enum TWSDLElement {
  weServiceIntf,
  weServiceImpl,
  weMessage,
  wePortType,
  weBinding,
  weService,
  weTypes,
  WeImport,
  weOperation,
  wePart
};
```

File

WSDLIntf

Description

This is record WSDLIntf.TWSDLElement.

74.2.3 WSDLIntf.TWString Record

Pascal

```
TWString = record
  WString: DOMString;
end;
```

C++

```
struct TWString {
  DOMString WString;
};
```

File

WSDLIntf

Description

This is record WSDLIntf.TWString.

74.3 Types

The following table lists types in this documentation.

Types

Name	Description
TWSDLElements (see page 7187)	This is type WSDLIntf.TWSDLElements.

74.3.1 WSDLIntf.TWSDLElements Type

Pascal

```
TWSDLElements = set of TWSDLElement;
```

C++

```
TWSDLElement TWSDLElements;
```

File

WSDLIntf

Description

This is type WSDLIntf.TWSDLElements.

74.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
Directives (see page 7189)	This is constant WSDLIntf.Directives.
DynArrayTypes (see page 7189)	This is constant WSDLIntf.DynArrayTypes.
Httpns (see page 7190)	This is constant WSDLIntf.Httpns.
Operators (see page 7190)	This is constant WSDLIntf.Operators.
OtherReservedWords (see page 7190)	This is constant WSDLIntf.OtherReservedWords.
ReservedWords (see page 7191)	This is constant WSDLIntf.ReservedWords.
SAddress (see page 7191)	This is constant WSDLIntf.SAddress.
SAnySimpleType (see page 7192)	This is constant WSDLIntf.SAnySimpleType.
SAnyType (see page 7192)	This is constant WSDLIntf.SAnyType.
SArray (see page 7192)	This is constant WSDLIntf.SArray.
SArrayOf (see page 7193)	This is constant WSDLIntf.SArrayOf.
SArrayType (see page 7193)	This is constant WSDLIntf.SArrayType.
SBinding (see page 7193)	This is constant WSDLIntf.SBinding.
SBody (see page 7194)	This is constant WSDLIntf.SBody.
ScalarArrayTypes (see page 7194)	This is constant WSDLIntf.ScalarArrayTypes.
SComplexType (see page 7194)	This is constant WSDLIntf.SComplexType.
SDefinitions (see page 7195)	This is constant WSDLIntf.SDefinitions.
SDynArray (see page 7195)	This is constant WSDLIntf.SDynArray.
SElement (see page 7195)	This is constant WSDLIntf.SElement.
SEncodingStyle (see page 7196)	This is constant WSDLIntf.SEncodingStyle.
SFault (see page 7196)	This is constant WSDLIntf.SFault.
SHeader (see page 7196)	This is constant WSDLIntf.SHeader.
SImport (see page 7197)	This is constant WSDLIntf.SImport.
SInOut (see page 7197)	This is constant WSDLIntf.SInOut.
SInput (see page 7197)	This is constant WSDLIntf.SInput.
SLocation (see page 7198)	This is constant WSDLIntf.SLocation.
SMessage (see page 7198)	This is constant WSDLIntf.SMessage.
SName (see page 7198)	This is constant WSDLIntf.SName.
SNameSpace (see page 7199)	This is constant WSDLIntf.SNameSpace.
SNsPrefix (see page 7199)	This is constant WSDLIntf.SNsPrefix.
Soapns (see page 7199)	This is constant WSDLIntf.Soapns.
SOperation (see page 7200)	This is constant WSDLIntf.SOperation.
SOutput (see page 7200)	This is constant WSDLIntf.SOutput.
SPart (see page 7200)	This is constant WSDLIntf.SPart.
SParts (see page 7201)	This is constant WSDLIntf.SParts.
SPort (see page 7201)	This is constant WSDLIntf.SPort.
SPortType (see page 7201)	This is constant WSDLIntf.SPortType.
SRequest (see page 7202)	This is constant WSDLIntf.SRequest.
SResponse (see page 7202)	This is constant WSDLIntf.SResponse.
SReturn (see page 7202)	This is constant WSDLIntf.SReturn.
SSchema (see page 7203)	This is constant WSDLIntf.SSchema.
SService (see page 7203)	This is constant WSDLIntf.SService.
SSoapAction (see page 7203)	This is constant WSDLIntf.SSoapAction.
SStyle (see page 7204)	This is constant WSDLIntf.SStyle.

STns (see page 7204)	This is constant WSDLIntf.STns.
STransport (see page 7204)	This is constant WSDLIntf.STransport.
SType (see page 7205)	This is constant WSDLIntf.SType.
STypes (see page 7205)	This is constant WSDLIntf.STypes.
SUnknown (see page 7205)	This is constant WSDLIntf.SUnknown.
SUse (see page 7206)	This is constant WSDLIntf.SUse.
SVerb (see page 7206)	This is constant WSDLIntf.SVerb.
tns (see page 7206)	This is constant WSDLIntf.tns.
Wsdlns (see page 7207)	This is constant WSDLIntf.Wsdlns.
xsdns (see page 7207)	This is constant WSDLIntf.xsdns.

74.4.1 WSDLIntf.Directives Constant

Pascal

```
Directives: array[0..43] of string =
('absolute','abstract','assembler','automated','cdecl','contains', 'default','dispid',
'dynamic', 'export','external','far','forward','implements','index','message',
'name','near','nodefault','overload','override','package','pascal','private','protected','publi
c',
'published','read','readonly','register','reintroduce','requires','resident','safecall','stdcal
l',
'stored','virtual','write','writeonly','on','platform','deprecated','local','varargs');
```

C++

```
array[0..43] of string Directives =
('absolute','abstract','assembler','automated','cdecl','contains', 'default','dispid',
'dynamic', 'export','external','far','forward','implements','index','message',
'name','near','nodefault','overload','override','package','pascal','private','protected','publi
c',
'published','read','readonly','register','reintroduce','requires','resident','safecall','stdcal
l',
'stored','virtual','write','writeonly','on','platform','deprecated','local','varargs');
```

File

WSDLIntf

Description

This is constant WSDLIntf.Directives.

74.4.2 WSDLIntf.DynArrayTypes Constant

Pascal

```
DynArrayTypes: array[0..12] of string =
('TIntegerDynArray','TCardinalDynArray','TWordDynArray',
'TSmallIntDynArray','TByteDynArray','TShortIntDynArray','TInt64DynArray','TLongWordDynArray','T
SingleDynArray',
'TDoubleDynArray','TBooleanDynArray','TStringDynArray','TWideStringDynArray');
```

C++

```
array[0..12] of string DynArrayTypes =
('TIntegerDynArray','TCardinalDynArray','TWordDynArray',
'TSmallIntDynArray','TByteDynArray','TShortIntDynArray','TInt64DynArray','TLongWordDynArray','T
SingleDynArray',
```

```
'TDoubleDynArray','TBooleanDynArray','TStringDynArray','TWideStringDynArray');
```

File

WSDLIntf

Description

This is constant WSDLIntf.DynArrayTypes.

74.4.3 WSDLIntf.Httpns Constant

Pascal

```
Httpns = Wsdlns +'http/';
```

C++

```
Httpns = Wsdlns +'http/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Httpns.

74.4.4 WSDLIntf.Operators Constant

Pascal

```
Operators: array[0..12] of string = ('+','-','*','/','@','^','=','>','<','<>','<=','>=',':=');
```

C++

```
array[0..12] of string Operators = ('+','-','*','/','@','^','=','>','<','<>','<=','>=',':=');
```

File

WSDLIntf

Description

This is constant WSDLIntf.Operators.

74.4.5 WSDLIntf.OtherReservedWords Constant

Pascal

```
OtherReservedWords: array[0..0] of string = ('Application');
```

C++

```
array[0..0] of string OtherReservedWords = ('Application');
```

File

WSDLIntf

Description

This is constant WSDLIntf.OtherReservedWords.

74.4.6 WSDLIntf.ReservedWords Constant

Pascal

```
ReservedWords: array[0..66] of string = ('and', 'array', 'as', 'asm', 'begin','case',
'class','const','constructor','destructor', 'dispinterface', 'div', 'do','downto', 'else',
'end', 'except', 'exports', 'file', 'finalization', 'finally', 'for','function', 'goto',
'high', 'if', 'implementation', 'in','inherited','initialization', 'inline','interface',
'is','label','library', 'low',
'mod','nil','not','object','of','or','out','packed','procedure',
'program','property','raise','record','repeat', 'resourcestring', 'set','shl','shr', 'string',
'then', 'threadvar', 'to','try','type','unit','until','uses','var','while','with','xor');
```

C++

```
array[0..66] of string ReservedWords = ('and', 'array', 'as', 'asm', 'begin','case',
'class','const','constructor','destructor', 'dispinterface', 'div', 'do','downto', 'else',
'end', 'except', 'exports', 'file', 'finalization', 'finally', 'for','function', 'goto',
'high', 'if', 'implementation', 'in','inherited','initialization', 'inline','interface',
'is','label','library', 'low',
'mod','nil','not','object','of','or','out','packed','procedure',
'program','property','raise','record','repeat', 'resourcestring', 'set','shl','shr', 'string',
'then', 'threadvar', 'to','try','type','unit','until','uses','var','while','with','xor');
```

File

WSDLIntf

Description

This is constant WSDLIntf.ReservedWords.

74.4.7 WSDLIntf.SAddress Constant

Pascal

```
SAddress = 'address';
```

C++

```
SAddress = 'address';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAddress.

74.4.8 WSDLIntf.SAnySimpleType Constant

Pascal

```
SAnySimpleType = 'anySimpleType';
```

C++

```
SAnySimpleType = 'anySimpleType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAnySimpleType.

74.4.9 WSDLIntf.SAnyType Constant

Pascal

```
SAnyType = 'anyType';
```

C++

```
SAnyType = 'anyType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SAnyType.

74.4.10 WSDLIntf.SArray Constant

Pascal

```
SArray = 'Array';
```

C++

```
SArray = 'Array';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArray.

74.4.11 WSDLIntf.SArrayOf Constant

Pascal

```
SArrayOf = 'ArrayOf';
```

C++

```
SArrayOf = 'ArrayOf';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArrayOf.

74.4.12 WSDLIntf.SArrayType Constant

Pascal

```
SArrayType = 'arrayType';
```

C++

```
SArrayType = 'arrayType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SArrayType.

74.4.13 WSDLIntf.SBinding Constant

Pascal

```
SBinding = 'binding';
```

C++

```
SBinding = 'binding';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SBinding.

74.4.14 WSDLIntf.SBody Constant

Pascal

```
SBody = 'body';
```

C++

```
SBody = 'body';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SBody.

74.4.15 WSDLIntf.ScalarArrayTypes Constant

Pascal

```
ScalarArrayTypes: array[0..12] of string =
('Integer','Cardinal','Word','SmallInt','Byte','ShortInt','Int64',
'LongWord','Single','Double','Boolean','String','WideString');
```

C++

```
array[0..12] of string ScalarArrayTypes =
('Integer','Cardinal','Word','SmallInt','Byte','ShortInt','Int64',
'LongWord','Single','Double','Boolean','String','WideString');
```

File

WSDLIntf

Description

This is constant WSDLIntf.ScalarArrayTypes.

74.4.16 WSDLIntf.SComplexType Constant

Pascal

```
SComplexType = 'complexType';
```

C++

```
SComplexType = 'complexType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SComplexType.

74.4.17 WSDLIntf.SDefinitions Constant

Pascal

```
SDefinitions = 'definitions';
```

C++

```
SDefinitions = 'definitions';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SDefinitions.

74.4.18 WSDLIntf.SDynArray Constant

Pascal

```
SDynArray = 'array of ';
```

C++

```
SDynArray = 'array of ';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SDynArray.

74.4.19 WSDLIntf.SElement Constant

Pascal

```
SElement = 'element';
```

C++

```
SElement = 'element';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SElement.

74.4.20 WSDLIntf.SEncodingStyle Constant

Pascal

```
SEncodingStyle = 'encodingStyle';
```

C++

```
SEncodingStyle = 'encodingStyle';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SEncodingStyle.

74.4.21 WSDLIntf.SFault Constant

Pascal

```
SFault = 'fault';
```

C++

```
SFault = 'fault';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SFault.

74.4.22 WSDLIntf.SHeader Constant

Pascal

```
SHeader = 'header';
```

C++

```
SHeader = 'header';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SHeader.

74.4.23 WSDLIntf.SImport Constant

Pascal

```
SImport = 'import';
```

C++

```
SImport = 'import';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SImport.

74.4.24 WSDLIntf.SInOut Constant

Pascal

```
SInOut = 'inout';
```

C++

```
SInOut = 'inout';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SInOut.

74.4.25 WSDLIntf.SInput Constant

Pascal

```
SInput = 'input';
```

C++

```
SInput = 'input';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SInput.

74.4.26 WSDLIntf.SLocation Constant

Pascal

```
SLocation = 'location';
```

C++

```
SLocation = 'location';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SLocation.

74.4.27 WSDLIntf.SMessage Constant

Pascal

```
SMessage = 'message';
```

C++

```
SMessage = 'message';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SMessage.

74.4.28 WSDLIntf.SName Constant

Pascal

```
SName = 'name';
```

C++

```
SName = 'name';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SName.

74.4.29 WSDLIntf.SNameSpace Constant

Pascal

```
SNameSpace = 'namespace';
```

C++

```
SNameSpace = 'namespace';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SNameSpace.

74.4.30 WSDLIntf.SNsPrefix Constant

Pascal

```
SNsPrefix = 'ns';
```

C++

```
SNsPrefix = 'ns';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SNsPrefix.

74.4.31 WSDLIntf.Soapns Constant

Pascal

```
Soapns = Wsdlns +'soap/';
```

C++

```
Soapns = Wsdlns +'soap/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Soapns.

74.4.32 WSDLIntf.SOperation Constant

Pascal

```
SOperation = 'operation';
```

C++

```
SOperation = 'operation';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SOperation.

74.4.33 WSDLIntf.SOutput Constant

Pascal

```
SOutput = 'output';
```

C++

```
SOutput = 'output';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SOutput.

74.4.34 WSDLIntf.SPart Constant

Pascal

```
SPart = 'part';
```

C++

```
SPart = 'part';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPart.

74.4.35 WSDLIntf.SParts Constant

Pascal

```
SParts = 'parts';
```

C++

```
SParts = 'parts';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SParts.

74.4.36 WSDLIntf.SPort Constant

Pascal

```
SPort = 'port';
```

C++

```
SPort = 'port';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPort.

74.4.37 WSDLIntf.SPortType Constant

Pascal

```
SPortType = 'portType';
```

C++

```
SPortType = 'portType';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SPortType.

74.4.38 WSDLIntf.SRequest Constant

Pascal

```
SRequest = 'Request';
```

C++

```
SRequest = 'Request';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SRequest.

74.4.39 WSDLIntf.SResponse Constant

Pascal

```
SResponse = 'Response';
```

C++

```
SResponse = 'Response';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SResponse.

74.4.40 WSDLIntf.SReturn Constant

Pascal

```
SReturn = 'return';
```

C++

```
SReturn = 'return';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SReturn.

74.4.41 WSDLIntf.SSchema Constant

Pascal

```
SSchema = 'schema';
```

C++

```
SSchema = 'schema';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SSchema.

74.4.42 WSDLIntf.SService Constant

Pascal

```
SService = 'service';
```

C++

```
SService = 'service';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SService.

74.4.43 WSDLIntf.SSoapAction Constant

Pascal

```
SSoapAction = 'soapAction';
```

C++

```
SSoapAction = 'soapAction';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SSoapAction.

74.4.44 WSDLIntf.SStyle Constant

Pascal

```
SStyle = 'style';
```

C++

```
SStyle = 'style';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SStyle.

74.4.45 WSDLIntf.STns Constant

Pascal

```
STns = 'targetNamespace';
```

C++

```
STns = 'targetNamespace';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STns.

74.4.46 WSDLIntf.STransport Constant

Pascal

```
STransport = 'transport';
```

C++

```
STransport = 'transport';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STransport.

74.4.47 WSDLIntf.SType Constant

Pascal

```
SType = 'type';
```

C++

```
SType = 'type';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SType.

74.4.48 WSDLIntf.STypes Constant

Pascal

```
STypes = 'types';
```

C++

```
STypes = 'types';
```

File

WSDLIntf

Description

This is constant WSDLIntf.STypes.

74.4.49 WSDLIntf.SUnknown Constant

Pascal

```
SUnknown = 'Unknown';
```

C++

```
SUnknown = 'Unknown';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SUnknown.

74.4.50 WSDLIntf.SUse Constant

Pascal

```
SUse = 'use';
```

C++

```
SUse = 'use';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SUse.

74.4.51 WSDLIntf.SVerb Constant

Pascal

```
SVerb = 'verb';
```

C++

```
SVerb = 'verb';
```

File

WSDLIntf

Description

This is constant WSDLIntf.SVerb.

74.4.52 WSDLIntf.tns Constant

Pascal

```
tns = 'http:/'+'/tempuri.org/';
```

C++

```
tns = 'http:/'+'/tempuri.org/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.tns.

74.4.53 WSDLIntf.Wsdlns Constant

Pascal

```
Wsdlns = 'http:/'+'/schemas.xmlsoap.org/wsdl/';
```

C++

```
Wsdlns = 'http:/'+'/schemas.xmlsoap.org/wsdl/';
```

File

WSDLIntf

Description

This is constant WSDLIntf.Wsdlns.

74.4.54 WSDLIntf.xsdns Constant

Pascal

```
xsdns = 'http:/'+'/www.w3.org/1999/XMLSchema';
```

C++

```
xsdns = 'http:/'+'/www.w3.org/1999/XMLSchema';
```

File

WSDLIntf

Description

This is constant WSDLIntf.xsdns.

75 WSDLItems Namespace

This is namespace WSDLItems.

Topics

Name	Description
Classes (see page 7211)	The following table lists classes in this documentation.
Interfaces (see page 7247)	The following table lists interfaces in this documentation.
Functions (see page 7259)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7261)	The following table lists structs, records, enums in this documentation.
Types (see page 7262)	The following table lists types in this documentation.

Classes

	Name	Description
	TWSDLItems (see page 7211)	TWSDLItems represents the definitions in a WSDL document.

Functions

	Name	Description
	HasDefinition (see page 7259)	This is function WSDLItems.HasDefinition.
	HasDefinition (see page 7259)	This is function WSDLItems.HasDefinition.
	NewQualifiedName (see page 7259)	This is function WSDLItems.NewQualifiedName.

Interfaces

	Name	Description
	IHeaderInfo (see page 7247)	This is class WSDLItems.IHeaderInfo.
	IQualifiedName (see page 7252)	IQualifiedName is the interface for obtaining an information about a name in an XML document.
	IWSDLItems (see page 7256)	IWSDLItems provides access to a WSDL document.

Structs, Records, Enums

	Name	Description
	IterateImportOptions (see page 7261)	This is record WSDLItems.IterateImportOptions.

Types

Name	Description
IHeaderInfoArray (see page 7262)	This is type WSDLItems.IHeaderInfoArray.
IQualifiedNameArray (see page 7262)	IQualifiedNameArray is a dynamic array of interfaces for obtaining names of XML items.

TWSDLIterateProc (see page 7263)	TWSDLIterateProc is the type for callbacks that execute when importing a WSDL document.

75.1 Classes

The following table lists classes in this documentation.

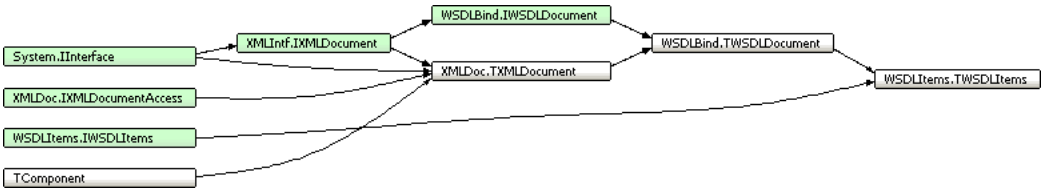
Classes

	Name	Description
	TWSDLItems (see page 7211)	TWSDLItems represents the definitions in a WSDL document.

75.1.1 TWSDLItems Class

TWSDLItems represents the definitions in a WSDL document.

Class Hierarchy



Pascal

```
TWSDLItems = class(TWSDLDocument, IWSDLItems);
```

C++

```
class TWSDLItems : public TWSDLDocument, public IWSDLItems;
```

File

WSDLItems

Description

Use TWSDLItems to represent a WSDL (Web Services Definition Language) document. A WSDL document is a special form of XML document that describes the contents of a Web Service application and how you call the services it implements. TWSDLItems augments its immediate ancestor (TWSDLDocument (see page 7063)) by introducing methods that provide easier access to the definitions in the WSDL document.

TWSDLItems can read an existing WSDL document from a file or server, it can be associated with an in-memory string that is the contents of a WSDL document, or it can create a new, empty WSDL document. TWSDLItems uses an external DOM (Document Object Model) parser to analyze the WSDL document. This parser is usually assigned by specifying the DOMVendor property. When you set the Active property to true, TWSDLItems uses the DOM parser to parse the WSDL document so that you can examine or modify it.

When TWSDLItems is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TWSDLItems instance is automatically freed. When TWSDLItems is created with an Owner, however, it behaves like any other component, and is freed by its Owner.

See Also

TWSDLView (see page 7271)

75.1.1.1 TWSDLItems Members

The following tables list the members exposed by TWSDLItems.

Protected Methods

	Name	Description
	_AddRef (see page 7354)	Called when an application uses a component interface.
	_Release (see page 7354)	Called when an application releases a component interface.
	AssignParseOptions (see page 7356)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7356)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7356)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7356)	This is CheckDOM, a member of class TXMLDocument.
	DefineProperties (see page 7358)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7359)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7359)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7359)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7360)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7360)	This is DoNodeChange, a member of class TXMLDocument.
	GetActive (see page 7360)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7361)	Returns the value of the AsyncLoadState property.
	GetChildNodeClass (see page 7361)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7361)	Returns the value of the ChildNodes property.
	GetDefinition (see page 7069)	Returns the value of the Definition property.
	GetDocumentElement (see page 7362)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7362)	Returns the value of the Node property.
	GetDocumentObject (see page 7363)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7363)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7363)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7363)	Returns the value of the Encoding property.
	GetFileName (see page 7364)	Returns the value of the FileName property.
	GetModified (see page 7364)	Returns the value of the Modified property.
	GetName (see page 7225)	Returns the name of the definitions node in the WSDL document.
	GetNodeIndentStr (see page 7364)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7365)	Returns the value of the Options property.
	GetParseOptions (see page 7365)	Returns the value of the ParseOptions property.
	GetPrologNode (see page 7365)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7366)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7366)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7366)	Returns the value of the StandAlone property.
	GetTargetNamespace (see page 7242)	Returns the value of the TargetNamespace (see page 7246) property.
	GetTargetNamespacePrefix (see page 7243)	Returns the prefix string that represents the target namespace of the WSDL document.
	GetVersion (see page 7366)	Returns the value of the Version property.
	GetXML (see page 7367)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7367)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7069)	This is LoadData, a member of class TWSDLDocument.
	LoadData (see page 7368)	This is LoadData, a member of class TXMLDocument.
	Loaded (see page 7368)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7371)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7372)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7374)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7375)	Saves the XML (see page 7392) document to an internal string list.

	SetActive (see page 7375)	Sets the value of the Active property.
	SetDefinition (see page 7070)	Sets the value of the Definition property.
	SetDocumentElement (see page 7375)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7376)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7376)	Sets the value of the Encoding property.
	SetFileName (see page 7376)	Sets the value of the FileName property.
	SetModified (see page 7377)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7377)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7377)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7378)	Sets the value of the Options property.
	SetParseOptions (see page 7378)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7378)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7378)	Sets the value of the StandAlone property.
	SetVersion (see page 7379)	Sets the value of the Version property.
	SetXML (see page 7379)	Sets the value of the XML property.
	SetXMLStrings (see page 7379)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7380)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7380)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
	DocBindingInfo (see page 7383)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7383)	This is DocSource, a member of class TXMLDocument.
	DOMParseOptions (see page 7385)	This is DOMParseOptions, a member of class TXMLDocument.
	DOMPersist (see page 7386)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7391)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7354)	This is the overview for the AddChild method overload.
	AddChild (see page 7643)	This is the overview for the AddChild method overload.
	AfterConstruction (see page 7069)	Initializes the TWSDLDocument (see page 7063) object after the last constructor has executed.
	AfterConstruction (see page 7355)	Responds after the last constructor has executed.
	CompareName (see page 7219)	Indicates whether two node names refer to the same element.
	Create (see page 7217)	This is the overview for the Create constructor overload.
	Create (see page 7351)	This is the overview for the Create constructor overload.
	CreateElement (see page 7357)	Creates a new element node that is associated with this document.
	CreateElement (see page 7644)	Creates a new element node that is associated with this document.
	CreateNode (see page 7357)	Creates a new node that is associated with this document.
	CreateNode (see page 7645)	Creates a new node that is associated with this document.
	Destroy (see page 7352)	Disposes of a TXMLDocument (see page 7347) object.
	GeneratePrefix (see page 7360)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 7646)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7646)	Returns the value of the Active (see page 7661) property.
	GetAsyncLoadState (see page 7647)	Returns the value of the AsyncLoadState (see page 7662) property.
	GetBindingForServicePort (see page 7220)	Returns the name of the binding associated with a specified service and port.
	GetBindingOfType (see page 7221)	Returns the names of all bindings associated with a specified port type.
	GetBindingOperationNode (see page 7221)	Returns the interface for a specified operation on a specified binding.
	GetBindings (see page 7222)	Fills a WideString (see page 3467) list with the names of all bindings defined in the WSDL document.

	GetBindingType (see page 7222)	Returns the port type to which a specified binding applies.
	GetChildNodes (see page 7647)	Returns the value of the ChildNodes (see page 7663) property.
	GetDefinition (see page 7166)	Returns the value of the Definition (see page 7167) property.
	GetDocBinding (see page 7362)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7647)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7648)	Returns the value of the DocumentElement (see page 7663) property.
	GetDocumentNode (see page 7648)	Returns the value of the Node (see page 7665) property.
	GetDocumentObject (see page 7479)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7648)	Returns the value of the DOMDocument (see page 7663) property.
	GetDOMPersist (see page 7479)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7649)	Returns the value of the Encoding (see page 7664) property.
	GetFileName (see page 7649)	Returns the value of the FileName (see page 7664) property.
	GetHttpBindingAttribute (see page 7223)	This is the overview for the GetHttpBindingAttribute method overload.
	GetImports (see page 7223)	Fills a WideString (see page 3467) list with the namespaces of all documents imported by this one.
	GetLocationForImport (see page 7224)	Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.
	GetMessageNode (see page 7224)	Returns the interface for the node that describes a message.
	GetMessages (see page 7224)	Fills a WideString (see page 3467) list with the names of all messages defined in the WSDL document.
	GetModified (see page 7649)	Returns the value of the Modified (see page 7665) property.
	GetNodeIndentStr (see page 7650)	Returns the value of the NodeIndentStr (see page 7666) property.
	GetOperationNode (see page 7225)	Returns the interface for the node that describes an operation.
	GetOperations (see page 7226)	Fills a WideString (see page 3467) list with the names of all operations defined for a specified port type.
	GetOperationsForBinding (see page 7226)	Fills a list with the names of all operations defined on the port type of a specified binding.
	GetOptions (see page 7650)	Returns the value of the Options (see page 7666) property.
	GetParseOptions (see page 7650)	Returns the value of the ParseOptions (see page 7667) property.
	GetPartNode (see page 7227)	Returns the interface for the node that describes a part of a message.
	GetParts (see page 7227)	Fills a list with the names of all parts defined for a specified message.
	GetPartsForOperation (see page 7228)	Fills a list with the names of all parts defined for input messages of a specified operation.
	GetPortsForService (see page 7229)	Fills a list with the names of all ports defined within a specified service.
	GetPortTypeNode (see page 7229)	Returns the interface for the node that describes a port type defined in the WSDL document.
	GetPortTypes (see page 7229)	Fills a list with the names of all port types defined in the WSDL document.
	GetSchemaNode (see page 7230)	Returns the interface for the schema node that defines a type.
	GetSchemaRef (see page 7650)	Returns the value of the SchemaRef (see page 7667) property.
	GetSchemas (see page 7230)	Fills a list with the namespaces for all schema definitions of types in the WSDL document.
	GetServiceAndPortOfBinding (see page 7231)	Retrieves the names of the service and port associated with a specified binding.
	GetServiceNode (see page 7231)	Returns the interface for the node that describes a service defined in the WSDL document.
	GetServices (see page 7232)	Fills a list with the names of all services defined in the WSDL document.
	GetSoapAction (see page 7232)	Returns the value of the soapAction attribute of a specified operation on a specified binding.
	GetSoapActionList (see page 7233)	Fills a list with the soapAction attributes for all operations on a specified SOAP binding.
	GetSoapAddressForServicePort (see page 7234)	Returns the address of a specified port from a specified service.
	GetSoapBindingAttribute (see page 7234)	Returns the value of a specified attribute of a specified SOAP binding.
	GetSoapBindingInputNode (see page 7235)	Returns the interface for the input node on a specified binding and operation.
	GetSoapBindingIONode (see page 7235)	Returns the interface for a node on a specified binding and operation.
	GetSoapBindingOutputNode (see page 7236)	Returns the interface for the output node on a specified binding and operation.
	GetSoapBodyAttribute (see page 7236)	Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.

	GetSoapBodyInputAttribute (see page 7237)	Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.
	GetSoapBodyNamespace (see page 7238)	Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.
	GetSoapBodyOutputAttribute (see page 7238)	Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.
	GetSoapHeaders (see page 7239)	Returns an interface for accessing a specified set of header nodes.
	GetSoapInputHeaders (see page 7240)	Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.
	GetSoapOperationAttribute (see page 7240)	Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.
	GetSoapOperationStyle (see page 7241)	Returns the value of the style attribute of a specified operation on a specified binding.
	GetSoapOutputHeaders (see page 7241)	Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.
	GetStandAlone (see page 7651)	Returns the value of the StandAlone (see page 7668) property.
	GetVersion (see page 7651)	Returns the value of the Version (see page 7668) property.
	GetWSDLDoc (see page 7243)	Returns the interface for this TWSDLItems (see page 7211) instance.
	GetWSDLItems (see page 7243)	Returns a reference to this TWSDLItems (see page 7211) instance.
	GetXML (see page 7651)	Returns the value of the XML (see page 7668) property.
	HasTypesNode (see page 7244)	Indicates whether the WSDL document includes any type definitions.
	IsEmptyDoc (see page 7367)	Indicates whether the TXMLDocument (see page 7347) instance represents an empty document.
	IsEmptyDoc (see page 7652)	Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.
	IsPortTypeHTTPBound (see page 7244)	Indicates whether a specified port type has an HTTP binding.
	IterateImports (see page 7244)	Executes a callback for every WSDL document that this one imports.
	Load (see page 7245)	Loads a WSDL document and activates it.
	LoadFromFile (see page 7368)	Loads an XML (see page 7392) document and activates it.
	LoadFromFile (see page 7652)	Loads an XML (see page 7668) document from disk.
	LoadFromStream (see page 7369)	Loads an XML (see page 7392) document from a stream and activates it.
	LoadFromStream (see page 7653)	Loads an XML (see page 7668) document from a stream and activates it.
	LoadFromXML (see page 7370)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
	NewInstance (see page 7370)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7371)	Updates the parsed XML (see page 7392) document to reflect external changes.
	Refresh (see page 7654)	Updates the parsed XML (see page 7668) document to reflect external changes.
	RegisterDocBinding (see page 7372)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7654)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7372)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7655)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7373)	Saves the XML (see page 7392) document to disk.
	SaveToFile (see page 7655)	Saves the XML (see page 7668) document to disk.
	SaveToStream (see page 7373)	Saves the XML (see page 7392) document to a stream.
	SaveToStream (see page 7656)	Saves the XML (see page 7668) document to a stream.
	SaveToXML (see page 7374)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7657)	Sets the value of the Active (see page 7661) property.
	SetDefinition (see page 7166)	Sets the value of the Definition (see page 7167) property.
	SetDocumentElement (see page 7657)	Sets the value of the DocumentElement (see page 7663) property.
	SetDOMDocument (see page 7658)	Sets the value of the DOMDocument (see page 7663) property.
	SetEncoding (see page 7658)	Sets the value of the Encoding (see page 7664) property.
	SetFileName (see page 7658)	Sets the value of the FileName (see page 7664) property.

	SetNodeIndentStr (see page 7659)	Sets the value of the NodeIndentStr (see page 7666) property.
	SetOnAsyncLoad (see page 7659)	Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.
	SetOptions (see page 7659)	Sets the value of the Options (see page 7666) property.
	SetParseOptions (see page 7660)	Sets the value of the ParseOptions (see page 7667) property.
	SetStandAlone (see page 7660)	Sets the value of the StandAlone (see page 7668) property.
	SetVersion (see page 7660)	Sets the value of the Version (see page 7668) property.
	SetXML (see page 7661)	Sets the value of the XML (see page 7668) property.

Public Properties

	Name	Description
	Active (see page 7661)	Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7382)	Indicates the state of the DOM parser when it parses the XML (see page 7392) document asynchronously.
R	AsyncLoadState (see page 7662)	Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.
R	ChildNodes (see page 7383)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7663)	Lists all of the child nodes of the document.
	Definition (see page 7167)	Provides access to the definitions contained in the WSDL document.
	Definition (see page 7070)	Provides access to the definitions contained in the WSDL document.
	DocumentElement (see page 7383)	Provides access to the root node of the XML (see page 7392) document.
	DocumentElement (see page 7663)	Provides access to the root node of the XML (see page 7668) document.
R	DocumentObject (see page 7479)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7384)	Provides access to the DOM interface for the parsed XML (see page 7392) document.
	DOMDocument (see page 7663)	Provides access to the DOM interface for the parsed XML (see page 7668) document.
	DOMImplementation (see page 7385)	Provides access to the DOM interface for document-independent functions.
R	DOMPersist (see page 7480)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7386)	Indicates the character set that the XML (see page 7392) document uses to encode characters.
	Encoding (see page 7664)	Indicates the character set that the XML (see page 7668) document uses to encode characters.
	FileName (see page 7664)	Indicates the XML (see page 7668) document file, if any, that this interface represents.
R	Modified (see page 7387)	Indicates whether the XML (see page 7392) document has been modified.
R	Modified (see page 7665)	Indicates whether the XML (see page 7668) document has been modified.
R	Node (see page 7388)	Provides access to the document node for the XML (see page 7392) document.
R	Node (see page 7665)	Provides access to the document node for the XML (see page 7668) document.
	NodeIndentStr (see page 7666)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.
	NSPrefixBase (see page 7389)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7666)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7667)	Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.
R	SchemaRef (see page 7391)	Returns the name of the schema associated with this XML (see page 7392) document.
R	SchemaRef (see page 7667)	Returns the name of the schema associated with this XML (see page 7668) document.
	StandAlone (see page 7391)	Indicates whether the XML (see page 7392) document includes any external markup declarations.
	StandAlone (see page 7668)	Indicates whether the XML (see page 7668) document includes any external markup declarations.
R	StreamLoader (see page 7245)	Provides access to an interface for loading the WSDL document.
R	TargetNamespace (see page 7246)	Provides access to an interface for loading the WSDL document.
	Version (see page 7391)	Indicates the XML (see page 7392) version that the document uses.
	Version (see page 7668)	Indicates the XML (see page 7668) version that the document uses.

	XML (see page 7668)	Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.

Published Events

	Name	Description
	AfterClose (see page 7393)	Occurs after the XML (see page 7392) document is closed.
	AfterNodeChange (see page 7393)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7394)	Occurs immediately after the XML (see page 7392) document is opened.
	BeforeClose (see page 7394)	Occurs immediately before the XML (see page 7392) document is closed.
	BeforeNodeChange (see page 7394)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7395)	Occurs immediately before the XML (see page 7392) document is opened.
	OnAsyncLoad (see page 7395)	Occurs at various stages during the asynchronous parsing of an XML (see page 7392) document.

Published Properties

	Name	Description
	Active (see page 7381)	Specifies whether the XML (see page 7392) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7386)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7392) document.
	FileName (see page 7387)	Specifies the file name for the XML (see page 7392) document associated with this TXMLDocument (see page 7347) object.
	NodeIndentStr (see page 7388)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7392) text.
	Options (see page 7389)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7390)	Specifies implementation-specific options that are followed when parsing the XML (see page 7392) document.
	XML (see page 7392)	Specifies the content of the XML document associated with this TXMLDocument (see page 7347) object.

75.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TWSDLItems.Create (TComponent) (see page 7217)	Creates and initializes a TWSDLItems (see page 7211) object.
	TWSDLItems.Create (TWSDLItems, IStreamLoader) (see page 7217)	Creates and initializes a TWSDLItems (see page 7211) object.

75.1.1.2.1 TWSDLItems.Create Constructor (TComponent)

Creates and initializes a TWSDLItems (see page 7211) object.

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
constructor Create(const WSDLItems: TWSDLItems; const StreamLoader: IStreamLoader); virtual;
overload;
```

C++

```
virtual __fastcall TWSDLItems(TComponent * AOwner);
virtual __fastcall TWSDLItems(const TWSDLItems WSDLItems, const IStreamLoader StreamLoader);
```

Description

Call Create to instantiate a TWSDLItems (see page 7211) component at runtime. Typically, TWSDLItems (see page 7211) instances are created by other components, such as THTTPRIO, that use them.

AOwner is another component, typically a form or data module, that is responsible for freeing the TWSDLItems (see page 7211) instance. It becomes the value of the Owner property.

WSDLItems (see page 7209) is another TWSDLItems (see page 7211) object that the new instance can use to extract a stream loader if none is specified using the StreamLoader (see page 7245) property,

StreamLoader (see page 7245) is an interface to use for loading the WSDL document from a server. It includes properties for specifying the user name, password, and proxy server needed to connect to the server that hosts the WSDL document.

If you do not specify a stream loader, the TWSDLItems (see page 7211) constructor creates one as follows: Using the first syntax, it creates a default stream loader object. Using the second syntax, it uses the stream loader from the TWSDLItems (see page 7211) instance that is the first parameter. Thus, when using the second syntax, you must supply either the WSDLItems (see page 7209) or StreamLoader (see page 7245) argument.

See Also

THTTPRIO (see page 1248)

Owner

75.1.1.3 TWSDLItems Methods

The methods of the TWSDLItems class are listed here.

Protected Methods

	Name	Description
	GetName (see page 7225)	Returns the name of the definitions node in the WSDL document.
	GetTargetNamespace (see page 7242)	Returns the value of the TargetNamespace (see page 7246) property.
	GetTargetNamespacePrefix (see page 7243)	Returns the prefix string that represents the target namespace of the WSDL document.

Public Methods

	Name	Description
	CompareName (see page 7219)	Indicates whether two node names refer to the same element.
	GetBindingForServicePort (see page 7220)	Returns the name of the binding associated with a specified service and port.
	GetBindingOfType (see page 7221)	Returns the names of all bindings associated with a specified port type.
	GetBindingOperationNode (see page 7221)	Returns the interface for a specified operation on a specified binding.
	GetBindings (see page 7222)	Fills a WideString (see page 3467) list with the names of all bindings defined in the WSDL document.
	GetBindingType (see page 7222)	Returns the port type to which a specified binding applies.
	GetHttpBindingAttribute (see page 7223)	This is the overview for the GetHttpBindingAttribute method overload.
	GetImports (see page 7223)	Fills a WideString (see page 3467) list with the namespaces of all documents imported by this one.
	GetLocationForImport (see page 7224)	Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.
	GetMessageNode (see page 7224)	Returns the interface for the node that describes a message.
	GetMessages (see page 7224)	Fills a WideString (see page 3467) list with the names of all messages defined in the WSDL document.
	GetOperationNode (see page 7225)	Returns the interface for the node that describes an operation.
	GetOperations (see page 7226)	Fills a WideString (see page 3467) list with the names of all operations defined for a specified port type.
	GetOperationsForBinding (see page 7226)	Fills a list with the names of all operations defined on the port type of a specified binding.

	GetPartNode (see page 7227)	Returns the interface for the node that describes a part of a message.
	GetParts (see page 7227)	Fills a list with the names of all parts defined for a specified message.
	GetPartsForOperation (see page 7228)	Fills a list with the names of all parts defined for input messages of a specified operation.
	GetPortsForService (see page 7229)	Fills a list with the names of all ports defined within a specified service.
	GetPortTypeNode (see page 7229)	Returns the interface for the node that describes a port type defined in the WSDL document.
	GetPortTypes (see page 7229)	Fills a list with the names of all port types defined in the WSDL document.
	GetSchemaNode (see page 7230)	Returns the interface for the schema node that defines a type.
	GetSchemas (see page 7230)	Fills a list with the namespaces for all schema definitions of types in the WSDL document.
	GetServiceAndPortOfBinding (see page 7231)	Retrieves the names of the service and port associated with a specified binding.
	GetServiceNode (see page 7231)	Returns the interface for the node that describes a service defined in the WSDL document.
	GetServices (see page 7232)	Fills a list with the names of all services defined in the WSDL document.
	GetSoapAction (see page 7232)	Returns the value of the soapAction attribute of a specified operation on a specified binding.
	GetSoapActionList (see page 7233)	Fills a list with the soapAction attributes for all operations on a specified SOAP binding.
	GetSoapAddressForServicePort (see page 7234)	Returns the address of a specified port from a specified service.
	GetSoapBindingAttribute (see page 7234)	Returns the value of a specified attribute of a specified SOAP binding.
	GetSoapBindingInputNode (see page 7235)	Returns the interface for the input node on a specified binding and operation.
	GetSoapBindingIONode (see page 7235)	Returns the interface for a node on a specified binding and operation.
	GetSoapBindingOutputNode (see page 7236)	Returns the interface for the output node on a specified binding and operation.
	GetSoapBodyAttribute (see page 7236)	Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.
	GetSoapBodyInputAttribute (see page 7237)	Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.
	GetSoapBodyNamespace (see page 7238)	Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.
	GetSoapBodyOutputAttribute (see page 7238)	Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.
	GetSoapHeaders (see page 7239)	Returns an interface for accessing a specified set of header nodes.
	GetSoapInputHeaders (see page 7240)	Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.
	GetSoapOperationAttribute (see page 7240)	Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.
	GetSoapOperationStyle (see page 7241)	Returns the value of the style attribute of a specified operation on a specified binding.
	GetSoapOutputHeaders (see page 7241)	Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.
	GetWSDLDoc (see page 7243)	Returns the interface for this TWSDLItems (see page 7211) instance.
	GetWSDLItems (see page 7243)	Returns a reference to this TWSDLItems (see page 7211) instance.
	HasTypesNode (see page 7244)	Indicates whether the WSDL document includes any type definitions.
	IsPortTypeHTTPBound (see page 7244)	Indicates whether a specified port type has an HTTP binding.
	IterateImports (see page 7244)	Executes a callback for every WSDL document that this one imports.
	Load (see page 7245)	Loads a WSDL document and activates it.

75.1.1.3.1 TWSDLItems.CompareName Method

Indicates whether two node names refer to the same element.

Pascal

```
function CompareName(const NodeName: DOMString; const OtherName: DOMString; const TNS:
DOMString = ''): boolean;
```

C++

```
__fastcall boolean CompareName(const DOMString NodeName, const DOMString OtherName, const
DOMString TNS = '');
```

Description

Call CompareName to compare the names of two nodes to determine whether they match. CompareName compares the node names, taking into account any namespace prefix added to the node names. CompareName uses the namespace prefix to look up the corresponding namespace URI. If either node name does not have a namespace prefix, CompareName uses the TargetNamespace (see page 7246) property for the first node name, and the supplied namespace for the second. CompareName can find a match between two node names that have different namespace prefixes, as long as both prefixes map to the same namespace URI.

NodeName and OtherName are the two node names to compare.

TNS is the target namespace for OtherName. If this is a nonempty string, it is used to replace the namespace of OtherName. If it is an empty string, OtherName uses the namespace associated with its namespace prefix, or the TargetNamespace (see page 7246) property if it has no namespace prefix.

CompareName returns true if the two names match, false if they differ (either in the local name, or in the namespace to which the names belong).

See Also

NodeName (see page 7693)

TargetNamespace (see page 7246)

75.1.1.3.2 TWSDLItems.GetBindingForServicePort Method

Returns the name of the binding associated with a specified service and port.

Pascal

```
function GetBindingForServicePort(const ServiceName: DOMString; const PortName: DOMString):
IQualifiedName;
```

C++

```
__fastcall IQualifiedName GetBindingForServicePort(const DOMString ServiceName, const
DOMString PortName);
```

Description

Call GetBindingForServicePort to retrieve the name of a binding that is associated with the specified service and port. The binding is defined in the WSDL document or another document that it imports. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

ServiceName is the name of the service to which the desired binding belongs.

PortName is the name of a port within the specified service that uses the binding.

See Also

GetBindings (see page 7222)

GetBindingType (see page 7222)

GetPortsForService (see page 7229)

GetServices (see page 7232)

GetBindingOfType (see page 7221)

GetServiceAndPortOfBinding (see page 7231)

75.1.1.3.3 TWSDLItems.GetBindingOfType Method

Returns the names of all bindings associated with a specified port type.

Pascal

```
function GetBindingOfType(const BindingTypeName: DOMString; const TNS: DOMString = ''):
IQualifiedNameArray; overload;
```

C++

```
__fastcall IQualifiedNameArray GetBindingOfType(const DOMString BindingTypeName, const
DOMString TNS = '');
```

Description

Call GetBindingOfType to retrieve the names of the bindings that are associated with the specified port type. Each bindings is defined in the WSDL document or another document that it imports. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

BindingTypeName is the type of the binding. This is the name of a port type for which the binding describes an encoding format and communications protocol.

TNS is the namespace in which BindingTypeName is defined. If this parameter is omitted, GetBindingOfType assumes the value of the TargetNamespace (see page 7246) property.

GetBindingOfType returns an array of qualified names for each binding defined for the specified type.

See Also

GetBindings (see page 7222)

GetPortTypes (see page 7229)

GetBindingType (see page 7222)

TargetNamespace (see page 7246)

75.1.1.3.4 TWSDLItems.GetBindingOperationNode Method

Returns the interface for a specified operation on a specified binding.

Pascal

```
function GetBindingOperationNode(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): IBindingOperation;
```

C++

```
__fastcall IBindingOperation GetBindingOperationNode(const DOMString BindingName, const
DOMString Operation, int OverloadIndex);
```

Description

Call GetBindingOperationNode to retrieve the operation interface for a SOAP binding. GetBindingOperationNode locates a binding with the specified name, and searches its child nodes for an operation node with the specified name. If there is more than one operation with the specified name (that is, if the method is overloaded), GetBindingOperationNode locates the desired overload based on its position.

BindingName is the name of the binding. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that binding node. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node to choose if there are overloaded methods. 0 indicates the first operation node with the specified name, 1 indicates the second operation node with that name, and so on.

GetBindingOperationNode returns an IBindingOperation (see page 7083) interface, which can be used to access the name, inputs, outputs, and faults associated with the operation. If GetBindingOperationNode is unable to locate an operation node for the specified binding and operation names, it returns nil (Delphi) or NULL (C++).

See Also

GetOperationNode (see page 7225)

75.1.1.3.5 TWSDLItems.GetBindings Method

Fills a WideString (see page 3467) list with the names of all bindings defined in the WSDL document.

Pascal

```
procedure GetBindings(BindingNames: TDOMStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetBindings(TDOMStrings BindingNames, Boolean QualifiedNames = False);
```

Description

Call GetBindings to retrieve the names of all bindings defined in the WSDL document. A binding identifies an encoding format (e.g. soap:rpc) and communications protocol (eg. HTML) that is available for a specific port type (interface).

BindingNames is an existing TWideStrings object to which the names of any defined bindings are added.

QualifiedNames indicates whether the binding names should be qualified using the target namespace prefix.

Note: GetBindings does not clear the WideString (see page 3467) list before adding the names of all defined bindings.

See Also

GetBindingForServicePort (see page 7220)

GetBindingType (see page 7222)

GetPortTypes (see page 7229)

75.1.1.3.6 TWSDLItems.GetBindingType Method

Returns the port type to which a specified binding applies.

Pascal

```
function GetBindingType(const BindingName: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetBindingType(const DOMString BindingName);
```

Description

Call GetBindingType to retrieve the names of the port type to which a specified binding applies.

BindingName is the name of a binding defined in the WSDL document.

See Also

GetBindingForServicePort (see page 7220)

GetBindings (see page 7222)

GetPortTypes (see page 7229)

GetBindingOfType (see page 7221)

75.1.1.3.7 GetHttpBindingAttribute Method

This is the overview for the GetHttpBindingAttribute method overload.

Overload List

	Name	Description
	TWSDLItems.GetHttpBindingAttribute (DOMString, DOMString) (see page 7223)	Returns the value of a specified attribute of a binding's <http:binding> node.
	TWSDLItems.GetHttpBindingAttribute (IQualifiedName, DOMString) (see page 7223)	Returns the value of a specified attribute of a binding's <http:binding> node.

75.1.1.3.7.1 TWSDLItems.GetHttpBindingAttribute Method (DOMString, DOMString)

Returns the value of a specified attribute of a binding's <http:binding> node.

Pascal

```
function GetHttpBindingAttribute(const BindingName: DOMString; Attribute: DOMString):
DOMString; overload;
function GetHttpBindingAttribute(const QualifiedName: IQualifiedName; Attribute: DOMString):
DOMString; overload;
```

C++

```
__fastcall DOMString GetHttpBindingAttribute(const DOMString BindingName, DOMString Attribute);
__fastcall DOMString GetHttpBindingAttribute(const IQualifiedName QualifiedName, DOMString
Attribute);
```

Description

Call GetHttpBindingAttribute to retrieve value of an attribute on the http binding tag of a specified binding. The http binding tag can come from the WSDL document or another document that it imports.

BindingName or QualifiedName is the name of the binding that includes the <http:binding> tag.

Attribute is the name of the desired attribute on the <http:binding> tag.

See Also

GetSoapBindingAttribute (see page 7234)

GetSoapOperationAttribute (see page 7240)

GetSoapBodyAttribute (see page 7236)

75.1.1.3.8 TWSDLItems.GetImports Method

Fills a WideString (see page 3467) list with the namespaces of all documents imported by this one.

Pascal

```
procedure GetImports(ImportNames: TDOMStrings);
```

C++

```
__fastcall GetImports(TDOMStrings ImportNames);
```

Description

Call GetImports to retrieve the namespaces for of all documents that are imported into this one using an <import> tag. The location of these imported documents can be retrieved from the namespace using the GetLocationForImport (see page 7224)

method.

ImportNames is an existing TWideStrings object to which the namespaces for any imports are added.

Note: GetImports does not clear the WideString (see page 3467) list before adding the namespaces for all imported documents.

See Also

GetLocationForImport (see page 7224)

75.1.1.3.9 TWSDLItems.GetLocationForImport Method

Fills a string list with the names of all messages that must be passed in or out when calling a specified operation.

Pascal

```
function GetLocationForImport(const ImportNameSpace: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetLocationForImport(const DOMString ImportNameSpace);
```

Description

Call GetLocationForImport to retrieve the URI that indicates the location of an imported document, given its namespace.

ImportNameSpace is the namespace that identifies another document that is imported into this WSDL document. It is one of the strings retrieved by the GetImports (see page 7223) method.

See Also

GetImports (see page 7223)

75.1.1.3.10 TWSDLItems.GetMessageNode Method

Returns the interface for the node that describes a message.

Pascal

```
function GetMessageNode(const MessageName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetMessageNode(const DOMString MessageName);
```

Description

Call GetMessageNode to obtain information about a message defined in the WSDL document. A message describes a single item of data that is passed in or out when executing an operation (method). The parameters and return value of a method each map onto a message.

MessageName is the name of the message whose node is returned.

See Also

GetMessages (see page 7224)

GetParts (see page 7227)

75.1.1.3.11 TWSDLItems.GetMessages Method

Fills a WideString (see page 3467) list with the names of all messages defined in the WSDL document.

Pascal

```
procedure GetMessages(MessageNames: TDOMStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetMessages(TDOMStrings MessageNames, Boolean QualifiedNames = False);
```

Description

Call GetMessages to retrieve the names of all messages defined in the WSDL document. Each message describes a single item of data that is passed in or out when executing an operation (method). The parameters and return value of a method each map onto a message.

MessageNames is an existing TWideStrings object to which the names of all messages are added.

QualifiedNames indicates whether the message names added to the list should be qualified using the target namespace prefix.

Note: GetMessages does not clear the WideString (see page 3467) list before adding the message names defined in the WSDL document.

See Also

GetMessageNode (see page 7224)

GetParts (see page 7227)

75.1.1.3.12 TWSDLItems.GetName Method

Returns the name of the definitions node in the WSDL document.

Pascal

```
function GetName: DOMString;
```

C++

```
__fastcall DOMString GetName();
```

Description

The root node of a WSDL document is always a definitions node. This node gives a name to the set of definitions in the WSDL document. Call GetName to retrieve that name.

See Also

Definition (see page 7070)

GetTargetNamespace (see page 7242)

75.1.1.3.13 TWSDLItems.GetOperationNode Method

Returns the interface for the node that describes an operation.

Pascal

```
function GetOperationNode(const PortTypeName: DOMString; const OperationName: DOMString):
IXMLNode;
```

C++

```
__fastcall IXMLNode GetOperationNode(const DOMString PortTypeName, const DOMString
OperationName);
```

Description

Call GetOperationNode to obtain information about an operation defined in the WSDL document. An operation corresponds to a

method on the interface that corresponds to a specified port type.

PortTypeName is the name of the port type that contains the specified operation.

OperationName is the name of the operation whose node is returned.

See Also

GetOperations (see page 7226)

GetOperationsForBinding (see page 7226)

GetPortTypes (see page 7229)

75.1.1.3.14 TWSDLItems.GetOperations Method

Fills a WideString (see page 3467) list with the names of all operations defined for a specified port type.

Pascal

```
procedure GetOperations(const PortTypeName: DOMString; OperationNames: TDOMStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetOperations(const DOMString PortTypeName, TDOMStrings OperationNames, Boolean
QualifiedNames = False);
```

Description

Call GetOperations to retrieve the names of all operations defined in for the port type specified by PortTypeName. An operation corresponds to a method on the interface that corresponds to the specified port type.

PortTypeName is the name of the port type whose operations are retrieved.

OperationNames is an existing TWideStrings object to which the names of all operations on PortTypeName are added.

QualifiedNames indicates whether the operation names added to the list should be qualified using the target namespace prefix.

Note: GetOperations does not clear the WideString (see page 3467) list before adding the names of the operations on the specified port type.

See Also

GetOperationNode (see page 7225)

GetOperationsForBinding (see page 7226)

GetPortTypes (see page 7229)

75.1.1.3.15 TWSDLItems.GetOperationsForBinding Method

Fills a list with the names of all operations defined on the port type of a specified binding.

Pascal

```
procedure GetOperationsForBinding(const BindingName: DOMString; OperationNames: TDOMStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetOperationsForBinding(const DOMString BindingName, TDOMStrings OperationNames,
Boolean QualifiedNames = False);
```

Description

Call GetOperationsForBinding to retrieve the names of all operations defined for the port type associated with a specified

binding. An operation corresponds to a method on the interface that corresponds to a port type. A binding represents a port type, plus the protocols used to encode data and transmit requests when calling that port type.

BindingName is the name of the binding associated with the port type whose operations are retrieved.

OperationNames is an existing TWideStrings object to which the names of all operations on the binding's port type are added.

QualifiedNames specifies whether the operation names should be qualified using the target namespace prefix.

Note: GetOperationsForBinding does not clear the WideString (see page 3467) list before adding the names of the operations.

See Also

GetOperationNode (see page 7225)

GetOperations (see page 7226)

GetBindings (see page 7222)

GetTargetNamespacePrefix (see page 7243)

75.1.1.3.16 TWSDLItems.GetPartNode Method

Returns the interface for the node that describes a part of a message.

Pascal

```
function GetPartNode(const MessageName: DOMString; const PartName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetPartNode(const DOMString MessageName, const DOMString PartName);
```

Description

Call GetPartNode to obtain information about a part that is defined for a message. Every message consists of one or more parts, each of which is associated with a definition from some type system.

MessageName is the name of the message that is defined using the part.

PartName is the name of the part whose node is returned.

See Also

GetParts (see page 7227)

GetMessages (see page 7224)

GetPartsForOperation (see page 7228)

75.1.1.3.17 TWSDLItems.GetParts Method

Fills a list with the names of all parts defined for a specified message.

Pascal

```
procedure GetParts(const MessageName: DOMString; PartNames: TDOMStrings; QualifiedNames:
Boolean = False);
```

C++

```
__fastcall GetParts(const DOMString MessageName, TDOMStrings PartNames, Boolean QualifiedNames
= False);
```

Description

Call GetParts to retrieve the names of all parts that make up a specified message. A part corresponds to definition from some type system.

MessageName is the name of the message whose parts are retrieved.

PartNames is an existing TWideStrings object to which the names of all parts on MessageName are added.

QualifiedNames indicates whether the part names added to the list should be qualified using the target namespace prefix.

Note: GetParts does not clear the WideString (see page 3467) list before adding the names of the parts of the specified message.

See Also

GetPartNode (see page 7227)

GetMessages (see page 7224)

GetPartsForOperation (see page 7228)

75.1.1.3.18 TWSDLItems.GetPartsForOperation Method

Fills a list with the names of all parts defined for input messages of a specified operation.

Pascal

```
function GetPartsForOperation(const PortTypeName: DOMString; const OperationName: DOMString;
OperationIndex: Integer; PartNames: TDOMStrings): Boolean;
```

C++

```
__fastcall Boolean GetPartsForOperation(const DOMString PortTypeName, const DOMString
OperationName, int OperationIndex, TDOMStrings PartNames);
```

Description

Call GetPartsForOperation to retrieve the names of all parts that make up input messages on a specified operation. The part definition must be in the WSDL document or a document that it imports. A part corresponds to definition from some type system.

PortTypeName is the name of the port type that contains the operation. A port type corresponds to an invokable interface in a Web Service application.

OperationName is the name of the operation from whose input messages the parts are retrieved. An operation corresponds to a method on the interface that corresponds to PortTypeName.

PartNames is an existing TWideStrings object to which the names of all parts on input messages (input parameters) of the operation specified by OperationName.

GetPartsForOperation returns true if it adds any part names to PartNames, false if there are no parts to add.

Note: GetPartsForOperation does not clear the WideString (see page 3467) list before adding the names of the parts of the specified message.

See Also

GetPartNode (see page 7227)

GetParts (see page 7227)

GetOperations (see page 7226)

75.1.1.3.19 TWSDLItems.GetPortsForService Method

Fills a list with the names of all ports defined within a specified service.

Pascal

```
procedure GetPortsForService(const ServiceName: DOMString; PortNames: TDOMStrings;
SkipHttpBindings: Boolean = True; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetPortsForService(const DOMString ServiceName, TDOMStrings PortNames, Boolean
SkipHttpBindings = True, Boolean QualifiedNames = False);
```

Description

Call GetPortsForService to retrieve the names of all ports defined for the specified service. A port represents a binding, plus the address where that binding can be found.

ServiceName is the name of the service whose ports are listed.

PortNames is an existing TWideStrings object to which the names of all ports in the specified service are added.

SkipHttpBindings indicates whether GetPortsForService should include port types whose definitions include an <http:binding> tag. When false, GetPortsForService retrieves all port type definitions. When true, GetPortsForService ignores port type definitions with simple HTTP bindings.

QualifiedNames indicates whether the port names added to the list should be qualified using the target namespace prefix.

Note: GetPortsForService does not clear the WideString (see page 3467) list before adding the names of the ports.

See Also

GetBindingForServicePort (see page 7220)

GetServices (see page 7232)

75.1.1.3.20 TWSDLItems.GetPortTypeNode Method

Returns the interface for the node that describes a port type defined in the WSDL document.

Pascal

```
function GetPortTypeNode(const PortTypeName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetPortTypeNode(const DOMString PortTypeName);
```

Description

Call GetPortTypeNode to obtain information about a port type that is defined in the WSDL document or a document that it imports. Each port type corresponds to an interface that the Web Service application can execute.

PortTypeName is the name of the port type whose node is returned.

See Also

GetPortTypes (see page 7229)

75.1.1.3.21 TWSDLItems.GetPortTypes Method

Fills a list with the names of all port types defined in the WSDL document.

Pascal

```
procedure GetPortTypes(PortTypeNames: TDOMStrings; SkipHttpBindings: Boolean = True;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetPortTypes(TDOMStrings PortTypeNames, Boolean SkipHttpBindings = True, Boolean
QualifiedNames = False);
```

Description

Call GetPortTypes to retrieve the names of all port types defined in the WSDL document or any document that it imports. Each port type corresponds to an interface that the Web Service application can execute.

PortTypeNames is an existing TWideStrings object to which the names of all port types are added.

SkipHttpBindings indicates whether GetPortTypes should include port types whose definitions include an <http:binding> tag. When false, GetPortTypes retrieves all port type definitions. When true, GetPortTypes ignores port type definitions with simple HTTP bindings.

QualifiedNames indicates whether the port type names added to the list should be qualified using the target namespace prefix.

Note: GetPortTypes does not clear the WideString (see page 3467) list before adding the names of the port types.

See Also

GetPortTypeNode (see page 7229)

GetOperations (see page 7226)

75.1.1.3.22 TWSDLItems.GetSchemaNode Method

Returns the interface for the schema node that defines a type.

Pascal

```
function GetSchemaNode(const SchemaTns: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSchemaNode(const DOMString SchemaTns);
```

Description

Call GetSchemaNode to obtain information about a schema definition for a type that is defined in the WSDL document.

SchemaTns is the target namespace of the schema definition for a type.

GetSchemaNode returns the corresponding node.

See Also

GetSchemas (see page 7230)

HasTypesNode (see page 7244)

75.1.1.3.23 TWSDLItems.GetSchemas Method

Fills a list with the namespaces for all schema definitions of types in the WSDL document.

Pascal

```
procedure GetSchemas(SchemaNames: TDOMStrings);
```

C++

```
__fastcall GetSchemas(TDOMStrings SchemaNames);
```

Description

Call GetSchemaTypes to retrieve the namespaces that correspond to schema definitions of types.

SchemaNames is an existing TWideStrings object to which the namespaces for the schema definitions are added.

Note: GetSchemas does not clear the WideString (see page 3467) list before adding the namespaces of the schema definitions.

See Also

GetSchemaNode (see page 7230)

HasTypesNode (see page 7244)

75.1.1.3.24 TWSDLItems.GetServiceAndPortOfBinding Method

Retrieves the names of the service and port associated with a specified binding.

Pascal

```
function GetServiceAndPortOfBinding(const BindingName: IQualifiedName; var ServiceName:
DOMString; var PortName: DOMString): Boolean;
```

C++

```
__fastcall Boolean GetServiceAndPortOfBinding(const IQualifiedName BindingName, DOMString
ServiceName, DOMString PortName);
```

Description

Call GetServiceAndPortOfBinding to retrieve the name of the service and port associated with a specified binding. GetServiceAndPortOfBinding performs the converse lookup from the GetBindingForServicePort (see page 7220) method.

BindingName specifies the qualified name of the binding for which you want to look up the service and port.

ServiceName returns the name of the service to which the binding belongs.

PortName returns the name of a port within the service that uses the binding.

GetServiceAndPortOfBinding returns true if it locates the specified binding and returns a service and port. It returns false if the specified binding is not defined in the WSDL document.

See Also

GetServices (see page 7232)

GetServiceNode (see page 7231)

GetBindingForServicePort (see page 7220)

GetPortsForService (see page 7229)

GetBindingOfType (see page 7221)

GetBindings (see page 7222)

75.1.1.3.25 TWSDLItems.GetServiceNode Method

Returns the interface for the node that describes a service defined in the WSDL document.

Pascal

```
function GetServiceNode(const ServiceName: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode GetServiceNode(const DOMString ServiceName);
```

Description

Call GetServiceNode to obtain information about a service that is defined in the WSDL document. A service is a collection of related ports.

ServiceName is the name of the service whose node is returned.

See Also

GetServices (see page 7232)

GetPortsForService (see page 7229)

GetServiceAndPortOfBinding (see page 7231)

75.1.1.3.26 TWSDLItems.GetServices Method

Fills a list with the names of all services defined in the WSDL document.

Pascal

```
procedure GetServices(ServiceNames: TDOMStrings; QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetServices(TDOMStrings ServiceNames, Boolean QualifiedNames = False);
```

Description

Call GetServices to retrieve the names of all services defined in the WSDL document or any document that it imports. Each service is a collection of related ports.

ServiceNames is an existing TWideStrings object to which the names of all services are added.

QualifiedNames indicates whether the service names added to the list should be qualified using the target namespace prefix.

Note: GetServices does not clear the WideString (see page 3467) list before adding the names of the services.

See Also

GetServiceNode (see page 7231)

GetPortsForService (see page 7229)

GetServiceAndPortOfBinding (see page 7231)

75.1.1.3.27 TWSDLItems.GetSoapAction Method

Returns the value of the soapAction attribute of a specified operation on a specified binding.

Pascal

```
function GetSoapAction(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapAction(const DOMString BindingName, const DOMString Operation, int
OverloadIndex);
```

Description

Call GetSoapAction to retrieve the value of the soapAction attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

OverloadIndex indicates which operation is desired when the operation is overloaded.

See Also

GetSoapActionList (see page 7233)

GetBindings (see page 7222)

GetOperationsForBinding (see page 7226)

GetSoapOperationAttribute (see page 7240)

75.1.1.3.28 TWSDLItems.GetSoapActionList Method

Fills a list with the soapAction attributes for all operations on a specified SOAP binding.

Pascal

```
procedure GetSoapActionList(const BindingName: DOMString; ActionList: TDOMStrings;
QualifiedNames: Boolean = False);
```

C++

```
__fastcall GetSoapActionList(const DOMString BindingName, TDOMStrings ActionList, Boolean
QualifiedNames = False);
```

Description

Call GetSoapActionList to retrieve the value of the soapAction attribute of each operation of a specified SOAP binding. Each soapAction is added as a string with the format

```
OperationName = soapActionName
```

where OperationName is the name of the operation with which the soapAction attribute is associated.

BindingName is the name of the binding associated with the port type whose operations are retrieved along with their SOAP actions.

OperationNames is an existing TWideStrings object to which the names of all operations on the binding's port type are added.

QualifiedNames specifies whether the operation names should be qualified using the target namespace prefix.

Note: GetSoapActionList does not clear the WideString (see page 3467) list before adding the names of the operations and their corresponding SOAP actions.

See Also

GetSoapAction (see page 7232)

GetBindings (see page 7222)

GetOperationsForBinding (see page 7226)

GetSoapOperationAttribute (see page 7240)

75.1.1.3.29 TWSDLItems.GetSoapAddressForServicePort Method

Returns the address of a specified port from a specified service.

Pascal

```
function GetSoapAddressForServicePort(const ServiceName: DOMString; const PortName:
DOMString): DOMString;
```

C++

```
__fastcall DOMString GetSoapAddressForServicePort(const DOMString ServiceName, const DOMString
PortName);
```

Description

Call GetSoapAddressForServicePort to retrieve the address to which a SOAP request should be sent if it wants to access a specified port from a specified service.

ServiceName is the name of the service that contains the desired port. It is defined in the WSDL document or another document that it imports.

PortName is the name of the port whose address is returned.

See Also

GetBindingForServicePort (see page 7220)

GetPortsForService (see page 7229)

75.1.1.3.30 TWSDLItems.GetSoapBindingAttribute Method

Returns the value of a specified attribute of a specified SOAP binding.

Pascal

```
function GetSoapBindingAttribute(const BindingName: DOMString; Attribute: DOMString):
DOMString;
```

C++

```
__fastcall DOMString GetSoapBindingAttribute(const DOMString BindingName, DOMString Attribute);
```

Description

Call GetSoapBindingAttribute to retrieve value of an attribute on a SOAP binding. GetSoapBindingAttribute locates a binding with the specified name that includes a <soap:binding> tag, and returns the specified attribute from that <soap:binding> tag.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Attribute is the name of the desired attribute on the <soap:binding> tag.

If BindingName does not identify a binding that includes a <soap:binding> tag that has the specified attribute, GetSoapBindingAttribute returns an empty string.

See Also

GetSoapBodyAttribute (see page 7236)

GetSoapOperationAttribute (see page 7240)

GetBindingForServicePort (see page 7220)

75.1.1.3.31 TWSDLItems.GetSoapBindingInputNode Method

Returns the interface for the input node on a specified binding and operation.

Pascal

```
function GetSoapBindingInputNode(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingInputNode(const DOMString BindingName, const DOMString
Operation, int OverloadIndex);
```

Description

Call GetSoapBindingInputNode to retrieve the node for input messages (parameters) on an operation of a SOAP binding. GetSoapBindingInputNode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the input node for that operation.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingInputNode is unable to locate an input node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingOutputNode (see page 7236)

GetSoapBodyAttribute (see page 7236)

GetSoapBindingIONode (see page 7235)

GetBindingForServicePort (see page 7220)

75.1.1.3.32 TWSDLItems.GetSoapBindingIONode Method

Returns the interface for a node on a specified binding and operation.

Pascal

```
function GetSoapBindingIONode(const BindingName: DOMString; const Operation: DOMString; Input:
Boolean; OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingIONode(const DOMString BindingName, const DOMString
Operation, Boolean Input, int OverloadIndex);
```

Description

Call GetSoapBindingIONode to retrieve a specified node on an operation of a SOAP binding. GetSoapBindingIONode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the child of that node with a specified name. This method provides the underlying implementation for the GetSoapBindingInputNode (see page 7235) and GetSoapBindingOutputNode (see page 7236) methods.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

Input (see page 3827) indicates whether the child node is an input node (true) or an output node (false).

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingIONode is unable to locate a node for the specified name, binding, and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingOutputNode (see page 7236)

GetSoapBodyAttribute (see page 7236)

GetSoapBindingInputNode (see page 7235)

GetBindingForServicePort (see page 7220)

75.1.1.3.33 TWSDLItems.GetSoapBindingOutputNode Method

Returns the interface for the output node on a specified binding and operation.

Pascal

```
function GetSoapBindingOutputNode(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetSoapBindingOutputNode(const DOMString BindingName, const DOMString
Operation, int OverloadIndex);
```

Description

Call GetSoapBindingOutputNode to retrieve the node for output messages (output parameters or return value) on an operation of a SOAP binding. GetSoapBindingOutputNode locates a binding with the specified name that includes a <soap:binding> tag, locates the specified operation node that is a child of that binding, and returns the output node for that operation.

BindingName is the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

If GetSoapBindingOutputNode is unable to locate an output node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapBindingInputNode (see page 7235)

GetSoapBodyAttribute (see page 7236)

GetSoapBindingIONode (see page 7235)

GetBindingForServicePort (see page 7220)

75.1.1.3.34 TWSDLItems.GetSoapBodyAttribute Method

Returns the value of an attribute of a body tag of a message type in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyAttribute(const BindingName: DOMString; const Operation: DOMString; const
IOType: DOMString; const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyAttribute(const DOMString BindingName, const DOMString
Operation, const DOMString IOType, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyAttribute to retrieve value of an attribute on the body tag for a message type that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the message.

IOType indicates the type of the message (input or output).

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetSoapBindingAttribute (see page 7234)

GetSoapOperationAttribute (see page 7240)

GetBindingForServicePort (see page 7220)

GetOperationsForBinding (see page 7226)

GetSoapBodyInputAttribute (see page 7237)

GetSoapBodyOutputAttribute (see page 7238)

75.1.1.3.35 TWSDLItems.GetSoapBodyInputAttribute Method

Returns the value of an attribute of a body tag of an input message in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyInputAttribute(const BindingName: DOMString; const Operation: DOMString;
const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyInputAttribute(const DOMString BindingName, const DOMString
Operation, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyInputAttribute to retrieve value of an attribute on the body tag for an input message that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the input message.

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

Note: GetSoapBodyInputAttribute returns the same value as GetSoapBodyAttribute (see page 7236) with an IOType of 'input'.

See Also

GetSoapBodyAttribute (see page 7236)

GetSoapBodyOutputAttribute (see page 7238)

GetSoapOperationAttribute (see page 7240)

GetBindingForServicePort (see page 7220)

GetOperationsForBinding (see page 7226)

75.1.1.3.36 TWSDLItems.GetSoapBodyNamespace Method

Returns the value of the first namespace attribute found on a SOAP body node inside a binding with a specified port type.

Pascal

```
function GetSoapBodyNamespace(const BindingPortType: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyNamespace(const DOMString BindingPortType);
```

Description

GetSoapBodyNamespace searches the bindings in the WSDL document and any documents it imports for one that has a port type that matches BindingPortType. It then searches the operations of that binding to find one whose messages include a SOAP body node. It returns the namespace attribute of that SOAP body node.

PortTypeName is the name of the port type (interface) that must appear in the binding.

See Also

GetSoapBodyAttribute (see page 7236)

GetPortTypes (see page 7229)

75.1.1.3.37 TWSDLItems.GetSoapBodyOutputAttribute Method

Returns the value of an attribute of a body tag of an output message in an operation on the port type of a SOAP binding.

Pascal

```
function GetSoapBodyOutputAttribute(const BindingName: DOMString; const Operation: DOMString;
const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapBodyOutputAttribute(const DOMString BindingName, const DOMString
Operation, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapBodyOutputAttribute to retrieve value of an attribute on the body tag for an output message that is part of an operation of the port type associated with a specified binding. The body tag can come from the WSDL document or another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the output message.

Attribute is the name of the desired attribute on the <soap:body> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

Note: GetSoapBodyOutputAttribute returns the same value as GetSoapBodyAttribute (see page 7236) with an IOType of 'output'.

See Also

GetSoapBodyAttribute (see page 7236)

GetSoapBodyInputAttribute (see page 7237)

GetSoapOperationAttribute (see page 7240)

GetBindingForServicePort (see page 7220)

GetOperationsForBinding (see page 7226)

75.1.1.3.38 TWSDLItems.GetSoapHeaders Method

Returns an interface for accessing a specified set of header nodes.

Pascal

```
function GetSoapHeaders(BindingName: IQualifiedName; const Operation: DOMString; Input:
Boolean; OverloadIndex: Integer; const MessageName: DOMString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapHeaders(IQualifiedName BindingName, const DOMString
Operation, Boolean Input, int OverloadIndex, const DOMString MessageName = '');
```

Description

Call GetSoapHeaders to retrieve a dynamic array of interfaces for accessing information about the header nodes of a specified operation on a specified binding. This method provides the underlying implementation for the GetSoapInputHeaders (see page 7240) and GetSoapOutputHeaders (see page 7241) methods.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

Input (see page 3827) indicates whether the header node should belong to the operation's input node (true) or output node (false).

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapHeaders does not check the message with which the header is associated.

If GetSoapHeaders is unable to locate a header node for the specified name, binding, and IOType, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapOutputHeaders (see page 7241)

GetSoapBindingIONode (see page 7235)

GetSoapInputHeaders (see page 7240)

GetBindingForServicePort (see page 7220)

GetPartNode (see page 7227)

75.1.1.3.39 TWSDLItems.GetSoapInputHeaders Method

Returns an interface for accessing the header nodes that belong to the input node of a specified binding and operation.

Pascal

```
function GetSoapInputHeaders(BindingName: IQualifiedName; const Operation: DOMString;
OverloadIndex: Integer; const MessageName: DOMString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapInputHeaders(IQualifiedName BindingName, const DOMString
Operation, int OverloadIndex, const DOMString MessageName = '');
```

Description

Call GetSoapInputHeaders to retrieve a dynamic array of interfaces for accessing the header nodes of the input node on a specified binding. GetSoapInputHeaders locates the specified input node using the GetSoapBindingIONode (see page 7235) method, and extracts all of its child header nodes. It then checks the message and part attributes of that input node, and returns a dynamic array of interfaces for the headers.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapInputHeaders does not check the message with which the header is associated.

If GetSoapInputHeaders is unable to locate an input header node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapOutputHeaders (see page 7241)

GetSoapBindingIONode (see page 7235)

GetSoapHeaders (see page 7239)

GetBindingForServicePort (see page 7220)

GetPartNode (see page 7227)

75.1.1.3.40 TWSDLItems.GetSoapOperationAttribute Method

Returns the value of an attribute on the <soap:operation> node of an operation on a SOAP binding.

Pascal

```
function GetSoapOperationAttribute(const BindingName: DOMString; const Operation: DOMString;
const Attribute: DOMString; OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapOperationAttribute(const DOMString BindingName, const DOMString
```

```
Operation, const DOMString Attribute, int OverloadIndex);
```

Description

Call GetSoapOperationAttribute to retrieve value of an attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

Attribute is the name of the desired attribute on the <soap:operation> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetSoapBindingAttribute (see page 7234)

GetSoapBodyAttribute (see page 7236)

75.1.1.3.41 TWSDLItems.GetSoapOperationStyle Method

Returns the value of the style attribute of a specified operation on a specified binding.

Pascal

```
function GetSoapOperationStyle(const BindingName: DOMString; const Operation: DOMString;
OverloadIndex: Integer): DOMString;
```

C++

```
__fastcall DOMString GetSoapOperationStyle(const DOMString BindingName, const DOMString
Operation, int OverloadIndex);
```

Description

Call GetSoapOperationStyle to retrieve the value of the style attribute on the <soap:operation> tag associated with an operation on a specified SOAP binding. The soap operation tag can be in the WSDL document or in another document that it imports.

BindingName is the name of the binding that includes the operation.

Operation is the name of the operation that includes the <soap:operation> tag.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

See Also

GetBindings (see page 7222)

GetOperationsForBinding (see page 7226)

GetSoapOperationAttribute (see page 7240)

75.1.1.3.42 TWSDLItems.GetSoapOutputHeaders Method

Returns an interface for accessing the header nodes that belong to the output node of a specified binding and operation.

Pascal

```
function GetSoapOutputHeaders(BindingName: IQualifiedName; const Operation: DOMString;
OverloadIndex: Integer; const MessageName: DOMString = ''): IHeaderInfoArray;
```

C++

```
__fastcall IHeaderInfoArray GetSoapOutputHeaders(IQualifiedName BindingName, const DOMString
Operation, int OverloadIndex, const DOMString MessageName = '');
```

Description

Call GetSoapOutputHeaders to retrieve a dynamic array of interfaces for accessing the header nodes of the output node on a specified binding. GetSoapOutputHeaders locates the specified output node using the GetSoapBindingIONode (see page 7235) method, and extracts all of its child header nodes. It then checks the message and part attributes of that output node, and returns a dynamic array of interfaces for the headers.

BindingName is an interface for reading the name of the binding that includes the <soap:binding> tag. A binding describes the encoding and transport protocols for an associated port type.

Operation is the name of the operation of that <soap:binding> tag. Operations correspond to methods on an invokable interface.

OverloadIndex indicates which operation node is desired when the binding has multiple operations with the same name (overloaded methods). 0 indicates the first operation with the specified name, 1 indicates the second operation node, and so on.

MessageName is the name of a Message with which the header should be associated. If MessageName is an empty string, GetSoapOutputHeaders does not check the message with which the header is associated.

If GetSoapOutputHeaders is unable to locate an output header node for the specified binding and operation, it returns nil (Delphi) or NULL (C++).

See Also

GetSoapInputHeaders (see page 7240)

GetSoapBindingIONode (see page 7235)

GetSoapHeaders (see page 7239)

GetBindingForServicePort (see page 7220)

GetPartNode (see page 7227)

75.1.1.3.43 TWSDLItems.GetTargetNamespace Method

Returns the value of the TargetNamespace (see page 7246) property.

Pascal

```
function GetTargetNamespace: DOMString;
```

C++

```
__fastcall DOMString GetTargetNamespace();
```

Description

GetTargetNamespace is the read access method for the TargetNamespace (see page 7246) property.

See Also

Definition (see page 7070)

GetTargetNamespacePrefix (see page 7243)

TargetNamespace (see page 7246)

75.1.1.3.44 TWSDLItems.GetTargetNamespacePrefix Method

Returns the prefix string that represents the target namespace of the WSDL document.

Pascal

```
function GetTargetNamespacePrefix: DOMString;
```

C++

```
__fastcall DOMString GetTargetNamespacePrefix();
```

Description

Call GetTargetNamespacePrefix to retrieve the prefix string that should be added to node names so that they explicitly use the target namespace of the WSDL document. Node names that include this prefix are defined in the namespace identified by the URL that is the value of the TargetNamespace (see page 7246) property.

See Also

TargetNamespace (see page 7246)

75.1.1.3.45 TWSDLItems.GetWSDLDoc Method

Returns the interface for this TWSDLItems (see page 7211) instance.

Pascal

```
function GetWSDLDoc: IWSDLDocument;
```

C++

```
__fastcall IWSDLDocument GetWSDLDoc();
```

Description

GetWSDLDoc returns the IWSDLDocument (see page 7164) for this TWSDLItems (see page 7211) instance. This interface can be used to access the Definition property, which supplies uninterpreted access to the definitions in the WSDL document.

See Also

Definition (see page 7070)

75.1.1.3.46 TWSDLItems.GetWSDLItems Method

Returns a reference to this TWSDLItems (see page 7211) instance.

Pascal

```
function GetWSDLItems: TWSDLItems;
```

C++

```
__fastcall TWSDLItems GetWSDLItems();
```

Description

GetWSDLItems returns a reference to the TWSDLItems (see page 7211) instance. This method is mostly used by code that holds an IWSDLItems (see page 7256) interface and needs to use the methods of the object itself.

See Also

IWSDLItems (see page 7256)

75.1.1.3.47 TWSDLItems.HasTypesNode Method

Indicates whether the WSDL document includes any type definitions.

Pascal

```
function HasTypesNode: Boolean;
```

C++

```
__fastcall Boolean HasTypesNode();
```

Description

HasTypesNode returns true if the WSDL document includes any type definitions, false otherwise.

75.1.1.3.48 TWSDLItems.IsPortTypeHTTPBound Method

Indicates whether a specified port type has an HTTP binding.

Pascal

```
function IsPortTypeHTTPBound(const PortType: DOMString): Boolean;
```

C++

```
__fastcall Boolean IsPortTypeHTTPBound(const DOMString PortType);
```

Description

Call IsPortTypeHTTPBound to determine whether there is an HTTP binding for a specified port type. A port type corresponds to an invokable interface.

PortType is the port type you want to check.

IsPortTypeHTTPBound returns true if the specified port type has an HTTP binding, false otherwise.

See Also

GetPortTypes (see page 7229)

GetPortTypeNode (see page 7229)

GetBindingOfType (see page 7221)

75.1.1.3.49 TWSDLItems.IterateImports Method

Executes a callback for every WSDL document that this one imports.

Pascal

```
procedure IterateImports(IterateProc: TWSDLIterateProc);
```

C++

```
__fastcall IterateImports(TWSDLIterateProc IterateProc);
```

Description

IterateImports is used to execute a callback before and after loading each WSDL document that this one imports, including those documents that are imported by documents that this one imports.

75.1.1.3.50 TWSDLItems.Load Method

Loads a WSDL document and activates it.

Pascal

```
procedure Load(const WSDLFileName: DOMString);
```

C++

```
__fastcall Load(const DOMString WSDLFileName);
```

Description

Load uses the stream loader specified by the StreamLoader (see page 7245) property to load a WSDL document into memory. It then sets itself to represent that WSDL document and activates it.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found.

See Also

StreamLoader (see page 7245)

LoadFromFile (see page 7368)

LoadFromStream (see page 7369)

LoadFromXML (see page 7370)

75.1.1.4 TWSDLItems Properties

The properties of the TWSDLItems class are listed here.

Public Properties

	Name	Description
R	StreamLoader (see page 7245)	Provides access to an interface for loading the WSDL document.
R	TargetNamespace (see page 7246)	Provides access to an interface for loading the WSDL document.

75.1.1.4.1 TWSDLItems.StreamLoader Property

Provides access to an interface for loading the WSDL document.

Pascal

```
property StreamLoader: IStreamLoader;
```

C++

```
__property IStreamLoader StreamLoader;
```

Description

StreamLoader is set by the TWSDLItems (see page 7211) constructor. It is an interface that specifies basic authentication information that can be used to load a WSDL document that is on another server. The IStreamLoader interface specifies the password, user name, and proxy server that can be used to connect to the server that hosts the WSDL document. TWSDLItems (see page 7211) uses this to load the WSDL document.

See Also

Create (see page 7217)

75.1.1.4.2 TWSDLItems.TargetNamespace Property

Provides access to an interface for loading the WSDL document.

Pascal

```
property TargetNamespace: DOMString;
```

C++

```
__property DOMString TargetNamespace;
```

Description

The root node of a WSDL document is always a definitions node. This node includes a TargetNamespace attribute that uniquely identifies the set of definitions in the WSDL document. Read (see page 3634) TargetNamespace to retrieve that namespace.

See Also

Definition (see page 7070)

GetTargetNamespacePrefix (see page 7243)

75.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IHeaderInfo (see page 7247)	This is class WSDLItems.IHeaderInfo.
	IQualifiedName (see page 7252)	IQualifiedName is the interface for obtaining an information about a name in an XML document.
	IWSDLItems (see page 7256)	IWSDLItems provides access to a WSDL document.

75.2.1 IHeaderInfo Interface

Class Hierarchy

WSDLItems.IHeaderInfo

Pascal

```
IHeaderInfo = interface;
```

C++

```
__interface IHeaderInfo;
```

File

WSDLItems

Description

This is class WSDLItems.IHeaderInfo.

75.2.1.1 IHeaderInfo Members

The following tables list the members exposed by IHeaderInfo.

Public Methods

	Name	Description
	GetEncodingStyle (see page 7248)	This is GetEncodingStyle, a member of class IHeaderInfo.
	GetName (see page 7248)	This is GetName, a member of class IHeaderInfo.
	GetNamespace (see page 7249)	This is GetNamespace, a member of class IHeaderInfo.
	GetPart (see page 7249)	This is GetPart, a member of class IHeaderInfo.
	GetRequired (see page 7249)	This is GetRequired, a member of class IHeaderInfo.
	GetUse (see page 7249)	This is GetUse, a member of class IHeaderInfo.
	SetEncodingStyle (see page 7249)	This is SetEncodingStyle, a member of class IHeaderInfo.
	SetName (see page 7250)	This is SetName, a member of class IHeaderInfo.
	SetNamespace (see page 7250)	This is SetNamespace, a member of class IHeaderInfo.
	SetPart (see page 7250)	This is SetPart, a member of class IHeaderInfo.
	SetRequired (see page 7250)	This is SetRequired, a member of class IHeaderInfo.
	SetUse (see page 7251)	This is SetUse, a member of class IHeaderInfo.

Public Properties

	Name	Description
	EncodingStyle (see page 7251)	This is EncodingStyle, a member of class IHeaderInfo.
	Name (see page 7251)	This is Name, a member of class IHeaderInfo.
	Namespace (see page 7251)	This is Namespace, a member of class IHeaderInfo.
	Part (see page 7252)	This is Part, a member of class IHeaderInfo.
	Required (see page 7252)	This is Required, a member of class IHeaderInfo.
	Use (see page 7252)	This is Use, a member of class IHeaderInfo.

75.2.1.2 IHeaderInfo Methods

The methods of the IHeaderInfo class are listed here.

Public Methods

	Name	Description
	GetEncodingStyle (see page 7248)	This is GetEncodingStyle, a member of class IHeaderInfo.
	GetName (see page 7248)	This is GetName, a member of class IHeaderInfo.
	GetNamespace (see page 7249)	This is GetNamespace, a member of class IHeaderInfo.
	GetPart (see page 7249)	This is GetPart, a member of class IHeaderInfo.
	GetRequired (see page 7249)	This is GetRequired, a member of class IHeaderInfo.
	GetUse (see page 7249)	This is GetUse, a member of class IHeaderInfo.
	SetEncodingStyle (see page 7249)	This is SetEncodingStyle, a member of class IHeaderInfo.
	SetName (see page 7250)	This is SetName, a member of class IHeaderInfo.
	SetNamespace (see page 7250)	This is SetNamespace, a member of class IHeaderInfo.
	SetPart (see page 7250)	This is SetPart, a member of class IHeaderInfo.
	SetRequired (see page 7250)	This is SetRequired, a member of class IHeaderInfo.
	SetUse (see page 7251)	This is SetUse, a member of class IHeaderInfo.

75.2.1.2.1 IHeaderInfo.GetEncodingStyle Method

Pascal

```
function GetEncodingStyle: DOMString;
```

C++

```
__fastcall DOMString GetEncodingStyle();
```

Description

This is GetEncodingStyle, a member of class IHeaderInfo.

75.2.1.2.2 IHeaderInfo.GetName Method

Pascal

```
function GetName: DOMString;
```

C++

```
__fastcall DOMString GetName();
```

Description

This is GetName, a member of class IHeaderInfo.

75.2.1.2.3 IHeaderInfo.GetNamespace Method

Pascal

```
function GetNamespace: DOMString;
```

C++

```
__fastcall DOMString GetNamespace();
```

Description

This is GetNamespace, a member of class IHeaderInfo.

75.2.1.2.4 IHeaderInfo.GetPart Method

Pascal

```
function GetPart: IPart;
```

C++

```
__fastcall IPart GetPart();
```

Description

This is GetPart, a member of class IHeaderInfo.

75.2.1.2.5 IHeaderInfo.GetRequired Method

Pascal

```
function GetRequired: Boolean;
```

C++

```
__fastcall Boolean GetRequired();
```

Description

This is GetRequired, a member of class IHeaderInfo.

75.2.1.2.6 IHeaderInfo.GetUse Method

Pascal

```
function GetUse: DOMString;
```

C++

```
__fastcall DOMString GetUse();
```

Description

This is GetUse, a member of class IHeaderInfo.

75.2.1.2.7 IHeaderInfo.SetEncodingStyle Method

Pascal

```
procedure SetEncodingStyle(const EncStyl: DOMString);
```

C++

```
__fastcall SetEncodingStyle(const DOMString EncStyl);
```

Description

This is SetEncodingStyle, a member of class IHeaderInfo.

75.2.1.2.8 IHeaderInfo.SetName Method

Pascal

```
procedure SetName(const N: DOMString);
```

C++

```
__fastcall SetName(const DOMString N);
```

Description

This is SetName, a member of class IHeaderInfo.

75.2.1.2.9 IHeaderInfo.SetNamespace Method

Pascal

```
procedure SetNamespace(const N: DOMString);
```

C++

```
__fastcall SetNamespace(const DOMString N);
```

Description

This is SetNamespace, a member of class IHeaderInfo.

75.2.1.2.10 IHeaderInfo.SetPart Method

Pascal

```
procedure SetPart(const P: IPart);
```

C++

```
__fastcall SetPart(const IPart P);
```

Description

This is SetPart, a member of class IHeaderInfo.

75.2.1.2.11 IHeaderInfo.SetRequired Method

Pascal

```
procedure SetRequired(Req: Boolean);
```

C++

```
__fastcall SetRequired(Boolean Req);
```

Description

This is SetRequired, a member of class IHeaderInfo.

75.2.1.2.12 IHeaderInfo.SetUse Method

Pascal

```
procedure SetUse(const U: DOMString);
```

C++

```
__fastcall SetUse(const DOMString U);
```

Description

This is SetUse, a member of class IHeaderInfo.

75.2.1.3 IHeaderInfo Properties

The properties of the IHeaderInfo class are listed here.

Public Properties

	Name	Description
	EncodingStyle (see page 7251)	This is EncodingStyle, a member of class IHeaderInfo.
	Name (see page 7251)	This is Name, a member of class IHeaderInfo.
	Namespace (see page 7251)	This is Namespace, a member of class IHeaderInfo.
	Part (see page 7252)	This is Part, a member of class IHeaderInfo.
	Required (see page 7252)	This is Required, a member of class IHeaderInfo.
	Use (see page 7252)	This is Use, a member of class IHeaderInfo.

75.2.1.3.1 IHeaderInfo.EncodingStyle Property

Pascal

```
property EncodingStyle: DOMString;
```

C++

```
__property DOMString EncodingStyle;
```

Description

This is EncodingStyle, a member of class IHeaderInfo.

75.2.1.3.2 IHeaderInfo.Name Property

Pascal

```
property Name: DOMString;
```

C++

```
__property DOMString Name;
```

Description

This is Name, a member of class IHeaderInfo.

75.2.1.3.3 IHeaderInfo.Namespace Property

Pascal

```
property Namespace: DOMString;
```

C++

```
__property DOMString Namespace;
```

Description

This is Namespace, a member of class IHeaderInfo.

75.2.1.3.4 IHeaderInfo.Part Property

Pascal

```
property Part: IPart;
```

C++

```
__property IPart Part;
```

Description

This is Part, a member of class IHeaderInfo.

75.2.1.3.5 IHeaderInfo.Required Property

Pascal

```
property Required: Boolean;
```

C++

```
__property Boolean Required;
```

Description

This is Required, a member of class IHeaderInfo.

75.2.1.3.6 IHeaderInfo.Use Property

Pascal

```
property Use: DOMString;
```

C++

```
__property DOMString Use;
```

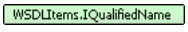
Description

This is Use, a member of class IHeaderInfo.

75.2.2 IQualifiedName Interface

IQualifiedName is the interface for obtaining an information about a name in an XML document.

Class Hierarchy



Pascal

```
IQualifiedName = interface;
```

C++

```
__interface IQualifiedName;
```

File

WSDLItems

Description

IQualifiedName introduces two properties, Name (see page 7255) and Namespace (see page 7255), which represent the local name and namespace URI for a name that is used in an XML document. Using IQualifiedName makes it easier to work with names where it is important to know the namespace in which the name is defined.

Note: C++ method declarations that use IQualifiedName use the _di_IQualifiedName type instead. This type is a DelphiInterface (see page 3256) wrapper around the IQualifiedName interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IQualifiedName >  _di_IQualifiedName;
```

See Also

TWSDLItems (see page 7211)

TMemoryStream

75.2.2.1 IQualifiedName Members

The following tables list the members exposed by IQualifiedName.

Public Methods

	Name	Description
	GetName (see page 7253)	Returns the value of the Name (see page 7255) property.
	GetNamespace (see page 7254)	Returns the value of the Namespace (see page 7255) property.
	SetName (see page 7254)	Sets the value of the Name (see page 7255) property.
	SetNamespace (see page 7254)	Sets the value of the Namespace (see page 7255) property.

Public Properties

	Name	Description
	Name (see page 7255)	Specifies the local name.
	Namespace (see page 7255)	Specifies the namespace in which the name is defined.

75.2.2.2 IQualifiedName Methods

The methods of the IQualifiedName class are listed here.

Public Methods

	Name	Description
	GetName (see page 7253)	Returns the value of the Name (see page 7255) property.
	GetNamespace (see page 7254)	Returns the value of the Namespace (see page 7255) property.
	SetName (see page 7254)	Sets the value of the Name (see page 7255) property.
	SetNamespace (see page 7254)	Sets the value of the Namespace (see page 7255) property.

75.2.2.2.1 IQualifiedName.GetName Method

Returns the value of the Name (see page 7255) property.

Pascal

```
function GetName: DOMString;
```

C++

```
__fastcall DOMString GetName();
```

Description

GetName is the read implementation for the Name (see page 7255) property.

See Also

Name (see page 7255)

75.2.2.2.2 IQualifiedName.GetNamespace Method

Returns the value of the Namespace (see page 7255) property.

Pascal

```
function GetNamespace: DOMString;
```

C++

```
__fastcall DOMString GetNamespace();
```

Description

GetNamespace is the read implementation for the Namespace (see page 7255) property.

See Also

Namespace (see page 7255)

75.2.2.2.3 IQualifiedName.SetName Method

Sets the value of the Name (see page 7255) property.

Pascal

```
procedure SetName(const name: DOMString);
```

C++

```
__fastcall SetName(const DOMString name);
```

Description

SetName is the write implementation for the Name (see page 7255) property.

Name (see page 7255) is the new name.

See Also

Name (see page 7255)

75.2.2.2.4 IQualifiedName.SetNamespace Method

Sets the value of the Namespace (see page 7255) property.

Pascal

```
procedure SetNamespace(const ns: DOMString);
```

C++

```
__fastcall SetNamespace(const DOMString ns);
```

Description

SetNamespace is the write implementation for the Namespace (see page 7255) property.

Namespace (see page 7255) is the new namespace in which to interpret Name (see page 7255).

See Also

Namespace (see page 7255)

75.2.2.3 IQualifiedName Properties

The properties of the IQualifiedName class are listed here.

Public Properties

	Name	Description
	Name (see page 7255)	Specifies the local name.
	Namespace (see page 7255)	Specifies the namespace in which the name is defined.

75.2.2.3.1 IQualifiedName.Name Property

Specifies the local name.

Pascal

```
property Name: DOMString;
```

C++

```
__property DOMString Name;
```

Description

Name specifies the local name portion of the qualified name. This name is defined within the context of the namespace that the Namespace (see page 7255) property specifies.

See Also

Namespace (see page 7255)

75.2.2.3.2 IQualifiedName.Namespace Property

Specifies the namespace in which the name is defined.

Pascal

```
property Namespace: DOMString;
```

C++

```
__property DOMString Namespace;
```

Description

Namespace specifies the namespace that is used to interpret the value of the Name (see page 7255) property.

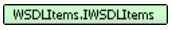
See Also

Name (see page 7255)

75.2.3 IWSDLItems Interface

IWSDLItems provides access to a WSDL document.

Class Hierarchy



Pascal

```
IWSDLItems = interface;
```

C++

```
__interface IWSDLItems;
```

File

WSDLItems

Description

IWSDLItems is an interface to the TWSDLItems (see page 7211) class. You can use this interface to load the WSDL document from a file or server, obtain access to the TWSDLItems (see page 7211) object whose methods provide information about the definitions in that document, and obtain an IWSDLDocument (see page 7164) interface for low-level access to the definitions in the document.

Note: C++ method declarations that use IWSDLItems use the _di_IWSDLItems type instead. This type is a DelphiInterface (see page 3256) wrapper around the IWSDLItems interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IWSDLDocument >  _di_IWSDLDocument;
```

See Also

TWSDLItems (see page 7211)

75.2.3.1 IWSDLItems Members

The following tables list the members exposed by IWSDLItems.

Public Methods

	Name	Description
	GetWSDLDoc (see page 7257)	Returns the IWSDLDocument (see page 7164) interface for the WSDL document.
	GetWSDLItems (see page 7257)	Returns a reference to the TWSDLItems (see page 7211) instance that implements this interface.
	Load (see page 7257)	Loads a WSDL document so that its definitions are available.

75.2.3.2 IWSDLItems Methods

The methods of the IWSDLItems class are listed here.

Public Methods

	Name	Description
	GetWSDLDoc (see page 7257)	Returns the IWSDLDocument (see page 7164) interface for the WSDL document.
	GetWSDLItems (see page 7257)	Returns a reference to the TWSDLItems (see page 7211) instance that implements this interface.
	Load (see page 7257)	Loads a WSDL document so that its definitions are available.

75.2.3.2.1 IWSDLItems.GetWSDLDoc Method

Returns the IWSDLDocument (see page 7164) interface for the WSDL document.

Pascal

```
function GetWSDLDoc: IWSDLDocument;
```

C++

```
__fastcall IWSDLDocument GetWSDLDoc();
```

Description

GetWSDLDoc returns a IWSDLDocument (see page 7164) interface, which supplies uninterpreted access to the definitions in the WSDL document. To access these definitions more easily, use the GetWSDLItems (see page 7257) method, which returns a reference to the underlying TWSDLItems (see page 7211) object.

See Also

GetWSDLItems (see page 7257)

75.2.3.2.2 IWSDLItems.GetWSDLItems Method

Returns a reference to the TWSDLItems (see page 7211) instance that implements this interface.

Pascal

```
function GetWSDLItems: TWSDLItems;
```

C++

```
__fastcall TWSDLItems GetWSDLItems();
```

Description

GetWSDLItems returns a reference to the underlying TWSDLItems (see page 7211) instance. This object has numerous methods that let you access the definitions in the WSDL document.

75.2.3.2.3 IWSDLItems.Load Method

Loads a WSDL document so that its definitions are available.

Pascal

```
procedure Load(const WSDLFileName: DOMString);
```

C++

```
__fastcall Load(const DOMString WSDLFileName);
```

Description

Load reads and interprets a specified WSDL document. This method should be called before you attempt to access the definitions in the WSDL document.

WSDLFileName is the name of a WSDL document file, an XML schema file, or the URL where such a document can be found.

75.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	HasDefinition (see page 7259)	This is function WSDLItems.HasDefinition.
	HasDefinition (see page 7259)	This is function WSDLItems.HasDefinition.
	NewQualifiedName (see page 7259)	This is function WSDLItems.NewQualifiedName.

75.3.1 WSDLItems.HasDefinition Function

Pascal

```
function HasDefinition(const WSDLDoc: IWSDLDocument): Boolean; overload;
```

C++

```
Boolean HasDefinition(const IWSDLDocument WSDLDoc);
```

File

WSDLItems

Description

This is function WSDLItems.HasDefinition.

75.3.2 WSDLItems.HasDefinition Function

Pascal

```
function HasDefinition(const WSDLItems: IWSDLItems): Boolean; overload;
```

C++

```
Boolean HasDefinition(const IWSDLItems WSDLItems);
```

File

WSDLItems

Description

This is function WSDLItems.HasDefinition.

75.3.3 WSDLItems.NewQualifiedName Function

Pascal

```
function NewQualifiedName(const Name: DOMString = ''; const Namespace: DOMString = ''):
```

```
  IQualifiedName;
```

C++

```
  IQualifiedName NewQualifiedName(const DOMString Name = '', const DOMString Namespace = '');
```

File

WSDLItems

Description

This is function WSDLItems.NewQualifiedName.

75.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	IterateImportOptions (see page 7261)	This is record WSDLItems.IterateImportOptions.

75.4.1 WSDLItems.IterateImportOptions Enumeration

Pascal

```
IterateImportOptions = (
  ioBeforeLoad,
  ioLoaded
);
```

C++

```
enum IterateImportOptions {
  ioBeforeLoad,
  ioLoaded
};
```

File

WSDLItems

Description

This is record WSDLItems.IterateImportOptions.

75.5 Types

The following table lists types in this documentation.

Types

Name	Description
IHeaderInfoArray (see page 7262)	This is type WSDLItems.IHeaderInfoArray.
IQualifiedNameArray (see page 7262)	IQualifiedNameArray is a dynamic array of interfaces for obtaining names of XML items.
TWSDLIterateProc (see page 7263)	TWSDLIterateProc is the type for callbacks that execute when importing a WSDL document.

75.5.1 WSDLItems.IHeaderInfoArray Type

Pascal

```
IHeaderInfoArray = array of IHeaderInfo;
```

C++

```
array of IHeaderInfo IHeaderInfoArray;
```

File

WSDLItems

Description

This is type WSDLItems.IHeaderInfoArray.

75.5.2 WSDLItems.IQualifiedNameArray Type

IQualifiedNameArray is a dynamic array of interfaces for obtaining names of XML items.

Pascal

```
IQualifiedNameArray = array of IQualifiedName;
```

C++

```
array of IQualifiedName IQualifiedNameArray;
```

File

WSDLItems

Description

IQualifiedNameArray is a dynamic array of IQualifiedName (see page 7252) interfaces. Each interface in the array can be used to access the name of an item from an XML document and the namespace in which that name is defined.

75.5.3 WSDLItems.TWSDLIterateProc Type

TWSDLIterateProc is the type for callbacks that execute when importing a WSDL document.

Pascal

```
TWSDLIterateProc = procedure (const Options: IterateImportOptions; const WSDLItems:
IWSDLItems; const XMLSchemaDoc: IXMLSchemaDoc; const Name: DOMString) of object;
```

C++

```
(const Options: IterateImportOptions; const WSDLItems: IWSDLItems; const XMLSchemaDoc:
IXMLSchemaDoc; const Name: DOMString) ( TWSDLIterateProc)();
```

File

WSDLItems

Description

TWSDLIterateProc is the type of a callback that executes for every WSDL document that is imported by being included in another..

Options indicates whether the callback is executing before or after the imported WSDL document is loaded.

WSDLItems (see page 7209) is the interface for the imported WSDL document. It is only available when Options is ioLoaded. If the imported document is an XML schema, it is nil (Delphi) or NULL (C++), and you should use XMLSchemaDoc instead.

XMLSchemaDoc is the interface for an imported XML schema when the imported document is an XML schema. It is only available when Options is ioLoaded. If the imported document is a WSDL document, it is nil (Delphi) or NULL (C++), and you should use WSDLItems (see page 7209) instead.

Name is the name (URL) of the imported document.

76 WSDLNode Namespace

This is namespace WSDLNode.

Topics

Name	Description
Classes (see page 7266)	The following table lists classes in this documentation.
Functions (see page 7277)	The following table lists functions in this documentation.

Classes

	Name	Description
	EWSDLLoadException (see page 7266)	This is class WSDLNode.EWSDLLoadException.
	TWSDLClientNode (see page 7266)	This is class WSDLNode.TWSDLClientNode.
	TWSDLView (see page 7271)	TWSDLView represents a specific binding and operation in a WSDL (see page 7276) document.

Functions

	Name	Description
	ActivateWSDL (see page 7277)	This is function WSDLNode.ActivateWSDL.

76.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EWSDLLoadException (see page 7266)	This is class WSDLNode.EWSDLLoadException.
	TWSDLClientNode (see page 7266)	This is class WSDLNode.TWSDLClientNode.
	TWSDLView (see page 7271)	TWSDLView represents a specific binding and operation in a WSDL (see page 7276) document.

76.1.1 EWSDLLoadException Class

Class Hierarchy

Pascal

```
EWSDLLoadException = class(Exception);
```

C++

```
class EWSDLLoadException : public Exception;
```

File

WSDLNode

Description

This is class WSDLNode.EWSDLLoadException.

76.1.2 TWSDLClientNode Class

Class Hierarchy

Pascal

```
TWSDLClientNode = class(TComponent, IWebNode);
```

C++

```
class TWSDLClientNode : public TComponent, public IWebNode;
```

File

WSDLNode

Description

This is class WSDLNode.TWSDLClientNode.

76.1.2.1 TWSDLClientNode Members

The following tables list the members exposed by TWSDLClientNode.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 7269)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 7270)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	BeforeExecute (see page 7267)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 6618)	This is the overview for the Execute method overload.
	Execute (see page 7268)	This is the overview for the Execute method overload.
	Notification (see page 7269)	Forwards notification messages to all owned components.

Public Properties

	Name	Description
	MimeBoundary (see page 6619)	Specifies the value of the MIME boundary header, if any, for the section of the document that contains the method call.

Published Properties

	Name	Description
	TransportNode (see page 7270)	This is TransportNode, a member of class TWSDLClientNode.
	WSDLView (see page 7271)	This is WSDLView, a member of class TWSDLClientNode.

76.1.2.2 TWSDLClientNode Methods

The methods of the TWSDLClientNode class are listed here.

Protected Methods

	Name	Description
	GetMimeBoundary (see page 7269)	Returns the value of the MimeBoundary property.
	SetMimeBoundary (see page 7270)	Sets the value of the MimeBoundary property.

Public Methods

	Name	Description
	BeforeExecute (see page 7267)	Performs any transport-specific setup before executing an encoded method call.
	Execute (see page 7268)	This is the overview for the Execute method overload.
	Notification (see page 7269)	Forwards notification messages to all owned components.

76.1.2.2.1 TWSDLClientNode.BeforeExecute Method

Performs any transport-specific setup before executing an encoded method call.

Pascal

```
procedure BeforeExecute(const IntfMetaData: TIntfMetaData; const MethodMetaData:
TIntfMethEntry; MethodIndex: Integer; AttachHandler: IMimeAttachmentHandler);
```

C++

```
__fastcall BeforeExecute(const TIntfMetaData IntfMetaData, const TIntfMethEntry
MethodMetaData, int MethodIndex, IMimeAttachmentHandler AttachHandler);
```

Description

BeforeExecute handles transport-specific tasks such as dynamically querying a WSDL document or setting up serialization options before a call is made to the Execute method. Remote interfaced objects (TRio) call this method after their OnBeforeExecute event, but before delegating the execution of the method to the Execute method.

IntfMD is the runtime type information (RTTI) of the invokable interface that includes the method to execute.

MethMD is describes the method that is about to be executed.

MethodIndex indicates which overload to use when methMD describes an overloaded method. 0 indicates the first overload, 1 indicates the second overload, and so on.

AttachHandler is an interface to use for handling any attachments that are used as parameters of the method.

See Also

Execute (see page 6618)

OnBeforeExecute

76.1.2.2.2 Execute Method

This is the overview for the Execute method overload.

Overload List

	Name	Description
	TWSDLClientNode.Execute (String, TStream) (see page 7268)	Executes an encoded method call and returns the encoded results in a stream.
	TWSDLClientNode.Execute (TStream) (see page 7269)	This is Execute, a member of class TWSDLClientNode.
	TWSDLClientNode.Execute (TStream, TStream) (see page 7269)	This is Execute, a member of class TWSDLClientNode.

76.1.2.2.2.1 TWSDLClientNode.Execute Method (String, TStream)

Executes an encoded method call and returns the encoded results in a stream.

Pascal

```
procedure Execute(const DataMsg: String; Resp: TStream); overload;
```

C++

```
__fastcall Execute(const UnicodeString DataMsg, TStream Resp);
```

Description

Execute interprets an encoded method call, executes it, and returns the encoded results using a stream. Typically, this method works in conjunction with an object that implements the IOPConvert interface, which marshals a method call into the DataMsg or Request string and unmarshals the return value.

DataMsg is an encoded method call. It represents the results of marshaling a method call on an invokable interface into a transportable string. The use of this syntax is not recommended.

Request is a stream from which to read the same string that is the value of the DataMsg parameter. This syntax is preferred because the stream need not store this value in memory (for example, it can be a file stream).

Response is a stream object that receives the encoded results that arise from executing the method call.

When using the third syntax, Execute returns a stream from which to read the encoded results from executing the method call. In this case, the caller is responsible for freeing the stream.

See Also

IOPConvert

BeforeExecute (see page 6617)

76.1.2.2.2.2 TWSDLClientNode.Execute Method (TStream)

Pascal

```
function Execute(const Request: TStream): TStream; overload;
```

C++

```
__fastcall TStream Execute(const TStream Request);
```

Description

This is Execute, a member of class TWSDLClientNode.

76.1.2.2.2.3 TWSDLClientNode.Execute Method (TStream, TStream)

Pascal

```
procedure Execute(const Request: TStream; Response: TStream); overload;
```

C++

```
__fastcall Execute(const TStream Request, TStream Response);
```

Description

This is Execute, a member of class TWSDLClientNode.

76.1.2.2.3 TWSDLClientNode.GetMimeBoundary Method

Returns the value of the MimeBoundary property.

Pascal

```
function GetMimeBoundary: string;
```

C++

```
__fastcall UnicodeString GetMimeBoundary();
```

Description

GetMimeBoundary is the read access method for the MimeBoundary property.

See Also

MimeBoundary (see page 6619)

76.1.2.2.4 TWSDLClientNode.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

76.1.2.2.5 TWSDLClientNode.SetMimeBoundary Method

Sets the value of the MimeBoundary property.

Pascal

```
procedure SetMimeBoundary(Value: string);
```

C++

```
__fastcall SetMimeBoundary(UnicodeString Value);
```

Description

SetMimeBoundary is the write implementation for the MimeBoundary property.

Value is the new value for MimeBoundary.

See Also

MimeBoundary (see page 6619)

76.1.2.3 TWSDLClientNode Properties

The properties of the TWSDLClientNode class are listed here.

Published Properties

	Name	Description
	TransportNode (see page 7270)	This is TransportNode, a member of class TWSDLClientNode.
	WSDLView (see page 7271)	This is WSDLView, a member of class TWSDLClientNode.

76.1.2.3.1 TWSDLClientNode.TransportNode Property

Pascal

```
property TransportNode: IWebNode;
```

C++

```
__property IWebNode TransportNode;
```

Description

This is TransportNode, a member of class TWSDLClientNode.

76.1.2.3.2 TWSDLClientNode.WSDLView Property

Pascal

```
property WSDLView: TWSDLView;
```

C++

```
__property TWSDLView WSDLView;
```

Description

This is WSDLView, a member of class TWSDLClientNode.

76.1.3 TWSDLView Class

TWSDLView represents a specific binding and operation in a WSDL (see page 7276) document.

Class Hierarchy

Pascal

```
TWSDLView = class(TComponent);
```

C++

```
class TWSDLView : public TComponent;
```

File

WSDLNode

Description

TWSDLView is used by components that work with WSDL (see page 7276) documents to represent a specific port and/or operation that is defined in a WSDL (see page 7276) document. The WSDL (see page 7276) property provides access to the WSDL (see page 7276) document itself. The other properties give the names of the entities of interest that are defined within that document.

See Also

THTTPReqResp (see page 1276)

TOPToSoapDomConvert

TWSDLItems (see page 7211)

76.1.3.1 TWSDLView Members

The following tables list the members exposed by TWSDLView.

Public Fields

	Name	Description
	IntfInfo (see page 7272)	This is IntfInfo, a member of class TWSDLView.

Public Methods

	Name	Description
	Activate (see page 7273)	This is Activate, a member of class TWSDLView.
	SetDesignState (see page 7273)	This is SetDesignState, a member of class TWSDLView.

Published Properties

	Name	Description
	Operation (see page 7273)	Identifies the target operation that is defined in the WSDL (see page 7276) document.
	Password (see page 7274)	This is Password, a member of class TWSDLView.
	Port (see page 7274)	Specifies the target port that is defined in the WSDL (see page 7276) document.
	PortType (see page 7274)	Specifies the target port type that is defined in the WSDL (see page 7276) document.
	Proxy (see page 7275)	This is Proxy, a member of class TWSDLView.
	Service (see page 7275)	Specifies the target service that is defined in the WSDL (see page 7276) document.
	UserName (see page 7275)	This is UserName, a member of class TWSDLView.
	WSDL (see page 7276)	Provides access to the definitions in the WSDL document.

76.1.3.2 TWSDLView Fields

The fields of the TWSDLView class are listed here.

Public Fields

	Name	Description
	IntfInfo (see page 7272)	This is IntfInfo, a member of class TWSDLView.

76.1.3.2.1 TWSDLView.IntfInfo Field

Pascal

```
IntfInfo: PTypeInfo;
```

C++

```
PTypeInfo IntfInfo;
```

Description

This is IntfInfo, a member of class TWSDLView.

76.1.3.3 TWSDLView Methods

The methods of the TWSDLView class are listed here.

Public Methods

	Name	Description
	Activate (see page 7273)	This is Activate, a member of class TWSDLView.
	SetDesignState (see page 7273)	This is SetDesignState, a member of class TWSDLView.

76.1.3.3.1 TWSDLView.Activate Method

Pascal

```
procedure Activate;
```

C++

```
__fastcall Activate();
```

Description

This is Activate, a member of class TWSDLView.

76.1.3.3.2 TWSDLView.SetDesignState Method

Pascal

```
procedure SetDesignState(Designing: Boolean);
```

C++

```
__fastcall SetDesignState(Boolean Designing);
```

Description

This is SetDesignState, a member of class TWSDLView.

76.1.3.4 TWSDLView Properties

The properties of the TWSDLView class are listed here.

Published Properties

	Name	Description
	Operation (see page 7273)	Identifies the target operation that is defined in the WSDL (see page 7276) document.
	Password (see page 7274)	This is Password, a member of class TWSDLView.
	Port (see page 7274)	Specifies the target port that is defined in the WSDL (see page 7276) document.
	PortType (see page 7274)	Specifies the target port type that is defined in the WSDL (see page 7276) document.
	Proxy (see page 7275)	This is Proxy, a member of class TWSDLView.
	Service (see page 7275)	Specifies the target service that is defined in the WSDL (see page 7276) document.
	UserName (see page 7275)	This is UserName, a member of class TWSDLView.
	WSDL (see page 7276)	Provides access to the definitions in the WSDL document.

76.1.3.4.1 TWSDLView.Operation Property

Identifies the target operation that is defined in the WSDL (see page 7276) document.

Pascal

```
property Operation: string;
```

C++

```
__property UnicodeString Operation;
```

Description

Operation is the name of an operation that is defined in the WSDL (see page 7276) document identified by the WSDL (see

page 7276) property. Operation maps to a method on the interface that corresponds to the PortType (see page 7274) property.

See Also

PortType (see page 7274)

WSDL (see page 7276)

76.1.3.4.2 TWSDLView.Password Property

Pascal

```
property Password: string;
```

C++

```
__property UnicodeString Password;
```

Description

This is Password, a member of class TWSDLView.

76.1.3.4.3 TWSDLView.Port Property

Specifies the target port that is defined in the WSDL (see page 7276) document.

Pascal

```
property Port: string;
```

C++

```
__property UnicodeString Port;
```

Description

Port is the name of a port that is defined in the WSDL (see page 7276) document identified by the WSDL (see page 7276) property. A Port represents the address of a Web Service (see page 7275) application, a transport protocol that is used for requests to that application, the data format to use for encoding requests, and the port type that defines the operations contained in those requests.

See Also

PortType (see page 7274)

WSDL (see page 7276)

Service (see page 7275)

76.1.3.4.4 TWSDLView.PortType Property

Specifies the target port type that is defined in the WSDL (see page 7276) document.

Pascal

```
property PortType: string;
```

C++

```
__property UnicodeString PortType;
```

Description

Port (see page 7274) is the name of a port type that is defined in the WSDL (see page 7276) document identified by the WSDL (see page 7276) property. A port type corresponds to an interface that the Web Service (see page 7275) application

implements.

The target method on that interface is identified by the Operation (see page 7273) property. The address and protocols to use when calling this interface are identified by the Port (see page 7274) property.

See Also

WSDL (see page 7276)

Port (see page 7274)

Operation (see page 7273)

76.1.3.4.5 TWSDLView.Proxy Property

Pascal

```
property Proxy: string;
```

C++

```
__property UnicodeString Proxy;
```

Description

This is Proxy, a member of class TWSDLView.

76.1.3.4.6 TWSDLView.Service Property

Specifies the target service that is defined in the WSDL (see page 7276) document.

Pascal

```
property Service: string;
```

C++

```
__property UnicodeString Service;
```

Description

Service is the name of the service of interest that is defined in the WSDL (see page 7276) document identified by the WSDL (see page 7276) property. A service is a collection of related ports, including the one specified by the Port (see page 7274) property.

See Also

Port (see page 7274)

WSDL (see page 7276)

76.1.3.4.7 TWSDLView.UserName Property

Pascal

```
property UserName: string;
```

C++

```
__property UnicodeString UserName;
```

Description

This is UserName, a member of class TWSDLView.

76.1.3.4.8 TWSDLView.WSDL Property

Provides access to the definitions in the WSDL document.

Pascal

```
property WSDL: TWSDLItems;
```

C++

```
__property TWSDLItems WSDL;
```

Description

WSDLItems (see page 7209) is a helper object whose methods return the definitions included in the WSDL document. All the other TWSDLView (see page 7271) properties refer to definitions contained in this WSDL document

76.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	ActivateWSDL (see page 7277)	This is function WSDLNode.ActivateWSDL.

76.2.1 WSDLNode.ActivateWSDL Function

Pascal

```
function ActivateWSDL(WSDL: TWSDLItems; const Name: string; const Password: string; const
Proxy: string): Boolean;
```

C++

```
Boolean ActivateWSDL(TWSDLItems WSDL, const UnicodeString Name, const UnicodeString Password,
const UnicodeString Proxy);
```

File

WSDLNode

Description

This is function WSDLNode.ActivateWSDL.

77 WSDLPub Namespace

This is namespace WSDLPub.

Topics

Name	Description
Classes (see page 7280)	The following table lists classes in this documentation.
Interfaces (see page 7295)	The following table lists interfaces in this documentation.
Functions (see page 7297)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7298)	The following table lists structs, records, enums in this documentation.
Types (see page 7299)	The following table lists types in this documentation.

Classes

	Name	Description
	TWSDLHTMLPublish (see page 7280)	TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.
	TWSDLPublish (see page 7291)	This is class WSDLPub.TWSDLPublish.

Functions

	Name	Description
	WSDLPubFactory (see page 7297)	This is function WSDLPub.WSDLPubFactory.

Interfaces

	Name	Description
	IWSDLPublish (see page 7295)	This is class WSDLPub.IWSDLPublish.

Structs, Records, Enums

	Name	Description
	SOAPPublishOptions (see page 7298)	SOAPPublishOptions and TSOAPPublishOptions (see page 7299) are flags that customize the definitions the TWSDLHTMLPublish (see page 7280) creates for a Web Service.

Types

Name	Description
TBeforePublishingWSDLEvent (see page 7299)	TBeforePublishingWSDLEvent is the type of the OnBeforePublishingWSDL event of TWSDLHTMLPublish (see page 7280).
TSOAPPublishOptions (see page 7299)	This is type WSDLPub.TSOAPPublishOptions.

77.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TWSDLHTMLPublish (see page 7280)	TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.
	TWSDLPublish (see page 7291)	This is class WSDLPub.TWSDLPublish.

77.1.1 TWSDLHTMLPublish Class

TWSDLHTMLPublish publishes a list of WSDL documents describing a Web Services application.

Class Hierarchy

Pascal

```
TWSDLHTMLPublish = class(TComponent, IWebDispatch);
```

C++

```
class TWSDLHTMLPublish : public TComponent, public IWebDispatch;
```

File

WSDLPub

Description

Add TWSDLHTMLPublish to the Web module of a Web Service application to publish a list of WSDL documents that describe how to call that Web Service application. By publishing WSDL documents for your Web Service application, you allow arbitrary clients to call on your Web Services.

TWSDLHTMLPublish publishes definitions it derives from all the invokable interfaces that are registered with the invocation registry (InvRegistry), as well as from all remotable classes and types that are registered with the remotable type registry (RemTypeRegistry). By default, it uses the address where you install the application that contains the TWSDLHTMLPublish component as the address where those services are available. If you deploy the implementation of those Web Services elsewhere, you can enable the WSDL administrator (using the AdminEnabled (see page 7287) property), and then launch the WSDL administrator from a Web browser to change the addresses specified in the generated WSDL documents.

TWSDLHTMLPublish automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the WSDL publisher without requiring the use of Web action items. Those messages include requests for the list of WSDL documents and requests for the WSDL administrator.

See Also

TWebModule

InvRegistry

RemTypeRegistry

Generating WSDL Documents for a Web Service Application

Using the SOAP Application Wizard

Writing Servers that Support Web Services

77.1.1.1 TWSDLHTMLPublish Members

The following tables list the members exposed by TWSDLHTMLPublish.

Protected Methods

	Name	Description
	AddInterfaceList (see page 7283)	This is AddInterfaceList, a member of class TWSDLHTMLPublish.
	AddPortList (see page 7283)	This is AddPortList, a member of class TWSDLHTMLPublish.
	GetHostScriptBaseURL (see page 7285)	This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.
	UpdatePortList (see page 7286)	This is UpdatePortList, a member of class TWSDLHTMLPublish.
	WSILInfo (see page 7286)	Returns the WS inspection document that describes the Web Service.

Public Methods

	Name	Description
	Create (see page 7281)	Creates an instance of TWSDLHTMLPublish (see page 7280).
	Destroy (see page 7282)	Destroys the instance of a TWSDLHTMLPublish (see page 7280) object.
	DispatchEnabled (see page 7283)	Indicates whether the WSDL publisher is enabled to respond to incoming messages.
	DispatchMask (see page 7283)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 7284)	Returns the method type of all HTTP messages to which this component responds.
	DispatchRequest (see page 7284)	Responds to an incoming message and generates the appropriate response message.
	GetTargetNamespace (see page 7285)	Returns the value of the TargetNamespace (see page 7288) property.
	HandleRequest (see page 7285)	Generates the response for an incoming request message.
	ServiceInfo (see page 7286)	Responds to requests that are otherwise not handled by the application.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 7289)	Occurs after the publisher generates a WSDL document for the Web Service application.
	OnBeforePublishingTypes (see page 7290)	Occurs before the publisher starts adding type definitions to the WSDL document.
	OnBeforePublishingWSDL (see page 7290)	Occurs before the publisher generates its WSDL document.
	OnPublishingType (see page 7290)	Occurs before the publisher generates the schema for a given type.

Published Properties

	Name	Description
	AdminEnabled (see page 7287)	Enables or disables the WSDL administrator.
	PublishOptions (see page 7288)	Controls the definitions TWSDLHTMLPublish (see page 7280) creates when it generates its WSDL documents.
	TargetNamespace (see page 7288)	Specifies the target namespace URL of the WSDL document.
	WebDispatch (see page 7288)	Lets the Web dispatcher identify what HTTP messages should return a list of WSDL documents.

77.1.1.2 TWSDLHTMLPublish.Create Constructor

Creates an instance of TWSDLHTMLPublish (see page 7280).

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TWSDLHTMLPublish(TComponent * AOwner);
```

Description

Call Create to instantiate a TWSDLHTMLPublish (see page 7280) object at runtime. TWSDLHTMLPublish (see page 7280) components added to web modules at design time are created automatically.

The AOwner parameter specifies another component (usually the web module) that is responsible for freeing the TWSDLHTMLPublish (see page 7280) instance. It becomes the value of the Owner property.

See Also

Owner

77.1.1.3 TWSDLHTMLPublish.Destroy Destructor

Destroys the instance of a TWSDLHTMLPublish (see page 7280) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TWSDLHTMLPublish();
```

Description

Use the Free method to free a TWSDLHTMLPublish (see page 7280) instance. Unlike the Destroy method, Free guards against a nil reference.

See Also

System::TObject::Free (see page 3372)

77.1.1.4 TWSDLHTMLPublish Methods

The methods of the TWSDLHTMLPublish class are listed here.

Protected Methods

	Name	Description
	AddInterfaceList (see page 7283)	This is AddInterfaceList, a member of class TWSDLHTMLPublish.
	AddPortList (see page 7283)	This is AddPortList, a member of class TWSDLHTMLPublish.
	GetHostScriptBaseURL (see page 7285)	This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.
	UpdatePortList (see page 7286)	This is UpdatePortList, a member of class TWSDLHTMLPublish.
	WSILInfo (see page 7286)	Returns the WS inspection document that describes the Web Service.

Public Methods

	Name	Description
	DispatchEnabled (see page 7283)	Indicates whether the WSDL publisher is enabled to respond to incoming messages.
	DispatchMask (see page 7283)	Returns a mask that matches the path portion of any URL to which this object responds.
	DispatchMethodType (see page 7284)	Returns the method type of all HTTP messages to which this component responds.

	DispatchRequest (see page 7284)	Responds to an incoming message and generates the appropriate response message.
	GetTargetNamespace (see page 7285)	Returns the value of the TargetNamespace (see page 7288) property.
	HandleRequest (see page 7285)	Generates the response for an incoming request message.
	ServiceInfo (see page 7286)	Responds to requests that are otherwise not handled by the application.

77.1.1.4.1 TWSDLHTMLPublish.AddInterfaceList Method

Pascal

```
procedure AddInterfaceList(htmldoc: TStringList; const WSDLBaseURL: String);
```

C++

```
__fastcall AddInterfaceList(TStringList htmldoc, const UnicodeString WSDLBaseURL);
```

Description

This is AddInterfaceList, a member of class TWSDLHTMLPublish.

77.1.1.4.2 TWSDLHTMLPublish.AddPortList Method

Pascal

```
procedure AddPortList(htmldoc: TStringList; const PortType: String);
```

C++

```
__fastcall AddPortList(TStringList htmldoc, const UnicodeString PortType);
```

Description

This is AddPortList, a member of class TWSDLHTMLPublish.

77.1.1.4.3 TWSDLHTMLPublish.DispatchEnabled Method

Indicates whether the WSDL publisher is enabled to respond to incoming messages.

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

Description

The DispatchEnabled method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to determine whether it can receive incoming messages. In TWSDLHTMLPublish (see page 7280), DispatchEnabled always returns true.

See Also

WebDispatch (see page 7288)

77.1.1.4.4 TWSDLHTMLPublish.DispatchMask Method

Returns a mask that matches the path portion of any URL to which this object responds.

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

Description

The DispatchMask method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to the this component. It returns a Mask derived from the Mask property of the WebDispatch (see page 7288) property.

See Also

WebDispatch (see page 7288)

Mask

77.1.1.4.5 TWSDLHTMLPublish.DispatchMethodType Method

Returns the method type of all HTTP messages to which this component responds.

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

Description

The DispatchMethodType method must be implemented by all auto-dispatching objects to allow Web Module (or Web Dispatcher) to identify the HTTP messages it should forward to this component. It returns the MethodType properties of WebDispatch (see page 7288).

See Also

MethodType

WebDispatch (see page 7288)

77.1.1.4.6 TWSDLHTMLPublish.DispatchRequest Method

Responds to an incoming message and generates the appropriate response message.

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

The Web Module (or Web Dispatcher) calls DispatchRequest when it identifies an HTTP message that matches the properties of the WebDispatch (see page 7288) property. DispatchRequest responds by generating an HTML page that lists the WSDL documents describing the Web Service, and including a button for the WSDL administrator if the AdminEnabled (see page 7287) property is true.

Sender is not used in the DispatchRequest method.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

DispatchRequest returns true if it successfully generates a response. It returns false if it could not handle the request.

See Also

WebDispatch (see page 7288)

AdminEnabled (see page 7287)

77.1.1.4.7 TWSDLHTMLPublish.GetHostScriptBaseURL Method

Pascal

```
function GetHostScriptBaseURL(Request: TWebRequest): String;
```

C++

```
__fastcall UnicodeString GetHostScriptBaseURL(TWebRequest Request);
```

Description

This is GetHostScriptBaseURL, a member of class TWSDLHTMLPublish.

77.1.1.4.8 TWSDLHTMLPublish.GetTargetNamespace Method

Returns the value of the TargetNamespace (see page 7288) property.

Pascal

```
function GetTargetNamespace: InvString;
```

C++

```
__fastcall InvString GetTargetNamespace();
```

Description

GetTargetNamespace is the read implementation for the TargetNamespace (see page 7288) property.

See Also

TargetNamespace (see page 7288)

77.1.1.4.9 TWSDLHTMLPublish.HandleRequest Method

Generates the response for an incoming request message.

Pascal

```
function HandleRequest(Resp: TStringList; const Path: String; const HostScriptBaseURL: String;
var ContentType: String; Request: TWebRequest): integer;
```

C++

```
__fastcall int HandleRequest(TStringList Resp, const UnicodeString Path, const UnicodeString
HostScriptBaseURL, UnicodeString ContentType, TWebRequest Request);
```

Description

The DispatchRequest (see page 7284) method calls HandleRequest to generate the content of the HTTP response message it fills out. Typically, applications should not need to call this method.

Resp is a string list to which HandleRequest writes the content of the response.

Path is the path information taken from the incoming request message.

HostScriptBaseURL is the URL for the script name to which the incoming request is addressed.

ContentType returns the content type of the response that is written to Resp. This becomes the value of a ContentType header on the response message.

Request is the request message for which this method generates a response.

HandleRequest returns the StatusCode for the generated response. This becomes the StatusCode property of the response message.

See Also

DispatchRequest (see page 7284)

StatusCode

77.1.1.4.10 TWSDLHTMLPublish.ServiceInfo Method

Responds to requests that are otherwise not handled by the application.

Pascal

```
procedure ServiceInfo(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var
Handled: Boolean);
```

C++

```
__fastcall ServiceInfo(TObject * Sender, TWebRequest Request, TWebResponse Response, Boolean
Handled);
```

Description

The Web Module (or Web Dispatcher) calls ServiceInfo when there are no other components in the application to handle an incoming request message. ServiceInfo responds by generating a default Web page that includes a link to the WSDL documents.

Sender is not used in the ServiceInfo method.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

Handled always returns true, indicating that ServiceInfo successfully generated a response.

See Also

DispatchRequest (see page 7284)

77.1.1.4.11 TWSDLHTMLPublish.UpdatePortList Method

Pascal

```
procedure UpdatePortList(PortList: TStrings; const PortType: String; const Command: String);
```

C++

```
__fastcall UpdatePortList(TStrings PortList, const UnicodeString PortType, const UnicodeString
Command);
```

Description

This is UpdatePortList, a member of class TWSDLHTMLPublish.

77.1.1.4.12 TWSDLHTMLPublish.WSILInfo Method

Returns the WS inspection document that describes the Web Service.

Pascal

```
procedure WSILInfo(const HostScriptBaseURL: string; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean);
```

C++

```
__fastcall WSILInfo(const UnicodeString HostScriptBaseURL, TWebRequest Request, TWebResponse
Response, Boolean Handled);
```

Description

The Web Module (or Web Dispatcher) calls WSILInfo when a request is made for the WS inspection document for the Web Service (when the path portion of the request ends in the string 'inspection.wisl'.. WSILInfo responds by generating the WSIL description.

HostScriptBaseURL is the base path for accessing the WSDL documents of the Web Service.

Request contains information about the incoming request message.

Response is a TWebResponse object that is filled out to indicate the response that should be returned to the Web client.

Handled always returns true, indicating that WSILInfo successfully generated a response.

See Also

DispatchRequest (see page 7284)

ServiceInfo (see page 7286)

77.1.1.5 TWSDLHTMLPublish Properties

The properties of the TWSDLHTMLPublish class are listed here.

Published Properties

	Name	Description
	AdminEnabled (see page 7287)	Enables or disables the WSDL administrator.
	PublishOptions (see page 7288)	Controls the definitions TWSDLHTMLPublish (see page 7280) creates when it generates its WSDL documents.
	TargetNamespace (see page 7288)	Specifies the target namespace URL of the WSDL document.
	WebDispatch (see page 7288)	Lets the Web dispatcher identify what HTTP messages should return a list of WSDL documents.

77.1.1.5.1 TWSDLHTMLPublish.AdminEnabled Property

Enables or disables the WSDL administrator.

Pascal

```
property AdminEnabled: Boolean;
```

C++

```
__property Boolean AdminEnabled;
```

Description

Use AdminEnabled to indicate whether this application implements the WSDL administrator. When AdminEnabled is true, pointing a Web browser at the URL that addresses this application using the path info specified by WebDispatch (see page 7288) launches a Web page that includes a button for the WSDL administrator. The WSDL administrator can then adjust the published WSDL documents to indicate an address other than the one where this application is installed as the location or locations where the Web Services are deployed.

When AdminEnabled is false, the URL that includes WebDispatch (see page 7288) publishes a list of WSDL documents but does not provide access to the WSDL administrator.

Tip: You can set AdminEnabled to true, update the URLs in the resulting WSDL documents, and then rebuild your application, setting AdminEnabled to false. The WSDL administrator stores your changes in a .ini file so that they appear in the list of WSDL documents even after you have changed AdminEnabled.

See Also

WebDispatch (see page 7288)

TargetNamespace (see page 7288)

Generating WSDL Documents for a Web Service Application

77.1.1.5.2 TWSDLHTMLPublish.PublishOptions Property

Controls the definitions TWSDLHTMLPublish (see page 7280) creates when it generates its WSDL documents.

Pascal

```
property PublishOptions: TSOAPPublishOptions;
```

C++

```
__property TSOAPPublishOptions PublishOptions;
```

Description

PublishOptions is a set of flags that customize the generated WSDL document.

See Also

TargetNamespace (see page 7288)

77.1.1.5.3 TWSDLHTMLPublish.TargetNamespace Property

Specifies the target namespace URL of the WSDL document.

Pascal

```
property TargetNamespace: InvString;
```

C++

```
__property InvString TargetNamespace;
```

Description

Use TargetNamespace to specify a namespace for the WSDL documents this component publishes. This is the namespace URL that the WSDL document uses for the definitions it publishes.

See Also

AdminEnabled (see page 7287)

77.1.1.5.4 TWSDLHTMLPublish.WebDispatch Property

Lets the Web dispatcher identify what HTTP messages should return a list of WSDL documents.

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

Use WebDispatch to specify the messages to which TWSDLHTMLPublish (see page 7280) responds by providing a list of WSDL documents. These documents describe the Web Services that this application knows about, based on the registered interfaces, remotable classes, and remotable types. If this application registers but does not implement the invokable interface, you should also enable the WSDL administrator and use the WSDL administrator to indicate where the application that implements the Web Services is deployed.

The TWebDispatch object that implements WebDispatch includes properties that let you specify the path portion of URLs to which this component responds with an HTML page that publishes WSDL documents describing the Web Service. It also specifies the values allowed in the MESSAGE header. By default, WebDispatch identifies any URL whose path portion begins with the string “wsdl”.

Note: You can't use WebDispatch to disable the TWSDLHTMLPublish (see page 7280) object's ability to publish WSDL documents describing the Web Services. Setting its Enabled property to false does nothing.

See Also

AdminEnabled (see page 7287)

InvRegistry

RemTypeRegistry

77.1.1.6 TWSDLHTMLPublish Events

The events of the TWSDLHTMLPublish class are listed here.

Published Events

	Name	Description
	OnAfterPublishingWSDL (see page 7289)	Occurs after the publisher generates a WSDL document for the Web Service application.
	OnBeforePublishingTypes (see page 7290)	Occurs before the publisher starts adding type definitions to the WSDL document.
	OnBeforePublishingWSDL (see page 7290)	Occurs before the publisher generates its WSDL document.
	OnPublishingType (see page 7290)	Occurs before the publisher generates the schema for a given type.

77.1.1.6.1 TWSDLHTMLPublish.OnAfterPublishingWSDL Event

Occurs after the publisher generates a WSDL document for the Web Service application.

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

Use OnAfterPublishingWSDL to perform special actions after the publisher has generated a WSDL document. For example, you can use this event to cache the generated WSDL document and then re-use the cached document in an OnBeforePublishingWSDL (see page 7290) event rather than requiring the publisher to generate a new document for every WSDL request it receives.

See Also

OnBeforePublishingWSDL (see page 7290)

77.1.1.6.2 TWSDLHTMLPublish.OnBeforePublishingTypes Event

Occurs before the publisher starts adding type definitions to the WSDL document.

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

Use OnBeforePublishingTypes to add your own type definitions to the WSDL document before the publisher adds definitions for the types that the invokable interfaces use and types that are registered with the remotable type registry. If the OnBeforePublishingTypes event handler adds a schema definition for a type, the publisher uses that definition rather than generating its own version.

See Also

OnBeforePublishingWSDL (see page 7290)

OnPublishingType (see page 7290)

77.1.1.6.3 TWSDLHTMLPublish.OnBeforePublishingWSDL Event

Occurs before the publisher generates its WSDL document.

Pascal

```
property OnBeforePublishingWSDL: TBeforePublishingWSDLEvent;
```

C++

```
__property TBeforePublishingWSDLEvent OnBeforePublishingWSDL;
```

Description

Use OnBeforePublishingWSDL to provide your own WSDL document rather than letting the publisher generate one dynamically. For example, the event handler can supply a cached version of the WSDL document that you saved previously from an OnAfterPublishingWSDL (see page 7289) event handler.

See Also

OnAfterPublishingWSDL (see page 7289)

OnBeforePublishingTypes (see page 7290)

77.1.1.6.4 TWSDLHTMLPublish.OnPublishingType Event

Occurs before the publisher generates the schema for a given type.

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

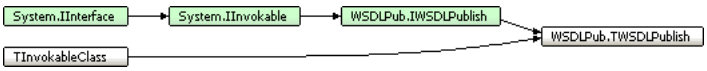
Use OnPublishingType to perform specific actions immediately before the publisher adds a type definition to the emerging WSDL document.

See Also

OnBeforePublishingTypes (see page 7290)

77.1.2 TWSDLPublish Class

Class Hierarchy



Pascal

```
TWSDLPublish = class(TInvokableClass, IWSDLPublish);
```

C++

```
class TWSDLPublish : public TInvokableClass, public IWSDLPublish;
```

File

WSDLPub

Description

This is class WSDLPub.TWSDLPublish.

77.1.2.1 TWSDLPublish Members

The following tables list the members exposed by TWSDLPublish.

Public Events

	Name	Description
	OnAfterPublishingWSDL (see page 7294)	This is OnAfterPublishingWSDL, a member of class TWSDLPublish.
	OnBeforePublishingTypes (see page 7294)	This is OnBeforePublishingTypes, a member of class TWSDLPublish.
	OnBeforePublishingWSDL (see page 7294)	This is OnBeforePublishingWSDL, a member of class TWSDLPublish.
	OnPublishingType (see page 7294)	This is OnPublishingType, a member of class TWSDLPublish.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	GetPortTypeEntries (see page 7292)	This is GetPortTypeEntries, a member of class TWSDLPublish.
	GetPortTypeList (see page 7292)	This is GetPortTypeList, a member of class TWSDLPublish.
	GetTypeSystemsList (see page 7292)	This is GetTypeSystemsList, a member of class TWSDLPublish.
	GetWSDLForPortType (see page 7293)	This is GetWSDLForPortType, a member of class TWSDLPublish.
	GetXSDForTypeSystem (see page 7293)	This is GetXSDForTypeSystem, a member of class TWSDLPublish.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

Public Properties

	Name	Description
	TargetNamespace (see page 7293)	This is TargetNamespace, a member of class TWSDLPublish.

77.1.2.2 TWSDLPublish Methods

The methods of the TWSDLPublish class are listed here.

Public Methods

	Name	Description
	GetPortTypeEntries (see page 7292)	This is GetPortTypeEntries, a member of class TWSDLPublish.
	GetPortTypeList (see page 7292)	This is GetPortTypeList, a member of class TWSDLPublish.
	GetTypeSystemsList (see page 7292)	This is GetTypeSystemsList, a member of class TWSDLPublish.
	GetWSDLForPortType (see page 7293)	This is GetWSDLForPortType, a member of class TWSDLPublish.
	GetXSDForTypeSystem (see page 7293)	This is GetXSDForTypeSystem, a member of class TWSDLPublish.

77.1.2.2.1 TWSDLPublish.GetPortTypeEntries Method

Pascal

```
procedure GetPortTypeEntries(var Entries: TInvRegIntfEntryArray);
```

C++

```
__fastcall GetPortTypeEntries(TInvRegIntfEntryArray Entries);
```

Description

This is GetPortTypeEntries, a member of class TWSDLPublish.

77.1.2.2.2 TWSDLPublish.GetPortTypeList Method

Pascal

```
function GetPortTypeList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetPortTypeList();
```

Description

This is GetPortTypeList, a member of class TWSDLPublish.

77.1.2.2.3 TWSDLPublish.GetTypeSystemsList Method

Pascal

```
function GetTypeSystemsList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetTypeSystemsList();
```

Description

This is GetTypeSystemsList, a member of class TWSDLPublish.

77.1.2.2.4 TWSDLPublish.GetWSDLForPortType Method

Pascal

```
function GetWSDLForPortType(const PortType: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetWSDLForPortType(const InvString PortType);
```

Description

This is GetWSDLForPortType, a member of class TWSDLPublish.

77.1.2.2.5 TWSDLPublish.GetXSDForTypeSystem Method

Pascal

```
function GetXSDForTypeSystem(const TypeSystem: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetXSDForTypeSystem(const InvString TypeSystem);
```

Description

This is GetXSDForTypeSystem, a member of class TWSDLPublish.

77.1.2.3 TWSDLPublish Properties

The properties of the TWSDLPublish class are listed here.

Public Properties

	Name	Description
	TargetNamespace (see page 7293)	This is TargetNamespace, a member of class TWSDLPublish.

77.1.2.3.1 TWSDLPublish.TargetNamespace Property

Pascal

```
property TargetNamespace: InvString;
```

C++

```
__property InvString TargetNamespace;
```

Description

This is TargetNamespace, a member of class TWSDLPublish.

77.1.2.4 TWSDLPublish Events

The events of the TWSDLPublish class are listed here.

Public Events

	Name	Description
	OnAfterPublishingWSDL (see page 7294)	This is OnAfterPublishingWSDL, a member of class TWSDLPublish.
	OnBeforePublishingTypes (see page 7294)	This is OnBeforePublishingTypes, a member of class TWSDLPublish.
	OnBeforePublishingWSDL (see page 7294)	This is OnBeforePublishingWSDL, a member of class TWSDLPublish.

	OnPublishingType (see page 7294)	This is OnPublishingType, a member of class TWSDLPublish.

77.1.2.4.1 TWSDLPublish.OnAfterPublishingWSDL Event

Pascal

```
property OnAfterPublishingWSDL: TAfterPublishingWSDLEvent;
```

C++

```
__property TAfterPublishingWSDLEvent OnAfterPublishingWSDL;
```

Description

This is OnAfterPublishingWSDL, a member of class TWSDLPublish.

77.1.2.4.2 TWSDLPublish.OnBeforePublishingTypes Event

Pascal

```
property OnBeforePublishingTypes: TBeforePublishingTypesEvent;
```

C++

```
__property TBeforePublishingTypesEvent OnBeforePublishingTypes;
```

Description

This is OnBeforePublishingTypes, a member of class TWSDLPublish.

77.1.2.4.3 TWSDLPublish.OnBeforePublishingWSDL Event

Pascal

```
property OnBeforePublishingWSDL: TBeforePublishingWSDLEvent;
```

C++

```
__property TBeforePublishingWSDLEvent OnBeforePublishingWSDL;
```

Description

This is OnBeforePublishingWSDL, a member of class TWSDLPublish.

77.1.2.4.4 TWSDLPublish.OnPublishingType Event

Pascal

```
property OnPublishingType: TPublishingTypeEvent;
```

C++

```
__property TPublishingTypeEvent OnPublishingType;
```

Description

This is OnPublishingType, a member of class TWSDLPublish.

77.2 Interfaces

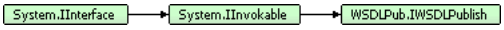
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IWSDLPublish (see page 7295)	This is class WSDLPub.IWSDLPublish.

77.2.1 IWSDLPublish Interface

Class Hierarchy



Pascal

```
IWSDLPublish = interface(IInvokable);
```

C++

```
__interface IWSDLPublish : public IInvokable;
```

File

WSDLPub

Description

This is class WSDLPub.IWSDLPublish.

77.2.1.1 IWSDLPublish Members

The following tables list the members exposed by IWSDLPublish.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	GetPortTypeList (see page 7296)	This is GetPortTypeList, a member of class IWSDLPublish.
	GetTypeSystemsList (see page 7296)	This is GetTypeSystemsList, a member of class IWSDLPublish.
	GetWSDLForPortType (see page 7296)	This is GetWSDLForPortType, a member of class IWSDLPublish.
	GetXSDForTypeSystem (see page 7296)	This is GetXSDForTypeSystem, a member of class IWSDLPublish.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.

77.2.1.2 IWSDLPublish Methods

The methods of the IWSDLPublish class are listed here.

Public Methods

	Name	Description
	GetPortTypeList (see page 7296)	This is GetPortTypeList, a member of class IWSDLPublish.
	GetTypeSystemsList (see page 7296)	This is GetTypeSystemsList, a member of class IWSDLPublish.
	GetWSDLForPortType (see page 7296)	This is GetWSDLForPortType, a member of class IWSDLPublish.
	GetXSDForTypeSystem (see page 7296)	This is GetXSDForTypeSystem, a member of class IWSDLPublish.

77.2.1.2.1 IWSDLPublish.GetPortTypeList Method

Pascal

```
function GetPortTypeList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetPortTypeList();
```

Description

This is GetPortTypeList, a member of class IWSDLPublish.

77.2.1.2.2 IWSDLPublish.GetTypeSystemsList Method

Pascal

```
function GetTypeSystemsList: InvStringDynArray; stdcall;
```

C++

```
__fastcall __stdcall InvStringDynArray GetTypeSystemsList();
```

Description

This is GetTypeSystemsList, a member of class IWSDLPublish.

77.2.1.2.3 IWSDLPublish.GetWSDLForPortType Method

Pascal

```
function GetWSDLForPortType(const PortType: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetWSDLForPortType(const InvString PortType);
```

Description

This is GetWSDLForPortType, a member of class IWSDLPublish.

77.2.1.2.4 IWSDLPublish.GetXSDForTypeSystem Method

Pascal

```
function GetXSDForTypeSystem(const TypeSystem: InvString): InvString; stdcall;
```

C++

```
__fastcall __stdcall InvString GetXSDForTypeSystem(const InvString TypeSystem);
```

Description

This is GetXSDForTypeSystem, a member of class IWSDLPublish.

77.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	WSDLPubFactory (see page 7297)	This is function WSDLPub.WSDLPubFactory.

77.3.1 WSDLPub.WSDLPubFactory Function

Pascal

```
procedure WSDLPubFactory(out obj: TObject);
```

C++

```
WSDLPubFactory(TObject * obj);
```

File

WSDLPub

Description

This is function WSDLPub.WSDLPubFactory.

77.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	SOAPPublishOptions (see page 7298)	SOAPPublishOptions and TSOAPPublishOptions (see page 7299) are flags that customize the definitions the TWSDLHTMLPublish (see page 7280) creates for a Web Service.

77.4.1 WSDLPub.SOAPPublishOptions Enumeration

SOAPPublishOptions and TSOAPPublishOptions (see page 7299) are flags that customize the definitions the TWSDLHTMLPublish (see page 7280) creates for a Web Service.

Pascal

```
SOAPPublishOptions = (
  poDefault,
  poPublishLocationAsSecure,
  poUTF8ContentType
);
```

C++

```
enum SOAPPublishOptions {
  poDefault,
  poPublishLocationAsSecure,
  poUTF8ContentType
};
```

File

WSDLPub

Description

TSOAPPublishOptions (see page 7299) is a set of SOAPPublishOptions values. SOAPPublishOptions values are flags that configure the definitions that TWSDLHTMLPublish (see page 7280) creates when describing a Web Service. The following table lists the possible values:

Value	Meaning
poDefault	Use the default options when generating WSDL documents.
poPublishLocationAsSecure	Add a property that indicates the Web Service is on a secure server. Clients should use https rather than http.

77.5 Types

The following table lists types in this documentation.

Types

Name	Description
TBeforePublishingWSDLEvent (see page 7299)	TBeforePublishingWSDLEvent is the type of the OnBeforePublishingWSDL event of TWSDLHTMLPublish (see page 7280).
TSOAPPublishOptions (see page 7299)	This is type WSDLPub.TSOAPPublishOptions.

77.5.1 WSDLPub.TBeforePublishingWSDLEvent Type

TBeforePublishingWSDLEvent is the type of the OnBeforePublishingWSDL event of TWSDLHTMLPublish (see page 7280).

Pascal

```
TBeforePublishingWSDLEvent = procedure (const IntfName: InvString; var WSDL: InvString; var
Handled: Boolean) of object;
```

C++

```
(const IntfName: InvString; var WSDL: InvString; var Handled: Boolean) (
TBeforePublishingWSDLEvent)();
```

File

WSDLPub

Description

TBeforePublishingWSDLEvent is the type of the OnBeforePublishingWSDL event handler, which allows an application to generate the content of a WSDL document on behalf of the WSDL publisher.

IntfName is the name of the invokable interface for which to generate a WSDL description.

WSDL returns a string of XML that makes up the content of the WSDL document that describes the specified interface.

Handled returns true if the event handler generates and returns a value for WSDL. If Handled returns true, the publisher uses the value of WSDL rather than generating its own WSDL document. If Handled returns false, the publisher generates a WSDL document for the interface and all types that it uses.

See Also

Procedural types

Creating events

77.5.2 WSDLPub.TSOAPPublishOptions Type

Pascal

```
TSOAPPublishOptions = set of SOAPPublishOptions;
```

C++

```
SOAPPublishOptions TSOAPPublishOptions;
```

File

WSDLPub

Description

This is type WSDLPub.TSOAPPublishOptions.

78 XMLBrokr Namespace

This is namespace XMLBrokr.

Topics

Name	Description
Classes (see page 7303)	The following table lists classes in this documentation.
Interfaces (see page 7337)	The following table lists interfaces in this documentation.
Functions (see page 7339)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7340)	The following table lists structs, records, enums in this documentation.
Types (see page 7341)	The following table lists types in this documentation.
Constants (see page 7343)	The following table lists constants in this documentation.

Classes

	Name	Description
	TXMLBroker (see page 7303)	TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.
	TXMLParams (see page 7333)	TXMLParams manages a list of parameters for an XML broker.

Constants

Name	Description
sPostDelta (see page 7343)	This is constant XMLBrokr.sPostDelta.
sProducer (see page 7343)	This is constant XMLBrokr.sProducer.
sRedirect (see page 7343)	This is constant XMLBrokr.sRedirect.

Functions

	Name	Description
	FormatXML (see page 7339)	This is function XMLBrokr.FormatXML.

Interfaces

	Name	Description
	INotifyConnectionChange (see page 7337)	This is class XMLBrokr.INotifyConnectionChange.

Structs, Records, Enums

	Name	Description
	TXMLOption (see page 7340)	TXMLOptions (see page 7342) is a set of flags that indicates how database information should be formatted in an XML data packet.

Types

Name	Description
TGetErrorResponseEvent (see page 7341)	This is type XMLBrokr.TGetErrorResponseEvent.
TGetResponseEvent (see page 7341)	This is type XMLBrokr.TGetResponseEvent.
TRequestRecordsEvent (see page 7342)	This is type XMLBrokr.TRequestRecordsEvent.
TRequestUpdateEvent (see page 7342)	This is type XMLBrokr.TRequestUpdateEvent.
TXMLOptions (see page 7342)	This is type XMLBrokr.TXMLOptions.

78.1 Classes

The following table lists classes in this documentation.

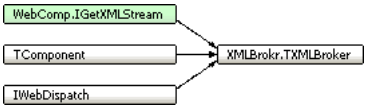
Classes

	Name	Description
	TXMLBroker (see page 7303)	TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.
	TXMLParams (see page 7333)	TXMLParams manages a list of parameters for an XML broker.

78.1.1 TXMLBroker Class

TXMLBroker fetches XML data packets from an application server and applies XML updates to the application server.

Class Hierarchy



Pascal

```
TXMLBroker = class(TComponent, IWebDispatch, IGetXMLStream);
```

C++

```
class TXMLBroker : public TComponent, public IWebDispatch, public IGetXMLStream;
```

File

XMLBrokr

Description

Use TXMLBroker in an application that is simultaneously the client of a multi-tiered database application and a Web server application. This class serves two major functions:

It fetches XML data packets from a provider using the IAppServer interface.

It receives HTTP messages from Web browsers that contain XML delta packets and applies them to the application server.

Add a TXMLBroker object to a Web module to make database information supplied by a provider on the application server available as XML. An InternetExpress page producer (TInetXPageProducer) in the same Web module uses this XML to build HTML pages that represent that database information.

The XML broker automatically registers itself with the Web module (or Web dispatcher) as an auto-dispatching object. This means that the Web module or Web dispatcher forwards all incoming HTTP messages directed to the XML broker without requiring the use of Web action items.

These incoming messages are assumed to be update messages created by the browser in response to HTML generated by a TApplyUpdatesButton component. The XML broker automatically applies the delta packet (the content of the HTTP request message) to the application server, and passes any update errors to a separate content producer, which can generate a response message.

See Also

TRemoteDataModule

IAppServer

TInetXPageProducer

TApplyUpdatesButton

Building an InternetExpress Application

78.1.1.1 TXMLBroker Members

The following tables list the members exposed by TXMLBroker.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 7308)	This is AS_ApplyUpdates, a member of class TXMLBroker.
	AS_FetchParams (see page 7309)	This is AS_FetchParams, a member of class TXMLBroker.
	AS_GetRecords (see page 7309)	This is AS_GetRecords, a member of class TXMLBroker.
	CheckInactive (see page 7309)	This is CheckInactive, a member of class TXMLBroker.
	DispatchEnabled (see page 7309)	
	DispatchMask (see page 7309)	
	DispatchMethodType (see page 7310)	
	DispatchRequest (see page 7310)	
	GetAppServer (see page 7310)	This is GetAppServer, a member of class TXMLBroker.
	GetConnected (see page 7311)	This is GetConnected, a member of class TXMLBroker.
	GetErrorResponse (see page 7312)	This is GetErrorResponse, a member of class TXMLBroker.
	GetHasAppServer (see page 7312)	This is GetHasAppServer, a member of class TXMLBroker.
	GetNotify (see page 7312)	This is GetNotify, a member of class TXMLBroker.
	GetNotifyCount (see page 7313)	This is GetNotifyCount, a member of class TXMLBroker.
	GetProducerName (see page 7313)	This is GetProducerName, a member of class TXMLBroker.
	GetRedirect (see page 7313)	This is GetRedirect, a member of class TXMLBroker.
	GetResponse (see page 7313)	This is GetResponse, a member of class TXMLBroker.
	GetXMLStream (see page 7314)	This is GetXMLStream, a member of class TXMLBroker.
V	Notification (see page 7316)	Forwards notification messages to all owned components.
	SendConnectEvent (see page 7319)	This is SendConnectEvent, a member of class TXMLBroker.
	SetAppServer (see page 7319)	This is SetAppServer, a member of class TXMLBroker.
	SetConnected (see page 7319)	This is SetConnected, a member of class TXMLBroker.
	SetParams (see page 7319)	This is SetParams, a member of class TXMLBroker.
	SetProviderName (see page 7320)	This is SetProviderName, a member of class TXMLBroker.
	SetReconcileProducer (see page 7320)	This is SetReconcileProducer, a member of class TXMLBroker.
	SetRemoteServer (see page 7321)	This is SetRemoteServer, a member of class TXMLBroker.
	SetWebDispatch (see page 7321)	This is SetWebDispatch, a member of class TXMLBroker.

Public Methods

	Name	Description
	AddNotify (see page 7307)	Adds an object to the Notify (see page 7325) property list.
	ApplyXMLUpdates (see page 7308)	Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.
V	Create (see page 7305)	Creates a new TXMLBroker (see page 7303) object.
V	Destroy (see page 7306)	Frees the memory associated with the TXMLBroker (see page 7303) object.
V	FetchParams (see page 7310)	Retrieves parameter values from the application server.
	GetDelta (see page 7311)	Extracts an XML delta packet from an HTTP request message.

	GetErrorCount (see page 7311)	Returns the number of errors in the delta packet returned by GetErrors (see page 7312).
	GetErrors (see page 7312)	Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.
	GetXMLRecords (see page 7314)	Retrieves an XML data packet from the application server.
	HTMLSubmitFormName (see page 7315)	Returns the name of the HTML form that submits updates to the XML broker.
	MasterRowSetVarName (see page 7315)	Returns the name of the javascript variable for the XML broker's dataset.
	RemoveNotify (see page 7316)	Removes an object to the Notify (see page 7325) property list.
V	RequestRecords (see page 7316)	Returns an XML data packet from the application server.
V	RequestUpdate (see page 7317)	Responds to an HTTP update request.
	RowSetVarName (see page 7318)	Returns the name of the javascript variable for the XML broker's dataset.
	SetProvider (see page 7320)	Associates the XML broker with a provider in the same process space.
	SubmitFormVarName (see page 7321)	Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 7315).

Public Properties

	Name	Description
	AppServer (see page 7322)	Provides access to the interface the XML broker uses to communicate with the application server.
R	HasAppServer (see page 7323)	Indicates whether the XML broker has an IAppServer interface for communicating with its provider.
R	Notify (see page 7325)	Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.
R	NotifyCount (see page 7325)	Indicates the number of objects in the Notify (see page 7325) property list.

Published Events

	Name	Description
	AfterDispatch (see page 7329)	Occurs after the XML broker generates the content of a response message.
	BeforeDispatch (see page 7330)	Occurs when the XML broker receives an HTTP request message.
	OnGetErrorResponse (see page 7330)	Occurs when there are errors applying updates to the application server.
	OnGetResponse (see page 7331)	Occurs when there are no errors applying updates to the application server.
	OnRequestRecords (see page 7332)	Occurs when the XML broker needs to fetch a data packet from its associated provider.
	OnRequestUpdate (see page 7332)	Occurs immediately before the XML broker applies updates to the application server.

Published Properties

	Name	Description
	Connected (see page 7322)	Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.
	MaxErrors (see page 7323)	Specifies the maximum number of update errors allowed before aborting an update operations.
	MaxRecords (see page 7324)	Specifies the maximum number of records that can appear in XML data packets.
	Params (see page 7325)	Contains parameter values that are sent to or retrieved from the application server.
	ProviderName (see page 7326)	Specifies the name of a registered provider from the application server.
	ReconcileProducer (see page 7327)	Specifies the producer that generates response message content when an update generates errors.
	RemoteServer (see page 7328)	Specifies the connection component that this XML broker uses to connect to an application server.
	WebDispatch (see page 7328)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

78.1.1.2 TXMLBroker.Create Constructor

Creates a new TXMLBroker (see page 7303) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLBroker(TComponent * AOwner);
```

Description

Most applications do not need to call the XML broker constructor. Typically the XML broker is added to a Web module at design time, so that the constructor is called automatically.

The Owner parameter specifies the component (usually the Web module) that is responsible for freeing the XML broker.

See Also

Owner

78.1.1.3 TXMLBroker.Destroy Destructor

Frees the memory associated with the TXMLBroker (see page 7303) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLBroker();
```

Description

Do not call Destroy directly. Instead,call Free, which verifies that the XML broker is not nil before calling Destroy.

See Also

System::TObject::Free (see page 3372)

78.1.1.4 TXMLBroker Methods

The methods of the TXMLBroker class are listed here.

Protected Methods

	Name	Description
	AS_ApplyUpdates (see page 7308)	This is AS_ApplyUpdates, a member of class TXMLBroker.
	AS_FetchParams (see page 7309)	This is AS_FetchParams, a member of class TXMLBroker.
	AS_GetRecords (see page 7309)	This is AS_GetRecords, a member of class TXMLBroker.
	CheckInactive (see page 7309)	This is CheckInactive, a member of class TXMLBroker.
	DispatchEnabled (see page 7309)	
	DispatchMask (see page 7309)	
	DispatchMethodType (see page 7310)	
	DispatchRequest (see page 7310)	
	GetAppServer (see page 7310)	This is GetAppServer, a member of class TXMLBroker.
	GetConnected (see page 7311)	This is GetConnected, a member of class TXMLBroker.
	GetErrorResponse (see page 7312)	This is GetErrorResponse, a member of class TXMLBroker.
	GetHasAppServer (see page 7312)	This is GetHasAppServer, a member of class TXMLBroker.
	GetNotify (see page 7312)	This is GetNotify, a member of class TXMLBroker.
	GetNotifyCount (see page 7313)	This is GetNotifyCount, a member of class TXMLBroker.
	GetProducerName (see page 7313)	This is GetProducerName, a member of class TXMLBroker.

	GetRedirect (see page 7313)	This is GetRedirect, a member of class TXMLBroker.
	GetResponse (see page 7313)	This is GetResponse, a member of class TXMLBroker.
	GetXMLStream (see page 7314)	This is GetXMLStream, a member of class TXMLBroker.
	Notification (see page 7316)	Forwards notification messages to all owned components.
	SendConnectEvent (see page 7319)	This is SendConnectEvent, a member of class TXMLBroker.
	SetAppServer (see page 7319)	This is SetAppServer, a member of class TXMLBroker.
	SetConnected (see page 7319)	This is SetConnected, a member of class TXMLBroker.
	SetParams (see page 7319)	This is SetParams, a member of class TXMLBroker.
	SetProviderName (see page 7320)	This is SetProviderName, a member of class TXMLBroker.
	SetReconcileProducer (see page 7320)	This is SetReconcileProducer, a member of class TXMLBroker.
	SetRemoteServer (see page 7321)	This is SetRemoteServer, a member of class TXMLBroker.
	SetWebDispatch (see page 7321)	This is SetWebDispatch, a member of class TXMLBroker.

Public Methods

	Name	Description
	AddNotify (see page 7307)	Adds an object to the Notify (see page 7325) property list.
	ApplyXMLUpdates (see page 7308)	Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.
	FetchParams (see page 7310)	Retrieves parameter values from the application server.
	GetDelta (see page 7311)	Extracts an XML delta packet from an HTTP request message.
	GetErrorCount (see page 7311)	Returns the number of errors in the delta packet returned by GetErrors (see page 7312).
	GetErrors (see page 7312)	Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.
	GetXMLRecords (see page 7314)	Retrieves an XML data packet from the application server.
	HTMLSubmitFormName (see page 7315)	Returns the name of the HTML form that submits updates to the XML broker.
	MasterRowSetVarName (see page 7315)	Returns the name of the javascript variable for the XML broker's dataset.
	RemoveNotify (see page 7316)	Removes an object to the Notify (see page 7325) property list.
	RequestRecords (see page 7316)	Returns an XML data packet from the application server.
	RequestUpdate (see page 7317)	Responds to an HTTP update request.
	RowSetVarName (see page 7318)	Returns the name of the javascript variable for the XML broker's dataset.
	SetProvider (see page 7320)	Associates the XML broker with a provider in the same process space.
	SubmitFormVarName (see page 7321)	Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 7315).

78.1.1.4.1 TXMLBroker.AddNotify Method

Adds an object to the Notify (see page 7325) property list.

Pascal

```
procedure AddNotify(ANotify: TObject);
```

C++

```
__fastcall AddNotify(TObject * ANotify);
```

Description

Call AddNotify to add an object to the list maintained by the Notify (see page 7325) property. Each object in this list is notified every time a property that affects the metadata of the XML broker's data packets changes. These include changes to the RemoteServer (see page 7328), AppServer (see page 7322), and ProviderName (see page 7326) properties.

ANotify specifies the object that wants to be notified of property changes. This object must support the INotifyConnectionChange (see page 7337) interface, which the XML broker uses to notify it of changes.

See Also

NotifyCount (see page 7325)

Notify (see page 7325)

RemoveNotify (see page 7316)

RemoteServer (see page 7328)

AppServer (see page 7322)

ProviderName (see page 7326)

78.1.1.4.2 TXMLBroker.ApplyXMLUpdates Method

Sends an XML delta packet of updated, inserted, and deleted records to the provider on the application server for writing to the database.

Pascal

```
function ApplyXMLUpdates(const Delta: string; out ErrorCount: Integer): string;
```

C++

```
__fastcall UnicodeString ApplyXMLUpdates(const UnicodeString Delta, int ErrorCount);
```

Description

Call ApplyXMLUpdates to send the updates in the XML delta packet specified by Delta to the application server. ApplyXMLUpdates is used internally (by the RequestUpdate (see page 7317) method) to apply updates in an XML delta packet and retrieve a delta packet containing any failed updates.

Delta is the XML delta packet that contains the insertions, deletions and modifications from a Web browser. Typically, it is retrieved from an HTTP request message using the GetDelta (see page 7311) method.

ErrorCount returns the number of errors encountered on the application server before it terminated the update operation. If ErrorCount is 0, all updates were applied successfully. ErrorCount can not exceed the value of MaxErrors (see page 7323).

ApplyXMLUpdates returns an XML delta packet containing all updates that were not successfully applied. If ErrorCount is 0, this is an empty string.

See Also

RequestUpdate (see page 7317)

AS_ApplyUpdates

MaxErrors (see page 7323)

GetDelta (see page 7311)

78.1.1.4.3 TXMLBroker.AS_ApplyUpdates Method

Pascal

```
function AS_ApplyUpdates(Delta: OleVariant; MaxErrors: Integer; out ErrorCount: Integer):
OleVariant;
```

C++

```
__fastcall OleVariant AS_ApplyUpdates(OleVariant Delta, int MaxErrors, int ErrorCount);
```

Description

This is AS_ApplyUpdates, a member of class TXMLBroker.

78.1.1.4.4 TXMLBroker.AS_FetchParams Method

Pascal

```
procedure AS_FetchParams;
```

C++

```
__fastcall AS_FetchParams();
```

Description

This is AS_FetchParams, a member of class TXMLBroker.

78.1.1.4.5 TXMLBroker.AS_GetRecords Method

Pascal

```
function AS_GetRecords(Count: Integer; out RecsOut: Integer; Options: Integer; const
CommandText: WideString; Params: OleVariant; var OwnerData: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant AS_GetRecords(int Count, int RecsOut, int Options, const BSTR
CommandText, OleVariant Params, OleVariant OwnerData);
```

Description

This is AS_GetRecords, a member of class TXMLBroker.

78.1.1.4.6 TXMLBroker.CheckInactive Method

Pascal

```
procedure CheckInactive;
```

C++

```
__fastcall CheckInactive();
```

Description

This is CheckInactive, a member of class TXMLBroker.

78.1.1.4.7 TXMLBroker.DispatchEnabled Method

Pascal

```
function DispatchEnabled: Boolean;
```

C++

```
__fastcall Boolean DispatchEnabled();
```

See Also

Enabled

78.1.1.4.8 TXMLBroker.DispatchMask Method

Pascal

```
function DispatchMask: TMask;
```

C++

```
__fastcall TMask DispatchMask();
```

See Also

Mask

78.1.1.4.9 TXMLBroker.DispatchMethodType Method

Pascal

```
function DispatchMethodType: TMethodType;
```

C++

```
__fastcall TMethodType DispatchMethodType();
```

See Also

MethodType

78.1.1.4.10 TXMLBroker.DispatchRequest Method

Pascal

```
function DispatchRequest(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean;
```

C++

```
__fastcall Boolean DispatchRequest(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

78.1.1.4.11 TXMLBroker.FetchParams Method

Retrieves parameter values from the application server.

Pascal

```
procedure FetchParams; virtual;
```

C++

```
virtual __fastcall FetchParams();
```

Description

Call FetchParams to initialize the Params (see page 7325) property with the current parameter values on the application server. FetchParams does nothing unless the provider component on the application server can supply parameter values.

Warning: if the XML broker uses a stateless application server, other clients can change the values returned by FetchParams at any time.

See Also

RowRequest

Params (see page 7325)

78.1.1.4.12 TXMLBroker.GetAppServer Method

Pascal

```
function GetAppServer: IAppServer;
```

C++

```
__fastcall IAppServer GetAppServer();
```

Description

This is GetAppServer, a member of class TXMLBroker.

78.1.1.4.13 TXMLBroker.GetConnected Method

Pascal

```
function GetConnected: Boolean;
```

C++

```
__fastcall Boolean GetConnected();
```

Description

This is GetConnected, a member of class TXMLBroker.

78.1.1.4.14 TXMLBroker.GetDelta Method

Extracts an XML delta packet from an HTTP request message.

Pascal

```
function GetDelta(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString GetDelta(TWebRequest Request);
```

Description

Call GetDelta to obtain the XML delta packet contained in the content of the request message specified by Request. This value can be used to apply the updates from the request message by calling ApplyXMLUpdates (see page 7308).

See Also

ApplyXMLUpdates (see page 7308)

78.1.1.4.15 TXMLBroker.GetErrorCount Method

Returns the number of errors in the delta packet returned by GetErrors (see page 7312).

Pascal

```
function GetErrorCount: Integer;
```

C++

```
__fastcall int GetErrorCount();
```

Description

Call GetErrorCount to determine the number errors included in the XML delta packet that the application server returns when an update operation causes errors.

See Also

OnGetErrorResponse (see page 7330)

ApplyXMLUpdates (see page 7308)

GetErrors (see page 7312)

78.1.1.4.16 TXMLBroker.GetErrorResponse Method

Pascal

```
function GetErrorResponse(ErrorCount: Integer; XMLErrors: string; Request: TWebRequest;
Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean GetErrorResponse(int ErrorCount, UnicodeString XMLErrors, TWebRequest
Request, TWebResponse Response);
```

Description

This is GetErrorResponse, a member of class TXMLBroker.

78.1.1.4.17 TXMLBroker.GetErrors Method

Returns the XML delta packet (if any) that describes the errors that occurred when applying updates.

Pascal

```
function GetErrors: string;
```

C++

```
__fastcall UnicodeString GetErrors();
```

Description

Call GetErrors to obtain the XML delta packet returned by the application server when an update operation included some errors.

Note: See the template err.html (included in the WebMidas directory) for one possible use for the XML delta packet of errors. In err.html, the string returned by GetErrors should replace the <#XMLEMBED_VAR_ERR> tag.

See Also

OnGetErrorResponse (see page 7330)

ApplyXMLUpdates (see page 7308)

GetErrorCount (see page 7311)

78.1.1.4.18 TXMLBroker.GetHasAppServer Method

Pascal

```
function GetHasAppServer: Boolean;
```

C++

```
__fastcall Boolean GetHasAppServer();
```

Description

This is GetHasAppServer, a member of class TXMLBroker.

78.1.1.4.19 TXMLBroker.GetNotify Method

Pascal

```
function GetNotify(Index: Integer): TObject;
```

C++

```
__fastcall TObject * GetNotify(int Index);
```

Description

This is GetNotify, a member of class TXMLBroker.

78.1.1.4.20 TXMLBroker.GetNotifyCount Method

Pascal

```
function GetNotifyCount: Integer;
```

C++

```
__fastcall int GetNotifyCount();
```

Description

This is GetNotifyCount, a member of class TXMLBroker.

78.1.1.4.21 TXMLBroker.GetProducerName Method

Pascal

```
function GetProducerName(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString GetProducerName(TWebRequest Request);
```

Description

This is GetProducerName, a member of class TXMLBroker.

78.1.1.4.22 TXMLBroker.GetRedirect Method

Pascal

```
function GetRedirect(Request: TWebRequest): string;
```

C++

```
__fastcall UnicodeString GetRedirect(TWebRequest Request);
```

Description

This is GetRedirect, a member of class TXMLBroker.

78.1.1.4.23 TXMLBroker.GetResponse Method

Pascal

```
function GetResponse(Request: TWebRequest; Response: TWebResponse): Boolean;
```

C++

```
__fastcall Boolean GetResponse(TWebRequest Request, TWebResponse Response);
```

Description

This is GetResponse, a member of class TXMLBroker.

78.1.1.4.24 TXMLBroker.GetXMLRecords Method

Retrieves an XML data packet from the application server.

Pascal

```
function GetXMLRecords(var RecsOut: Integer; var OwnerData: OleVariant; XMLOptions:
TXMLOptions): string;
```

C++

```
__fastcall UnicodeString GetXMLRecords(int RecsOut, OleVariant OwnerData, TXMLOptions
XMLOptions);
```

Description

Call GetXMLRecords to obtain the XML data packet from the application server. GetXMLRecords calls the interface specified by AppServer (see page 7322), requesting an XML data packet with at most MaxRecords (see page 7324) records. It returns the resulting XML data packet in a form that is ready for insertion into an HTML document.

RecsOut returns the number of records in the XML data packet.

On entry, OwnerData contains an application-defined value that is passed to the provider's BeforeGetRecords event handler. OwnerData returns an application-defined value that is set by the provider's AfterGetRecords event handler. Typically, this parameter is used to transmit persistent state information to and from a stateless application server.

XMLOptions is a set of flags that indicate how the broker should format the XML data packet after it is retrieved from the application server.

GetXMLRecords returns the formatted XML data packet.

Note: Unlike the RequestRecords (see page 7316) method, GetXMLRecords bypasses the OnRequestRecords (see page 7332) event.

See Also

RequestRecords (see page 7316)

AS_GetRecords

MaxRecords (see page 7324)

AppServer (see page 7322)

BeforeGetRecords

AfterGetRecords

Using an XML Broker

78.1.1.4.25 TXMLBroker.GetXMLStream Method

Pascal

```
function GetXMLStream(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLStream(Boolean Owned);
```

Description

This is GetXMLStream, a member of class TXMLBroker.

78.1.1.4.26 TXMLBroker.HTMLSubmitFormName Method

Returns the name of the HTML form that submits updates to the XML broker.

Pascal

```
function HTMLSubmitFormName: string;
```

C++

```
__fastcall UnicodeString HTMLSubmitFormName();
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares an HTML form that submits the updates to the XML broker. HTMLSubmitFormName is the name of that HTML form.

The HTML form named by this return value sends the HTTP request messages containing an XML delta packet of updates to be applied.

See Also

SubmitFormVarName (see page 7321)

Content

78.1.1.4.27 TXMLBroker.MasterRowSetVarName Method

Returns the name of the javascript variable for the XML broker's dataset.

Pascal

```
function MasterRowSetVarName(Path: TStrings): string;
```

C++

```
__fastcall UnicodeString MasterRowSetVarName(TStrings Path);
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares a javascript variable of type RowSet for the javascript object that represents the XML broker's dataset. When the variable is first declared, the InternetExpress page producer calls MasterRowSetVarName.

Path is a set of variable names for all the master datasets for which the XML broker's dataset is a detail set. These names come from previous calls to the MasterRowSetVarName method of their XML brokers. The last string in Path refers to the XML broker's dataset itself.

MasterRowSetVarName returns the name of the XML broker, qualified by all strings in Path except for the last one, which is assumed to refer to the XML broker's dataset. Thus, to change the name of the generated variable for an XML broker's dataset, change the Name property of the XML broker.

Note: To retrieve the name of the XML broker preceded by all the datasets in Path, use the RowSetVarName (see page 7318) method instead.

See Also

Name

Content

RowSetVarName (see page 7318)

78.1.1.4.28 TXMLBroker.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

78.1.1.4.29 TXMLBroker.RemoveNotify Method

Removes an object to the Notify (see page 7325) property list.

Pascal

```
procedure RemoveNotify(ANotify: TObject);
```

C++

```
__fastcall RemoveNotify(TObject * ANotify);
```

Description

Call RemoveNotify to remove an object from the list maintained by the Notify (see page 7325) property. Each object in this list is notified every time a property that affects the metadata of the XML broker's data packets changes.

See Also

NotifyCount (see page 7325)

Notify (see page 7325)

AddNotify (see page 7307)

78.1.1.4.30 TXMLBroker.RequestRecords Method

Returns an XML data packet from the application server.

Pascal

```
function RequestRecords(Sender: TObject; Request: TWebRequest; out RecCount: Integer; var
OwnerData: OleVariant; XMLOptions: TXMLOptions): string; virtual;
```

C++

```
virtual __fastcall UnicodeString RequestRecords(TObject * Sender, TWebRequest Request, int
RecCount, OleVariant OwnerData, TXMLOptions XMLOptions);
```

Description

Call RequestRecords to obtain a current XML data packet, while first allowing the OnRequestRecords (see page 7332) event handler to customize it. After generating the OnRequestRecords (see page 7332) event, RequestRecords fetches the XML data packet by calling GetXMLRecords (see page 7314), unless the event handler supplies the XML data packet first.

Sender is the object that is requesting the XML data packet (usually an InternetExpress page producer).

Request describes an HTTP request message that originated the need for an XML data packet. Typically, this is a request message for which the InternetExpress page producer is creating a response.

RecCount returns the number of records in the XML data packet.

On entry, OwnerData contains an application-defined value that is passed to the OnRequestRecords (see page 7332) event handler, and then forwarded (with any changes provided by the event handler) to the provider's BeforeGetRecords event handler. OwnerData returns an application-defined value that is set by the provider's AfterGetRecords event handler. Typically, this parameter is used to transmit persistent state information to and from a stateless application server.

XMLOptions is a set of flags indicating how the XML broker should format the XML data packet that the application server or OnRequestRecords (see page 7332) event handler supplies.

RequestRecords returns the XML data packet.

See Also

GetXMLRecords (see page 7314)

AS_GetRecords

OnRequestRecords (see page 7332)

AppServer (see page 7322)

BeforeGetRecords

AfterGetRecords

Using an XML Broker

78.1.1.4.31 TXMLBroker.RequestUpdate Method

Responds to an HTTP update request.

Pascal

```
function RequestUpdate(Sender: TObject; Request: TWebRequest; Response: TWebResponse):
Boolean; virtual;
```

C++

```
virtual __fastcall Boolean RequestUpdate(TObject * Sender, TWebRequest Request, TWebResponse
Response);
```

Description

Call RequestUpdate when using a Web action item to respond to update requests from a Web browser. When using a Web

action item to respond to update requests, instead of letting the XML broker receive the update requests directly, you should set the Enabled property of the WebDispatch (see page 7328) to false.

RequestUpdate generates an OnRequestUpdate (see page 7332) event, to allow the event handler to apply the updates in the HTTP request message and generate a response. If the OnRequestUpdate (see page 7332) event handler does not handle the message, RequestUpdate applies the updates using the ApplyXMLUpdates (see page 7308) method. Finally, if there are no update errors, RequestUpdate generates an OnGetResponse (see page 7331) event. If there were update errors, RequestResponse generates an OnGetErrorResponse (see page 7330) event and (if the event handler does not generate a response) assigns the content of ReconcileProducer (see page 7327) to the response message. If there are update errors and the response is not completed by an event handler or the ReconcileProducer (see page 7327), the XML broker raises an exception.

Sender is the object that received the request message.

Request describes an HTTP request message that contains the updates that need to be applied.

Response is the emerging response object that should be completed by RequestUpdate so that the XML broker can send it back to the Web browser.

RequestUpdate returns true if the response message is ready to send back to the Web browser, false if it still needs more processing.

See Also

ApplyXMLUpdates (see page 7308)

AS_ApplyUpdates

OnRequestUpdate (see page 7332)

AppServer (see page 7322)

BeforeApplyUpdates

AfterApplyUpdates

OnGetResponse (see page 7331)

OnGetErrorResponse (see page 7330)

ReconcileProducer (see page 7327)

WebDispatch (see page 7328)

Enabled

78.1.1.4.32 TXMLBroker.RowSetVarName Method

Returns the name of the javascript variable for the XML broker's dataset.

Pascal

```
function RowSetVarName(Path: TStrings): string;
```

C++

```
__fastcall UnicodeString RowSetVarName(TStrings Path);
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares a javascript variable of type RowSet for the javascript object that represents the XML broker's dataset. When the InternetExpress page producer needs to determine whether a RowSet variable was already declared, it looks up the variable name by calling RowSetVarName.

Path is a set of variable names for all the master datasets for which the XML broker's dataset is a detail set.

RowSetVarName returns the name of the XML broker, qualified by all strings in Path.

Note: Variable names are assigned for an XML broker's dataset using the MasterRowSetVarName (see page 7315) method.

See Also

Name

Content

MasterRowSetVarName (see page 7315)

78.1.1.4.33 TXMLBroker.SendConnectEvent Method

Pascal

```
procedure SendConnectEvent(Connecting: Boolean);
```

C++

```
__fastcall SendConnectEvent(Boolean Connecting);
```

Description

This is SendConnectEvent, a member of class TXMLBroker.

78.1.1.4.34 TXMLBroker.SetAppServer Method

Pascal

```
procedure SetAppServer(Value: IAppServer);
```

C++

```
__fastcall SetAppServer(IAppServer Value);
```

Description

This is SetAppServer, a member of class TXMLBroker.

78.1.1.4.35 TXMLBroker.SetConnected Method

Pascal

```
procedure SetConnected(Value: Boolean);
```

C++

```
__fastcall SetConnected(Boolean Value);
```

Description

This is SetConnected, a member of class TXMLBroker.

78.1.1.4.36 TXMLBroker.SetParams Method

Pascal

```
procedure SetParams(const Value: TXMLParams);
```

C++

```
__fastcall SetParams(const TXMLParams Value);
```

Description

This is SetParams, a member of class TXMLBroker.

78.1.1.4.37 TXMLBroker.SetProvider Method

Associates the XML broker with a provider in the same process space.

Pascal

```
procedure SetProvider(Provider: TComponent);
```

C++

```
__fastcall SetProvider(TComponent * Provider);
```

Description

Use SetProvider to specify a local provider component that provides data to and applies updates from the XML broker. The XML broker communicates with this provider using the IAppServer interface specified by the AppServer (see page 7322) property.

Do not use SetProvider for remote providers on an application server. If the provider is not in the same process space as the XML broker, use the ProviderName (see page 7326) property.

The Provider parameter must be a descendant of TCustomProvider. Otherwise, SetProvider raises an exception.

See Also

AppServer (see page 7322)

ProviderName (see page 7326)

TDataSetProvider

TCustomProvider

78.1.1.4.38 TXMLBroker.SetProviderName Method

Pascal

```
procedure SetProviderName(const Value: string);
```

C++

```
__fastcall SetProviderName(const UnicodeString Value);
```

Description

This is SetProviderName, a member of class TXMLBroker.

78.1.1.4.39 TXMLBroker.SetReconcileProducer Method

Pascal

```
procedure SetReconcileProducer(const Value: TCustomContentProducer);
```

C++

```
__fastcall SetReconcileProducer(const TCustomContentProducer Value);
```

Description

This is SetReconcileProducer, a member of class TXMLBroker.

78.1.1.4.40 TXMLBroker.SetRemoteServer Method

Pascal

```
procedure SetRemoteServer(Value: TCustomRemoteServer);
```

C++

```
__fastcall SetRemoteServer(TCustomRemoteServer Value);
```

Description

This is SetRemoteServer, a member of class TXMLBroker.

78.1.1.4.41 TXMLBroker.SetWebDispatch Method

Pascal

```
procedure SetWebDispatch(const Value: TWebDispatch);
```

C++

```
__fastcall SetWebDispatch(const TWebDispatch Value);
```

Description

This is SetWebDispatch, a member of class TXMLBroker.

78.1.1.4.42 TXMLBroker.SubmitFormVarName Method

Returns the name of the variable that refers to the HTML form named by HTMLSubmitFormName (see page 7315).

Pascal

```
function SubmitFormVarName: string;
```

C++

```
__fastcall UnicodeString SubmitFormVarName();
```

Description

When the InternetExpress page producer assembles an HTML document that uses the XML data packets managed by the XML broker, it declares an HTML form that submits the updates to the XML broker. In addition, it declares a variable that refers to that form. SubmitFormVarName is the name of that variable.

See Also

HTMLSubmitFormName (see page 7315)

Content

78.1.1.5 TXMLBroker Properties

The properties of the TXMLBroker class are listed here.

Public Properties

	Name	Description
	AppServer (see page 7322)	Provides access to the interface the XML broker uses to communicate with the application server.
R	HasAppServer (see page 7323)	Indicates whether the XML broker has an IAppServer interface for communicating with its provider.

	Notify (see page 7325)	Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.
	NotifyCount (see page 7325)	Indicates the number of objects in the Notify (see page 7325) property list.

Published Properties

	Name	Description
	Connected (see page 7322)	Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.
	MaxErrors (see page 7323)	Specifies the maximum number of update errors allowed before aborting an update operations.
	MaxRecords (see page 7324)	Specifies the maximum number of records that can appear in XML data packets.
	Params (see page 7325)	Contains parameter values that are sent to or retrieved from the application server.
	ProviderName (see page 7326)	Specifies the name of a registered provider from the application server.
	ReconcileProducer (see page 7327)	Specifies the producer that generates response message content when an update generates errors.
	RemoteServer (see page 7328)	Specifies the connection component that this XML broker uses to connect to an application server.
	WebDispatch (see page 7328)	Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

78.1.1.5.1 TXMLBroker.AppServer Property

Provides access to the interface the XML broker uses to communicate with the application server.

Pascal

```
property AppServer: IAppServer;
```

C++

```
__property IAppServer AppServer;
```

Description

Use AppServer to communicate with the providers that supply data to and resolve updates for the XML broker. If the providers are on a remote application server, AppServer has the same value as the AppServer property of the connection component. If the XML broker uses a provider in the same process space, AppServer is the interface to an internally created object that dispatches IAppServer calls to the local provider.

Applications don't need to work directly with the IAppServer interface. The properties and methods of the XML broker are automatically translated into IAppServer calls as needed.

See Also

HasAppServer (see page 7323)

AppServer

SetProvider (see page 7320)

78.1.1.5.2 TXMLBroker.Connected Property

Determines whether the XML broker has an active connection to the provider that supplies its data and applies its updates.

Pascal

```
property Connected: Boolean;
```

C++

```
__property Boolean Connected;
```

Description

Use Connected to determine whether the XML broker can supply XML data packets and apply updates. If the XML broker uses a provider on an application server, setting Connected to true sets the Connected property of the connection component to true. If the XML broker uses a local provider component, setting Connected to true causes the XML broker to generate an internal object that communicates with the provider using the IAppServer interface. Thus, the value of Connected indicates whether an IAppServer interface is available for communicating with a provider, and so the value of Connected is the same as the value of HasAppServer (see page 7323).

You can't set Connected to true unless the XML broker has a provider to which it can connect. Trying to set Connected to true before the ProviderName (see page 7326) property is set or a local provider is assigned using the SetProvider (see page 7320) method causes the XML broker to raise an exception.

Note: In the Web Page editor, default fields or parameter values may not appear if the associated XML broker is not connected.

See Also

HasAppServer (see page 7323)

AppServer (see page 7322)

Connected

TDataSetProvider

ProviderName (see page 7326)

SetProvider (see page 7320)

78.1.1.5.3 TXMLBroker.HasAppServer Property

Indicates whether the XML broker has an IAppServer interface for communicating with its provider.

Pascal

```
property HasAppServer: Boolean;
```

C++

```
__property Boolean HasAppServer;
```

Description

Read (see page 3634) HasAppServer to determine whether the XML broker can get its data from an IAppServer interface. When HasAppServer is true, the XML broker can interact with its provider through the IAppServer interface.

Use HasAppServer to check whether the XML broker is connected rather than reading the AppServer (see page 7322) property. Reading AppServer (see page 7322) causes the XML broker to obtain an IAppServer interface if it does not already have one.

See Also

IAppServer

AppServer (see page 7322)

Connected (see page 7322)

78.1.1.5.4 TXMLBroker.MaxErrors Property

Specifies the maximum number of update errors allowed before aborting an update operations.

Pascal

```
property MaxErrors: Integer;
```

C++

```
__property int MaxErrors;
```

Description

Set MaxErrors to indicate the maximum number of errors that the associated provider should allow before prematurely stopping an update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

See Also

AS_ApplyUpdates

ReconcileProducer (see page 7327)

ApplyXMLUpdates (see page 7308)

78.1.1.5.5 TXMLBroker.MaxRecords Property

Specifies the maximum number of records that can appear in XML data packets.

Pascal

```
property MaxRecords: Integer;
```

C++

```
__property int MaxRecords;
```

Description

Set MaxRecords to limit the size of XML data packets retrieved from the associated provider.

When MaxRecords is –1, XML data packets contain all the records in the provider's dataset.

When MaxRecords is 0, XML data packets contain only metadata.

When MaxRecords is greater than 0, XML data packets contain up to MaxRecords records, depending on the number of records in the provider's dataset.

Tip: Because an InternetExpress application sends the entire data packet to client applications, it is important to limit the size of data packets. Otherwise, the time to download web pages containing data packets can be prohibitively long.

Note: When limiting the number of records that appear in data packets, you may need to fetch several data packets before all data is returned. Because InternetExpress applications are always stateless, you will need to supply information about what record should start the next data packet as the OwnerData parameter of GetXMLRecords (see page 7314) or RequestRecords (see page 7316). You can use a query form to obtain information about the last record seen by a particular Web client.

See Also

AS_GetRecords

GetXMLRecords (see page 7314)

RequestRecords (see page 7316)

TQueryForm

Using an XML Broker

78.1.1.5.6 TXMLBroker.Notify Property

Lists the objects that are notified when any properties affecting the XML broker's connection to its provider change.

Pascal

```
property Notify [Index: Integer]: TObject;
```

C++

```
__property TObject * Notify[int Index];
```

Description

Read (see page 3634) Notify to access the objects that respond when the XML broker's connection to its provider changes. These objects are sent a notification when the ProviderName (see page 7326), AppServer (see page 7322), or RemoteServer (see page 7328) property changes.

Most applications do not need to use the Notify property. It allows the Web page editor to remain synchronized with property changes made at design time.

See Also

NotifyCount (see page 7325)

AddNotify (see page 7307)

RemoveNotify (see page 7316)

RemoteServer (see page 7328)

AppServer (see page 7322)

ProviderName (see page 7326)

78.1.1.5.7 TXMLBroker.NotifyCount Property

Indicates the number of objects in the Notify (see page 7325) property list.

Pascal

```
property NotifyCount: Integer;
```

C++

```
__property int NotifyCount;
```

Description

NotifyCount specifies the number of objects listed in the Notify (see page 7325) property.

See Also

Notify (see page 7325)

AddNotify (see page 7307)

RemoveNotify (see page 7316)

78.1.1.5.8 TXMLBroker.Params Property

Contains parameter values that are sent to or retrieved from the application server.

Pascal

```
property Params: TXMLParams;
```

C++

```
__property TXMLParams Params;
```

Description

Use Params to specify parameter values that should be passed to a source dataset on the application server. These parameter values are used in one of two ways:

If the associated provider represents a query or stored procedure, they supply parameter values that the application server uses when executing the query or stored procedure.

If the associated provider represents a table, they limit the records that are sent in data packets so that they only include records where each field with the same name as a parameter has the same value as the parameter.

After the application specifies parameters (at design time or runtime), they are sent to the application server automatically when the XML broker fetches data.

To send parameter values to a query or stored procedure, add a parameter object (TParam) to Params for every parameter on the query or stored procedure for which you want to supply a value. Assign each parameter object the same Name, DataType, and ParamType as the corresponding parameter on the query or stored procedure. These values are automatically encoded and sent to the application server when the XML broker fetches data.

To send parameter values to a table, add a parameter object for each field in the table that is used to limit the values sent in data packets. Assign each parameter object the same Name and DataType as the corresponding field component on the table. These parameter values are automatically sent when the client dataset fetches data, and the records returned in data packets from the application server only contain records that match the assigned value on the corresponding fields.

Tip: You can assign parameter values from the content fields supplied to a request message by TSubmitQueryButton using the AssignStrings method of Params.

Note: To initialize Params so that it reflects the parameters of a query or stored procedure on an application server, call the FetchParams (see page 7310) method.

See Also

Params

Params

FetchParams (see page 7310)

TSubmitQueryButton

AssignStrings (see page 7335)

Using an XML Broker

78.1.1.5.9 TXMLBroker.ProviderName Property

Specifies the name of a registered provider from the application server.

Pascal

```
property ProviderName: string;
```

C++

```
__property UnicodeString ProviderName;
```

Description

Use ProviderName to specify the provider with which to communicate on the server (or in the same Web module). This property is passed as a parameter to IAppServer interface calls when the XML broker fetches data or applies updates. If an application server contains multiple provider objects, an XML broker can set ProviderName to any of them once the connection component establishes a connection to the application server.

ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer (see page 7328) property is set.

Note: ProviderName can also be used to link the XML broker to a provider in the same process space, as long as the provider component has the same Owner as the XML broker. If the local provider component has a different Owner, use the SetProvider (see page 7320) method instead.

See Also

RemoteServer (see page 7328)

TDataSetProvider

IAppServer

SetProvider (see page 7320)

Using an XML Broker

78.1.1.5.10 TXMLBroker.ReconcileProducer Property

Specifies the producer that generates response message content when an update generates errors.

Pascal

```
property ReconcileProducer: TCustomContentProducer;
```

C++

```
__property TCustomContentProducer ReconcileProducer;
```

Description

Use ReconcileProducer to specify the component that generates the content of response messages to update requests when the update operation generated errors.

When the XML broker applies updates, the application server returns a delta packet containing all records that could not be updated because of errors. The XML broker can attempt to respond to those errors or generate its own response message using the OnGetErrorResponse (see page 7330) event. However, unless the OnGetErrorResponse (see page 7330) event handler generates an HTML document to be sent in the response message and indicates that the update is completely handled, the XML broker must still generate a response to the update operation. To do so, it calls the Content method of the ReconcileProducer to generate the content of the response message.

Tip: The Reconcile producer will probably need information about the update errors that the application server returned. You can provide it with this information either by initializing the Reconcile producer in an OnGetErrorResponse (see page 7330) event handler or by having the Reconcile producer call the GetErrors (see page 7312) method.

Note: If there is no ReconcileProducer assigned and an update generates errors, the XML broker raises an exception unless the OnGetErrorResponse (see page 7330) event handler generates a response message.

Tip: The InetXCustom demo contains a TReconcilePageProducer component that you can use as the value of the ReconcileProducer property or as the basis of your own reconcile producer.

See Also

MaxErrors (see page 7323)

OnGetErrorResponse (see page 7330)

ApplyXMLUpdates (see page 7308)

GetErrors (see page 7312)

Using an XML Broker

78.1.1.5.11 TXMLBroker.RemoteServer Property

Specifies the connection component that this XML broker uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

Use RemoteServer to specify the connection component that establishes a connection to the remote application server. The connection component obtains an IAppServer interface for data exchange and maintains the interface of the remote data module on the application server.

If a Web application uses more than connection component, the application can change which connection an XML broker uses by changing the value for this property.

See Also

ProviderName (see page 7326)

TSocketConnection (see page 221)

TDCOMConnection

TWebConnection (see page 254)

AppServer

IAppServer

Using an XML Broker

78.1.1.5.12 TXMLBroker.WebDispatch Property

Lets the Web dispatcher identify what HTTP messages should be forwarded to the XML broker.

Pascal

```
property WebDispatch: TWebDispatch;
```

C++

```
__property TWebDispatch WebDispatch;
```

Description

Use WebDispatch to specify the HTTP messages to which the XML broker responds. Typically, these originate from a button added to HTML forms by TApplyUpdatesButton.

The TWebDispatch object that implements WebDispatch includes properties that let you specify the path portion of URLs to which the XML broker responds, as well as the types of messages it accepts (typically post messages). You can also use this property to disable the XML broker so that the Web dispatcher does not send it any HTTP messages. When the XML broker is disabled in this way, it can still provide XML data packets to an InternetExpress page producer, but any update messages must be handled by a Web action item.

See Also

TWebModule

TWebDispatcher

TApplyUpdatesButton

Using an XML Broker

78.1.1.6 TXMLBroker Events

The events of the TXMLBroker class are listed here.

Published Events

	Name	Description
	AfterDispatch (see page 7329)	Occurs after the XML broker generates the content of a response message.
	BeforeDispatch (see page 7330)	Occurs when the XML broker receives an HTTP request message.
	OnGetErrorResponse (see page 7330)	Occurs when there are errors applying updates to the application server.
	OnGetResponse (see page 7331)	Occurs when there are no errors applying updates to the application server.
	OnRequestRecords (see page 7332)	Occurs when the XML broker needs to fetch a data packet from its associated provider.
	OnRequestUpdate (see page 7332)	Occurs immediately before the XML broker applies updates to the application server.

78.1.1.6.1 TXMLBroker.AfterDispatch Event

Occurs after the XML broker generates the content of a response message.

Pascal

```
property AfterDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent AfterDispatch;
```

Description

Write (see page 3707) an AfterDispatch event handler to perform any final actions before sending a response back to the Web browser that sent the XML broker an update message. Usually by the time AfterDispatch occurs, the XML broker has already applied the updates and generated the response message (using the ReconcileProducer (see page 7327) or one of the events that occurs when the XML broker receives an update request).

See Also

WebDispatch (see page 7328)

BeforeDispatch (see page 7330)

OnRequestUpdate (see page 7332)

OnGetResponse (see page 7331)

OnGetErrorResponse (see page 7330)

ReconcileProducer (see page 7327)

Using an XML Broker

78.1.1.6.2 TXMLBroker.BeforeDispatch Event

Occurs when the XML broker receives an HTTP request message.

Pascal

```
property BeforeDispatch: THTTPMethodEvent;
```

C++

```
__property THTTPMethodEvent BeforeDispatch;
```

Description

Write (see page 3707) a BeforeDispatch event handler to initialize the response to an update request or to handle any messages sent to the XML broker that are not update requests. If the event handler completely handles the request, filling out the response object so that it is ready to send, it should set the Handled parameter to true, thereby preventing the XML broker from applying the updates.

Unless the BeforeDispatch event handler sets the Handled parameter to true, the XML broker receives an OnRequestUpdate (see page 7332) event after the BeforeDispatch event handler exits. After the OnRequestUpdate (see page 7332) event, the XML broker applies the updates to the application server and generates the contents of the response based on whether there were any update errors. Finally, the XML broker receives an AfterDispatch (see page 7329) event where it can perform any final changes to the response message before it is sent back to the Web browser.

If the BeforeDispatch event handler sets the Handled parameter to true, the XML broker receives an AfterDispatch (see page 7329) event, but does not attempt to apply updates from the request message or receive any of the events that accompany that process.

See Also

WebDispatch (see page 7328)

AfterDispatch (see page 7329)

OnRequestUpdate (see page 7332)

Using an XML Broker

78.1.1.6.3 TXMLBroker.OnGetErrorResponse Event

Occurs when there are errors applying updates to the application server.

Pascal

```
property OnGetErrorResponse: TGetErrorResponseEvent;
```

C++

```
__property TGetErrorResponseEvent OnGetErrorResponse;
```

Description

Write (see page 3707) an OnGetErrorResponse event handler to generate the response message after an update request

encountered errors or to initialize a ReconcileProducer (see page 7327) before it generates the response.

The Sender parameter is the XML broker that applied updates to the application server and received back a delta packet of update errors.

ErrorCount is the number of update errors found by the application server before it halted the update operation.

XMLErrors is a string of XML returned by the application server. It is a delta packet containing all the updates that could not be applied.

Request is the original request message that contains the delta packet of updates from the Web browser.

Response is the emerging response message that will be sent to the Web browser in response to the update request. Fill in the properties of this object to generate a response (such as an error message or a reconcile form) that indicates the errors in XMLErrors.

Handled indicates whether the event handler completed filling out Response. Set Handled to true if the event handler completes the response for the Web browser. Leave Handled as false if ReconcileProducer (see page 7327) should generate the content of Response.

Note: If the OnGetErrorResponse event handler does not set Handled to true and there is no ReconcileProducer (see page 7327), the XML broker raises an exception.

See Also

MaxErrors (see page 7323)

ReconcileProducer (see page 7327)

OnGetResponse (see page 7331)

ApplyXMLUpdates (see page 7308)

Using an XML Broker

78.1.1.6.4 TXMLBroker.OnGetResponse Event

Occurs when there are no errors applying updates to the application server.

Pascal

```
property OnGetResponse: TGetResponseEvent;
```

C++

```
__property TGetResponseEvent OnGetResponse;
```

Description

Write (see page 3707) an OnGetResponse event handler to edit the response message after an update request is successfully applied.

The Sender parameter is the XML broker that applied updates to the application server with no errors.

Request is the original request message that contains the delta packet of updates from the Web browser.

Response is the emerging response message that will be sent to the Web browser in response to the update request. Fill in the properties of this object to generate a response (such as refreshing the data).

Handled indicates whether the event handler completed filling out Response. Set Handled to true if the event handler completes the response for the Web browser. Leave Handled as false if the XML broker should send a redirect response that sends the browser back to the page that generated the update request.

See Also

OnGetErrorResponse (see page 7330)

ApplyXMLUpdates (see page 7308)

Using an XML Broker

78.1.1.6.5 TXMLBroker.OnRequestRecords Event

Occurs when the XML broker needs to fetch a data packet from its associated provider.

Pascal

```
property OnRequestRecords: TRequestRecordsEvent;
```

C++

```
__property TRequestRecordsEvent OnRequestRecords;
```

Description

Write (see page 3707) an OnRequestRecords event handler to provide an alternate mechanism for providing XML data packets or to perform other actions whenever the XML broker needs to fetch an XML data packet. For example, the OnRequestRecords event handler might call GetXMLRecords (see page 7314) to fetch an XML data packet from the application server and then edit its contents.

OnRequestRecords occurs when the InternetExpress page producer requests an XML data packet (usually in response to a request message) or when the application calls the XML broker's RequestRecords (see page 7316) method.

The Sender parameter is the object that originates the request (usually an InternetExpress page producer).

Request is the request message that prompted the request for an XML data packet. This value may be nil (Delphi) or NULL (C++) if the request did not arise from a page producer or if the page producer is not processing a request message.

OwnerData contains an application-defined variant that is sent to the provider's BeforeGetRecords event handler. The InternetExpress page producer that is requesting records or the caller of RequestRecords (see page 7316) may initialize this value. OwnerData can be used to pass persistent state information to a stateless application server (for example, the first record that should appear in the data packet when MaxRecords (see page 7324) is greater than 0).

Records optionally returns an XML data packet as a string. If Records is an empty string on exit from the event handler, the XML broker automatically fetches the XML data packet from its associated provider. Otherwise, the XML broker uses the value of Records as the new XML data packet.

See Also

RequestRecords (see page 7316)

GetXMLRecords (see page 7314)

GetXMLData

BeforeGetRecords

MaxRecords (see page 7324)

Using an XML Broker

78.1.1.6.6 TXMLBroker.OnRequestUpdate Event

Occurs immediately before the XML broker applies updates to the application server.

Pascal

```
property OnRequestUpdate: TRequestUpdateEvent;
```

C++

```
__property TRequestUpdateEvent OnRequestUpdate;
```

Description

Write (see page 3707) an OnRequestUpdate event handler to provide an alternate mechanism for applying updates to the application server. For example, the OnRequestUpdate event handler might call GetDelta (see page 7311) to extract the delta packet from the request message and then preprocess the XML string before applying the updates with a call to ApplyXMLUpdates (see page 7308).

OnRequestUpdate occurs immediately after the BeforeDispatch (see page 7330) event or when the application calls the XML broker's RequestUpdate (see page 7317) method.

The Sender parameter is the object that received the update request (the XML broker or a Web action item).

Request is the request message that contains the updates to be applied. The XML delta packet can be extracted from this request object using the GetDelta (see page 7311) method.

Response is the emerging response message that will be sent back to the Web browser that generated the update request. The event handler can set properties on this response object to partially or completely specify the message will be returned to the Web browser.

Handled indicates whether the event handler applied the updates. If Handled is true on exit from the event handler, the XML broker does not apply the updates, does not receive an OnGetResponse (see page 7331) or OnGetErrorResponse (see page 7330) event, and does not call on the ReconcileProducer (see page 7327) to generate a response message. (It does still receive an AfterDispatch (see page 7329) event). If Handled is false on exit from the event handler, the XML broker proceeds to apply the updates contained in the request message and receives all the associated events.

Note: If the event handler applies the updates, it must also generate the response message.

See Also

BeforeDispatch (see page 7330)

ApplyXMLUpdates (see page 7308)

RequestUpdate (see page 7317)

GetDelta (see page 7311)

OnGetResponse (see page 7331)

OnGetErrorResponse (see page 7330)

ReconcileProducer (see page 7327)

AfterDispatch (see page 7329)

Using an XML Broker

78.1.2 TXMLParams Class

TXMLParams manages a list of parameters for an XML broker.

Class Hierarchy

Pascal

```
TXMLParams = class(TParams);
```

C++

```
class TXMLParams : public TParams;
```

File

XMLBrokr

Description

XML brokers use TXMLParams to manage a list of parameters that it passes to a provider on the application server. These parameters supply values to a query or stored procedure on the application server, or limit the records provided from a table on the application server.

Use the properties and methods of TXMLParams to:

Access a specific parameter.

Add or delete parameters from the list.

Get or set the values of individual parameters.

Assign parameter values from a list such as the one supplied in a request message from a Web browser.

Compare parameters.

Iterate through all parameters.

Copy (see page 3547) a set of parameters to another parameter list.

See Also

TParam

TXMLBroker (see page 7303)

78.1.2.1 TXMLParams Members

The following tables list the members exposed by TXMLParams.

Protected Methods

	Name	Description
	AssignTo (see page 7335)	Copies the properties of an object to a destination object.

Public Methods

	Name	Description
	AssignStrings (see page 7335)	Assigns values to the parameters from a list of name/value pairs.

78.1.2.2 TXMLParams Methods

The methods of the TXMLParams class are listed here.

Protected Methods

	Name	Description
	AssignTo (see page 7335)	Copies the properties of an object to a destination object.

Public Methods

	Name	Description
	AssignStrings (see page 7335)	Assigns values to the parameters from a list of name/value pairs.

78.1.2.2.1 TXMLParams.AssignStrings Method

Assigns values to the parameters from a list of name/value pairs.

Pascal

```
procedure AssignStrings(Value: TStrings);
```

C++

```
__fastcall AssignStrings(TStrings Value);
```

Description

Use AssignStrings to update the values of the parameters in the Items list to match the values in the list specified by Value. For each string in Value of the form Name=Value, AssignStrings locates the parameter (if any) that has the same name as the name portion of the string. It then assigns the Value portion of the string as the parameter's value.

AssignStrings makes it very convenient to assign parameter values that are received as the content of a request message. In particular, TSubmitQueryButton generates an HTML control that sends the values in an HTML query form to the Web application. If the query form displays parameter values that should be sent to a provider on the application server, you can call AssignStrings with the ContentFields property of that Web request message.

See Also

Items

ParamValues

ContentFields

TQueryForm

TSubmitQueryButton

78.1.2.2.2 TXMLParams.AssignTo Method

Copies the properties of an object to a destination object.

Pascal

```
procedure AssignTo(Dest: TPersistent); override;
```

C++

```
virtual __fastcall AssignTo(TPersistent * Dest);
```

Description

Override the AssignTo method to extend the functionality of the Assign method of destination objects so that they handle newly created object classes. When defining a new object class, override the Assign method for every existing object class that should be able to copy its properties to the new class. Override the AssignTo method for every existing class to which the new class can copy.

The Assign method of TPersistent calls AssignTo if the descendant object does not succeed in copying the properties of a source object. The AssignTo method defined by TPersistent raises an EConvertError exception.

For example, given the following code in which A and B are instance variables:

```
A.Assign(B); {Delphi}
A->Assign(B); // C++
```

if A knows how to handle B, then it does so and returns. If A doesn't know how to handle B's type, execution will trickle to the TPersistent version of Assign, which calls:

```
B.AssignTo(A); {Delphi}
B->AssignTo(A); // C++
```

If B knows how to copy to A, the assignment succeeds. Otherwise, TPersistent raises an exception.

See Also

Assign

EConvertError (see page 3919)

78.2 Interfaces

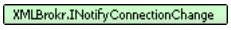
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	INotifyConnectionChange (see page 7337)	This is class XMLBrokr.INotifyConnectionChange.

78.2.1 INotifyConnectionChange Interface

Class Hierarchy



Pascal

```
INotifyConnectionChange = interface;
```

C++

```
__interface INotifyConnectionChange;
```

File

XMLBrokr

Description

This is class XMLBrokr.INotifyConnectionChange.

78.2.1.1 INotifyConnectionChange Members

The following tables list the members exposed by INotifyConnectionChange.

Public Methods

	Name	Description
	ConnectionChange (see page 7338)	This is ConnectionChange, a member of class INotifyConnectionChange.

78.2.1.2 INotifyConnectionChange Methods

The methods of the INotifyConnectionChange class are listed here.

Public Methods

	Name	Description
	ConnectionChange (see page 7338)	This is ConnectionChange, a member of class INotifyConnectionChange.

78.2.1.2.1 INotifyConnectionChange.ConnectionChange Method

Pascal

```
procedure ConnectionChange(Sender: TComponent; Connected: Boolean);
```

C++

```
__fastcall ConnectionChange(TComponent * Sender, Boolean Connected);
```

Description

This is ConnectionChange, a member of class INotifyConnectionChange.

78.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	FormatXML (see page 7339)	This is function XMLBrokr.FormatXML.

78.3.1 XMLBrokr.FormatXML Function

Pascal

```
function FormatXML(const Value: string; XMLOptions: TXMLOptions): string;
```

C++

```
UnicodeString FormatXML(const UnicodeString Value, TXMLOptions XMLOptions);
```

File

XMLBrokr

Description

This is function XMLBrokr.FormatXML.

78.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TXMLOption (see page 7340)	TXMLOptions (see page 7342) is a set of flags that indicates how database information should be formatted in an XML data packet.

78.4.1 XMLBrokr.TXMLOption Enumeration

TXMLOptions (see page 7342) is a set of flags that indicates how database information should be formatted in an XML data packet.

Pascal

```
TXMLOption = (
  xoQuote
);
```

C++

```
enum TXMLOption {
  xoQuote
};
```

File

XMLBrokr

Description

TXMLOption is the type for flags that influence how an XML broker formats the XML string that an application server provides. After the XML broker fetches data from an application server, it reformats the data using a set of TXMLOption flags.

TXMLOptions (see page 7342) is a set of zero or more TXMLOption flags.

The following table lists the TXMLOption flags:

Value	Meaning
xoQuote	The XML string is divided into readable lines, each of which is enclosed in quotation marks.

See Also

TBaseInetXPageProducer

TXMLBroker (see page 7303)

IAppServer

78.5 Types

The following table lists types in this documentation.

Types

Name	Description
TGetErrorResponseEvent (see page 7341)	This is type XMLBrokr.TGetErrorResponseEvent.
TGetResponseEvent (see page 7341)	This is type XMLBrokr.TGetResponseEvent.
TRequestRecordsEvent (see page 7342)	This is type XMLBrokr.TRequestRecordsEvent.
TRequestUpdateEvent (see page 7342)	This is type XMLBrokr.TRequestUpdateEvent.
TXMLOptions (see page 7342)	This is type XMLBrokr.TXMLOptions.

78.5.1 XMLBrokr.TGetErrorResponseEvent Type

Pascal

```
TGetErrorResponseEvent = procedure (Sender: TObject; ErrorCount: Integer; XMLErrors: string;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; ErrorCount: Integer; XMLErrors: string; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean) ( TGetErrorResponseEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TGetErrorResponseEvent.

78.5.2 XMLBrokr.TGetResponseEvent Type

Pascal

```
TGetResponseEvent = procedure (Sender: TObject;Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean) of object;
```

C++

```
(Sender: TObject;Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
TGetResponseEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TGetResponseEvent.

78.5.3 XMLBrokr.TRequestRecordsEvent Type

Pascal

```
TRequestRecordsEvent = procedure (Sender: TObject; Request: TWebRequest; out RecCount:
Integer; var OwnerData: OleVariant; var Records: string) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; out RecCount: Integer; var OwnerData: OleVariant; var
Records: string) ( TRequestRecordsEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TRequestRecordsEvent.

78.5.4 XMLBrokr.TRequestUpdateEvent Type

Pascal

```
TRequestUpdateEvent = procedure (Sender: TObject; Request: TWebRequest; Response:
TWebResponse; var Handled: Boolean) of object;
```

C++

```
(Sender: TObject; Request: TWebRequest; Response: TWebResponse; var Handled: Boolean) (
TRequestUpdateEvent)();
```

File

XMLBrokr

Description

This is type XMLBrokr.TRequestUpdateEvent.

78.5.5 XMLBrokr.TXMLOptions Type

Pascal

```
TXMLOptions = set of TXMLOption;
```

C++

```
TXMLOption TXMLOptions;
```

File

XMLBrokr

Description

This is type XMLBrokr.TXMLOptions.

78.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
sPostDelta (see page 7343)	This is constant XMLBrokr.sPostDelta.
sProducer (see page 7343)	This is constant XMLBrokr.sProducer.
sRedirect (see page 7343)	This is constant XMLBrokr.sRedirect.

78.6.1 XMLBrokr.sPostDelta Constant

Pascal

```
sPostDelta = 'postdelta';
```

C++

```
sPostDelta = 'postdelta';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sPostDelta.

78.6.2 XMLBrokr.sProducer Constant

Pascal

```
sProducer = 'PRODUCER';
```

C++

```
sProducer = 'PRODUCER';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sProducer.

78.6.3 XMLBrokr.sRedirect Constant

Pascal

```
sRedirect = 'REDIRECT';
```

C++

```
sRedirect = 'REDIRECT';
```

File

XMLBrokr

Description

This is constant XMLBrokr.sRedirect.

79 XMLDoc Namespace

This is namespace XMLDoc.

Topics

Name	Description
Classes (see page 7347)	The following table lists classes in this documentation.
Interfaces (see page 7478)	The following table lists interfaces in this documentation.
Functions (see page 7488)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7494)	The following table lists structs, records, enums in this documentation.
Types (see page 7498)	The following table lists types in this documentation.
Variables (see page 7502)	The following table lists variables in this documentation.

Classes

	Name	Description
	TXMLDocument (see page 7347)	TXMLDocument represents an XML (see page 7392) document.
	TXMLNode (see page 7396)	TXMLNode represents a node in an XML document.
	TXMLNodeCollection (see page 7444)	TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.
	TXMLNodeList (see page 7460)	TXMLNodeList represents a set of nodes in an XML document.

Functions

	Name	Description
	CheckEncoding (see page 7488)	This is function XMLDoc.CheckEncoding.
	CreateDOMNode (see page 7488)	Creates a new DOM node and returns its interface.
	DetectCharEncoding (see page 7489)	This is function XMLDoc.DetectCharEncoding.
	FormatXMLData (see page 7490)	Formats a string of XML so that it is more readable.
	LoadXMLData (see page 7490)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLData (see page 7490)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLDocument (see page 7491)	Returns the interface for an XML document given the name of an XML file.
	NewXMLDocument (see page 7491)	Returns the interface for a new, empty XML document.
	XMLDocError (see page 7492)	Raises an EXMLDocError (see page 7639) exception.
	XMLDocError (see page 7492)	Raises an EXMLDocError (see page 7639) exception.
	XMLStringToUnicodeString (see page 7492)	This is function XMLDoc.XMLStringToUnicodeString.
	XMLStringToWideString (see page 7492)	This is function XMLDoc.XMLStringToWideString.

Interfaces

	Name	Description
	IXMLDocumentAccess (see page 7478)	This is class XMLDoc.IXMLDocumentAccess.
	IXMLNodeAccess (see page 7480)	This is class XMLDoc.IXMLNodeAccess.

Structs, Records, Enums

	Name	Description
	TNodeChange (see page 7494)	TNodeChange indicates the type of change made to a node in an XML document.
	TNodeClassInfo (see page 7495)	TNodeClassInfo and TNodeClassArray (see page 7499) represent information about the classes that implement XML node interfaces.
	TNodeListOperation (see page 7495)	TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList (see page 7703) interface.
	TXMLDocumentSource (see page 7496)	This is record XMLDoc.TXMLDocumentSource.
	TXMLPrologItem (see page 7497)	This is record XMLDoc.TXMLPrologItem.

Types

Name	Description
TNodeChangeEvent (see page 7498)	TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.
TNodeClassArray (see page 7499)	This is type XMLDoc.TNodeClassArray.
TNodeListNotification (see page 7499)	TNodeListNotification is the type of the callback the TXMLNodeList (see page 7460) calls when it adds, deletes, or creates XML nodes.
TXMLNodeArray (see page 7500)	TXMLNodeArray is a dynamic array of TXMLNode (see page 7396) objects.
TXMLNodeClass (see page 7501)	TXMLNodeClass is a class reference to a TXMLNode (see page 7396) descendant.
TXMLNodeCollectionClass (see page 7501)	TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 7444) descendant.

Variables

Name	Description
UseXSDBooleanStrings (see page 7502)	This is variable XMLDoc.UseXSDBooleanStrings.

79.1 Classes

The following table lists classes in this documentation.

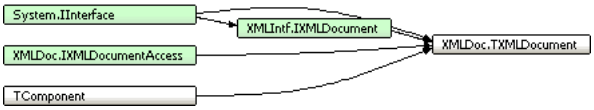
Classes

	Name	Description
	TXMLDocument (see page 7347)	TXMLDocument represents an XML (see page 7392) document.
	TXMLNode (see page 7396)	TXMLNode represents a node in an XML document.
	TXMLNodeCollection (see page 7444)	TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.
	TXMLNodeList (see page 7460)	TXMLNodeList represents a set of nodes in an XML document.

79.1.1 TXMLDocument Class

TXMLDocument represents an XML (see page 7392) document.

Class Hierarchy



Pascal

```
TXMLDocument = class(TComponent, IInterface, IXMLDocument, IXMLDocumentAccess);
```

C++

```
class TXMLDocument : public TComponent, public IInterface, public IXMLDocument, public
IXMLDocumentAccess;
```

File

XMLDoc

Description

Use TXMLDocument to represent an XML (see page 7392) document. TXMLDocument can read an existing XML (see page 7392) document from a file, it can be associated with an in-memory string that is the contents of an XML (see page 7392) document, or it can create a new, empty XML (see page 7392) document.

You can use TXMLDocument directly to load an XML (see page 7392) document, read and edit it, and save any changes. You can also use TXMLDocument to access the objects generated by the XML (see page 7392) Data Binding wizard.

TXMLDocument uses an external DOM (Document Object Model) parser to analyze the XML (see page 7392) document. This parser is usually assigned by specifying the DOMVendor (see page 7386) property.

When you set the Active (see page 7381) property to true, TXMLDocument uses the DOM parser to parse the XML (see page 7392) document so that you can examine or modify it. In particular, the DocumentElement (see page 7383) property provides an interface to the root node of the document. You can use that interface to access its child nodes and to add or delete child nodes.

TXMLDocument implements the IXMLDocument (see page 7640) interface. Applications obtain this interface from TXMLDocument or from the nodes in the document.

When TXMLDocument is created without an Owner, it behaves like an interfaced object. That is, when all references to its interface are released, the TXMLDocument instance is automatically freed. When TXMLDocument is created with an Owner, however, it behaves like any other component, and is freed by its Owner. When you add a TXMLDocument component from the component palette to a form or data module, it is automatically created with an Owner. When the TXMLDocument component is created using the global LoadXMLDocument (see page 7491) function (or by a function that the XML (see page 7392) Data Binding wizard generates to return the root node of the document), the function creates a TXMLDocument instance without an Owner.

See Also

IXMLNode (see page 7669)

EXMLDocError (see page 7639)

IXMLDocument (see page 7640)

LoadXMLDocument (see page 7491)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

79.1.1.1 TXMLDocument Members

The following tables list the members exposed by TXMLDocument.

Protected Methods

	Name	Description
	_AddRef (see page 7354)	Called when an application uses a component interface.
	_Release (see page 7354)	Called when an application releases a component interface.
	AssignParseOptions (see page 7356)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7356)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7356)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7356)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7358)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7359)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7359)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7359)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7360)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7360)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7360)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7361)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7361)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7361)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7362)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7362)	Returns the value of the Node property.
	GetDocumentObject (see page 7363)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7363)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7363)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7363)	Returns the value of the Encoding property.
	GetFileName (see page 7364)	Returns the value of the FileName property.
	GetModified (see page 7364)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7364)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7365)	Returns the value of the Options property.
	GetParseOptions (see page 7365)	Returns the value of the ParseOptions property.

	GetPrologNode (see page 7365)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7366)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7366)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7366)	Returns the value of the StandAlone property.
	GetVersion (see page 7366)	Returns the value of the Version property.
	GetXML (see page 7367)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7367)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7368)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7368)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7371)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7372)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7374)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7375)	Saves the XML (see page 7392) document to an internal string list.
V	SetActive (see page 7375)	Sets the value of the Active property.
	SetDocumentElement (see page 7375)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7376)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7376)	Sets the value of the Encoding property.
	SetFileName (see page 7376)	Sets the value of the FileName property.
	SetModified (see page 7377)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7377)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7377)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7378)	Sets the value of the Options property.
	SetParseOptions (see page 7378)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7378)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7378)	Sets the value of the StandAlone property.
	SetVersion (see page 7379)	Sets the value of the Version property.
	SetXML (see page 7379)	Sets the value of the XML property.
	SetXMLStrings (see page 7379)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7380)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7380)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7383)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7383)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7385)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7386)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7391)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	AddChild (see page 7354)	This is the overview for the AddChild method overload.
	AddChild (see page 7643)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7355)	Responds after the last constructor has executed.
	Create (see page 7351)	This is the overview for the Create constructor overload.
	CreateElement (see page 7357)	Creates a new element node that is associated with this document.
	CreateNode (see page 7357)	Creates a new node that is associated with this document.
V	Destroy (see page 7352)	Disposes of a TXMLDocument (see page 7347) object.
	GeneratePrefix (see page 7360)	Generates a unique namespace prefix that can be used for a node in the document.
	GetDocBinding (see page 7362)	Replaces the document element with an instance of a specified implementation class.
	IsEmptyDoc (see page 7367)	Indicates whether the TXMLDocument (see page 7347) instance represents an empty document.
	LoadFromFile (see page 7368)	Loads an XML (see page 7392) document and activates it.

	LoadFromStream (see page 7369)	Loads an XML (see page 7392) document from a stream and activates it.
	LoadFromXML (see page 7370)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
V	NewInstance (see page 7370)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7371)	Updates the parsed XML (see page 7392) document to reflect external changes.
	RegisterDocBinding (see page 7372)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7372)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7373)	Saves the XML (see page 7392) document to disk.
	SaveToStream (see page 7373)	Saves the XML (see page 7392) document to a stream.
	SaveToXML (see page 7374)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.

Public Properties

	Name	Description
R	AsyncLoadState (see page 7382)	Indicates the state of the DOM parser when it parses the XML (see page 7392) document asynchronously.
R	ChildNodes (see page 7383)	Lists all of the child nodes of the document.
	DocumentElement (see page 7383)	Provides access to the root node of the XML (see page 7392) document.
R	DocumentObject (see page 7479)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7384)	Provides access to the DOM interface for the parsed XML (see page 7392) document.
	DOMImplementation (see page 7385)	Provides access to the DOM interface for document-independent functions.
	Encoding (see page 7386)	Indicates the character set that the XML (see page 7392) document uses to encode characters.
R	Modified (see page 7387)	Indicates whether the XML (see page 7392) document has been modified.
R	Node (see page 7388)	Provides access to the document node for the XML (see page 7392) document.
	NSPrefixBase (see page 7389)	Specifies the base string for automatically-generated namespace prefixes.
R	SchemaRef (see page 7391)	Returns the name of the schema associated with this XML (see page 7392) document.
	StandAlone (see page 7391)	Indicates whether the XML (see page 7392) document includes any external markup declarations.
	Version (see page 7391)	Indicates the XML (see page 7392) version that the document uses.

Published Events

	Name	Description
	AfterClose (see page 7393)	Occurs after the XML (see page 7392) document is closed.
	AfterNodeChange (see page 7393)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7394)	Occurs immediately after the XML (see page 7392) document is opened.
	BeforeClose (see page 7394)	Occurs immediately before the XML (see page 7392) document is closed.
	BeforeNodeChange (see page 7394)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7395)	Occurs immediately before the XML (see page 7392) document is opened.
	OnAsyncLoad (see page 7395)	Occurs at various stages during the asynchronous parsing of an XML (see page 7392) document.

Published Properties

	Name	Description
	Active (see page 7381)	Specifies whether the XML (see page 7392) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7386)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7392) document.
	FileName (see page 7387)	Specifies the file name for the XML (see page 7392) document associated with this TXMLDocument (see page 7347) object.
	NodeIndentStr (see page 7388)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7392) text.

	Options (see page 7389)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7390)	Specifies implementation-specific options that are followed when parsing the XML (see page 7392) document.
	XML (see page 7392)	Specifies the content of the XML document associated with this TXMLDocument (see page 7347) object.

79.1.1.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TXMLDocument.Create (DOMString) (see page 7351)	Instantiates a TXMLDocument (see page 7347) object.
	TXMLDocument.Create (TComponent) (see page 7351)	Instantiates a TXMLDocument (see page 7347) object.

79.1.1.2.1 TXMLDocument.Create Constructor (DOMString)

Instantiates a TXMLDocument (see page 7347) object.

Pascal

```
constructor Create(AOwner: TComponent); override; overload;
constructor Create(const AFileName: DOMString); overload; reintroduce;
```

C++

```
virtual __fastcall TXMLDocument(TComponent * AOwner);
__fastcall TXMLDocument(const DOMString AFileName);
```

Description

Call Create to instantiate a TXMLDocument (see page 7347) component at runtime. You do not need to explicitly create TXMLDocument (see page 7347) components that are added to forms or data modules at design time: they are created automatically.

AFileName specifies the file that the new TXMLDocument (see page 7347) instance represents.

AOwner is another component that is responsible for freeing the TXMLDocument (see page 7347) instance. It becomes the value of the Owner property.

TXMLDocument (see page 7347) components that are created at runtime without an owner are freed automatically when all references to their IXMLDocument (see page 7640) interface are released. This differs from TXMLDocument (see page 7347) components added to a form or data module at design time or those created with an owner. When added to a form or data module at design time, the object is explicitly freed by its Owner (the form or data module).

See Also

FileName (see page 7387)

IXMLDocument (see page 7640)

Owner

79.1.1.3 TXMLDocument.Destroy Destructor

Disposes of a TXMLDocument (see page 7347) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLDocument();
```

Description

It is not necessary to explicitly free a TXMLDocument (see page 7347) component. If the component is created with an Owner, that Owner handles the task of freeing the document. If the component is created without an Owner, it is freed automatically when the reference count on its IXMLDocument (see page 7640) interface drops to zero.

See Also

Create (see page 7351)

79.1.1.4 TXMLDocument Methods

The methods of the TXMLDocument class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 7354)	Called when an application uses a component interface.
	_Release (see page 7354)	Called when an application releases a component interface.
	AssignParseOptions (see page 7356)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7356)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7356)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7356)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7358)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7359)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7359)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7359)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7360)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7360)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7360)	Returns the value of the Active property.
	GetAsyncLoadState (see page 7361)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7361)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7361)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7362)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7362)	Returns the value of the Node property.
	GetDocumentObject (see page 7363)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7363)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7363)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7363)	Returns the value of the Encoding property.
	GetFileName (see page 7364)	Returns the value of the FileName property.
	GetModified (see page 7364)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7364)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7365)	Returns the value of the Options property.
	GetParseOptions (see page 7365)	Returns the value of the ParseOptions property.

	GetPrologNode (see page 7365)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7366)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7366)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7366)	Returns the value of the StandAlone property.
	GetVersion (see page 7366)	Returns the value of the Version property.
	GetXML (see page 7367)	Returns the value of the XML property.
	InternalSetPrologValue (see page 7367)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7368)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7368)	Initializes the component after the form file has been read into memory.
	ReadDOMVendor (see page 7371)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7372)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7374)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7375)	Saves the XML (see page 7392) document to an internal string list.
V	SetActive (see page 7375)	Sets the value of the Active property.
	SetDocumentElement (see page 7375)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7376)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7376)	Sets the value of the Encoding property.
	SetFileName (see page 7376)	Sets the value of the FileName property.
	SetModified (see page 7377)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7377)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7377)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7378)	Sets the value of the Options property.
	SetParseOptions (see page 7378)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7378)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7378)	Sets the value of the StandAlone property.
	SetVersion (see page 7379)	Sets the value of the Version property.
	SetXML (see page 7379)	Sets the value of the XML property.
	SetXMLStrings (see page 7379)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7380)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7380)	This is XMLStringsChanging, a member of class TXMLDocument.

Public Methods

	Name	Description
	AddChild (see page 7354)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7355)	Responds after the last constructor has executed.
	CreateElement (see page 7357)	Creates a new element node that is associated with this document.
	CreateNode (see page 7357)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7360)	Generates a unique namespace prefix that can be used for a node in the document.
	GetDocBinding (see page 7362)	Replaces the document element with an instance of a specified implementation class.
	IsEmptyDoc (see page 7367)	Indicates whether the TXMLDocument (see page 7347) instance represents an empty document.
	LoadFromFile (see page 7368)	Loads an XML (see page 7392) document and activates it.
	LoadFromStream (see page 7369)	Loads an XML (see page 7392) document from a stream and activates it.
	LoadFromXML (see page 7370)	This is the overview for the LoadFromXML method overload.
V	NewInstance (see page 7370)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	Refresh (see page 7371)	Updates the parsed XML (see page 7392) document to reflect external changes.
	RegisterDocBinding (see page 7372)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7372)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7373)	Saves the XML (see page 7392) document to disk.
	SaveToStream (see page 7373)	Saves the XML (see page 7392) document to a stream.
	SaveToXML (see page 7374)	This is the overview for the SaveToXML method overload.

79.1.1.4.1 TXMLDocument._AddRef Method

Called when an application uses a component interface.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef is a basic implementation of the IInterface method, _AddRef.

Note: If the component is a wrapper for a COM object, __AddRef calls the _AddRef method of that COM object, and returns the resulting reference count.

In all other cases, __AddRef simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __AddRef to implement reference counting.

See Also

QueryInterface

_Release

_AddRef (see page 3503)

79.1.1.4.2 TXMLDocument._Release Method

Called when an application releases a component interface.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release is a basic implementation of the IInterface method, _Release.

__Release returns the resulting value of the reference count for the component's interface.

Note:

In all other cases, __Release simply returns –1 and takes no action. This allows the component to implement interfaces where reference counting is not required. More sophisticated components should override __Release to implement reference counting.

See Also

QueryInterface

_AddRef

79.1.1.4.3 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	TXMLDocument.AddChild (DOMString) (see page 7355)	Creates and adds a node to the child nodes of this document.
	TXMLDocument.AddChild (DOMString, DOMString) (see page 7355)	Creates and adds a node to the child nodes of this document.

79.1.1.4.3.1 TXMLDocument.AddChild Method (DOMString)

Creates and adds a node to the child nodes of this document.

Pascal

```
function AddChild(const TagName: DOMString): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI);
```

Description

Call AddChild to add a new child element node to the document itself. The new node is added to the end of the list maintained by the ChildNodes (see page 7383) property.

TagName is the tag name for the new element node.

NamespaceURI identifies the namespace, if any, that includes the new node's definition.

AddChild returns the interface for the newly created node.

Note: Do not call AddChild to add a child to the document element of this document. When adding data to the XML (see page 7392) document, use the AddChild method of the document element or of the node in the hierarchy that should be the parent of the new node.

See Also

AddChild (see page 7672)

CreateNode (see page 7357)

CreateElement (see page 7357)

ChildNodes (see page 7383)

DocumentElement (see page 7383)

79.1.1.4.4 TXMLDocument.AfterConstruction Method

Responds after the last constructor has executed.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

System::TObject::AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly

in your applications.

The System::TObject::AfterConstruction method implemented in System::TObject does nothing. Override this method when creating a class that takes some action after the object is created. For example, TCustomForm overrides System::TObject::AfterConstruction to generate an OnCreate event.

See Also

System::TObject::BeforeDestruction (see page 3358)

OnCreate

System::TObject::Create (see page 3355)

79.1.1.4.5 TXMLDocument.AssignParseOptions Method

Pascal

```
procedure AssignParseOptions;
```

C++

```
__fastcall AssignParseOptions();
```

Description

This is AssignParseOptions, a member of class TXMLDocument.

79.1.1.4.6 TXMLDocument.CheckActive Method

Pascal

```
procedure CheckActive;
```

C++

```
__fastcall CheckActive();
```

Description

This is CheckActive, a member of class TXMLDocument.

79.1.1.4.7 TXMLDocument.CheckAutoSave Method

Pascal

```
procedure CheckAutoSave; dynamic;
```

C++

```
__fastcall CheckAutoSave();
```

Description

This is CheckAutoSave, a member of class TXMLDocument.

79.1.1.4.8 TXMLDocument.CheckDOM Method

Pascal

```
procedure CheckDOM; dynamic;
```

C++

```
__fastcall CheckDOM();
```

Description

This is CheckDOM, a member of class TXMLDocument.

79.1.1.4.9 TXMLDocument.CreateElement Method

Creates a new element node that is associated with this document.

Pascal

```
function CreateElement(const TagOrData: DOMString; const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateElement(const DOMString TagOrData, const DOMString NamespaceURI);
```

Description

Call CreateElement to create a new generic element node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7383) list of any node in the document (including this document's ChildNodes (see page 7383) property). It can also be assigned to the DocumentElement (see page 7383) property to replace the entire data hierarchy of this document.

TagOrData provides the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

CreateElement returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7355) method.

See Also

AddChild (see page 7672)

ChildNodes (see page 7383)

DocumentElement (see page 7383)

AddChild (see page 7355)

CreateNode (see page 7357)

79.1.1.4.10 TXMLDocument.CreateNode Method

Creates a new node that is associated with this document.

Pascal

```
function CreateNode(const NameOrData: DOMString; NodeType: TNodeType = ntElement; const
AddlData: DOMString = ''): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNode(const DOMString NameOrData, TNodeType NodeType = ntElement,
const DOMString AddlData = '');
```

Description

Call CreateNode to create a new generic XML (see page 7392) node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7383) or AttributeNodes list of any node in the document (including this document's ChildNodes (see page 7383) property). It can also be assigned to the DocumentElement (see page 7383) property to replace the entire data hierarchy of this document.

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

CreateNode returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7355) method.

See Also

AddChild (see page 7672)

ChildNodes (see page 7383)

AttributeNodes (see page 7438)

DocumentElement (see page 7383)

AddChild (see page 7355)

CreateElement (see page 7357)

CreateDOMNode (see page 7488)

79.1.1.4.11 TXMLDocument.DefineProperties Method

Designates methods for storing an object's unpublished data on a stream such as a form file.

Pascal

```
procedure DefineProperties(Filer: TFiler); override;
```

C++

```
virtual __fastcall DefineProperties(TFiler Filer);
```

Description

TComponent overrides the DefineProperties method defined in TPersistent to define "fake" Top and Left properties. These are defined so that components that are not controls can be manipulated at design-time. However, the Top and Left properties are hidden, that is, they are not published, because only controls appear at run-time.

DefineProperties is virtual; descendant classes can override it. When overriding DefineProperties, be aware that the Ancestor property of Filer might be set, and that this property can determine whether or not it is appropriate to write properties.

DefineProperties is called automatically as part of the component streaming system; do not call it directly.

See Also

Ancestor

DefineBinaryProperty

DefineProperties

DefineProperty

79.1.1.4.12 TXMLDocument.DoAfterClose Method

Pascal

```
procedure DoAfterClose; dynamic;
```

C++

```
__fastcall DoAfterClose();
```

Description

This is DoAfterClose, a member of class TXMLDocument.

79.1.1.4.13 TXMLDocument.DoAfterOpen Method

Pascal

```
procedure DoAfterOpen; dynamic;
```

C++

```
__fastcall DoAfterOpen();
```

Description

This is DoAfterOpen, a member of class TXMLDocument.

79.1.1.4.14 TXMLDocument.DoBeforeClose Method

Pascal

```
procedure DoBeforeClose; dynamic;
```

C++

```
__fastcall DoBeforeClose();
```

Description

This is DoBeforeClose, a member of class TXMLDocument.

79.1.1.4.15 TXMLDocument.DoBeforeOpen Method

Pascal

```
procedure DoBeforeOpen; dynamic;
```

C++

```
__fastcall DoBeforeOpen();
```

Description

This is DoBeforeOpen, a member of class TXMLDocument.

79.1.1.4.16 TXMLDocument.DoNodeChange Method

Pascal

```
procedure DoNodeChange(const Node: IXMLNode; ChangeType: TNodeChange; BeforeOperation:
Boolean);
```

C++

```
__fastcall DoNodeChange(const IXMLNode Node, TNodeChange ChangeType, Boolean BeforeOperation);
```

Description

This is DoNodeChange, a member of class TXMLDocument.

79.1.1.4.17 TXMLDocument.GeneratePrefix Method

Generates a unique namespace prefix that can be used for a node in the document.

Pascal

```
function GeneratePrefix(const Node: IXMLNode): DOMString;
```

C++

```
__fastcall DOMString GeneratePrefix(const IXMLNode Node);
```

Description

Call GeneratePrefix to generate a new namespace prefix that is not already used in the document to represent a namespace URI. GeneratePrefix uses the NSPrefixBase (see page 7389) property to generate a unique string to act as a namespace prefix. This unique string consists of NSPrefixBase (see page 7389) followed by a number that ensures uniqueness. Thus, for example, if NSPrefixBase (see page 7389) is 'NS', automatically-generated namespace prefixes have the form 'NS1', 'NS2', and so on.

Node (see page 7388) is the interface for the node which will use the newly-generated namespace prefix.

The generated namespace prefixes are only unique within a single document. If two document components have the same value for NSPrefixBase (see page 7389), they will generate the same namespace prefixes.

See Also

NSPrefixBase (see page 7389)

79.1.1.4.18 TXMLDocument.GetActive Method

Returns the value of the Active property.

Pascal

```
function GetActive: Boolean; virtual;
```

C++

```
virtual __fastcall Boolean GetActive();
```

Description

GetActive is the read implementation of the Active property.

See Also

Active (see page 7661)

79.1.1.4.19 TXMLDocument.GetAsyncLoadState Method

Returns the value of the AsyncLoadState property.

Pascal

```
function GetAsyncLoadState: Integer;
```

C++

```
__fastcall int GetAsyncLoadState();
```

Description

GetAsyncLoadState is the read implementation of the AsyncLoadState property.

See Also

AsyncLoadState (see page 7662)

79.1.1.4.20 TXMLDocument.GetChildNodeClass Method

Pascal

```
function GetChildNodeClass(const Node: IDOMNode): TXMLNodeClass; virtual;
```

C++

```
virtual __fastcall TXMLNodeClass GetChildNodeClass(const IDOMNode Node);
```

Description

This is GetChildNodeClass, a member of class TXMLDocument.

79.1.1.4.21 TXMLDocument.GetChildNodes Method

Returns the value of the ChildNodes property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes property.

See Also

ChildNodes (see page 7663)

79.1.1.4.22 TXMLDocument.GetDocBinding Method

Replaces the document element with an instance of a specified implementation class.

Pascal

```
function GetDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString
= ''): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call GetDocBinding to specify the implementation class and tag name for the document element. After replacing the specified node with an instance of the specified class, GetDocBinding returns the interface of the document element.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7383)

RegisterDocBinding (see page 7372)

ChildNodes (see page 7383)

79.1.1.4.23 TXMLDocument.GetDocumentElement Method

Returns the value of the DocumentElement property.

Pascal

```
function GetDocumentElement: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentElement();
```

Description

GetDocumentElement is the read implementation of the DocumentElement property.

See Also

DocumentElement (see page 7663)

79.1.1.4.24 TXMLDocument.GetDocumentNode Method

Returns the value of the Node property.

Pascal

```
function GetDocumentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentNode();
```

Description

GetDocumentNode is the read implementation of the Node property.

See Also

Node (see page 7665)

79.1.1.4.25 TXMLDocument.GetDocumentObject Method

Pascal

```
function GetDocumentObject: TXMLDocument;
```

C++

```
__fastcall TXMLDocument GetDocumentObject();
```

Description

This is GetDocumentObject, a member of class TXMLDocument.

79.1.1.4.26 TXMLDocument.GetDOMDocument Method

Returns the value of the DOMDocument property.

Pascal

```
function GetDOMDocument: IDOMDocument;
```

C++

```
__fastcall IDOMDocument GetDOMDocument();
```

Description

GetDOMDocument is the read implementation of the DOMDocument property.

See Also

DOMDocument (see page 7663)

79.1.1.4.27 TXMLDocument.GetDOMPersist Method

Pascal

```
function GetDOMPersist: IDOMPersist;
```

C++

```
__fastcall IDOMPersist GetDOMPersist();
```

Description

This is GetDOMPersist, a member of class TXMLDocument.

79.1.1.4.28 TXMLDocument.GetEncoding Method

Returns the value of the Encoding property.

Pascal

```
function GetEncoding: DOMString;
```

C++

```
__fastcall DOMString GetEncoding();
```

Description

GetEncoding is the read implementation of the Encoding property.

See Also

Encoding (see page 7664)

79.1.1.4.29 TXMLDocument.GetFileName Method

Returns the value of the FileName property.

Pascal

```
function GetFileName: DOMString;
```

C++

```
__fastcall DOMString GetFileName();
```

Description

GetFileName is the read implementation of the FileName property.

See Also

FileName (see page 7664)

79.1.1.4.30 TXMLDocument.GetModified Method

Returns the value of the Modified property.

Pascal

```
function GetModified: Boolean;
```

C++

```
__fastcall Boolean GetModified();
```

Description

GetModified is the read implementation of the Modified property.

See Also

Modified (see page 7665)

79.1.1.4.31 TXMLDocument.GetNodeIndentStr Method

Returns the value of the NodeIndentStr property.

Pascal

```
function GetNodeIndentStr: DOMString;
```

C++

```
__fastcall DOMString GetNodeIndentStr();
```

Description

GetNodeIndentStr is the read implementation of the NodeIndentStr property.

See Also

NodeIndentStr (see page 7666)

79.1.1.4.32 TXMLDocument.GetOptions Method

Returns the value of the Options property.

Pascal

```
function GetOptions: TXMLDocOptions;
```

C++

```
__fastcall TXMLDocOptions GetOptions();
```

Description

GetOptions is the read implementation of the Options property.

See Also

Options (see page 7666)

79.1.1.4.33 TXMLDocument.GetParseOptions Method

Returns the value of the ParseOptions property.

Pascal

```
function GetParseOptions: TParseOptions;
```

C++

```
__fastcall TParseOptions GetParseOptions();
```

Description

GetParseOptions is the read implementation of the ParseOptions property.

See Also

ParseOptions (see page 7667)

79.1.1.4.34 TXMLDocument.GetPrologNode Method

Pascal

```
function GetPrologNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetPrologNode();
```

Description

This is GetPrologNode, a member of class TXMLDocument.

79.1.1.4.35 TXMLDocument.GetPrologValue Method

Pascal

```
function GetPrologValue(PrologItem: TXMLPrologItem; const Default: DOMString = ''): DOMString;
```

C++

```
__fastcall DOMString GetPrologValue(TXMLPrologItem PrologItem, const DOMString Default = '');
```

Description

This is GetPrologValue, a member of class TXMLDocument.

79.1.1.4.36 TXMLDocument.GetSchemaRef Method

Returns the value of the SchemaRef property.

Pascal

```
function GetSchemaRef: DOMString;
```

C++

```
__fastcall DOMString GetSchemaRef();
```

Description

GetSchemaRef is the read implementation of the SchemaRef property.

See Also

SchemaRef (see page 7667)

79.1.1.4.37 TXMLDocument.GetStandAlone Method

Returns the value of the StandAlone property.

Pascal

```
function GetStandAlone: DOMString;
```

C++

```
__fastcall DOMString GetStandAlone();
```

Description

GetStandAlone is the read implementation of the StandAlone property.

See Also

StandAlone (see page 7668)

79.1.1.4.38 TXMLDocument.GetVersion Method

Returns the value of the Version property.

Pascal

```
function GetVersion: DOMString;
```

C++

```
__fastcall DOMString GetVersion();
```

Description

GetVersion is the read implementation of the Version property.

See Also

Version (see page 7668)

79.1.1.4.39 TXMLDocument.GetXML Method

Returns the value of the XML property.

Pascal

```
function GetXML: TStrings;
```

C++

```
__fastcall TStrings GetXML();
```

Description

GetXML is the read implementation of the XML property.

See Also

XML (see page 7668)

79.1.1.4.40 TXMLDocument.InternalSetPrologValue Method

Pascal

```
function InternalSetPrologValue(const PrologNode: IXMLNode; const Value: Variant; PrologItem:
TXMLPrologItem): string;
```

C++

```
__fastcall UnicodeString InternalSetPrologValue(const IXMLNode PrologNode, const Variant
Value, TXMLPrologItem PrologItem);
```

Description

This is InternalSetPrologValue, a member of class TXMLDocument.

79.1.1.4.41 TXMLDocument.IsEmptyDoc Method

Indicates whether the TXMLDocument (see page 7347) instance represents an empty document.

Pascal

```
function IsEmptyDoc: Boolean;
```

C++

```
__fastcall Boolean IsEmptyDoc();
```

Description

Call IsEmptyDoc to determine whether the TXMLDocument (see page 7347) instance represents an empty document. IsEmptyDoc returns true if the DOMDocument (see page 7384) property is not set (for example, if there is no DOM implementation) or if this object represents a document with no child nodes.

See Also

XML (see page 7392)

ChildNodes (see page 7383)

DOMDocument (see page 7384)

79.1.1.4.42 TXMLDocument.LoadData Method

Pascal

```
procedure LoadData; dynamic;
```

C++

```
__fastcall LoadData();
```

Description

This is LoadData, a member of class TXMLDocument.

79.1.1.4.43 TXMLDocument.Loaded Method

Initializes the component after the form file has been read into memory.

Pascal

```
procedure Loaded; override;
```

C++

```
virtual __fastcall Loaded();
```

Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

Note: All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

Warning: Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

See Also

ComponentState

Create

TReader

79.1.1.4.44 TXMLDocument.LoadFromFile Method

Loads an XML (see page 7392) document and activates it.

Pascal

```
procedure LoadFromFile(const AFileName: DOMString = '');
```

C++

```
__fastcall LoadFromFile(const DOMString AFileName = '');
```

Description

Call LoadFromFile to load the XML (see page 7392) document specified by AFileName and set the Active (see page 7381) property to true so that you can examine or modify the document.

AFileName is the name of the XML (see page 7392) document to load from disk. If AFileName is an empty string, TXMLDocument (see page 7347) uses the value of the FileName (see page 7387) property. If AFileName is not an empty string, TXMLDocument (see page 7347) changes the FileName (see page 7387) property to AFileName.

Once you have loaded an XML (see page 7392) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 7373) method.

See Also

SaveToFile (see page 7373)

FileName (see page 7387)

Active (see page 7381)

Refresh (see page 7371)

LoadFromXML (see page 7370)

LoadFromStream (see page 7369)

79.1.1.4.45 TXMLDocument.LoadFromStream Method

Loads an XML (see page 7392) document from a stream and activates it.

Pascal

```
procedure LoadFromStream(const Stream: TStream; EncodingType: TXMLEncodingType = xetUnknown);
```

C++

```
__fastcall LoadFromStream(const TStream Stream, TXMLEncodingType EncodingType = xetUnknown);
```

Description

Call LoadFromStream to load the XML (see page 7392) document from a stream.

Stream is a stream object that can be used to read the string of XML (see page 7392) that makes up the document.

EncodingType indicates the character set that is used for Stream. If EncodingType is xetUnknown (the default), LoadFromStream attempts to infer the encoding type from the XML (see page 7392) encoding declaration at the beginning of the document.

After loading the document from Stream, LoadFromStream sets the Active (see page 7381) property to true.

See Also

SaveToStream (see page 7373)

LoadFromFile (see page 7368)

Active (see page 7381)

LoadFromXML (see page 7370)

79.1.1.4.46 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	TXMLDocument.LoadFromXML (AnsiString) (see page 7370)	Loads a string representation of an XML (see page 7392) document and activates it.
	TXMLDocument.LoadFromXML (DOMString) (see page 7370)	Loads a string representation of an XML (see page 7392) document and activates it.

79.1.1.4.46.1 TXMLDocument.LoadFromXML Method (AnsiString)

Loads a string representation of an XML (see page 7392) document and activates it.

Pascal

```
procedure LoadFromXML(const XML: AnsiString); overload;
procedure LoadFromXML(const XML: DOMString); overload;
```

C++

```
__fastcall LoadFromXML(const AnsiString XML);
__fastcall LoadFromXML(const DOMString XML);
```

Description

Call LoadFromXML to assign a string as the value of the XML (see page 7392) document. Unlike the XML (see page 7392) property, which lets you assign XML (see page 7392) on a line-by-line basis, LoadFromXML treats the text of the XML (see page 7392) document as a whole.

The XML (see page 7392) parameter is a string containing the text of an XML (see page 7392) document. It should represent the XML (see page 7392) text encoded using UTF-8, Unicode, or UTF-16.

After assigning the XML (see page 7392) property as the contents of the document, LoadFromXML sets the Active (see page 7381) property to true.

See Also

SaveToXML (see page 7374)

LoadFromFile (see page 7368)

Active (see page 7381)

LoadFromStream (see page 7369)

XML (see page 7392)

79.1.1.4.47 TXMLDocument.NewInstance Method

Allocates memory for an instance of an object type and returns a pointer to that new instance.

Pascal

```
class function NewInstance: TObject; override;
```

C++

```
virtual __fastcall TObject * NewInstance();
```

Description

All constructors call System::TObject::NewInstance automatically. System::TObject::NewInstance calls System::TObject::InstanceSize to determine how much memory to allocate from the heap to contain a particular instance. Do not call System::TObject::NewInstance directly.

Override System::TObject::NewInstance only for special memory allocation requirements. For example, when allocating a large number of identical objects that all need to be in memory at the same time, you could allocate a single block of memory for the entire group, then override System::TObject::NewInstance to use part of that larger block for each instance.

If you override System::TObject::NewInstance to allocate memory, you may need to override System::TObject::FreeInstance to deallocate the memory.

Note: By default System::TObject::NewInstance calls System::TObject::InitInstance.

See Also

System::TObject::FreeInstance (see page 3374)

System::TObject::InstanceSize (see page 3377)

System::TObject::InitInstance (see page 3377)

79.1.1.4.48 TXMLDocument.ReadDOMVendor Method

Pascal

```
procedure ReadDOMVendor(Reader: TReader);
```

C++

```
__fastcall ReadDOMVendor(TReader Reader);
```

Description

This is ReadDOMVendor, a member of class TXMLDocument.

79.1.1.4.49 TXMLDocument.Refresh Method

Updates the parsed XML (see page 7392) document to reflect external changes.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the XML (see page 7392) document specified by FileName (see page 7387) has changed due to an external cause. Refresh updates the parsed version of the XML (see page 7392) document (which can be traversed starting with DocumentElement (see page 7383)) to reflect the current contents of the file specified by FileName (see page 7387).

The Active (see page 7381) property must be true when you call Refresh. If the XML (see page 7392) document is not active when you call Refresh, TXMLDocument (see page 7347) raises an EXMLDocError (see page 7639) exception.

Note: Do not call Refresh after making changes directly to the XML (see page 7392) property, because assigning a value to XML (see page 7392) automatically closes the XML (see page 7392) document. To refresh the XML (see page 7392) document after assigning a new value to XML (see page 7392), set the Active (see page 7381) property to true.

See Also

FileName (see page 7387)

Active (see page 7381)

DOMDocument (see page 7384)

DocumentElement (see page 7383)

Resync (see page 7372)

79.1.1.4.50 TXMLDocument.RegisterDocBinding Method

Registers an implementation class for child nodes with a specified tag name.

Pascal

```
procedure RegisterDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI:
DOMString = '');
```

C++

```
__fastcall RegisterDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call RegisterDocBinding to specify the implementation class for one of the child nodes of the document node. RegisterDocBinding replaces the specified node with an instance of the specified class.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7383)

GetDocBinding (see page 7362)

ChildNodes (see page 7383)

79.1.1.4.51 TXMLDocument.ReleaseDoc Method

Pascal

```
procedure ReleaseDoc(const CheckSave: Boolean = True); dynamic;
```

C++

```
__fastcall ReleaseDoc(const Boolean CheckSave = True);
```

Description

This is ReleaseDoc, a member of class TXMLDocument.

79.1.1.4.52 TXMLDocument.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the document to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7383)

Refresh (see page 7371)

79.1.1.4.53 TXMLDocument.SaveToFile Method

Saves the XML (see page 7392) document to disk.

Pascal

```
procedure SaveToFile(const AFileName: DOMString = ''); dynamic;
```

C++

```
__fastcall SaveToFile(const DOMString AFileName = '');
```

Description

Call SaveToFile to save any modifications you have made to the parsed XML (see page 7392) document.

AFileName is the name of the file to save. If AFileName is an empty string, TXMLDocument (see page 7347) uses the value of the FileName (see page 7387) property.

See Also

LoadFromFile (see page 7368)

FileName (see page 7387)

SaveToXML (see page 7374)

SaveToStream (see page 7373)

Using Code That the XML Data Binding Wizard Generates

79.1.1.4.54 TXMLDocument.SaveToStream Method

Saves the XML (see page 7392) document to a stream.

Pascal

```
procedure SaveToStream(const Stream: TStream);
```

C++

```
__fastcall SaveToStream(const TStream Stream);
```

Description

Call SaveToStream to save the contents of the XML (see page 7392) document to the stream specified by Stream. Regardless of the encoding system of the original XML (see page 7392) document, SaveToStream always saves the stream in UTF-16.

See Also

LoadFromStream (see page 7369)

SaveToXML (see page 7374)

SaveToFile (see page 7373)

79.1.1.4.55 TXMLDocument.SaveToUTF8String Method

Pascal

```
procedure SaveToUTF8String(var XML: UTF8String);
```

C++

```
__fastcall SaveToUTF8String(UTF8String XML);
```

Description

This is SaveToUTF8String, a member of class TXMLDocument.

79.1.1.4.56 SaveToXML Method

This is the overview for the SaveToXML method overload.

Overload List

	Name	Description
	TXMLDocument.SaveToXML (DOMString) (see page 7374)	Saves the XML (see page 7392) document to a string-type variable.
	TXMLDocument.SaveToXML (UTF8String) (see page 7374)	Saves the XML (see page 7392) document to a string-type variable.
	TXMLDocument.SaveToXML (WideString) (see page 7374)	Saves the XML (see page 7392) document to a string-type variable.

79.1.1.4.56.1 TXMLDocument.SaveToXML Method (DOMString)

Saves the XML (see page 7392) document to a string-type variable.

Pascal

```
procedure SaveToXML(var XML: DOMString); overload;
procedure SaveToXML(var XML: WideString); overload;
procedure SaveToXML(var XML: UTF8String); overload;
```

C++

```
__fastcall SaveToXML(DOMString XML);
__fastcall SaveToXML(BSTR XML);
__fastcall SaveToXML(UTF8String XML);
```

Description

Call SaveToXML to save the contents of the XML (see page 7392) document to the string-type variable specified by XML (see page 7392). SaveToXML writes the contents of XML (see page 7392) document using UTF-8 or UTF-16 as an encoding system, depending on the type of the XML (see page 7392) parameter.

Unlike the XML (see page 7392) property, which lets you write individual lines from the XML (see page 7392) document, SaveToXML writes the entire text of the XML (see page 7392) document.

See Also

LoadFromXML (see page 7370)

SaveToStream (see page 7373)

SaveToFile (see page 7373)

XML (see page 7392)

79.1.1.4.57 TXMLDocument.SaveToXMLStrings Method

Saves the XML (see page 7392) document to an internal string list.

Pascal

```
procedure SaveToXMLStrings;
```

C++

```
__fastcall SaveToXMLStrings();
```

Description

The SaveToXMLStrings procedure saves the content of the XML (see page 7392) document to a TStringList. SaveToXMLStrings writes the content of the XML (see page 7392) document in Unicode.

See Also

SaveToXML (see page 7374)

XML (see page 7392)

79.1.1.4.58 TXMLDocument.SetActive Method

Sets the value of the Active property.

Pascal

```
procedure SetActive(const Value: Boolean); virtual;
```

C++

```
virtual __fastcall SetActive(const Boolean Value);
```

Description

SetActive is the write implementation of the Active property.

Value is the new value to set.

See Also

Active (see page 7661)

79.1.1.4.59 TXMLDocument.SetDocumentElement Method

Sets the value of the DocumentElement property.

Pascal

```
procedure SetDocumentElement(const Value: IXMLNode);
```

C++

```
__fastcall SetDocumentElement(const IXMLNode Value);
```

Description

SetDocumentElement is the write implementation of the DocumentElement property.

Value is the interface for the node tree that replaces the current document element.

See Also

DocumentElement (see page 7663)

79.1.1.4.60 TXMLDocument.SetDOMDocument Method

Sets the value of the DOMDocument property.

Pascal

```
procedure SetDOMDocument(const Value: IDOMDocument);
```

C++

```
__fastcall SetDOMDocument(const IDOMDocument Value);
```

Description

SetDOMDocument is the write implementation of the DOMDocument property.

Value is the DOM interface for the document to assign.

See Also

DOMDocument (see page 7663)

79.1.1.4.61 TXMLDocument.SetEncoding Method

Sets the value of the Encoding property.

Pascal

```
procedure SetEncoding(const Value: DOMString);
```

C++

```
__fastcall SetEncoding(const DOMString Value);
```

Description

SetEncoding is the write implementation of the Encoding property.

Value is the value of the encoding attribute of the document type node.

See Also

Encoding (see page 7664)

79.1.1.4.62 TXMLDocument.SetFileName Method

Sets the value of the FileName property.

Pascal

```
procedure SetFileName(const Value: DOMString);
```

C++

```
__fastcall SetFileName(const DOMString Value);
```

Description

SetFileName is the write implementation of the FileName property.

Value is the name of an XML document or, if the vendor supports it, an URL for the XML document.

See Also

FileName (see page 7664)

79.1.1.4.63 TXMLDocument.SetModified Method

Pascal

```
procedure SetModified(const Value: Boolean);
```

C++

```
__fastcall SetModified(const Boolean Value);
```

Description

This is SetModified, a member of class TXMLDocument.

79.1.1.4.64 TXMLDocument.SetNodeIndentStr Method

Sets the value of the NodeIndentStr property.

Pascal

```
procedure SetNodeIndentStr(const Value: DOMString);
```

C++

```
__fastcall SetNodeIndentStr(const DOMString Value);
```

Description

SetNodeIndentStr is the write implementation of the NodeIndentStr property.

Value is the string that is inserted before nested nodes to indicate a level of nesting.

See Also

NodeIndentStr (see page 7666)

79.1.1.4.65 TXMLDocument.SetOnAsyncLoad Method

Supplies an event handler that responds when the AsyncLoadState property changes.

Pascal

```
procedure SetOnAsyncLoad(const Value: TAsyncEventHandler);
```

C++

```
__fastcall SetOnAsyncLoad(const TAsyncEventHandler Value);
```

Description

Use SetOnAsyncLoad to supply a callback that responds when the asynchronous parsing of an XML document progresses to a certain point.

Value is a method that is called when the AsyncLoadState property changes to reflect a new stage in the parsing process.

See Also

AsyncLoadState (see page 7662)

79.1.1.4.66 TXMLDocument.SetOptions Method

Sets the value of the Options property.

Pascal

```
procedure SetOptions(const Value: TXMLDocOptions);
```

C++

```
__fastcall SetOptions(const TXMLDocOptions Value);
```

Description

GetOptions is the write implementation of the Options property.

Value is the set of options to assign.

See Also

Options (see page 7666)

79.1.1.4.67 TXMLDocument.SetParseOptions Method

Sets the value of the ParseOptions property.

Pascal

```
procedure SetParseOptions(const Value: TParseOptions);
```

C++

```
__fastcall SetParseOptions(const TParseOptions Value);
```

Description

GetParseOptions is the write implementation of the ParseOptions property.

Value is the set of parser options to assign.

See Also

ParseOptions (see page 7667)

79.1.1.4.68 TXMLDocument.SetPrologValue Method

Pascal

```
procedure SetPrologValue(const Value: Variant; PrologItem: TXMLPrologItem);
```

C++

```
__fastcall SetPrologValue(const Variant Value, TXMLPrologItem PrologItem);
```

Description

This is SetPrologValue, a member of class TXMLDocument.

79.1.1.4.69 TXMLDocument.SetStandAlone Method

Sets the value of the StandAlone property.

Pascal

```
procedure SetStandAlone(const Value: DOMString);
```

C++

```
__fastcall SetStandAlone(const DOMString Value);
```

Description

SetStandAlone is the write implementation of the StandAlone property.

Value is the value of the stand-alone attribute of the document type node.

See Also

StandAlone (see page 7668)

79.1.1.4.70 TXMLDocument.SetVersion Method

Sets the value of the Version property.

Pascal

```
procedure SetVersion(const Value: DOMString);
```

C++

```
__fastcall SetVersion(const DOMString Value);
```

Description

SetVersion is the write implementation of the Version property.

Value is the value of the version attribute of the document type node.

See Also

Version (see page 7668)

79.1.1.4.71 TXMLDocument.SetXML Method

Sets the value of the XML property.

Pascal

```
procedure SetXML(const Value: TStrings);
```

C++

```
__fastcall SetXML(const TStrings Value);
```

Description

SetXML is the write implementation of the XML property.

Value contains the raw (unparsed) XML to assign.

See Also

XML (see page 7668)

79.1.1.4.72 TXMLDocument.SetXMLStrings Method

Pascal

```
procedure SetXMLStrings(const Value: string);
```

C++

```
__fastcall SetXMLStrings(const UnicodeString Value);
```

Description

This is SetXMLStrings, a member of class TXMLDocument.

79.1.1.4.73 TXMLDocument.WriteDOMVendor Method

Pascal

```
procedure WriteDOMVendor(Writer: TWriter);
```

C++

```
__fastcall WriteDOMVendor(TWriter Writer);
```

Description

This is WriteDOMVendor, a member of class TXMLDocument.

79.1.1.4.74 TXMLDocument.XMLStringsChanging Method

Pascal

```
procedure XMLStringsChanging(Sender: TObject);
```

C++

```
__fastcall XMLStringsChanging(TObject * Sender);
```

Description

This is XMLStringsChanging, a member of class TXMLDocument.

79.1.1.5 TXMLDocument Properties

The properties of the TXMLDocument class are listed here.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7383)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7383)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7385)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7386)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7391)	This is PrefixID, a member of class TXMLDocument.

Public Properties

	Name	Description
R	AsyncLoadState (see page 7382)	Indicates the state of the DOM parser when it parses the XML (see page 7392) document asynchronously.
R	ChildNodes (see page 7383)	Lists all of the child nodes of the document.
	DocumentElement (see page 7383)	Provides access to the root node of the XML (see page 7392) document.
	DOMDocument (see page 7384)	Provides access to the DOM interface for the parsed XML (see page 7392) document.
	DOMImplementation (see page 7385)	Provides access to the DOM interface for document-independent functions.
	Encoding (see page 7386)	Indicates the character set that the XML (see page 7392) document uses to encode characters.
R	Modified (see page 7387)	Indicates whether the XML (see page 7392) document has been modified.

R	Node (see page 7388)	Provides access to the document node for the XML (see page 7392) document.
	NSPrefixBase (see page 7389)	Specifies the base string for automatically-generated namespace prefixes.
R	SchemaRef (see page 7391)	Returns the name of the schema associated with this XML (see page 7392) document.
	StandAlone (see page 7391)	Indicates whether the XML (see page 7392) document includes any external markup declarations.
	Version (see page 7391)	Indicates the XML (see page 7392) version that the document uses.

Published Properties

	Name	Description
	Active (see page 7381)	Specifies whether the XML (see page 7392) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7386)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7392) document.
	FileName (see page 7387)	Specifies the file name for the XML (see page 7392) document associated with this TXMLDocument (see page 7347) object.
	NodeIndentStr (see page 7388)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7392) text.
	Options (see page 7389)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7390)	Specifies implementation-specific options that are followed when parsing the XML (see page 7392) document.
	XML (see page 7392)	Specifies the content of the XML document associated with this TXMLDocument (see page 7347) object.

79.1.1.5.1 TXMLDocument.Active Property

Specifies whether the XML (see page 7392) document has been parsed so that the DOM interfaces are available to examine or change its contents.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before you can use TXMLDocument (see page 7347) to examine or change the contents of an XML (see page 7392) document, it must be active. Setting Active to true causes TXMLDocument (see page 7347) to do the following:

1.Generate a BeforeOpen (see page 7395) event.

2.If necessary, load the DOM parser and assign the DOMImplementation (see page 7385) property.

3.Parse the XML (see page 7392) document specified by FileName (see page 7387) or, if FileName (see page 7387) is an empty string, by the XML (see page 7392) property using the options specified by ParseOptions (see page 7390). If both FileName (see page 7387) and XML (see page 7392) are empty strings, TXMLDocument (see page 7347) creates an empty document. The interface for the parsed document is assigned to the DOMDocument (see page 7384) property.

4. Generate an AfterOpen (see page 7394) event.

Note: Once the DOMDocument (see page 7384) property is assigned, you can also use the DocumentElement (see page 7383) property.

Setting Active to false causes TXMLDocument (see page 7347) to do the following:

1.Generate a BeforeClose (see page 7394) event.

2.Release (see page 3641) all of the DOM interfaces, such as DOMDocument (see page 7384) and DocumentElement (

see page 7383), that represent the parsed XML (see page 7392) document.

3. Generate an AfterClose (see page 7393) event.

See Also

DOMDocument (see page 7384)

DocumentElement (see page 7383)

FileName (see page 7387)

XML (see page 7392)

ParseOptions (see page 7390)

BeforeOpen (see page 7395)

AfterOpen (see page 7394)

DOMImplementation (see page 7385)

BeforeClose (see page 7394)

AfterClose (see page 7393)

Using TXMLDocument

79.1.1.5.2 TXMLDocument.AsyncLoadState Property

Indicates the state of the DOM parser when it parses the XML (see page 7392) document asynchronously.

Pascal

```
property AsyncLoadState: Integer;
```

C++

```
__property int AsyncLoadState;
```

Description

Use AsyncLoadState to determine that status of the DOM parser when it is parsing an XML (see page 7392) document asynchronously. AsyncLoadState is only meaningful when the value of ParseOptions (see page 7390) is poAsyncLoad.

The value of AsyncLoadState is vendor-specific. That is, each DOM parser that loads and parses an XML (see page 7392) document asynchronously supplies its own flags to indicate state information.

When the DOM vendor is MSXML, AsyncLoadState can have any of the following values:

Value	Meaning
0	Uninitialized: The AsyncLoadState is not assigned. (Usually indicates that the parser has not tried to parse a document or it is not parsing the document asynchronously.)
1	Loading: The parser is currently parsing the structure of the XML (see page 7392) document.
2	Loaded (see page 7368): The parser has finished reading and parsing the structure of the XML (see page 7392) document, but the corresponding DOM interfaces are not yet available.
3	Interactive: The parser has parsed the structure of the XML (see page 7392) document, the corresponding DOM interfaces are available, but the parser has not finished reading all the node values.
4	Complete: The parser has fully read and parsed the XML (see page 7392) document.

See Also

OnAsyncLoad (see page 7395)

ParseOptions (see page 7390)

79.1.1.5.3 TXMLDocument.ChildNodes Property

Lists all of the child nodes of the document.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access all of the nodes that are children of the document object. In addition to DocumentElement (see page 7383), which is the root of the data hierarchy, ChildNodes can include nodes for comments, processing instructions, and so on.

Note: ChildNodes is not available unless the XML (see page 7392) document is Active (see page 7381).

See Also

DocumentElement (see page 7383)

Active (see page 7381)

79.1.1.5.4 TXMLDocument.DocBindingInfo Property

Pascal

```
property DocBindingInfo: TNodeClassArray;
```

C++

```
__property TNodeClassArray DocBindingInfo;
```

Description

This is DocBindingInfo, a member of class TXMLDocument.

79.1.1.5.5 TXMLDocument.DocSource Property

Pascal

```
property DocSource: TXMLDocumentSource;
```

C++

```
__property TXMLDocumentSource DocSource;
```

Description

This is DocSource, a member of class TXMLDocument.

79.1.1.5.6 TXMLDocument.DocumentElement Property

Provides access to the root node of the XML (see page 7392) document.

Pascal

```
property DocumentElement: IXMLNode;
```

C++

```
__property IXMLNode DocumentElement;
```

Description

Use DocumentElement to access the root node of the XML (see page 7392) document. Because XML (see page 7392) documents are hierarchical in structure, DocumentElement is the entry point to all data in the XML (see page 7392) document. Setting DocumentElement replaces the entire data hierarchy with a new hierarchy of nodes.

Note: For some DOM implementations, if you set DocumentElement, all of the new nodes must already be associated with this document. That is, the new value should be a node that came from the existing node hierarchy or that was created using the CreateNode (see page 7357) method.

The IXMLNode (see page 7669) interface includes properties and methods for examining and changing the text and attributes of a node, for adding and deleting child nodes, and for traversing the hierarchy of nodes rooted at that node.

Note: DocumentElement is not available unless the XML (see page 7392) document is Active (see page 7381).

See Also

DOMDocument (see page 7384)

Active (see page 7381)

CreateNode (see page 7357)

GetDocBinding (see page 7362)

Using TXMLDocument

Working with XML Nodes

79.1.1.5.7 TXMLDocument.DOMDocument Property

Provides access to the DOM interface for the parsed XML (see page 7392) document.

Pascal

```
property DOMDocument: IDOMDocument;
```

C++

```
__property IDOMDocument DOMDocument;
```

Description

Use DOMDocument to directly access the IDOMDocument (see page 7530) interface for the parsed XML (see page 7392) document. This is the interface supported by the DOM parser for the document that is represented in raw (unparsed) form by the FileName (see page 7387) or XML (see page 7392) property.

DOMDocument provides access to the IDOMImplementation (see page 7552) and IDOMElement (see page 7542) interfaces that are surfaced as the DOMImplementation (see page 7385) and DocumentElement (see page 7383) properties. It also provides access to an IDOMDocumentType (see page 7538) interface that describes the structure of the XML (see page 7392) document, and supports a number of methods for creating elements within the document.

Most applications do not need to use the DOMDocument property directly. Instead, it supplies the underlying support for parsing and manipulating the nodes accessed through DocumentElement (see page 7383). If you do want to make use of DOMDocument's low-level support, you must be familiar with the W3C Document Object Model (DOM) Level 1 Specification.

Note: DOMDocument is not available unless the XML (see page 7392) document is Active (see page 7381).

See Also

Active (see page 7381)

DocumentElement (see page 7383)

DOMImplementation (see page 7385)

XML (see page 7392)

FileName (see page 7387)

79.1.1.5.8 TXMLDocument.DOMImplementation Property

Provides access to the DOM interface for document-independent functions.

Pascal

```
property DOMImplementation: IDOMImplementation;
```

C++

```
__property IDOMImplementation DOMImplementation;
```

Description

Use DOMImplementation to directly access the IDOMImplemenation interface identified by the DOMVendor (see page 7386) property. This is the interface TXMLDocument (see page 7347) uses to generate the IDOMDocument (see page 7530) interface that is the value of DOMDocument (see page 7384).

Typically, the value of DOMImplementation is obtained automatically, based on the value of the DOMVendor (see page 7386) property. You can, however, assign the DOMImplementation property directly to an IDOMImplementation (see page 7552) interface that you have obtained through some other mechanism.

Most applications do not need to use the DOMImplementation property directly. If you do want to make use of DOMImplementation's low-level support, you must be familiar with the W3C Document Object Model (DOM) Level 2 Specification.

You must assign a value to either DOMVendor (see page 7386) or DOMImplementation before you can set the Active (see page 7381) property to true.

See Also

DOMVendor (see page 7386)

DOMDocument (see page 7384)

DOMVendors (see page 7622)

Active (see page 7381)

79.1.1.5.9 TXMLDocument.DOMParseOptions Property

Pascal

```
property DOMParseOptions: IDOMParseOptions;
```

C++

```
__property IDOMParseOptions DOMParseOptions;
```

Description

This is DOMParseOptions, a member of class TXMLDocument.

79.1.1.5.10 TXMLDocument.DOMPersist Property

Pascal

```
property DOMPersist: IDOMPersist;
```

C++

```
__property IDOMPersist DOMPersist;
```

Description

This is DOMPersist, a member of class TXMLDocument.

79.1.1.5.11 TXMLDocument.DOMVendor Property

Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7392) document.

Pascal

```
property DOMVendor: TDOMVendor;
```

C++

```
__property TDOMVendor DOMVendor;
```

Description

DOMVendor is the object that represents a registered DOM implementation. TXMLDocument (see page 7347) uses this object to obtain an IDOMImplementation (see page 7552) interface, which it can use to access all the other DOM interfaces needed for parsing and manipulating the XML (see page 7392) document. Once DOMVendor is assigned, TXMLDocument (see page 7347) obtains the IDOMImplementation (see page 7552) interface (when needed) and assigns it to the DOMImplementation (see page 7385) property.

DOMVendor must be registered before it can be used to obtain an IDOMImplementation (see page 7552) interface. For a list of all the currently registered DOM implementations, use the global DOMVendors (see page 7622) variable. To obtain a specific DOM implementation, given its description, use the global GetDOMVendor (see page 7615) function. Use the RegisterDOMVendor (see page 7617) procedure to register a new DOM implementation.

You must assign a value to either DOMVendor or DOMImplementation (see page 7385) before you can set the Active (see page 7381) property to true.

See Also

DOMImplementation (see page 7385)

DOMVendors (see page 7622)

Active (see page 7381)

RegisterDOMVendor (see page 7617)

GetDOMVendor (see page 7615)

Using TXMLDocument

79.1.1.5.12 TXMLDocument.Encoding Property

Indicates the character set that the XML (see page 7392) document uses to encode characters.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Encoding is the encoding name that appears in the document type declaration of the XML (see page 7392) document. It is a string such as 'UTF-8', 'UTF-16', or 'ISO-10646-UCS-2' that indicates the character set used in the document.

See Also

FileName (see page 7387)

DOMDocument (see page 7384)

XML (see page 7392)

79.1.1.5.13 TXMLDocument.FileName Property

Specifies the file name for the XML (see page 7392) document associated with this TXMLDocument (see page 7347) object.

Pascal

```
property FileName: DOMString;
```

C++

```
__property DOMString FileName;
```

Description

Set FileName to specify the XML (see page 7392) document you want to examine or manipulate using the TXMLDocument (see page 7347) object. Once you have set FileName to the name of an XML (see page 7392) document, you can activate the XML (see page 7392) document and examine or manipulate it using the DocumentElement (see page 7383) property.

When you set the value of FileName, the XML (see page 7392) property automatically changes (at runtime) to represent the contents of that file.

Note: Setting the value of FileName automatically sets the Active (see page 7381) property to false.

After activating the XML (see page 7392) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 7373) method.

Tip: If the underlying DOM implementation supports it, you can use FileName to specify an URL rather than a file on disk.

See Also

XML (see page 7392)

LoadFromFile (see page 7368)

SaveToFile (see page 7373)

DocumentElement (see page 7383)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

79.1.1.5.14 TXMLDocument.Modified Property

Indicates whether the XML (see page 7392) document has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Read (see page 3634) Modified to determine whether the application has made any changes to the nodes in the XML (see page 7392) document. When Modified is true, the value of the XML (see page 7392) property may differ from the contents of the file specified by FileName (see page 7387). You can call the SaveToFile (see page 7373) method to update the XML (see page 7392) document file to reflect the changes.

Note: Modfied changes to false when you call the Refresh (see page 7371) or SaveToFile (see page 7373) method.

See Also

BeforeNodeChange (see page 7394)

AfterNodeChange (see page 7393)

Refresh (see page 7371)

SaveToFile (see page 7373)

XML (see page 7392)

FileName (see page 7387)

79.1.1.5.15 TXMLDocument.Node Property

Provides access to the document node for the XML (see page 7392) document.

Pascal

```
property Node: IXMLNode;
```

C++

```
__property IXMLNode Node;
```

Description

Read (see page 3634) Node to access the document node for the XML (see page 7392) document. The document node represents information about the entire document. It may have several child nodes, which represent comments, processing instructions, and which include the DocumentElement (see page 7383) node that acts as the root of the data hierarchy.

See Also

DocumentElement (see page 7383)

ChildNodes (see page 7383)

79.1.1.5.16 TXMLDocument.NodeIndentStr Property

Indicates the string that is inserted before nested nodes in the formatted XML (see page 7392) text.

Pascal

```
property NodeIndentStr: DOMString;
```

C++

```
__property DOMString NodeIndentStr;
```

Description

Use NodeIndentStr to customize how TXMLDocument (see page 7347) formats nodes that are added. When the Options (see page 7389) property includes doNodeAutoIndent, this string is inserted in the generated XML (see page 7392) before any newly added node, once for each nesting level.

If you do not assign a value to NodeIndentStr, TXMLDocument (see page 7347) inserts two spaces to represent each nesting level.

See Also

Options (see page 7389)

NestingLevel (see page 7428)

FormatXMLData (see page 7490)

XML (see page 7392)

79.1.1.5.17 TXMLDocument.NSPrefixBase Property

Specifies the base string for automatically-generated namespace prefixes.

Pascal

```
property NSPrefixBase: DOMString;
```

C++

```
__property DOMString NSPrefixBase;
```

Description

The AddChild (see page 7355) method for each node allows the caller to specify that the new node should use an automatically-generated namespace prefix. When this option is specified, the node uses NSPrefixBase to generate a unique string to act as a namespace prefix. This unique string consists of NSPrefixBase followed by a number that ensures uniqueness. Thus, for example, if NSPrefixBase is 'NS', automatically-generated namespace prefixes have the form 'NS1', 'NS2', and so on.

The generated namespace prefixes are only unique within a single document. If two document components have the same value for NSPrefixBase, they will generate the same namespace prefixes.

See Also

AddChild (see page 7672)

GeneratePrefix (see page 7360)

79.1.1.5.18 TXMLDocument.Options Property

Influences various aspects of the document's behavior.

Pascal

```
property Options: TXMLDocOptions;
```

C++

```
__property TXMLDocOptions Options;
```

Description

Set Options to configure how you want the document to respond when the application tries to read nodes or attributes that do not exist, when it creates a new namespace when adding a node, when it generates XML (see page 7392) text from the nodes in the document, or whether it saves its value automatically when you close the document. Options can include any of the following

values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 7703) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML (see page 7392) text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.
doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null (see page 5056) Variant (see page 3421) (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild (see page 7355) method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild (see page 7355) creates the new namespace but does not add an attribute to declare it.
doAutoSave	When you close the document (set Active (see page 7381) to false), any changes are automatically saved back to the file specified by FileName (see page 7387) or to the XML (see page 7392) property.

See Also

Nodes (see page 7714)

Attributes (see page 7688)

AddChild (see page 7672)

Active (see page 7381)

FileName (see page 7387)

XML (see page 7392)

79.1.1.5.19 TXMLDocument.ParseOptions Property

Specifies implementation-specific options that are followed when parsing the XML (see page 7392) document.

Pascal

```
property ParseOptions: TParseOptions;
```

C++

```
__property TParseOptions ParseOptions;
```

Description

Set ParseOptions to influence the way the DOM implementation parses the XML (see page 7392) document. Different DOM implementations support different options. If you attempt to use an option that is not supported by your DOM implementation, TXMLDocument (see page 7347) raises an EXMLDocError (see page 7639) exception.

See Also

DOMImplementation (see page 7385)

EXMLDocError (see page 7639)

Using TXMLDocument

79.1.1.5.20 TXMLDocument.PrefixID Property

Pascal

```
property PrefixID: Integer;
```

C++

```
__property int PrefixID;
```

Description

This is PrefixID, a member of class TXMLDocument.

79.1.1.5.21 TXMLDocument.SchemaRef Property

Returns the name of the schema associated with this XML (see page 7392) document.

Pascal

```
property SchemaRef: DOMString;
```

C++

```
__property DOMString SchemaRef;
```

Description

Read (see page 3634) SchemaRef to determine the schema associated with this XML (see page 7392) document. If the document does not have an attribute that identifies an associated schema, the value of SchemaRef is an empty string.

See Also

DOMDocument (see page 7384)

79.1.1.5.22 TXMLDocument.StandAlone Property

Indicates whether the XML (see page 7392) document includes any external markup declarations.

Pascal

```
property StandAlone: DOMString;
```

C++

```
__property DOMString StandAlone;
```

Description

StandAlone is the value of the stand-alone attribute on the document type declaration of the XML (see page 7392) document. StandAlone has a value of 'yes' or 'no'.

When StandAlone is 'yes', the document contains no external markup declarations. It may, however, still contain references to external entities, as long as they are not declarations.

When StandAlone is 'no', the document may or may not contain external markup declarations.

See Also

DOMDocument (see page 7384)

79.1.1.5.23 TXMLDocument.Version Property

Indicates the XML (see page 7392) version that the document uses.

Pascal

```
property Version: DOMString;
```

C++

```
__property DOMString Version;
```

Description

Version indicates the XML (see page 7392) version number that the document uses. An XML (see page 7392) version number is a string of the form 'n.m', where n is the major version number and m is the minor version number.

See Also

XML (see page 7392)

79.1.1.5.24 TXMLDocument.XML Property

Specifies the content of the XML document associated with this TXMLDocument (see page 7347) object.

Pascal

```
property XML: TStrings;
```

C++

```
__property TStrings XML;
```

Description

Read (see page 3634) XML to obtain the contents of the associated XML document. Set XML to specify the XML document you want to examine or manipulate. Once XML is set, you can activate the XML document and examine or manipulate it using the DocumentElement (see page 7383) property.

XML is a TStrings object. This means you can supply a value using the Strings editor at design time, or manipulate the XML line by line at runtime. To read or write the XML in the document as a single string, use the LoadFromXML (see page 7370) or SaveToXML (see page 7374) method.

See Also

FileName (see page 7387)

LoadFromXML (see page 7370)

SaveToXML (see page 7374)

DocumentElement (see page 7383)

Using TXMLDocument

Using Code That the XML Data Binding Wizard Generates

79.1.1.6 TXMLDocument Events

The events of the TXMLDocument class are listed here.

Published Events

	Name	Description
	AfterClose (see page 7393)	Occurs after the XML (see page 7392) document is closed.
	AfterNodeChange (see page 7393)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7394)	Occurs immediately after the XML (see page 7392) document is opened.
	BeforeClose (see page 7394)	Occurs immediately before the XML (see page 7392) document is closed.

	BeforeNodeChange (see page 7394)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7395)	Occurs immediately before the XML (see page 7392) document is opened.
	OnAsyncLoad (see page 7395)	Occurs at various stages during the asynchronous parsing of an XML (see page 7392) document.

79.1.1.6.1 TXMLDocument.AfterClose Event

Occurs after the XML (see page 7392) document is closed.

Pascal

```
property AfterClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterClose;
```

Description

Write (see page 3707) an AfterClose event handler to take specific action after the XML (see page 7392) document interfaces are released. For example, you can use an AfterClose event to update your user interface to indicate that the XML (see page 7392) document is no longer available.

AfterClose is the last event to occur when the Active (see page 7381) property is set to false.

See Also

BeforeClose (see page 7394)

Active (see page 7381)

79.1.1.6.2 TXMLDocument.AfterNodeChange Event

Occurs after a change is made to one of the nodes in the document.

Pascal

```
property AfterNodeChange: TNodeChangeEvent;
```

C++

```
__property TNodeChangeEvent AfterNodeChange;
```

Description

Write (see page 3707) an AfterNodeChange event handler to take specific action after a change is made to a node in the XML (see page 7392) document. When an application makes a change to one of the nodes, the following steps occur:

1. The Modified (see page 7387) property is set to true.
2. A BeforeNodeChange (see page 7394) event occurs.
3. The change is made to the node.
4. An AfterNodeChange event occurs.

See Also

BeforeNodeChange (see page 7394)

Modified (see page 7387)

ChildNodes (see page 7383)

79.1.1.6.3 TXMLDocument.AfterOpen Event

Occurs immediately after the XML (see page 7392) document is opened.

Pascal

```
property AfterOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent AfterOpen;
```

Description

Write (see page 3707) an AfterOpen event handler to take specific action once the XML (see page 7392) document interfaces become available. For example, you can use an AfterOpen event to update your user interface to reflect the structure of the XML (see page 7392) document once it is opened.

AfterOpen is the last event to occur when the Active (see page 7381) property is set to true.

See Also

BeforeOpen (see page 7395)

Active (see page 7381)

79.1.1.6.4 TXMLDocument.BeforeClose Event

Occurs immediately before the XML (see page 7392) document is closed.

Pascal

```
property BeforeClose: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeClose;
```

Description

Write (see page 3707) a BeforeClose event handler to take specific action before the XML (see page 7392) document interfaces are released. For example, you can use an BeforeClose event to cache information about the XML (see page 7392) document before it becomes unavailable.

BeforeClose is the first event to occur when the Active (see page 7381) property is set to false.

See Also

AfterClose (see page 7393)

Active (see page 7381)

79.1.1.6.5 TXMLDocument.BeforeNodeChange Event

Occurs before a change is made to one of the nodes in the document.

Pascal

```
property BeforeNodeChange: TNodeChangeEvent;
```

C++

```
__property TNodeChangeEvent BeforeNodeChange;
```

Description

Write (see page 3707) a BeforeNodeChange event handler to take specific action before a change is made to a node in the XML (see page 7392) document. When an application makes a change to one of the nodes, the following steps occur:

1. The Modified (see page 7387) property is set to true.
2. A BeforeNodeChange event occurs.
3. The change is made to the node.
4. An BeforeNodeChange event occurs.

See Also

AfterNodeChange (see page 7393)

Modified (see page 7387)

ChildNodes (see page 7383)

79.1.1.6.6 TXMLDocument.BeforeOpen Event

Occurs immediately before the XML (see page 7392) document is opened.

Pascal

```
property BeforeOpen: TNotifyEvent;
```

C++

```
__property TNotifyEvent BeforeOpen;
```

Description

Write (see page 3707) a BeforeOpen event handler to take specific action before the XML (see page 7392) document interfaces become available. For example, you can use a BeforeOpen event to provide feedback to the user that the XML (see page 7392) document is being parsed (which can be cleared in an AfterOpen (see page 7394) event handler).

BeforeOpen is the first event to occur when the Active (see page 7381) property is set to true.

See Also

AfterOpen (see page 7394)

Active (see page 7381)

79.1.1.6.7 TXMLDocument.OnAsyncLoad Event

Occurs at various stages during the asynchronous parsing of an XML (see page 7392) document.

Pascal

```
property OnAsyncLoad: TAsyncEventHandler;
```

C++

```
__property TAsyncEventHandler OnAsyncLoad;
```

Description

Write (see page 3707) a OnAsyncLoad event handler to respond appropriately during the asynchronous parsing of an XML (see page 7392) document. For example, OnAsyncLoad can update a progress indicator, or initiate operations that can't be performed until the document is fully parsed. OnAsyncLoad occurs every time the AsyncLoadState (see page 7382) property

changes to reflect a new stage in the parsing process.

See Also

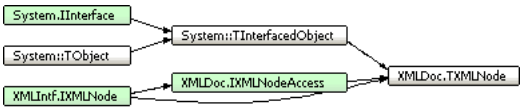
AsyncLoadState (see page 7382)

XML (see page 7392)

79.1.2 TXMLNode Class

TXMLNode represents a node in an XML document.

Class Hierarchy



Pascal

```
TXMLNode = class(TInterfacedObject, IXMLNode, IXMLNodeAccess);
```

C++

```
class TXMLNode : public TInterfacedObject, public IXMLNode, public IXMLNodeAccess;
```

File

XMLDoc

Description

The XML Data Binding Wizard uses TXMLNode as a base class for the classes it generates to represent nodes that have child nodes. For example, if the XML document includes the following:

```
<Address country="US">
  <Name>
    <First> John </First>
    <MI> Q. </MI>
    <Last> Public </Last>
  </Name>
  <Street> 123 Easy Street </Street>
  <City> Anytown </City>
  <State> CA </State>
</Address>
```

then the XML Data Binding Wizard generates TXMLNode descendants for the elements Address and Name. Child nodes and node attributes appear as properties of the TXMLNode descendant.

Note: The XML Data Binding Wizard also generates code that can create TXMLNode objects for attributes as well as element nodes. However, there are no special TXMLNode descendants for attribute nodes. Rather, each attribute is represented directly by TXMLNode.

TXMLNode implements the IXMLNode (see page 7669) interface. Each TXMLNode descendant implements a class-specific interface that descends from IXMLNode (see page 7669). Typically, applications do not work directly with TXMLNode. Instead, they use the IXMLNode (see page 7669) interface or the IXMLNode (see page 7669) descendant that is implemented by a TXMLNode descendant.

See Also

TXMLDocument (see page 7347)

TXMLNodeCollection (see page 7444)

IXMLNode (see page 7669)

TXMLNodeList (see page 7460)

79.1.2.1 TXMLNode Members

The following tables list the members exposed by TXMLNode.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.

	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.

R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 3345)	Responds after the last constructor executes.
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7400)	Instantiates a Create object.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
V	ToString (see page 3380)	Returns a string containing the class name.

	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Attributes (see page 7688)	Represents the values of this node's attributes.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

79.1.2.2 TXMLNode.Create Constructor

Instantiates a Create object.

Pascal

```
constructor Create(const ADOMNode: IDOMNode; const AParentNode: TXMLNode; const OwnerDoc:
TXMLDocument);
```

C++

```
__fastcall TXMLNode(const IDOMNode ADOMNode, const TXMLNode AParentNode, const TXMLDocument
OwnerDoc);
```

Description

Create instantiates a new Create object to wrap an existing DOM node implementation. Using the first syntax,

ADOMNode is the interface for the DOM implementation of the node.

AParentNode is the implementation of the parent node of this node.

OwnerDoc represents the XML document in which the new node occurs.

Using the second syntax,

HostNode (see page 7441) is another Create object that represents the host node of this node. In this case, this node is a "hosted" node, which means it shares the same underlying parent and DOM node as the host node, but represents a different view of the host node. For example, a node that contains some child nodes that are all of the same type creates a hosted node to represent only the set of repeating child nodes.

Typically, applications do not directly call the Create constructor. Instead, new nodes are created automatically for an XML document as necessary when the application reads their interfaces, starting with the DocumentElement property of the TXMLDocument (see page 7347) object. To add new nodes to a document, applications call the parent node's AddChild (see page 7404) method.

See Also

OwnerDocument (see page 7442)

ParentNode (see page 7443)

DocumentElement (see page 7383)

AddChild (see page 7404)

79.1.2.3 TXMLNode.CreateHosted Constructor

Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.

Pascal

```
constructor CreateHosted(HostNode: TXMLNode);
```

C++

```
__fastcall TXMLNode(TXMLNode HostNode);
```

Description

CreateHosted instantiates a new TXMLNode (see page 7396) object to represent a specific view on an existing TXMLNode (see page 7396) object.

HostNode (see page 7441) is another TXMLNode (see page 7396) object that represents the host node of this node. The new node becomes a "hosted" node, which means it shares the same underlying parent and DOM node as the host node, but represents a different view of the host node. For example, a node that contains some child nodes that are all of the same type creates a hosted node to represent only the set of repeating child nodes.

Typically, applications do not directly call CreateHosted to create a hosted node. Instead, this method is called in code generated by the XML Data Binding Wizard, or by the CreateCollection (see page 7411) method of the host node.

HostNode (see page 7441) (or, if HostNode (see page 7441) is hosted, the host node of HostNode (see page 7441)) becomes the value of the new node's HostNode (see page 7441) property, and the new node is added to the HostedNodes (see page 7441) property list of the new host node.

See Also

HostedNodes (see page 7441)

HostNode (see page 7441)

OnHostAttrNotify (see page 7443)

OnHostChildNotify (see page 7444)

CreateCollection (see page 7411)

79.1.2.4 TXMLNode.Destroy Destructor

Frees the memory associated with the TXMLNode (see page 7396) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLNode();
```

Description

Do not explicitly destroy TXMLNode (see page 7396) objects. As interfaced objects, XML nodes are destroyed automatically when the reference count of their IXMLNode (see page 7669) interface drops to zero.

See Also

System::TInterfacedObject::RefCount (see page 3347)

79.1.2.5 TXMLNode Methods

The methods of the TXMLNode class are listed here.

Protected Methods

	Name	Description
	_AddRef (see page 7403)	Increments the reference count for this node.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7408)	Responds to notifications of changes in the list of child nodes.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNodeName (see page 7421)	Returns the node name.

	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.
	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7433)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.

79.1.2.5.1 TXMLNode._AddRef Method

Increments the reference count for this node.

Pascal

```
function _AddRef: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _AddRef();
```

Description

__AddRef implements the IInterface (see page 3502) or IUnknown (see page 3756) method that is called when an application obtains a reference to this node's interface pointer. __AddRef increments the RefCount property. In addition, to ensure that the document object is alive, it also increments the reference count for the owning document.

When the caller is through with the interface, it calls _Release (see page 7404). This allows the node to free itself when it is no longer used (when the reference count goes to zero).

See Also

_Release (see page 7404)

OwnerDocument (see page 7442)

System::TInterfacedObject::RefCount (see page 3347)

79.1.2.5.2 TXMLNode._Release Method

Decrements the reference count for this node.

Pascal

```
function _Release: Integer; stdcall;
```

C++

```
__fastcall __stdcall int _Release();
```

Description

__Release implements the IInterface (see page 3502) or IUnknown (see page 3756) method that is called when an application releases a reference to this node's interface pointer. __Release decrements the RefCount property. In addition, it also decrements the reference count for the owning document, which it incremented when the _AddRef (see page 7403) method was called.

When __Release causes the node's reference count to drop to zero, __Release causes the node to free itself.

See Also

_AddRef (see page 7403)

OwnerDocument (see page 7442)

System::TInterfacedObject::RefCount (see page 3347)

79.1.2.5.3 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	TXMLNode.AddChild (DOMString, DOMString, Boolean, Integer) (see page 7404)	Adds a new child node to this node.
	TXMLNode.AddChild (DOMString, DOMString, TXMLNodeClass, Integer) (see page 7404)	Adds a new child node to this node.
	TXMLNode.AddChild (DOMString, Integer) (see page 7404)	Adds a new child node to this node.

79.1.2.5.3.1 TXMLNode.AddChild Method (DOMString, DOMString, Boolean, Integer)

Adds a new child node to this node.

Pascal

```
function AddChild(const TagName: DOMString; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; GenPrefix: Boolean
= False; Index: Integer = -1): IXMLNode; overload;
```

```
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; NodeClass:
TXMLNodeClass; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI, Boolean
GenPrefix = False, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI,
TXMLNodeClass NodeClass, int Index = -1);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected AddChild method. Only the first two versions of AddChild are available on the IXMLNode (see page 7669) interface. The third syntax is used by TXMLNode (see page 7396) descendants that the XML data binding wizard creates.

AddChild creates a new element node as the child of this node.

TagName provides the tag name of the newly created node.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

NamespaceURI identifies the namespace that includes the new node's definition. If NamespaceURI is omitted (the first syntax), the namespace of the new node is deduced from the namespace prefix of TagName. If TagName has no namespace prefix and there is no NamespaceURI parameter, the namespace is the same as this node's NamespaceURI property.

GenPrefix controls whether AddChild generates a namespace prefix for the namespace URI when the NamespaceURI parameter identifies a URI that is not already declared in the document. Note that setting GenPrefix to true does not cause AddChild to add a declaration of the new namespace prefix unless the document's Options property includes doNamespaceDecl.

NodeClass identifies the implementation class to use for the new child node. It must be TXMLNode (see page 7396) or one of its descendants.

AddChild returns the interface for the newly created child node.

Tip: AddChild only adds element nodes. To add other types of child nodes, use the XML document's CreateNode method instead, and then add the resulting node to the ChildNodes (see page 7439) property array.

See Also

ChildNodes (see page 7439)

AddChild (see page 7672)

CreateNode (see page 7357)

Options (see page 7389)

CreateElement (see page 7357)

NSPrefixBase (see page 7389)

DeclareNamespace (see page 7412)

79.1.2.5.4 TXMLNode.AddHostedNode Method

Adds a node to the HostedNodes (see page 7441) property list.

Pascal

```
procedure AddHostedNode(Node: TXMLNode);
```

C++

```
__fastcall AddHostedNode(TXMLNode Node);
```

Description

AddHostedNode is used internally to add nodes to the HostedNodes (see page 7441) array.

Node is the node to add to the array.

See Also

HostedNodes (see page 7441)

RemoveHostedNode (see page 7431)

79.1.2.5.5 TXMLNode.AttributeListNotify Method

Responds to notifications of changes in the list of attributes.

Pascal

```
procedure AttributeListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean);
```

C++

```
__fastcall AttributeListNotify(TNodeListOperation Operation, IXMLNode Node, const OleVariant
IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the AttributeNodes (see page 7438) property calls AttributeListNotify when attribute nodes are inserted or removed, or when a new attribute node needs to be created. AttributeListNotify generates a BeforeNodeChange or AfterNodeChange event on the document object, and, if the change has not yet occurred, updates the underlying DOM implementation to reflect the change.

Operation indicates what type of change occurred.

Node is the interface for the attribute node that was added or removed. If a new attribute node needs to be created, Node returns the new node.

IndexOrName is the name of the attribute when Operation is nlCreateNode.

BeforeOperation indicates whether the call occurs before the attribute node is added or removed.

See Also

AttributeNodes (see page 7438)

TNodeListNotification (see page 7499)

BeforeNodeChange (see page 7394)

AfterNodeChange (see page 7393)

79.1.2.5.6 TXMLNode.CheckNotHosted Method

Raises an EXMLDocError (see page 7639) exception if this is a hosted node.

Pascal

```
procedure CheckNotHosted;
```

C++

```
__fastcall CheckNotHosted();
```

Description

Applications can't call this protected method. It is used internally in methods that can only be used by the main node that corresponds to an element. If the node represents a hosted node (that is, if the node is used to represent a subset of the main, or host, node), then CheckNotHosted raises an EXMLDocError (see page 7639) exception.

For example, the CreateCollection (see page 7411) method calls CheckNotHosted, because only the main node for an element can host collection nodes.

See Also

HostNode (see page 7441)

HostedNodes (see page 7441)

EXMLDocError (see page 7639)

CreateCollection (see page 7411)

79.1.2.5.7 TXMLNode.CheckReadOnly Method

Raises an EXMLDocError (see page 7639) exception if this is a read-only node.

Pascal

```
procedure CheckReadOnly;
```

C++

```
__fastcall CheckReadOnly();
```

Description

Applications can't call this protected method. It is used internally in methods that change the value, children, or attributes of the node, to check whether the change is permitted, and if not, raise an exception.

See Also

GetReadOnly (see page 7425)

SetReadOnly (see page 7436)

ReadOnly (see page 7696)

EXMLDocError (see page 7639)

79.1.2.5.8 TXMLNode.CheckTextNode Method

Generates a DOM text node implementation if it does not yet exist.

Pascal

```
procedure CheckTextNode;
```

C++

```
__fastcall CheckTextNode();
```

Description

Applications can't call this protected method. It is used internally to generate a DOM node implementation for text values.

If the XML node is an element that has child nodes (other than a single DOM text node), CheckTextNode raises an exception.

79.1.2.5.9 TXMLNode.ChildListNotify Method

Responds to notifications of changes in the list of child nodes.

Pascal

```
procedure ChildListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean); virtual;
```

C++

```
virtual __fastcall ChildListNotify(TNodeListOperation Operation, IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the ChildNodes (see page 7439) property calls ChildListNotify when child nodes are inserted or removed, or when a node needs to be created. ChildListNotify generates a BeforeNodeChange or AfterNodeChange event on the document object, and, if the change has not yet occurred, updates the underlying DOM implementation to reflect the change.

Operation indicates what type of change is occurring.

Node is the interface for the child node about which this method is called. When nodes are added or removed, this is the node that is added or removed. When a node needs to be created, this parameter returns the newly created node.

IndexOrName is the index of the node that is added or removed when Operation is nlInsert or nlRemove. When Operation is nlCreateNode, IndexOrName is the name of the child node to create.

BeforeOperation indicates whether the call occurs before the child node is added or removed.

See Also

ChildNodes (see page 7439)

AddChild (see page 7404)

TNodeListNotification (see page 7499)

BeforeNodeChange (see page 7394)

AfterNodeChange (see page 7393)

79.1.2.5.10 TXMLNode.ClearDocumentRef Method

Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).

Pascal

```
procedure ClearDocumentRef;
```

C++

```
__fastcall ClearDocumentRef();
```

Description

Applications can't call this protected method. The document that contains this node calls ClearDocumentRef when it is freed so that the node does not maintain a reference to a non-existent document.

In addition to setting the OwnerDocument (see page 7442) property to nil (Delphi) or NULL (C++), ClearDocumentRef propagates this call to all of this nodes child nodes and attribute nodes.

See Also

OwnerDocument (see page 7442)

ChildNodes (see page 7439)

AttributeNodes (see page 7438)

79.1.2.5.11 TXMLNode.CloneNode Method

Creates a copy of this node and returns its interface.

Pascal

```
function CloneNode(Deep: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode CloneNode(Boolean Deep);
```

Description

Applications can't call the protected CloneNode method. To create a copy of this node, use the CloneNode method of this node's IXMLNode (see page 7669) interface.

CloneNode returns a copy of this node that has no parent. Any attributes of this node are copied as well, including attribute nodes that represent default values.

Deep indicates whether the child nodes of this node should be cloned as well. For example, if an element node contains any text, CloneNode only copies then text when deep is true, because in the underlying DOM implementation, the text is contained in a child node.

See Also

ChildNodes (see page 7439)

AddChild (see page 7404)

ParentNode (see page 7443)

CloneNode (see page 7673)

cloneNode (see page 7560)

79.1.2.5.12 TXMLNode.CreateAttributeList Method

Creates the object that implements the AttributeNodes (see page 7438) property.

Pascal

```
function CreateAttributeList: IXMLNodeList; dynamic;
```

C++

```
__fastcall IXMLNodeList CreateAttributeList();
```

Description

Applications can't call the protected CreateAttributeList method. This method is used internally the first time an application reads the AttributeNodes (see page 7438) property.

CreateAttributeList creates an implementation class for the AttributeNodes (see page 7438) property, telling it to call the AttributeListNotify (see page 7406) method when it changes the list of attributes for this node, and filling it with nodes based on the current attributes of this node.

See Also

AttributeNodes (see page 7438)

AttributeListNotify (see page 7406)

79.1.2.5.13 TXMLNode.CreateAttributeNode Method

Creates a new TXMLNode (see page 7396) object for an attribute of this node.

Pascal

```
function CreateAttributeNode(const ADOMNode: IDOMNode): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode CreateAttributeNode(const IDOMNode ADOMNode);
```

Description

Applications can't call the protected CreateAttributeNode method. TXMLNode (see page 7396) uses this method internally to generate the nodes that populate the AttributeNodes (see page 7438) property.

CreateAttributeNode creates a new TXMLNode (see page 7396) object for the specified IDOMNode (see page 7557) interface and returns its interface.

ADOMNode is the interface for the DOM implementation's version of the attribute node.

See Also

AttributeNodes (see page 7438)

79.1.2.5.14 TXMLNode.CreateChildList Method

Creates the object that implements the ChildNodes (see page 7439) property.

Pascal

```
function CreateChildList: IXMLNodeList; dynamic;
```

C++

```
__fastcall IXMLNodeList CreateChildList();
```

Description

Applications can't call the protected CreateChildList method. This method is used internally the first time an application reads the ChildNodes (see page 7439) property.

CreateChildList creates an implementation class for the ChildNodes (see page 7439) property, telling it to call the ChildListNotify (see page 7408) method when it changes the list of attributes for this node, and filling it with nodes that are based on the current children of this node.

See Also

ChildNodes (see page 7439)

ChildListNotify (see page 7408)

79.1.2.5.15 TXMLNode.CreateChildNode Method

Creates a new TXMLNode (see page 7396) object for a child node of this node.

Pascal

```
function CreateChildNode(const ADOMNode: IDOMNode): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode CreateChildNode(const IDOMNode ADOMNode);
```

Description

Applications can't call the protected CreateChildNode method. This method is used internally to create the implementation object for a new child node.

CreateChildNode creates a new TXMLNode (see page 7396) descendant for the specified IDOMNode (see page 7557) interface, with this node as the parent. If the node name of the DOM node matches one of the names listed by the ChildNodeClasses (see page 7438) property, the corresponding descendant of TXMLNode (see page 7396) becomes the implementation class of the new node. Otherwise, the new node is an instance of TXMLNode (see page 7396).

ADOMNode is the interface for the DOM implementation's version of the node.

CreateChildNode returns the interface for the newly created child node.

See Also

Create (see page 7400)

ChildNodeClasses (see page 7438)

CreateDOMNode (see page 7488)

79.1.2.5.16 TXMLNode.CreateCollection Method

Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.

Pascal

```
function CreateCollection(const CollectionClass: TXMLNodeCollectionClass; const ItemInterface:
TGuid; const ItemTag: DOMString; ItemNS: DOMString = ''): TXMLNodeCollection;
```

C++

```
__fastcall TXMLNodeCollection CreateCollection(const TXMLNodeCollectionClass CollectionClass,
const TGuid ItemInterface, const DOMString ItemTag, DOMString ItemNS = '');
```

Description

Applications can't call the protected CreateCollection method. It is used internally to generate the collections that are listed in the HostedNodes (see page 7441) property.

CollectionClass specifies the class (a TXMLNodeCollection (see page 7444) descendant) that implements the collection of child nodes.

ItemInterface specifies the interface for each child node in the collection.

ItemTag specifies the tag name for each child node in the collection.

ItemNS specifies the namespace in which ItemTag is defined.

CreateCollection returns the new hosted collection node. This new node has the same parent node, DOM interface, attribute nodes, and child nodes as this node, but it also maintains a list of only those child nodes with the specified tag and interface. The new collection node lists this node as the value of its HostNode (see page 7441) property.

Note: CreateCollection raises an exception if this is a hosted node.

See Also

HostedNodes (see page 7441)

ChildNodes (see page 7439)

ParentNode (see page 7443)

DOMNode (see page 7440)

HostNode (see page 7441)

ItemInterface (see page 7458)

ItemTag (see page 7459)

ItemNS (see page 7458)

79.1.2.5.17 TXMLNode.DeclareNamespace Method

Adds an attribute to the node that declares a specified namespace URI.

Pascal

```
procedure DeclareNamespace(const Prefix: DOMString; const URI: DOMString);
```

C++

```
__fastcall DeclareNamespace(const DOMString Prefix, const DOMString URI);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected DeclareNamespace method. DeclareNamespace adds an attribute to this node that declares the specified namespace URI as the value of the specified prefix.

Prefix is the namespace prefix that appears in tag names that reference the namespace.

URI is the corresponding namespace URI.

Note: This method is called automatically for newly created nodes when the document's Options property includes doNamespaceDecl.

See Also

Options (see page 7389)

AttributeNodes (see page 7438)

FindNamespaceDecl (see page 7414)

DeclareNamespace (see page 7674)

79.1.2.5.18 TXMLNode.DOMElement Method

Returns the IDOMElement (see page 7542) interface for this node if it exists.

Pascal

```
function DOMElement: IDOMElement;
```

C++

```
__fastcall IDOMElement DOMElement();
```

Description

Application's can't call the protected DOMElement method. TXMLNode (see page 7396) uses this method internally to obtain the DOM element interface, which it uses for working with attributes.

If the TXMLNode (see page 7396) instance does not represent an element node, DOMElement raises an EXMLDocError (see page 7639) exception.

See Also

AttributeNodes (see page 7438)

GetAttribute (see page 7415)

SetAttribute (see page 7432)

79.1.2.5.19 TXMLNode.DoNodeChange Method

Generates a BeforeNodeChange or AfterNodeChange event on this node's document.

Pascal

```
procedure DoNodeChange(ChangeType: TNodeChange; BeforeOperation: Boolean); virtual;
```

C++

```
virtual __fastcall DoNodeChange(TNodeChange ChangeType, Boolean BeforeOperation);
```

Description

Applications can't call the protected DoNodeChange method. It is called internally before and after every change that is made to the node. DoNodeChange checks whether the node's OwnerDocument (see page 7442) property is set, and if so, generates a BeforeNodeChange or AfterNodeChange event on that document, as appropriate.

ChangeType indicates the type of change that is occurring to the node.

BeforeOperation is true if the change is about to occur, false if the change has just occurred.

See Also

AfterNodeChange (see page 7393)

BeforeNodeChange (see page 7394)

OwnerDocument (see page 7442)

79.1.2.5.20 TXMLNode.FindHostedNode Method

Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.

Pascal

```
function FindHostedNode(const NodeClass: TXMLNodeClass): IXMLNode;
```

C++

```
__fastcall IXMLNode FindHostedNode(const TXMLNodeClass NodeClass);
```

Description

Applications can't call the protected FindHostedNode method. Descendant classes call this method to locate a hosted node of a particular class type.

FindHostedNode searches the nodes in the HostedNodes (see page 7441) list and returns the first one it finds that is an instance of descendant of NodeClass. If this node does not have a hosted node of the specified class, FindHostedNode returns

nil (Delphi) or NULL (C++).

See Also

HostedNodes (see page 7441)

CreateCollection (see page 7411)

79.1.2.5.21 TXMLNode.FindNamespaceDecl Method

Returns the attribute node that declares a specified Namespace URI.

Pascal

```
function FindNamespaceDecl(const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNamespaceDecl(const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected FindNamespaceDecl method. FindNamespaceDecl searches the node hierarchy that contains this node for an attribute that specifies the mapping between a namespace prefix and the namespace URI specified by NamespaceURI. If it finds such a mapping, FindNamespaceDecl returns the interface for the Attribute node. If it does not find a mapping, FindNamespaceDecl returns nil (Delphi) or NULL (C++).

NamespaceURI is the namespace URI to locate.

See Also

FindNamespaceURI (see page 7414)

DeclareNamespace (see page 7412)

FindNamespaceDecl (see page 7674)

79.1.2.5.22 TXMLNode.FindNamespaceURI Method

Returns the namespace URI for a namespace prefix or qualified tag name.

Pascal

```
function FindNamespaceURI(const TagOrPrefix: DOMString): DOMString;
```

C++

```
__fastcall DOMString FindNamespaceURI(const DOMString TagOrPrefix);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected FindNamespaceURI method. FindNamespaceDecl (see page 7414) searches the node hierarchy that contains this node for an attribute that specifies the mapping between the namespace prefix identified by TagOrPrefix and a namespace URI. If it finds such a mapping, FindNamespaceURI returns the namespace URI. If it does not find a mapping, FindNamespaceURI returns an empty string.

TagOrPrefix is the namespace prefix or a tag name that includes the namespace prefix.

See Also

GetNamespaceURI (see page 7421)

GetPrefix (see page 7424)

FindNamespaceDecl (see page 7414)

FindNamespaceURI (see page 7674)

79.1.2.5.23 TXMLNode.GetAttribute Method

Returns the value of one of this node's attributes.

Pascal

```
function GetAttribute(const AttrName: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttribute(const DOMString AttrName);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetAttribute method. GetAttribute returns the value of an attribute of this node.

AttrName is the name of the attribute.

GetAttribute returns the value of the specified attribute. Typically, this value is a string. If the node does not have an attribute with the name specified by AttrName, the value that GetAttribute returns is determined by the document's Options property. When Options include doAttrNull, GetAttribute returns a Null (see page 5056) Variant (see page 3421) for missing attributes. When Options does not include doAttrNull, GetAttribute returns an empty string as the value of missing attributes.

If this node is not an element node, GetAttribute raises an EXMLDocError (see page 7639) exception.

See Also

SetAttribute (see page 7432)

AttributeNodes (see page 7438)

GetAttributeNS (see page 7416)

79.1.2.5.24 TXMLNode.GetAttributeNodes Method

Returns the value of the AttributeNodes (see page 7438) property.

Pascal

```
function GetAttributeNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetAttributeNodes();
```

Description

GetAttributeNodes is the read implementation of the AttributeNodes (see page 7438) property on the IXMLNode (see page 7669) interface for this node.

GetAttributeNodes generates the list that implements the AttributeNodes (see page 7438) property the first time an application reads the AttributeNodes (see page 7438) property.

See Also

GetAttribute (see page 7415)

AttributeNodes (see page 7438)

79.1.2.5.25 TXMLNode.GetAttributeNS Method

Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.

Pascal

```
function GetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetAttributeNS method. GetAttributeNS returns the value of an attribute of this node.

AttrName is the name of the attribute.

NamespaceURI identifies the namespace in which the attribute resides.

GetAttributeNS returns the value of the specified attribute. If AttrName and NamespaceURI do not identify an attribute of this node, GetAttributeNS returns a Null (see page 5056) Variant (see page 3421).

If this node is not an element node, GetAttributeNS raises an EXMLDocError (see page 7639) exception.

See Also

GetAttribute (see page 7415)

Attributes (see page 7688)

GetNamespaceURI (see page 7421)

SetAttributeNS (see page 7433)

AttributeNodes (see page 7438)

79.1.2.5.26 TXMLNode.GetChildNodeClasses Method

Returns a list of the classes that can implement child nodes of this node.

Pascal

```
function GetChildNodeClasses: TNodeClassArray;
```

C++

```
__fastcall TNodeClassArray GetChildNodeClasses();
```

Description

GetChildNodeClasses is used internally to obtain a list of the classes that can be used to represent child nodes of this node.

If this node is not a hosted node, GetChildNodeClasses returns an array containing every class that has been registered using the RegisterChildNode (see page 7430) method. If this node is a hosted node, GetChildNodeClasses returns the ChildNodeClasses (see page 7438) property of the host node.

See Also

ChildNodeClasses (see page 7438)

RegisterChildNode (see page 7430)

HostNode (see page 7441)

79.1.2.5.27 TXMLNode.GetChildNodes Method

Returns the value of the ChildNodes (see page 7439) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the protected read implementation of the ChildNodes (see page 7439) property.

GetChildNodes generates the list that implements the ChildNodes (see page 7439) property the first time an application reads the ChildNodes (see page 7439) property.

See Also

GetChildValue (see page 7417)

ChildNodes (see page 7439)

79.1.2.5.28 TXMLNode.GetChildValue Method

Returns the value of a specified child node.

Pascal

```
function GetChildValue(const IndexOrName: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetChildValue(const OleVariant IndexOrName);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetChildValue method. GetChildValue returns the value of the node.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

If IndexOrName does not identify a child of this node, GetChildValue nil (Delphi) or NULL (C++). Otherwise, the return value depends on the type of the child node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, GetChildValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

Note: Attributes are not considered child nodes of a node. To get the values of this node's attributes, use the GetAttribute (

see page 7415) method instead.

See Also

ChildNodes (see page 7439)

ParentNode (see page 7443)

GetAttribute (see page 7415)

GetNodeValue (see page 7423)

79.1.2.5.29 TXMLNode.GetCollection Method

Returns the value of the Collection (see page 7439) property.

Pascal

```
function GetCollection: IXMLNodeCollection;
```

C++

```
__fastcall IXMLNodeCollection GetCollection();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetCollection method. GetCollection returns the value of the Collection (see page 7439) property. If this node is not a member of a collection (that is, if it does not represent one of a set of child nodes that are all the same type), then GetCollection returns nil (Delphi) or NULL (C++).

See Also

Collection (see page 7439)

79.1.2.5.30 TXMLNode.GetDOMNode Method

Returns the value of the DOMNode (see page 7440) property.

Pascal

```
function GetDOMNode: IDOMNode;
```

C++

```
__fastcall IDOMNode GetDOMNode();
```

Description

GetDOMNode is the protected read implementation of the DOMNode (see page 7440) property.

See Also

DOMNode (see page 7440)

79.1.2.5.31 TXMLNode.GetHasChildNodes Method

Indicates whether this node has any child nodes.

Pascal

```
function GetHasChildNodes: Boolean;
```

C++

```
__fastcall Boolean GetHasChildNodes();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetHasChildNodes method. GetHasChildNodes returns true if the underlying DOM node has any child nodes, false if it doesn't.

Note that even if the underlying DOM node has a child node, there may not be a TXMLNode (see page 7396) object to represent that child. For example, text nodes do not, by default, have corresponding TXMLNode (see page 7396) wrappers. Rather, the text becomes the value of the parent element node.

See Also

ChildNodes (see page 7439)

DOMNode (see page 7440)

GetIsTextElement (see page 7419)

79.1.2.5.32 TXMLNode.GetHostedNodes Method

Returns a list an array of the hosted nodes that share this node's attribute and child node lists.

Pascal

```
function GetHostedNodes: TXMLNodeArray;
```

C++

```
__fastcall TXMLNodeArray GetHostedNodes();
```

Description

GetHostedNodes is the read implementation of the HostedNodes (see page 7441) property. It returns a list an array of all nodes that represent aspects of this node, sharing its attributes and list of child nodes.

See Also

HostedNodes (see page 7441)

79.1.2.5.33 TXMLNode.GetHostNode Method

Returns the value of the HostNode (see page 7441) property.

Pascal

```
function GetHostNode: TXMLNode;
```

C++

```
__fastcall TXMLNode GetHostNode();
```

Description

GetHostNode returns the node that uses this node to represent a particular aspect of it, sharing the same attributes and list of child nodes. If this is not a hosted node, GetHostNode returns nil (Delphi) or NULL (C++).

See Also

HostNode (see page 7441)

79.1.2.5.34 TXMLNode.GetIsTextElement Method

Indicates whether the node has a single text value.

Pascal

```
function GetIsTextElement: Boolean;
```

C++

```
__fastcall Boolean GetIsTextElement();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetIsTextElement method. GetIsTextElement returns true if the node represents an XML fragment such as

```
<Title> Understanding XML </Title>
```

where the node has a tag name (in this case Title) and a single text value (in this case 'Understanding XML').

GetIsTextElement returns false if the node is any other type of node.

Note: In the underlying DOM implementation, nodes for which GetIsTextElement is true are element nodes that have a single child node that is a text node. TXMLNode (see page 7396) flattens this out so that the child text node does not appear as a child node. Instead, the text can be accessed through the GetText (see page 7426) method.

See Also

GetText (see page 7426)

GetHasChildNodes (see page 7418)

79.1.2.5.35 TXMLNode.GetLocalName Method

Returns the name of the node without any namespace prefix.

Pascal

```
function GetLocalName: DOMString;
```

C++

```
__fastcall DOMString GetLocalName();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetLocalName method. GetLocalName returns the local name of the underlying DOM node.

For element nodes, the local name is derived from the name that appears in the XML tag. For example, the local name for <xsi:Name First="John" Last="Doe"> is 'Name'.

For attribute nodes, the local name is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <Name First=John Last=Doe> results in two attribute nodes, with the local names 'First' and 'Last'.

For any other node types, the local name is nil (Delphi) or NULL (C++).

Note: If both GetPrefix (see page 7424) and GetLocalName return a value for a node, the node name (returned by GetNodeName (see page 7421)) is the combination of these two values.

See Also

GetNodeName (see page 7421)

GetNodeType (see page 7422)

GetPrefix (see page 7424)

GetNamespaceURI (see page 7421)

localName (see page 7570)

79.1.2.5.36 TXMLNode.GetNamespaceURI Method

Returns the URI for the namespace used in the qualified node name.

Pascal

```
function GetNamespaceURI: DOMString;
```

C++

```
__fastcall DOMString GetNamespaceURI();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetNamespaceURI method. GetNamespaceURI returns the URI for the namespace that should be used to interpret the node's name.

Only element and attribute nodes can have a namespace URI value. For any other node type, GetNamespaceURI returns nil (Delphi) or NULL (C++).

The namespace URI is only available if it is explicitly assigned when the node is created. By default, TXMLNode (see page 7396) does not assign namespaces to the new nodes it creates.

See Also

GetNodeName (see page 7421)

GetNodeType (see page 7422)

GetPrefix (see page 7424)

GetLocalName (see page 7420)

CreateDOMNode (see page 7488)

namespaceURI (see page 7571)

79.1.2.5.37 TXMLNode.GetNodeName Method

Returns the node name.

Pascal

```
function GetNodeName: DOMString;
```

C++

```
__fastcall DOMString GetNodeName();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetNodeName method. GetNodeName returns the name of the underlying DOM node.

The node's name depends on the type of the node, as indicated in the following table:

NodeType	NodeName
ntAttribute	The attribute name
ntElement	The tag name
ntText	'#text'
ntCData	'#cdata-section'

ntEntityRef	The name of the entity reference.
ntEntity	The entity name
ntProcessingInstr	the target of the processing instruction
ntComment	'#comment'
ntDocument	'#document'
ntDocType	The document type name
ntDocFragment	'#document-fragment'
ntNotation	The notation name

Note: If GetPrefix (see page 7424) and GetLocalName (see page 7420) both return values for a node, the node name is the combination of these two values.

See Also

GetNodeType (see page 7422)

GetPrefix (see page 7424)

GetLocalName (see page 7420)

79.1.2.5.38 TXMLNode.GetNodeObject Method

Returns a reference to this TXMLNode (see page 7396) object.

Pascal

```
function GetNodeObject: TXMLNode;
```

C++

```
__fastcall TXMLNode GetNodeObject();
```

Description

GetNodeObject returns Self (Delphi) or this (C++).

79.1.2.5.39 TXMLNode.GetNodeType Method

Returns the type of the node.

Pascal

```
function GetNodeType: TNodeType;
```

C++

```
__fastcall TNodeType GetNodeType();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetNodeType method. GetNodeType returns the type of the underlying DOM node that this TXMLNode (see page 7396) object represents.

See Also

GetIsTextElement (see page 7419)

79.1.2.5.40 TXMLNode.GetNodeValue Method

Returns the value of this node.

Pascal

```
function GetNodeValue: OleVariant;
```

C++

```
__fastcall OleVariant GetNodeValue();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetNodeValue method. GetNodeValue returns the value of the node. This value depends on the type of the node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, GetNodeValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

See Also

GetChildValue (see page 7417)

GetText (see page 7426)

79.1.2.5.41 TXMLNode.GetOwnerDocument Method

Returns the interface for the document in which this node appears.

Pascal

```
function GetOwnerDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument GetOwnerDocument();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetOwnerDocument method. GetOwnerDocument returns the value of the OwnerDocument (see page 7442) property.

See Also

OwnerDocument (see page 7442)

79.1.2.5.42 TXMLNode.GetParentNode Method

Returns the interface for the node's parent node.

Pascal

```
function GetParentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetParentNode();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetParentNode method. GetParentNode returns the interface of the node's parent node.

See Also

ParentNode (see page 7443)

79.1.2.5.43 TXMLNode.GetPrefix Method

Returns the namespace prefix of the node's name.

Pascal

```
function GetPrefix: DOMString;
```

C++

```
__fastcall DOMString GetPrefix();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetPrefix method. GetPrefix returns the namespace prefix of an element or attribute node's name. A namespace prefix is a symbolic name for a namespace URI.

For element nodes, the namespace prefix is derived from the name that appears in the XML tag. For example, the namespace prefix for <xsi:Name First="John" Last="Doe"> is 'xsi'.

For attribute nodes, the namespace prefix is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <xsi:Name xsn:First="John" Last="Doe"> results in two attribute nodes, with namespace prefixes of 'xsn' and '', respectively. Note that the attribute never inherits a namespace prefix from the node to which it is attached.

For any other node types, the namespace prefix is nil (Delphi) or NULL (C++).

Note: If both GetPrefix and GetLocalName (see page 7420) return a value for a node, the node name (returned by GetNodeName (see page 7421)) is the combination of these two values.

See Also

GetLocalName (see page 7420)

GetNodeType (see page 7422)

GetNamespaceURI (see page 7421)

GetPrefixedName (see page 7425)

79.1.2.5.44 TXMLNode.GetPrefixedName Method

Returns a node name includes a namespace prefix for a specified namespace URI.

Pascal

```
function GetPrefixedName(const Name: DOMString; const NamespaceURI: DOMString): DOMString;
```

C++

```
__fastcall DOMString GetPrefixedName(const DOMString Name, const DOMString NamespaceURI);
```

Description

GetPrefixedName adds a prefix to a local node name based on the specified namespace URI. If there is no corresponding namespace already declared or if the name is already prefixed, GetPrefixedName returns the original node name.

Name is the node name to which the namespace prefix should be added.

NamespaceURI is the URI for the namespace that should be identified by the namespace prefix.

Note: If the document containing this node does not include doAutoPrefix in its Options property, GetPrefixedName always returns the value of Name, without trying to generate or add a namespace prefix.

See Also

GetLocalName (see page 7420)

GetPrefix (see page 7424)

GetNamespaceURI (see page 7421)

Options (see page 7389)

79.1.2.5.45 TXMLNode.GetReadOnly Method

Indicates whether the node can be modified.

Pascal

```
function GetReadOnly: Boolean;
```

C++

```
__fastcall Boolean GetReadOnly();
```

Description

GetReadOnly returns true if the node can't be modified. Read (see page 3634)-only nodes raise an EXMLDocError (see page 7639) exception when an attempt is made to change their value, child nodes, or attributes.

GetReadOnly returns false if the node can be modified.

Note: Use the ReadOnly property on the IXMLNode (see page 7669) interface to specify whether a node is read-only.

See Also

SetReadOnly (see page 7436)

ReadOnly (see page 7696)

EXMLDocError (see page 7639)

79.1.2.5.46 TXMLNode.GetText Method

Returns the text value of the node.

Pascal

```
function GetText: DOMString;
```

C++

```
__fastcall DOMString GetText();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetText method. GetText returns the text of the node. For example, if the node represents an XML fragment such as

```
<Title> Understanding XML </Title>
```

GetText returns the string 'Understanding XML'.

GetText is intended for use when the GetIsTextElement (see page 7419) method returns true. If GetIsTextElement (see page 7419) returns false, GetText returns an empty string if the node has no children, otherwise it raises an exception.

See Also

GetIsTextElement (see page 7419)

GetNodeValue (see page 7423)

SetText (see page 7436)

79.1.2.5.47 TXMLNode.GetXML Method

Returns the XML that corresponds to the subtree rooted at this node.

Pascal

```
function GetXML: DOMString;
```

C++

```
__fastcall DOMString GetXML();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected GetXML method. GetXML returns the XML that corresponds to this node and any child nodes it contains.

Note: GetXML is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

GetText (see page 7426)

ChildNodes (see page 7439)

DOMVendor (see page 7386)

79.1.2.5.48 HasAttribute Method

This is the overview for the HasAttribute method overload.

Overload List

	Name	Description
	TXMLNode.HasAttribute (DOMString) (see page 7427)	Indicates whether the node has an attribute with a specified name.
	TXMLNode.HasAttribute (DOMString, DOMString) (see page 7427)	Indicates whether the node has an attribute with a specified name.

79.1.2.5.48.1 TXMLNode.HasAttribute Method (DOMString)

Indicates whether the node has an attribute with a specified name.

Pascal

```
function HasAttribute(const Name: DOMString): Boolean; overload;
function HasAttribute(const Name: DOMString; const NamespaceURI: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasAttribute(const DOMString Name);
__fastcall Boolean HasAttribute(const DOMString Name, const DOMString NamespaceURI);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected HasAttribute method.

Name is the name of the attribute about which you want to know.

NamespaceURI is the namespace of the attribute.

HasAttribute returns true if this node has the specified attribute, false otherwise.

See Also

AttributeNodes (see page 7438)

GetAttribute (see page 7415)

GetAttributeNS (see page 7416)

79.1.2.5.49 HasChildNode Method

This is the overview for the HasChildNode method overload.

Overload List

	Name	Description
	TXMLNode.HasChildNode (DOMString) (see page 7427)	Indicates whether the node has a child node with the specified name.
	TXMLNode.HasChildNode (DOMString, DOMString) (see page 7427)	Indicates whether the node has a child node with the specified name.

79.1.2.5.49.1 TXMLNode.HasChildNode Method (DOMString)

Indicates whether the node has a child node with the specified name.

Pascal

```
function HasChildNode(const ChildTag: DOMString): Boolean; overload;
```

```
function HasChildNode(const ChildTag: DOMString; const NamespaceURI: DOMString): Boolean;
overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag);
__fastcall Boolean HasChildNode(const DOMString ChildTag, const DOMString NamespaceURI);
```

Description

Applications can't call the protected HasChildNode method. It is used internally to determine when the underlying DOM node has a child with the specified name.

ChildTag is the local tag name of the child to find.

NamespaceURI identifies the namespace in which ChildTag is defined.

HasChildNode returns true if the underlying DOM node includes the specified child node. Otherwise, it returns false.

See Also

ChildNodes (see page 7439)

DOMNode (see page 7440)

79.1.2.5.50 TXMLNode.InternalAddChild Method

Provides the underlying implementation for the AddChild (see page 7404) method.

Pascal

```
function InternalAddChild(NodeClass: TXMLNodeClass; const NodeName: DOMString; const
NamespaceURI: DOMString; Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode InternalAddChild(TXMLNodeClass NodeClass, const DOMString NodeName, const
DOMString NamespaceURI, int Index);
```

Description

Applications can't call the protected InternalAddChild method. It is used internally to perform all the tasks common to the various overloads of the AddChild (see page 7404) method. InternalAddChild creates a new element node as the child of this node and returns its interface.

NodeClass specifies the implementation class to use for the new child node. It must be TXMLNode (see page 7396) or one of its descendants.

NodeName specifies the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

See Also

AddChild (see page 7404)

79.1.2.5.51 TXMLNode.NestingLevel Method

Returns the number of parents for this node in the node hierarchy.

Pascal

```
function NestingLevel: Integer;
```

C++

```
__fastcall int NestingLevel();
```

Description

Application's can't call the protected NestingLevel property. It is used internally when formatting the XML text that corresponds to this XML document.

NestingLevel returns the number of ancestors for this node in the node hierarchy.

See Also

ParentNode (see page 7443)

ChildNodes (see page 7439)

79.1.2.5.52 TXMLNode.NextSibling Method

Returns the next child of this node's parent.

Pascal

```
function NextSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode NextSibling();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected NextSibling method. NextSibling returns the node that follows this one in the parent node's ChildNodes (see page 7439) property list.

If this node is the last node in its parent's child list, NextSibling raises an exception.

See Also

ParentNode (see page 7443)

PreviousSibling (see page 7430)

ChildNodes (see page 7439)

79.1.2.5.53 TXMLNode.Normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure Normalize;
```

C++

```
__fastcall Normalize();
```

Description

Use the IXMLNode (see page 7669) interface to call the protected Normalize method. Normalize consolidates adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes, only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes.

See Also

ChildNodes (see page 7439)

AttributeNodes (see page 7438)

79.1.2.5.54 TXMLNode.PreviousSibling Method

Returns the previous child of this node's parent.

Pascal

```
function PreviousSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode PreviousSibling();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected PreviousSibling method. PreviousSibling returns the node that precedes this one in the parent node's ChildNodes (see page 7439) property list.

If this node is the first node in its parent's child list, PreviousSibling raises an exception.

See Also

ParentNode (see page 7443)

NextSibling (see page 7429)

ChildNodes (see page 7439)

79.1.2.5.55 TXMLNode.RegisterChildNode Method

Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.

Pascal

```
procedure RegisterChildNode(const TagName: DOMString; ChildNodeClass: TXMLNodeClass;
NamespaceURI: DOMString = '');
```

C++

```
__fastcall RegisterChildNode(const DOMString TagName, TXMLNodeClass ChildNodeClass, DOMString
NamespaceURI = '');
```

Description

Applications can't call the protected RegisterChildNode method. It is called by the RegisterChildNodes (see page 7431) method to register a TXMLNode (see page 7396) descendant so that it can be used as a child of this node. Each TXMLNode (see page 7396) descendant is specific to the structure of the XML document, with properties that correspond to the child nodes and attributes of a node.

RegisterChildNode adds an entry to the ChildNodeClasses (see page 7438) property for the specified node class and tag name.

TagName is the tag name of the child node that the class represents.

NodeClass is the TXMLNode (see page 7396) descendant that implements child nodes that have the specified tag name.

NamespaceURI identifies the namespace in which TagName is defined.

Note: RegisterChildNode raises an exception if this is a hosted node.

See Also

RegisterChildNodes (see page 7431)

ChildNodeClasses (see page 7438)

79.1.2.5.56 TXMLNode.RegisterChildNodes Method

Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.

Pascal

```
procedure RegisterChildNodes(const TagNames: array of DOMString; const NodeClasses: array of
TXMLNodeClass);
```

C++

```
__fastcall RegisterChildNodes(const array of DOMString TagNames, const array of TXMLNodeClass
NodeClasses);
```

Description

Applications can't call the protected RegisterChildNodes method. It is used in code that the XML Data Binding wizard generates to register the special TXMLNode (see page 7396) descendants the are created for nodes in the XML document. Each of these descendant classes is specific to the structure of the XML document, with properties that correspond to the child nodes and attributes of the node.

RegisterChildNodes adds an entry to the ChildNodeClasses (see page 7438) property for each specified node class and tag name.

TagNames is an array containing the names of the child nodes that the classes represent.

TagNames_Size is the index of the last name in TagNames (one less than the number of elements).

NodeClasses is an array containing the TXMLNode (see page 7396) descendants that implement child nodes that have the tag names in the corresponding entries in TagNames.

NodeClasses_Size is the index of the last class in NodeClasses (one less than the number of elements).

See Also

RegisterChildNode (see page 7430)

ChildNodeClasses (see page 7438)

79.1.2.5.57 TXMLNode.RemoveHostedNode Method

Removes a node from the HostedNodes (see page 7441) property list.

Pascal

```
procedure RemoveHostedNode(Node: TXMLNode);
```

C++

```
__fastcall RemoveHostedNode(TXMLNode Node);
```

Description

RemoveHostedNode is used internally to remove nodes from the HostedNodes (see page 7441) array.

Node is the node to remove from the array.

See Also

HostedNodes (see page 7441)

AddHostedNode (see page 7405)

79.1.2.5.58 TXMLNode.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected Resync method. Resync forces the node to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7439)

AttributeNodes (see page 7438)

Refresh (see page 7371)

79.1.2.5.59 TXMLNode.SetAttribute Method

Sets the value of one of this node's attributes.

Pascal

```
procedure SetAttribute(const AttrName: DOMString; const Value: OleVariant);
```

C++

```
__fastcall SetAttribute(const DOMString AttrName, const OleVariant Value);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected SetAttribute method. SetAttribute sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

GetAttribute (see page 7415)

AttributeNodes (see page 7438)

SetAttributeNS (see page 7433)

79.1.2.5.60 TXMLNode.SetAttributeNodes Method

Sets the value of the AttributeNodes (see page 7438) property.

Pascal

```
procedure SetAttributeNodes(const Value: IXMLNodeList); virtual;
```

C++

```
virtual __fastcall SetAttributeNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the AttributeNodes (see page 7438) property to supply the interface that implements that property.

See Also

AttributeNodes (see page 7438)

79.1.2.5.61 TXMLNode.SetAttributeNS Method

Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

Pascal

```
procedure SetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString; const
Value: OleVariant);
```

C++

```
__fastcall SetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI, const
OleVariant Value);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected SetAttributeNS method. SetAttributeNS sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

NamespaceURI is the namespace for the attribute.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

SetAttribute (see page 7432)

GetAttributeNS (see page 7416)

AttributeNodes (see page 7438)

79.1.2.5.62 TXMLNode.SetChildNodes Method

Sets the value of the ChildNodes (see page 7439) property.

Pascal

```
procedure SetChildNodes(const Value: IXMLNodeList); virtual;
```

C++

```
virtual __fastcall SetChildNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the ChildNodes (see page 7439) property to supply the interface that implements that property.

See Also

ChildNodes (see page 7439)

79.1.2.5.63 TXMLNode.SetChildValue Method

Sets the value of a specified child node.

Pascal

```
procedure SetChildValue(const IndexOrName: OleVariant; const Value: OleVariant);
```

C++

```
__fastcall SetChildValue(const OleVariant IndexOrName, const OleVariant Value);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected SetChildValue method. SetChildValue sets the value of a specified child node to a specified value.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node. If IndexOrName does not identify a child of this node, SetChildValue raises an exception.

Value is the value to assign. It's interpretation depends on the type of the child node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, SetChildValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the child node is any other node type, SetChildValue raises an exception.	

Note: Attributes are not considered child nodes of a node. To set the values of this node's attributes, use the SetAttribute (see page 7432) method instead.

See Also

ChildNodes (see page 7439)

ParentNode (see page 7443)

SetAttribute (see page 7432)

SetNodeValue (see page 7435)

79.1.2.5.64 TXMLNode.SetCollection Method

Sets the value of the Collection (see page 7439) property.

Pascal

```
procedure SetCollection(const Value: TXMLNodeCollection);
```

C++

```
__fastcall SetCollection(const TXMLNodeCollection Value);
```

Description

SetCollection assigns Value as the value of the Collection (see page 7439) property. This protected method is called when this node is added to the collection specified by the Value property.

Value is a TXMLNodeCollection (see page 7444) descendant that represents a repeating set of child nodes, of which this node is a member.

See Also

Collection (see page 7439)

79.1.2.5.65 TXMLNode.SetNodeValue Method

Sets the value of this node.

Pascal

```
procedure SetNodeValue(const Value: OleVariant);
```

C++

```
__fastcall SetNodeValue(const OleVariant Value);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected SetNodeValue method. SetNodeValue sets the value of this node.

Value is the value to assign. It's meaning depends on the type of the node, as indicated in the following table:

NodeType	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, SetNodeValue raises an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the node is any other node type, SetNodeValue raises an exception.	

See Also

SetChildValue (see page 7434)

SetText (see page 7436)

79.1.2.5.66 TXMLNode.SetParentNode Method

Sets the value of the ParentNode (see page 7443) property.

Pascal

```
procedure SetParentNode(const Value: TXMLNode); virtual;
```

C++

```
virtual __fastcall SetParentNode(const TXMLNode Value);
```

Description

Applications can't use this protected property. It is used internally when nodes are added to the ChildNodes (see page 7439) or AttributeNodes (see page 7438) property.

See Also

ParentNode (see page 7443)

ChildNodes (see page 7439)

AttributeNodes (see page 7438)

79.1.2.5.67 TXMLNode.SetReadOnly Method

Specifies whether the node can be modified.

Pascal

```
procedure SetReadOnly(const Value: Boolean);
```

C++

```
__fastcall SetReadOnly(const Boolean Value);
```

Description

SetReadOnly assigns the read-only status of the node.

Value is the new read-only status of the node. When Value is true, the node raises an EXMLDocError (see page 7639) exception when an attempt is made to change its value, child nodes, or attributes. When Value is false, the node can be modified.

See Also

GetReadOnly (see page 7425)

ReadOnly (see page 7696)

EXMLDocError (see page 7639)

79.1.2.5.68 TXMLNode.SetText Method

Sets the text value of the node.

Pascal

```
procedure SetText(const Value: DOMString);
```

C++

```
__fastcall SetText(const DOMString Value);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected SetText method. SetText replaces the text of the node.

SetText is intended for use when the GetIsTextElement (see page 7419) method returns true. If GetIsTextElement (see page 7419) returns false, SetText adds a new text value to a node that has no children, otherwise, it raises an exception.

See Also

GetIsTextElement (see page 7419)

SetNodeValue (see page 7435)

GetText (see page 7426)

79.1.2.5.69 TransformNode Method

This is the overview for the TransformNode method overload.

Overload List

	Name	Description
	TXMLNode.TransformNode (IXMLNode, IXMLDocument) (see page 7437)	Transforms the subtree rooted at this node, using a specified XSL interface.
	TXMLNode.TransformNode (IXMLNode, WideString) (see page 7437)	Transforms the subtree rooted at this node, using a specified XSL interface.

79.1.2.5.69.1 TXMLNode.TransformNode Method (IXMLNode, IXMLDocument)

Transforms the subtree rooted at this node, using a specified XSL interface.

Pascal

```
procedure TransformNode(const stylesheet: IXMLNode; var output: WideString); overload;
procedure TransformNode(const stylesheet: IXMLNode; const output: IXMLDocument); overload;
```

C++

```
__fastcall TransformNode(const IXMLNode stylesheet, BSTR output);
__fastcall TransformNode(const IXMLNode stylesheet, const IXMLDocument output);
```

Description

Use the TXMLNode (see page 7396) object's IXMLNode (see page 7669) interface to call the protected TransformNode method.

stylesheet is a node in an XML document that represents an XSL transformation. Typically, it is the document element of that document.

output represents the result of the transformation. If output is a WideString (see page 3467), TransformNode returns a string of XML. If output is the interface to an XML document, TransformNode replaces the content of that document with the results of the transformation.

Note: TransformNode is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

GetXML (see page 7426)

79.1.2.6 TXMLNode Properties

The properties of the TXMLNode class are listed here.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

79.1.2.6.1 TXMLNode.AttributeNodes Property

Lists the interfaces for nodes that represent attributes of this node.

Pascal

```
property AttributeNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList AttributeNodes;
```

Description

TXMLNode (see page 7396) uses AttributeNodes to keep track of its attributes. AttributeNodes is an IXMLNodeList (see page 7703) interface, which can be used to access individual attribute nodes or to add attributes to or delete them from this node.

Applications can't use this protected property to access the attributes of a node. Instead, they use the AttributeNodes property of the IXMLNode (see page 7669) interface for this node.

Attributes modify a node in an XML document. For example, given the node

```
<Movie genre=scifi title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

Only element nodes have attributes, and not every element node has them. If the node does not have any attributes, AttributeNodes is the interface for an empty list.

Note: Attributes are not considered child nodes of a node and can't be accessed using the ChildNodes (see page 7439) property.

See Also

IXMLNode (see page 7669)

ChildNodes (see page 7439)

AttributeNodes (see page 7688)

79.1.2.6.2 TXMLNode.ChildNodeClasses Property

Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.

Pascal

```
property ChildNodeClasses: TNodeClassArray;
```

C++

```
__property TNodeClassArray ChildNodeClasses;
```

Description

ChildNodeClasses is used internally when creating child nodes for this node. The XML Data Binding wizard defines TXMLNode (see page 7396) descendants for each node in the XML document that represents a node that has (or can have) children. ChildNodeClasses lists only those TXMLNode (see page 7396) descendants that can represent child nodes of this node. These classes become the types of the corresponding properties on the TXMLNode (see page 7396) descendant that represents this node.

Classes are added to ChildNodeClasses by the RegisterChildNode (see page 7430) method.

See Also

RegisterChildNode (see page 7430)

RegisterChildNodes (see page 7431)

ChildNodes (see page 7439)

CreateChildNode (see page 7410)

79.1.2.6.3 TXMLNode.ChildNodes Property

Lists the child nodes of this node.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

TXMLNode (see page 7396) uses ChildNodes to keep track of the subnodes of a node. ChildNodes is an IXMLNodeList (see page 7703) interface, which can be used to access individual child nodes or to add child nodes to or delete them from this node. Each child node specifies this node as its parent node.

Applications can't use this protected property to access the child nodes of a node. Instead, they use the ChildNodes property of the IXMLNode (see page 7669) interface for this node.

Note: Attributes are not considered child nodes of a node. To get the values of this node's attributes, use the Attributes property instead.

See Also

IXMLNode (see page 7669)

ChildNodes (see page 7689)

ParentNode (see page 7443)

79.1.2.6.4 TXMLNode.Collection Property

Indicates the collection in which this node appears.

Pascal

```
property Collection: TXMLNodeCollection;
```

C++

```
__property TXMLNodeCollection Collection;
```

Description

Collection indicates the collection that contains this node if it represents one of a repeating set of child nodes. The node specified by Collection represents the same node in the XML document as this node's ParentNode (see page 7443) property, but it is specialized to work with this node and any other child nodes of the same type.

The node identified by Collection is one of the collections listed by the parent node's HostedNodes (see page 7441) property, and it lists ParentNode (see page 7443) as the value of its HostNode (see page 7441) property.

For example, consider the following XML fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

If this node represents one of the <student> nodes, then ParentNode (see page 7443) represents the <students> node, and Collection represents a copy of the <students> node that is used for only those children that are <student> nodes, and not the <auditor> nodes.

Applications can't use this protected property to access the Collection that contains this node. Instead, they use the Collection property of the IXMLNode (see page 7669) interface for this node.

See Also

ParentNode (see page 7443)

HostedNodes (see page 7441)

HostNode (see page 7441)

Collection (see page 7690)

79.1.2.6.5 TXMLNode.DOMNode Property

Provides access to the IDOMNode (see page 7557) interface for this node.

Pascal

```
property DOMNode: IDOMNode;
```

C++

```
__property IDOMNode DOMNode;
```

Description

IDOMNode (see page 7557) is the interface for this node provided by the DOM implementation that parsed the XML document. It can be accessed through the DOMNode property of the node's IXMLNode (see page 7669) interface.

Typically, applications that contain TXMLNode (see page 7396) descendants use the IXMLNode (see page 7669) interface to work with a node rather than the IDOMNode (see page 7557) interface. IDOMNode (see page 7557) is used more typically in applications that work directly with the DOM interfaces.

The value of DOMNode is set to the IDOMNode (see page 7557) interface that is provided to the TXMLNode (see page 7396) constructor.

See Also

DOMNode (see page 7691)

Create (see page 7400)

79.1.2.6.6 TXMLNode.HostedNodes Property

Lists the nodes that share this node's lists of child and attribute nodes.

Pascal

```
property HostedNodes: TXMLNodeArray;
```

C++

```
__property TXMLNodeArray HostedNodes;
```

Description

TXMLNode (see page 7396) uses HostedNodes to keep track of the nodes it uses to represent this same node, restricted to a specialized purpose. For example, any node that represents a subset of this node's child nodes that are all of the same type is a hosted node.

Each object in the HostedNodes list is a representation of a particular aspect of this node. Each hosted node acts as a sibling to this TXMLNode (see page 7396) object but does not represent a separate element in the XML document. Rather, it represents a different view of this element.

For example, consider the following XML fragment:

```
<students>
  <fulltime student>Joe Smith</fulltime student>
  <fulltime student>Jane Doe</fulltime student>
  <auditor>Bill Jones</auditor>
</students>
```

The TXMLNode (see page 7396) that represents students would define two hosted nodes for collections of repeating child elements, one for the fulltime students, and one for the auditors. The two collections share the same underlying DOM node as students, but each represents only those child elements of a particular type.

See Also

CreateCollection (see page 7411)

ChildNodes (see page 7439)

TXMLNodeCollection (see page 7444)

Collection (see page 7439)

HostNode (see page 7441)

79.1.2.6.7 TXMLNode.HostNode Property

Indicates the main object for the node to which this node corresponds when it represents a specialized view.

Pascal

```
property HostNode: TXMLNode;
```

C++

```
__property TXMLNode HostNode;
```

Description

If this node represents a hosted node, HostNode is the main node for which this node represents a subset or specialized view. For example, if this node represents a set of repeating child elements, HostNode is the node that uses this node to represent its repeating child elements.

For example, consider the following XML fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
  <auditor name="Susan Brown"></auditor>
</students>
```

The <students> node spawns two hosted nodes to represent the collections of repeating child nodes: one for all of the child nodes of type <student>, and one for all of the child nodes of type <auditor>. If this node represents one of those hosted nodes, then HostNode is the <students> node to which this node belongs. The host node lists this node in its HostedNodes (see page 7441) property.

Applications can't use this protected property to access the host node that created this node. Instead, they use the HostNode property of the IXMLNode (see page 7669) interface for this node.

See Also

Collection (see page 7439)

HostedNodes (see page 7441)

TXMLNodeCollection (see page 7444)

79.1.2.6.8 TXMLNode.OwnerDocument Property

Provides access to the document that contains this node.

Pascal

```
property OwnerDocument: TXMLDocument;
```

C++

```
__property TXMLDocument OwnerDocument;
```

Description

Applications can't use this protected property to access the document that contains this. Instead, they use the OwnerDocument property of the IXMLNode (see page 7669) interface for this node.

The value of OwnerDocument is set to the TXMLDocument (see page 7347) that is provided to the TXMLNode (see page 7396) constructor.

See Also

Create (see page 7400)

OwnerDocument (see page 7695)

79.1.2.6.9 TXMLNode.ParentNode Property

Identifies the object that represents this node's parent node.

Pascal

```
property ParentNode: TXMLNode;
```

C++

```
__property TXMLNode ParentNode;
```

Description

TXMLNode (see page 7396) uses ParentNode to access the implementation of the node's parent, if it exists. Some nodes (for example attribute nodes) do not have parents.

Applications can't use this protected property to access the parent node of a node. Instead, they use the ParentNode property of the IXMLNode (see page 7669) interface for this node.

See Also

ChildNodes (see page 7439)

ParentNode (see page 7695)

79.1.2.7 TXMLNode Events

The events of the TXMLNode class are listed here.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

79.1.2.7.1 TXMLNode.OnHostAttrNotify Event

Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).

Pascal

```
property OnHostAttrNotify: TNodeListNotification;
```

C++

```
__property TNodeListNotification OnHostAttrNotify;
```

Description

Use the OnHostAttrNotify event to update a TXMLNode (see page 7396) descendant that represents a hosted node (such as a collection of repeating child elements) when the host node changes its attribute list.

The AttributeListNotify (see page 7406) method of a host node generates an OnHostAttrNotify event on all of its hosted nodes. This allows the hosted nodes (for example, the nodes that represent a subset of the child nodes that are all the same type) to update their own representation of node attributes so that they reflect the attributes of the host node.

See Also

OnHostChildNotify (see page 7444)

AttributeListNotify (see page 7406)

HostNode (see page 7441)

HostedNodes (see page 7441)

AttributeNodes (see page 7438)

79.1.2.7.2 TXMLNode.OnHostChildNotify Event

Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Pascal

```
property OnHostChildNotify: TNodeListNotification;
```

C++

```
__property TNodeListNotification OnHostChildNotify;
```

Description

Use the OnHostChildNotify event to update a TXMLNode (see page 7396) descendant that represents a hosted node (such as a collection of repeating child elements) when the host node changes its list of child nodes. For example, TXMLNodeCollection (see page 7444) assigns its UpdateCollectionList method as an OnHostChildNotify event handler so that its list of repeating child elements reflects any changes to the host node's list of child nodes.

The ChildListNotify (see page 7408) method of a host node generates an OnHostChildNotify event on all of its hosted nodes.

See Also

OnHostAttrNotify (see page 7443)

ChildListNotify (see page 7408)

HostNode (see page 7441)

HostedNodes (see page 7441)

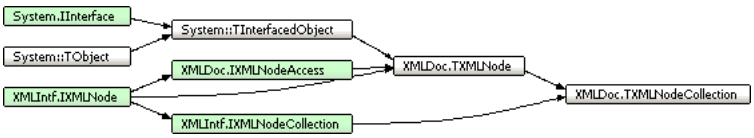
ChildNodes (see page 7439)

UpdateCollectionList (see page 7457)

79.1.3 TXMLNodeCollection Class

TXMLNodeCollection represents a node in an XML document whose children are instances of the same repeating element.

Class Hierarchy



Pascal

```
TXMLNodeCollection = class(TXMLNode, IXMLNodeCollection);
```

C++

```
class TXMLNodeCollection : public TXMLNode, public IXMLNodeCollection;
```

File

XMLDoc

Description

The XML Data Binding Wizard uses TXMLNodeCollection as a base class for the classes it generates to represent nodes that have repeating elements for child nodes. For example, if the XML document includes the following:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

then the XML Data Binding Wizard generates 3 nodes to represent the <students> node: a TXMLNode (see page 7396) descendant to represent the entire <students> node, and two TXMLNodeCollection descendants to represent the repeating elements: one for the child nodes with the tag <student>, and one for the child nodes with the tag <auditor>.

As with any of the classes the Data Binding Wizard generates to represent nodes in an XML document, child nodes and node attributes appear as properties of the TXMLNodeCollection descendant.

TXMLNodeCollection implements the IXMLNodeCollection (see page 7698) interface. Each TXMLNodeCollection descendant implements a class-specific interface that descends from IXMLNodeCollection (see page 7698). Typically, applications do not work directly with TXMLNodeCollection. Instead, they use the IXMLNodeCollection (see page 7698) interface or the IXMLNodeCollection (see page 7698) descendant that is implemented by a TXMLNodeCollection descendant.

See Also

TXMLDocument (see page 7347)

TXMLNode (see page 7396)

IXMLNodeCollection (see page 7698)

79.1.3.1 TXMLNodeCollection Members

The following tables list the members exposed by TXMLNodeCollection.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
	OnHostAttrNotify (see page 7443)	Occurs when changes occur to the attribute list of this node's HostNode (see page 7441).
	OnHostChildNotify (see page 7444)	Occurs when changes occur to the list of child nodes on this node's HostNode (see page 7441).

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_AddRef (see page 7403)	Increments the reference count for this node.

	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	_Release (see page 7404)	Decrements the reference count for this node.
	AddChild (see page 7404)	This is the overview for the AddChild method overload.
	AddHostedNode (see page 7405)	Adds a node to the HostedNodes (see page 7441) property list.
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
	AttributeListNotify (see page 7406)	Responds to notifications of changes in the list of attributes.
	CheckNotHosted (see page 7406)	Raises an EXMLDocError (see page 7639) exception if this is a hosted node.
	CheckReadOnly (see page 7407)	Raises an EXMLDocError (see page 7639) exception if this is a read-only node.
	CheckTextNode (see page 7407)	Generates a DOM text node implementation if it does not yet exist.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	ClearDocumentRef (see page 7408)	Sets the value of OwnerDocument (see page 7442) to nil (Delphi) or NULL (C++).
	CloneNode (see page 7409)	Creates a copy of this node and returns its interface.
	CreateAttributeList (see page 7409)	Creates the object that implements the AttributeNodes (see page 7438) property.
V	CreateAttributeNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for an attribute of this node.
	CreateChildList (see page 7410)	Creates the object that implements the ChildNodes (see page 7439) property.
V	CreateChildNode (see page 7410)	Creates a new TXMLNode (see page 7396) object for a child node of this node.
	CreateCollection (see page 7411)	Creates an instance of a TXMLNodeCollection (see page 7444) descendant and returns it.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	DeclareNamespace (see page 7412)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	DOMElement (see page 7412)	Returns the IDOMElement (see page 7542) interface for this node if it exists.
V	DoNodeChange (see page 7413)	Generates a BeforeNodeChange or AfterNodeChange event on this node's document.
	FindHostedNode (see page 7413)	Returns the IXMLNode (see page 7669) interface for a hosted node given its implementation class.
	FindNamespaceDecl (see page 7414)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7414)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7415)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7415)	Returns the value of the AttributeNodes (see page 7438) property.
	GetAttributeNS (see page 7416)	Returns the value of a specified attribute when you must explicitly specify the namespace for the attribute.
	GetChildNodeClasses (see page 7416)	Returns a list of the classes that can implement child nodes of this node.
	GetChildNodes (see page 7417)	Returns the value of the ChildNodes (see page 7439) property.
	GetChildValue (see page 7417)	Returns the value of a specified child node.
	GetCollection (see page 7418)	Returns the value of the Collection (see page 7439) property.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
	GetDOMNode (see page 7418)	Returns the value of the DOMNode (see page 7440) property.
	GetHasChildNodes (see page 7418)	Indicates whether this node has any child nodes.
	GetHostedNodes (see page 7419)	Returns a list an array of the hosted nodes that share this node's attribute and child node lists.
	GetHostNode (see page 7419)	Returns the value of the HostNode (see page 7441) property.
	GetIsTextElement (see page 7419)	Indicates whether the node has a single text value.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetLocalName (see page 7420)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7421)	Returns the URI for the namespace used in the qualified node name.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7421)	Returns the node name.
	GetNodeObject (see page 7422)	Returns a reference to this TXMLNode (see page 7396) object.
	GetNodeType (see page 7422)	Returns the type of the node.
	GetNodeValue (see page 7423)	Returns the value of this node.

	GetOwnerDocument (see page 7423)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7424)	Returns the interface for the node's parent node.
	GetPrefix (see page 7424)	Returns the namespace prefix of the node's name.
	GetPrefixedName (see page 7425)	Returns a node name includes a namespace prefix for a specified namespace URI.
	GetReadOnly (see page 7425)	Indicates whether the node can be modified.
	GetText (see page 7426)	Returns the text value of the node.
	GetXML (see page 7426)	Returns the XML that corresponds to the subtree rooted at this node.
	HasAttribute (see page 7426)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7427)	This is the overview for the HasChildNode method overload.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	InternalAddChild (see page 7428)	Provides the underlying implementation for the AddChild (see page 7404) method.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	NestingLevel (see page 7428)	Returns the number of parents for this node in the node hierarchy.
	NextSibling (see page 7429)	Returns the next child of this node's parent.
	Normalize (see page 7429)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7430)	Returns the previous child of this node's parent.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	RegisterChildNode (see page 7430)	Registers a descendant of TXMLNode (see page 7396) that can represent the children of this node.
	RegisterChildNodes (see page 7431)	Registers a set of TXMLNode (see page 7396) descendants that can represent the children of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
	RemoveHostedNode (see page 7431)	Removes a node from the HostedNodes (see page 7441) property list.
	Resync (see page 7432)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7432)	Sets the value of one of this node's attributes.
V	SetAttributeNodes (see page 7432)	Sets the value of the AttributeNodes (see page 7438) property.
	SetAttributeNS (see page 7433)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	SetChildValue (see page 7434)	Sets the value of a specified child node.
	SetCollection (see page 7434)	Sets the value of the Collection (see page 7439) property.
	SetNodeValue (see page 7435)	Sets the value of this node.
V	SetParentNode (see page 7435)	Sets the value of the ParentNode (see page 7443) property.
	SetReadOnly (see page 7436)	Specifies whether the node can be modified.
	SetText (see page 7436)	Sets the text value of the node.
	TransformNode (see page 7437)	This is the overview for the TransformNode method overload.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Protected Properties

	Name	Description
R	AttributeNodes (see page 7438)	Lists the interfaces for nodes that represent attributes of this node.
R	ChildNodeClasses (see page 7438)	Lists the classes that the XML Data Binding wizard defines to represent child nodes of this node.
	ChildNodes (see page 7439)	Lists the child nodes of this node.
	Collection (see page 7439)	Indicates the collection in which this node appears.
R	Count (see page 7457)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7440)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HostedNodes (see page 7441)	Lists the nodes that share this node's lists of child and attribute nodes.
	HostNode (see page 7441)	Indicates the main object for the node to which this node corresponds when it represents a specialized view.

	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.
R	OwnerDocument (see page 7442)	Provides access to the document that contains this node.
R	ParentNode (see page 7443)	Identifies the object that represents this node's parent node.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.
V	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 7400)	Instantiates a Create object.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	CreateHosted (see page 7401)	Creates a TXMLNode (see page 7396) instance that represents the same DOM node as another "host" node.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 7401)	Frees the memory associated with the TXMLNode (see page 7396) object.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
V	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.

	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
V	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
V	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
V	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
V	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
V	ToString (see page 3380)	Returns a string containing the class name.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
R	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

79.1.3.2 TXMLNodeCollection Methods

The methods of the TXMLNodeCollection class are listed here.

Protected Methods

	Name	Description
V	AddItem (see page 7451)	Adds a new repetition of the repeating child element to this node.
V	ChildListNotify (see page 7452)	Responds to notifications of changes in the list of child nodes.
	Clear (see page 7452)	Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.
	CreateItemList (see page 7453)	Creates the object that implements the List (see page 7459) property.
	Delete (see page 7453)	Removes one of the repeating child nodes from this node.
	GetCount (see page 7454)	Returns the value of the Count (see page 7457) property.
V	GetList (see page 7454)	Returns the value of the List (see page 7459) property.
	GetNode (see page 7454)	Returns the interface for one of the repeating child nodes.
	InsertInCollection (see page 7455)	Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.
	IsCollectionItem (see page 7455)	Indicates whether a specified node represents the type of node that appears as a repeating child of this node.
	Remove (see page 7456)	Removes one of the repeating child nodes from this node.
V	SetChildNodes (see page 7456)	Sets the value of the ChildNodes (see page 7439) property.
	UpdateCollectionList (see page 7457)	Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Public Methods

	Name	Description
V	AfterConstruction (see page 7451)	Initializes the node collection object once it is fully created.

79.1.3.2.1 TXMLNodeCollection.AddItem Method

Adds a new repetition of the repeating child element to this node.

Pascal

```
function AddItem(Index: Integer): IXMLNode; virtual;
```

C++

```
virtual __fastcall IXMLNode AddItem(int Index);
```

Description

Applications can't call the protected AddItem method. It is used internally to create a new child node for this node that is an instance of the repeating element. The new child node is given the first tag name listed by the ItemTag (see page 7459) property.

Index indicates the position of the new child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

Note: Index refers to the list of repeating elements (the List (see page 7459) property), not the list of all children (the ChildNodes (see page 7439) property).

AddItem returns the interface for the newly created child node.

See Also

AddChild (see page 7404)

InsertInCollection (see page 7455)

ItemTag (see page 7459)

List (see page 7459)

ChildNodes (see page 7439)

79.1.3.2.2 TXMLNodeCollection.AfterConstruction Method

Initializes the node collection object once it is fully created.

Pascal

```
procedure AfterConstruction; override;
```

C++

```
virtual __fastcall AfterConstruction();
```

Description

AfterConstruction is called automatically after the object's last constructor has executed. Do not call it explicitly in your applications.

AfterConstruction sets the UpdateCollectionList (see page 7457) method as the event handler for the collection's OnHostChildNotify (see page 7444) event. This allows the collection to update itself when the list of child nodes (for which this collection manages a subset) changes.

See Also

OnHostChildNotify (see page 7444)

UpdateCollectionList (see page 7457)

79.1.3.2.3 TXMLNodeCollection.ChildListNotify Method

Responds to notifications of changes in the list of child nodes.

Pascal

```
procedure ChildListNotify(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean); override;
```

C++

```
virtual __fastcall ChildListNotify(TNodeListOperation Operation, IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. The class that implements the ChildNodes (see page 7439) property calls ChildListNotify when child nodes are inserted, removed, or need to be created. ChildListNotify calls the inherited method to generate a BeforeNodeChange or AfterNodeChange event and to ensure that the DOM implementation for this node reflects the change. After calling the inherited method, ChildListNotify updates the List (see page 7459) property so that it stays in sync with the ChildNodes (see page 7439) property.

Operation indicates what type of change is about to occur.

Node is the interface for the child node that is added or removed. If Operation is nlCreateNode, Node returns a newly created child node.

IndexOrName is the index of the node that is added or removed when Operation is nlInsert or nlRemove. When Operation is nlCreateNode, IndexOrName is the name or value of the new node to create.

BeforeOperation indicates whether the call occurs before or after the change.

See Also

ChildNodes (see page 7439)

AddChild (see page 7404)

List (see page 7459)

BeforeNodeChange (see page 7394)

AfterNodeChange (see page 7393)

79.1.3.2.4 TXMLNodeCollection.Clear Method

Removes all child nodes in the List (see page 7459) property from the ChildNodes (see page 7439) and List (see page 7459) properties.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Use the IXMLNodeCollection (see page 7698) interface for this node to call the protected Clear method. Clear deletes every child node from the collection (all the nodes in the List (see page 7459) property), along with the corresponding nodes in the ChildNodes (see page 7439) property.

See Also

List (see page 7459)

Delete (see page 7453)

Remove (see page 7456)

ChildNodes (see page 7439)

Clear (see page 7700)

79.1.3.2.5 TXMLNodeCollection.CreateItemList Method

Creates the object that implements the List (see page 7459) property.

Pascal

```
procedure CreateItemList(CheckFirst: Boolean = False);
```

C++

```
__fastcall CreateItemList(Boolean CheckFirst = False);
```

Description

Applications can't call the protected CreateItemList method. It is used internally when TXMLNodeCollection (see page 7444) needs to create a separate object to implement the List (see page 7459) property. A separate implementation object is only needed if this node contains child nodes that are not in the collection. Otherwise, TXMLNodeCollection (see page 7444) can simply use the ChildNodes (see page 7439) property.

CheckFirst specifies whether CreateItemList should check whether the ChildNodes (see page 7439) property currently includes any nodes that are not in the collection. When CheckFirst is true, CreateItemList first checks the ChildNodes (see page 7439) property, and if all nodes are in the collection, it does not create a separate list to store the nodes of the collection. When CheckFirst is false (or when the ChildNodes (see page 7439) property includes nodes that are not in the collection), CreateItemList instantiates a list to hold the nodes in the collection, and adds to it any nodes listed in ChildNodes (see page 7439) that belong in the collection.

See Also

List (see page 7459)

ChildNodes (see page 7439)

79.1.3.2.6 TXMLNodeCollection.Delete Method

Removes one of the repeating child nodes from this node.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Use the TXMLNodeCollection (see page 7444) object's IXMLNodeCollection (see page 7698) interface to call the protected Delete method. Delete removes a specified child node from this node.

Index identifies the child to remove. It is the index of the child in the list of repeating child nodes (The List (see page 7459) property, not the ChildNodes (see page 7439) property). The index 0 represents the first repetition, 1 represents the second

repetition, and so on.

See Also

Remove (see page 7456)

Delete (see page 7700)

List (see page 7459)

ChildNodes (see page 7439)

79.1.3.2.7 TXMLNodeCollection.GetCount Method

Returns the value of the Count (see page 7457) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the protected read implementation of the Count (see page 7457) property. It returns the number of nodes in the List (see page 7459) property.

See Also

Count (see page 7457)

List (see page 7459)

79.1.3.2.8 TXMLNodeCollection.GetList Method

Returns the value of the List (see page 7459) property.

Pascal

```
function GetList: IXMLNodeList; virtual;
```

C++

```
virtual __fastcall IXMLNodeList GetList();
```

Description

GetList is the protected read implementation of the List (see page 7459) property.

See Also

List (see page 7459)

79.1.3.2.9 TXMLNodeCollection.GetNode Method

Returns the interface for one of the repeating child nodes.

Pascal

```
function GetNode(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(int Index);
```

Description

Use the TXMLNodeCollection (see page 7444) object's IXMLNodeCollection (see page 7698) interface to call the protected GetNode method. GetNode returns an interface for working with one of the repeating child nodes.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on. Note that Index is the index of the node in the List (see page 7459) property, not in the index in the ChildNodes (see page 7439) property.

See Also

Nodes (see page 7703)

GetCount (see page 7454)

GetNode (see page 7701)

ChildNodes (see page 7439)

79.1.3.2.10 TXMLNodeCollection.InsertInCollection Method

Copies a child node from the ChildNodes (see page 7439) property to the List (see page 7459) property.

Pascal

```
procedure InsertInCollection(Node: IXMLNode; Index: Integer);
```

C++

```
__fastcall InsertInCollection(IXMLNode Node, int Index);
```

Description

Applications can't call this protected method. It is called automatically when nodes are added to the ChildNodes (see page 7439) property list.

Node is the node that was just added to the ChildNodes (see page 7439) property.

Index is the index of Node in the ChildNodes (see page 7439) property.

InsertInCollection first checks whether Node belongs in the collection. If so, it adds it to the List (see page 7459) property in the position that corresponds to Index (that is, in the index of Node when you ignore all the nodes in ChildNodes (see page 7439) that do not belong in the collection.)

See Also

ChildNodes (see page 7439)

List (see page 7459)

79.1.3.2.11 TXMLNodeCollection.IsCollectionItem Method

Indicates whether a specified node represents the type of node that appears as a repeating child of this node.

Pascal

```
function IsCollectionItem(const Node: IXMLNode): Boolean;
```

C++

```
__fastcall Boolean IsCollectionItem(const IXMLNode Node);
```

Description

Applications can't call the protected IsCollectionItem method. TXMLNodeCollection (see page 7444) uses this method

internally to distinguish nodes that represent instances of the repeating child node type.

IsCollectionItem checks whether Node implements the interface specified by the ItemInterface (see page 7458) property . If so, it returns true. Otherwise, it returns false.

See Also

List (see page 7459)

ItemInterface (see page 7458)

79.1.3.2.12 TXMLNodeCollection.Remove Method

Removes one of the repeating child nodes from this node.

Pascal

```
function Remove(const AItem: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode AItem);
```

Description

Use the TXMLNodeCollection (see page 7444) object's IXMLNodeCollection (see page 7698) interface to call the protected Remove method. Remove removes a specified child node from this node.

AItem is the interface of the child to remove. It must represent on of the repeating child nodes.

Remove deletes the child node from both the List (see page 7459) property and from the ChildNodes (see page 7439) property. It returns the index that the child node held in the List (see page 7459) property.

To remove a child that is not one of the repeating child nodes, use the ChildNodes (see page 7439) property instead.

See Also

Delete (see page 7453)

List (see page 7459)

ChildNodes (see page 7439)

79.1.3.2.13 TXMLNodeCollection.SetChildNodes Method

Sets the value of the ChildNodes (see page 7439) property.

Pascal

```
procedure SetChildNodes(const Value: IXMLNodeList); override;
```

C++

```
virtual __fastcall SetChildNodes(const IXMLNodeList Value);
```

Description

Applications can't call this protected method. It is called internally the first time an application reads the ChildNodes (see page 7439) property to supply the interface that implements that property.

TXMLNodeCollection (see page 7444) overrides the inherited method to update the List (see page 7459) property as well.

See Also

ChildNodes (see page 7439)

List (see page 7459)

79.1.3.2.14 TXMLNodeCollection.UpdateCollectionList Method

Updates the List (see page 7459) property to reflect changes to the set of child nodes.

Pascal

```
procedure UpdateCollectionList(Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean);
```

C++

```
__fastcall UpdateCollectionList(TNodeListOperation Operation, IXMLNode Node, const OleVariant
IndexOrName, Boolean BeforeOperation);
```

Description

Applications can't call this protected method. This method is called automatically when the ChildNodes (see page 7439) property changes so that the collection can update the List (see page 7459) property to reflect the repeating elements in the current set of child nodes.

Operation indicates whether the change to ChildNodes (see page 7439) is the addition of a new node (nlInsert) or the deletion of a child node (nlRemove).

Node is the interface for a child node that is added or removed.

IndexOrName is the index or tag name of the node that is added or removed.

BeforeOperation indicates whether this method is called before ChildNodes (see page 7439) has changed or after the change has occurred.

See Also

List (see page 7459)

ChildListNotify (see page 7452)

ChildNodes (see page 7439)

OnHostChildNotify (see page 7444)

79.1.3.3 TXMLNodeCollection Properties

The properties of the TXMLNodeCollection class are listed here.

Protected Properties

	Name	Description
R	Count (see page 7457)	Indicates the number of repeating child nodes.
	ItemInterface (see page 7458)	Indicates the interface type for the repeating child nodes.
	ItemNS (see page 7458)	Indicates the default namespace URI for the tag names of repeating child elements.
	ItemTag (see page 7459)	Indicates the tag name that occurs on the repeating child elements.
R	List (see page 7459)	Lists the interfaces for the children that are repeated elements.

79.1.3.3.1 TXMLNodeCollection.Count Property

Indicates the number of repeating child nodes.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Use the IXMLNodeCollection (see page 7698) interface to read this protected property. Count is the number of child nodes for this node that are repetitions of the repeating child node type. That is, Count is the number of nodes in the List (see page 7459) property.

Use Count to obtain an upper limit when addressing child nodes by index (for example, when iterating over all nodes in using the GetNode (see page 7454) method).

Count is changed when you add child nodes using the AddChild (see page 7404) method or delete them using the Delete (see page 7453) or Remove (see page 7456) method. You can access the individual repeating child nodes using the GetNode (see page 7454) method.

See Also

GetNode (see page 7454)

AddChild (see page 7672)

Remove (see page 7456)

Delete (see page 7453)

List (see page 7459)

79.1.3.3.2 TXMLNodeCollection.ItemInterface Property

Indicates the interface type for the repeating child nodes.

Pascal

```
property ItemInterface: TGuid;
```

C++

```
__property TGuid ItemInterface;
```

Description

Applications can't use this protected property. It is used internally to identify the interface for all child nodes that represent the repeating child element.

ItemInterface is the GUID for the default interface of the class that implements the nodes in the collection.

See Also

IXMLNode (see page 7669)

IsCollectionItem (see page 7455)

List (see page 7459)

ItemTag (see page 7459)

ItemNS (see page 7458)

79.1.3.3.3 TXMLNodeCollection.ItemNS Property

Indicates the default namespace URI for the tag names of repeating child elements.

Pascal

```
property ItemNS: DOMString;
```

C++

```
__property DOMString ItemNS;
```

Description

Applications can't use this protected property. It is used internally to identify the namespace used to interpret tag names on members of this node collection.

See Also

NamespaceURI (see page 7693)

ItemInterface (see page 7458)

List (see page 7459)

ItemTag (see page 7459)

79.1.3.3.4 TXMLNodeCollection.ItemTag Property

Indicates the tag name that occurs on the repeating child elements.

Pascal

```
property ItemTag: DOMString;
```

C++

```
__property DOMString ItemTag;
```

Description

Applications can't use this protected property. It is used internally to identify the tag name that appears on members of this node collection.

ItemTag is interpreted in the context of the namespace specified by the ItemNS (see page 7458) property.

See Also

NodeName (see page 7693)

ItemInterface (see page 7458)

List (see page 7459)

ItemNS (see page 7458)

79.1.3.3.5 TXMLNodeCollection.List Property

Lists the interfaces for the children that are repeated elements.

Pascal

```
property List: IXMLNodeList;
```

C++

```
__property IXMLNodeList List;
```

Description

Applications can't use the protected List property. TXMLNodeCollection (see page 7444) uses this property internally to identify those child nodes that are repeated elements. Because TXMLNodeCollection (see page 7444) can correspond to a

node in the XML document that has additional children that are not instances of the repeating node type, List is kept separate from the ChildNodes (see page 7439) property, which lists all child nodes.

Applications that need to access the child nodes in this list should use the Nodes property of the IXMLNodeCollection (see page 7698) interface for this node or the Items property of an IXMLNodeCollection (see page 7698) descendant that is generated by the XML Data Binding wizard.

See Also

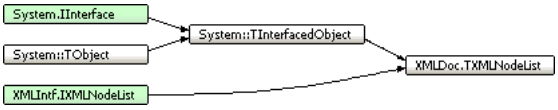
Nodes (see page 7703)

ChildNodes (see page 7439)

79.1.4 TXMLNodeList Class

TXMLNodeList represents a set of nodes in an XML document.

Class Hierarchy



Pascal

```
TXMLNodeList = class(TInterfacedObject, IXMLNodeList);
```

C++

```
class TXMLNodeList : public TInterfacedObject, public IXMLNodeList;
```

File

XMLDoc

Description

TXMLNodeList is used to represent a set of related nodes (IXMLNode (see page 7669) interfaces) in an XML document. For example, TXMLNodeList is used to represent all of the children of a node, or all of the attributes of a node, or all of the nodes in a collection of repeating nodes.

TXMLNodeList can be used to add or delete nodes from the collection, or to access specific nodes.

Typically, applications do not use TXMLNodeList directly. Instead, they access it through the IXMLNodeList (see page 7703) interface.

See Also

ChildNodes (see page 7439)

AttributeNodes (see page 7438)

TXMLNodeCollection (see page 7444)

IXMLNodeList (see page 7703)

IXMLNode (see page 7669)

TObjectList

TInterfaceList

79.1.4.1 TXMLNodeList Members

The following tables list the members exposed by TXMLNodeList.

Protected Data Members

	Name	Description
	FRefCount (see page 3344)	This is FRefCount, a member of class TInterfacedObject.

Protected Events

	Name	Description
R	NotificationProc (see page 7476)	Specifies a callback that TXMLNodeList (see page 7460) calls when nodes are added to or removed from the list, or when new nodes need to be created

Protected Methods

	Name	Description
	_AddRef (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_AddRef method.
	_Release (see page 3345)	Implements the IInterface (see page 3502) System::TInterfacedObject::_Release method.
	Add (see page 7464)	Adds a new node to the end of the list.
	BeginUpdate (see page 7465)	Increments the value of UpdateCount (see page 7476).
	Clear (see page 7465)	Removes all nodes from the list.
	Delete (see page 7465)	This is the overview for the Delete method overload.
	DoNotify (see page 7466)	Executes the callback specified by NotificationProc (see page 7476).
	EndUpdate (see page 7467)	Decrements the value of UpdateCount (see page 7476).
	FindNode (see page 7467)	This is the overview for the FindNode method overload.
	FindSibling (see page 7468)	Returns a node that appears a specified amount before or after another node.
	First (see page 7469)	Returns the first node in the list.
	Get (see page 7469)	Returns a specified node in the list.
	GetCount (see page 7470)	Returns the number of nodes in the list.
	GetNode (see page 7470)	Returns a specified node from the list.
	GetUpdateCount (see page 7471)	Returns the value of the UpdateCount (see page 7476) property.
	IndexOf (see page 7471)	This is the overview for the IndexOf method overload.
	Insert (see page 7472)	Inserts a new node into a specified position in the list.
	InternalInsert (see page 7472)	Provides the underlying implementation of the Insert (see page 7472) method
	Last (see page 7473)	Returns the last node in the list.
	QueryInterface (see page 3347)	Implements the IInterface (see page 3502) System::TInterfacedObject::QueryInterface method.
	Remove (see page 7473)	Removes a specified node from the list.
	ReplaceNode (see page 7473)	Replaces a node in the list with another node.

Protected Properties

	Name	Description
R	Count (see page 7474)	Indicates the number of nodes in the collection.
R	DefaultNamespaceURI (see page 7475)	Identifies the default Namespace for the nodes in this collection.
R	List (see page 7475)	Stores the interfaces for every node in the list.
R	Owner (see page 7475)	Specifies the node that uses this TXMLNodeList (see page 7460) object to manage a set of its child or attribute nodes.
R	UpdateCount (see page 7476)	Indicates the number of times an application has called BeginUpdate (see page 7465) without a matching call to EndUpdate (see page 7467).

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.

	AfterConstruction (see page 3345)	Responds after the last constructor executes.
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3346)	Checks that the reference count is zero before allowing the interfaced object to be destroyed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7462)	Instantiates a TXMLNodeList (see page 7460) object.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Delete (see page 7706)	This is the overview for the Delete method overload.
	Destroy (see page 7463)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	FindNode (see page 7708)	This is the overview for the FindNode method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	IndexOf (see page 7711)	This is the overview for the IndexOf method overload.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3346)	Allocates memory for each instance of an interfaced object.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ToString (see page 3380)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	Nodes (see page 7714)	Provides access to the nodes in the list.
	RefCount (see page 3347)	Indicates the number of interface pointers currently dependent upon the object.

79.1.4.2 TXMLNodeList.Create Constructor

Instantiates a TXMLNodeList (see page 7460) object.

Pascal

```
constructor Create(Owner: TXMLNode; const DefaultNamespaceURI: DOMString; NotificationProc:
```

```
TNodeListNotification);
```

C++

```
__fastcall TXMLNodeList(TXMLNode Owner, const DOMString DefaultNamespaceURI,
TNodeListNotification NotificationProc);
```

Description

Createinstantiates a new TXMLNodeList (see page 7460) object. Typically, TXMLNodeList (see page 7460) is created by another object that represents a node. The TXMLNodeList (see page 7460) represents a set of nodes that are of interest to the other object that creates it.

AOwner is the node that creates TXMLNodeList (see page 7460) to represent a set of its child nodes or attributes.

DefaultNamespaceURI (see page 7475) identifies the namespace that is used by default for nodes added to this list when no namespace is explicitly specified.

NotificationProc (see page 7476) is a callback that TXMLNodeList (see page 7460) calls when nodes are added to or removed from the list, or when new nodes need to be created.

See Also

DefaultNamespaceURI (see page 7475)

Owner (see page 7475)

NotificationProc (see page 7476)

79.1.4.3 TXMLNodeList.Destroy Destructor

Disposes of an object instance.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLNodeList();
```

Description

Do not call System::TObject::Destroy directly. Call System::TObject::Free instead. Free verifies that the object reference is not nil before calling System::TObject::Destroy.

The System::TObject::Destroy method defined by System::TObject deallocates memory. Descendant objects usually define a destructor that is customized for that particular kind of object.

When declaring a System::TObject::Destroy method in a descendant, always add the override directive to the declaration and call the inherited System::TObject::Destroy as the last statement in the overriding method. Since System::TObject::Destroy is a virtual method, overriding it ensures that the proper inherited behavior occurs.

Note: If an exception escapes from the constructor, the destructor is called to destroy the partially constructed object instance that failed to initialize completely. Therefore, destructors should check that allocated resources, such as handles, were actually allocated before trying to release them, since their value might be zero.

System::TObject::Destroy should be implemented so that it calls System::TObject::Free on all subobjects created within the object's constructor (that is, allocated by the constructor). Unlike System::TObject::Destroy, System::TObject::Free provides a safeguard when destroying objects that are nil.

See Also

System::TObject::Free (see page 3372)

79.1.4.4 TXMLNodeList Methods

The methods of the TXMLNodeList class are listed here.

Protected Methods

	Name	Description
	Add (see page 7464)	Adds a new node to the end of the list.
	BeginUpdate (see page 7465)	Increments the value of UpdateCount (see page 7476).
	Clear (see page 7465)	Removes all nodes from the list.
	Delete (see page 7465)	This is the overview for the Delete method overload.
	DoNotify (see page 7466)	Executes the callback specified by NotificationProc (see page 7476).
	EndUpdate (see page 7467)	Decrements the value of UpdateCount (see page 7476).
	FindNode (see page 7467)	This is the overview for the FindNode method overload.
	FindSibling (see page 7468)	Returns a node that appears a specified amount before or after another node.
	First (see page 7469)	Returns the first node in the list.
	Get (see page 7469)	Returns a specified node in the list.
	GetCount (see page 7470)	Returns the number of nodes in the list.
	GetNode (see page 7470)	Returns a specified node from the list.
	GetUpdateCount (see page 7471)	Returns the value of the UpdateCount (see page 7476) property.
	IndexOf (see page 7471)	This is the overview for the IndexOf method overload.
	Insert (see page 7472)	Inserts a new node into a specified position in the list.
	InternalInsert (see page 7472)	Provides the underlying implementation of the Insert (see page 7472) method
	Last (see page 7473)	Returns the last node in the list.
	Remove (see page 7473)	Removes a specified node from the list.
	ReplaceNode (see page 7473)	Replaces a node in the list with another node.

79.1.4.4.1 TXMLNodeList.Add Method

Adds a new node to the end of the list.

Pascal

```
function Add(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Add(const IXMLNode Node);
```

Description

Call Add to add a node to the end of the list. Add returns the index of the node once it is added, where 0 is the index of the first node in the list, 1 is the index of the second node, and so on.

Node is the node to add to the list.

See Also

Delete (see page 7466)

Remove (see page 7473)

IndexOf (see page 7471)

Insert (see page 7472)

List (see page 7475)

79.1.4.4.2 TXMLNodeList.BeginUpdate Method

Increments the value of UpdateCount (see page 7476).

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before making several changes to the nodes in the list (such as adding or removing several nodes). BeginUpdate increments the UpdateCount (see page 7476) property, to keep track of the fact that the list is in the middle of an update. When the update is complete, call EndUpdate (see page 7467) to decrement the UpdateCount (see page 7476) property.

Every call to BeginUpdate should be matched by a call to EndUpdate (see page 7467), so that the UpdateCount (see page 7476) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7476)

EndUpdate (see page 7467)

79.1.4.4.3 TXMLNodeList.Clear Method

Removes all nodes from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the list.

Note: Clear does not call the BeginUpdate (see page 7465) and EndUpdate (see page 7467) methods. NotificationProc (see page 7476) is called for every node in the list as it is removed.

See Also

Delete (see page 7466)

Remove (see page 7473)

List (see page 7475)

NotificationProc (see page 7476)

79.1.4.4.4 Delete Method

This is the overview for the Delete method overload.

Overload List

	Name	Description
	TXMLNodeList.Delete (DOMString) (see page 7466)	Removes a specified node from the list.
	TXMLNodeList.Delete (DOMString, DOMString) (see page 7466)	Removes a specified node from the list.
	TXMLNodeList.Delete (Integer) (see page 7466)	Removes a specified node from the list.

79.1.4.4.4.1 TXMLNodeList.Delete Method (DOMString)

Removes a specified node from the list.

Pascal

```
function Delete(const Index: Integer): Integer; overload;
function Delete(const Name: DOMString): Integer; overload;
function Delete(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int Delete(const int Index);
__fastcall int Delete(const DOMString Name);
__fastcall int Delete(const DOMString Name, const DOMString NamespaceURI);
```

Description

Delete removes the node specified by the Index or Name parameter.

Name identifies the node to remove from the list. This is the local name of the node to remove.

NamespaceURI identifies the namespace to use when interpreting Name. If the node is identified by Name, but no NamespaceURI parameter is provided, Delete uses DefaultNamespaceURI (see page 7475).

Index identifies the node to remove by index rather than name. Index ranges from 0 to one less than the value of the Count (see page 7474) property.

Delete returns the index of the node that was removed. If there was no node that matched the value of Index or Name, Delete returns –1.

See Also

Remove (see page 7473)

Clear (see page 7465)

IndexOf (see page 7471)

LocalName (see page 7692)

DefaultNamespaceURI (see page 7475)

NamespaceURI (see page 7693)

79.1.4.4.5 TXMLNodeList.DoNotify Method

Executes the callback specified by NotificationProc (see page 7476).

Pascal

```
function DoNotify(Operation: TNodeListOperation; const Node: IXMLNode; const IndexOrName:
```

```
OleVariant; BeforeOperation: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode DoNotify(TNodeListOperation Operation, const IXMLNode Node, const
OleVariant IndexOrName, Boolean BeforeOperation);
```

Description

DoNotify executes the callback that is assigned to NotificationProc (see page 7476) when the list adds, deletes, or needs to create nodes in the list. The parameters are supplied as parameters to the callback, and DoNotify returns the node that was added, deleted, or created.

Operation indicates what change is about to occur or has just occurred.

Node is the node that is being added or removed from the list, or that acts as a parent if the callback must create a node (when Operation is nlCreateNode).

IndexOrName is the name of the node to be created when Operation is nlCreateNode. Otherwise, it is the name or index of the node being deleted or added.

BeforeOperation indicates whether the call occurs before the indicated operation or after. When nodes are inserted and removed, a call is made first with BeforeOperation set to true, then the operation occurs, and then a second callback is made with BeforeOperation set to false. When nodes must be created, only one call is made, with BeforeOperation set to true.

See Also

NotificationProc (see page 7476)

79.1.4.4.6 TXMLNodeList.EndUpdate Method

Decrements the value of UpdateCount (see page 7476).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after making several changes to the nodes in the list (such as adding or removing several nodes). EndUpdate decrements the UpdateCount (see page 7476) property, which was incremented by a previous call to BeginUpdate (see page 7465).

Every call to BeginUpdate (see page 7465) should be matched by a call to EndUpdate, so that the UpdateCount (see page 7476) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7476)

BeginUpdate (see page 7465)

79.1.4.4.7 FindNode Method

This is the overview for the FindNode method overload.

Overload List

	Name	Description
	TXMLNodeList.FindNode (DOMString) (see page 7468)	Returns a specified node from the list.
	TXMLNodeList.FindNode (DOMString, DOMString) (see page 7468)	Returns a specified node from the list.
	TXMLNodeList.FindNode (TGuid) (see page 7468)	Returns a specified node from the list.

79.1.4.4.7.1 TXMLNodeList.FindNode Method (DOMString)

Returns a specified node from the list.

Pascal

```
function FindNode(NodeName: DOMString): IXMLNode; overload;
function FindNode(NodeName: DOMString; NamespaceURI: DOMString): IXMLNode; overload;
function FindNode(ChildNodeType: TGuid): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode FindNode(DOMString NodeName);
__fastcall IXMLNode FindNode(DOMString NodeName, DOMString NamespaceURI);
__fastcall IXMLNode FindNode(TGuid ChildNodeType);
```

Description

Call FindNode to access a particular node in the list.

NodeName is the node to access. It specifies the LocalName property of the desired node.

NamespaceURI identifies the namespace that qualifies NodeName. It is the same as the NamespaceURI property of the desired node. If the first syntax is used (omitting NamespaceURI), FindNode uses the value of the DefaultNamespaceURI (see page 7475) property.

ChildNodeType identifies the node to access by its interface type (GUID), rather than by name. When using ChildNodeType, FindNode returns the first child node it finds that implements the specified interface.

FindNode returns the interface for the node if it is in the list. If NodeName or ChildNodeType does not specify a node in the list, FindNode returns nil (Delphi) or NULL (C++).

See Also

FindSibling (see page 7468)

First (see page 7469)

Last (see page 7473)

IndexOf (see page 7471)

LocalName (see page 7692)

NamespaceURI (see page 7693)

DefaultNamespaceURI (see page 7475)

79.1.4.4.8 TXMLNodeList.FindSibling Method

Returns a node that appears a specified amount before or after another node.

Pascal

```
function FindSibling(const Node: IXMLNode; Delta: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode FindSibling(const IXMLNode Node, int Delta);
```

Description

Call FindSibling to access the node whose position has a specified relationship to another node.

Node is a node in the list to use as a reference point.

Delta indicates where the desired node appears, relative to Node. If Delta is positive, FindSibling returns the node that appears Delta positions after Node. If Delta is negative, FindSibling returns a node that appears before Node.

FindSibling returns the node that appears at the position offset by Delta, relative to the position of Node. If Delta specifies a position before the first node or after the last node in the list, FindSibling returns nil (Delphi) or NULL (C++).

See Also

FindNode (see page 7468)

First (see page 7469)

Last (see page 7473)

IndexOf (see page 7471)

79.1.4.4.9 TXMLNodeList.First Method

Returns the first node in the list.

Pascal

```
function First: IXMLNode;
```

C++

```
__fastcall IXMLNode First();
```

Description

Call First to access the first node in the list. If the list is empty, First raises an exception.

See Also

FindNode (see page 7468)

FindSibling (see page 7468)

Last (see page 7473)

79.1.4.4.10 TXMLNodeList.Get Method

Returns a specified node in the list.

Pascal

```
function Get(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode Get(int Index);
```

Description

Call Get to retrieve a node from the list, given its index.

Index specifies the node to fetch, where 0 identifies the first node, 1 identifies the second node, and so on. Index should be less than the value of the Count (see page 7474) property.

See Also

List (see page 7475)

GetNode (see page 7470)

Count (see page 7474)

79.1.4.4.11 TXMLNodeList.GetCount Method

Returns the number of nodes in the list.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 7474) property.

See Also

Count (see page 7474)

79.1.4.4.12 TXMLNodeList.GetNode Method

Returns a specified node from the list.

Pascal

```
function GetNode(const IndexOrName: OleVariant): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(const OleVariant IndexOrName);
```

Description

GetNode returns the interface to a node that is identified by name or by position.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, GetNode tries to create a new node with the name specified by IndexOrName. If it can't create the new node, GetNode raises an exception.

See Also

FindNode (see page 7468)

List (see page 7475)

LocalName (see page 7692)

Get (see page 7469)

79.1.4.4.13 TXMLNodeList.GetUpdateCount Method

Returns the value of the UpdateCount (see page 7476) property.

Pascal

```
function GetUpdateCount: Integer;
```

C++

```
__fastcall int GetUpdateCount();
```

Description

GetUpdateCount is the read implementation of the UpdateCount (see page 7476) property.

See Also

UpdateCount (see page 7476)

79.1.4.4.14 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	TXMLNodeList.IndexOf (DOMString) (see page 7471)	Returns the index of a specified node.
	TXMLNodeList.IndexOf (DOMString, DOMString) (see page 7471)	Returns the index of a specified node.
	TXMLNodeList.IndexOf (IXMLNode) (see page 7471)	Returns the index of a specified node.

79.1.4.4.14.1 TXMLNodeList.IndexOf Method (DOMString)

Returns the index of a specified node.

Pascal

```
function IndexOf(const Node: IXMLNode): Integer; overload;
function IndexOf(const Name: DOMString): Integer; overload;
function IndexOf(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int IndexOf(const IXMLNode Node);
__fastcall int IndexOf(const DOMString Name);
__fastcall int IndexOf(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call IndexOf to locate a node in the list. You can specify which node to locate using its name or its interface.

Name is the LocalName property of the node to locate.

NamespaceURI identifies the namespace to use when interpreting LocalName. If Name is used without a NamespaceURI parameter (the first syntax), IndexOf uses the value of the DefaultNamespaceURI (see page 7475) property.

Node is the interface of the node to locate.

IndexOf returns the index of the specified node, where 0 is the index of the first node, 1 is the index of the second node, and so on. If the specified node is not in the list, IndexOf returns -1.

See Also

FindNode (see page 7468)

LocalName (see page 7692)

NamespaceURI (see page 7693)

DefaultNamespaceURI (see page 7475)

79.1.4.4.15 TXMLNodeList.Insert Method

Inserts a new node into a specified position in the list.

Pascal

```
procedure Insert(Index: Integer; const Node: IXMLNode);
```

C++

```
__fastcall Insert(int Index, const IXMLNode Node);
```

Description

Call Insert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index does not specify a valid index, Insert raises an exception.

Node is the node to add to the list.

See Also

Delete (see page 7466)

Remove (see page 7473)

IndexOf (see page 7471)

Add (see page 7464)

79.1.4.4.16 TXMLNodeList.InternalInsert Method

Provides the underlying implementation of the Insert (see page 7472) method

Pascal

```
function InternalInsert(Index: Integer; const Node: IXMLNode): Integer;
```

C++

```
__fastcall int InternalInsert(int Index, const IXMLNode Node);
```

Description

The Insert (see page 7472) method calls InternalInsert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index is –1, the node is added to the end of the list.

Node is the node to add to the list.

InternalInsert returns the index of the node after it is inserted. This is the same as the Index parameter when Index is not –1.

See Also

Insert (see page 7472)

79.1.4.4.17 TXMLNodeList.Last Method

Returns the last node in the list.

Pascal

```
function Last: IXMLNode;
```

C++

```
__fastcall IXMLNode Last();
```

Description

Call Last to access the last node in the list. If the list is empty, Last raises an exception.

See Also

FindNode (see page 7468)

FindSibling (see page 7468)

First (see page 7469)

79.1.4.4.18 TXMLNodeList.Remove Method

Removes a specified node from the list.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes the specified node from the list.

Node is the node to remove from the list.

Remove returns the index of Node before it was removed. If node is not a node in the list, Remove returns -1.

See Also

Delete (see page 7466)

Clear (see page 7465)

79.1.4.4.19 TXMLNodeList.ReplaceNode Method

Replaces a node in the list with another node.

Pascal

```
function ReplaceNode(const OldNode: IXMLNode; const NewNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode ReplaceNode(const IXMLNode OldNode, const IXMLNode NewNode);
```

Description

Call ReplaceNode to replace the node specified by OldNode with the node specified by NewNode.

OldNode is the node to replace. If OldNode does not appear in the list, then ReplaceNode adds the new node to the end of the list.

NewNode is the node to add to the list in place of OldNode.

ReplaceNode returns OldNode (even if OldNode did not appear in the list).

See Also

List (see page 7475)

79.1.4.5 TXMLNodeList Properties

The properties of the TXMLNodeList class are listed here.

Protected Properties

	Name	Description
	Count (see page 7474)	Indicates the number of nodes in the collection.
	DefaultNamespaceURI (see page 7475)	Identifies the default Namespace for the nodes in this collection.
	List (see page 7475)	Stores the interfaces for every node in the list.
	Owner (see page 7475)	Specifies the node that uses this TXMLNodeList (see page 7460) object to manage a set of its child or attribute nodes.
	UpdateCount (see page 7476)	Indicates the number of times an application has called BeginUpdate (see page 7465) without a matching call to EndUpdate (see page 7467).

79.1.4.5.1 TXMLNodeList.Count Property

Indicates the number of nodes in the collection.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3634) Count to determine the number of nodes in the list. To change the value of Count, use the IXMLNodeList (see page 7703) interface. Add (see page 7464) nodes using the Add (see page 7464) or Insert (see page 7472) method, or remove them using the Delete (see page 7466) or Remove (see page 7473) method.

See Also

Nodes (see page 7714)

Add (see page 7705)

Delete (see page 7707)

Remove (see page 7713)

Insert (see page 7712)

79.1.4.5.2 TXMLNodeList.DefaultNamespaceURI Property

Identifies the default Namespace for the nodes in this collection.

Pascal

```
property DefaultNamespaceURI: DOMString;
```

C++

```
__property DOMString DefaultNamespaceURI;
```

Description

Read (see page 3634) DefaultNamespaceURI to determine the namespace that is used by default when locating nodes in the list. The IndexOf (see page 7471), FindNode (see page 7468), and Delete (see page 7466) methods all use this namespace when trying to match the name of a node if a namespace is not explicitly supplied.

The value of DefaultNamespaceURI is set by the TXMLNodeList (see page 7460) constructor.

See Also

NamespaceURI (see page 7693)

Create (see page 7462)

IndexOf (see page 7471)

FindNode (see page 7468)

Delete (see page 7466)

79.1.4.5.3 TXMLNodeList.List Property

Stores the interfaces for every node in the list.

Pascal

```
property List: IInterfaceList;
```

C++

```
__property IInterfaceList List;
```

Description

TXMLNodeList (see page 7460) uses List internally to store and manage the interfaces to the nodes it represents.

See Also

Count (see page 7474)

Nodes (see page 7714)

79.1.4.5.4 TXMLNodeList.Owner Property

Specifies the node that uses this TXMLNodeList (see page 7460) object to manage a set of its child or attribute nodes.

Pascal

```
property Owner: TXMLNode;
```

C++

```
__property TXMLNode Owner;
```

Description

Owner designates the node that instantiates TXMLNodeList (see page 7460) to manage a set of its child or attribute nodes. This value is supplied to the TXMLNodeList (see page 7460) constructor.

See Also

ChildNodes (see page 7439)

Create (see page 7462)

AttributeNodes (see page 7438)

79.1.4.5.5 TXMLNodeList.UpdateCount Property

Indicates the number of times an application has called BeginUpdate (see page 7465) without a matching call to EndUpdate (see page 7467).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether all update operations have been completed. When an application starts making changes to the list, it calls BeginUpdate (see page 7465), which increments UpdateCount. When those changes are complete, it calls EndUpdate (see page 7467), which decrements UpdateCount. Thus, UpdateCount allows calls to BeginUpdate (see page 7465) and EndUpdate (see page 7467) to be nested.

Applications can avoid responding to updates in a piecemeal fashion by checking UpdateCount and only responding when it is 0. For example, the callback that is supplied to the TXMLNodeList (see page 7460) constructor typically checks UpdateCount before responding to change notifications.

See Also

BeginUpdate (see page 7465)

EndUpdate (see page 7467)

Create (see page 7462)

79.1.4.6 TXMLNodeList Events

The events of the TXMLNodeList class are listed here.

Protected Events

	Name	Description
	NotificationProc (see page 7476)	Specifies a callback that TXMLNodeList (see page 7460) calls when nodes are added to or removed from the list, or when new nodes need to be created

79.1.4.6.1 TXMLNodeList.NotificationProc Event

Specifies a callback that TXMLNodeList (see page 7460) calls when nodes are added to or removed from the list, or when new nodes need to be created

Pascal

```
property NotificationProc: TNodeListNotification;
```

C++

```
__property TNodeListNotification NotificationProc;
```

Description

The node that instantiates TXMLNodeList (see page 7460) to manage a set of its child or attribute nodes supplies the value of this callback to the TXMLNodeList (see page 7460) constructor. TXMLNodeList (see page 7460) then calls this callback when its methods are used to add, create, or remove nodes.

See Also

ChildListNotify (see page 7408)

Create (see page 7462)

AttributeListNotify (see page 7406)

79.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IXMLDocumentAccess (see page 7478)	This is class XMLDoc.IXMLDocumentAccess.
	IXMLNodeAccess (see page 7480)	This is class XMLDoc.IXMLNodeAccess.

79.2.1 IXMLDocumentAccess Interface

Class Hierarchy

XMLDoc.IXMLDocumentAccess

Pascal

```
IXMLDocumentAccess = interface;
```

C++

```
__interface IXMLDocumentAccess;
```

File

XMLDoc

Description

This is class XMLDoc.IXMLDocumentAccess.

79.2.1.1 IXMLDocumentAccess Members

The following tables list the members exposed by IXMLDocumentAccess.

Public Methods

	Name	Description
	GetDocumentObject (see page 7479)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMPersist (see page 7479)	This is GetDOMPersist, a member of class IXMLDocumentAccess.

Public Properties

	Name	Description
R	DocumentObject (see page 7479)	This is DocumentObject, a member of class IXMLDocumentAccess.
R	DOMPersist (see page 7480)	This is DOMPersist, a member of class IXMLDocumentAccess.

79.2.1.2 IXMLDocumentAccess Methods

The methods of the IXMLDocumentAccess class are listed here.

Public Methods

	Name	Description
	GetDocumentObject (see page 7479)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMPersist (see page 7479)	This is GetDOMPersist, a member of class IXMLDocumentAccess.

79.2.1.2.1 IXMLDocumentAccess.GetDocumentObject Method

Pascal

```
function GetDocumentObject: TXMLDocument;
```

C++

```
__fastcall TXMLDocument GetDocumentObject();
```

Description

This is GetDocumentObject, a member of class IXMLDocumentAccess.

79.2.1.2.2 IXMLDocumentAccess.GetDOMPersist Method

Pascal

```
function GetDOMPersist: IDOMPersist;
```

C++

```
__fastcall IDOMPersist GetDOMPersist();
```

Description

This is GetDOMPersist, a member of class IXMLDocumentAccess.

79.2.1.3 IXMLDocumentAccess Properties

The properties of the IXMLDocumentAccess class are listed here.

Public Properties

	Name	Description
R	DocumentObject (see page 7479)	This is DocumentObject, a member of class IXMLDocumentAccess.
R	DOMPersist (see page 7480)	This is DOMPersist, a member of class IXMLDocumentAccess.

79.2.1.3.1 IXMLDocumentAccess.DocumentObject Property

Pascal

```
property DocumentObject: TXMLDocument;
```

C++

```
__property TXMLDocument DocumentObject;
```

Description

This is DocumentObject, a member of class IXMLDocumentAccess.

79.2.1.3.2 IXMLDocumentAccess.DOMPersist Property

Pascal

```
property DOMPersist: IDOMPersist;
```

C++

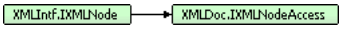
```
__property IDOMPersist DOMPersist;
```

Description

This is DOMPersist, a member of class IXMLDocumentAccess.

79.2.2 IXMLNodeAccess Interface

Class Hierarchy



Pascal

```
IXMLNodeAccess = interface(IXMLNode);
```

C++

```
__interface IXMLNodeAccess : public IXMLNode;
```

File

XMLDoc

Description

This is class XMLDoc.IXMLNodeAccess.

79.2.2.1 IXMLNodeAccess Members

The following tables list the members exposed by IXMLNodeAccess.

Public Methods

	Name	Description
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.

	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.

R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

79.2.2.2 IXMLNodeAccess Methods

The methods of the IXMLNodeAccess class are listed here.

Public Methods

	Name	Description
	AddChild (see page 7482)	This is AddChild, a member of class IXMLNodeAccess.
	CheckTextNode (see page 7483)	This is CheckTextNode, a member of class IXMLNodeAccess.
	ClearDocumentRef (see page 7483)	This is ClearDocumentRef, a member of class IXMLNodeAccess.
	CreateAttributeNode (see page 7483)	This is CreateAttributeNode, a member of class IXMLNodeAccess.
	CreateChildNode (see page 7483)	This is CreateChildNode, a member of class IXMLNodeAccess.
	CreateCollection (see page 7483)	This is CreateCollection, a member of class IXMLNodeAccess.
	DOMElement (see page 7484)	This is DOMElement, a member of class IXMLNodeAccess.
	FindHostedNode (see page 7484)	This is FindHostedNode, a member of class IXMLNodeAccess.
	GetChildNodeClasses (see page 7484)	This is GetChildNodeClasses, a member of class IXMLNodeAccess.
	GetHostNode (see page 7484)	This is GetHostNode, a member of class IXMLNodeAccess.
	GetNodeObject (see page 7485)	This is GetNodeObject, a member of class IXMLNodeAccess.
	HasChildNode (see page 7485)	This is the overview for the HasChildNode method overload.
	InternalAddChild (see page 7485)	This is InternalAddChild, a member of class IXMLNodeAccess.
	NestingLevel (see page 7486)	This is NestingLevel, a member of class IXMLNodeAccess.
	RegisterChildNode (see page 7486)	This is RegisterChildNode, a member of class IXMLNodeAccess.
	RegisterChildNodes (see page 7486)	This is RegisterChildNodes, a member of class IXMLNodeAccess.
	SetCollection (see page 7486)	This is SetCollection, a member of class IXMLNodeAccess.
	SetParentNode (see page 7487)	This is SetParentNode, a member of class IXMLNodeAccess.

79.2.2.2.1 IXMLNodeAccess.AddChild Method

Pascal

```
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; NodeClass:
TXMLNodeClass; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI,
TXMLNodeClass NodeClass, int Index = -1);
```

Description

This is AddChild, a member of class IXMLNodeAccess.

79.2.2.2.2 IXMLNodeAccess.CheckTextNode Method

Pascal

```
procedure CheckTextNode;
```

C++

```
__fastcall CheckTextNode();
```

Description

This is CheckTextNode, a member of class IXMLNodeAccess.

79.2.2.2.3 IXMLNodeAccess.ClearDocumentRef Method

Pascal

```
procedure ClearDocumentRef;
```

C++

```
__fastcall ClearDocumentRef();
```

Description

This is ClearDocumentRef, a member of class IXMLNodeAccess.

79.2.2.2.4 IXMLNodeAccess.CreateAttributeNode Method

Pascal

```
function CreateAttributeNode(const ADOMNode: IDOMNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateAttributeNode(const IDOMNode ADOMNode);
```

Description

This is CreateAttributeNode, a member of class IXMLNodeAccess.

79.2.2.2.5 IXMLNodeAccess.CreateChildNode Method

Pascal

```
function CreateChildNode(const ADOMNode: IDOMNode): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateChildNode(const IDOMNode ADOMNode);
```

Description

This is CreateChildNode, a member of class IXMLNodeAccess.

79.2.2.2.6 IXMLNodeAccess.CreateCollection Method

Pascal

```
function CreateCollection(const CollectionClass: TXMLNodeCollectionClass; const ItemIterface:
TGuid; const ItemTag: DOMString; ItemNS: DOMString = ''): TXMLNodeCollection;
```

C++

```
__fastcall TXMLNodeCollection CreateCollection(const TXMLNodeCollectionClass CollectionClass,
const TGuid ItemIterface, const DOMString ItemTag, DOMString ItemNS = '');
```

Description

This is CreateCollection, a member of class IXMLNodeAccess.

79.2.2.2.7 IXMLNodeAccess.DOMElement Method

Pascal

```
function DOMElement: IDOMElement;
```

C++

```
__fastcall IDOMElement DOMElement();
```

Description

This is DOMElement, a member of class IXMLNodeAccess.

79.2.2.2.8 IXMLNodeAccess.FindHostedNode Method

Pascal

```
function FindHostedNode(const NodeClass: TXMLNodeClass): IXMLNode;
```

C++

```
__fastcall IXMLNode FindHostedNode(const TXMLNodeClass NodeClass);
```

Description

This is FindHostedNode, a member of class IXMLNodeAccess.

79.2.2.2.9 IXMLNodeAccess.GetChildNodeClasses Method

Pascal

```
function GetChildNodeClasses: TNodeClassArray;
```

C++

```
__fastcall TNodeClassArray GetChildNodeClasses();
```

Description

This is GetChildNodeClasses, a member of class IXMLNodeAccess.

79.2.2.2.10 IXMLNodeAccess.GetHostNode Method

Pascal

```
function GetHostNode: TXMLNode;
```

C++

```
__fastcall TXMLNode GetHostNode();
```

Description

This is GetHostNode, a member of class IXMLNodeAccess.

79.2.2.2.11 IXMLNodeAccess.GetNodeObject Method

Pascal

```
function GetNodeObject: TXMLNode;
```

C++

```
__fastcall TXMLNode GetNodeObject();
```

Description

This is GetNodeObject, a member of class IXMLNodeAccess.

79.2.2.2.12 HasChildNode Method

This is the overview for the HasChildNode method overload.

Overload List

	Name	Description
	IXMLNodeAccess.HasChildNode (DOMString) (see page 7485)	This is HasChildNode, a member of class IXMLNodeAccess.
	IXMLNodeAccess.HasChildNode (DOMString, DOMString) (see page 7485)	This is HasChildNode, a member of class IXMLNodeAccess.

79.2.2.2.12.1 IXMLNodeAccess.HasChildNode Method (DOMString)

Pascal

```
function HasChildNode(const ChildTag: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag);
```

Description

This is HasChildNode, a member of class IXMLNodeAccess.

79.2.2.2.12.2 IXMLNodeAccess.HasChildNode Method (DOMString, DOMString)

Pascal

```
function HasChildNode(const ChildTag: DOMString; const NamespaceURI: DOMString): Boolean;
overload;
```

C++

```
__fastcall Boolean HasChildNode(const DOMString ChildTag, const DOMString NamespaceURI);
```

Description

This is HasChildNode, a member of class IXMLNodeAccess.

79.2.2.2.13 IXMLNodeAccess.InternalAddChild Method

Pascal

```
function InternalAddChild(NodeClass: TXMLNodeClass; const NodeName: DOMString; const
NamespaceURI: DOMString; Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode InternalAddChild(TXMLNodeClass NodeClass, const DOMString NodeName, const
DOMString NamespaceURI, int Index);
```

Description

This is InternalAddChild, a member of class IXMLNodeAccess.

79.2.2.2.14 IXMLNodeAccess.NestingLevel Method

Pascal

```
function NestingLevel: Integer;
```

C++

```
__fastcall int NestingLevel();
```

Description

This is NestingLevel, a member of class IXMLNodeAccess.

79.2.2.2.15 IXMLNodeAccess.RegisterChildNode Method

Pascal

```
procedure RegisterChildNode(const TagName: DOMString; ChildNodeClass: TXMLNodeClass;
NamespaceURI: DOMString = '');
```

C++

```
__fastcall RegisterChildNode(const DOMString TagName, TXMLNodeClass ChildNodeClass, DOMString
NamespaceURI = '');
```

Description

This is RegisterChildNode, a member of class IXMLNodeAccess.

79.2.2.2.16 IXMLNodeAccess.RegisterChildNodes Method

Pascal

```
procedure RegisterChildNodes(const TagNames: array of DOMString; const NodeClasses: array of
TXMLNodeClass);
```

C++

```
__fastcall RegisterChildNodes(const array of DOMString TagNames, const array of TXMLNodeClass
NodeClasses);
```

Description

This is RegisterChildNodes, a member of class IXMLNodeAccess.

79.2.2.2.17 IXMLNodeAccess.SetCollection Method

Pascal

```
procedure SetCollection(const Value: TXMLNodeCollection);
```

C++

```
__fastcall SetCollection(const TXMLNodeCollection Value);
```

Description

This is SetCollection, a member of class IXMLNodeAccess.

79.2.2.2.18 IXMLNodeAccess.SetParentNode Method

Pascal

```
procedure SetParentNode(const Value: TXMLNode);
```

C++

```
__fastcall SetParentNode(const TXMLNode Value);
```

Description

This is SetParentNode, a member of class IXMLNodeAccess.

79.2.2.3 IXMLNodeAccess Properties

The properties of the IXMLNodeAccess class are listed here.

Public Properties

	Name	Description
	ChildNodeClasses (see page 7487)	This is ChildNodeClasses, a member of class IXMLNodeAccess.
	HostNode (see page 7487)	This is HostNode, a member of class IXMLNodeAccess.

79.2.2.3.1 IXMLNodeAccess.ChildNodeClasses Property

Pascal

```
property ChildNodeClasses: TNodeClassArray;
```

C++

```
__property TNodeClassArray ChildNodeClasses;
```

Description

This is ChildNodeClasses, a member of class IXMLNodeAccess.

79.2.2.3.2 IXMLNodeAccess.HostNode Property

Pascal

```
property HostNode: TXMLNode;
```

C++

```
__property TXMLNode HostNode;
```

Description

This is HostNode, a member of class IXMLNodeAccess.

79.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	CheckEncoding (see page 7488)	This is function XMLDoc.CheckEncoding.
	CreateDOMNode (see page 7488)	Creates a new DOM node and returns its interface.
	DetectCharEncoding (see page 7489)	This is function XMLDoc.DetectCharEncoding.
	FormatXMLData (see page 7490)	Formats a string of XML so that it is more readable.
	LoadXMLData (see page 7490)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLData (see page 7490)	Creates and returns the interface of an XML document object, given a string of XML.
	LoadXMLDocument (see page 7491)	Returns the interface for an XML document given the name of an XML file.
	NewXMLDocument (see page 7491)	Returns the interface for a new, empty XML document.
	XMLDocError (see page 7492)	Raises an EXMLDocError (see page 7639) exception.
	XMLDocError (see page 7492)	Raises an EXMLDocError (see page 7639) exception.
	XMLStringToUnicodeString (see page 7492)	This is function XMLDoc.XMLStringToUnicodeString.
	XMLStringToWideString (see page 7492)	This is function XMLDoc.XMLStringToWideString.

79.3.1 XMLDoc.CheckEncoding Function

Pascal

```
procedure CheckEncoding(var XMLData: DOMString; const ValidEncodings: array of string);
```

C++

```
CheckEncoding(DOMString XMLData, const array of string ValidEncodings);
```

File

XMLDoc

Description

This is function XMLDoc.CheckEncoding.

79.3.2 XMLDoc.CreateDOMNode Function

Creates a new DOM node and returns its interface.

Pascal

```
function CreateDOMNode(Doc: IDOMDocument; const NameOrData: DOMString; NodeType: TNodeType =
ntElement; const AddlData: DOMString = ''): IDOMNode;
```

C++

```
IDOMNode CreateDOMNode(IDOMDocument Doc, const DOMString NameOrData, TNodeType NodeType =
ntElement, const DOMString AddlData = '');
```

File

XMLDoc

Description

CreateDOMNode generates a new DOM node.

Doc is the DOM interface for the document to which the new node will belong.

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

CreateDOMNode returns the interface for the newly created DOM node.

79.3.3 XMLDoc.DetectCharEncoding Function

Pascal

```
function DetectCharEncoding(S: TStream): TXmlEncodingType;
```

C++

```
TXmlEncodingType DetectCharEncoding(TStream S);
```

File

XMLDoc

Description

This is function XMLDoc.DetectCharEncoding.

79.3.4 XMLDoc.FormatXMLData Function

Formats a string of XML so that it is more readable.

Pascal

```
function FormatXMLData(const XMLData: DOMString): DOMString;
```

C++

```
DOMString FormatXMLData(const DOMString XMLData);
```

File

XMLDoc

Description

Use FormatXMLData to convert a string of XML into a format that represents its structure. FormatXMLData changes the input string (XMLData) so that each element node appears on its own line, indented appropriately to reflect its nesting in the node hierarchy.

79.3.5 XMLDoc.LoadXMLData Function

Creates and returns the interface of an XML document object, given a string of XML.

Pascal

```
function LoadXMLData(const XMLData: DOMString): IXMLDocument; overload;
function LoadXMLData(const XMLData: AnsiString): IXMLDocument; overload;
```

C++

```
IXMLDocument LoadXMLData(const DOMString XMLData);
IXMLDocument LoadXMLData(const AnsiString XMLData);
```

File

XMLDoc

Description

Call LoadXMLData to create an XML document that represents a specified XML string and obtain its interface.

XMLData is the XML for which you want an IXMLDocument (see page 7640) interface.

LoadXMLData returns an interface for the parsed value of XMLData.

LoadXMLData creates a new TXMLDocument (see page 7347) object to parse the XML and returns its IXMLDocument (see page 7640). The TXMLDocument (see page 7347) component is not given an owner, which means that it behaves like an interfaced object. When the application releases the interface that LoadXMLData returns, the TXMLDocument (see page 7347) instance is automatically freed.

See Also

TXMLDocument (see page 7347)

NewXMLDocument (see page 7491)

LoadXMLDocument (see page 7491)

79.3.6 XMLDoc.LoadXMLDocument Function

Returns the interface for an XML document given the name of an XML file.

Pascal

```
function LoadXMLDocument(const FileName: DOMString): IXMLDocument;
```

C++

```
IXMLDocument LoadXMLDocument(const DOMString FileName);
```

File

XMLDoc

Description

Call LoadXMLDocument to load an XML document from a file on disk and obtain its interface. LoadXMLDocument creates a new TXMLDocument (see page 7347) object and uses it to parse the file specified by the FileName parameter. The TXMLDocument (see page 7347) component is not given an owner, which means that it behaves like an interfaced object. When the application releases the interface that LoadXMLDocument returns, the TXMLDocument (see page 7347) instance is automatically freed.

See Also

TXMLDocument (see page 7347)

NewXMLDocument (see page 7491)

LoadXMLData (see page 7490)

79.3.7 XMLDoc.NewXMLDocument Function

Returns the interface for a new, empty XML document.

Pascal

```
function NewXMLDocument(Version: DOMString = '1.0'): IXMLDocument;
```

C++

```
IXMLDocument NewXMLDocument(DOMString Version = '1.0');
```

File

XMLDoc

Description

Call NewXMLDocument to create a TXMLDocument (see page 7347) object for a new, empty XML document.

Version specifies the XML version that the document should use. It becomes the value of the new document's Version property.

See Also

TXMLDocument (see page 7347)

LoadXMLDocument (see page 7491)

IXMLDocument (see page 7640)

79.3.8 XMLDoc.XMLDocError Function

Raises an EXMLDocError (see page 7639) exception.

Pascal

```
procedure XMLDocError(const Msg: string); overload;
procedure XMLDocError(const Msg: string; const Args: array of const); overload;
```

C++

```
XMLDocError(const UnicodeString Msg);
XMLDocError(const UnicodeString Msg, const array of const Args);
```

File

XMLDoc

Description

Call XMLDocError to raise an EXMLDocError (see page 7639) exception. Using XMLDocError rather than explicitly creating the EXMLDocError (see page 7639) instance can result in more space-efficient code.

Msg is the error message associated with the EXMLDocError (see page 7639) instance.

Args supplies arguments for any format specifiers embedded in Msg.

Note: Args_Size is the index of the last entry in the Args array. That is one less than the total number of arguments.

See Also

EXMLDocError (see page 7639)

79.3.9 XMLDoc.XMLStringToUnicodeString Function

Pascal

```
function XMLStringToUnicodeString(const XMLString: AnsiString): UnicodeString;
```

C++

```
UnicodeString XMLStringToUnicodeString(const AnsiString XMLString);
```

File

XMLDoc

Description

This is function XMLDoc.XMLStringToUnicodeString.

79.3.10 XMLDoc.XMLStringToWideString Function

Pascal

```
function XMLStringToWideString(const XMLString: AnsiString): WideString; deprecated;
```

C++

```
BSTR XMLStringToWideString(const AnsiString XMLString);
```

File

XMLDoc

Description

This is function XMLDoc.XMLStringToWideString.

79.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TNodeChange (see page 7494)	TNodeChange indicates the type of change made to a node in an XML document.
	TNodeListOperation (see page 7495)	TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList (see page 7703) interface.
	TXMLDocumentSource (see page 7496)	This is record XMLDoc.TXMLDocumentSource.
	TXMLPrologItem (see page 7497)	This is record XMLDoc.TXMLPrologItem.

Records

	Name	Description
	TNodeClassInfo (see page 7495)	TNodeClassInfo and TNodeClassArray (see page 7499) represent information about the classes that implement XML node interfaces.

79.4.1 XMLDoc.TNodeChange Enumeration

TNodeChange indicates the type of change made to a node in an XML document.

Pascal

```
TNodeChange = (
  ncUpdateValue,
  ncInsertChild,
  ncRemoveChild,
  ncAddAttribute,
  ncRemoveAttribute
);
```

C++

```
enum TNodeChange {
  ncUpdateValue,
  ncInsertChild,
  ncRemoveChild,
  ncAddAttribute,
  ncRemoveAttribute
};
```

File

XMLDoc

Description

The following table lists the possible values:

Value	Meaning
ncUpdateValue	The value of the node is changed.
ncInsertChild	A new child node is added to the node.
ncRemoveChild	A child node is removed from the node.

ncAddAttribute	A new attribute is added to the node.
ncRemoveAttribute	An attribute is removed from the node.

See Also

TNodeChangeEvent (see page 7498)

79.4.2 XMLDoc.TNodeClassInfo Record

TNodeClassInfo and TNodeClassArray (see page 7499) represent information about the classes that implement XML node interfaces.

Pascal

```
TNodeClassInfo = record
  NodeName: DOMString;
  NamespaceURI: DOMString;
  NodeClass: TXMLNodeClass;
end;
```

C++

```
struct TNodeClassInfo {
  DOMString NodeName;
  DOMString NamespaceURI;
  TXMLNodeClass NodeClass;
};
```

File

XMLDoc

Description

TNodeClassArray (see page 7499) is a dynamic array of TNodeClassInfo values. Each element of the array describes the implementation of a node.

TNodeClassInfo describes the name and implementation class for a node in an XML document. It contains the following fields:

Field	Meaning
NodeName	The name of the node. This string describes the identifier or use of the node. For example, on element nodes, this is the tag name. On attribute nodes, this is the name of the attribute. On comment nodes, this is the string "#comment". This is the value returned by a node's NodeName property, and is used as a property name in classes generated by the XML Data Binding Wizard.
NamespaceURI	The namespace of the node's name. This string is a Uniform Resource Identifier that identifies the namespace that contains NodeName. It is used to interpret the value of NodeName.
NodeClass	The class of a TXMLNode (see page 7396) descendant that represents the node.

79.4.3 XMLDoc.TNodeListOperation Enumeration

TNodeListOperation indicates what type of change is about to occur using an IXMLNodeList (see page 7703) interface.

Pascal

```
TNodeListOperation = (
  nlInsert,
  nlRemove,
  nlCreateNode
);
```

C++

```
enum TNodeListOperation {
  nlInsert,
  nlRemove,
  nlCreateNode
};
```

File

XMLDoc

Description

When an application makes a change using the IXMLNodeList (see page 7703) interface, the implementing object sends its owner a notification of that change. TNodeListOperation identifies the type of change that is about to occur. The following table lists the possible values:

Value	Meaning
nlInsert	An existing node is about to be added to the list.
nlRemove	A node is about to be removed from the list.
nlCreateNode	A new node needs to be created so that it can be added to the list.

79.4.4 XMLDoc.TXMLDocumentSource Enumeration

Pascal

```
TXMLDocumentSource = (
  xdsNone,
  xdsXMLProperty,
  xdsXMLData,
  xdsFile,
  xdsStream
);
```

C++

```
enum TXMLDocumentSource {
  xdsNone,
  xdsXMLProperty,
  xdsXMLData,
  xdsFile,
  xdsStream
};
```

File

XMLDoc

Description

This is record XMLDoc.TXMLDocumentSource.

79.4.5 XMLDoc.TXMLPrologItem Enumeration

Pascal

```
TXMLPrologItem = (
  xpVersion,
  xpEncoding,
  xpStandalone
);
```

C++

```
enum TXMLPrologItem {
  xpVersion,
  xpEncoding,
  xpStandalone
};
```

File

XMLDoc

Description

This is record XMLDoc.TXMLPrologItem.

79.5 Types

The following table lists types in this documentation.

Types

Name	Description
TNodeChangeEvent (see page 7498)	TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.
TNodeClassArray (see page 7499)	This is type XMLDoc.TNodeClassArray.
TNodeListNotification (see page 7499)	TNodeListNotification is the type of the callback the TXMLNodeList (see page 7460) calls when it adds, deletes, or creates XML nodes.
TXMLNodeArray (see page 7500)	TXMLNodeArray is a dynamic array of TXMLNode (see page 7396) objects.
TXMLNodeClass (see page 7501)	TXMLNodeClass is a class reference to a TXMLNode (see page 7396) descendant.
TXMLNodeCollectionClass (see page 7501)	TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 7444) descendant.

79.5.1 XMLDoc.TNodeChangeEvent Type

TNodeChangeEvent is the type for event handlers that respond when XML nodes are changed.

Pascal

```
TNodeChangeEvent = procedure (const Node: IXMLNode; ChangeType: TNodeChange) of object;
```

C++

```
(const Node: IXMLNode; ChangeType: TNodeChange) ( TNodeChangeEvent)();
```

File

XMLDoc

Description

TNodeChangeEvent is the type for the BeforeNodeChange and AfterNodeChange event handlers. These events occur before and after changes are made to the nodes in an XML document.

Node identifies the node to which the change occurs.

ChangeType indicates the type of change made to Node.

See Also

BeforeNodeChange (see page 7394)

AfterNodeChange (see page 7393)

Procedural types

Creating events

79.5.2 XMLDoc.TNodeClassArray Type

Pascal

```
TNodeClassArray = array of TNodeClassInfo;
```

C++

```
array of TNodeClassInfo TNodeClassArray;
```

File

XMLDoc

Description

This is type XMLDoc.TNodeClassArray.

79.5.3 XMLDoc.TNodeListNotification Type

TNodeListNotification is the type of the callback the TXMLNodeList (see page 7460) calls when it adds, deletes, or creates XML nodes.

Pascal

```
TNodeListNotification = procedure (Operation: TNodeListOperation; var Node: IXMLNode; const
IndexOrName: OleVariant; BeforeOperation: Boolean) of object;
```

C++

```
(Operation: TNodeListOperation; var Node: IXMLNode; const IndexOrName: OleVariant;
BeforeOperation: Boolean) ( TNodeListNotification)();
```

File

XMLDoc

Description

TNodeListNotification is the type for the callback supplied to TXMLNodeList (see page 7460) in its constructor. TXMLNodeList (see page 7460) calls this procedure before and after it adds or deletes nodes in the list and when it needs to create a new node.

Operation indicates what change is about to occur or has just occurred.

Node specifies the node that is the focus of that change. It's use depends on the operation, as indicated in the following table:

Operation	Node
nlInsert	The node that is about to be added to the list or that has just been added.
nlRemove	The node that is about to be removed from the list or that has just been removed.
nlCreateNode	The newly-created node. This should be created by the callback and returned as the Node parameter.
ntElement	The tag name.
ntAttribute	The name of the attribute.
ntText	The value of the node.

ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

IndexOrName is used only when Operation is nlCreateNode. It is the value assigned to the Nodes property using the index of a node that does not yet exist. The interpretation of IndexOrName depends on the type of node that the callback creates, as indicated in the following table:

NodeType	IndexOrName
ntElement	The tag name.
ntAttribute	The name of the attribute.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.

BeforeOperation indicates whether the callback occurs before the indicated operation or after. When nodes are inserted and removed, the callback is called first with BeforeOperation set to true, then the operation occurs, and then a second callback is made with BeforeOperation set to false. When nodes must be created, only one call is made, with BeforeOperation set to true.

See Also

TXMLNodeList (see page 7460)

TXMLNode (see page 7396)

IXMLNodeList (see page 7703)

79.5.4 XMLDoc.TXMLNodeArray Type

TXMLNodeArray is a dynamic array of TXMLNode (see page 7396) objects.

Pascal

```
TXMLNodeArray = array of TXMLNode;
```

C++

```
array of TXMLNode TXMLNodeArray;
```

File

XMLDoc

Description

TXMLNodeArray is a dynamic array of TXMLNode (see page 7396) objects, each of which represents a node in an XML document.

79.5.5 XMLDoc.TXMLNodeClass Type

TXMLNodeClass is a class reference to a TXMLNode (see page 7396) descendant.

Pascal

```
TXMLNodeClass = class of TXMLNode;
```

C++

```
TXMLNode TXMLNodeClass;
```

File

XMLDoc

Description

TXMLNodeClass describes a class that descends from TXMLNode (see page 7396).

See Also

Class References

79.5.6 XMLDoc.TXMLNodeCollectionClass Type

TXMLNodeCollectionClass is a class reference to a TXMLNodeCollection (see page 7444) descendant.

Pascal

```
TXMLNodeCollectionClass = class of TXMLNodeCollection;
```

C++

```
TXMLNodeCollection TXMLNodeCollectionClass;
```

File

XMLDoc

Description

TXMLNodeCollectionClass describes a class that descends from TXMLNodeCollection (see page 7444).

See Also

Class References

79.6 Variables

The following table lists variables in this documentation.

Variables

Name	Description
UseXSDBooleanStrings (see page 7502)	This is variable XMLDoc.UseXSDBooleanStrings.

79.6.1 XMLDoc.UseXSDBooleanStrings Variable

Pascal

```
UseXSDBooleanStrings: Boolean = True;
```

C++

```
Boolean UseXSDBooleanStrings = True;
```

File

XMLDoc

Description

This is variable XMLDoc.UseXSDBooleanStrings.

80 xmldom Namespace

This is namespace xmldom.

Topics

Name	Description
Classes (see page 7506)	The following table lists classes in this documentation.
Interfaces (see page 7521)	The following table lists interfaces in this documentation.
Functions (see page 7612)	The following table lists functions in this documentation.
Types (see page 7619)	The following table lists types in this documentation.
Variables (see page 7622)	The following table lists variables in this documentation.
Constants (see page 7624)	The following table lists constants in this documentation.

Classes

	Name	Description
	DOMException (see page 7506)	This is class xmldom.DOMException.
	EDOMParseError (see page 7508)	EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.
	TDOMVendor (see page 7514)	TDOMVendor is the base class for components that provide access to a DOM implementation.
	TDOMVendorList (see page 7516)	TDOMVendorList represents a collection of available DOM implementations.

Constants

Name	Description
ATTRIBUTE_NODE (see page 7624)	This is constant xmldom.ATTRIBUTE_NODE.
CDATA_SECTION_NODE (see page 7625)	This is constant xmldom.CDATA_SECTION_NODE.
COMMENT_NODE (see page 7625)	This is constant xmldom.COMMENT_NODE.
DOCUMENT_FRAGMENT_NODE (see page 7625)	This is constant xmldom.DOCUMENT_FRAGMENT_NODE.
DOCUMENT_NODE (see page 7626)	This is constant xmldom.DOCUMENT_NODE.
DOCUMENT_TYPE_NODE (see page 7626)	This is constant xmldom.DOCUMENT_TYPE_NODE.
DOMSTRING_SIZE_ERR (see page 7626)	This is constant xmldom.DOMSTRING_SIZE_ERR.
DOMWrapperVersion (see page 7627)	Indicates the version number for the vendor-neutral DOM interfaces.
ELEMENT_NODE (see page 7627)	This is constant xmldom.ELEMENT_NODE.
ENTITY_NODE (see page 7627)	This is constant xmldom.ENTITY_NODE.
ENTITY_REFERENCE_NODE (see page 7628)	This is constant xmldom.ENTITY_REFERENCE_NODE.
HIERARCHY_REQUEST_ERR (see page 7628)	This is constant xmldom.HIERARCHY_REQUEST_ERR.
INDEX_SIZE_ERR (see page 7628)	This is constant xmldom.INDEX_SIZE_ERR.

INUSE_ATTRIBUTE_ERR (see page 7629)	This is constant xmldom.INUSE_ATTRIBUTE_ERR.
INVALID_ACCESS_ERR (see page 7629)	This is constant xmldom.INVALID_ACCESS_ERR.
INVALID_CHARACTER_ERR (see page 7629)	This is constant xmldom.INVALID_CHARACTER_ERR.
INVALID_MODIFICATION_ERR (see page 7630)	This is constant xmldom.INVALID_MODIFICATION_ERR.
INVALID_STATE_ERR (see page 7630)	This is constant xmldom.INVALID_STATE_ERR.
NAMESPACE_ERR (see page 7630)	This is constant xmldom.NAMESPACE_ERR.
NO_DATA_ALLOWED_ERR (see page 7631)	This is constant xmldom.NO_DATA_ALLOWED_ERR.
NO_MODIFICATION_ALLOWED_ERR (see page 7631)	This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.
NOT_FOUND_ERR (see page 7631)	This is constant xmldom.NOT_FOUND_ERR.
NOT_SUPPORTED_ERR (see page 7632)	This is constant xmldom.NOT_SUPPORTED_ERR.
NOTATION_NODE (see page 7632)	This is constant xmldom.NOTATION_NODE.
NSDelim (see page 7632)	This is constant xmldom.NSDelim.
PROCESSING_INSTRUCTION_NODE (see page 7633)	This is constant xmldom.PROCESSING_INSTRUCTION_NODE.
SEncoding (see page 7633)	This is constant xmldom.SEncoding.
SHttp (see page 7633)	This is constant xmldom.SHttp.
SStandalone (see page 7634)	This is constant xmldom.SStandalone.
SVersion (see page 7634)	This is constant xmldom.SVersion.
SXML (see page 7634)	This is constant xmldom.SXML.
SXMLNamespaceURI (see page 7635)	This is constant xmldom.SXMLNamespaceURI.
SXMLNS (see page 7635)	This is constant xmldom.SXMLNS.
SXMLPrefixNamespaceURI (see page 7635)	This is constant xmldom.SXMLPrefixNamespaceURI.
SYNTAX_ERR (see page 7636)	This is constant xmldom.SYNTAX_ERR.
TEXT_NODE (see page 7636)	This is constant xmldom.TEXT_NODE.
WRONG_DOCUMENT_ERR (see page 7636)	This is constant xmldom.WRONG_DOCUMENT_ERR.

Functions

	Name	Description
	DOMVendorNotSupported (see page 7612)	This is function xmldom.DOMVendorNotSupported.
	ExtractLocalName (see page 7612)	Strips the namespace prefix, if present, from the tag name of an XML node.
	ExtractPrefix (see page 7613)	Returns the namespace prefix of an XML node's tag name.
	GetDOM (see page 7614)	Returns the top-level DOM interface for a registered DOM implementation.
	GetDOMNodeEx (see page 7614)	This is function xmldom.GetDOMNodeEx.
	GetDOMVendor (see page 7615)	Returns the TDOMVendor (see page 7514) object for a specified DOM vendor.
	IsPrefixed (see page 7615)	Indicates whether an XML tag name includes a namespace prefix.
	MakeNodeName (see page 7616)	Assembles a full tag name from a local name and namespace prefix.
	NodeMatches (see page 7616)	Indicates whether a specified node matches a given namespace and tag name.
	RegisterDOMVendor (see page 7617)	Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.
	SameNamespace (see page 7617)	Indicates whether a specified node is defined within a specified namespace.
	SameNamespace (see page 7617)	Indicates whether a specified node is defined within a specified namespace.
	UnRegisterDOMVendor (see page 7618)	Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 7617) .

Interfaces

	Name	Description
	IDOMAttr (see page 7521)	This is class xmldom.IDOMAttr.
	IDOMCDATASection (see page 7525)	This is class xmldom.IDOMCDATASection.
	IDOMCharacterData (see page 7526)	This is class xmldom.IDOMCharacterData.
	IDOMComment (see page 7530)	This is class xmldom.IDOMComment.
	IDOMDocument (see page 7530)	This is class xmldom.IDOMDocument.
	IDOMDocumentFragment (see page 7537)	This is class xmldom.IDOMDocumentFragment.
	IDOMDocumentType (see page 7538)	This is class xmldom.IDOMDocumentType.
	IDOMElement (see page 7542)	This is class xmldom.IDOMElement.
	IDOMEntity (see page 7549)	This is class xmldom.IDOMEntity.
	IDOMEntityReference (see page 7552)	This is class xmldom.IDOMEntityReference.

	IDOMImplementation (see page 7552)	This is class xmldom.IDOMImplementation.
	IDOMNamedNodeMap (see page 7554)	This is class xmldom.IDOMNamedNodeMap.
	IDOMNode (see page 7557)	IDOMNode is the DOM interface for a node in an XML document.
	IDOMNodeEx (see page 7577)	This is class xmldom.IDOMNodeEx.
	IDOMNodeList (see page 7581)	This is class xmldom.IDOMNodeList.
	IDOMNodeSelect (see page 7583)	This is class xmldom.IDOMNodeSelect.
	IDOMNotation (see page 7584)	This is class xmldom.IDOMNotation.
	IDOMParseError (see page 7586)	This is class xmldom.IDOMParseError.
	IDOMParseOptions (see page 7591)	This is class xmldom.IDOMParseOptions.
	IDOMPersist (see page 7595)	This is class xmldom.IDOMPersist.
	IDOMProcessingInstruction (see page 7599)	This is class xmldom.IDOMProcessingInstruction.
	IDOMText (see page 7602)	This is class xmldom.IDOMText.
	IDOMXMLProlog (see page 7604)	This is class xmldom.IDOMXMLProlog.
	IDOMXSLProcessor (see page 7607)	This is class xmldom.IDOMXSLProcessor.

Types

Name	Description
DOMNodeType (see page 7619)	DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.
DOMString (see page 7619)	DOMString is the type for strings that are used by a DOM parser.
DOMTimeStamp (see page 7620)	This is type xmldom.DOMTimeStamp.
TAsyncEventHandler (see page 7620)	TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.
TDOMVendorArray (see page 7621)	This is type xmldom.TDOMVendorArray.

Variables

Name	Description
DefaultDOMVendor (see page 7622)	Identifies the DOM implementation that is used by default.
DOMVendors (see page 7622)	Lists all registered DOM Vendors.

80.1 Classes

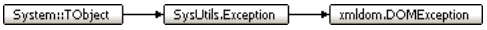
The following table lists classes in this documentation.

Classes

	Name	Description
	DOMException (see page 7506)	This is class xmldom.DOMException.
	EDOMParseError (see page 7508)	EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.
	TDOMVendor (see page 7514)	TDOMVendor is the base class for components that provide access to a DOM implementation.
	TDOMVendorList (see page 7516)	TDOMVendorList represents a collection of available DOM implementations.

80.1.1 DOMException Class

Class Hierarchy



Pascal

```
DOMException = class(Exception);
```

C++

```
class DOMException : public Exception;
```

File

xmldom

Description

This is class xmldom.DOMException.

80.1.1.1 DOMException Members

The following tables list the members exposed by DOMException.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	code (see page 7508)	This is code, a member of class DOMException.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 3355)	Constructs an object and initializes its data before the object is first used.
	Create (see page 3952)	Instantiates an instance of an exception with a simple message string.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
	Destroy (see page 3356)	Disposes of an object instance.
	Destroy (see page 3960)	Disposes of an object instance.
	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.
	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3380)	Returns a string containing the class name.
	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.

80.1.1.2 DOMException Fields

The fields of the DOMException class are listed here.

Public Fields

	Name	Description
	code (see page 7508)	This is code, a member of class DOMException.

80.1.1.2.1 DOMException.code Field

Pascal

```
code: Word;
```

C++

```
Word code;
```

Description

This is code, a member of class DOMException.

80.1.2 EDOMParseError Class

EDOMParseError is the exception class for errors that occur when a DOM implementation parses an XML document.

Class Hierarchy

Pascal

```
EDOMParseError = class(Exception);
```

C++

```
class EDOMParseError : public Exception;
```

File

xmldom

Description

EDOMParseError is raised when TXMLDocument encounters an error while parsing an XML document. It wraps the information that is available from the DOM implementation's IDOMParseError (see page 7586) interface.

See Also

TXMLDocument (see page 7347)

80.1.2.1 EDOMParseError Members

The following tables list the members exposed by EDOMParseError.

Protected Methods

	Name	Description
	GetStackTrace (see page 3962)	This is GetStackTrace, a member of class Exception.
V	RaisingException (see page 3962)	This is RaisingException, a member of class Exception.
	SetInnerException (see page 3962)	This is SetInnerException, a member of class Exception.
	SetStackInfo (see page 3963)	This is SetStackInfo, a member of class Exception.

Protected Properties

	Name	Description
R	ParseError (see page 7512)	This is ParseError, a member of class EDOMParseError.

Public Fields

	Name	Description
	CleanUpStackInfoProc (see page 3961)	This is CleanUpStackInfoProc, a member of class Exception.
	GetExceptionStackInfoProc (see page 3961)	This is GetExceptionStackInfoProc, a member of class Exception.
	GetStackInfoStringProc (see page 3961)	This is GetStackInfoStringProc, a member of class Exception.

Public Methods

	Name	Description
V	AfterConstruction (see page 3357)	Responds after the last constructor has executed.
V	BeforeDestruction (see page 3358)	Responds before the first destructor executes.
	ClassInfo (see page 3358)	Returns a pointer to the runtime type information (RTTI) table for the object type.
	ClassName (see page 3359)	Returns a string indicating the type of the object instance (as opposed to the type of the variable passed as an argument).
	ClassNameIs (see page 3361)	Determines whether an object is of a specific type.
	ClassParent (see page 3361)	Returns the type of the immediate ancestor of a class.
	ClassType (see page 3364)	Returns the class reference for the object's class.
	CleanupInstance (see page 3367)	Performs finalization on long strings, variants, and interface variables within a class.
	Create (see page 7510)	This is Create, a member of class EDOMParseError.
	CreateFmt (see page 3954)	Instantiates an instance of an exception with a formatted message string.
	CreateFmtHelp (see page 3955)	Instantiates an instance of an exception with a formatted message string and a help-context ID.
	CreateHelp (see page 3955)	Creates an instance of an exception with a simple message string and a help-context ID.
	CreateRes (see page 3956)	This is the overview for the CreateRes constructor overload.
	CreateResFmt (see page 3957)	This is the overview for the CreateResFmt constructor overload.
	CreateResFmtHelp (see page 3958)	This is the overview for the CreateResFmtHelp constructor overload.
	CreateResHelp (see page 3959)	This is the overview for the CreateResHelp constructor overload.
V	DefaultHandler (see page 3367)	Provides the interface for a method that processes message records.
V	Destroy (see page 3356)	Disposes of an object instance.
V	Destroy (see page 3960)	Disposes of an object instance.
V	Dispatch (see page 3369)	Calls message-handling methods for the object, based on the contents of the Message parameter.
V	Equals (see page 3371)	Checks if the current instance and the parameter are equal.
	FieldAddress (see page 3371)	This is the overview for the FieldAddress method overload.
	Free (see page 3372)	Destroys an object and frees its associated memory, if necessary.

	FreeInstance (see page 3374)	Deallocates memory allocated by a previous call to the System::TObject::NewInstance (see page 3379) method.
	GetBaseException (see page 3962)	This is GetBaseException, a member of class Exception.
	GetHashCode (see page 3375)	Returns an integer containing the hash code.
	GetInterface (see page 3375)	Retrieves a specified interface.
	GetInterfaceEntry (see page 3376)	Returns the entry for a specific interface implemented in a class.
	GetInterfaceTable (see page 3376)	Returns a pointer to a structure containing all of the interfaces implemented by a given class.
	InheritsFrom (see page 3376)	Determines the relationship of two object types.
	InitInstance (see page 3377)	Initializes a newly allocated object instance to all zeros and initializes the instance's virtual method table pointer.
	InstanceSize (see page 3377)	Returns the size in bytes of each instance of the object type.
	MethodAddress (see page 3378)	This is the overview for the MethodAddress method overload.
	MethodName (see page 3379)	Returns the name of a class method by address.
	NewInstance (see page 3379)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	RaiseOuterException (see page 3962)	This is RaiseOuterException, a member of class Exception.
	SafeCallException (see page 3380)	Handles exceptions in methods declared using the safecall calling convention.
	ThrowOuterException (see page 3963)	This is ThrowOuterException, a member of class Exception.
	ToString (see page 3380)	Returns a string containing the class name.
	ToString (see page 3963)	Returns a string containing the class name.
	UnitName (see page 3380)	Returns a string containing the unit name.

Public Properties

	Name	Description
	BaseException (see page 3964)	This is BaseException, a member of class Exception.
	ErrorCode (see page 7511)	Indicates the error code that the DOM parser returned.
	FilePos (see page 7511)	Indicates the offset in the XML document where the parser encountered an error.
	HelpContext (see page 3964)	Indicates the context ID number for context-sensitive online help associated with the exception.
	InnerException (see page 3965)	This is InnerException, a member of class Exception.
	Line (see page 7512)	Indicates the line of the XML document where the parser encountered an error.
	LinePos (see page 7512)	Indicates the position within the problem line where the parser encountered an error.
	Message (see page 3965)	Message contains the text string to display in the exception dialog box when the exception is raised.
	Reason (see page 7513)	Indicates the error code that the DOM parser returned.
	SrcText (see page 7513)	Indicates the text of the line where the parser encountered an error.
	StackInfo (see page 3966)	This is StackInfo, a member of class Exception.
	StackTrace (see page 3966)	This is StackTrace, a member of class Exception.
	URL (see page 7513)	Identifies the source that could not be parsed by its URL.

80.1.2.2 EDOMParseError.Create Constructor

Pascal

```
constructor Create(const ParseError: IDOMParseError; const Msg: string);
```

C++

```
__fastcall EDOMParseError(const IDOMParseError ParseError, const UnicodeString Msg);
```

Description

This is Create, a member of class EDOMParseError.

80.1.2.3 EDOMParseError Properties

The properties of the EDOMParseError class are listed here.

Protected Properties

	Name	Description
	ParseError (see page 7512)	This is ParseError, a member of class EDOMParseError.

Public Properties

	Name	Description
	ErrorCode (see page 7511)	Indicates the error code that the DOM parser returned.
	FilePos (see page 7511)	Indicates the offset in the XML document where the parser encountered an error.
	Line (see page 7512)	Indicates the line of the XML document where the parser encountered an error.
	LinePos (see page 7512)	Indicates the position within the problem line where the parser encountered an error.
	Reason (see page 7513)	Indicates the error code that the DOM parser returned.
	SrcText (see page 7513)	Indicates the text of the line where the parser encountered an error.
	URL (see page 7513)	Identifies the source that could not be parsed by its URL.

80.1.2.3.1 EDOMParseError.ErrorCode Property

Indicates the error code that the DOM parser returned.

Pascal

```
property ErrorCode: Integer;
```

C++

```
__property int ErrorCode;
```

Description

ErrorCode is the error code supplied by the DOM parser when it encountered a problem parsing the XML document.

See Also

Reason (see page 7513)

80.1.2.3.2 EDOMParseError.FilePos Property

Indicates the offset in the XML document where the parser encountered an error.

Pascal

```
property FilePos: Integer;
```

C++

```
__property int FilePos;
```

Description

FilePos is the offset in the XML document where the parser detected the error.

See Also

Line (see page 7512)

LinePos (see page 7512)

SrcText (see page 7513)

URL (see page 7513)

80.1.2.3.3 EDOMParseError.Line Property

Indicates the line of the XML document where the parser encountered an error.

Pascal

```
property Line: Integer;
```

C++

```
__property int Line;
```

Description

Line is the line of the XML document where the parser detected the error.

See Also

FilePos (see page 7511)

LinePos (see page 7512)

SrcText (see page 7513)

URL (see page 7513)

80.1.2.3.4 EDOMParseError.LinePos Property

Indicates the position within the problem line where the parser encountered an error.

Pascal

```
property LinePos: Integer;
```

C++

```
__property int LinePos;
```

Description

LinePos is the offset in the line specified by the Line (see page 7512) property where the DOM parser detected an error.

See Also

Line (see page 7512)

FilePos (see page 7511)

SrcText (see page 7513)

URL (see page 7513)

80.1.2.3.5 EDOMParseError.ParseError Property

Pascal

```
property ParseError: IDOMParseError;
```

C++

```
__property IDOMParseError ParseError;
```

Description

This is ParseError, a member of class EDOMParseError.

80.1.2.3.6 EDOMParseError.Reason Property

Indicates the error code that the DOM parser returned.

Pascal

```
property Reason: DOMString;
```

C++

```
__property DOMString Reason;
```

Description

Reason is a short text string that describes the error that was encountered. It corresponds to the error code specified by ErrorCode (see page 7511).

See Also

ErrorCode (see page 7511)

Message (see page 3965)

80.1.2.3.7 EDOMParseError.SrcText Property

Indicates the text of the line where the parser encountered an error.

Pascal

```
property SrcText: DOMString;
```

C++

```
__property DOMString SrcText;
```

Description

SrcText is the text from the XML document that the DOM parser could not interpret.

See Also

Line (see page 7512)

80.1.2.3.8 EDOMParseError.URL Property

Identifies the source that could not be parsed by its URL.

Pascal

```
property URL: DOMString;
```

C++

```
__property DOMString URL;
```

Description

URL identifies the problem the DOM parser encountered by its URL.

See Also

SrcText (see page 7513)

80.1.3 TDOMVendor Class

TDOMVendor is the base class for components that provide access to a DOM implementation.

Class Hierarchy

Pascal

```
TDOMVendor = class;
```

C++

```
class TDOMVendor;
```

File

xmldom

Description

Note: Use TDOMVendor as a base class for a class that can be used to represent a DOM implementation. TDOMVendor introduces two abstract (pure virtual) methods, which provide information about a specific DOM implementation. One returns the IDOMImplementation (see page 7552) interface that acts as the top-level interface for a DOM vendor's implementation. The other returns a unique string that identifies the DOM implementation.

Once a TDOMVendor object has been registered, TXMLDocument objects can assign the DOM implementation as the value of their DOMVendor property by choosing the identifying string in the Object Inspector.

See Also

RegisterDOMVendor (see page 7617)

UnRegisterDOMVendor (see page 7618)

TXMLDocument (see page 7347)

GetDOMVendor (see page 7615)

Using the Document Object Model

80.1.3.1 TDOMVendor Members

The following tables list the members exposed by TDOMVendor.

Public Methods

	Name	Description
	Description (see page 7515)	Returns a string that uniquely identifies the associated DOM implementation.
	DOMImplementation (see page 7515)	Returns the IDOMImplementation (see page 7552) interface for the associated DOM implementation.

80.1.3.2 TDOMVendor Methods

The methods of the TDOMVendor class are listed here.

Public Methods

	Name	Description
	Description (see page 7515)	Returns a string that uniquely identifies the associated DOM implementation.
	DOMImplementation (see page 7515)	Returns the IDOMImplementation (see page 7552) interface for the associated DOM implementation.

80.1.3.2.1 TDOMVendor.Description Method

Returns a string that uniquely identifies the associated DOM implementation.

Pascal

```
function Description: string; virtual; abstract;
```

C++

```
virtual __fastcall UnicodeString Description() = 0;
```

Description

Description returns the name, or description, of the associated DOM implementation. This is the string that appears in the Object Inspector for a TXMLDocument object's DOMVendor property.

Description is an abstract (pure virtual) method in TDOMVendor (see page 7514). Descendant classes override this method to provide a unique identifier for the DOM implementation.

See Also

DOMVendor (see page 7386)

GetDOMVendor (see page 7615)

80.1.3.2.2 TDOMVendor.DOMImplementation Method

Returns the IDOMImplementation (see page 7552) interface for the associated DOM implementation.

Pascal

```
function DOMImplementation: IDOMImplementation; virtual; abstract;
```

C++

```
virtual __fastcall IDOMImplementation DOMImplementation() = 0;
```

Description

DOMImplementation returns the top-level interface for a DOM implementation. This interface is the starting point for any application that uses the DOM implementation to parse and edit an XML document.

DOMImplementation is an abstract (pure virtual) method in TDOMVendor (see page 7514). Descendant classes override this method to return the IDOMImplementation (see page 7552) interface for the associated DOM implementation.

Note: For more information about the IDOMImplementation (see page 7552), see the W3C Document Object Model (DOM) Level 2 Specification or the documentation provided by your DOM vendor.

See Also

DOMImplementation (see page 7385)

Description (see page 7515)

80.1.4 TDOMVendorList Class

TDOMVendorList represents a collection of available DOM implementations.

Class Hierarchy

Pascal

```
TDOMVendorList = class;
```

C++

```
class TDOMVendorList;
```

File

xmldom

Description

TDOMVendorList is the type of the global DOMVendors (see page 7622) variable. It represents a collection of DOM implementations (TDOMVendor (see page 7514) objects).

TDOMVendorList introduces a single property, Vendors (see page 7519), that can be used to iterate through all the DOM vendors in the collection. It also introduces methods that can be used to add, delete, or locate vendors.

Note: Do not explicitly use TDOMVendorList methods to add or delete vendors from the global DOMVendors (see page 7622) variable. Instead, add or remove vendors from this global list by calling the global RegisterDOMVendor (see page 7617) and UnRegisterDOMVendor (see page 7618) procedures.

See Also

DOMVendors (see page 7622)

TXMLDocument (see page 7347)

RegisterDOMVendor (see page 7617)

TDOMVendor (see page 7514)

UnRegisterDOMVendor (see page 7618)

80.1.4.1 TDOMVendorList Members

The following tables list the members exposed by TDOMVendorList.

Protected Methods

	Name	Description
	GetVendors (see page 7518)	This is GetVendors, a member of class TDOMVendorList.

Public Methods

	Name	Description
	Add (see page 7517)	Adds a new DOM implementation to the end of the list.
	Count (see page 7517)	Returns the number of DOM vendors in the collection.
	Find (see page 7518)	Returns a vendor from the list, given its description.
	Remove (see page 7519)	Removes a specified vendor from the list.

Public Properties

	Name	Description
	Vendors (see page 7519)	Lists the DOM vendors maintained by this TDOMVendorList (see page 7516) object.

80.1.4.2 TDOMVendorList Methods

The methods of the TDOMVendorList class are listed here.

Protected Methods

	Name	Description
	GetVendors (see page 7518)	This is GetVendors, a member of class TDOMVendorList.

Public Methods

	Name	Description
	Add (see page 7517)	Adds a new DOM implementation to the end of the list.
	Count (see page 7517)	Returns the number of DOM vendors in the collection.
	Find (see page 7518)	Returns a vendor from the list, given its description.
	Remove (see page 7519)	Removes a specified vendor from the list.

80.1.4.2.1 TDOMVendorList.Add Method

Adds a new DOM implementation to the end of the list.

Pascal

```
procedure Add(const Vendor: TDOMVendor);
```

C++

```
__fastcall Add(const TDOMVendor Vendor);
```

Description

Call Add to add a DOM vendor to the end of the list.

Vendor is an object that provides access to the IDOMImplementation (see page 7552) interface for the new DOM vendor.

If the list already includes a DOM vendor with the same description as that supplied by Vendor, Add raises an exception.

See Also

Count (see page 7517)

Vendors (see page 7519)

Find (see page 7518)

Remove (see page 7519)

Description (see page 7515)

80.1.4.2.2 TDOMVendorList.Count Method

Returns the number of DOM vendors in the collection.

Pascal

```
function Count: Integer;
```

C++

```
__fastcall int Count();
```

Description

Call Count to determine the number of DOM implementations stored in the list. Count returns a value that can be used as an upper bound when iterating over the vendors using the Vendors (see page 7519) property.

Count increases when the Add (see page 7517) method adds a vendor to the list and decreases when the Remove (see page 7519) method removes a vendor from the list.

See Also

Vendors (see page 7519)

Add (see page 7517)

Remove (see page 7519)

80.1.4.2.3 TDOMVendorList.Find Method

Returns a vendor from the list, given its description.

Pascal

```
function Find(const VendorDesc: string): TDOMVendor;
```

C++

```
__fastcall TDOMVendor Find(const UnicodeString VendorDesc);
```

Description

Call Find to access a vendor in the list, given its description. If there is no vendor in the list with the specified description, Find returns nil (Delphi) or NULL (C++).

VendorDesc is the string that is returned by the TDOMVendor (see page 7514) object's Description method.

Find returns a TDOMVendor (see page 7514) object, which can be used as a value for the DOMVendor property of a TXMLDocument component.

See Also

Vendors (see page 7519)

Description (see page 7515)

DOMVendor (see page 7386)

80.1.4.2.4 TDOMVendorList.GetVendors Method

Pascal

```
function GetVendors(Index: Integer): TDOMVendor;
```

C++

```
__fastcall TDOMVendor GetVendors(int Index);
```

Description

This is GetVendors, a member of class TDOMVendorList.

80.1.4.2.5 TDOMVendorList.Remove Method

Removes a specified vendor from the list.

Pascal

```
procedure Remove(const Vendor: TDOMVendor);
```

C++

```
__fastcall Remove(const TDOMVendor Vendor);
```

Description

Remove removes the specified vendor from the list.

Vendor is the DOM vendor to remove from the list. If the list does not include a vendor with the same description as this parameter, Remove does nothing.

See Also

Count (see page 7517)

Vendors (see page 7519)

Find (see page 7518)

Add (see page 7517)

Description (see page 7515)

80.1.4.3 TDOMVendorList Properties

The properties of the TDOMVendorList class are listed here.

Public Properties

	Name	Description
	Vendors (see page 7519)	Lists the DOM vendors maintained by this TDOMVendorList (see page 7516) object.

80.1.4.3.1 TDOMVendorList.Vendors Property

Lists the DOM vendors maintained by this TDOMVendorList (see page 7516) object.

Pascal

```
property Vendors [Index: Integer]: TDOMVendor;
```

C++

```
__property TDOMVendor Vendors[int Index];
```

Description

Use Vendors to access the DOM vendors in this collection. Index is the index of the vendor. Its value ranges from 0 to one less than the value returned by the Count (see page 7517) method.

Note: In Delphi, Vendors is the default property for TDOMVendorList (see page 7516). This means that the property name can be omitted in code. That is, the statement

for i := 0 to DOMVendors. Count (see page 7517) do

StringList1.Add (see page 7517)(DOMVendors.Vendors[i].Description);

Can be written

for i := 0 to DOMVendors.Count (see page 7517) do

StringList1.Add (see page 7517)(DOMVendors (see page 7622)[i].Description);

Note: In C++, Vendors can be accessed using the [] operator, to achieve an affect similar to the default property in Delphi.

See Also

Count (see page 7517)

Add (see page 7517)

Find (see page 7518)

Remove (see page 7519)

80.2 Interfaces

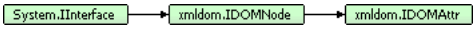
The following table lists interfaces in this documentation.

Interfaces

	Name	Description
	IDOMAttr (see page 7521)	This is class xmldom.IDOMAttr.
	IDOMCDATASection (see page 7525)	This is class xmldom.IDOMCDATASection.
	IDOMCharacterData (see page 7526)	This is class xmldom.IDOMCharacterData.
	IDOMComment (see page 7530)	This is class xmldom.IDOMComment.
	IDOMDocument (see page 7530)	This is class xmldom.IDOMDocument.
	IDOMDocumentFragment (see page 7537)	This is class xmldom.IDOMDocumentFragment.
	IDOMDocumentType (see page 7538)	This is class xmldom.IDOMDocumentType.
	IDOMElement (see page 7542)	This is class xmldom.IDOMElement.
	IDOMEntity (see page 7549)	This is class xmldom.IDOMEntity.
	IDOMEntityReference (see page 7552)	This is class xmldom.IDOMEntityReference.
	IDOMImplementation (see page 7552)	This is class xmldom.IDOMImplementation.
	IDOMNamedNodeMap (see page 7554)	This is class xmldom.IDOMNamedNodeMap.
	IDOMNode (see page 7557)	IDOMNode is the DOM interface for a node in an XML document.
	IDOMNodeEx (see page 7577)	This is class xmldom.IDOMNodeEx.
	IDOMNodeList (see page 7581)	This is class xmldom.IDOMNodeList.
	IDOMNodeSelect (see page 7583)	This is class xmldom.IDOMNodeSelect.
	IDOMNotation (see page 7584)	This is class xmldom.IDOMNotation.
	IDOMParseError (see page 7586)	This is class xmldom.IDOMParseError.
	IDOMParseOptions (see page 7591)	This is class xmldom.IDOMParseOptions.
	IDOMPersist (see page 7595)	This is class xmldom.IDOMPersist.
	IDOMProcessingInstruction (see page 7599)	This is class xmldom.IDOMProcessingInstruction.
	IDOMText (see page 7602)	This is class xmldom.IDOMText.
	IDOMXMLProlog (see page 7604)	This is class xmldom.IDOMXMLProlog.
	IDOMXSLProcessor (see page 7607)	This is class xmldom.IDOMXSLProcessor.

80.2.1 IDOMAttr Interface

Class Hierarchy



Pascal

```
IDOMAttr = interface(IDOMNode);
```

C++

```
__interface IDOMAttr : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMAttr.

80.2.1.1 IDOMAttr Members

The following tables list the members exposed by IDOMAttr.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_name (see page 7523)	This is get_name, a member of class IDOMAttr.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_ownerElement (see page 7523)	This is get_ownerElement, a member of class IDOMAttr.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	get_specified (see page 7523)	This is get_specified, a member of class IDOMAttr.
	get_value (see page 7524)	This is get_value, a member of class IDOMAttr.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	set_value (see page 7524)	This is set_value, a member of class IDOMAttr.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	name (see page 7524)	This is name, a member of class IDOMAttr.
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.

R	ownerElement (see page 7524)	This is ownerElement, a member of class IDOMAttr.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.
R	specified (see page 7525)	This is specified, a member of class IDOMAttr.
	value (see page 7525)	This is value, a member of class IDOMAttr.

80.2.1.2 IDOMAttr Methods

The methods of the IDOMAttr class are listed here.

Public Methods

	Name	Description
	get_name (see page 7523)	This is get_name, a member of class IDOMAttr.
	get_ownerElement (see page 7523)	This is get_ownerElement, a member of class IDOMAttr.
	get_specified (see page 7523)	This is get_specified, a member of class IDOMAttr.
	get_value (see page 7524)	This is get_value, a member of class IDOMAttr.
	set_value (see page 7524)	This is set_value, a member of class IDOMAttr.

80.2.1.2.1 IDOMAttr.get_name Method

Pascal

```
function get_name: DOMString; safecall;
```

C++

```
__fastcall DOMString get_name();
```

Description

This is get_name, a member of class IDOMAttr.

80.2.1.2.2 IDOMAttr.get_ownerElement Method

Pascal

```
function get_ownerElement: IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement get_ownerElement();
```

Description

This is get_ownerElement, a member of class IDOMAttr.

80.2.1.2.3 IDOMAttr.get_specified Method

Pascal

```
function get_specified: WordBool; safecall;
```

C++

```
__fastcall WordBool get_specified();
```

Description

This is get_specified, a member of class IDOMAttr.

80.2.1.2.4 IDOMAttr.get_value Method

Pascal

```
function get_value: DOMString; safecall;
```

C++

```
__fastcall DOMString get_value();
```

Description

This is get_value, a member of class IDOMAttr.

80.2.1.2.5 IDOMAttr.set_value Method

Pascal

```
procedure set_value(const attributeValue: DOMString); safecall;
```

C++

```
__fastcall set_value(const DOMString attributeValue);
```

Description

This is set_value, a member of class IDOMAttr.

80.2.1.3 IDOMAttr Properties

The properties of the IDOMAttr class are listed here.

Public Properties

	Name	Description
R	name (see page 7524)	This is name, a member of class IDOMAttr.
R	ownerElement (see page 7524)	This is ownerElement, a member of class IDOMAttr.
R	specified (see page 7525)	This is specified, a member of class IDOMAttr.
	value (see page 7525)	This is value, a member of class IDOMAttr.

80.2.1.3.1 IDOMAttr.name Property

Pascal

```
property name: DOMString;
```

C++

```
__property DOMString name;
```

Description

This is name, a member of class IDOMAttr.

80.2.1.3.2 IDOMAttr.ownerElement Property

Pascal

```
property ownerElement: IDOMElement;
```

C++

```
__property IDOMElement ownerElement;
```

Description

This is ownerElement, a member of class IDOMAttr.

80.2.1.3.3 IDOMAttr.specified Property

Pascal

```
property specified: WordBool;
```

C++

```
__property WordBool specified;
```

Description

This is specified, a member of class IDOMAttr.

80.2.1.3.4 IDOMAttr.value Property

Pascal

```
property value: DOMString;
```

C++

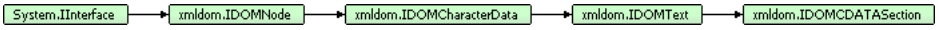
```
__property DOMString value;
```

Description

This is value, a member of class IDOMAttr.

80.2.2 IDOMCDATASection Interface

Class Hierarchy



Pascal

```
IDOMCDATASection = interface(IDOMText);
```

C++

```
__interface IDOMCDATASection : public IDOMText;
```

File

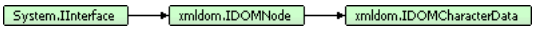
xmldom

Description

This is class xmldom.IDOMCDATASection.

80.2.3 IDOMCharacterData Interface

Class Hierarchy



Pascal

```
IDOMCharacterData = interface(IDOMNode);
```

C++

```
__interface IDOMCharacterData : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMCharacterData.

80.2.3.1 IDOMCharacterData Members

The following tables list the members exposed by IDOMCharacterData.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	appendData (see page 7527)	This is appendData, a member of class IDOMCharacterData.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	deleteData (see page 7528)	This is deleteData, a member of class IDOMCharacterData.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_data (see page 7528)	This is get_data, a member of class IDOMCharacterData.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_length (see page 7528)	This is get_length, a member of class IDOMCharacterData.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	insertData (see page 7528)	This is insertData, a member of class IDOMCharacterData.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	replaceData (see page 7528)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 7529)	This is set_data, a member of class IDOMCharacterData.
	set_nodeValue (see page 7567)	Sets the value of the node.
	substringData (see page 7529)	This is substringData, a member of class IDOMCharacterData.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
	data (see page 7529)	This is data, a member of class IDOMCharacterData.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	length (see page 7529)	This is length, a member of class IDOMCharacterData.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.

80.2.3.2 IDOMCharacterData Methods

The methods of the IDOMCharacterData class are listed here.

Public Methods

	Name	Description
	appendData (see page 7527)	This is appendData, a member of class IDOMCharacterData.
	deleteData (see page 7528)	This is deleteData, a member of class IDOMCharacterData.
	get_data (see page 7528)	This is get_data, a member of class IDOMCharacterData.
	get_length (see page 7528)	This is get_length, a member of class IDOMCharacterData.
	insertData (see page 7528)	This is insertData, a member of class IDOMCharacterData.
	replaceData (see page 7528)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 7529)	This is set_data, a member of class IDOMCharacterData.
	substringData (see page 7529)	This is substringData, a member of class IDOMCharacterData.

80.2.3.2.1 IDOMCharacterData.appendData Method

Pascal

```
procedure appendData(const data: DOMString); safecall;
```

C++

```
__fastcall appendData(const DOMString data);
```

Description

This is appendData, a member of class IDOMCharacterData.

80.2.3.2.2 IDOMCharacterData.deleteData Method

Pascal

```
procedure deleteData(offset: Integer; count: Integer); safecall;
```

C++

```
__fastcall deleteData(int offset, int count);
```

Description

This is deleteData, a member of class IDOMCharacterData.

80.2.3.2.3 IDOMCharacterData.get_data Method

Pascal

```
function get_data: DOMString; safecall;
```

C++

```
__fastcall DOMString get_data();
```

Description

This is get_data, a member of class IDOMCharacterData.

80.2.3.2.4 IDOMCharacterData.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMCharacterData.

80.2.3.2.5 IDOMCharacterData.insertData Method

Pascal

```
procedure insertData(offset: Integer; const data: DOMString); safecall;
```

C++

```
__fastcall insertData(int offset, const DOMString data);
```

Description

This is insertData, a member of class IDOMCharacterData.

80.2.3.2.6 IDOMCharacterData.replaceData Method

Pascal

```
procedure replaceData(offset: Integer; count: Integer; const data: DOMString); safecall;
```

C++

```
__fastcall replaceData(int offset, int count, const DOMString data);
```

Description

This is replaceData, a member of class IDOMCharacterData.

80.2.3.2.7 IDOMCharacterData.set_data Method

Pascal

```
procedure set_data(const data: DOMString); safecall;
```

C++

```
__fastcall set_data(const DOMString data);
```

Description

This is set_data, a member of class IDOMCharacterData.

80.2.3.2.8 IDOMCharacterData.substringData Method

Pascal

```
function substringData(offset: Integer; count: Integer): DOMString; safecall;
```

C++

```
__fastcall DOMString substringData(int offset, int count);
```

Description

This is substringData, a member of class IDOMCharacterData.

80.2.3.3 IDOMCharacterData Properties

The properties of the IDOMCharacterData class are listed here.

Public Properties

	Name	Description
	data (see page 7529)	This is data, a member of class IDOMCharacterData.
R	length (see page 7529)	This is length, a member of class IDOMCharacterData.

80.2.3.3.1 IDOMCharacterData.data Property

Pascal

```
property data: DOMString;
```

C++

```
__property DOMString data;
```

Description

This is data, a member of class IDOMCharacterData.

80.2.3.3.2 IDOMCharacterData.length Property

Pascal

```
property length: Integer;
```

C++

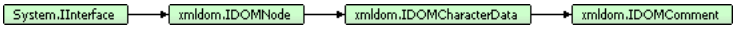
```
__property int length;
```

Description

This is length, a member of class IDOMCharacterData.

80.2.4 IDOMComment Interface

Class Hierarchy



Pascal

```
IDOMComment = interface(IDOMCharacterData);
```

C++

```
__interface IDOMComment : public IDOMCharacterData;
```

File

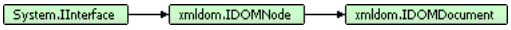
xmldom

Description

This is class xmldom.IDOMComment.

80.2.5 IDOMDocument Interface

Class Hierarchy



Pascal

```
IDOMDocument = interface(IDOMNode);
```

C++

```
__interface IDOMDocument : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMDocument.

80.2.5.1 IDOMDocument Members

The following tables list the members exposed by IDOMDocument.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	createAttribute (see page 7532)	This is createAttribute, a member of class IDOMDocument.
	createAttributeNS (see page 7533)	This is createAttributeNS, a member of class IDOMDocument.
	createCDATASection (see page 7533)	This is createCDATASection, a member of class IDOMDocument.
	createComment (see page 7533)	This is createComment, a member of class IDOMDocument.
	createDocumentFragment (see page 7533)	This is createDocumentFragment, a member of class IDOMDocument.
	createElement (see page 7533)	This is createElement, a member of class IDOMDocument.
	createElementNS (see page 7534)	This is createElementNS, a member of class IDOMDocument.
	createEntityReference (see page 7534)	This is createEntityReference, a member of class IDOMDocument.
	createProcessingInstruction (see page 7534)	This is createProcessingInstruction, a member of class IDOMDocument.
	createTextNode (see page 7534)	This is createTextNode, a member of class IDOMDocument.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_doctype (see page 7535)	This is get_doctype, a member of class IDOMDocument.
	get_documentElement (see page 7535)	This is get_documentElement, a member of class IDOMDocument.
	get_domImplementation (see page 7535)	This is get_domImplementation, a member of class IDOMDocument.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	getElementById (see page 7535)	This is getElementById, a member of class IDOMDocument.
	getElementsByTagName (see page 7536)	This is getElementsByTagName, a member of class IDOMDocument.
	getElementsByTagNameNS (see page 7536)	This is getElementsByTagNameNS, a member of class IDOMDocument.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	importNode (see page 7536)	This is importNode, a member of class IDOMDocument.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_documentElement (see page 7536)	This is set_documentElement, a member of class IDOMDocument.
	set_nodeValue (see page 7567)	Sets the value of the node.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	doctype (see page 7537)	This is doctype, a member of class IDOMDocument.
	documentElement (see page 7537)	This is documentElement, a member of class IDOMDocument.

R	domImplementation (see page 7537)	This is domImplementation, a member of class IDOMDocument.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.

80.2.5.2 IDOMDocument Methods

The methods of the IDOMDocument class are listed here.

Public Methods

	Name	Description
	createAttribute (see page 7532)	This is createAttribute, a member of class IDOMDocument.
	createAttributeNS (see page 7533)	This is createAttributeNS, a member of class IDOMDocument.
	createCDATASection (see page 7533)	This is createCDATASection, a member of class IDOMDocument.
	createComment (see page 7533)	This is createComment, a member of class IDOMDocument.
	createDocumentFragment (see page 7533)	This is createDocumentFragment, a member of class IDOMDocument.
	createElement (see page 7533)	This is createElement, a member of class IDOMDocument.
	createElementNS (see page 7534)	This is createElementNS, a member of class IDOMDocument.
	createEntityReference (see page 7534)	This is createEntityReference, a member of class IDOMDocument.
	createProcessingInstruction (see page 7534)	This is createProcessingInstruction, a member of class IDOMDocument.
	createTextNode (see page 7534)	This is createTextNode, a member of class IDOMDocument.
	get_doctype (see page 7535)	This is get_doctype, a member of class IDOMDocument.
	get_documentElement (see page 7535)	This is get_documentElement, a member of class IDOMDocument.
	get_domImplementation (see page 7535)	This is get_domImplementation, a member of class IDOMDocument.
	getElementById (see page 7535)	This is getElementById, a member of class IDOMDocument.
	getElementsByTagName (see page 7536)	This is getElementsByTagName, a member of class IDOMDocument.
	getElementsByTagNameNS (see page 7536)	This is getElementsByTagNameNS, a member of class IDOMDocument.
	importNode (see page 7536)	This is importNode, a member of class IDOMDocument.
	set_documentElement (see page 7536)	This is set_documentElement, a member of class IDOMDocument.

80.2.5.2.1 IDOMDocument.createAttribute Method

Pascal

```
function createAttribute(const name: DOMString): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr createAttribute(const DOMString name);
```

Description

This is createAttribute, a member of class IDOMDocument.

80.2.5.2.2 IDOMDocument.createAttributeNS Method

Pascal

```
function createAttributeNS(const namespaceURI: DOMString; const qualifiedName: DOMString):
IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr createAttributeNS(const DOMString namespaceURI, const DOMString
qualifiedName);
```

Description

This is createAttributeNS, a member of class IDOMDocument.

80.2.5.2.3 IDOMDocument.createCDATASection Method

Pascal

```
function createCDATASection(const data: DOMString): IDOMCDATASection; safecall;
```

C++

```
__fastcall IDOMCDATASection createCDATASection(const DOMString data);
```

Description

This is createCDATASection, a member of class IDOMDocument.

80.2.5.2.4 IDOMDocument.createComment Method

Pascal

```
function createComment(const data: DOMString): IDOMComment; safecall;
```

C++

```
__fastcall IDOMComment createComment(const DOMString data);
```

Description

This is createComment, a member of class IDOMDocument.

80.2.5.2.5 IDOMDocument.createDocumentFragment Method

Pascal

```
function createDocumentFragment: IDOMDocumentFragment; safecall;
```

C++

```
__fastcall IDOMDocumentFragment createDocumentFragment();
```

Description

This is createDocumentFragment, a member of class IDOMDocument.

80.2.5.2.6 IDOMDocument.createElement Method

Pascal

```
function createElement(const tagName: DOMString): IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement createElement(const DOMString tagName);
```

Description

This is createElement, a member of class IDOMDocument.

80.2.5.2.7 IDOMDocument.createElementNS Method

Pascal

```
function createElementNS(const namespaceURI: DOMString; const qualifiedName: DOMString):
IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement createElementNS(const DOMString namespaceURI, const DOMString
qualifiedName);
```

Description

This is createElementNS, a member of class IDOMDocument.

80.2.5.2.8 IDOMDocument.createEntityReference Method

Pascal

```
function createEntityReference(const name: DOMString): IDOMEntityReference; safecall;
```

C++

```
__fastcall IDOMEntityReference createEntityReference(const DOMString name);
```

Description

This is createEntityReference, a member of class IDOMDocument.

80.2.5.2.9 IDOMDocument.createProcessingInstruction Method

Pascal

```
function createProcessingInstruction(const target: DOMString; const data: DOMString):
IDOMProcessingInstruction; safecall;
```

C++

```
__fastcall IDOMProcessingInstruction createProcessingInstruction(const DOMString target, const
DOMString data);
```

Description

This is createProcessingInstruction, a member of class IDOMDocument.

80.2.5.2.10 IDOMDocument.createTextNode Method

Pascal

```
function createTextNode(const data: DOMString): IDOMText; safecall;
```

C++

```
__fastcall IDOMText createTextNode(const DOMString data);
```

Description

This is createTextNode, a member of class IDOMDocument.

80.2.5.2.11 IDOMDocument.get_doctype Method

Pascal

```
function get_doctype: IDOMDocumentType; safecall;
```

C++

```
__fastcall IDOMDocumentType get_doctype();
```

Description

This is get_doctype, a member of class IDOMDocument.

80.2.5.2.12 IDOMDocument.get_documentElement Method

Pascal

```
function get_documentElement: IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement get_documentElement();
```

Description

This is get_documentElement, a member of class IDOMDocument.

80.2.5.2.13 IDOMDocument.get_domImplementation Method

Pascal

```
function get_domImplementation: IDOMImplementation; safecall;
```

C++

```
__fastcall IDOMImplementation get_domImplementation();
```

Description

This is get_domImplementation, a member of class IDOMDocument.

80.2.5.2.14 IDOMDocument.getElementById Method

Pascal

```
function getElementById(const elementId: DOMString): IDOMElement; safecall;
```

C++

```
__fastcall IDOMElement getElementById(const DOMString elementId);
```

Description

This is getElementById, a member of class IDOMDocument.

80.2.5.2.15 IDOMDocument.getElementsByTagName Method

Pascal

```
function getElementsByTagName(const tagName: DOMString): IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagName(const DOMString tagName);
```

Description

This is getElementsByTagName, a member of class IDOMDocument.

80.2.5.2.16 IDOMDocument.getElementsByTagNameNS Method

Pascal

```
function getElementsByTagNameNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagNameNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getElementsByTagNameNS, a member of class IDOMDocument.

80.2.5.2.17 IDOMDocument.importNode Method

Pascal

```
function importNode(importedNode: IDOMNode; deep: WordBool): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode importNode(IDOMNode importedNode, WordBool deep);
```

Description

This is importNode, a member of class IDOMDocument.

80.2.5.2.18 IDOMDocument.set_documentElement Method

Pascal

```
procedure set_documentElement(const Element: IDOMElement); safecall;
```

C++

```
__fastcall set_documentElement(const IDOMElement Element);
```

Description

This is set_documentElement, a member of class IDOMDocument.

80.2.5.3 IDOMDocument Properties

The properties of the IDOMDocument class are listed here.

Public Properties

	Name	Description
	doctype (see page 7537)	This is doctype, a member of class IDOMDocument.
	documentElement (see page 7537)	This is documentElement, a member of class IDOMDocument.
	domImplementation (see page 7537)	This is domImplementation, a member of class IDOMDocument.

80.2.5.3.1 IDOMDocument.doctype Property

Pascal

```
property doctype: IDOMDocumentType;
```

C++

```
__property IDOMDocumentType doctype;
```

Description

This is doctype, a member of class IDOMDocument.

80.2.5.3.2 IDOMDocument.documentElement Property

Pascal

```
property documentElement: IDOMElement;
```

C++

```
__property IDOMElement documentElement;
```

Description

This is documentElement, a member of class IDOMDocument.

80.2.5.3.3 IDOMDocument.domImplementation Property

Pascal

```
property domImplementation: IDOMImplementation;
```

C++

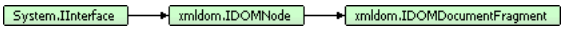
```
__property IDOMImplementation domImplementation;
```

Description

This is domImplementation, a member of class IDOMDocument.

80.2.6 IDOMDocumentFragment Interface

Class Hierarchy



Pascal

```
IDOMDocumentFragment = interface(IDOMNode);
```

C++

```
__interface IDOMDocumentFragment : public IDOMNode;
```

File

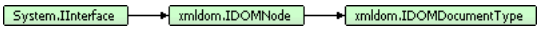
xmldom

Description

This is class xmldom.IDOMDocumentFragment.

80.2.7 IDOMDocumentType Interface

Class Hierarchy



Pascal

```
IDOMDocumentType = interface(IDOMNode);
```

C++

```
__interface IDOMDocumentType : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMDocumentType.

80.2.7.1 IDOMDocumentType Members

The following tables list the members exposed by IDOMDocumentType.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_entities (see page 7540)	This is get_entities, a member of class IDOMDocumentType.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_internalSubset (see page 7540)	This is get_internalSubset, a member of class IDOMDocumentType.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_name (see page 7540)	This is get_name, a member of class IDOMDocumentType.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_notations (see page 7540)	This is get_notations, a member of class IDOMDocumentType.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.

	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	get_publicId (see page 7540)	This is get_publicId, a member of class IDOMDocumentType.
	get_systemId (see page 7541)	This is get_systemId, a member of class IDOMDocumentType.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	entities (see page 7541)	This is entities, a member of class IDOMDocumentType.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	internalSubset (see page 7541)	This is internalSubset, a member of class IDOMDocumentType.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	name (see page 7542)	This is name, a member of class IDOMDocumentType.
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	notations (see page 7542)	This is notations, a member of class IDOMDocumentType.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.
R	publicId (see page 7542)	This is publicId, a member of class IDOMDocumentType.
R	systemId (see page 7542)	This is systemId, a member of class IDOMDocumentType.

80.2.7.2 IDOMDocumentType Methods

The methods of the IDOMDocumentType class are listed here.

Public Methods

	Name	Description
	get_entities (see page 7540)	This is get_entities, a member of class IDOMDocumentType.
	get_internalSubset (see page 7540)	This is get_internalSubset, a member of class IDOMDocumentType.
	get_name (see page 7540)	This is get_name, a member of class IDOMDocumentType.
	get_notations (see page 7540)	This is get_notations, a member of class IDOMDocumentType.
	get_publicId (see page 7540)	This is get_publicId, a member of class IDOMDocumentType.
	get_systemId (see page 7541)	This is get_systemId, a member of class IDOMDocumentType.

80.2.7.2.1 IDOMDocumentType.get_entities Method

Pascal

```
function get_entities: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_entities();
```

Description

This is get_entities, a member of class IDOMDocumentType.

80.2.7.2.2 IDOMDocumentType.get_internalSubset Method

Pascal

```
function get_internalSubset: DOMString; safecall;
```

C++

```
__fastcall DOMString get_internalSubset();
```

Description

This is get_internalSubset, a member of class IDOMDocumentType.

80.2.7.2.3 IDOMDocumentType.get_name Method

Pascal

```
function get_name: DOMString; safecall;
```

C++

```
__fastcall DOMString get_name();
```

Description

This is get_name, a member of class IDOMDocumentType.

80.2.7.2.4 IDOMDocumentType.get_notations Method

Pascal

```
function get_notations: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_notations();
```

Description

This is get_notations, a member of class IDOMDocumentType.

80.2.7.2.5 IDOMDocumentType.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMDocumentType.

80.2.7.2.6 IDOMDocumentType.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMDocumentType.

80.2.7.3 IDOMDocumentType Properties

The properties of the IDOMDocumentType class are listed here.

Public Properties

	Name	Description
	entities (see page 7541)	This is entities, a member of class IDOMDocumentType.
	internalSubset (see page 7541)	This is internalSubset, a member of class IDOMDocumentType.
	name (see page 7542)	This is name, a member of class IDOMDocumentType.
	notations (see page 7542)	This is notations, a member of class IDOMDocumentType.
	publicId (see page 7542)	This is publicId, a member of class IDOMDocumentType.
	systemId (see page 7542)	This is systemId, a member of class IDOMDocumentType.

80.2.7.3.1 IDOMDocumentType.entities Property

Pascal

```
property entities: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap entities;
```

Description

This is entities, a member of class IDOMDocumentType.

80.2.7.3.2 IDOMDocumentType.internalSubset Property

Pascal

```
property internalSubset: DOMString;
```

C++

```
__property DOMString internalSubset;
```

Description

This is internalSubset, a member of class IDOMDocumentType.

80.2.7.3.3 IDOMDocumentType.name Property

Pascal

```
property name: DOMString;
```

C++

```
__property DOMString name;
```

Description

This is name, a member of class IDOMDocumentType.

80.2.7.3.4 IDOMDocumentType.notations Property

Pascal

```
property notations: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap notations;
```

Description

This is notations, a member of class IDOMDocumentType.

80.2.7.3.5 IDOMDocumentType.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMDocumentType.

80.2.7.3.6 IDOMDocumentType.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

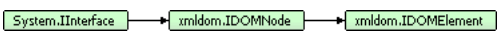
```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMDocumentType.

80.2.8 IDOMElement Interface

Class Hierarchy



Pascal

```
IDOMElement = interface(IDOMNode);
```

C++

```
__interface IDOMElement : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMElement.

80.2.8.1 IDOMElement Members

The following tables list the members exposed by IDOMElement.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	get_tagName (see page 7545)	This is get_tagName, a member of class IDOMElement.
	getAttribute (see page 7545)	This is getAttribute, a member of class IDOMElement.
	getAttributeNode (see page 7545)	This is getAttributeNode, a member of class IDOMElement.
	getAttributeNodeNS (see page 7545)	This is getAttributeNodeNS, a member of class IDOMElement.
	getAttributeNS (see page 7545)	This is getAttributeNS, a member of class IDOMElement.
	getElementsByTagName (see page 7546)	This is getElementsByTagName, a member of class IDOMElement.
	getElementsByTagNameNS (see page 7546)	This is getElementsByTagNameNS, a member of class IDOMElement.
	hasAttribute (see page 7546)	This is hasAttribute, a member of class IDOMElement.
	hasAttributeNS (see page 7546)	This is hasAttributeNS, a member of class IDOMElement.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeAttribute (see page 7547)	This is removeAttribute, a member of class IDOMElement.
	removeAttributeNode (see page 7547)	This is removeAttributeNode, a member of class IDOMElement.
	removeAttributeNS (see page 7547)	This is removeAttributeNS, a member of class IDOMElement.

	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	setAttribute (see page 7547)	This is setAttribute, a member of class IDOMElement.
	setAttributeNode (see page 7547)	This is setAttributeNode, a member of class IDOMElement.
	setAttributeNodeNS (see page 7548)	This is setAttributeNodeNS, a member of class IDOMElement.
	setAttributeNS (see page 7548)	This is setAttributeNS, a member of class IDOMElement.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.
R	tagName (see page 7548)	This is tagName, a member of class IDOMElement.

80.2.8.2 IDOMElement Methods

The methods of the IDOMElement class are listed here.

Public Methods

	Name	Description
	get_tagName (see page 7545)	This is get_tagName, a member of class IDOMElement.
	getAttribute (see page 7545)	This is getAttribute, a member of class IDOMElement.
	getAttributeNode (see page 7545)	This is getAttributeNode, a member of class IDOMElement.
	getAttributeNodeNS (see page 7545)	This is getAttributeNodeNS, a member of class IDOMElement.
	getAttributeNS (see page 7545)	This is getAttributeNS, a member of class IDOMElement.
	getElementsByTagName (see page 7546)	This is getElementsByTagName, a member of class IDOMElement.
	getElementsByTagNameNS (see page 7546)	This is getElementsByTagNameNS, a member of class IDOMElement.
	hasAttribute (see page 7546)	This is hasAttribute, a member of class IDOMElement.
	hasAttributeNS (see page 7546)	This is hasAttributeNS, a member of class IDOMElement.
	removeAttribute (see page 7547)	This is removeAttribute, a member of class IDOMElement.
	removeAttributeNode (see page 7547)	This is removeAttributeNode, a member of class IDOMElement.
	removeAttributeNS (see page 7547)	This is removeAttributeNS, a member of class IDOMElement.
	setAttribute (see page 7547)	This is setAttribute, a member of class IDOMElement.
	setAttributeNode (see page 7547)	This is setAttributeNode, a member of class IDOMElement.
	setAttributeNodeNS (see page 7548)	This is setAttributeNodeNS, a member of class IDOMElement.
	setAttributeNS (see page 7548)	This is setAttributeNS, a member of class IDOMElement.

80.2.8.2.1 IDOMElement.get_tagName Method

Pascal

```
function get_tagName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_tagName();
```

Description

This is get_tagName, a member of class IDOMElement.

80.2.8.2.2 IDOMElement.getAttribute Method

Pascal

```
function getAttribute(const name: DOMString): DOMString; safecall;
```

C++

```
__fastcall DOMString getAttribute(const DOMString name);
```

Description

This is getAttribute, a member of class IDOMElement.

80.2.8.2.3 IDOMElement.getAttributeNode Method

Pascal

```
function getAttributeNode(const name: DOMString): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr getAttributeNode(const DOMString name);
```

Description

This is getAttributeNode, a member of class IDOMElement.

80.2.8.2.4 IDOMElement.getAttributeNodeNS Method

Pascal

```
function getAttributeNodeNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr getAttributeNodeNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getAttributeNodeNS, a member of class IDOMElement.

80.2.8.2.5 IDOMElement.getAttributeNS Method

Pascal

```
function getAttributeNS(const namespaceURI: DOMString; const localName: DOMString): DOMString;
safecall;
```

C++

```
__fastcall DOMString getAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is getAttributeNS, a member of class IDOMElement.

80.2.8.2.6 IDOMElement.getElementsByTagName Method

Pascal

```
function getElementsByTagName(const name: DOMString): IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagName(const DOMString name);
```

Description

This is getElementsByTagName, a member of class IDOMElement.

80.2.8.2.7 IDOMElement.getElementsByTagNameNS Method

Pascal

```
function getElementsByTagNameNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList getElementsByTagNameNS(const DOMString namespaceURI, const DOMString
localName);
```

Description

This is getElementsByTagNameNS, a member of class IDOMElement.

80.2.8.2.8 IDOMElement.hasAttribute Method

Pascal

```
function hasAttribute(const name: DOMString): WordBool; safecall;
```

C++

```
__fastcall WordBool hasAttribute(const DOMString name);
```

Description

This is hasAttribute, a member of class IDOMElement.

80.2.8.2.9 IDOMElement.hasAttributeNS Method

Pascal

```
function hasAttributeNS(const namespaceURI: DOMString; const localName: DOMString): WordBool;
safecall;
```

C++

```
__fastcall WordBool hasAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is hasAttributeNS, a member of class IDOMElement.

80.2.8.2.10 IDOMElement.removeAttribute Method

Pascal

```
procedure removeAttribute(const name: DOMString); safecall;
```

C++

```
__fastcall removeAttribute(const DOMString name);
```

Description

This is removeAttribute, a member of class IDOMElement.

80.2.8.2.11 IDOMElement.removeAttributeNode Method

Pascal

```
function removeAttributeNode(const oldAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr removeAttributeNode(const IDOMAttr oldAttr);
```

Description

This is removeAttributeNode, a member of class IDOMElement.

80.2.8.2.12 IDOMElement.removeAttributeNS Method

Pascal

```
procedure removeAttributeNS(const namespaceURI: DOMString; const localName: DOMString);
safecall;
```

C++

```
__fastcall removeAttributeNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is removeAttributeNS, a member of class IDOMElement.

80.2.8.2.13 IDOMElement.setAttribute Method

Pascal

```
procedure setAttribute(const name: DOMString; const value: DOMString); safecall;
```

C++

```
__fastcall setAttribute(const DOMString name, const DOMString value);
```

Description

This is setAttribute, a member of class IDOMElement.

80.2.8.2.14 IDOMElement.setAttributeNode Method

Pascal

```
function setAttributeNode(const newAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr setAttributeNode(const IDOMAttr newAttr);
```

Description

This is setAttributeNode, a member of class IDOMElement.

80.2.8.2.15 IDOMElement.setAttributeNodeNS Method

Pascal

```
function setAttributeNodeNS(const newAttr: IDOMAttr): IDOMAttr; safecall;
```

C++

```
__fastcall IDOMAttr setAttributeNodeNS(const IDOMAttr newAttr);
```

Description

This is setAttributeNodeNS, a member of class IDOMElement.

80.2.8.2.16 IDOMElement.setAttributeNS Method

Pascal

```
procedure setAttributeNS(const namespaceURI: DOMString; const qulifiedName: DOMString; const
value: DOMString); safecall;
```

C++

```
__fastcall setAttributeNS(const DOMString namespaceURI, const DOMString qulifiedName, const
DOMString value);
```

Description

This is setAttributeNS, a member of class IDOMElement.

80.2.8.3 IDOMElement Properties

The properties of the IDOMElement class are listed here.

Public Properties

	Name	Description
	tagName (see page 7548)	This is tagName, a member of class IDOMElement.

80.2.8.3.1 IDOMElement.tagName Property

Pascal

```
property tagName: DOMString;
```

C++

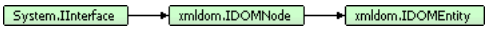
```
__property DOMString tagName;
```

Description

This is tagName, a member of class IDOMElement.

80.2.9 IDOMEntity Interface

Class Hierarchy



Pascal

```
IDOMEntity = interface(IDOMNode);
```

C++

```
__interface IDOMEntity : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMEntity.

80.2.9.1 IDOMEntity Members

The following tables list the members exposed by IDOMEntity.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_notationName (see page 7550)	This is get_notationName, a member of class IDOMEntity.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	get_publicId (see page 7550)	This is get_publicId, a member of class IDOMEntity.
	get_systemId (see page 7551)	This is get_systemId, a member of class IDOMEntity.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.

	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	notationName (see page 7551)	This is notationName, a member of class IDOMEntity.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.
R	publicId (see page 7551)	This is publicId, a member of class IDOMEntity.
R	systemId (see page 7551)	This is systemId, a member of class IDOMEntity.

80.2.9.2 IDOMEntity Methods

The methods of the IDOMEntity class are listed here.

Public Methods

	Name	Description
	get_notationName (see page 7550)	This is get_notationName, a member of class IDOMEntity.
	get_publicId (see page 7550)	This is get_publicId, a member of class IDOMEntity.
	get_systemId (see page 7551)	This is get_systemId, a member of class IDOMEntity.

80.2.9.2.1 IDOMEntity.get_notationName Method

Pascal

```
function get_notationName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_notationName();
```

Description

This is get_notationName, a member of class IDOMEntity.

80.2.9.2.2 IDOMEntity.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMEntity.

80.2.9.2.3 IDOMEntity.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMEntity.

80.2.9.3 IDOMEntity Properties

The properties of the IDOMEntity class are listed here.

Public Properties

	Name	Description
	notationName (see page 7551)	This is notationName, a member of class IDOMEntity.
	publicId (see page 7551)	This is publicId, a member of class IDOMEntity.
	systemId (see page 7551)	This is systemId, a member of class IDOMEntity.

80.2.9.3.1 IDOMEntity.notationName Property

Pascal

```
property notationName: DOMString;
```

C++

```
__property DOMString notationName;
```

Description

This is notationName, a member of class IDOMEntity.

80.2.9.3.2 IDOMEntity.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMEntity.

80.2.9.3.3 IDOMEntity.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

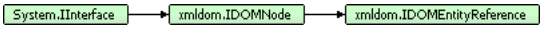
```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMEntity.

80.2.10 IDOMEntityReference Interface

Class Hierarchy



Pascal

```
IDOMEntityReference = interface(IDOMNode);
```

C++

```
__interface IDOMEntityReference : public IDOMNode;
```

File

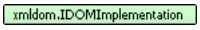
xmldom

Description

This is class xmldom.IDOMEntityReference.

80.2.11 IDOMImplementation Interface

Class Hierarchy



Pascal

```
IDOMImplementation = interface;
```

C++

```
__interface IDOMImplementation;
```

File

xmldom

Description

This is class xmldom.IDOMImplementation.

80.2.11.1 IDOMImplementation Members

The following tables list the members exposed by IDOMImplementation.

Public Methods

	Name	Description
	createDocument (see page 7553)	This is createDocument, a member of class IDOMImplementation.
	createDocumentType (see page 7553)	This is createDocumentType, a member of class IDOMImplementation.
	hasFeature (see page 7553)	This is hasFeature, a member of class IDOMImplementation.

80.2.11.2 IDOMImplementation Methods

The methods of the IDOMImplementation class are listed here.

Public Methods

	Name	Description
	createDocument (see page 7553)	This is createDocument, a member of class IDOMImplementation.
	createDocumentType (see page 7553)	This is createDocumentType, a member of class IDOMImplementation.
	hasFeature (see page 7553)	This is hasFeature, a member of class IDOMImplementation.

80.2.11.2.1 IDOMImplementation.createDocument Method

Pascal

```
function createDocument(const namespaceURI: DOMString; const qualifiedName: DOMString;
doctype: IDOMDocumentType): IDOMDocument; safecall;
```

C++

```
__fastcall IDOMDocument createDocument(const DOMString namespaceURI, const DOMString
qualifiedName, IDOMDocumentType doctype);
```

Description

This is createDocument, a member of class IDOMImplementation.

80.2.11.2.2 IDOMImplementation.createDocumentType Method

Pascal

```
function createDocumentType(const qualifiedName: DOMString; const publicId: DOMString; const
systemId: DOMString): IDOMDocumentType; safecall;
```

C++

```
__fastcall IDOMDocumentType createDocumentType(const DOMString qualifiedName, const DOMString
publicId, const DOMString systemId);
```

Description

This is createDocumentType, a member of class IDOMImplementation.

80.2.11.2.3 IDOMImplementation.hasFeature Method

Pascal

```
function hasFeature(const feature: DOMString; const version: DOMString): WordBool;
```

C++

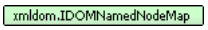
```
__fastcall WordBool hasFeature(const DOMString feature, const DOMString version);
```

Description

This is hasFeature, a member of class IDOMImplementation.

80.2.12 IDOMNamedNodeMap Interface

Class Hierarchy



Pascal

```
IDOMNamedNodeMap = interface;
```

C++

```
__interface IDOMNamedNodeMap;
```

File

xmldom

Description

This is class xmldom.IDOMNamedNodeMap.

80.2.12.1 IDOMNamedNodeMap Members

The following tables list the members exposed by IDOMNamedNodeMap.

Public Methods

	Name	Description
	get_item (see page 7555)	This is get_item, a member of class IDOMNamedNodeMap.
	get_length (see page 7555)	This is get_length, a member of class IDOMNamedNodeMap.
	getNamedItem (see page 7555)	This is getNamedItem, a member of class IDOMNamedNodeMap.
	getNamedItemNS (see page 7555)	This is getNamedItemNS, a member of class IDOMNamedNodeMap.
	removeNamedItem (see page 7556)	This is removeNamedItem, a member of class IDOMNamedNodeMap.
	removeNamedItemNS (see page 7556)	This is removeNamedItemNS, a member of class IDOMNamedNodeMap.
	setNamedItem (see page 7556)	This is setNamedItem, a member of class IDOMNamedNodeMap.
	setNamedItemNS (see page 7556)	This is setNamedItemNS, a member of class IDOMNamedNodeMap.

Public Properties

	Name	Description
R	item (see page 7557)	This is item, a member of class IDOMNamedNodeMap.
R	length (see page 7557)	This is length, a member of class IDOMNamedNodeMap.

80.2.12.2 IDOMNamedNodeMap Methods

The methods of the IDOMNamedNodeMap class are listed here.

Public Methods

	Name	Description
	get_item (see page 7555)	This is get_item, a member of class IDOMNamedNodeMap.
	get_length (see page 7555)	This is get_length, a member of class IDOMNamedNodeMap.
	getNamedItem (see page 7555)	This is getNamedItem, a member of class IDOMNamedNodeMap.
	getNamedItemNS (see page 7555)	This is getNamedItemNS, a member of class IDOMNamedNodeMap.
	removeNamedItem (see page 7556)	This is removeNamedItem, a member of class IDOMNamedNodeMap.
	removeNamedItemNS (see page 7556)	This is removeNamedItemNS, a member of class IDOMNamedNodeMap.

	setNamedItem (see page 7556)	This is setNamedItem, a member of class IDOMNamedNodeMap.
	setNamedItemNS (see page 7556)	This is setNamedItemNS, a member of class IDOMNamedNodeMap.

80.2.12.2.1 IDOMNamedNodeMap.get_item Method

Pascal

```
function get_item(index: Integer): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_item(int index);
```

Description

This is get_item, a member of class IDOMNamedNodeMap.

80.2.12.2.2 IDOMNamedNodeMap.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMNamedNodeMap.

80.2.12.2.3 IDOMNamedNodeMap.getNamedItem Method

Pascal

```
function getNamedItem(const name: DOMString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode getNamedItem(const DOMString name);
```

Description

This is getNamedItem, a member of class IDOMNamedNodeMap.

80.2.12.2.4 IDOMNamedNodeMap.getNamedItemNS Method

Pascal

```
function getNamedItemNS(const namespaceURI: DOMString; const localName: DOMString): IDOMNode;
safecall;
```

C++

```
__fastcall IDOMNode getNamedItemNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is getNamedItemNS, a member of class IDOMNamedNodeMap.

80.2.12.2.5 IDOMNamedNodeMap.removeNamedItem Method

Pascal

```
function removeNamedItem(const name: DOMString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeNamedItem(const DOMString name);
```

Description

This is removeNamedItem, a member of class IDOMNamedNodeMap.

80.2.12.2.6 IDOMNamedNodeMap.removeNamedItemNS Method

Pascal

```
function removeNamedItemNS(const namespaceURI: DOMString; const localName: DOMString):
IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeNamedItemNS(const DOMString namespaceURI, const DOMString localName);
```

Description

This is removeNamedItemNS, a member of class IDOMNamedNodeMap.

80.2.12.2.7 IDOMNamedNodeMap.setNamedItem Method

Pascal

```
function setNamedItem(const arg: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode setNamedItem(const IDOMNode arg);
```

Description

This is setNamedItem, a member of class IDOMNamedNodeMap.

80.2.12.2.8 IDOMNamedNodeMap.setNamedItemNS Method

Pascal

```
function setNamedItemNS(const arg: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode setNamedItemNS(const IDOMNode arg);
```

Description

This is setNamedItemNS, a member of class IDOMNamedNodeMap.

80.2.12.3 IDOMNamedNodeMap Properties

The properties of the IDOMNamedNodeMap class are listed here.

Public Properties

	Name	Description
	item (see page 7557)	This is item, a member of class IDOMNamedNodeMap.
	length (see page 7557)	This is length, a member of class IDOMNamedNodeMap.

80.2.12.3.1 IDOMNamedNodeMap.item Property

Pascal

```
property item [index: Integer]: IDOMNode;
```

C++

```
__property IDOMNode item[int index];
```

Description

This is item, a member of class IDOMNamedNodeMap.

80.2.12.3.2 IDOMNamedNodeMap.length Property

Pascal

```
property length: Integer;
```

C++

```
__property int length;
```

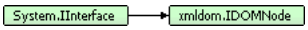
Description

This is length, a member of class IDOMNamedNodeMap.

80.2.13 IDOMNode Interface

IDOMNode is the DOM interface for a node in an XML document.

Class Hierarchy



Pascal

```
IDOMNode = interface(IInterface);
```

C++

```
__interface IDOMNode : public IInterface;
```

File

xmldom

Description

IDOMNode is a vendor-neutral version of the DOM interface for a node in an XML document. It conforms to the W3CXML DOM level 2 specification. Use IDOMNode to work with a node in an XML document when you are using standard DOM interfaces rather than the IXMLNode interface. (IXMLNode is the node interface used by TXMLDocument document.)

IDOMNode represents an element in an XML document that has been parsed by a DOM parser. It includes properties and methods for navigating the node hierarchy of an XML document, for accessing the attributes (see page 7568) of a node, and

for drilling down to obtain the node's value.

For more information about IDOMNode and the other standard DOM interfaces, see the DOM specification on the W3 website or the documentation provided by your DOM vendor.

Note: C++ method declarations that use IDOMNode use the _di_IDOMNode type instead. This type is a DelphiInterface (see page 3256) wrapper around the IDOMNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IDOMNode >  _di_IDOMNode;
```

See Also

IXMLNode (see page 7669)

TXMLDocument (see page 7347)

CreateDOMNode (see page 7488)

80.2.13.1 IDOMNode Members

The following tables list the members exposed by IDOMNode.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.

	childNodes (see page 7568)	Represents the child nodes of the node.
	firstChild (see page 7569)	Represents the first child node of this node.
	lastChild (see page 7569)	Represents the last child node of this node.
	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
	nextSibling (see page 7571)	Represents the next child of this node's parent.
	nodeName (see page 7572)	Indicates the name of the node.
	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
	prefix (see page 7576)	Indicates the namespace prefix for the node.
	previousSibling (see page 7576)	Represents the preceding child of this node's parent.

80.2.13.2 IDOMNode Methods

The methods of the IDOMNode class are listed here.

Public Methods

	Name	Description
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

80.2.13.2.1 IDOMNode.appendChild Method

Adds a new child node to this node.

Pascal

```
function appendChild(const newChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode appendChild(const IDOMNode newChild);
```

Description

Use appendChild to add a child node to the end of this node's list of children. If the new node is already in the document's node hierarchy, it is first removed from its parent node. If this node is read-only, newChild was created using a different document, or if newChild is of a nodeType (see page 7572) that is not permitted as a child of this node, appendChild raises an exception.

newChild is the child node to add. It can be a node that currently has another parent node, or a new node created using the methods of the document interface. If it is of nodeType (see page 7572) DOCUMENT_FRAGMENT_NODE (see page 7625), the entire contents of the document fragment are moved into this node's childNodes (see page 7568) property list.

appendChild returns the interface for the newly added child node.

See Also

insertBefore (see page 7565)

childNodes (see page 7568)

replaceChild (see page 7566)

removeChild (see page 7566)

nodeType (see page 7572)

ownerDocument (see page 7575)

80.2.13.2.2 IDOMNode.cloneNode Method

Returns a duplicate of this node.

Pascal

```
function cloneNode(deep: WordBool): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode cloneNode(WordBool deep);
```

Description

Use cloneNode to obtain a copy of this node. The cloneNode method returns a copy of this node that has no parent. Any attributes (see page 7568) are copied as well, including any attribute nodes that represent default values.

deep indicates whether the child nodes of this node should be cloned as well. For example, if an element node contains any text, cloneNode only copies then text when deep is true, because the text is contained in a child node of type TEXT_NODE (see page 7636).

Note: When cloning an attribute node directly (as opposed to copying the attributes (see page 7568) when cloning an element node), the returned node's specified property is always true, even if this node does not represent a specified attribute.

Note: Cloning an immutable subtree produces a mutable copy, but the children of an entity-reference node are read-only.

Note: DOM implementations may differ in the way they clone nodes of type DOCUMENT_NODE (see page 7626), DOCUMENT_TYPE_NODE (see page 7626), ENTITY_NODE (see page 7627), and NOTATION_NODE (see page 7632).

See Also

insertBefore (see page 7565)

childNodes (see page 7568)

replaceChild (see page 7566)

removeChild (see page 7566)

nodeType (see page 7572)

ownerDocument (see page 7575)

80.2.13.2.3 IDOMNode.get_attributes Method

Returns the value of the get_attributes property.

Pascal

```
function get_attributes: IDOMNamedNodeMap; safecall;
```

C++

```
__fastcall IDOMNamedNodeMap get_attributes();
```

Description

get_get_attributes is the read implementation of the get_attributes property.

80.2.13.2.4 IDOMNode.get_childNodes Method

Returns the value of the get_childNodes property.

Pascal

```
function get_childNodes: IDOMNodeList; safecall;
```

C++

```
__fastcall IDOMNodeList get_childNodes();
```

Description

get_get_childNodes is the read implementation of the get_childNodes property.

80.2.13.2.5 IDOMNode.get_firstChild Method

Returns the value of the get_firstChild property.

Pascal

```
function get_firstChild: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_firstChild();
```

Description

get_get_firstChild is the read implementation of the get_firstChild property.

80.2.13.2.6 IDOMNode.get_lastChild Method

Returns the value of the get_lastChild property.

Pascal

```
function get_lastChild: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_lastChild();
```

Description

get_get_lastChild is the read implementation of the get_lastChild property.

80.2.13.2.7 IDOMNode.get_localName Method

Returns the value of the get_localName property.

Pascal

```
function get_localName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_localName();
```

Description

get_get_localName is the read implementation of the get_localName property.

80.2.13.2.8 IDOMNode.get_namespaceURI Method

Returns the value of the get_namespaceURI property.

Pascal

```
function get_namespaceURI: DOMString; safecall;
```

C++

```
__fastcall DOMString get_namespaceURI();
```

Description

get_get_namespaceURI is the read implementation of the get_namespaceURI property.

80.2.13.2.9 IDOMNode.get_nextSibling Method

Returns the value of the get_nextSibling property.

Pascal

```
function get_nextSibling: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_nextSibling();
```

Description

get_get_nextSibling is the read implementation of the get_nextSibling property.

80.2.13.2.10 IDOMNode.get_nodeName Method

Returns the value of the get_nodeName property.

Pascal

```
function get_nodeName: DOMString; safecall;
```

C++

```
__fastcall DOMString get_nodeName();
```

Description

get_get_nodeName is the read implementation of the get_nodeName property.

80.2.13.2.11 IDOMNode.get_nodeType Method

Returns the value of the get_nodeType property.

Pascal

```
function get_nodeType: DOMNodeType; safecall;
```

C++

```
__fastcall DOMNodeType get_nodeType();
```

Description

get_get_nodeType is the read implementation of the get_nodeType property.

80.2.13.2.12 IDOMNode.get_nodeValue Method

Returns the value of the get_nodeValue property.

Pascal

```
function get_nodeValue: DOMString; safecall;
```

C++

```
__fastcall DOMString get_nodeValue();
```

Description

get_get_nodeValue is the read implementation of the get_nodeValue property.

80.2.13.2.13 IDOMNode.get_ownerDocument Method

Returns the value of the get_ownerDocument property.

Pascal

```
function get_ownerDocument: IDOMDocument; safecall;
```

C++

```
__fastcall IDOMDocument get_ownerDocument();
```

Description

get_get_ownerDocument is the read implementation of the get_ownerDocument property.

80.2.13.2.14 IDOMNode.get_parentNode Method

Returns the value of the get_parentNode property.

Pascal

```
function get_parentNode: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_parentNode();
```

Description

get_get_parentNode is the read implementation of the get_parentNode property.

80.2.13.2.15 IDOMNode.get_prefix Method

Returns the value of the get_prefix property.

Pascal

```
function get_prefix: DOMString; safecall;
```

C++

```
__fastcall DOMString get_prefix();
```

Description

get_get_prefix is the read implementation of the get_prefix property.

80.2.13.2.16 IDOMNode.get_previousSibling Method

Returns the value of the get_previousSibling property.

Pascal

```
function get_previousSibling: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_previousSibling();
```

Description

get_get_previousSibling is the read implementation of the get_previousSibling property.

80.2.13.2.17 IDOMNode.hasChildNodes Method

Indicates whether the node has any child nodes.

Pascal

```
function hasChildNodes: WordBool; safecall;
```

C++

```
__fastcall WordBool hasChildNodes();
```

Description

hasChildNodes returns true if this node has any child nodes, false if it has no children.

See Also

childNodes (see page 7568)

firstChild (see page 7569)

lastChild (see page 7569)

80.2.13.2.18 IDOMNode.insertBefore Method

Inserts a new child node before a specified child of the node.

Pascal

```
function insertBefore(const newChild: IDOMNode; const refChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode insertBefore(const IDOMNode newChild, const IDOMNode refChild);
```

Description

Use insertBefore to add a child node at a specified position in this node's list of children. If the new node is already in the document's node hierarchy, it is first removed from its parent node. If this node is read-only, refChild is a node that is not in this node's child list, newChild was created using a different document, or if newChild is of a nodeType (see page 7572) that is not permitted as a child of this node, insertBefore raises an exception.

newChild is the child node to add. It can be a node that currently has another parent node, or a new node created using the methods of the document interface. If it is of nodeType (see page 7572) DOCUMENT_FRAGMENT_NODE (see page 7625), the entire contents of the document fragment are moved into this node's childNodes (see page 7568) property list.

refChild is a child of this node before which to insert the new node. If refChild is nil (Delphi) or NULL (C++), the new node is added to the end of this node's child list.

insertBefore returns the interface for the newly added child node.

See Also

appendChild (see page 7559)

childNodes (see page 7568)

replaceChild (see page 7566)

removeChild (see page 7566)

nodeType (see page 7572)

ownerDocument (see page 7575)

80.2.13.2.19 IDOMNode.normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure normalize; safecall;
```

C++

```
__fastcall normalize();
```

Description

Call normalize to consolidate adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes (see page 7568), only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes. This is useful for operations (such as XPointer lookups) that depend on a particular structure in the node hierarchy.

See Also

childNodes (see page 7568)

nodeType (see page 7572)

attributes (see page 7568)

80.2.13.2.20 IDOMNode.removeChild Method

Removes a child node from this node.

Pascal

```
function removeChild(const childNode: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode removeChild(const IDOMNode childNode);
```

Description

removeChild removes a specified child node from this node, and returns its interface. If this node is read-only, or if the specified child node is not a child of this node, removeChild raises an exception.

childNode the node to remove.

See Also

appendChild (see page 7559)

childNodes (see page 7568)

replaceChild (see page 7566)

80.2.13.2.21 IDOMNode.replaceChild Method

Replaces one of this node's children with a specified node.

Pascal

```
function replaceChild(const newChild: IDOMNode; const oldChild: IDOMNode): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode replaceChild(const IDOMNode newChild, const IDOMNode oldChild);
```

Description

Use replaceChild to remove a child of this node and replace it with the node specified by newChild. If the new node is already in the document's node hierarchy, it is first removed from its parent node.

If this node is read-only, refChild is a node that is not in this node's child list, newChild was created using a different document, or if newChild is of a nodeType (see page 7572) that is not permitted as a child of this node, replaceChild raises an exception.

newChild is the child node that replaces oldChild. It can be a node that currently has another parent node, or a new node created using the methods of the document interface. If it is of nodeType (see page 7572) DOCUMENT_FRAGMENT_NODE (see page 7625), the entire contents of the document fragment are moved into this node's childNodes (see page 7568) property list.

refChild is a child that newChild replaces. If refChild is nil (Delphi) or NULL (C++), the new node is added to the end of this node's child list.

replaceChild returns the interface for refChild (the replaced node).

See Also

appendChild (see page 7559)

childNodes (see page 7568)

insertBefore (see page 7565)

80.2.13.2.22 IDOMNode.set_nodeValue Method

Sets the value of the node.

Pascal

```
procedure set_nodeValue(value: DOMString); safecall;
```

C++

```
__fastcall set_nodeValue(DOMString value);
```

Description

set_set_nodeValue is the write implementation of the set_nodeValue property.

value is the new value for the node.

80.2.13.2.23 IDOMNode.supports Method

Indicates whether the node supports a specified feature.

Pascal

```
function supports(const feature: DOMString; const version: DOMString): WordBool;
```

C++

```
__fastcall WordBool supports(const DOMString feature, const DOMString version);
```

Description

supports returns true if the DOM implementation implements the specified feature and this node supports that feature. Otherwise, supports returns false.

feature is the name of the feature in question.

version is the version of the feature. If no version is specified, supports returns true if any version of the feature is supported.

80.2.13.3 IDOMNode Properties

The properties of the IDOMNode class are listed here.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.

R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.

80.2.13.3.1 IDOMNode.attributes Property

Represents the attributes of an element node.

Pascal

```
property attributes: IDOMNamedNodeMap;
```

C++

```
__property IDOMNamedNodeMap attributes;
```

Description

Use attributes to access the attribute nodes for this node. The IDOMNamedNodeMap (see page 7554) interface provides access to a collection of DOM nodes that can be identified by name.

Attributes modify a node in an XML document. For example, given the node for

```
<Movie genre=scifi title="Star Wars">
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

DOM parsers represent each attribute as a separate node, which is listed by IDOMNamedNodeMap (see page 7554).

Only element nodes have attributes, and not every element node has them. If the node does not have any attributes, the value of attributes is nil (Delphi) or NULL (C++).

Note: Attributes are not considered child nodes of a node and can't be accessed using the childNodes (see page 7568) property.

See Also

childNodes (see page 7568)

80.2.13.3.2 IDOMNode.childNodes Property

Represents the child nodes of the node.

Pascal

```
property childNodes: IDOMNodeList;
```

C++

```
__property IDOMNodeList childNodes;
```

Description

The childNodes property lists the nodes that are children of this DOM node. If this node has no child nodes, childNodes is the interface of an empty list.

You can use childNodes to iterate through a node's children, or you can the firstChild (see page 7569) and lastChild (see page 7569) properties along with the nextSibling (see page 7571) method on each child node.

Note: Attributes are not considered child nodes of a node. To get the values of this node's attributes (see page 7568), use the attributes (see page 7568) property instead.

Note: IDOMNode (see page 7557) identifies child nodes slightly differently than does the IXMLNode interface. When using the DOM, the values of many nodes are treated as separate child nodes, where the two are merged when using IXMLNode. For example, given the following fragment from an XML document:

```
<Time>8:00 pm est</Time>
```

Note: The DOM parses this string into two nodes: the <Time (see page 4297)> node (an element node), and a single child node of the <Time (see page 4297)> node (an text node with the value '8:00 pm est'). IXMLNode treats this as a single node with the name 'Time (see page 4297)' and the value '8:00 pm est'.

See Also

attributes (see page 7568)

parentNode (see page 7575)

firstChild (see page 7569)

lastChild (see page 7569)

nextSibling (see page 7571)

previousSibling (see page 7576)

hasChildNodes (see page 7564)

replaceChild (see page 7566)

removeChild (see page 7566)

appendChild (see page 7559)

IXMLNode (see page 7669)

80.2.13.3.3 IDOMNode.firstChild Property

Represents the first child node of this node.

Pascal

```
property firstChild: IDOMNode;
```

C++

```
__property IDOMNode firstChild;
```

Description

Use firstChild to obtain the IDOMNode (see page 7557) interface of this node's first child node. If the node does not have any children, the value of firstChild is nil (Delphi) or NULL (C++).

The firstChild property is useful for iterating through all the children of a node.

See Also

childNodes (see page 7568)

lastChild (see page 7569)

nextSibling (see page 7571)

previousSibling (see page 7576)

hasChildNodes (see page 7564)

80.2.13.3.4 IDOMNode.lastChild Property

Represents the last child node of this node.

Pascal

```
property lastChild: IDOMNode;
```

C++

```
__property IDOMNode lastChild;
```

Description

Use lastChild to obtain the IDOMNode (see page 7557) interface of this node's last child node. If there are no child nodes, the value of lastChild is nil (Delphi) or NULL (C++).

The lastChild property is useful when iterating through all the children of a node.

See Also

childNodes (see page 7568)

firstChild (see page 7569)

nextSibling (see page 7571)

previousSibling (see page 7576)

80.2.13.3.5 IDOMNode.localName Property

Indicates the name of the node without any namespace prefix (see page 7576).

Pascal

```
property localName: DOMString;
```

C++

```
__property DOMString localName;
```

Description

Read (see page 3634) localName to determine the name of the DOM node with any namespace prefix (see page 7576) stripped away.

The localName property is only defined for element nodes and attribute nodes (nodes with nodeType (see page 7572) ELEMENT_NODE (see page 7627) and ATTRIBUTE_NODE (see page 7624)).

For element nodes, the local name is derived from the name that appears in the XML tag. For example, the local name for <xsi:Name First=John Last=Doe> is 'Name'.

For attribute nodes, the local name is derived from the string that appears to the left of the equals sign in the attribute declaration. For example, the element <Name First=John Last=Doe> results in two attribute nodes, with the local names 'First' and 'Last'.

For any other node types, localName is nil (Delphi) or NULL (C++).

Note: If the prefix (see page 7576) and localName properties are defined for a node, nodeName (see page 7572) is the combination of those two values.

Note: localName is nil (Delphi) or NULL (C++) for nodes created with a DOM level 1 interface such as the document's createElement method, even if they are element or attribute nodes.

See Also

prefix (see page 7576)

nodeType (see page 7572)

nodeName (see page 7572)

80.2.13.3.6 IDOMNode.namespaceURI Property

Indicates the URI for the namespace used in the qualified node name.

Pascal

```
property namespaceURI: DOMString;
```

C++

```
__property DOMString namespaceURI;
```

Description

Read (see page 3634) namespaceURI to determine the URI for the namespace assigned to the node when it is created. If a namespace URI was not given when the node was created, namespaceURI is nil (Delphi) or NULL (C++). That is, namespaceURI does not supply a value if the namespace URI must be computed from namespace declarations in scope.

Only element and attribute nodes can have a namespaceURI value. For any other node types, namespaceURI is always nil (Delphi) or NULL (C++). Attribute nodes do not inherit a namespace from the element node to which they are attached. If an attribute is not explicitly given a namespace, its namespaceURI is nil (Delphi) or NULL (C++).

Note: namespaceURI is nil (Delphi) or NULL (C++) for nodes created with a DOM level 1 interface such as the document's createElement method, even if they are element or attribute nodes.

See Also

localName (see page 7570)

prefix (see page 7576)

nodeType (see page 7572)

nodeName (see page 7572)

80.2.13.3.7 IDOMNode.nextSibling Property

Represents the next child of this node's parent.

Pascal

```
property nextSibling: IDOMNode;
```

C++

```
__property IDOMNode nextSibling;
```

Description

Use nextSibling to obtain the IDOMNode (see page 7557) interface of the node that immediately follows this node. If the node's parent does not have any more child nodes after this one, the value of nextSibling is nil (Delphi) or NULL (C++).

The nextSibling property is useful for iterating through all the children of a node.

See Also

childNodes (see page 7568)

parentNode (see page 7575)

previousSibling (see page 7576)

hasChildNodes (see page 7564)

80.2.13.3.8 IDOMNode.nodeName Property

Indicates the name of the node.

Pascal

```
property nodeName: DOMString;
```

C++

```
__property DOMString nodeName;
```

Description

Read (see page 3634) nodeName to determine the name of the DOM node. The value of nodeName depends on the type of the node, as indicated in the following table:

nodeType (see page 7572)	nodeName
ATTRIBUTE_NODE (see page 7624)	The name of the attribute.
ELEMENT_NODE (see page 7627)	The tag name for the node.
TEXT_NODE (see page 7636)	#text
CDATA_SECTION_NODE (see page 7625)	#cdata-section
ENTITY_REFERENCE_NODE (see page 7628)	The name of the entity reference.
ENTITY_NODE (see page 7627)	The entity name.
PROCESSING_INSTRUCTION_NODE (see page 7633)	The target of the processing instruction.
COMMENT_NODE (see page 7625)	#comment
DOCUMENT_NODE (see page 7626)	#document
DOCUMENT_TYPE_NODE (see page 7626)	The document type name.
DOCUMENT_FRAGMENT_NODE (see page 7625)	#document-fragment
NOTATION_NODE (see page 7632)	The notation name.

Note: If the prefix (see page 7576) and localName (see page 7570) properties are defined for a node, nodeName is the combination of those two values.

See Also

localName (see page 7570)

nodeType (see page 7572)

nodeValue (see page 7574)

prefix (see page 7576)

80.2.13.3.9 IDOMNode.nodeType Property

Indicates the type of the node.

Pascal

```
property nodeType: DOMNodeType;
```

C++

```
__property DOMNodeType nodeType;
```

Description

Read (see page 3634) nodeType to determine the type of the node. The type reflects the way the node is used in the XML document. The following table lists the possible values:

nodeType	Meaning
ATTRIBUTE_NODE (see page 7624)	The node represents an attribute of an element. It is not the child of another node, but its value can be accessed using the attributes (see page 7568) property of the element node's interface. An attribute node can have child nodes of type TEXT_NODE (see page 7636) and ENTITY_REFERENCE_NODE (see page 7628).
ELEMENT_NODE (see page 7627)	The node represents a tagged element. The child nodes of an element node can be of types: ELEMENT_NODE (see page 7627), TEXT_NODE (see page 7636), COMMENT_NODE (see page 7625), CDATA_SECTION_NODE (see page 7625), ENTITY_REFERENCE_NODE (see page 7628), and PROCESSING_INSTRUCTION_NODE (see page 7633). Element nodes can also have attributes (see page 7568) (ATTRIBUTE_NODE (see page 7624)). Element nodes can be the child of a node of type DOCUMENT_FRAGMENT_NODE (see page 7625), ELEMENT_NODE (see page 7627), DOCUMENT_NODE (see page 7626), and ENTITY_REFERENCE_NODE (see page 7628).
TEXT_NODE (see page 7636)	The node represents the text value associated with another node. A text node cannot have any child nodes, but can appear as the child node of a node of type ATTRIBUTE_NODE (see page 7624), ELEMENT_NODE (see page 7627), DOCUMENT_FRAGMENT_NODE (see page 7625), or ENTITY_REFERENCE_NODE (see page 7628).
CDATA_SECTION_NODE (see page 7625)	The node represents a CDATA section in the XML source. CDATA sections identify blocks of text that would otherwise be interpreted as markup. A CDATA_SECTION_NODE (see page 7625) node can't have any child nodes. They can appear as the child of an ELEMENT_NODE (see page 7627), DOCUMENT_FRAGMENT_NODE (see page 7625), or ENTITY_REFERENCE_NODE (see page 7628) node.
ENTITY_REFERENCE_NODE (see page 7628)	The node represents a reference to an entity in the XML document. This can be any type of entity, including character entity references. The children of an entity reference node can be of the following types: ELEMENT_NODE (see page 7627), COMMENT_NODE (see page 7625), CDATA_SECTION_NODE (see page 7625), PROCESSING_INSTRUCTION_NODE (see page 7633), TEXT_NODE (see page 7636), and ENTITY_REFERENCE_NODE (see page 7628). The entity reference node can appear as the child of a DOCUMENT_FRAGMENT_NODE (see page 7625), ATTRIBUTE_NODE (see page 7624), ELEMENT_NODE (see page 7627), or ENTITY_REFERENCE_NODE (see page 7628) node.
ENTITY_NODE (see page 7627)	The node represents an expanded entity. Entity nodes can have child nodes that represent the expanded entity (for example, TEXT_NODE (see page 7636) and ENTITY_REFERENCE_NODE (see page 7628) nodes). Entity nodes only appear as the child of a DOCUMENT_TYPE_NODE (see page 7626) node.
PROCESSING_INSTRUCTION_NODE (see page 7633)	The node represents a processing instruction (PI) from the XML document. A PI node cannot have any child nodes, but can appear as the child node of a node of type DOCUMENT_NODE (see page 7626), DOCUMENT_FRAGMENT_NODE (see page 7625), ELEMENT_NODE (see page 7627), or ENTITY_REFERENCE_NODE (see page 7628).

COMMENT_NODE (see page 7625)	The node represents a comment in the XML document. Comment nodes do not have child nodes. They appear as the child of an ELEMENT_NODE (see page 7627), DOCUMENT_NODE (see page 7626), DOCUMENT_FRAGMENT_NODE (see page 7625), or ENTITY_REFERENCE_NODE (see page 7628) node.
DOCUMENT_NODE (see page 7626)	The node represents a document object, which is the root of the entire XML document. Document nodes have a single ELEMENT_NODE (see page 7627) node as a child (the documentElement). In addition, they can have child nodes of type COMMENT_NODE (see page 7625), DOCUMENT_TYPE_NODE (see page 7626), and PROCESSING_INSTRUCTION_NODE (see page 7633). Because the document is the root of the XML document, it never appears as a child node.
DOCUMENT_TYPE_NODE (see page 7626)	The node represents the document type declaration, indicated by the <!DOCTYPE > tag. The document type node can child nodes of type NOTATION_NODE (see page 7632) and ENTITY_NODE (see page 7627). It is appears the child of the document node.
DOCUMENT_FRAGMENT_NODE (see page 7625)	The node represents a document fragment. A document fragment node associates a node or subtree with a document without actually being contained in the document. Document fragment nodes can have child nodes of type ELEMENT_NODE (see page 7627), COMMENT_NODE (see page 7625), PROCESSING_INSTRUCTION_NODE (see page 7633), CDATA_SECTION_NODE (see page 7625), TEXT_NODE (see page 7636), and ENTITY_REFERENCE_NODE (see page 7628). It never appears as the child of another node.
NOTATION_NODE (see page 7632)	The node represents a notation in the document type declaration. It always appears as the child of a DOCUMENT_TYPE_NODE (see page 7626) node and never has any child nodes.

See Also

nodeName (see page 7572)

nodeValue (see page 7574)

80.2.13.3.10 IDOMNode.nodeValue Property

Indicates the value of the node.

Pascal

```
property nodeValue: DOMString;
```

C++

```
__property DOMString nodeValue;
```

Description

Read (see page 3634) nodeValue to determine the value of the DOM node. Set nodeValue to change the value of a node.

Note: You can't set the value of a node whose value is listed as nil (Delphi) or NULL (C++) in the following table.

The value of nodeValue depends on the type of the node, as indicated in the following table:

nodeType (see page 7572)	nodeValue
ATTRIBUTE_NODE (see page 7624)	The value of the attribute.
ELEMENT_NODE (see page 7627)	nil (Delphi) or NULL (C++).
TEXT_NODE (see page 7636)	The content of the text node.

CDATA_SECTION_NODE (see page 7625)	The content of the CDATA section.
ENTITY_REFERENCE_NODE (see page 7628)	nil (Delphi) or NULL (C++).
ENTITY_NODE (see page 7627)	nil (Delphi) or NULL (C++).
PROCESSING_INSTRUCTION_NODE (see page 7633)	The content of the processing instruction except for the target.
COMMENT_NODE (see page 7625)	The content of the comment.
DOCUMENT_NODE (see page 7626)	nil (Delphi) or NULL (C++).
DOCUMENT_TYPE_NODE (see page 7626)	nil (Delphi) or NULL (C++).
DOCUMENT_FRAGMENT_NODE (see page 7625)	nil (Delphi) or NULL (C++).
NOTATION_NODE (see page 7632)	nil (Delphi) or NULL (C++).

See Also

localName (see page 7570)

nodeType (see page 7572)

80.2.13.3.11 IDOMNode.ownerDocument Property

Represents the interface for the document that contains this node.

Pascal

```
property ownerDocument: IDOMDocument;
```

C++

```
__property IDOMDocument ownerDocument;
```

Description

Use ownerDocument to obtain the interface of the document that contains this node. The IDOMDocument (see page 7530) interface can be used to create new nodes.

If this node has nodeType (see page 7572) of DOCUMENT_NODE (see page 7626) or DOCUMENT_TYPE_NODE (see page 7626), the value of ownerDocument is nil (Delphi) or NULL (C++).

See Also

parentNode (see page 7575)

nodeType (see page 7572)

80.2.13.3.12 IDOMNode.parentNode Property

Provides access to the interface of the node's parent node.

Pascal

```
property parentNode: IDOMNode;
```

C++

```
__property IDOMNode parentNode;
```

Description

Read (see page 3634) parentNode to access the node's parent node, if it exists. If the node does not have a parent node, the value of parentNode is nil (Delphi) or NULL (C++).

Nodes of type ATTRIBUTE_NODE (see page 7624), DOCUMENT_NODE (see page 7626), and DOCUMENT_FRAGMENT_NODE (see page 7625) never have parents. Nodes of type ENTITY_NODE (see page 7627) and NOTATION_NODE (see page 7632) may not have a parent. Nodes of other types have a parent unless they have just been created but not yet added to the hierarchy, or if they have just been removed from the hierarchy.

See Also

nodeType (see page 7572)

80.2.13.3.13 IDOMNode.prefix Property

Indicates the namespace prefix for the node.

Pascal

```
property prefix: DOMString;
```

C++

```
__property DOMString prefix;
```

Description

Read (see page 3634) prefix to determine the namespace prefix for an element or attribute node.

For element nodes, prefix is derived from the name that appears in the XML tag. For example, the prefix for <xsi:Name First=John Last=Doe> is 'xsi'.

For attribute nodes, prefix is derived from the string that appears to the left of the equals sign in the attribute declaration. For example, with the element <xsi:Name xsi:First=John xsi:Last=Doe>, both attribute nodes have the prefix of 'xsi'. Attribute nodes do not inherit a prefix value from the element node to which they are attached. For example, with the element <xsi:Name First=John Last=Doe>, both attributes (see page 7568) have a prefix of nil (Delphi) or NULL (C++).

For any other node types, prefix is nil (Delphi) or NULL (C++).

Note: If the prefix and localName (see page 7570) properties are defined for a node, nodeName (see page 7572) is the combination of those two values.

Note: prefix is nil (Delphi) or NULL (C++) for nodes created with a DOM level 1 interface such as the document's createElement method, even if they are element or attribute nodes.

See Also

localName (see page 7570)

namespaceURI (see page 7571)

nodeType (see page 7572)

nodeName (see page 7572)

80.2.13.3.14 IDOMNode.previousSibling Property

Represents the preceding child of this node's parent.

Pascal

```
property previousSibling: IDOMNode;
```

C++

```
__property IDOMNode previousSibling;
```

Description

Use previousSibling to obtain the IDOMNode (see page 7557) interface of the node that immediately precedes this node. If the node's parent does not have any more child nodes before this one, the value of previousSibling is nil (Delphi) or NULL (C++).

See Also

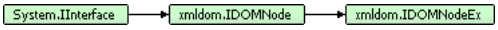
childNodes (see page 7568)

parentNode (see page 7575)

hasChildNodes (see page 7564)

80.2.14 IDOMNodeEx Interface

Class Hierarchy



Pascal

```
IDOMNodeEx = interface(IDOMNode);
```

C++

```
__interface IDOMNodeEx : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMNodeEx.

80.2.14.1 IDOMNodeEx Members

The following tables list the members exposed by IDOMNodeEx.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.

	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	get_text (see page 7579)	This is get_text, a member of class IDOMNodeEx.
	get_xml (see page 7579)	This is get_xml, a member of class IDOMNodeEx.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	set_text (see page 7579)	This is set_text, a member of class IDOMNodeEx.
	supports (see page 7567)	Indicates whether the node supports a specified feature.
	transformNode (see page 7579)	This is the overview for the transformNode method overload.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.
	text (see page 7580)	This is text, a member of class IDOMNodeEx.
R	xml (see page 7580)	This is xml, a member of class IDOMNodeEx.

80.2.14.2 IDOMNodeEx Methods

The methods of the IDOMNodeEx class are listed here.

Public Methods

	Name	Description
	get_text (see page 7579)	This is get_text, a member of class IDOMNodeEx.
	get_xml (see page 7579)	This is get_xml, a member of class IDOMNodeEx.
	set_text (see page 7579)	This is set_text, a member of class IDOMNodeEx.
	transformNode (see page 7579)	This is the overview for the transformNode method overload.

80.2.14.2.1 IDOMNodeEx.get_text Method

Pascal

```
function get_text: DOMString; safecall;
```

C++

```
__fastcall DOMString get_text();
```

Description

This is get_text, a member of class IDOMNodeEx.

80.2.14.2.2 IDOMNodeEx.get_xml Method

Pascal

```
function get_xml: DOMString; safecall;
```

C++

```
__fastcall DOMString get_xml();
```

Description

This is get_xml, a member of class IDOMNodeEx.

80.2.14.2.3 IDOMNodeEx.set_text Method

Pascal

```
procedure set_text(const Value: DOMString); safecall;
```

C++

```
__fastcall set_text(const DOMString Value);
```

Description

This is set_text, a member of class IDOMNodeEx.

80.2.14.2.4 transformNode Method

This is the overview for the transformNode method overload.

Overload List

	Name	Description
	IDOMNodeEx.transformNode (IDOMNode, IDOMDocument) (see page 7579)	This is transformNode, a member of class IDOMNodeEx.
	IDOMNodeEx.transformNode (IDOMNode, WideString) (see page 7580)	This is transformNode, a member of class IDOMNodeEx.

80.2.14.2.4.1 IDOMNodeEx.transformNode Method (IDOMNode, IDOMDocument)

Pascal

```
procedure transformNode(const stylesheet: IDOMNode; const output: IDOMDocument); safecall;
overload;
```

C++

```
__fastcall transformNode(const IDOMNode stylesheet, const IDOMDocument output);
```

Description

This is transformNode, a member of class IDOMNodeEx.

80.2.14.2.4.2 IDOMNodeEx.transformNode Method (IDOMNode, WideString)

Pascal

```
procedure transformNode(const stylesheet: IDOMNode; var output: WideString); safecall;
overload;
```

C++

```
__fastcall transformNode(const IDOMNode stylesheet, BSTR output);
```

Description

This is transformNode, a member of class IDOMNodeEx.

80.2.14.3 IDOMNodeEx Properties

The properties of the IDOMNodeEx class are listed here.

Public Properties

	Name	Description
	text (see page 7580)	This is text, a member of class IDOMNodeEx.
R	xml (see page 7580)	This is xml, a member of class IDOMNodeEx.

80.2.14.3.1 IDOMNodeEx.text Property

Pascal

```
property text: DOMString;
```

C++

```
__property DOMString text;
```

Description

This is text, a member of class IDOMNodeEx.

80.2.14.3.2 IDOMNodeEx.xml Property

Pascal

```
property xml: DOMString;
```

C++

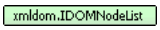
```
__property DOMString xml;
```

Description

This is xml, a member of class IDOMNodeEx.

80.2.15 IDOMNodeList Interface

Class Hierarchy



Pascal

```
IDOMNodeList = interface;
```

C++

```
__interface IDOMNodeList;
```

File

xmldom

Description

This is class xmldom.IDOMNodeList.

80.2.15.1 IDOMNodeList Members

The following tables list the members exposed by IDOMNodeList.

Public Methods

	Name	Description
	get_item (see page 7581)	This is get_item, a member of class IDOMNodeList.
	get_length (see page 7582)	This is get_length, a member of class IDOMNodeList.

Public Properties

	Name	Description
R	item (see page 7582)	This is item, a member of class IDOMNodeList.
R	length (see page 7582)	This is length, a member of class IDOMNodeList.

80.2.15.2 IDOMNodeList Methods

The methods of the IDOMNodeList class are listed here.

Public Methods

	Name	Description
	get_item (see page 7581)	This is get_item, a member of class IDOMNodeList.
	get_length (see page 7582)	This is get_length, a member of class IDOMNodeList.

80.2.15.2.1 IDOMNodeList.get_item Method

Pascal

```
function get_item(index: Integer): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode get_item(int index);
```

Description

This is get_item, a member of class IDOMNodeList.

80.2.15.2.2 IDOMNodeList.get_length Method

Pascal

```
function get_length: Integer; safecall;
```

C++

```
__fastcall int get_length();
```

Description

This is get_length, a member of class IDOMNodeList.

80.2.15.3 IDOMNodeList Properties

The properties of the IDOMNodeList class are listed here.

Public Properties

	Name	Description
	item (see page 7582)	This is item, a member of class IDOMNodeList.
	length (see page 7582)	This is length, a member of class IDOMNodeList.

80.2.15.3.1 IDOMNodeList.item Property

Pascal

```
property item [index: Integer]: IDOMNode;
```

C++

```
__property IDOMNode item[int index];
```

Description

This is item, a member of class IDOMNodeList.

80.2.15.3.2 IDOMNodeList.length Property

Pascal

```
property length: Integer;
```

C++

```
__property int length;
```

Description

This is length, a member of class IDOMNodeList.

80.2.16 IDOMNodeSelect Interface

Class Hierarchy

Pascal

```
IDOMNodeSelect = interface(IInterface);
```

C++

```
__interface IDOMNodeSelect : public IInterface;
```

File

xmldom

Description

This is class xmldom.IDOMNodeSelect.

80.2.16.1 IDOMNodeSelect Members

The following tables list the members exposed by IDOMNodeSelect.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	selectNode (see page 7583)	This is selectNode, a member of class IDOMNodeSelect.
	selectNodes (see page 7584)	This is selectNodes, a member of class IDOMNodeSelect.

80.2.16.2 IDOMNodeSelect Methods

The methods of the IDOMNodeSelect class are listed here.

Public Methods

	Name	Description
	selectNode (see page 7583)	This is selectNode, a member of class IDOMNodeSelect.
	selectNodes (see page 7584)	This is selectNodes, a member of class IDOMNodeSelect.

80.2.16.2.1 IDOMNodeSelect.selectNode Method

Pascal

```
function selectNode(const nodePath: WideString): IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode selectNode(const BSTR nodePath);
```

Description

This is selectNode, a member of class IDOMNodeSelect.

80.2.16.2.2 IDOMNodeSelect.selectNodes Method

Pascal

```
function selectNodes(const nodePath: WideString): IDOMNodeList; safecall;
```

C++

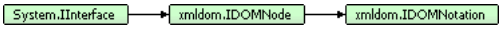
```
__fastcall IDOMNodeList selectNodes(const BSTR nodePath);
```

Description

This is selectNodes, a member of class IDOMNodeSelect.

80.2.17 IDOMNotation Interface

Class Hierarchy



Pascal

```
IDOMNotation = interface(IDOMNode);
```

C++

```
__interface IDOMNotation : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMNotation.

80.2.17.1 IDOMNotation Members

The following tables list the members exposed by IDOMNotation.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.

	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	get_publicId (see page 7585)	This is get_publicId, a member of class IDOMNotation.
	get_systemId (see page 7586)	This is get_systemId, a member of class IDOMNotation.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_nodeValue (see page 7567)	Sets the value of the node.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.
R	publicId (see page 7586)	This is publicId, a member of class IDOMNotation.
R	systemId (see page 7586)	This is systemId, a member of class IDOMNotation.

80.2.17.2 IDOMNotation Methods

The methods of the IDOMNotation class are listed here.

Public Methods

	Name	Description
	get_publicId (see page 7585)	This is get_publicId, a member of class IDOMNotation.
	get_systemId (see page 7586)	This is get_systemId, a member of class IDOMNotation.

80.2.17.2.1 IDOMNotation.get_publicId Method

Pascal

```
function get_publicId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_publicId();
```

Description

This is get_publicId, a member of class IDOMNotation.

80.2.17.2.2 IDOMNotation.get_systemId Method

Pascal

```
function get_systemId: DOMString; safecall;
```

C++

```
__fastcall DOMString get_systemId();
```

Description

This is get_systemId, a member of class IDOMNotation.

80.2.17.3 IDOMNotation Properties

The properties of the IDOMNotation class are listed here.

Public Properties

	Name	Description
R	publicId (see page 7586)	This is publicId, a member of class IDOMNotation.
R	systemId (see page 7586)	This is systemId, a member of class IDOMNotation.

80.2.17.3.1 IDOMNotation.publicId Property

Pascal

```
property publicId: DOMString;
```

C++

```
__property DOMString publicId;
```

Description

This is publicId, a member of class IDOMNotation.

80.2.17.3.2 IDOMNotation.systemId Property

Pascal

```
property systemId: DOMString;
```

C++

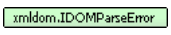
```
__property DOMString systemId;
```

Description

This is systemId, a member of class IDOMNotation.

80.2.18 IDOMParseError Interface

Class Hierarchy



Pascal

```
IDOMParseError = interface;
```

C++

```
__interface IDOMParseError;
```

File

xmldom

Description

This is class xmldom.IDOMParseError.

80.2.18.1 IDOMParseError Members

The following tables list the members exposed by IDOMParseError.

Public Methods

	Name	Description
	get_errorCode (see page 7588)	This is get_errorCode, a member of class IDOMParseError.
	get_filepos (see page 7588)	This is get_filepos, a member of class IDOMParseError.
	get_line (see page 7588)	This is get_line, a member of class IDOMParseError.
	get_linepos (see page 7588)	This is get_linepos, a member of class IDOMParseError.
	get_reason (see page 7588)	This is get_reason, a member of class IDOMParseError.
	get_srcText (see page 7589)	This is get_srcText, a member of class IDOMParseError.
	get_url (see page 7589)	This is get_url, a member of class IDOMParseError.

Public Properties

	Name	Description
R	errorCode (see page 7589)	This is errorCode, a member of class IDOMParseError.
R	filePos (see page 7590)	This is filePos, a member of class IDOMParseError.
R	line (see page 7590)	This is line, a member of class IDOMParseError.
R	linePos (see page 7590)	This is linePos, a member of class IDOMParseError.
R	reason (see page 7590)	This is reason, a member of class IDOMParseError.
R	srcText (see page 7590)	This is srcText, a member of class IDOMParseError.
R	url (see page 7591)	This is url, a member of class IDOMParseError.

80.2.18.2 IDOMParseError Methods

The methods of the IDOMParseError class are listed here.

Public Methods

	Name	Description
	get_errorCode (see page 7588)	This is get_errorCode, a member of class IDOMParseError.
	get_filepos (see page 7588)	This is get_filepos, a member of class IDOMParseError.
	get_line (see page 7588)	This is get_line, a member of class IDOMParseError.
	get_linepos (see page 7588)	This is get_linepos, a member of class IDOMParseError.
	get_reason (see page 7588)	This is get_reason, a member of class IDOMParseError.
	get_srcText (see page 7589)	This is get_srcText, a member of class IDOMParseError.
	get_url (see page 7589)	This is get_url, a member of class IDOMParseError.

80.2.18.2.1 IDOMParseError.get_errorCode Method

Pascal

```
function get_errorCode: Integer; safecall;
```

C++

```
__fastcall int get_errorCode();
```

Description

This is get_errorCode, a member of class IDOMParseError.

80.2.18.2.2 IDOMParseError.get_filepos Method

Pascal

```
function get_filepos: Integer; safecall;
```

C++

```
__fastcall int get_filepos();
```

Description

This is get_filepos, a member of class IDOMParseError.

80.2.18.2.3 IDOMParseError.get_line Method

Pascal

```
function get_line: Integer; safecall;
```

C++

```
__fastcall int get_line();
```

Description

This is get_line, a member of class IDOMParseError.

80.2.18.2.4 IDOMParseError.get_linepos Method

Pascal

```
function get_linepos: Integer; safecall;
```

C++

```
__fastcall int get_linepos();
```

Description

This is get_linepos, a member of class IDOMParseError.

80.2.18.2.5 IDOMParseError.get_reason Method

Pascal

```
function get_reason: DOMString; safecall;
```

C++

```
__fastcall DOMString get_reason();
```

Description

This is get_reason, a member of class IDOMParseError.

80.2.18.2.6 IDOMParseError.get_srcText Method

Pascal

```
function get_srcText: DOMString; safecall;
```

C++

```
__fastcall DOMString get_srcText();
```

Description

This is get_srcText, a member of class IDOMParseError.

80.2.18.2.7 IDOMParseError.get_url Method

Pascal

```
function get_url: DOMString; safecall;
```

C++

```
__fastcall DOMString get_url();
```

Description

This is get_url, a member of class IDOMParseError.

80.2.18.3 IDOMParseError Properties

The properties of the IDOMParseError class are listed here.

Public Properties

	Name	Description
	errorCode (see page 7589)	This is errorCode, a member of class IDOMParseError.
	filePos (see page 7590)	This is filePos, a member of class IDOMParseError.
	line (see page 7590)	This is line, a member of class IDOMParseError.
	linePos (see page 7590)	This is linePos, a member of class IDOMParseError.
	reason (see page 7590)	This is reason, a member of class IDOMParseError.
	srcText (see page 7590)	This is srcText, a member of class IDOMParseError.
	url (see page 7591)	This is url, a member of class IDOMParseError.

80.2.18.3.1 IDOMParseError.errorCode Property

Pascal

```
property errorCode: Integer;
```

C++

```
__property int errorCode;
```

Description

This is errorCode, a member of class IDOMParseError.

80.2.18.3.2 IDOMParseError.filePos Property

Pascal

```
property filePos: Integer;
```

C++

```
__property int filePos;
```

Description

This is filePos, a member of class IDOMParseError.

80.2.18.3.3 IDOMParseError.line Property

Pascal

```
property line: Integer;
```

C++

```
__property int line;
```

Description

This is line, a member of class IDOMParseError.

80.2.18.3.4 IDOMParseError.linePos Property

Pascal

```
property linePos: Integer;
```

C++

```
__property int linePos;
```

Description

This is linePos, a member of class IDOMParseError.

80.2.18.3.5 IDOMParseError.reason Property

Pascal

```
property reason: DOMString;
```

C++

```
__property DOMString reason;
```

Description

This is reason, a member of class IDOMParseError.

80.2.18.3.6 IDOMParseError.srcText Property

Pascal

```
property srcText: DOMString;
```

C++

```
__property DOMString srcText;
```

Description

This is srcText, a member of class IDOMParseError.

80.2.18.3.7 IDOMParseError.url Property

Pascal

```
property url: DOMString;
```

C++

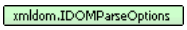
```
__property DOMString url;
```

Description

This is url, a member of class IDOMParseError.

80.2.19 IDOMParseOptions Interface

Class Hierarchy



Pascal

```
IDOMParseOptions = interface;
```

C++

```
__interface IDOMParseOptions;
```

File

xmldom

Description

This is class xmldom.IDOMParseOptions.

80.2.19.1 IDOMParseOptions Members

The following tables list the members exposed by IDOMParseOptions.

Public Methods

	Name	Description
	get_async (see page 7592)	This is get_async, a member of class IDOMParseOptions.
	get_preserveWhiteSpace (see page 7592)	This is get_preserveWhiteSpace, a member of class IDOMParseOptions.
	get_resolveExternals (see page 7592)	This is get_resolveExternals, a member of class IDOMParseOptions.
	get_validate (see page 7593)	This is get_validate, a member of class IDOMParseOptions.
	set_async (see page 7593)	This is set_async, a member of class IDOMParseOptions.
	set_preserveWhiteSpace (see page 7593)	This is set_preserveWhiteSpace, a member of class IDOMParseOptions.
	set_resolveExternals (see page 7593)	This is set_resolveExternals, a member of class IDOMParseOptions.
	set_validate (see page 7593)	This is set_validate, a member of class IDOMParseOptions.

Public Properties

	Name	Description
	async (see page 7594)	This is async, a member of class IDOMParseOptions.

	preserveWhiteSpace (see page 7594)	This is preserveWhiteSpace, a member of class IDOMParseOptions.
	resolveExternals (see page 7594)	This is resolveExternals, a member of class IDOMParseOptions.
	validate (see page 7595)	This is validate, a member of class IDOMParseOptions.

80.2.19.2 IDOMParseOptions Methods

The methods of the IDOMParseOptions class are listed here.

Public Methods

	Name	Description
	get_async (see page 7592)	This is get_async, a member of class IDOMParseOptions.
	get_preserveWhiteSpace (see page 7592)	This is get_preserveWhiteSpace, a member of class IDOMParseOptions.
	get_resolveExternals (see page 7592)	This is get_resolveExternals, a member of class IDOMParseOptions.
	get_validate (see page 7593)	This is get_validate, a member of class IDOMParseOptions.
	set_async (see page 7593)	This is set_async, a member of class IDOMParseOptions.
	set_preserveWhiteSpace (see page 7593)	This is set_preserveWhiteSpace, a member of class IDOMParseOptions.
	set_resolveExternals (see page 7593)	This is set_resolveExternals, a member of class IDOMParseOptions.
	set_validate (see page 7593)	This is set_validate, a member of class IDOMParseOptions.

80.2.19.2.1 IDOMParseOptions.get_async Method

Pascal

```
function get_async: Boolean; safecall;
```

C++

```
__fastcall Boolean get_async();
```

Description

This is get_async, a member of class IDOMParseOptions.

80.2.19.2.2 IDOMParseOptions.get_preserveWhiteSpace Method

Pascal

```
function get_preserveWhiteSpace: Boolean; safecall;
```

C++

```
__fastcall Boolean get_preserveWhiteSpace();
```

Description

This is get_preserveWhiteSpace, a member of class IDOMParseOptions.

80.2.19.2.3 IDOMParseOptions.get_resolveExternals Method

Pascal

```
function get_resolveExternals: Boolean; safecall;
```

C++

```
__fastcall Boolean get_resolveExternals();
```

Description

This is get_resolveExternals, a member of class IDOMParseOptions.

80.2.19.2.4 IDOMParseOptions.get_validate Method

Pascal

```
function get_validate: Boolean; safecall;
```

C++

```
__fastcall Boolean get_validate();
```

Description

This is get_validate, a member of class IDOMParseOptions.

80.2.19.2.5 IDOMParseOptions.set_async Method

Pascal

```
procedure set_async(Value: Boolean); safecall;
```

C++

```
__fastcall set_async(Boolean Value);
```

Description

This is set_async, a member of class IDOMParseOptions.

80.2.19.2.6 IDOMParseOptions.set_preserveWhiteSpace Method

Pascal

```
procedure set_preserveWhiteSpace(Value: Boolean); safecall;
```

C++

```
__fastcall set_preserveWhiteSpace(Boolean Value);
```

Description

This is set_preserveWhiteSpace, a member of class IDOMParseOptions.

80.2.19.2.7 IDOMParseOptions.set_resolveExternals Method

Pascal

```
procedure set_resolveExternals(Value: Boolean); safecall;
```

C++

```
__fastcall set_resolveExternals(Boolean Value);
```

Description

This is set_resolveExternals, a member of class IDOMParseOptions.

80.2.19.2.8 IDOMParseOptions.set_validate Method

Pascal

```
procedure set_validate(Value: Boolean); safecall;
```

C++

```
__fastcall set_validate(Boolean Value);
```

Description

This is set_validate, a member of class IDOMParseOptions.

80.2.19.3 IDOMParseOptions Properties

The properties of the IDOMParseOptions class are listed here.

Public Properties

	Name	Description
	async (see page 7594)	This is async, a member of class IDOMParseOptions.
	preserveWhiteSpace (see page 7594)	This is preserveWhiteSpace, a member of class IDOMParseOptions.
	resolveExternals (see page 7594)	This is resolveExternals, a member of class IDOMParseOptions.
	validate (see page 7595)	This is validate, a member of class IDOMParseOptions.

80.2.19.3.1 IDOMParseOptions.async Property

Pascal

```
property async: Boolean;
```

C++

```
__property Boolean async;
```

Description

This is async, a member of class IDOMParseOptions.

80.2.19.3.2 IDOMParseOptions.preserveWhiteSpace Property

Pascal

```
property preserveWhiteSpace: Boolean;
```

C++

```
__property Boolean preserveWhiteSpace;
```

Description

This is preserveWhiteSpace, a member of class IDOMParseOptions.

80.2.19.3.3 IDOMParseOptions.resolveExternals Property

Pascal

```
property resolveExternals: Boolean;
```

C++

```
__property Boolean resolveExternals;
```

Description

This is resolveExternals, a member of class IDOMParseOptions.

80.2.19.3.4 IDOMParseOptions.validate Property

Pascal

```
property validate: Boolean;
```

C++

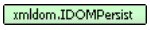
```
__property Boolean validate;
```

Description

This is validate, a member of class IDOMParseOptions.

80.2.20 IDOMPersist Interface

Class Hierarchy



Pascal

```
IDOMPersist = interface;
```

C++

```
__interface IDOMPersist;
```

File

xmldom

Description

This is class xmldom.IDOMPersist.

80.2.20.1 IDOMPersist Members

The following tables list the members exposed by IDOMPersist.

Public Methods

	Name	Description
	asyncLoadState (see page 7596)	This is asyncLoadState, a member of class IDOMPersist.
	get_xml (see page 7596)	This is get_xml, a member of class IDOMPersist.
	load (see page 7596)	This is load, a member of class IDOMPersist.
	loadFromStream (see page 7597)	This is the overview for the loadFromStream method overload.
	loadxml (see page 7597)	This is loadxml, a member of class IDOMPersist.
	save (see page 7597)	This is save, a member of class IDOMPersist.
	saveToStream (see page 7598)	This is the overview for the saveToStream method overload.
	set_OnAsyncLoad (see page 7598)	This is set_OnAsyncLoad, a member of class IDOMPersist.

Public Properties

	Name	Description
R	xml (see page 7599)	This is xml, a member of class IDOMPersist.

80.2.20.2 IDOMPersist Methods

The methods of the IDOMPersist class are listed here.

Public Methods

	Name	Description
	asyncLoadState (see page 7596)	This is asyncLoadState, a member of class IDOMPersist.
	get_xml (see page 7596)	This is get_xml, a member of class IDOMPersist.
	load (see page 7596)	This is load, a member of class IDOMPersist.
	loadFromStream (see page 7597)	This is the overview for the loadFromStream method overload.
	loadxml (see page 7597)	This is loadxml, a member of class IDOMPersist.
	save (see page 7597)	This is save, a member of class IDOMPersist.
	saveToStream (see page 7598)	This is the overview for the saveToStream method overload.
	set_OnAsyncLoad (see page 7598)	This is set_OnAsyncLoad, a member of class IDOMPersist.

80.2.20.2.1 IDOMPersist.asyncLoadState Method

Pascal

```
function asyncLoadState: Integer; safecall;
```

C++

```
__fastcall int asyncLoadState();
```

Description

This is asyncLoadState, a member of class IDOMPersist.

80.2.20.2.2 IDOMPersist.get_xml Method

Pascal

```
function get_xml: DOMString; safecall;
```

C++

```
__fastcall DOMString get_xml();
```

Description

This is get_xml, a member of class IDOMPersist.

80.2.20.2.3 IDOMPersist.load Method

Pascal

```
function load(source: OleVariant): WordBool; safecall;
```

C++

```
__fastcall WordBool load(OleVariant source);
```

Description

This is load, a member of class IDOMPersist.

80.2.20.2.4 loadFromStream Method

This is the overview for the loadFromStream method overload.

Overload List

	Name	Description
	IDOMPersist.loadFromStream (IStream) (see page 7597)	This is loadFromStream, a member of class IDOMPersist.
	IDOMPersist.loadFromStream (TStream) (see page 7597)	This is loadFromStream, a member of class IDOMPersist.

80.2.20.2.4.1 IDOMPersist.loadFromStream Method (IStream)

Pascal

```
function loadFromStream(const stream: IStream): WordBool; safecall; overload;
```

C++

```
__fastcall WordBool loadFromStream(const IStream stream);
```

Description

This is loadFromStream, a member of class IDOMPersist.

80.2.20.2.4.2 IDOMPersist.loadFromStream Method (TStream)

Pascal

```
function loadFromStream(const stream: TStream): WordBool; safecall; overload;
```

C++

```
__fastcall WordBool loadFromStream(const TStream stream);
```

Description

This is loadFromStream, a member of class IDOMPersist.

80.2.20.2.5 IDOMPersist.loadxml Method

Pascal

```
function loadxml(const Value: DOMString): WordBool; safecall;
```

C++

```
__fastcall WordBool loadxml(const DOMString Value);
```

Description

This is loadxml, a member of class IDOMPersist.

80.2.20.2.6 IDOMPersist.save Method

Pascal

```
procedure save(destination: OleVariant); safecall;
```

C++

```
__fastcall save(OleVariant destination);
```

Description

This is save, a member of class IDOMPersist.

80.2.20.2.7 saveToStream Method

This is the overview for the saveToStream method overload.

Overload List

	Name	Description
	IDOMPersist.saveToStream (IStream) (see page 7598)	This is saveToStream, a member of class IDOMPersist.
	IDOMPersist.saveToStream (TStream) (see page 7598)	This is saveToStream, a member of class IDOMPersist.

80.2.20.2.7.1 IDOMPersist.saveToStream Method (IStream)

Pascal

```
procedure saveToStream(const stream: IStream); safecall; overload;
```

C++

```
__fastcall saveToStream(const IStream stream);
```

Description

This is saveToStream, a member of class IDOMPersist.

80.2.20.2.7.2 IDOMPersist.saveToStream Method (TStream)

Pascal

```
procedure saveToStream(const stream: TStream); safecall; overload;
```

C++

```
__fastcall saveToStream(const TStream stream);
```

Description

This is saveToStream, a member of class IDOMPersist.

80.2.20.2.8 IDOMPersist.set_OnAsyncLoad Method

Pascal

```
procedure set_OnAsyncLoad(const Sender: TObject; EventHandler: TAsyncEventHandler); safecall;
```

C++

```
__fastcall set_OnAsyncLoad(const TObject * Sender, TAsyncEventHandler EventHandler);
```

Description

This is set_OnAsyncLoad, a member of class IDOMPersist.

80.2.20.3 IDOMPersist Properties

The properties of the IDOMPersist class are listed here.

Public Properties

	Name	Description
	xml (see page 7599)	This is xml, a member of class IDOMPersist.

80.2.20.3.1 IDOMPersist.xml Property

Pascal

```
property xml: DOMString;
```

C++

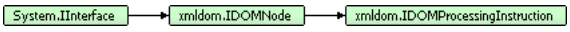
```
__property DOMString xml;
```

Description

This is xml, a member of class IDOMPersist.

80.2.21 IDOMProcessingInstruction Interface

Class Hierarchy



Pascal

```
IDOMProcessingInstruction = interface(IDOMNode);
```

C++

```
__interface IDOMProcessingInstruction : public IDOMNode;
```

File

xmldom

Description

This is class xmldom.IDOMProcessingInstruction.

80.2.21.1 IDOMProcessingInstruction Members

The following tables list the members exposed by IDOMProcessingInstruction.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.

	get_data (see page 7601)	This is get_data, a member of class IDOMProcessingInstruction.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	get_target (see page 7601)	This is get_target, a member of class IDOMProcessingInstruction.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	set_data (see page 7601)	This is set_data, a member of class IDOMProcessingInstruction.
	set_nodeValue (see page 7567)	Sets the value of the node.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
	data (see page 7601)	This is data, a member of class IDOMProcessingInstruction.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.
R	target (see page 7602)	This is target, a member of class IDOMProcessingInstruction.

80.2.21.2 IDOMProcessingInstruction Methods

The methods of the IDOMProcessingInstruction class are listed here.

Public Methods

	Name	Description
	get_data (see page 7601)	This is get_data, a member of class IDOMProcessingInstruction.
	get_target (see page 7601)	This is get_target, a member of class IDOMProcessingInstruction.
	set_data (see page 7601)	This is set_data, a member of class IDOMProcessingInstruction.

80.2.21.2.1 IDOMProcessingInstruction.get_data Method

Pascal

```
function get_data: DOMString; safecall;
```

C++

```
__fastcall DOMString get_data();
```

Description

This is get_data, a member of class IDOMProcessingInstruction.

80.2.21.2.2 IDOMProcessingInstruction.get_target Method

Pascal

```
function get_target: DOMString; safecall;
```

C++

```
__fastcall DOMString get_target();
```

Description

This is get_target, a member of class IDOMProcessingInstruction.

80.2.21.2.3 IDOMProcessingInstruction.set_data Method

Pascal

```
procedure set_data(const value: DOMString); safecall;
```

C++

```
__fastcall set_data(const DOMString value);
```

Description

This is set_data, a member of class IDOMProcessingInstruction.

80.2.21.3 IDOMProcessingInstruction Properties

The properties of the IDOMProcessingInstruction class are listed here.

Public Properties

	Name	Description
	data (see page 7601)	This is data, a member of class IDOMProcessingInstruction.
R	target (see page 7602)	This is target, a member of class IDOMProcessingInstruction.

80.2.21.3.1 IDOMProcessingInstruction.data Property

Pascal

```
property data: DOMString;
```

C++

```
__property DOMString data;
```

Description

This is data, a member of class IDOMProcessingInstruction.

80.2.21.3.2 IDOMProcessingInstruction.target Property

Pascal

```
property target: DOMString;
```

C++

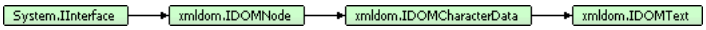
```
__property DOMString target;
```

Description

This is target, a member of class IDOMProcessingInstruction.

80.2.22 IDOMText Interface

Class Hierarchy



Pascal

```
IDOMText = interface(IDOMCharacterData);
```

C++

```
__interface IDOMText : public IDOMCharacterData;
```

File

xmldom

Description

This is class xmldom.IDOMText.

80.2.22.1 IDOMText Members

The following tables list the members exposed by IDOMText.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	appendChild (see page 7559)	Adds a new child node to this node.
	appendData (see page 7527)	This is appendData, a member of class IDOMCharacterData.
	cloneNode (see page 7560)	Returns a duplicate of this node.
	deleteData (see page 7528)	This is deleteData, a member of class IDOMCharacterData.
	get_attributes (see page 7561)	Returns the value of the get_attributes property.
	get_childNodes (see page 7561)	Returns the value of the get_childNodes property.
	get_data (see page 7528)	This is get_data, a member of class IDOMCharacterData.
	get_firstChild (see page 7561)	Returns the value of the get_firstChild property.
	get_lastChild (see page 7561)	Returns the value of the get_lastChild property.

	get_length (see page 7528)	This is get_length, a member of class IDOMCharacterData.
	get_localName (see page 7562)	Returns the value of the get_localName property.
	get_namespaceURI (see page 7562)	Returns the value of the get_namespaceURI property.
	get_nextSibling (see page 7562)	Returns the value of the get_nextSibling property.
	get_nodeName (see page 7562)	Returns the value of the get_nodeName property.
	get_nodeType (see page 7563)	Returns the value of the get_nodeType property.
	get_nodeValue (see page 7563)	Returns the value of the get_nodeValue property.
	get_ownerDocument (see page 7563)	Returns the value of the get_ownerDocument property.
	get_parentNode (see page 7563)	Returns the value of the get_parentNode property.
	get_prefix (see page 7564)	Returns the value of the get_prefix property.
	get_previousSibling (see page 7564)	Returns the value of the get_previousSibling property.
	hasChildNodes (see page 7564)	Indicates whether the node has any child nodes.
	insertBefore (see page 7565)	Inserts a new child node before a specified child of the node.
	insertData (see page 7528)	This is insertData, a member of class IDOMCharacterData.
	normalize (see page 7565)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	removeChild (see page 7566)	Removes a child node from this node.
	replaceChild (see page 7566)	Replaces one of this node's children with a specified node.
	replaceData (see page 7528)	This is replaceData, a member of class IDOMCharacterData.
	set_data (see page 7529)	This is set_data, a member of class IDOMCharacterData.
	set_nodeValue (see page 7567)	Sets the value of the node.
	splitText (see page 7604)	This is splitText, a member of class IDOMText.
	substringData (see page 7529)	This is substringData, a member of class IDOMCharacterData.
	supports (see page 7567)	Indicates whether the node supports a specified feature.

Public Properties

	Name	Description
R	attributes (see page 7568)	Represents the attributes of an element node.
R	childNodes (see page 7568)	Represents the child nodes of the node.
	data (see page 7529)	This is data, a member of class IDOMCharacterData.
R	firstChild (see page 7569)	Represents the first child node of this node.
R	lastChild (see page 7569)	Represents the last child node of this node.
R	length (see page 7529)	This is length, a member of class IDOMCharacterData.
R	localName (see page 7570)	Indicates the name of the node without any namespace prefix (see page 7576).
R	namespaceURI (see page 7571)	Indicates the URI for the namespace used in the qualified node name.
R	nextSibling (see page 7571)	Represents the next child of this node's parent.
R	nodeName (see page 7572)	Indicates the name of the node.
R	nodeType (see page 7572)	Indicates the type of the node.
	nodeValue (see page 7574)	Indicates the value of the node.
R	ownerDocument (see page 7575)	Represents the interface for the document that contains this node.
R	parentNode (see page 7575)	Provides access to the interface of the node's parent node.
R	prefix (see page 7576)	Indicates the namespace prefix for the node.
R	previousSibling (see page 7576)	Represents the preceding child of this node's parent.

80.2.22.2 IDOMText Methods

The methods of the IDOMText class are listed here.

Public Methods

	Name	Description
	splitText (see page 7604)	This is splitText, a member of class IDOMText.

80.2.22.2.1 IDOMText.splitText Method

Pascal

```
function splitText(offset: Integer): IDOMText; safecall;
```

C++

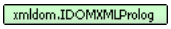
```
__fastcall IDOMText splitText(int offset);
```

Description

This is splitText, a member of class IDOMText.

80.2.23 IDOMXMLProlog Interface

Class Hierarchy



Pascal

```
IDOMXMLProlog = interface;
```

C++

```
__interface IDOMXMLProlog;
```

File

xmldom

Description

This is class xmldom.IDOMXMLProlog.

80.2.23.1 IDOMXMLProlog Members

The following tables list the members exposed by IDOMXMLProlog.

Public Methods

	Name	Description
	get_Encoding (see page 7605)	This is get_Encoding, a member of class IDOMXMLProlog.
	get_Standalone (see page 7605)	This is get_Standalone, a member of class IDOMXMLProlog.
	get_Version (see page 7605)	This is get_Version, a member of class IDOMXMLProlog.
	set_Encoding (see page 7605)	This is set_Encoding, a member of class IDOMXMLProlog.
	set_Standalone (see page 7606)	This is set_Standalone, a member of class IDOMXMLProlog.
	set_Version (see page 7606)	This is set_Version, a member of class IDOMXMLProlog.

Public Properties

	Name	Description
	Encoding (see page 7606)	This is Encoding, a member of class IDOMXMLProlog.
	Standalone (see page 7607)	This is Standalone, a member of class IDOMXMLProlog.
	Version (see page 7607)	This is Version, a member of class IDOMXMLProlog.

80.2.23.2 IDOMXMLProlog Methods

The methods of the IDOMXMLProlog class are listed here.

Public Methods

	Name	Description
	get_Encoding (see page 7605)	This is get_Encoding, a member of class IDOMXMLProlog.
	get_Standalone (see page 7605)	This is get_Standalone, a member of class IDOMXMLProlog.
	get_Version (see page 7605)	This is get_Version, a member of class IDOMXMLProlog.
	set_Encoding (see page 7605)	This is set_Encoding, a member of class IDOMXMLProlog.
	set_Standalone (see page 7606)	This is set_Standalone, a member of class IDOMXMLProlog.
	set_Version (see page 7606)	This is set_Version, a member of class IDOMXMLProlog.

80.2.23.2.1 IDOMXMLProlog.get_Encoding Method

Pascal

```
function get_Encoding: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Encoding();
```

Description

This is get_Encoding, a member of class IDOMXMLProlog.

80.2.23.2.2 IDOMXMLProlog.get_Standalone Method

Pascal

```
function get_Standalone: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Standalone();
```

Description

This is get_Standalone, a member of class IDOMXMLProlog.

80.2.23.2.3 IDOMXMLProlog.get_Version Method

Pascal

```
function get_Version: DOMString; safecall;
```

C++

```
__fastcall DOMString get_Version();
```

Description

This is get_Version, a member of class IDOMXMLProlog.

80.2.23.2.4 IDOMXMLProlog.set_Encoding Method

Pascal

```
procedure set_Encoding(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Encoding(const DOMString Value);
```

Description

This is set_Encoding, a member of class IDOMXMLProlog.

80.2.23.2.5 IDOMXMLProlog.set_Standalone Method

Pascal

```
procedure set_Standalone(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Standalone(const DOMString Value);
```

Description

This is set_Standalone, a member of class IDOMXMLProlog.

80.2.23.2.6 IDOMXMLProlog.set_Version Method

Pascal

```
procedure set_Version(const Value: DOMString); safecall;
```

C++

```
__fastcall set_Version(const DOMString Value);
```

Description

This is set_Version, a member of class IDOMXMLProlog.

80.2.23.3 IDOMXMLProlog Properties

The properties of the IDOMXMLProlog class are listed here.

Public Properties

	Name	Description
	Encoding (see page 7606)	This is Encoding, a member of class IDOMXMLProlog.
	Standalone (see page 7607)	This is Standalone, a member of class IDOMXMLProlog.
	Version (see page 7607)	This is Version, a member of class IDOMXMLProlog.

80.2.23.3.1 IDOMXMLProlog.Encoding Property

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

This is Encoding, a member of class IDOMXMLProlog.

80.2.23.3.2 IDOMXMLProlog.Standalone Property

Pascal

```
property Standalone: DOMString;
```

C++

```
__property DOMString Standalone;
```

Description

This is Standalone, a member of class IDOMXMLProlog.

80.2.23.3.3 IDOMXMLProlog.Version Property

Pascal

```
property Version: DOMString;
```

C++

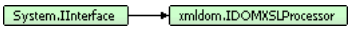
```
__property DOMString Version;
```

Description

This is Version, a member of class IDOMXMLProlog.

80.2.24 IDOMXSLProcessor Interface

Class Hierarchy



Pascal

```
IDOMXSLProcessor = interface(IInterface);
```

C++

```
__interface IDOMXSLProcessor : public IInterface;
```

File

xmldom

Description

This is class xmldom.IDOMXSLProcessor.

80.2.24.1 IDOMXSLProcessor Members

The following tables list the members exposed by IDOMXSLProcessor.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	Get_input (see page 7608)	This is Get_input, a member of class IDOMXSLProcessor.

	Get_output (see page 7608)	This is Get_output, a member of class IDOMXSLProcessor.
	Get_stylesheet (see page 7609)	This is Get_stylesheet, a member of class IDOMXSLProcessor.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	reset (see page 7609)	This is reset, a member of class IDOMXSLProcessor.
	Set_input (see page 7609)	This is Set_input, a member of class IDOMXSLProcessor.
	Set_output (see page 7609)	This is Set_output, a member of class IDOMXSLProcessor.
	setParameter (see page 7609)	This is setParameter, a member of class IDOMXSLProcessor.
	transform (see page 7610)	This is transform, a member of class IDOMXSLProcessor.

Public Properties

	Name	Description
	input (see page 7610)	This is input, a member of class IDOMXSLProcessor.
	output (see page 7610)	This is output, a member of class IDOMXSLProcessor.
R	stylesheet (see page 7611)	This is stylesheet, a member of class IDOMXSLProcessor.

80.2.24.2 IDOMXSLProcessor Methods

The methods of the IDOMXSLProcessor class are listed here.

Public Methods

	Name	Description
	Get_input (see page 7608)	This is Get_input, a member of class IDOMXSLProcessor.
	Get_output (see page 7608)	This is Get_output, a member of class IDOMXSLProcessor.
	Get_stylesheet (see page 7609)	This is Get_stylesheet, a member of class IDOMXSLProcessor.
	reset (see page 7609)	This is reset, a member of class IDOMXSLProcessor.
	Set_input (see page 7609)	This is Set_input, a member of class IDOMXSLProcessor.
	Set_output (see page 7609)	This is Set_output, a member of class IDOMXSLProcessor.
	setParameter (see page 7609)	This is setParameter, a member of class IDOMXSLProcessor.
	transform (see page 7610)	This is transform, a member of class IDOMXSLProcessor.

80.2.24.2.1 IDOMXSLProcessor.Get_input Method

Pascal

```
function Get_input: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_input();
```

Description

This is Get_input, a member of class IDOMXSLProcessor.

80.2.24.2.2 IDOMXSLProcessor.Get_output Method

Pascal

```
function Get_output: OleVariant; safecall;
```

C++

```
__fastcall OleVariant Get_output();
```

Description

This is Get_output, a member of class IDOMXSLProcessor.

80.2.24.2.3 IDOMXSLProcessor.Get_stylesheet Method

Pascal

```
function Get_stylesheet: IDOMNode; safecall;
```

C++

```
__fastcall IDOMNode Get_stylesheet();
```

Description

This is Get_stylesheet, a member of class IDOMXSLProcessor.

80.2.24.2.4 IDOMXSLProcessor.reset Method

Pascal

```
procedure reset; safecall;
```

C++

```
__fastcall reset();
```

Description

This is reset, a member of class IDOMXSLProcessor.

80.2.24.2.5 IDOMXSLProcessor.Set_input Method

Pascal

```
procedure Set_input(const value: OleVariant); safecall;
```

C++

```
__fastcall Set_input(const OleVariant value);
```

Description

This is Set_input, a member of class IDOMXSLProcessor.

80.2.24.2.6 IDOMXSLProcessor.Set_output Method

Pascal

```
procedure Set_output(const value: OleVariant); safecall;
```

C++

```
__fastcall Set_output(const OleVariant value);
```

Description

This is Set_output, a member of class IDOMXSLProcessor.

80.2.24.2.7 IDOMXSLProcessor.setParameter Method

Pascal

```
procedure setParameter(const Name: DOMString; Value: OleVariant; const namespaceURI:
DOMString); safecall;
```

C++

```
__fastcall setParameter(const DOMString Name, OleVariant Value, const DOMString namespaceURI);
```

Description

This is setParameter, a member of class IDOMXSLProcessor.

80.2.24.2.8 IDOMXSLProcessor.transform Method

Pascal

```
function transform: WordBool; safecall;
```

C++

```
__fastcall WordBool transform();
```

Description

This is transform, a member of class IDOMXSLProcessor.

80.2.24.3 IDOMXSLProcessor Properties

The properties of the IDOMXSLProcessor class are listed here.

Public Properties

	Name	Description
	input (see page 7610)	This is input, a member of class IDOMXSLProcessor.
	output (see page 7610)	This is output, a member of class IDOMXSLProcessor.
R	stylesheet (see page 7611)	This is stylesheet, a member of class IDOMXSLProcessor.

80.2.24.3.1 IDOMXSLProcessor.input Property

Pascal

```
property input: OleVariant;
```

C++

```
__property OleVariant input;
```

Description

This is input, a member of class IDOMXSLProcessor.

80.2.24.3.2 IDOMXSLProcessor.output Property

Pascal

```
property output: OleVariant;
```

C++

```
__property OleVariant output;
```

Description

This is output, a member of class IDOMXSLProcessor.

80.2.24.3.3 IDOMXSLProcessor.stylesheet Property

Pascal

```
property stylesheet: IDOMNode;
```

C++

```
__property IDOMNode stylesheet;
```

Description

This is stylesheet, a member of class IDOMXSLProcessor.

80.3 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DOMVendorNotSupported (see page 7612)	This is function xmldom.DOMVendorNotSupported.
	ExtractLocalName (see page 7612)	Strips the namespace prefix, if present, from the tag name of an XML node.
	ExtractPrefix (see page 7613)	Returns the namespace prefix of an XML node's tag name.
	GetDOM (see page 7614)	Returns the top-level DOM interface for a registered DOM implementation.
	GetDOMNodeEx (see page 7614)	This is function xmldom.GetDOMNodeEx.
	GetDOMVendor (see page 7615)	Returns the TDOMVendor (see page 7514) object for a specified DOM vendor.
	IsPrefixed (see page 7615)	Indicates whether an XML tag name includes a namespace prefix.
	MakeNodeName (see page 7616)	Assembles a full tag name from a local name and namespace prefix.
	NodeMatches (see page 7616)	Indicates whether a specified node matches a given namespace and tag name.
	RegisterDOMVendor (see page 7617)	Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.
	SameNamespace (see page 7617)	Indicates whether a specified node is defined within a specified namespace.
	SameNamespace (see page 7617)	Indicates whether a specified node is defined within a specified namespace.
	UnRegisterDOMVendor (see page 7618)	Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 7617) .

80.3.1 xmldom.DOMVendorNotSupported Function

Pascal

```
procedure DOMVendorNotSupported(const PropOrMethod: string; const VendorName: string);
```

C++

```
DOMVendorNotSupported(const UnicodeString PropOrMethod, const UnicodeString VendorName);
```

File

xmldom

Description

This is function xmldom.DOMVendorNotSupported.

80.3.2 xmldom.ExtractLocalName Function

Strips the namespace prefix, if present, from the tag name of an XML node.

Pascal

```
function ExtractLocalName(const AName: DOMString): DOMString;
```

C++

```
DOMString ExtractLocalName(const DOMString AName);
```

File

xmldom

Description

Call ExtractLocalName to convert a full tag name for an XML node into the corresponding local name.

AName is the tag name to be converted.

ExtractLocalName returns the value of AName with any namespace prefix stripped away.

See Also

ExtractPrefix (see page 7613)

IsPrefixed (see page 7615)

NodeMatches (see page 7616)

IXMLNode (see page 7669)

SameNamespace (see page 7617)

80.3.3 xmldom.ExtractPrefix Function

Returns the namespace prefix of an XML node's tag name.

Pascal

```
function ExtractPrefix(const AName: DOMString): DOMString;
```

C++

```
DOMString ExtractPrefix(const DOMString AName);
```

File

xmldom

Description

Call ExtractPrefix to determine the namespace prefix, if any, of a full tag name for an XML node.

AName is the tag name of the XML node.

ExtractPrefix returns the namespace prefix, not including the colon (:) that separates the namespace from the local name of the XML node. If the tag name does not include a namespace prefix, ExtractPrefix returns an empty string.

See Also

ExtractLocalName (see page 7612)

IsPrefixed (see page 7615)

SameNamespace (see page 7617)

NodeMatches (see page 7616)

80.3.4 xmldom.GetDOM Function

Returns the top-level DOM interface for a registered DOM implementation.

Pascal

```
function GetDOM(const VendorDesc: string = ''): IDOMImplementation;
```

C++

```
IDOMImplementation GetDOM(const UnicodeString VendorDesc = '');
```

File

xmldom

Description

Call GetDOM to obtain an IDOMImplementation (see page 7552) interface that you can use for parsing and editing XML documents. The IDOMImplementation (see page 7552) interface is the top-level DOM interface, which you can use to check for the presence of a specific feature or to obtain an interface for working with a DOM document.

VendorDesc is a short descriptive name that identifies the DOM implementation. This name is returned by the Description method of the DOM implementation factory that is used to register the implementation.

Note: The DOM implementation must be registered before you can access it using GetDOM. To register a DOM implementation, use the RegisterDOMVendor (see page 7617) procedure.

See Also

RegisterDOMVendor (see page 7617)

DOMVendors (see page 7622)

GetDOMVendor (see page 7615)

TXMLDocument (see page 7347)

Using the Document Object Model

80.3.5 xmldom.GetDOMNodeEx Function

Pascal

```
function GetDOMNodeEx(const Node: IDOMNode): IDOMNodeEx;
```

C++

```
IDOMNodeEx GetDOMNodeEx(const IDOMNode Node);
```

File

xmldom

Description

This is function xmldom.GetDOMNodeEx.

80.3.6 xmldom.GetDOMVendor Function

Returns the TDOMVendor (see page 7514) object for a specified DOM vendor.

Pascal

```
function GetDOMVendor(VendorDesc: string): TDOMVendor;
```

C++

```
TDOMVendor GetDOMVendor(UnicodeString VendorDesc);
```

File

xmldom

Description

Call GetDOMVendor to obtain the TDOMVendor (see page 7514) object that represents a registered DOM implementation. You can then assign this return value as the DOMVendor property of TXMLDocument, or call this object's DOMImplementation method to access the IDOMImplementation (see page 7552) interface for the DOM implementation.

VendorDesc is the descriptive string that identifies the particular DOM implementation that is desired. This is the value returned by the TDOMVendor (see page 7514) object's Description method.

See Also

RegisterDOMVendor (see page 7617)

DOMVendors (see page 7622)

TXMLDocument (see page 7347)

80.3.7 xmldom.IsPrefixed Function

Indicates whether an XML tag name includes a namespace prefix.

Pascal

```
function IsPrefixed(const AName: DOMString): Boolean;
```

C++

```
Boolean IsPrefixed(const DOMString AName);
```

File

xmldom

Description

Call IsPrefixed to determine whether a tag name includes a namespace prefix.

AName is the tag name of the XML node.

IsPrefixed returns true if AName includes a namespace prefix followed by the colon (:) separator character, false otherwise.

See Also

ExtractLocalName (see page 7612)

ExtractPrefix (see page 7613)

IXMLNode (see page 7669)

80.3.8 xmldom.MakeNodeName Function

Assembles a full tag name from a local name and namespace prefix.

Pascal

```
function MakeNodeName(const Prefix: DOMString; const LocalName: DOMString): DOMString;
```

C++

```
DOMString MakeNodeName(const DOMString Prefix, const DOMString LocalName);
```

File

xmldom

Description

Call MakeNodeName to create a tag name from a namespace prefix and a local tag name.

Prefix is the namespace prefix.

LocalName is the local name for the node.

MakeNodeName returns the full tag name that consists of the namespace prefix, followed by a colon and then the local name.

See Also

ExtractPrefix (see page 7613)

ExtractLocalName (see page 7612)

80.3.9 xmldom.NodeMatches Function

Indicates whether a specified node matches a given namespace and tag name.

Pascal

```
function NodeMatches(const Node: IDOMNode; const TagName: DOMString; const NamespaceURI:
DOMString): Boolean; overload;
```

C++

```
Boolean NodeMatches(const IDOMNode Node, const DOMString TagName, const DOMString
NamespaceURI);
```

File

xmldom

Description

Call NodeMatches to determine whether the node with the interface specified by Node refers to the same node as is described by the TagName and namespaceURI parameters.

NodeMatches returns true if

The value of Node's namespaceURI property is the same as the namespaceURI parameter.

The local name or tag name of Node is the same as the TagName parameter.

Tip: Given an IXMLNode interface, you can obtain the IDOMNode (see page 7557) interface required for the Node parameter by reading its DOMNode property.

See Also

ExtractLocalName (see page 7612)

ExtractPrefix (see page 7613)

IXMLNode (see page 7669)

SameNamespace (see page 7617)

80.3.10 xmldom.RegisterDOMVendor Function

Registers an implementation of the DOM interfaces so that it is available to the TXMLDocument class.

Pascal

```
procedure RegisterDOMVendor(const Vendor: TDOMVendor);
```

C++

```
RegisterDOMVendor(const TDOMVendor Vendor);
```

File

xmldom

Description

Call RegisterDOMVendor to register a DOM implementation that you want to use for parsing XML documents. A DOM (Document Object Model) implementation is a set of objects that implement the standard DOM interfaces.

If you are using another DOM implementation, you must create a TDOMVendor (see page 7514) descendant that can supply the IDOMImplementation (see page 7552) interface and description string for the DOM implementation. Pass an instance of this object as the Vendor parameter to RegisterDOMVendor.

See Also

UnRegisterDOMVendor (see page 7618)

TXMLDocument (see page 7347)

DOMVendors (see page 7622)

Using the Document Object Model

80.3.11 xmldom.SameNamespace Function

Indicates whether a specified node is defined within a specified namespace.

Pascal

```
function SameNamespace(const Node: IDOMNode; const namespaceURI: WideString): Boolean;
overload;
function SameNamespace(const URI1: WideString; const URI2: WideString): Boolean; overload;
```

C++

```
Boolean SameNamespace(const IDOMNode Node, const BSTR namespaceURI);
```

```
Boolean SameNamespace(const BSTR URI1, const BSTR URI2);
```

File

xmldom

Description

Call SameNamespace to determine whether the two parameters identify the same namespace.

Using the first syntax, the Node parameter identifies a node and the namespaceURI parameter a namespace. SameNamespace checks whether the node is defined in the specified namespace.

Using the second syntax, both parameters are namespace URI values. SameNamespace checks whether they both identify the same namespace.

Tip: Given an IXMLNode interface, you can obtain the IDOMNode (see page 7557) interface required for the Node parameter by reading its DOMNode property.

See Also

ExtractLocalName (see page 7612)

ExtractPrefix (see page 7613)

IXMLNode (see page 7669)

80.3.12 xmldom.UnRegisterDOMVendor Function

Unregisters a DOM implementation that was registered using RegisterDOMVendor (see page 7617) .

Pascal

```
procedure UnRegisterDOMVendor(const Vendor: TDOMVendor);
```

C++

```
UnRegisterDOMVendor(const TDOMVendor Vendor);
```

File

xmldom

Description

Call UnRegisterDOMVendor to unregister a DOM implementation when it is no longer available. After unregistering a DOM implementation, it can't be used by TXMLDocument for parsing XML documents.

Vendor is the TDOMVendor (see page 7514) descendant the represents the DOM implementation that is to be unregistered. It must be the same object used when registering the DOM implementation when you called RegisterDOMVendor (see page 7617).

See Also

RegisterDOMVendor (see page 7617)

TXMLDocument (see page 7347)

DOMVendors (see page 7622)

Using the Document Object Model

80.4 Types

The following table lists types in this documentation.

Types

Name	Description
DOMNodeType (see page 7619)	DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.
DOMString (see page 7619)	DOMString is the type for strings that are used by a DOM parser.
DOMTimeStamp (see page 7620)	This is type xmldom.DOMTimeStamp.
TAsyncEventHandler (see page 7620)	TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.
TDOMVendorArray (see page 7621)	This is type xmldom.TDOMVendorArray.

80.4.1 xmldom.DOMNodeType Type

DOMNodeType indicates the type of a node in an XML document, as parsed by a DOM parser.

Pascal

```
DOMNodeType = Word;
```

C++

```
Word DOMNodeType;
```

File

xmldom

Description

DOMNodeType is the type of the nodeType property on the IDOMNode (see page 7557) interface. It is a flag, defined by W3C, that indicates the role played by a node in the parsed version of an XML document.

See Also

IDOMNode (see page 7557)

80.4.2 xmldom.DOMString Type

DOMString is the type for strings that are used by a DOM parser.

Pascal

```
DOMString = UnicodeString;
```

C++

```
UnicodeString DOMString;
```

File

xmldom

Description

DOM implementations work with wide characters. The strings that are used, therefore, when working with DOM interfaces, are of type DOMString, which simply maps to the WideString (see page 3467) type.

80.4.3 xmldom.DOMTimeStamp Type

Pascal

```
DOMTimeStamp = Int64;
```

C++

```
Int64 DOMTimeStamp;
```

File

xmldom

Description

This is type xmldom.DOMTimeStamp.

80.4.4 xmldom.TAsyncEventHandler Type

TAsyncEventHandler is the type of event handlers that respond when the AsyncLoadState of an XML document changes.

Pascal

```
TAsyncEventHandler = procedure (Sender: TObject; AsyncLoadState: Integer) of object;
```

C++

```
(Sender: TObject; AsyncLoadState: Integer) ( TAsyncEventHandler)();
```

File

xmldom

Description

TAsyncEventHandler is the type for event handlers that respond at various stages during the asynchronous parsing of an XML document.

Sender is the TXMLDocument object that receives the event.

AsyncLoadState is an integer that indicates the current stage of the parsing process. Its value is vendor-specific, and corresponds to the document's AsyncLoadState property.

See Also

TXMLDocument (see page 7347)

IXMLDocument (see page 7640)

AsyncLoadState (see page 7382)

80.4.5 xmldom.TDOMVendorArray Type

Pascal

```
TDOMVendorArray = array of TDOMVendor;
```

C++

```
array of TDOMVendor TDOMVendorArray;
```

File

xmldom

Description

This is type xmldom.TDOMVendorArray.

80.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
DefaultDOMVendor (see page 7622)	Identifies the DOM implementation that is used by default.
DOMVendors (see page 7622)	Lists all registered DOM Vendors.

80.5.1 xmldom.DefaultDOMVendor Variable

Identifies the DOM implementation that is used by default.

Pascal

```
DefaultDOMVendor: string;
```

C++

```
UnicodeString DefaultDOMVendor;
```

File

xmldom

Description

DefaultDOMVendor is the descriptive string that was used to register the default DOM implementation. The default DOM implementation is used to parse an XML document when a vendor is not explicitly specified.

Tip: To obtain the IDOMImplementation (see page 7552) interface from DefaultDOMVendor, use the global GetDOMVendor (see page 7615) function.

Note: The value of DefaultDOMVendor should be the same as the value of the DefaultVendorName variable.

See Also

RegisterDOMVendor (see page 7617)

UnRegisterDOMVendor (see page 7618)

TXMLDocument (see page 7347)

GetDOMVendor (see page 7615)

DOMVendors (see page 7622)

80.5.2 xmldom.DOMVendors Variable

Lists all registered DOM Vendors.

Pascal

```
DOMVendors: TDOMVendorList;
```

C++

```
TDOMVendorList DOMVendors;
```

File

xmldom

Description

Use DOMVendors to determine all the DOM implementations currently registered with an application. For example, you can iterate through the Vendors property to obtain each registered TDOMVendor (see page 7514) object, which can then be used to extract its IDOMImplementation (see page 7552) interface or a descriptive string. Any of the TDOMVendor (see page 7514) objects in the list can be assigned as the DOMVendor property of an XML document component.

Do not add or delete vendors using this global variable: Instead, add and remove vendors by calling the RegisterDOMVendor (see page 7617) or UnRegisterDOMVendor (see page 7618) procedure.

See Also

RegisterDOMVendor (see page 7617)

UnRegisterDOMVendor (see page 7618)

TDOMVendor (see page 7514)

TXMLDocument (see page 7347)

GetDOMVendor (see page 7615)

DefaultDOMVendor (see page 7622)

80.6 Constants

The following table lists constants in this documentation.

Constants

Name	Description
ATTRIBUTE_NODE (see page 7624)	This is constant xmldom.ATTRIBUTE_NODE.
CDATA_SECTION_NODE (see page 7625)	This is constant xmldom.CDATA_SECTION_NODE.
COMMENT_NODE (see page 7625)	This is constant xmldom.COMMENT_NODE.
DOCUMENT_FRAGMENT_NODE (see page 7625)	This is constant xmldom.DOCUMENT_FRAGMENT_NODE.
DOCUMENT_NODE (see page 7626)	This is constant xmldom.DOCUMENT_NODE.
DOCUMENT_TYPE_NODE (see page 7626)	This is constant xmldom.DOCUMENT_TYPE_NODE.
DOMSTRING_SIZE_ERR (see page 7626)	This is constant xmldom.DOMSTRING_SIZE_ERR.
DOMWrapperVersion (see page 7627)	Indicates the version number for the vendor-neutral DOM interfaces.
ELEMENT_NODE (see page 7627)	This is constant xmldom.ELEMENT_NODE.
ENTITY_NODE (see page 7627)	This is constant xmldom.ENTITY_NODE.
ENTITY_REFERENCE_NODE (see page 7628)	This is constant xmldom.ENTITY_REFERENCE_NODE.
HIERARCHY_REQUEST_ERR (see page 7628)	This is constant xmldom.HIERARCHY_REQUEST_ERR.
INDEX_SIZE_ERR (see page 7628)	This is constant xmldom.INDEX_SIZE_ERR.
INUSE_ATTRIBUTE_ERR (see page 7629)	This is constant xmldom.INUSE_ATTRIBUTE_ERR.
INVALID_ACCESS_ERR (see page 7629)	This is constant xmldom.INVALID_ACCESS_ERR.
INVALID_CHARACTER_ERR (see page 7629)	This is constant xmldom.INVALID_CHARACTER_ERR.
INVALID_MODIFICATION_ERR (see page 7630)	This is constant xmldom.INVALID_MODIFICATION_ERR.
INVALID_STATE_ERR (see page 7630)	This is constant xmldom.INVALID_STATE_ERR.
NAMESPACE_ERR (see page 7630)	This is constant xmldom.NAMESPACE_ERR.
NO_DATA_ALLOWED_ERR (see page 7631)	This is constant xmldom.NO_DATA_ALLOWED_ERR.
NO_MODIFICATION_ALLOWED_ERR (see page 7631)	This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.
NOT_FOUND_ERR (see page 7631)	This is constant xmldom.NOT_FOUND_ERR.
NOT_SUPPORTED_ERR (see page 7632)	This is constant xmldom.NOT_SUPPORTED_ERR.
NOTATION_NODE (see page 7632)	This is constant xmldom.NOTATION_NODE.
NSDelim (see page 7632)	This is constant xmldom.NSDelim.
PROCESSING_INSTRUCTION_NODE (see page 7633)	This is constant xmldom.PROCESSING_INSTRUCTION_NODE.
SEncoding (see page 7633)	This is constant xmldom.SEncoding.
SHttp (see page 7633)	This is constant xmldom.SHttp.
SStandalone (see page 7634)	This is constant xmldom.SStandalone.
SVersion (see page 7634)	This is constant xmldom.SVersion.
SXML (see page 7634)	This is constant xmldom.SXML.
SXMLNamespaceURI (see page 7635)	This is constant xmldom.SXMLNamespaceURI.
SXMLNS (see page 7635)	This is constant xmldom.SXMLNS.
SXMLPrefixNamespaceURI (see page 7635)	This is constant xmldom.SXMLPrefixNamespaceURI.
SYNTAX_ERR (see page 7636)	This is constant xmldom.SYNTAX_ERR.
TEXT_NODE (see page 7636)	This is constant xmldom.TEXT_NODE.
WRONG_DOCUMENT_ERR (see page 7636)	This is constant xmldom.WRONG_DOCUMENT_ERR.

80.6.1 xmldom.ATTRIBUTE_NODE Constant

Pascal

```
ATTRIBUTE_NODE = 2;
```

C++

```
ATTRIBUTE_NODE = 2;
```

File

xmldom

Description

This is constant xmldom.ATTRIBUTE_NODE.

80.6.2 xmldom.CDATA_SECTION_NODE Constant

Pascal

```
CDATA_SECTION_NODE = 4;
```

C++

```
CDATA_SECTION_NODE = 4;
```

File

xmldom

Description

This is constant xmldom.CDATA_SECTION_NODE.

80.6.3 xmldom.COMMENT_NODE Constant

Pascal

```
COMMENT_NODE = 8;
```

C++

```
COMMENT_NODE = 8;
```

File

xmldom

Description

This is constant xmldom.COMMENT_NODE.

80.6.4 xmldom.DOCUMENT_FRAGMENT_NODE Constant

Pascal

```
DOCUMENT_FRAGMENT_NODE = 11;
```

C++

```
DOCUMENT_FRAGMENT_NODE = 11;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_FRAGMENT_NODE.

80.6.5 xmldom.DOCUMENT_NODE Constant

Pascal

```
DOCUMENT_NODE = 9;
```

C++

```
DOCUMENT_NODE = 9;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_NODE.

80.6.6 xmldom.DOCUMENT_TYPE_NODE Constant

Pascal

```
DOCUMENT_TYPE_NODE = 10;
```

C++

```
DOCUMENT_TYPE_NODE = 10;
```

File

xmldom

Description

This is constant xmldom.DOCUMENT_TYPE_NODE.

80.6.7 xmldom.DOMSTRING_SIZE_ERR Constant

Pascal

```
DOMSTRING_SIZE_ERR = 2;
```

C++

```
DOMSTRING_SIZE_ERR = 2;
```

File

xmldom

Description

This is constant xmldom.DOMSTRING_SIZE_ERR.

80.6.8 xmldom.DOMWrapperVersion Constant

Indicates the version number for the vendor-neutral DOM interfaces.

Pascal

```
DOMWrapperVersion = 1.3;
```

C++

```
DOMWrapperVersion = 1.3;
```

File

xmldom

Description

DOMWrapperVersion specifies the version number of the DOM interfaces (IDOMImplementation (see page 7552), IDOMNode (see page 7557), IDOMElement (see page 7542), and so on).

80.6.9 xmldom.ELEMENT_NODE Constant

Pascal

```
ELEMENT_NODE = 1;
```

C++

```
ELEMENT_NODE = 1;
```

File

xmldom

Description

This is constant xmldom.ELEMENT_NODE.

80.6.10 xmldom.ENTITY_NODE Constant

Pascal

```
ENTITY_NODE = 6;
```

C++

```
ENTITY_NODE = 6;
```

File

xmldom

Description

This is constant xmldom.ENTITY_NODE.

80.6.11 xmldom.ENTITY_REFERENCE_NODE Constant

Pascal

```
ENTITY_REFERENCE_NODE = 5;
```

C++

```
ENTITY_REFERENCE_NODE = 5;
```

File

xmldom

Description

This is constant xmldom.ENTITY_REFERENCE_NODE.

80.6.12 xmldom.HIERARCHY_REQUEST_ERR Constant

Pascal

```
HIERARCHY_REQUEST_ERR = 3;
```

C++

```
HIERARCHY_REQUEST_ERR = 3;
```

File

xmldom

Description

This is constant xmldom.HIERARCHY_REQUEST_ERR.

80.6.13 xmldom.INDEX_SIZE_ERR Constant

Pascal

```
INDEX_SIZE_ERR = 1;
```

C++

```
INDEX_SIZE_ERR = 1;
```

File

xmldom

Description

This is constant xmldom.INDEX_SIZE_ERR.

80.6.14 xmldom.INUSE_ATTRIBUTE_ERR Constant

Pascal

```
INUSE_ATTRIBUTE_ERR = 10;
```

C++

```
INUSE_ATTRIBUTE_ERR = 10;
```

File

xmldom

Description

This is constant xmldom.INUSE_ATTRIBUTE_ERR.

80.6.15 xmldom.INVALID_ACCESS_ERR Constant

Pascal

```
INVALID_ACCESS_ERR = 15;
```

C++

```
INVALID_ACCESS_ERR = 15;
```

File

xmldom

Description

This is constant xmldom.INVALID_ACCESS_ERR.

80.6.16 xmldom.INVALID_CHARACTER_ERR Constant

Pascal

```
INVALID_CHARACTER_ERR = 5;
```

C++

```
INVALID_CHARACTER_ERR = 5;
```

File

xmldom

Description

This is constant xmldom.INVALID_CHARACTER_ERR.

80.6.17 xmldom.INVALID_MODIFICATION_ERR Constant

Pascal

```
INVALID_MODIFICATION_ERR = 13;
```

C++

```
INVALID_MODIFICATION_ERR = 13;
```

File

xmldom

Description

This is constant xmldom.INVALID_MODIFICATION_ERR.

80.6.18 xmldom.INVALID_STATE_ERR Constant

Pascal

```
INVALID_STATE_ERR = 11;
```

C++

```
INVALID_STATE_ERR = 11;
```

File

xmldom

Description

This is constant xmldom.INVALID_STATE_ERR.

80.6.19 xmldom.NAMESPACE_ERR Constant

Pascal

```
NAMESPACE_ERR = 14;
```

C++

```
NAMESPACE_ERR = 14;
```

File

xmldom

Description

This is constant xmldom.NAMESPACE_ERR.

80.6.20 xmldom.NO_DATA_ALLOWED_ERR Constant

Pascal

```
NO_DATA_ALLOWED_ERR = 6;
```

C++

```
NO_DATA_ALLOWED_ERR = 6;
```

File

xmldom

Description

This is constant xmldom.NO_DATA_ALLOWED_ERR.

80.6.21 xmldom.NO_MODIFICATION_ALLOWED_ERR Constant

Pascal

```
NO_MODIFICATION_ALLOWED_ERR = 7;
```

C++

```
NO_MODIFICATION_ALLOWED_ERR = 7;
```

File

xmldom

Description

This is constant xmldom.NO_MODIFICATION_ALLOWED_ERR.

80.6.22 xmldom.NOT_FOUND_ERR Constant

Pascal

```
NOT_FOUND_ERR = 8;
```

C++

```
NOT_FOUND_ERR = 8;
```

File

xmldom

Description

This is constant xmldom.NOT_FOUND_ERR.

80.6.23 xmldom.NOT_SUPPORTED_ERR Constant

Pascal

```
NOT_SUPPORTED_ERR = 9;
```

C++

```
NOT_SUPPORTED_ERR = 9;
```

File

xmldom

Description

This is constant xmldom.NOT_SUPPORTED_ERR.

80.6.24 xmldom.NOTATION_NODE Constant

Pascal

```
NOTATION_NODE = 12;
```

C++

```
NOTATION_NODE = 12;
```

File

xmldom

Description

This is constant xmldom.NOTATION_NODE.

80.6.25 xmldom.NSDelim Constant

Pascal

```
NSDelim = ':';
```

C++

```
NSDelim = ':';
```

File

xmldom

Description

This is constant xmldom.NSDelim.

80.6.26 xmldom.PROCESSING_INSTRUCTION_NODE Constant

Pascal

```
PROCESSING_INSTRUCTION_NODE = 7;
```

C++

```
PROCESSING_INSTRUCTION_NODE = 7;
```

File

xmldom

Description

This is constant xmldom.PROCESSING_INSTRUCTION_NODE.

80.6.27 xmldom.SEncoding Constant

Pascal

```
SEncoding = 'encoding';
```

C++

```
SEncoding = 'encoding';
```

File

xmldom

Description

This is constant xmldom.SEncoding.

80.6.28 xmldom.SHttp Constant

Pascal

```
SHttp = 'http:/';
```

C++

```
SHttp = 'http:/';
```

File

xmldom

Description

This is constant xmldom.SHttp.

80.6.29 xmldom.SStandalone Constant

Pascal

```
SStandalone = 'standalone';
```

C++

```
SStandalone = 'standalone';
```

File

xmldom

Description

This is constant xmldom.SStandalone.

80.6.30 xmldom.SVersion Constant

Pascal

```
SVersion = 'version';
```

C++

```
SVersion = 'version';
```

File

xmldom

Description

This is constant xmldom.SVersion.

80.6.31 xmldom.SXML Constant

Pascal

```
SXML = 'xml';
```

C++

```
SXML = 'xml';
```

File

xmldom

Description

This is constant xmldom.SXML.

80.6.32 xmldom.SXMLNamespaceURI Constant

Pascal

```
SXMLNamespaceURI = SHttp+'/www.w3.org/2000/xmlns/';
```

C++

```
SXMLNamespaceURI = SHttp+'/www.w3.org/2000/xmlns/';
```

File

xmldom

Description

This is constant xmldom.SXMLNamespaceURI.

80.6.33 xmldom.SXMLNS Constant

Pascal

```
SXMLNS = 'xmlns';
```

C++

```
SXMLNS = 'xmlns';
```

File

xmldom

Description

This is constant xmldom.SXMLNS.

80.6.34 xmldom.SXMLPrefixNamespaceURI Constant

Pascal

```
SXMLPrefixNamespaceURI = SHttp+'/www.w3.org/XML/1998/namespace';
```

C++

```
SXMLPrefixNamespaceURI = SHttp+'/www.w3.org/XML/1998/namespace';
```

File

xmldom

Description

This is constant xmldom.SXMLPrefixNamespaceURI.

80.6.35 xmldom.SYNTAX_ERR Constant

Pascal

```
SYNTAX_ERR = 12;
```

C++

```
SYNTAX_ERR = 12;
```

File

xmldom

Description

This is constant xmldom.SYNTAX_ERR.

80.6.36 xmldom.TEXT_NODE Constant

Pascal

```
TEXT_NODE = 3;
```

C++

```
TEXT_NODE = 3;
```

File

xmldom

Description

This is constant xmldom.TEXT_NODE.

80.6.37 xmldom.WRONG_DOCUMENT_ERR Constant

Pascal

```
WRONG_DOCUMENT_ERR = 4;
```

C++

```
WRONG_DOCUMENT_ERR = 4;
```

File

xmldom

Description

This is constant xmldom.WRONG_DOCUMENT_ERR.

81 XMLIntf Namespace

This is namespace XMLIntf.

Topics

Name	Description
Classes (see page 7639)	The following table lists classes in this documentation.
Interfaces (see page 7640)	The following table lists interfaces in this documentation.
Structs, Records, Enums (see page 7716)	The following table lists structs, records, enums in this documentation.
Types (see page 7721)	The following table lists types in this documentation.
Constants (see page 7723)	The following table lists constants in this documentation.

Classes

	Name	Description
	EXMLDocError (see page 7639)	EXMLDocError is the exception class for errors editing or parsing an XML document.

Constants

Name	Description
NodeTypeNames (see page 7723)	This is constant XMLIntf.NodeTypeNames.

Interfaces

	Name	Description
	IXMLDocument (see page 7640)	IXMLDocument represents an XML (see page 7668) document.
	IXMLNode (see page 7669)	IXMLNode is the interface for a node in an XML (see page 7697) document.
	IXMLNodeCollection (see page 7698)	IXMLNodeCollection is the interface for a set of child nodes that are all the same type.
	IXMLNodeList (see page 7703)	IXMLNodeList represents a set of nodes in an XML document.

Structs, Records, Enums

	Name	Description
	TNodeType (see page 7716)	TNodeType identifies the type of a node in an XML document.
	TParseOption (see page 7718)	TParseOption and TParseOptions (see page 7721) represent implementation-specific options that are followed when parsing the XML document.
	TXMLDocOption (see page 7718)	TXMLDocOption and TXMLDocOptions (see page 7722) indicate types of information that an XML document automatically supplies.
	TXMLEncodingType (see page 7719)	TXMLEncodingType indicates the character set used to encode characters in an XML document.

Types

Name	Description
TParseOptions (see page 7721)	TParseOptions and TParseOption (see page 7718) represent implementation-specific options that are followed when parsing the XML document.
TXMLDocOptions (see page 7722)	This is type XMLIntf.TXMLDocOptions.

81.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXMLDocError (see page 7639)	EXMLDocError is the exception class for errors editing or parsing an XML document.

81.1.1 EXMLDocError Class

EXMLDocError is the exception class for errors editing or parsing an XML document.

Class Hierarchy

Pascal

```
EXMLDocError = class(Exception);
```

C++

```
class EXMLDocError : public Exception;
```

File

XMLIntf

Description

EXMLDocError is raised by TXMLDocument or TXMLNode when

An error occurs when loading an XML document.

An error occurs when creating a node in an XML document.

A required DOM interface is not found.

An option is requested that is not supported by the DOM implementation.

An attempt is made to perform an action that is not consistent with the specified node type.

See Also

XMLDocError (see page 7492)

TXMLDocument (see page 7347)

TXMLNode (see page 7396)

81.2 Interfaces

The following table lists interfaces in this documentation.

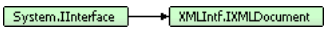
Interfaces

	Name	Description
	IXMLDocument (see page 7640)	IXMLDocument represents an XML (see page 7668) document.
	IXMLNode (see page 7669)	IXMLNode is the interface for a node in an XML (see page 7697) document.
	IXMLNodeCollection (see page 7698)	IXMLNodeCollection is the interface for a set of child nodes that are all the same type.
	IXMLNodeList (see page 7703)	IXMLNodeList represents a set of nodes in an XML document.

81.2.1 IXMLDocument Interface

IXMLDocument represents an XML (see page 7668) document.

Class Hierarchy



Pascal

```
IXMLDocument = interface(IInterface);
```

C++

```
__interface IXMLDocument : public IInterface;
```

File

XMLIntf

Description

Use IXMLDocument to work with an XML (see page 7668) document. Applications can obtain an IXMLDocument interface through a TXMLDocument component, using the global LoadXMLDocument function, or through the OwnerDocument property of any node in the document.

Like the related DOM interface IDOMDocument (see page 7530), you can use IXMLDocument to access the contents of an XML (see page 7668) document and to create new nodes. In addition, IXMLDocument lets you load an XML (see page 7668) document that is stored on disk or assign a string of raw (unparsed) XML (see page 7668) as the contents of the document.

TXMLDocument implements the IXMLDocument interface. Typically, the TXMLDocument component is the starting point for working with an XML (see page 7668) document.

Note: C++ method declarations that use IXMLDocument use the _di_IXMLDocument type instead. This type is a DelphiInterface (see page 3256) wrapper around the IXMLDocument interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLDocument >  _di_IXMLDocument;
```

See Also

IXMLNode (see page 7669)

EXMLDocError (see page 7639)

TXMLDocument (see page 7347)

LoadXMLDocument (see page 7491)

81.2.1.1 IXMLDocument Members

The following tables list the members exposed by IXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7643)	This is the overview for the AddChild method overload.
	CreateElement (see page 7644)	Creates a new element node that is associated with this document.
	CreateNode (see page 7645)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7646)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7646)	Returns the value of the Active (see page 7661) property.
	GetAsyncLoadState (see page 7647)	Returns the value of the AsyncLoadState (see page 7662) property.
	GetChildNodes (see page 7647)	Returns the value of the ChildNodes (see page 7663) property.
	GetDocBinding (see page 7647)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7648)	Returns the value of the DocumentElement (see page 7663) property.
	GetDocumentNode (see page 7648)	Returns the value of the Node (see page 7665) property.
	GetDOMDocument (see page 7648)	Returns the value of the DOMDocument (see page 7663) property.
	GetEncoding (see page 7649)	Returns the value of the Encoding (see page 7664) property.
	GetFileName (see page 7649)	Returns the value of the FileName (see page 7664) property.
	GetModified (see page 7649)	Returns the value of the Modified (see page 7665) property.
	GetNodeIndentStr (see page 7650)	Returns the value of the NodeIndentStr (see page 7666) property.
	GetOptions (see page 7650)	Returns the value of the Options (see page 7666) property.
	GetParseOptions (see page 7650)	Returns the value of the ParseOptions (see page 7667) property.
	GetSchemaRef (see page 7650)	Returns the value of the SchemaRef (see page 7667) property.
	GetStandAlone (see page 7651)	Returns the value of the StandAlone (see page 7668) property.
	GetVersion (see page 7651)	Returns the value of the Version (see page 7668) property.
	GetXML (see page 7651)	Returns the value of the XML (see page 7668) property.
	IsEmptyDoc (see page 7652)	Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.
	LoadFromFile (see page 7652)	Loads an XML (see page 7668) document from disk.
	LoadFromStream (see page 7653)	Loads an XML (see page 7668) document from a stream and activates it.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7654)	Updates the parsed XML (see page 7668) document to reflect external changes.
	RegisterDocBinding (see page 7654)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7655)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7655)	Saves the XML (see page 7668) document to disk.
	SaveToStream (see page 7656)	Saves the XML (see page 7668) document to a stream.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7657)	Sets the value of the Active (see page 7661) property.
	SetDocumentElement (see page 7657)	Sets the value of the DocumentElement (see page 7663) property.
	SetDOMDocument (see page 7658)	Sets the value of the DOMDocument (see page 7663) property.
	SetEncoding (see page 7658)	Sets the value of the Encoding (see page 7664) property.

	SetFileName (see page 7658)	Sets the value of the FileName (see page 7664) property.
	SetNodeIndentStr (see page 7659)	Sets the value of the NodeIndentStr (see page 7666) property.
	SetOnAsyncLoad (see page 7659)	Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.
	SetOptions (see page 7659)	Sets the value of the Options (see page 7666) property.
	SetParseOptions (see page 7660)	Sets the value of the ParseOptions (see page 7667) property.
	SetStandAlone (see page 7660)	Sets the value of the StandAlone (see page 7668) property.
	SetVersion (see page 7660)	Sets the value of the Version (see page 7668) property.
	SetXML (see page 7661)	Sets the value of the XML (see page 7668) property.

Public Properties

	Name	Description
	Active (see page 7661)	Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7662)	Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.
R	ChildNodes (see page 7663)	Lists all of the child nodes of the document.
	DocumentElement (see page 7663)	Provides access to the root node of the XML (see page 7668) document.
	DOMDocument (see page 7663)	Provides access to the DOM interface for the parsed XML (see page 7668) document.
	Encoding (see page 7664)	Indicates the character set that the XML (see page 7668) document uses to encode characters.
	FileName (see page 7664)	Indicates the XML (see page 7668) document file, if any, that this interface represents.
R	Modified (see page 7665)	Indicates whether the XML (see page 7668) document has been modified.
R	Node (see page 7665)	Provides access to the document node for the XML (see page 7668) document.
	NodeIndentStr (see page 7666)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.
	Options (see page 7666)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7667)	Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.
R	SchemaRef (see page 7667)	Returns the name of the schema associated with this XML (see page 7668) document.
	StandAlone (see page 7668)	Indicates whether the XML (see page 7668) document includes any external markup declarations.
	Version (see page 7668)	Indicates the XML (see page 7668) version that the document uses.
	XML (see page 7668)	Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.

81.2.1.2 IXMLDocument Methods

The methods of the IXMLDocument class are listed here.

Public Methods

	Name	Description
	AddChild (see page 7643)	This is the overview for the AddChild method overload.
	CreateElement (see page 7644)	Creates a new element node that is associated with this document.
	CreateNode (see page 7645)	Creates a new node that is associated with this document.
	GeneratePrefix (see page 7646)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7646)	Returns the value of the Active (see page 7661) property.
	GetAsyncLoadState (see page 7647)	Returns the value of the AsyncLoadState (see page 7662) property.
	GetChildNodes (see page 7647)	Returns the value of the ChildNodes (see page 7663) property.
	GetDocBinding (see page 7647)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7648)	Returns the value of the DocumentElement (see page 7663) property.

	GetDocumentNode (see page 7648)	Returns the value of the Node (see page 7665) property.
	GetDOMDocument (see page 7648)	Returns the value of the DOMDocument (see page 7663) property.
	GetEncoding (see page 7649)	Returns the value of the Encoding (see page 7664) property.
	GetFileName (see page 7649)	Returns the value of the FileName (see page 7664) property.
	GetModified (see page 7649)	Returns the value of the Modified (see page 7665) property.
	GetNodeIndentStr (see page 7650)	Returns the value of the NodeIndentStr (see page 7666) property.
	GetOptions (see page 7650)	Returns the value of the Options (see page 7666) property.
	GetParseOptions (see page 7650)	Returns the value of the ParseOptions (see page 7667) property.
	GetSchemaRef (see page 7650)	Returns the value of the SchemaRef (see page 7667) property.
	GetStandAlone (see page 7651)	Returns the value of the StandAlone (see page 7668) property.
	GetVersion (see page 7651)	Returns the value of the Version (see page 7668) property.
	GetXML (see page 7651)	Returns the value of the XML (see page 7668) property.
	IsEmptyDoc (see page 7652)	Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.
	LoadFromFile (see page 7652)	Loads an XML (see page 7668) document from disk.
	LoadFromStream (see page 7653)	Loads an XML (see page 7668) document from a stream and activates it.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
	Refresh (see page 7654)	Updates the parsed XML (see page 7668) document to reflect external changes.
	RegisterDocBinding (see page 7654)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7655)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7655)	Saves the XML (see page 7668) document to disk.
	SaveToStream (see page 7656)	Saves the XML (see page 7668) document to a stream.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7657)	Sets the value of the Active (see page 7661) property.
	SetDocumentElement (see page 7657)	Sets the value of the DocumentElement (see page 7663) property.
	SetDOMDocument (see page 7658)	Sets the value of the DOMDocument (see page 7663) property.
	SetEncoding (see page 7658)	Sets the value of the Encoding (see page 7664) property.
	SetFileName (see page 7658)	Sets the value of the FileName (see page 7664) property.
	SetNodeIndentStr (see page 7659)	Sets the value of the NodeIndentStr (see page 7666) property.
	SetOnAsyncLoad (see page 7659)	Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.
	SetOptions (see page 7659)	Sets the value of the Options (see page 7666) property.
	SetParseOptions (see page 7660)	Sets the value of the ParseOptions (see page 7667) property.
	SetStandAlone (see page 7660)	Sets the value of the StandAlone (see page 7668) property.
	SetVersion (see page 7660)	Sets the value of the Version (see page 7668) property.
	SetXML (see page 7661)	Sets the value of the XML (see page 7668) property.

81.2.1.2.1 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	IXMLDocument.AddChild (DOMString) (see page 7643)	Creates and adds a node to the child nodes of this document.
	IXMLDocument.AddChild (DOMString, DOMString) (see page 7643)	Creates and adds a node to the child nodes of this document.

81.2.1.2.1.1 IXMLDocument.AddChild Method (DOMString)

Creates and adds a node to the child nodes of this document.

Pascal

```
function AddChild(const TagName: DOMString): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI);
```

Description

Call AddChild to add a new child element node to the document itself. The new node is added to the end of the list maintained by the ChildNodes (see page 7663) property.

TagName is the tag name for the new element node.

NamespaceURI identifies the namespace, if any, that includes the new node's definition.

AddChild returns the interface for the newly created node.

Note: Do not call AddChild to add a child to the document element of this document. When adding data to the XML (see page 7668) document, use the AddChild method of the document element or of the node in the hierarchy that should be the parent of the new node.

See Also

AddChild (see page 7672)

CreateNode (see page 7645)

CreateElement (see page 7644)

ChildNodes (see page 7663)

DocumentElement (see page 7663)

81.2.1.2.2 IXMLDocument.CreateElement Method

Creates a new element node that is associated with this document.

Pascal

```
function CreateElement(const TagOrData: DOMString; const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateElement(const DOMString TagOrData, const DOMString NamespaceURI);
```

Description

Call CreateElement to create a new generic element node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7663) list of any node in the document (including this document's ChildNodes (see page 7663) property). It can also be assigned to the DocumentElement (see page 7663) property to replace the entire data hierarchy of this document.

TagOrData provides the tag name of the newly created node.

NamespaceURI identifies the namespace that includes the new node's definition.

CreateElement returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7643) method.

See Also

AddChild (see page 7672)

ChildNodes (see page 7663)

DocumentElement (see page 7663)

AddChild (see page 7643)

CreateNode (see page 7645)

81.2.1.2.3 IXMLDocument.CreateNode Method

Creates a new node that is associated with this document.

Pascal

```
function CreateNode(const NameOrData: DOMString; NodeType: TNodeType = ntElement; const
AddlData: DOMString = ''): IXMLNode;
```

C++

```
__fastcall IXMLNode CreateNode(const DOMString NameOrData, TNodeType NodeType = ntElement,
const DOMString AddlData = '');
```

Description

Call CreateNode to create a new generic XML (see page 7668) node. The resulting node does not have a parent, but can be added to the ChildNodes (see page 7663) or AttributeNodes list of any node in the document (including this document's ChildNodes (see page 7663) property).

NameOrData provides the tag name or value of the newly created node. Its interpretation depends on the type of node created, as indicated in the following table:

NodeType	NameOrData
ntElement	The tag name.
ntAttribute	The attribute name.
ntText	The value of the node.
ntCData	The value of the CDATA section.
ntEntityRef	The name of the referenced entity.
ntProcessingInstr	The target of the processing instruction.
ntComment	The value (text) of the comment.
ntDocFragment	Not used.
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.
ntProcessingInstr	The content of the processing instruction, except for the target.

NodeType indicates the type of node to create. It can only be one of the types listed in the previous table.

The meaning of AddlData depends on the node type, as indicated in the following table:

NodeType	AddlData
ntElement	The namespace URI that qualifies the tag name.
ntAttribute	The namespace URI that qualifies the attribute name.

ntProcessingInstr	The content of the processing instruction, except for the target.

CreateNode returns the interface for the new node.

Note: To add a node as the child of another node in this document, it is simpler to call that node's AddChild (see page 7643) method.

See Also

AddChild (see page 7672)

ChildNodes (see page 7663)

AttributeNodes (see page 7688)

DocumentElement (see page 7663)

CreateElement (see page 7644)

AddChild (see page 7643)

81.2.1.2.4 IXMLDocument.GeneratePrefix Method

Generates a unique namespace prefix that can be used for a node in the document.

Pascal

```
function GeneratePrefix(const Node: IXMLNode): DOMString;
```

C++

```
__fastcall DOMString GeneratePrefix(const IXMLNode Node);
```

Description

The AddChild (see page 7643) method for each node allows the caller to specify that the new node should use an automatically-generated namespace prefix. When this option is specified, the node calls GeneratePrefix to generate a unique string to act as a namespace prefix.

Node (see page 7665) is the interface for the node which will use the newly-generated namespace prefix.

GeneratePrefix returns a unique string that is not already used in the document to represent a namespace URI. The generated namespace prefixes are only unique within a single document.

See Also

AddChild (see page 7672)

81.2.1.2.5 IXMLDocument.GetActive Method

Returns the value of the Active (see page 7661) property.

Pascal

```
function GetActive: Boolean;
```

C++

```
__fastcall Boolean GetActive();
```

Description

GetActive is the read implementation of the Active (see page 7661) property.

See Also

Active (see page 7661)

81.2.1.2.6 IXMLDocument.GetAsyncLoadState Method

Returns the value of the AsyncLoadState (see page 7662) property.

Pascal

```
function GetAsyncLoadState: Integer;
```

C++

```
__fastcall int GetAsyncLoadState();
```

Description

GetAsyncLoadState is the read implementation of the AsyncLoadState (see page 7662) property.

See Also

AsyncLoadState (see page 7662)

81.2.1.2.7 IXMLDocument.GetChildNodes Method

Returns the value of the ChildNodes (see page 7663) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes (see page 7663) property.

See Also

ChildNodes (see page 7663)

81.2.1.2.8 IXMLDocument.GetDocBinding Method

Replaces a specified child node with an instance of a specified implementation class.

Pascal

```
function GetDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI: DOMString
= ''): IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call GetDocBinding to specify the implementation class for one of the child nodes of the document node. After replacing the specified node with an instance of the specified class, GetDocBinding returns the interface of the document element.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7663)

RegisterDocBinding (see page 7654)

ChildNodes (see page 7663)

81.2.1.2.9 IXMLDocument.GetDocumentElement Method

Returns the value of the DocumentElement (see page 7663) property.

Pascal

```
function GetDocumentElement: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentElement();
```

Description

GetDocumentElement is the read implementation of the DocumentElement (see page 7663) property.

See Also

DocumentElement (see page 7663)

81.2.1.2.10 IXMLDocument.GetDocumentNode Method

Returns the value of the Node (see page 7665) property.

Pascal

```
function GetDocumentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetDocumentNode();
```

Description

GetDocumentNode is the read implementation of the Node (see page 7665) property.

See Also

Node (see page 7665)

81.2.1.2.11 IXMLDocument.GetDOMDocument Method

Returns the value of the DOMDocument (see page 7663) property.

Pascal

```
function GetDOMDocument: IDOMDocument;
```

C++

```
__fastcall IDOMDocument GetDOMDocument();
```

Description

GetDOMDocument is the read implementation of the DOMDocument (see page 7663) property.

See Also

DOMDocument (see page 7663)

81.2.1.2.12 IXMLDocument.GetEncoding Method

Returns the value of the Encoding (see page 7664) property.

Pascal

```
function GetEncoding: DOMString;
```

C++

```
__fastcall DOMString GetEncoding();
```

Description

GetEncoding is the read implementation of the Encoding (see page 7664) property.

See Also

Encoding (see page 7664)

81.2.1.2.13 IXMLDocument.GetFileName Method

Returns the value of the FileName (see page 7664) property.

Pascal

```
function GetFileName: DOMString;
```

C++

```
__fastcall DOMString GetFileName();
```

Description

GetFileName is the read implementation of the FileName (see page 7664) property.

See Also

FileName (see page 7664)

81.2.1.2.14 IXMLDocument.GetModified Method

Returns the value of the Modified (see page 7665) property.

Pascal

```
function GetModified: Boolean;
```

C++

```
__fastcall Boolean GetModified();
```

Description

GetModified is the read implementation of the Modified (see page 7665) property.

See Also

Modified (see page 7665)

81.2.1.2.15 IXMLDocument.GetNodeIndentStr Method

Returns the value of the NodeIndentStr (see page 7666) property.

Pascal

```
function GetNodeIndentStr: DOMString;
```

C++

```
__fastcall DOMString GetNodeIndentStr();
```

Description

GetNodeIndentStr is the read implementation of the NodeIndentStr (see page 7666) property.

See Also

NodeIndentStr (see page 7666)

81.2.1.2.16 IXMLDocument.GetOptions Method

Returns the value of the Options (see page 7666) property.

Pascal

```
function GetOptions: TXMLDocOptions;
```

C++

```
__fastcall TXMLDocOptions GetOptions();
```

Description

GetOptions is the read implementation of the Options (see page 7666) property.

See Also

Options (see page 7666)

81.2.1.2.17 IXMLDocument.GetParseOptions Method

Returns the value of the ParseOptions (see page 7667) property.

Pascal

```
function GetParseOptions: TParseOptions;
```

C++

```
__fastcall TParseOptions GetParseOptions();
```

Description

GetParseOptions is the read implementation of the ParseOptions (see page 7667) property.

See Also

ParseOptions (see page 7667)

81.2.1.2.18 IXMLDocument.GetSchemaRef Method

Returns the value of the SchemaRef (see page 7667) property.

Pascal

```
function GetSchemaRef: DOMString;
```

C++

```
__fastcall DOMString GetSchemaRef();
```

Description

GetSchemaRef is the read implementation of the SchemaRef (see page 7667) property.

See Also

SchemaRef (see page 7667)

81.2.1.2.19 IXMLDocument.GetStandAlone Method

Returns the value of the StandAlone (see page 7668) property.

Pascal

```
function GetStandAlone: DOMString;
```

C++

```
__fastcall DOMString GetStandAlone();
```

Description

GetStandAlone is the read implementation of the StandAlone (see page 7668) property.

See Also

StandAlone (see page 7668)

81.2.1.2.20 IXMLDocument.GetVersion Method

Returns the value of the Version (see page 7668) property.

Pascal

```
function GetVersion: DOMString;
```

C++

```
__fastcall DOMString GetVersion();
```

Description

GetVersion is the read implementation of the Version (see page 7668) property.

See Also

Version (see page 7668)

81.2.1.2.21 IXMLDocument.GetXML Method

Returns the value of the XML (see page 7668) property.

Pascal

```
function GetXML: TStrings;
```

C++

```
__fastcall TStrings GetXML();
```

Description

GetXML is the read implementation of the XML (see page 7668) property.

See Also

XML (see page 7668)

81.2.1.2.22 IXMLDocument.IsEmptyDoc Method

Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.

Pascal

```
function IsEmptyDoc: Boolean;
```

C++

```
__fastcall Boolean IsEmptyDoc();
```

Description

Call IsEmptyDoc to determine whether the IXMLDocument (see page 7640) instance represents an empty document. IsEmptyDoc returns true if the DOMDocument (see page 7663) property is not set (for example, if there is no DOM implementation) or if this interface represents a document with no child nodes.

See Also

XML (see page 7668)

ChildNodes (see page 7663)

DOMDocument (see page 7663)

81.2.1.2.23 IXMLDocument.LoadFromFile Method

Loads an XML (see page 7668) document from disk.

Pascal

```
procedure LoadFromFile(const AFileName: DOMString);
```

C++

```
__fastcall LoadFromFile(const DOMString AFileName);
```

Description

Call LoadFromFile to load the XML (see page 7668) document specified by AFileName so that you can examine or modify it.

AFileName is the name of the file to load.

Once you have loaded an XML (see page 7668) document, any changes you make to the document are not saved back to disk until you call the SaveToFile (see page 7655) method.

See Also

SaveToFile (see page 7655)

XML (see page 7668)

FileName (see page 7664)

81.2.1.2.24 IXMLDocument.LoadFromStream Method

Loads an XML (see page 7668) document from a stream and activates it.

Pascal

```
procedure LoadFromStream(const Stream: TStream; EncodingType: TXMLEncodingType = xetUnknown);
```

C++

```
__fastcall LoadFromStream(const TStream Stream, TXMLEncodingType EncodingType = xetUnknown);
```

Description

Call LoadFromStream to load the XML (see page 7668) document from a stream.

Stream is a stream object that can be used to read the string of XML (see page 7668) that makes up the document.

EncodingType indicates the character set that is used for Stream. If EncodingType is xetUnknown (the default), LoadFromStream attempts to infer the encoding type from the XML (see page 7668) encoding declaration at the beginning of the document.

After loading the document from Stream, LoadFromStream sets the Active (see page 7661) property to true.

See Also

SaveToStream (see page 7656)

LoadFromFile (see page 7652)

LoadFromXML (see page 7653)

Active (see page 7661)

81.2.1.2.25 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	IXMLDocument.LoadFromXML (AnsiString) (see page 7653)	Loads a string representation of an XML (see page 7668) document and activates it.
	IXMLDocument.LoadFromXML (DOMString) (see page 7653)	Loads a string representation of an XML (see page 7668) document and activates it.

81.2.1.2.25.1 IXMLDocument.LoadFromXML Method (AnsiString)

Loads a string representation of an XML (see page 7668) document and activates it.

Pascal

```
procedure LoadFromXML(const XML: AnsiString); overload;
procedure LoadFromXML(const XML: DOMString); overload;
```

C++

```
__fastcall LoadFromXML(const AnsiString XML);
__fastcall LoadFromXML(const DOMString XML);
```

Description

Call LoadFromXML to assign a string as the value of the XML (see page 7668) document. Unlike the XML (see page 7668)

property, which lets you assign XML (see page 7668) on a line-by-line basis, LoadFromXML treats the text of the XML (see page 7668) document as a whole.

After assigning the XML (see page 7668) property as the contents of the document, LoadFromXML sets the Active (see page 7661) property to true.

See Also

SaveToXML (see page 7656)

LoadFromFile (see page 7652)

Active (see page 7661)

LoadFromStream (see page 7653)

XML (see page 7668)

81.2.1.2.26 IXMLDocument.Refresh Method

Updates the parsed XML (see page 7668) document to reflect external changes.

Pascal

```
procedure Refresh;
```

C++

```
__fastcall Refresh();
```

Description

Call Refresh when the XML (see page 7668) document specified by FileName (see page 7664) has changed due to an external cause. Refresh updates the parsed version of the XML (see page 7668) document (which can be traversed starting with DocumentElement (see page 7663)) to reflect the current contents of the file specified by FileName (see page 7664).

The Active (see page 7661) property must be true when you call Refresh.

Note: Do not call Refresh after making changes directly to the XML (see page 7668) property, because assigning a value to XML (see page 7668) automatically closes the XML (see page 7668) document. To refresh the XML (see page 7668) document after assigning a new value to XML (see page 7668), set the Active (see page 7661) property to true.

See Also

FileName (see page 7664)

Active (see page 7661)

DOMDocument (see page 7663)

DocumentElement (see page 7663)

Resync (see page 7655)

81.2.1.2.27 IXMLDocument.RegisterDocBinding Method

Registers an implementation class for child nodes with a specified tag name.

Pascal

```
procedure RegisterDocBinding(const TagName: DOMString; DocNodeClass: TClass; NamespaceURI:
DOMString = '');
```

C++

```
__fastcall RegisterDocBinding(const DOMString TagName, TClass DocNodeClass, DOMString
NamespaceURI = '');
```

Description

Call RegisterDocBinding to specify the implementation class for one of the child nodes of the document node. RegisterDocBinding replaces the specified node with an instance of the specified class.

TagName is the tag name of the child node to be replaced. Typically, this identifies the document element.

DocNodeClass is the implementation class to use for all child nodes with the specified tag name.

NamespaceURI is the URI of the namespace in which TagName is defined.

See Also

DocumentElement (see page 7663)

GetDocBinding (see page 7647)

ChildNodes (see page 7663)

81.2.1.2.28 IXMLDocument.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the document to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7663)

81.2.1.2.29 IXMLDocument.SaveToFile Method

Saves the XML (see page 7668) document to disk.

Pascal

```
procedure SaveToFile(const AFileName: DOMString);
```

C++

```
__fastcall SaveToFile(const DOMString AFileName);
```

Description

Call SaveToFile to save any modifications you have made to the parsed XML (see page 7668) document.

AFileName is the name of the file to save.

Tip: To save the modifications back to the file from which the document was originally read, use the value of the FileName (see page 7664) property for the AFileName parameter.

See Also

LoadFromFile (see page 7652)

XML (see page 7668)

FileName (see page 7664)

81.2.1.2.30 IXMLDocument.SaveToStream Method

Saves the XML (see page 7668) document to a stream.

Pascal

```
procedure SaveToStream(const Stream: TStream);
```

C++

```
__fastcall SaveToStream(const TStream Stream);
```

Description

Call SaveToStream to save the contents of the XML (see page 7668) document to the stream specified by Stream. Regardless of the encoding system of the original XML (see page 7668) document, SaveToStream always saves the stream in UTF-16.

See Also

LoadFromStream (see page 7653)

SaveToXML (see page 7656)

SaveToFile (see page 7655)

81.2.1.2.31 SaveToXML Method

This is the overview for the SaveToXML method overload.

Overload List

	Name	Description
	IXMLDocument.SaveToXML (DOMString) (see page 7656)	Saves the XML (see page 7668) document to a string-type variable.
	IXMLDocument.SaveToXML (UTF8String) (see page 7656)	Saves the XML (see page 7668) document to a string-type variable.
	IXMLDocument.SaveToXML (WideString) (see page 7656)	Saves the XML (see page 7668) document to a string-type variable.

81.2.1.2.31.1 IXMLDocument.SaveToXML Method (DOMString)

Saves the XML (see page 7668) document to a string-type variable.

Pascal

```
procedure SaveToXML(var XML: DOMString); overload;
procedure SaveToXML(var XML: WideString); overload;
procedure SaveToXML(var XML: UTF8String); overload;
```

C++

```
__fastcall SaveToXML(DOMString XML);
```

```
__fastcall SaveToXML(BSTR XML);
__fastcall SaveToXML(UTF8String XML);
```

Description

Call SaveToXML to save the contents of the XML (see page 7668) document to the string-type variable specified by XML (see page 7668). SaveToXML writes the contents of XML (see page 7668) document using UTF-8 or UTF-16 as an encoding system, depending on the type of the XML (see page 7668) parameter.

Unlike the XML (see page 7668) property, which lets you write individual lines from the XML (see page 7668) document, SaveToXML writes the entire text of the XML (see page 7668) document.

See Also

LoadFromXML (see page 7653)

SaveToStream (see page 7656)

SaveToFile (see page 7655)

XML (see page 7668)

81.2.1.2.32 IXMLDocument.SetActive Method

Sets the value of the Active (see page 7661) property.

Pascal

```
procedure SetActive(const Value: Boolean);
```

C++

```
__fastcall SetActive(const Boolean Value);
```

Description

SetActive is the write implementation of the Active (see page 7661) property.

Value is the new value to set.

See Also

Active (see page 7661)

81.2.1.2.33 IXMLDocument.SetDocumentElement Method

Sets the value of the DocumentElement (see page 7663) property.

Pascal

```
procedure SetDocumentElement(const Value: IXMLNode);
```

C++

```
__fastcall SetDocumentElement(const IXMLNode Value);
```

Description

SetDocumentElement is the write implementation of the DocumentElement (see page 7663) property.

Value is the interface for the node tree that replaces the current document element.

See Also

DocumentElement (see page 7663)

81.2.1.2.34 IXMLDocument.SetDOMDocument Method

Sets the value of the DOMDocument (see page 7663) property.

Pascal

```
procedure SetDOMDocument(const Value: IDOMDocument);
```

C++

```
__fastcall SetDOMDocument(const IDOMDocument Value);
```

Description

SetDOMDocument is the write implementation of the DOMDocument (see page 7663) property.

Value is the DOM interface for the document to assign.

See Also

DOMDocument (see page 7663)

81.2.1.2.35 IXMLDocument.SetEncoding Method

Sets the value of the Encoding (see page 7664) property.

Pascal

```
procedure SetEncoding(const Value: DOMString);
```

C++

```
__fastcall SetEncoding(const DOMString Value);
```

Description

SetEncoding is the write implementation of the Encoding (see page 7664) property.

Value is the value of the encoding attribute of the document type node.

See Also

Encoding (see page 7664)

81.2.1.2.36 IXMLDocument.SetFileName Method

Sets the value of the FileName (see page 7664) property.

Pascal

```
procedure SetFileName(const Value: DOMString);
```

C++

```
__fastcall SetFileName(const DOMString Value);
```

Description

SetFileName is the write implementation of the FileName (see page 7664) property.

Value is the name of an XML (see page 7668) document or, if the vendor supports it, an URL for the XML (see page 7668) document.

See Also

FileName (see page 7664)

81.2.1.2.37 IXMLDocument.SetNodeIndentStr Method

Sets the value of the NodeIndentStr (see page 7666) property.

Pascal

```
procedure SetNodeIndentStr(const Value: DOMString);
```

C++

```
__fastcall SetNodeIndentStr(const DOMString Value);
```

Description

SetNodeIndentStr is the write implementation of the NodeIndentStr (see page 7666) property.

Value is the string that is inserted before nested nodes to indicate a level of nesting.

See Also

NodeIndentStr (see page 7666)

81.2.1.2.38 IXMLDocument.SetOnAsyncLoad Method

Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.

Pascal

```
procedure SetOnAsyncLoad(const Value: TAsyncEventHandler);
```

C++

```
__fastcall SetOnAsyncLoad(const TAsyncEventHandler Value);
```

Description

Use SetOnAsyncLoad to supply a callback that responds when the asynchronous parsing of an XML (see page 7668) document progresses to a certain point.

Value is a method that is called when the AsyncLoadState (see page 7662) property changes to reflect a new stage in the parsing process.

See Also

AsyncLoadState (see page 7662)

81.2.1.2.39 IXMLDocument.SetOptions Method

Sets the value of the Options (see page 7666) property.

Pascal

```
procedure SetOptions(const Value: TXMLDocOptions);
```

C++

```
__fastcall SetOptions(const TXMLDocOptions Value);
```

Description

GetOptions (see page 7650) is the write implementation of the Options (see page 7666) property.

Value is the set of options to assign.

See Also

Options (see page 7666)

81.2.1.2.40 IXMLDocument.SetParseOptions Method

Sets the value of the ParseOptions (see page 7667) property.

Pascal

```
procedure SetParseOptions(const Value: TParseOptions);
```

C++

```
__fastcall SetParseOptions(const TParseOptions Value);
```

Description

GetParseOptions (see page 7650) is the write implementation of the ParseOptions (see page 7667) property.

Value is the set of parser options to assign.

See Also

ParseOptions (see page 7667)

81.2.1.2.41 IXMLDocument.SetStandAlone Method

Sets the value of the StandAlone (see page 7668) property.

Pascal

```
procedure SetStandAlone(const Value: DOMString);
```

C++

```
__fastcall SetStandAlone(const DOMString Value);
```

Description

SetStandAlone is the write implementation of the StandAlone (see page 7668) property.

Value is the value of the stand-alone attribute of the document type node.

See Also

StandAlone (see page 7668)

81.2.1.2.42 IXMLDocument.SetVersion Method

Sets the value of the Version (see page 7668) property.

Pascal

```
procedure SetVersion(const Value: DOMString);
```

C++

```
__fastcall SetVersion(const DOMString Value);
```

Description

SetVersion is the write implementation of the Version (see page 7668) property.

Value is the value of the version attribute of the document type node.

See Also

Version (see page 7668)

81.2.1.2.43 IXMLDocument.SetXML Method

Sets the value of the XML (see page 7668) property.

Pascal

```
procedure SetXML(const Value: TStrings);
```

C++

```
__fastcall SetXML(const TStrings Value);
```

Description

SetXML is the write implementation of the XML (see page 7668) property.

Value contains the raw (unparsed) XML (see page 7668) to assign.

See Also

XML (see page 7668)

81.2.1.3 IXMLDocument Properties

The properties of the IXMLDocument class are listed here.

Public Properties

	Name	Description
	Active (see page 7661)	Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7662)	Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.
R	ChildNodes (see page 7663)	Lists all of the child nodes of the document.
	DocumentElement (see page 7663)	Provides access to the root node of the XML (see page 7668) document.
	DOMDocument (see page 7663)	Provides access to the DOM interface for the parsed XML (see page 7668) document.
	Encoding (see page 7664)	Indicates the character set that the XML (see page 7668) document uses to encode characters.
	FileName (see page 7664)	Indicates the XML (see page 7668) document file, if any, that this interface represents.
R	Modified (see page 7665)	Indicates whether the XML (see page 7668) document has been modified.
R	Node (see page 7665)	Provides access to the document node for the XML (see page 7668) document.
	NodeIndentStr (see page 7666)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.
	Options (see page 7666)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7667)	Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.
R	SchemaRef (see page 7667)	Returns the name of the schema associated with this XML (see page 7668) document.
	StandAlone (see page 7668)	Indicates whether the XML (see page 7668) document includes any external markup declarations.
	Version (see page 7668)	Indicates the XML (see page 7668) version that the document uses.
	XML (see page 7668)	Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.

81.2.1.3.1 IXMLDocument.Active Property

Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.

Pascal

```
property Active: Boolean;
```

C++

```
__property Boolean Active;
```

Description

Before you can use IXMLDocument (see page 7640) to examine or change the contents of an XML (see page 7668) document, it must be active. Setting Active to true results in the parsing of the XML (see page 7668) document specified by the FileName (see page 7664) property or by the XML (see page 7668) property. If both FileName (see page 7664) and XML (see page 7668) are empty strings, Setting Active to true causes IXMLDocument (see page 7640) to represent an empty document.

Active must be true before you can access the DOMDocument (see page 7663) or DocumentElement (see page 7663) properties.

See Also

DOMDocument (see page 7663)

DocumentElement (see page 7663)

FileName (see page 7664)

XML (see page 7668)

81.2.1.3.2 IXMLDocument.AsyncLoadState Property

Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.

Pascal

```
property AsyncLoadState: Integer;
```

C++

```
__property int AsyncLoadState;
```

Description

Use AsyncLoadState to determine that status of the DOM parser when it is parsing an XML (see page 7668) document asynchronously. The value of AsyncLoadState is vendor-specific. That is, each DOM parser that loads and parses an XML (see page 7668) document asynchronously supplies its own flags to indicate state information.

When the DOM vendor is MSXML, AsyncLoadState can have any of the following values:

Value	Meaning
0	Uninitialized: The AsyncLoadState is not assigned. (Usually indicates that the parser has not tried to parse a document or it is not parsing the document asynchronously.)
1	Loading: The parser is currently parsing the structure of the XML (see page 7668) document.
2	Loaded: The parser has finished reading and parsing the structure of the XML (see page 7668) document, but the corresponding DOM interfaces are not yet available.
3	Interactive: The parser has parsed the structure of the XML (see page 7668) document, the corresponding DOM interfaces are available, but the parser has not finished reading all the node values.
4	Complete: The parser has fully read and parsed the XML (see page 7668) document.

See Also

SetOnAsyncLoad (see page 7659)

81.2.1.3.3 IXMLDocument.ChildNodes Property

Lists all of the child nodes of the document.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access all of the nodes that are children of the document object. In addition to DocumentElement (see page 7663), which is the root of the data hierarchy, ChildNodes can include nodes for comments, processing instructions, and so on.

See Also

DocumentElement (see page 7663)

Node (see page 7665)

81.2.1.3.4 IXMLDocument.DocumentElement Property

Provides access to the root node of the XML (see page 7668) document.

Pascal

```
property DocumentElement: IXMLNode;
```

C++

```
__property IXMLNode DocumentElement;
```

Description

Use DocumentElement to access the root node of the XML (see page 7668) document. Because XML (see page 7668) documents are hierarchical in structure, DocumentElement is the entry point to all data in the XML (see page 7668) document. Setting DocumentElement replaces the entire data hierarchy with a new hierarchy of nodes.

The IXMLNode (see page 7669) interface includes properties and methods for examining and changing the text and attributes of a node, for adding and deleting child nodes, and for traversing the hierarchy of nodes rooted at that node.

See Also

DOMDocument (see page 7663)

ChildNodes (see page 7663)

81.2.1.3.5 IXMLDocument.DOMDocument Property

Provides access to the DOM interface for the parsed XML (see page 7668) document.

Pascal

```
property DOMDocument: IDOMDocument;
```

C++

```
__property IDOMDocument DOMDocument;
```

Description

Use DOMDocument to directly access the IDOMDocument (see page 7530) interface for the parsed XML (see page 7668) document. This is the interface supported by the DOM parser for the document.

Most applications do not need to use the DOMDocument property directly. Instead, it supplies the underlying support for parsing and manipulating the nodes accessed through DocumentElement (see page 7663). If you do want to make use of DOMDocument's low-level support, you must be familiar with the W3C Document Object Model (DOM) Specification.

See Also

DocumentElement (see page 7663)

81.2.1.3.6 IXMLDocument.Encoding Property

Indicates the character set that the XML (see page 7668) document uses to encode characters.

Pascal

```
property Encoding: DOMString;
```

C++

```
__property DOMString Encoding;
```

Description

Encoding is the encoding name that appears in the document type declaration of the XML (see page 7668) document. It is a string such as 'UTF-8', 'UTF-16', or 'ISO-10646-UCS-2' that indicates the character set used in the document.

See Also

FileName (see page 7664)

DOMDocument (see page 7663)

XML (see page 7668)

81.2.1.3.7 IXMLDocument.FileName Property

Indicates the XML (see page 7668) document file, if any, that this interface represents.

Pascal

```
property FileName: DOMString;
```

C++

```
__property DOMString FileName;
```

Description

Use FileName to get or set the name of the XML (see page 7668) file that this document represents. FileName is set automatically when an application calls the LoadFromFile (see page 7652) method or when this interface is obtained from the global LoadXMLDocument function.

When you set the value of FileName, the XML (see page 7668) property automatically changes (at runtime) to represent the contents of that file.

Note: The content of this XML (see page 7668) document, which is available as the value of the XML (see page 7668) property, may not be the same as the content of the specified file. If an application uses IXMLDocument (see page 7640) to

edit the document, those changes are not saved back to the XML (see page 7668) document file until you call the SaveToFile (see page 7655) method.

Tip: If the underlying DOM implementation supports it, you can use FileName to specify an URL rather than a file on disk.

See Also

SaveToFile (see page 7655)

LoadFromFile (see page 7652)

LoadXMLDocument (see page 7491)

XML (see page 7668)

81.2.1.3.8 IXMLDocument.Modified Property

Indicates whether the XML (see page 7668) document has been modified.

Pascal

```
property Modified: Boolean;
```

C++

```
__property Boolean Modified;
```

Description

Read (see page 3634) Modified to determine whether the application has made any changes to the nodes in the XML (see page 7668) document. When Modified is true, the value of the XML (see page 7668) property may differ from the contents of the file specified by FileName (see page 7664). You can call the SaveToFile (see page 7655) method to update the XML (see page 7668) document file to reflect the changes.

See Also

Resync (see page 7655)

SaveToFile (see page 7655)

XML (see page 7668)

FileName (see page 7664)

81.2.1.3.9 IXMLDocument.Node Property

Provides access to the document node for the XML (see page 7668) document.

Pascal

```
property Node: IXMLNode;
```

C++

```
__property IXMLNode Node;
```

Description

Read (see page 3634) Node to access the document node for the XML (see page 7668) document. The document node represents information about the entire document. It may have several child nodes, which represent comments, processing instructions, and which include the DocumentElement (see page 7663) node that acts as the root of the data hierarchy.

See Also

DocumentElement (see page 7663)

ChildNodes (see page 7663)

81.2.1.3.10 IXMLDocument.NodeIndentStr Property

Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.

Pascal

```
property NodeIndentStr: DOMString;
```

C++

```
__property DOMString NodeIndentStr;
```

Description

Use NodeIndentStr to customize how the document formats nodes that are added. When the Options (see page 7666) property includes doNodeAutoIndent, this string is inserted in the generated XML (see page 7668) before any newly added node, once for each nesting level.

If you do not assign a value to NodeIndentStr, IXMLDocument (see page 7640) inserts two spaces to represent each nesting level.

See Also

Options (see page 7666)

NestingLevel (see page 7428)

FormatXMLData (see page 7490)

XML (see page 7668)

81.2.1.3.11 IXMLDocument.Options Property

Influences various aspects of the document's behavior.

Pascal

```
property Options: TXMLDocOptions;
```

C++

```
__property TXMLDocOptions Options;
```

Description

Set Options to configure how you want the document to respond when the application tries to read nodes or attributes that do not exist, when it creates a new namespace when adding a node, when it generates XML (see page 7668) text from the nodes in the document, or whether it saves its value automatically when you close the document. Options can include any of the following values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 7703) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML (see page 7668) text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.

doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null Variant (see page 3421) (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild (see page 7643) method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild (see page 7643) creates the new namespace but does not add an attribute to declare it.
doAutoSave	When you close the document (set Active (see page 7661) to false), any changes are automatically saved back to the file specified by FileName (see page 7664) or to the XML (see page 7668) property.

See Also

Nodes (see page 7714)

Attributes (see page 7688)

AddChild (see page 7672)

81.2.1.3.12 IXMLDocument.ParseOptions Property

Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.

Pascal

```
property ParseOptions: TParseOptions;
```

C++

```
__property TParseOptions ParseOptions;
```

Description

Set ParseOptions to influence the way the DOM implementation parses the XML (see page 7668) document. Different DOM implementations support different options. If you attempt to use an option that is not supported by your DOM implementation, IXMLDocument (see page 7640) raises an EXMLDocError (see page 7639) exception.

See Also

DOMImplementation (see page 7385)

EXMLDocError (see page 7639)

81.2.1.3.13 IXMLDocument.SchemaRef Property

Returns the name of the schema associated with this XML (see page 7668) document.

Pascal

```
property SchemaRef: DOMString;
```

C++

```
__property DOMString SchemaRef;
```

Description

Read (see page 3634) SchemaRef to determine the schema associated with this XML (see page 7668) document. If the document does not have an attribute that identifies an associated schema, the value of SchemaRef is an empty string.

See Also

DOMDocument (see page 7663)

81.2.1.3.14 IXMLDocument.StandAlone Property

Indicates whether the XML (see page 7668) document includes any external markup declarations.

Pascal

```
property StandAlone: DOMString;
```

C++

```
__property DOMString StandAlone;
```

Description

StandAlone is the value of the stand-alone attribute on the document type declaration of the XML (see page 7668) document. StandAlone has a value of 'yes' or 'no'.

When StandAlone is 'yes', the document contains no external markup declarations. It may, however, still contain references to external entities, as long as they are not declarations.

When StandAlone is 'no', the document may or may not contain external markup declarations.

See Also

DOMDocument (see page 7663)

81.2.1.3.15 IXMLDocument.Version Property

Indicates the XML (see page 7668) version that the document uses.

Pascal

```
property Version: DOMString;
```

C++

```
__property DOMString Version;
```

Description

Version indicates the XML (see page 7668) version number that the document uses. An XML (see page 7668) version number is a string of the form 'n.m', where n is the major version number and m is the minor version number.

See Also

XML (see page 7668)

81.2.1.3.16 IXMLDocument.XML Property

Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.

Pascal

```
property XML: TStrings;
```

C++

```
__property TStrings XML;
```

Description

Read (see page 3634) XML to obtain the contents of the associated XML document. Set XML to specify the raw (unparsed)

XML you want to examine or manipulate.

XML is a TStrings object. This means you can manipulate the XML line by line. To read or write the XML in the document as a single string, use the LoadFromXML (see page 7653) or SaveToXML (see page 7656) method.

Note: If you call the LoadFromFile (see page 7652), LoadFromStream (see page 7653), or LoadFromXML (see page 7653) method, XML is automatically updated to reflect the contents of that file.

See Also

LoadFromFile (see page 7652)

DocumentElement (see page 7663)

TXMLDocument (see page 7347)

LoadFromXML (see page 7653)

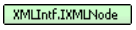
SaveToXML (see page 7656)

LoadFromStream (see page 7653)

81.2.2 IXMLNode Interface

IXMLNode is the interface for a node in an XML (see page 7697) document.

Class Hierarchy



Pascal

```
IXMLNode = interface;
```

C++

```
__interface IXMLNode;
```

File

XMLIntf

Description

Use IXMLNode to work with a node in an XML (see page 7697) document. The IXMLNode interface is designed to work with TXMLDocument and the code generated by the Data Binding wizard.

IXMLNode is similar to the IDOMNode (see page 7557) interface that is defined as part of the Document Object Model (DOM) specification. However, IXMLNode is designed to simplify the task of navigating and editing XML (see page 7697) documents by

including some of the properties and methods from the IDOMElement (see page 7542) interface, resulting in a simpler model for examining an XML (see page 7697) node.

including additional properties and methods to simplify access to data in common XML (see page 7697) structures.

The Data Binding wizard generates descendants of IXMLNode that further simplify the task of working with XML (see page 7697) documents by converting the attributes and child nodes of a node into properties of the node interface.

Note: C++ method declarations that use IXMLNode use the _di_IXMLNode type instead. This type is a DelphiInterface (see page 3256) wrapper around the IXMLNode interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNode >  _di_IXMLNode;
```

See Also

TXMLNode (see page 7396)

TXMLDocument (see page 7347)

IXMLNodeCollection (see page 7698)

IDOMNode (see page 7557)

IXMLNodeList (see page 7703)

Working with XML Nodes

81.2.2.1 IXMLNode Members

The following tables list the members exposed by IXMLNode.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

81.2.2.2 IXMLNode Methods

The methods of the IXMLNode class are listed here.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.

	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

81.2.2.2.1 AddChild Method

This is the overview for the AddChild method overload.

Overload List

	Name	Description
	IXMLNode.AddChild (DOMString, DOMString, Boolean, Integer) (see page 7672)	Adds a new child node to this node.
	IXMLNode.AddChild (DOMString, Integer) (see page 7672)	Adds a new child node to this node.

81.2.2.2.1.1 IXMLNode.AddChild Method (DOMString, DOMString, Boolean, Integer)

Adds a new child node to this node.

Pascal

```
function AddChild(const TagName: DOMString; Index: Integer = -1): IXMLNode; overload;
function AddChild(const TagName: DOMString; const NamespaceURI: DOMString; GenPrefix: Boolean
= False; Index: Integer = -1): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode AddChild(const DOMString TagName, int Index = -1);
__fastcall IXMLNode AddChild(const DOMString TagName, const DOMString NamespaceURI, Boolean
GenPrefix = False, int Index = -1);
```

Description

AddChild creates a new element node as the child of this node.

TagName provides the tag name of the newly created node.

NamespaceURI (see page 7693) identifies the namespace that includes the new node's definition. If NamespaceURI (see page 7693) is omitted (the second syntax), the namespace of the new node is deduced from the namespace prefix of TagName. If TagName has no namespace prefix and there is no NamespaceURI (see page 7693) parameter, the namespace is the

same as this node's NamespaceURI (see page 7693) property.

GenPrefix controls whether AddChild generates a namespace prefix for the namespace URI when the NamespaceURI (see page 7693) parameter identifies a URI that is not already declared in the document. Note that setting GenPrefix to true does not cause AddChild to add a declaration of the new namespace prefix unless the document's Options property includes doNamespaceDecl.

Index indicates the position of the child node in this node's list of children, where 0 is the first position, 1 is the second position, and so on. If Index is –1, the new node is added to the end.

AddChild returns the interface for the newly created child node.

Tip: AddChild only adds element nodes. To add other types of child nodes, use the XML (see page 7697) document's CreateNode method instead, and then add the resulting node to the ChildNodes (see page 7689) property array.

See Also

Delete (see page 7707)

ChildNodes (see page 7689)

CreateNode (see page 7645)

NamespaceURI (see page 7693)

Options (see page 7389)

Working with XML Nodes

81.2.2.2.2 IXMLNode.CloneNode Method

Creates a copy of this node and returns its interface.

Pascal

```
function CloneNode(Deep: Boolean): IXMLNode;
```

C++

```
__fastcall IXMLNode CloneNode(Boolean Deep);
```

Description

CloneNode returns a copy of this node that has no parent. Any attributes of this node are copied as well, including attribute nodes that represent default values.

Deep indicates whether the child nodes of this node should be cloned as well. For example, if an element node contains any text, CloneNode only copies then text when deep is true, because in the underlying DOM implementation, the text is contained in a child node.

After calling CloneNode to create a new node, you can add that node into the ChildNodes (see page 7689) property of another node (or, if this node represents an attribute, into the AttributeNodes (see page 7688) property) by calling ChildNodes (see page 7689). ->Add or ChildNodes (see page 7689). ->Insert (see page 3601).

See Also

ChildNodes (see page 7689)

AddChild (see page 7672)

ParentNode (see page 7695)

cloneNode (see page 7560)

Add (see page 7705)

Insert (see page 7712)

81.2.2.2.3 IXMLNode.DeclareNamespace Method

Adds an attribute to the node that declares a specified namespace URI.

Pascal

```
procedure DeclareNamespace(const Prefix: DOMString; const URI: DOMString);
```

C++

```
__fastcall DeclareNamespace(const DOMString Prefix, const DOMString URI);
```

Description

DeclareNamespace adds an attribute to this node that declares the specified namespace URI as the value of the specified prefix.

Prefix (see page 7696) is the namespace prefix that appears in tag names that reference the namespace.

URI is the corresponding namespace URI.

Note: This method is called automatically for newly created nodes when the document's Options property includes doNamespaceDecl.

See Also

Options (see page 7666)

Attributes (see page 7688)

FindNamespaceDecl (see page 7674)

81.2.2.2.4 IXMLNode.FindNamespaceDecl Method

Returns the attribute node that declares a specified Namespace URI.

Pascal

```
function FindNamespaceDecl(const NamespaceURI: DOMString): IXMLNode;
```

C++

```
__fastcall IXMLNode FindNamespaceDecl(const DOMString NamespaceURI);
```

Description

FindNamespaceDecl searches the node hierarchy that contains this node for an attribute that specifies the mapping between a namespace prefix and the namespace URI identified by NamespaceURI (see page 7693). If it finds such a mapping, FindNamespaceDecl returns the interface for the Attribute node. If it does not find a mapping, FindNamespaceDecl returns nil (Delphi) or NULL (C++).

NamespaceURI (see page 7693) is the namespace URI to locate.

See Also

FindNamespaceURI (see page 7674)

DeclareNamespace (see page 7674)

81.2.2.2.5 IXMLNode.FindNamespaceURI Method

Returns the namespace URI for a namespace prefix or qualified tag name.

Pascal

```
function FindNamespaceURI(const TagOrPrefix: DOMString): DOMString;
```

C++

```
__fastcall DOMString FindNamespaceURI(const DOMString TagOrPrefix);
```

Description

FindNamespaceDecl (see page 7674) searches the node hierarchy that contains this node for an attribute that specifies the mapping between the namespace prefix identified by TagOrPrefix and a namespace URI. If it finds such a mapping, FindNamespaceURI returns the namespace URI. If it does not find a mapping, FindNamespaceURI returns an empty string.

TagOrPrefix is the namespace prefix or a tag name that includes the namespace prefix.

See Also

GetNamespaceURI (see page 7679)

GetPrefix (see page 7680)

FindNamespaceDecl (see page 7674)

81.2.2.2.6 IXMLNode.GetAttribute Method

Returns the value of one of this node's attributes.

Pascal

```
function GetAttribute(const AttrName: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttribute(const DOMString AttrName);
```

Description

GetAttribute is the read implementation of the Attributes (see page 7688) property.

AttrName is the name of the attribute.

GetAttribute returns the value of the specified attribute. Typically, this value is a string. If the node does not have an attribute with the name specified by AttrName, GetAttribute returns a Null Variant (see page 3421).

Note: Do not call GetAttribute unless this node is an element node. Only element nodes have attributes.

See Also

Attributes (see page 7688)

GetAttributeNS (see page 7676)

81.2.2.2.7 IXMLNode.GetAttributeNodes Method

Returns the value of the AttributeNodes (see page 7688) property.

Pascal

```
function GetAttributeNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetAttributeNodes();
```

Description

GetAttributeNodes is the read implementation of the AttributeNodes (see page 7688) property.

See Also

AttributeNodes (see page 7688)

81.2.2.2.8 IXMLNode.GetAttributeNS Method

Returns the value of a specified attribute.

Pascal

```
function GetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString): OleVariant;
```

C++

```
__fastcall OleVariant GetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI);
```

Description

Call GetAttributeNS to determine the value of an attribute when you must explicitly specify the namespace for the attribute.

AttrName is the name of the attribute whose value you want to fetch.

NamespaceURI (see page 7693) identifies the namespace in which the attribute resides.

GetAttributeNS returns the value of the attribute. If AttrName and NamespaceURI (see page 7693) do not identify an attribute of this node, GetAttributeNS returns a Null Variant (see page 3421).

See Also

AttributeNodes (see page 7688)

Attributes (see page 7688)

NamespaceURI (see page 7693)

SetAttributeNS (see page 7684)

81.2.2.2.9 IXMLNode.GetChildNodes Method

Returns the value of the ChildNodes (see page 7689) property.

Pascal

```
function GetChildNodes: IXMLNodeList;
```

C++

```
__fastcall IXMLNodeList GetChildNodes();
```

Description

GetChildNodes is the read implementation of the ChildNodes (see page 7689) property.

See Also

ChildNodes (see page 7689)

81.2.2.2.10 IXMLNode.GetChildValue Method

Returns the value of a specified child node.

Pascal

```
function GetChildValue(const IndexOrName: OleVariant): OleVariant;
```

C++

```
__fastcall OleVariant GetChildValue(const OleVariant IndexOrName);
```

Description

GetChildValue is the read implementation of the ChildValues (see page 7690) property.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

The return value depends on the type of the child node, as indicated in the following table:

NodeType (see page 7694)	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, reading the child's value causes an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntEntityRef	nil (Delphi) or NULL (C++)
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
ntDocFragment	nil (Delphi) or NULL (C++)

See Also

ChildValues (see page 7690)

81.2.2.2.11 IXMLNode.GetCollection Method

Returns the value of the Collection (see page 7690) property.

Pascal

```
function GetCollection: IXMLNodeCollection;
```

C++

```
__fastcall IXMLNodeCollection GetCollection();
```

Description

GetCollection is the read implementation of the Collection (see page 7690) property.

See Also

Collection (see page 7690)

81.2.2.2.12 IXMLNode.GetDOMNode Method

Returns the value of the DOMNode (see page 7691) property.

Pascal

```
function GetDOMNode: IDOMNode;
```

C++

```
__fastcall IDOMNode GetDOMNode();
```

Description

GetDOMNode is the read implementation of the DOMNode (see page 7691) property.

See Also

DOMNode (see page 7691)

81.2.2.2.13 IXMLNode.GetHasChildNodes Method

Indicates whether the node has any child nodes.

Pascal

```
function GetHasChildNodes: Boolean;
```

C++

```
__fastcall Boolean GetHasChildNodes();
```

Description

GetHasChildNodes is the read implementation of the HasChildNodes (see page 7691) property.

See Also

HasChildNodes (see page 7691)

81.2.2.2.14 IXMLNode.GetIsTextElement Method

Indicates whether the node has a single text value.

Pascal

```
function GetIsTextElement: Boolean;
```

C++

```
__fastcall Boolean GetIsTextElement();
```

Description

GetIsTextElement is the read implementation of the IsTextElement (see page 7692) property.

See Also

IsTextElement (see page 7692)

81.2.2.2.15 IXMLNode.GetLocalName Method

Returns the name of the node without any namespace prefix.

Pascal

```
function GetLocalName: DOMString;
```

C++

```
__fastcall DOMString GetLocalName();
```

Description

GetLocalName is the read implementation of the LocalName (see page 7692) property.

See Also

LocalName (see page 7692)

81.2.2.2.16 IXMLNode.GetNamespaceURI Method

Returns the namespace URI for the node's qualified name.

Pascal

```
function GetNamespaceURI: DOMString;
```

C++

```
__fastcall DOMString GetNamespaceURI();
```

Description

GetNamespaceURI is the read implementation of the NamespaceURI (see page 7693) property.

See Also

NamespaceURI (see page 7693)

81.2.2.2.17 IXMLNode.GetNodeName Method

Returns the name of the node.

Pascal

```
function GetNodeName: DOMString;
```

C++

```
__fastcall DOMString GetNodeName();
```

Description

GetNodeName is the read implementation of the NodeName (see page 7693) property.

See Also

NodeName (see page 7693)

81.2.2.2.18 IXMLNode.GetNodeType Method

Returns the type of the node.

Pascal

```
function GetNodeType: TNodeType;
```

C++

```
__fastcall TNodeType GetNodeType();
```

Description

GetNodeType is the read implementation of the NodeType (see page 7694) property.

See Also

NodeType (see page 7694)

81.2.2.2.19 IXMLNode.GetNodeValue Method

Returns the value of the node.

Pascal

```
function GetNodeValue: OleVariant;
```

C++

```
__fastcall OleVariant GetNodeValue();
```

Description

GetNodeValue is the read implementation of the NodeValue (see page 7695) property.

See Also

NodeValue (see page 7695)

81.2.2.2.20 IXMLNode.GetOwnerDocument Method

Returns the interface for the document in which this node appears.

Pascal

```
function GetOwnerDocument: IXMLDocument;
```

C++

```
__fastcall IXMLDocument GetOwnerDocument();
```

Description

GetOwnerDocument is the read implementation of the OwnerDocument (see page 7695) property.

See Also

OwnerDocument (see page 7695)

81.2.2.2.21 IXMLNode.GetParentNode Method

Returns the interface for the node's parent node.

Pascal

```
function GetParentNode: IXMLNode;
```

C++

```
__fastcall IXMLNode GetParentNode();
```

Description

GetParentNode is the read implementation of the ParentNode (see page 7695) property.

See Also

ParentNode (see page 7695)

81.2.2.2.22 IXMLNode.GetPrefix Method

Returns the namespace prefix from the node's name.

Pascal

```
function GetPrefix: DOMString;
```

C++

```
__fastcall DOMString GetPrefix();
```

Description

GetPrefix is the read implementation of the Prefix (see page 7696) property.

See Also

Prefix (see page 7696)

81.2.2.2.23 IXMLNode.GetReadOnly Method

Returns the value of the ReadOnly (see page 7696) property.

Pascal

```
function GetReadOnly: Boolean;
```

C++

```
__fastcall Boolean GetReadOnly();
```

Description

GetReadOnly is the read implementation of the ReadOnly (see page 7696) property.

See Also

ReadOnly (see page 7696)

81.2.2.2.24 IXMLNode.GetText Method

Returns the text value of the node.

Pascal

```
function GetText: DOMString;
```

C++

```
__fastcall DOMString GetText();
```

Description

GetText is the read implementation for the Text (see page 7697) property.

See Also

Text (see page 7697)

81.2.2.2.25 IXMLNode.GetXML Method

Returns the value of the XML (see page 7697) property.

Pascal

```
function GetXML: DOMString;
```

C++

```
__fastcall DOMString GetXML();
```

Description

GetXML is the read implementation of this node's XML (see page 7697) property.

See Also

XML (see page 7697)

81.2.2.2.26 HasAttribute Method

This is the overview for the HasAttribute method overload.

Overload List

	Name	Description
	IXMLNode.HasAttribute (DOMString) (see page 7682)	Indicates whether the node has an attribute with a specified name.
	IXMLNode.HasAttribute (DOMString, DOMString) (see page 7682)	Indicates whether the node has an attribute with a specified name.

81.2.2.2.26.1 IXMLNode.HasAttribute Method (DOMString)

Indicates whether the node has an attribute with a specified name.

Pascal

```
function HasAttribute(const Name: DOMString): Boolean; overload;
function HasAttribute(const Name: DOMString; const NamespaceURI: DOMString): Boolean; overload;
```

C++

```
__fastcall Boolean HasAttribute(const DOMString Name);
__fastcall Boolean HasAttribute(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call HasAttribute to determine whether this node has a specified attribute.

Name is the name of the attribute about which you want to know.

NamespaceURI (see page 7693) is the namespace of the attribute.

HasAttribute returns true if this node has the specified attribute, false otherwise.

See Also

AttributeNodes (see page 7688)

Attributes (see page 7688)

81.2.2.2.27 IXMLNode.NextSibling Method

Returns the next child of this node's parent.

Pascal

```
function NextSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode NextSibling();
```

Description

NextSibling returns the node that follows this one in the parent node's ChildNodes (see page 7689) property list.

If this node is the last node in its parent's child list, NextSibling raises an exception.

See Also

ParentNode (see page 7695)

PreviousSibling (see page 7683)

ChildNodes (see page 7689)

81.2.2.2.28 IXMLNode.Normalize Method

Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.

Pascal

```
procedure Normalize;
```

C++

```
__fastcall Normalize();
```

Description

Use the IXMLNode (see page 7669) interface to call the protected Normalize method. Normalize consolidates adjacent text nodes and remove any empty text nodes in the subtree beneath this node. This method ensures that for all child nodes of this node and of its attributes, only structure nodes (element nodes, comments, processing instructions, CDATA sections, and entity references) separate text nodes and that there are no empty text nodes.

See Also

ChildNodes (see page 7689)

AttributeNodes (see page 7688)

81.2.2.2.29 IXMLNode.PreviousSibling Method

Returns the previous child of this node's parent.

Pascal

```
function PreviousSibling: IXMLNode;
```

C++

```
__fastcall IXMLNode PreviousSibling();
```

Description

PreviousSibling returns the node that precedes this one in the parent node's ChildNodes (see page 7689) property list.

If this node is the first node in its parent's child list, PreviousSibling raises an exception.

See Also

ParentNode (see page 7695)

NextSibling (see page 7682)

ChildNodes (see page 7689)

81.2.2.2.30 IXMLNode.Resync Method

Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.

Pascal

```
procedure Resync;
```

C++

```
__fastcall Resync();
```

Description

Resync forces the node to reread the values of its child nodes and attribute nodes, discarding any changes that have not been propagated to the underlying DOM implementation.

See Also

ChildNodes (see page 7689)

AttributeNodes (see page 7688)

Refresh (see page 7371)

81.2.2.2.31 IXMLNode.SetAttribute Method

Sets the value of one of this node's attributes.

Pascal

```
procedure SetAttribute(const AttrName: DOMString; const Value: OleVariant);
```

C++

```
__fastcall SetAttribute(const DOMString AttrName, const OleVariant Value);
```

Description

SetAttribute is the write implementation for the Attributes (see page 7688) property.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

Attributes (see page 7688)

SetAttributeNS (see page 7684)

81.2.2.2.32 IXMLNode.SetAttributeNS Method

Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.

Pascal

```
procedure SetAttributeNS(const AttrName: DOMString; const NamespaceURI: DOMString; const
Value: OleVariant);
```

C++

```
__fastcall SetAttributeNS(const DOMString AttrName, const DOMString NamespaceURI, const
OleVariant Value);
```

Description

SetAttributeNS sets the value of a specified attribute of this node.

AttrName is the name of the attribute. If the attribute already exists for the node, its value is changed. If the attribute does not already exist, a new attribute is created.

NamespaceURI (see page 7693) is the namespace for the attribute.

Value is the value to assign to the attribute. This value is a string that is not parsed. Any markup (for example, for an entity reference) is treated as literal text, and must include appropriate escape sequences. If Value is NULL, the attribute is removed from this node. If Value is an empty string, the attribute is assigned an empty value.

See Also

SetAttribute (see page 7684)

GetAttributeNS (see page 7676)

AttributeNodes (see page 7688)

81.2.2.2.33 IXMLNode.SetChildValue Method

Sets the value of a specified child node.

Pascal

```
procedure SetChildValue(const IndexOrName: OleVariant; const Value: OleVariant);
```

C++

```
__fastcall SetChildValue(const OleVariant IndexOrName, const OleVariant Value);
```

Description

SetChildValue is the write implementation of the ChildValues (see page 7690) property.

IndexOrName identifies the desired child node by either its interface, or by a string that is the local name of the child node.

Value is the value to assign.

See Also

ChildValues (see page 7690)

81.2.2.2.34 IXMLNode.SetNodeValue Method

Sets the value of this node.

Pascal

```
procedure SetNodeValue(const Value: OleVariant);
```

C++

```
__fastcall SetNodeValue(const OleVariant Value);
```

Description

SetNodeValue is the write implementation of the NodeValue (see page 7695) property.

Value is the value to assign.

See Also

NodeValue (see page 7695)

81.2.2.2.35 IXMLNode.SetReadOnly Method

Specifies whether the node can be modified.

Pascal

```
procedure SetReadOnly(const Value: Boolean);
```

C++

```
__fastcall SetReadOnly(const Boolean Value);
```

Description

SetReadOnly is the write implementation of the ReadOnly (see page 7696) property.

Value is the new read-only status of the node.

See Also

ReadOnly (see page 7696)

81.2.2.2.36 IXMLNode.SetText Method

Sets the text value of the node.

Pascal

```
procedure SetText(const Value: DOMString);
```

C++

```
__fastcall SetText(const DOMString Value);
```

Description

SetText is the write implementation for the Text (see page 7697) property.

Value is the new text value for the node.

See Also

Text (see page 7697)

81.2.2.2.37 TransformNode Method

This is the overview for the TransformNode method overload.

Overload List

	Name	Description
	IXMLNode.TransformNode (IXMLNode, IXMLDocument) (see page 7687)	Transforms the subtree rooted at this node, using a specified XSL interface.
	IXMLNode.TransformNode (IXMLNode, WideString) (see page 7687)	Transforms the subtree rooted at this node, using a specified XSL interface.

81.2.2.2.37.1 IXMLNode.TransformNode Method (IXMLNode, IXMLDocument)

Transforms the subtree rooted at this node, using a specified XSL interface.

Pascal

```
procedure TransformNode(const stylesheet: IXMLNode; var output: WideString); overload;
procedure TransformNode(const stylesheet: IXMLNode; const output: IXMLDocument); overload;
```

C++

```
__fastcall TransformNode(const IXMLNode stylesheet, BSTR output);
__fastcall TransformNode(const IXMLNode stylesheet, const IXMLDocument output);
```

Description

Call TransformNode to transform the XML (see page 7697) that is the value of the XML (see page 7697) property using the transformation specified by an XSL document.

stylesheet is a node in an XML (see page 7697) document that represents an XSL transformation. Typically, it is the document element of that document.

output represents the result of the transformation. If output is a WideString (see page 3467), TransformNode returns a string of XML (see page 7697). If output is the interface to an XML (see page 7697) document, TransformNode replaces the content of that document with the results of the transformation.

Note: TransformNode is only available when the DOMVendor used to parse this node's XML (see page 7697) document is MSXML.

See Also

XML (see page 7697)

81.2.2.3 IXMLNode Properties

The properties of the IXMLNode class are listed here.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.
R	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
R	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
R	NodeName (see page 7693)	Indicates the node name.
R	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
R	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
R	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
R	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
R	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

81.2.2.3.1 IXMLNode.AttributeNodes Property

Lists the interfaces for nodes that represent attributes of this node.

Pascal

```
property AttributeNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList AttributeNodes;
```

Description

Use AttributeNodes to access the attributes of this node using an IXMLNode (see page 7669) interface. Attributes (see page 7688) modify a node in an XML (see page 7697) document. For example, given the node for

```
<Movie genre="scifi" title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

If an attribute consists of a simple Name=Value pair, it is simpler to use the Attributes (see page 7688) property to read or set attribute values. AttributeNodes is an IXMLNodeList (see page 7703) interface, which can be used to

Read (see page 3634) the count of attributes before iterating over all the attributes of this node.

Add or delete attribute nodes from this node.

Access attributes through an XML (see page 7697) node interface (for example, to read the attribute names from the nodes).

Only element nodes have attributes, and not every element node has them. If the node does not have any attributes, AttributeNodes is the interface for an empty list.

Note: Attributes (see page 7688) are not considered child nodes of a node and can't be accessed using the ChildNodes (see page 7689) property.

See Also

ChildNodes (see page 7689)

Attributes (see page 7688)

81.2.2.3.2 IXMLNode.Attributes Property

Represents the values of this node's attributes.

Pascal

```
property Attributes [const AttrName: DOMString]: OleVariant;
```

C++

```
__property OleVariant Attributes[DOMString const AttrName];
```

Description

Use Attributes to read or set the values of this node's attributes. Attributes modify a node in an XML (see page 7697) document. For example, given the node for

```
<Movie genre="scifi" title="Star Wars"> ... </Movie>
```

both genre and title are attributes of the node, with values of "scifi" and "Star Wars", respectively.

The AttrName index gives the name of the attribute to access. In the previous example, "genre" and "title" are the names of the attributes. When setting Attributes, if you specify the name of an attribute that already exists for a node, you change that attribute's value. If you specify an attribute name that does not correspond to an existing attribute of the node, a new attribute is

created.

When reading Attributes, if you specify the name of an attribute that does not exist for the node, the value depends on the Options property of the document that contains the node. If Options includes doAttrNull, reading an attribute that does not exist for the node results in a Null Variant (see page 3421). When Options does not include doAttrNull, reading an attribute that does not exist results in an empty string.

The value of Attributes is the text of the specified attribute. When setting attributes, the value string is not parsed. Any markup in the supplied value (for example, an entity reference) is treated as literal text. If you set an attribute to nil (Delphi) or NULL (C++), it is removed from this node. Note that you can set an attribute's value to an empty string.

See Also

AttributeNodes (see page 7688)

GetAttributeNS (see page 7676)

SetAttributeNS (see page 7684)

Options (see page 7389)

Working with XML Nodes

81.2.2.3.3 IXMLNode.ChildNodes Property

Represents the child nodes of the node.

Pascal

```
property ChildNodes: IXMLNodeList;
```

C++

```
__property IXMLNodeList ChildNodes;
```

Description

Use ChildNodes to access the children of a node. ChildNodes is an IXMLNodeList (see page 7703) interface, which can be used to access individual child nodes or to add child nodes to or delete them from this node. Each child node specifies this node as its parent node.

Tip: To get or set the values of the child nodes rather than their interfaces, use the ChildValues (see page 7690) property instead.

Note: Attributes (see page 7688) are not considered child nodes of a node. To get the values of this node's attributes, use the Attributes (see page 7688) or AttributeNodes (see page 7688) property instead.

See Also

ChildValues (see page 7690)

AttributeNodes (see page 7688)

ParentNode (see page 7695)

Attributes (see page 7688)

Working with XML Nodes

81.2.2.3.4 IXMLNode.ChildValues Property

Represents the values of this node's child nodes.

Pascal

```
property ChildValues [const IndexOrName: OleVariant]: OleVariant;
```

C++

```
__property OleVariant ChildValues[OleVariant const IndexOrName];
```

Description

Use ChildValues to access the values of this node's children.

IndexOrName identifies the desired child node either by its interface, or by a string that is the local name of the child node. If IndexOrName does not identify a child of this node, trying to get or set ChildValues causes an exception.

The value of a child node depends on the type of the child node, as indicated in the following table:

NodeType (see page 7694)	Value
ntElement	If the element contains only text, this is that text value. Otherwise, it is the Value of the first child node.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.
If the child node is any other node type, reading or setting ChildValues causes an exception.	

Note: In Delphi, ChildValues is the default property for IXMLNode (see page 7669). That means that you can omit the property name when using this property. For example

```
Edit1.Text := XMLDocument1.DocumentElement.ChildValues['Name'];
```

can also be written

```
Edit1.Text := XMLDocument1.DocumentElement['Name'];
```

Note: Attributes (see page 7688) are not considered child nodes of a node. To get or set the values of this node's attributes, use the Attributes (see page 7688) property instead.

See Also

ChildNodes (see page 7689)

ParentNode (see page 7695)

Attributes (see page 7688)

81.2.2.3.5 IXMLNode.Collection Property

Indicates the collection in which this node appears.

Pascal

```
property Collection: IXMLNodeCollection;
```

C++

```
__property IXMLNodeCollection Collection;
```

Description

Read (see page 3634) Collection to determine whether this node is one of a repeating set of child nodes. The node specified by Collection represents the same node in the XML (see page 7697) document as this node's ParentNode (see page 7695) property, but it is specialized to work with this node and any other child nodes of the same type.

For example, consider the following XML (see page 7697) fragment:

```
<students>
  <student name="John Smith">
    <grade>incomplete</grade>
  </student>
  <student name="Jane Doe">
    <grade>B+</grade>
  </student>
  <auditor name="Bill Jones"></auditor>
</students>
```

If this node represents one of the <student> nodes, then ParentNode (see page 7695) represents the <students> node, and Collection represents a copy of the <students> node that is used for only those children that are <student> nodes, and not the <auditor> nodes.

If this node does not represent one of a set of repeating child elements, Collection is nil (Delphi) or NULL (C++).

See Also

ParentNode (see page 7695)

81.2.2.3.6 IXMLNode.DOMNode Property

Provides access to the IDOMNode (see page 7557) interface for this node.

Pascal

```
property DOMNode: IDOMNode;
```

C++

```
__property IDOMNode DOMNode;
```

Description

IDOMNode (see page 7557) is the interface for this node provided by the DOM implementation that parsed the XML (see page 7697) document. You can use this interface to work with the node using the lower-level properties and methods of the DOM.

Typically, applications use the IXMLNode (see page 7669) interface to work with a node rather than the IDOMNode (see page 7557) interface. For more information on DOM interfaces, see the DOM specification on the W3 website or the documentation provided by your DOM vendor.

81.2.2.3.7 IXMLNode.HasChildNodes Property

Indicates whether this node has any child nodes.

Pascal

```
property HasChildNodes: Boolean;
```

C++

```
__property Boolean HasChildNodes;
```

Description

HasChildNodes is true if the underlying DOM node has any child nodes, false if it has no children.

Note that in the underlying DOM implementation, given the following XML (see page 7697)

```
<Title> Understanding XML </Title>
```

the "Title" node has a single child (a text node with the value 'Understanding XML (see page 7697)').

See Also

ChildNodes (see page 7689)

DOMNode (see page 7691)

81.2.2.3.8 IXMLNode.IsTextElement Property

Indicates whether the node has a single text value.

Pascal

```
property IsTextElement: Boolean;
```

C++

```
__property Boolean IsTextElement;
```

Description

Read (see page 3634) IsTextElement to determine whether this node represents a single text value. For example, if the node represents the following:

```
<Title> Understanding XML </Title>
```

IsTextElement is true, because the node represents a tagged element and the text that is its value.

Note: In the underlying DOM implementation, nodes for which IsTextElement is true are element nodes that have a single child node that is a text node. IXMLNode (see page 7669) flattens this out so that you can use the element node to work with the text value (through the Text (see page 7697) property), rather than requiring you to use a separate node for the text value.

See Also

NodeType (see page 7694)

Text (see page 7697)

Working with XML Nodes

81.2.2.3.9 IXMLNode.LocalName Property

Indicates the name of the node without any namespace prefix.

Pascal

```
property LocalName: DOMString;
```

C++

```
__property DOMString LocalName;
```

Description

Read (see page 3634) LocalName to determine the local name of the underlying DOM node. LocalName is only defined for

nodes of type ntElement or ntAttrribute.

For element nodes, LocalName is derived from the name that appears in the XML (see page 7697) tag. For example, the local name for <xsi:Name First="John" Last="Doe"> is 'Name'. (The string "xsi" is called the prefix).

For attribute nodes, the local name is derived from the string that appears to the left of the equals sign in the attribute declaration. For example, the element <xsi:Name xsi:First="John" xsi:Last="Doe"> results in two attribute nodes, with the local names 'First' and 'Last'.

For any other node types, localName is nil (Delphi) or NULL (C++).

To get the name of a node when localName is not defined, use the NodeName (see page 7693) property instead. If the node's name has no namespace prefix, LocalName has the same value as NodeName (see page 7693).

See Also

NamespaceURI (see page 7693)

Prefix (see page 7696)

NodeName (see page 7693)

81.2.2.3.10 IXMLNode.NamespaceURI Property

Identifies the URI for the namespace used in the qualified node name.

Pascal

```
property NamespaceURI: DOMString;
```

C++

```
__property DOMString NamespaceURI;
```

Description

NamespaceURI is the URI for the namespace that should be used to interpret the node's name.

Only element and attribute nodes can have a namespace URI value. For any other node type, NamespaceURI is an empty string.

The namespace URI is only available if it is explicitly assigned when the node is created. By default, IXMLNode (see page 7669) does not assign namespaces to the new nodes it creates.

See Also

NodeName (see page 7693)

NodeType (see page 7694)

Prefix (see page 7696)

LocalName (see page 7692)

CreateDOMNode (see page 7488)

namespaceURI (see page 7571)

81.2.2.3.11 IXMLNode.NodeName Property

Indicates the node name.

Pascal

```
property NodeName: DOMstring;
```

C++

```
__property DOMstring NodeName;
```

Description

NodeName is the name of the underlying DOM node.

The node's name depends on the type of the node, as indicated in the following table:

NodeType (see page 7694)	NodeName
ntAttribute	The attribute name
ntElement	The tag name
ntText	'#text'
ntCData	'#cdata-section'
ntEntityRef	The name of the entity reference.
ntEntity	The entity name
ntProcessingInstr	the target of the processing instruction
ntComment	'#comment'
ntDocument	'#document'
ntDocType	The document type name
ntDocFragment	'#document-fragment'
ntNotation	The notation name

Note: If Prefix (see page 7696) and LocalName (see page 7692) are both defined for a node, the node name is the combination of these two values.

See Also

NodeType (see page 7694)

Prefix (see page 7696)

LocalName (see page 7692)

81.2.2.3.12 IXMLNode.NodeType Property

Indicates the type of the node.

Pascal

```
property NodeType: TNodeType;
```

C++

```
__property TNodeType NodeType;
```

Description

Read (see page 3634) NodeType to determine the type of the node. The type reflects the way the node is used in the XML (see page 7697) document.

See Also

IsTextElement (see page 7692)

81.2.2.3.13 IXMLNode.NodeValue Property

Specifies the value of this node.

Pascal

```
property NodeValue: OleVariant;
```

C++

```
__property OleVariant NodeValue;
```

Description

Use NodeValue to get or set the value of the node. The meaning of a node's value depends on the type of the node, as indicated in the following table:

NodeType (see page 7694)	Value
ntAttribute	The attribute value
ntElement	If the element contains only text, this is that text value. Otherwise, trying to read or set NodeValue causes an exception.
ntText	The text
ntCData	The content of the CDATA section.
ntProcessingInstr	The content of the processing instruction except for the target.
ntComment	The value (text) of the comment.

Trying to read or set the value of any other node type causes an exception.

See Also

ChildValues (see page 7690)

Text (see page 7697)

81.2.2.3.14 IXMLNode.OwnerDocument Property

Identifies the document in which this node appears.

Pascal

```
property OwnerDocument: IXMLDocument;
```

C++

```
__property IXMLDocument OwnerDocument;
```

Description

Read (see page 3634) OwnerDocument to obtain the IXMLDocument (see page 7640) interface of the document to which this node belongs.

See Also

ParentNode (see page 7695)

81.2.2.3.15 IXMLNode.ParentNode Property

Identifies the interface of the node's parent node.

Pascal

```
property ParentNode: IXMLNode;
```

C++

```
__property IXMLNode ParentNode;
```

Description

Read (see page 3634) ParentNode to access the node's parent node.

See Also

ChildNodes (see page 7689)

81.2.2.3.16 IXMLNode.Prefix Property

Identifies the namespace prefix of the node's name.

Pascal

```
property Prefix: DOMString;
```

C++

```
__property DOMString Prefix;
```

Description

Prefix is the namespace prefix of an element or attribute node's name. A namespace prefix is a symbolic name for a namespace URI.

For element nodes, the namespace prefix is derived from the name that appears in the XML (see page 7697) tag. For example, the namespace prefix for <xsi:Name First="John" Last="Doe"> is 'xsi'.

For attribute nodes, the namespace prefix is derived from the string that appears to the left of the equals sign in the attribute declaration. For example the element <xsi:Name xsn:First="John" Last="Doe"> results in two attribute nodes, with namespace prefixes of 'xsn' and '', respectively. Note that the attribute never inherits a namespace prefix from the node to which it is attached.

For any other node types, the namespace prefix is nil (Delphi) or NULL (C++).

See Also

LocalName (see page 7692)

NodeType (see page 7694)

NamespaceURI (see page 7693)

81.2.2.3.17 IXMLNode.ReadOnly Property

Specifies whether the node can be modified.

Pascal

```
property ReadOnly: Boolean;
```

C++

```
__property Boolean ReadOnly;
```

Description

ReadOnly specifies whether the value, child nodes, or attributes of the node can be changed. When ReadOnly is true, the node can't be modified. Read (see page 3634)-only nodes raise an exception when an attempt is made to modify them.

When ReadOnly is false, the node can be modified.

See Also

AttributeNodes (see page 7688)

ChildNodes (see page 7689)

NodeValue (see page 7695)

81.2.2.3.18 IXMLNode.Text Property

Represents the Text of the node.

Pascal

```
property Text: DOMString;
```

C++

```
__property DOMString Text;
```

Description

Text represents the text of the node. For example, if the node corresponds to

```
<Title> Understanding XML </Title>
```

The text of the node is the string 'Understanding XML (see page 7697)'.

Text is intended for use nodes where the IsTextElement (see page 7692) property is true. If IsTextElement (see page 7692) is false, then

If the node has no children, the value of Text is an empty string. Setting Text is this case result in a node where IsTextElement (see page 7692) istrue.

If the node has children (other than a single DOM text node), reading or setting Text causes an exception.

See Also

IsTextElement (see page 7692)

NodeValue (see page 7695)

Working with XML Nodes

81.2.2.3.19 IXMLNode.XML Property

Indicates the XML that corresponds to the subtree rooted at this node.

Pascal

```
property XML: DOMString;
```

C++

```
__property DOMString XML;
```

Description

XML is the XML that corresponds to the document fragment that consists of this node and any child nodes it contains.

Note: XML is only available when the DOMVendor used to parse this node's XML document is MSXML.

See Also

Text (see page 7697)

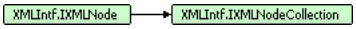
ChildValues (see page 7690)

DOMVendor (see page 7386)

81.2.3 IXMLNodeCollection Interface

IXMLNodeCollection is the interface for a set of child nodes that are all the same type.

Class Hierarchy



Pascal

```
IXMLNodeCollection = interface(IXMLNode);
```

C++

```
__interface IXMLNodeCollection : public IXMLNode;
```

File

XMLIntf

Description

Use IXMLNodeCollection to work with repeating elements in an XML (see page 7697) document. The XML (see page 7697) Data Binding wizard generates descendants of IXMLNodeCollection interface to represent sets of child nodes that are all of a particular type.

IXMLNodeCollection is a descendant of IXMLNode (see page 7669). As such, it provides generic support for working with a node in an XML (see page 7697) document. In addition, IXMLNodeCollection introduces properties and methods that manage a set of child nodes that are all the same type of node (that all have the same tag names or that are defined as the same type in an XML (see page 7697) schema).

Note: C++ method declarations that use IXMLNodeCollection use the _di_IXMLNodeCollection type instead. This type is a DelphiInterface (see page 3256) wrapper around the IXMLNodeCollection interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNodeCollection >
_di_IXMLNodeCollection;
```

See Also

TXMLDocument (see page 7347)

IXMLNode (see page 7669)

TXMLNodeCollection (see page 7444)

81.2.3.1 IXMLNodeCollection Members

The following tables list the members exposed by IXMLNodeCollection.

Public Methods

	Name	Description
	AddChild (see page 7672)	This is the overview for the AddChild method overload.
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.

	CloneNode (see page 7673)	Creates a copy of this node and returns its interface.
	DeclareNamespace (see page 7674)	Adds an attribute to the node that declares a specified namespace URI.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	FindNamespaceDecl (see page 7674)	Returns the attribute node that declares a specified Namespace URI.
	FindNamespaceURI (see page 7674)	Returns the namespace URI for a namespace prefix or qualified tag name.
	GetAttribute (see page 7675)	Returns the value of one of this node's attributes.
	GetAttributeNodes (see page 7675)	Returns the value of the AttributeNodes (see page 7688) property.
	GetAttributeNS (see page 7676)	Returns the value of a specified attribute.
	GetChildNodes (see page 7676)	Returns the value of the ChildNodes (see page 7689) property.
	GetChildValue (see page 7676)	Returns the value of a specified child node.
	GetCollection (see page 7677)	Returns the value of the Collection (see page 7690) property.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetDOMNode (see page 7677)	Returns the value of the DOMNode (see page 7691) property.
	GetHasChildNodes (see page 7678)	Indicates whether the node has any child nodes.
	GetIsTextElement (see page 7678)	Indicates whether the node has a single text value.
	GetLocalName (see page 7678)	Returns the name of the node without any namespace prefix.
	GetNamespaceURI (see page 7679)	Returns the namespace URI for the node's qualified name.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	GetNodeName (see page 7679)	Returns the name of the node.
	GetNodeType (see page 7679)	Returns the type of the node.
	GetNodeValue (see page 7680)	Returns the value of the node.
	GetOwnerDocument (see page 7680)	Returns the interface for the document in which this node appears.
	GetParentNode (see page 7680)	Returns the interface for the node's parent node.
	GetPrefix (see page 7680)	Returns the namespace prefix from the node's name.
	GetReadOnly (see page 7681)	Returns the value of the ReadOnly (see page 7696) property.
	GetText (see page 7681)	Returns the text value of the node.
	GetXML (see page 7681)	Returns the value of the XML (see page 7697) property.
	HasAttribute (see page 7682)	This is the overview for the HasAttribute method overload.
	NextSibling (see page 7682)	Returns the next child of this node's parent.
	Normalize (see page 7683)	Converts the subtree beneath this node to the structure it would have if it were just loaded from a document.
	PreviousSibling (see page 7683)	Returns the previous child of this node's parent.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.
	Resync (see page 7684)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SetAttribute (see page 7684)	Sets the value of one of this node's attributes.
	SetAttributeNS (see page 7684)	Sets the value of one of this node's attributes when you must explicitly specify the attribute's namespace.
	SetChildValue (see page 7685)	Sets the value of a specified child node.
	SetNodeValue (see page 7685)	Sets the value of this node.
	SetReadOnly (see page 7686)	Specifies whether the node can be modified.
	SetText (see page 7686)	Sets the text value of the node.
	TransformNode (see page 7686)	This is the overview for the TransformNode method overload.

Public Properties

	Name	Description
R	AttributeNodes (see page 7688)	Lists the interfaces for nodes that represent attributes of this node.
	Attributes (see page 7688)	Represents the values of this node's attributes.
R	ChildNodes (see page 7689)	Represents the child nodes of the node.
	ChildValues (see page 7690)	Represents the values of this node's child nodes.
R	Collection (see page 7690)	Indicates the collection in which this node appears.
R	Count (see page 7702)	Indicates the number of repeating child nodes.
R	DOMNode (see page 7691)	Provides access to the IDOMNode (see page 7557) interface for this node.
R	HasChildNodes (see page 7691)	Indicates whether this node has any child nodes.
R	IsTextElement (see page 7692)	Indicates whether the node has a single text value.

	LocalName (see page 7692)	Indicates the name of the node without any namespace prefix.
	NamespaceURI (see page 7693)	Identifies the URI for the namespace used in the qualified node name.
	NodeName (see page 7693)	Indicates the node name.
	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.
	NodeType (see page 7694)	Indicates the type of the node.
	NodeValue (see page 7695)	Specifies the value of this node.
	OwnerDocument (see page 7695)	Identifies the document in which this node appears.
	ParentNode (see page 7695)	Identifies the interface of the node's parent node.
	Prefix (see page 7696)	Identifies the namespace prefix of the node's name.
	ReadOnly (see page 7696)	Specifies whether the node can be modified.
	Text (see page 7697)	Represents the Text of the node.
	XML (see page 7697)	Indicates the XML that corresponds to the subtree rooted at this node.

81.2.3.2 IXMLNodeCollection Methods

The methods of the IXMLNodeCollection class are listed here.

Public Methods

	Name	Description
	Clear (see page 7700)	Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.
	Delete (see page 7700)	Removes one of the repeating child nodes from this node.
	GetCount (see page 7701)	Returns the value of the Count (see page 7702) property.
	GetNode (see page 7701)	Returns the interface for one of the repeating child nodes.
	Remove (see page 7702)	Removes one of the repeating child nodes from this node.

81.2.3.2.1 IXMLNodeCollection.Clear Method

Removes all child nodes in the Nodes (see page 7703) property from the ChildList and Nodes (see page 7703) properties.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Clear deletes every child node from the collection (all the nodes in the Nodes (see page 7703) property), along with the corresponding nodes in the ChildNodes (see page 7689) property.

See Also

Nodes (see page 7703)

Delete (see page 7700)

Remove (see page 7702)

ChildNodes (see page 7689)

81.2.3.2.2 IXMLNodeCollection.Delete Method

Removes one of the repeating child nodes from this node.

Pascal

```
procedure Delete(Index: Integer);
```

C++

```
__fastcall Delete(int Index);
```

Description

Delete removes a specified child node from this node.

Index identifies the child to remove. It is the index of the child in the Nodes (see page 7703) property list of repeating child nodes. The index 0 represents the first repetition, 1 represents the second repetition, and so on.

See Also

Remove (see page 7702)

AddChild (see page 7672)

Nodes (see page 7703)

Clear (see page 7700)

81.2.3.2.3 IXMLNodeCollection.GetCount Method

Returns the value of the Count (see page 7702) property.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 7702) property.

See Also

Count (see page 7702)

81.2.3.2.4 IXMLNodeCollection.GetNode Method

Returns the interface for one of the repeating child nodes.

Pascal

```
function GetNode(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(int Index);
```

Description

GetNode is the read implementation of the Nodes (see page 7703) property.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on.

See Also

Nodes (see page 7703)

81.2.3.2.5 IXMLNodeCollection.Remove Method

Removes one of the repeating child nodes from this node.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes a specified child node from the collection of repeating child nodes. It removes the node from both the collection, and from the ChildNodes (see page 7689) property, which lists all children of this node.

Node is the interface of the child to remove. It must represent on of the repeating child nodes.

To remove a child that is not one of the repeating child nodes, use the ChildNodes (see page 7689) property instead.

See Also

Delete (see page 7700)

ChildNodes (see page 7689)

Nodes (see page 7703)

81.2.3.3 IXMLNodeCollection Properties

The properties of the IXMLNodeCollection class are listed here.

Public Properties

	Name	Description
	Count (see page 7702)	Indicates the number of repeating child nodes.
	Nodes (see page 7703)	Lists the interfaces of the repeating child nodes.

81.2.3.3.1 IXMLNodeCollection.Count Property

Indicates the number of repeating child nodes.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3634) Count to obtain the number of repeating children of this node. Use Count to obtain an upper limit when addressing child nodes by index (for example, when iterating over all nodes in the Nodes (see page 7703) property list).

Count is changed when you add child nodes using the AddChild (see page 7672) method or delete them using the Delete (see page 7700) or Remove (see page 7702) method. You can access the individual repeating child nodes using the Nodes (see page 7703) property.

See Also

Nodes (see page 7703)

AddChild (see page 7672)

Remove (see page 7702)

Delete (see page 7700)

81.2.3.3.2 IXMLNodeCollection.Nodes Property

Lists the interfaces of the repeating child nodes.

Pascal

```
property Nodes [Index: Integer]: IXMLNode;
```

C++

```
__property IXMLNode Nodes[int Index];
```

Description

Use Nodes to access nodes from the set of repeating child nodes. The value of Nodes is the IXMLNode (see page 7669) interface that represents a specified child node.

Index identifies which repetition of the child node is desired, where 0 is the first repetition, 1 is the second repetition, and so on.

Note: The XML (see page 7697) Data Binding wizard generates a nearly identical property in IXMLNodeCollection (see page 7698) descendants called Items. Items differs from Nodes in that the value is an IXMLNode (see page 7669) descendant specific to the type of the repeating element, rather than the generic IXMLNode (see page 7669) interface.

Note: In Delphi, ChildNodes (see page 7689) is the default property for IXMLNodeCollection (see page 7698). That means that you can omit the property name when using this property. In IXMLNodeCollection (see page 7698) descendants that are generated by the XML (see page 7697) Data Binding wizard, the Items property is the default property instead.

Note: As a descendant of IXMLNode (see page 7669), IXMLNodeCollection (see page 7698) also inherits the ChildNodes (see page 7689) property, which lists all the child nodes for this node. Nodes differs from ChildNodes (see page 7689) in that it contains only those child nodes that are in the repeating collection.

See Also

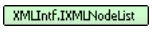
Count (see page 7702)

ChildNodes (see page 7689)

81.2.4 IXMLNodeList Interface

IXMLNodeList represents a set of nodes in an XML document.

Class Hierarchy



Pascal

```
IXMLNodeList = interface;
```

C++

```
__interface IXMLNodeList;
```

File

XMLIntf

Description

Use IXMLNodeList to work with a set of related nodes in an XML document. IXMLNodeList can be used to add or delete nodes from the set, or to access specific nodes.

Note: C++ method declarations that use IXMLNodeList use the _di_IXMLNodeList type instead. This type is a DelphiInterface (see page 3256) wrapper around the IXMLNodeList interface:

```
typedef System::DelphiInterfaceDelphiInterface_object< IXMLNodeList >  _di_IXMLNodeList;
```

See Also

TXMLNodeList (see page 7460)

IXMLNode (see page 7669)

TXMLDocument (see page 7347)

81.2.4.1 IXMLNodeList Members

The following tables list the members exposed by IXMLNodeList.

Public Methods

	Name	Description
	Add (see page 7705)	Adds a new node to the end of the list.
	BeginUpdate (see page 7705)	Increments the value of UpdateCount (see page 7715).
	Clear (see page 7706)	Removes all nodes from the list.
	Delete (see page 7706)	This is the overview for the Delete method overload.
	EndUpdate (see page 7707)	Decrements the value of UpdateCount (see page 7715).
	FindNode (see page 7708)	This is the overview for the FindNode method overload.
	FindSibling (see page 7709)	Returns a node that appears a specified amount before or after another node.
	First (see page 7709)	Returns the first node in the list.
	Get (see page 7710)	Returns a specified node in the list.
	GetCount (see page 7710)	Returns the number of nodes in the list.
	GetNode (see page 7710)	Returns a specified node from the list.
	GetUpdateCount (see page 7711)	Returns the value of the UpdateCount (see page 7715) property.
	IndexOf (see page 7711)	This is the overview for the IndexOf method overload.
	Insert (see page 7712)	Inserts a new node into a specified position in the list.
	Last (see page 7712)	Returns the last node in the list.
	Remove (see page 7713)	Removes a specified node from the list.
	ReplaceNode (see page 7713)	Replaces a specified node in the list with another node.

Public Properties

	Name	Description
R	Count (see page 7714)	Indicates the number of nodes in the list.
R	Nodes (see page 7714)	Provides access to the nodes in the list.
R	UpdateCount (see page 7715)	Indicates the number of times an application has called BeginUpdate (see page 7705) without a matching call to EndUpdate (see page 7707).

81.2.4.2 IXMLNodeList Methods

The methods of the IXMLNodeList class are listed here.

Public Methods

	Name	Description
	Add (see page 7705)	Adds a new node to the end of the list.
	BeginUpdate (see page 7705)	Increments the value of UpdateCount (see page 7715).
	Clear (see page 7706)	Removes all nodes from the list.
	Delete (see page 7706)	This is the overview for the Delete method overload.
	EndUpdate (see page 7707)	Decrements the value of UpdateCount (see page 7715).
	FindNode (see page 7708)	This is the overview for the FindNode method overload.
	FindSibling (see page 7709)	Returns a node that appears a specified amount before or after another node.
	First (see page 7709)	Returns the first node in the list.
	Get (see page 7710)	Returns a specified node in the list.
	GetCount (see page 7710)	Returns the number of nodes in the list.
	GetNode (see page 7710)	Returns a specified node from the list.
	GetUpdateCount (see page 7711)	Returns the value of the UpdateCount (see page 7715) property.
	IndexOf (see page 7711)	This is the overview for the IndexOf method overload.
	Insert (see page 7712)	Inserts a new node into a specified position in the list.
	Last (see page 7712)	Returns the last node in the list.
	Remove (see page 7713)	Removes a specified node from the list.
	ReplaceNode (see page 7713)	Replaces a specified node in the list with another node.

81.2.4.2.1 IXMLNodeList.Add Method

Adds a new node to the end of the list.

Pascal

```
function Add(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Add(const IXMLNode Node);
```

Description

Call Add to add a node to the end of the list. Add returns the index of the node once it is added, where 0 is the index of the first node in the list, 1 is the index of the second node, and so on.

Node is the node to add to the list.

See Also

Delete (see page 7707)

Remove (see page 7713)

IndexOf (see page 7711)

Insert (see page 7712)

Nodes (see page 7714)

81.2.4.2.2 IXMLNodeList.BeginUpdate Method

Increments the value of UpdateCount (see page 7715).

Pascal

```
procedure BeginUpdate;
```

C++

```
__fastcall BeginUpdate();
```

Description

Call BeginUpdate before making several changes to the nodes in the list (such as adding or removing several nodes). BeginUpdate increments the UpdateCount (see page 7715) property, to keep track of the fact that the list is in the middle of an update. When the update is complete, call EndUpdate (see page 7707) to decrement the UpdateCount (see page 7715) property.

Every call to BeginUpdate should be matched by a call to EndUpdate (see page 7707), so that the UpdateCount (see page 7715) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7715)

EndUpdate (see page 7707)

81.2.4.2.3 IXMLNodeList.Clear Method

Removes all nodes from the list.

Pascal

```
procedure Clear;
```

C++

```
__fastcall Clear();
```

Description

Call Clear to empty the list.

Note: Clear does not call the BeginUpdate (see page 7705) and EndUpdate (see page 7707) methods, even though it may result in the deletion of more than one node.

See Also

Delete (see page 7707)

Remove (see page 7713)

Nodes (see page 7714)

81.2.4.2.4 Delete Method

This is the overview for the Delete method overload.

Overload List

	Name	Description
	IXMLNodeList.Delete (DOMString) (see page 7707)	Removes a specified node from the list.
	IXMLNodeList.Delete (DOMString, DOMString) (see page 7707)	Removes a specified node from the list.

	IXMLNodeList.Delete (Integer) (see page 7707)	Removes a specified node from the list.

81.2.4.2.4.1 IXMLNodeList.Delete Method (DOMString)

Removes a specified node from the list.

Pascal

```
function Delete(const Index: Integer): Integer; overload;
function Delete(const Name: DOMString): Integer; overload;
function Delete(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int Delete(const int Index);
__fastcall int Delete(const DOMString Name);
__fastcall int Delete(const DOMString Name, const DOMString NamespaceURI);
```

Description

Delete removes the node specified by the Index or Name parameter.

Name identifies the node to remove from the list. This is the local name of the node to remove.

NamespaceURI identifies the namespace to use when interpreting Name (if necessary).

Index identifies the node to remove by index rather than name. Index ranges from 0 to one less than the value of the Count (see page 7714) property.

Delete returns the index of the node that was removed. If there was no node that matched the value of Index or Name, Delete returns –1.

See Also

Remove (see page 7713)

Clear (see page 7706)

IndexOf (see page 7711)

LocalName (see page 7692)

NamespaceURI (see page 7693)

Working with XML Nodes

81.2.4.2.5 IXMLNodeList.EndUpdate Method

Decrements the value of UpdateCount (see page 7715).

Pascal

```
procedure EndUpdate;
```

C++

```
__fastcall EndUpdate();
```

Description

Call EndUpdate after making several changes to the nodes in the list (such as adding or removing several nodes). EndUpdate decrements the UpdateCount (see page 7715) property, which was incremented by a previous call to BeginUpdate (see page 7705).

Every call to BeginUpdate (see page 7705) should be matched by a call to EndUpdate, so that the UpdateCount (see page

7715) property accurately reflects whether an update operation is underway.

See Also

UpdateCount (see page 7715)

BeginUpdate (see page 7705)

81.2.4.2.6 FindNode Method

This is the overview for the FindNode method overload.

Overload List

	Name	Description
	IXMLNodeList.FindNode (DOMString) (see page 7708)	Returns a specified node from the list.
	IXMLNodeList.FindNode (DOMString, DOMString) (see page 7708)	Returns a specified node from the list.
	IXMLNodeList.FindNode (TGuid) (see page 7708)	Returns a specified node from the list.

81.2.4.2.6.1 IXMLNodeList.FindNode Method (DOMString)

Returns a specified node from the list.

Pascal

```
function FindNode(NodeName: DOMString): IXMLNode; overload;
function FindNode(NodeName: DOMString; NamespaceURI: DOMString): IXMLNode; overload;
function FindNode(ChildNodeType: TGuid): IXMLNode; overload;
```

C++

```
__fastcall IXMLNode FindNode(DOMString NodeName);
__fastcall IXMLNode FindNode(DOMString NodeName, DOMString NamespaceURI);
__fastcall IXMLNode FindNode(TGuid ChildNodeType);
```

Description

Call FindNode to access a particular node in the list.

NodeName is the node to access. It specifies the LocalName property of the desired node.

NamespaceURI identifies the namespace that qualifies NodeName. It is the same as the NamespaceURI property of the desired node.

ChildNodeType identifies the node to access by its interface type (GUID), rather than by name.

FindNode returns the interface for the node if it is in the list. If NodeName or ChildNodeType does not specify a node in the list, FindNode returns nil (Delphi) or NULL (C++).

See Also

FindSibling (see page 7709)

First (see page 7709)

Last (see page 7712)

IndexOf (see page 7711)

LocalName (see page 7692)

Nodes (see page 7714)

NamespaceURI (see page 7693)

81.2.4.2.7 IXMLNodeList.FindSibling Method

Returns a node that appears a specified amount before or after another node.

Pascal

```
function FindSibling(const Node: IXMLNode; Delta: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode FindSibling(const IXMLNode Node, int Delta);
```

Description

Call FindSibling to access the node whose position has a specified relationship to another node.

Node is a node in the list to use as a reference point.

Delta indicates where the desired node appears, relative to Node. If Delta is positive, FindSibling returns the node that appears Delta positions after Node. If Delta is negative, FindSibling returns a node that appears before Node.

FindSibling returns the node that appears at the position offset by Delta, relative to the position of Node. If Delta specifies a position before the first node or after the last node in the list, FindSibling returns nil (Delphi) or NULL (C++).

See Also

FindNode (see page 7708)

First (see page 7709)

Last (see page 7712)

IndexOf (see page 7711)

81.2.4.2.8 IXMLNodeList.First Method

Returns the first node in the list.

Pascal

```
function First: IXMLNode;
```

C++

```
__fastcall IXMLNode First();
```

Description

Call First to access the first node in the list. If the list is empty, First raises an exception.

See Also

FindNode (see page 7708)

FindSibling (see page 7709)

Last (see page 7712)

Nodes (see page 7714)

81.2.4.2.9 IXMLNodeList.Get Method

Returns a specified node in the list.

Pascal

```
function Get(Index: Integer): IXMLNode;
```

C++

```
__fastcall IXMLNode Get(int Index);
```

Description

Call Get to retrieve a node from the list, given its index.

Index specifies the node to fetch, where 0 identifies the first node, 1 identifies the second node, and so on. Index should be less than the value of the Count (see page 7714) property.

See Also

Nodes (see page 7714)

GetNode (see page 7710)

Count (see page 7714)

81.2.4.2.10 IXMLNodeList.GetCount Method

Returns the number of nodes in the list.

Pascal

```
function GetCount: Integer;
```

C++

```
__fastcall int GetCount();
```

Description

GetCount is the read implementation of the Count (see page 7714) property.

See Also

Count (see page 7714)

81.2.4.2.11 IXMLNodeList.GetNode Method

Returns a specified node from the list.

Pascal

```
function GetNode(const IndexOrName: OleVariant): IXMLNode;
```

C++

```
__fastcall IXMLNode GetNode(const OleVariant IndexOrName);
```

Description

GetNode is the read implementation of the Nodes (see page 7714) property.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, GetNode tries to create a new node with the name specified by IndexOrName. If it can't create the new node, GetNode raises an exception.

See Also

FindNode (see page 7708)

Nodes (see page 7714)

LocalName (see page 7692)

Get (see page 7710)

81.2.4.2.12 IXMLNodeList.GetUpdateCount Method

Returns the value of the UpdateCount (see page 7715) property.

Pascal

```
function GetUpdateCount: Integer;
```

C++

```
__fastcall int GetUpdateCount();
```

Description

GetUpdateCount is the read implementation of the UpdateCount (see page 7715) property.

See Also

UpdateCount (see page 7715)

81.2.4.2.13 IndexOf Method

This is the overview for the IndexOf method overload.

Overload List

	Name	Description
	IXMLNodeList.IndexOf (DOMString) (see page 7711)	Returns the index of a specified node.
	IXMLNodeList.IndexOf (DOMString, DOMString) (see page 7711)	Returns the index of a specified node.
	IXMLNodeList.IndexOf (IXMLNode) (see page 7711)	Returns the index of a specified node.

81.2.4.2.13.1 IXMLNodeList.IndexOf Method (DOMString)

Returns the index of a specified node.

Pascal

```
function IndexOf(const Node: IXMLNode): Integer; overload;
function IndexOf(const Name: DOMString): Integer; overload;
function IndexOf(const Name: DOMString; const NamespaceURI: DOMString): Integer; overload;
```

C++

```
__fastcall int IndexOf(const IXMLNode Node);
```

```
__fastcall int IndexOf(const DOMString Name);
__fastcall int IndexOf(const DOMString Name, const DOMString NamespaceURI);
```

Description

Call IndexOf to locate a node in the list. You can specify which node to locate using its name or its interface.

Name is the LocalName property of the node to locate.

NamespaceURI identifies the namespace to use when interpreting LocalName (if necessary).

Node is the interface of the node to locate.

IndexOf returns the index of the specified node, where 0 is the index of the first node, 1 is the index of the second node, and so on. If the specified node is not in the list, IndexOf returns -1.

See Also

FindNode (see page 7708)

LocalName (see page 7692)

Nodes (see page 7714)

NamespaceURI (see page 7693)

81.2.4.2.14 IXMLNodeList.Insert Method

Inserts a new node into a specified position in the list.

Pascal

```
procedure Insert(Index: Integer; const Node: IXMLNode);
```

C++

```
__fastcall Insert(int Index, const IXMLNode Node);
```

Description

Call Insert to add a node at the position specified by Index.

Index specifies where to insert the node, where 0 is the first position, 1 is second position, and so on. If Index does not specify a valid index, Insert raises an exception.

Node is the node to add to the list.

See Also

Delete (see page 7707)

Remove (see page 7713)

IndexOf (see page 7711)

Add (see page 7705)

Nodes (see page 7714)

81.2.4.2.15 IXMLNodeList.Last Method

Returns the last node in the list.

Pascal

```
function Last: IXMLNode;
```

C++

```
__fastcall IXMLNode Last();
```

Description

Call Last to access the last node in the list. If the list is empty, Last raises an exception.

See Also

FindNode (see page 7708)

FindSibling (see page 7709)

First (see page 7709)

Nodes (see page 7714)

81.2.4.2.16 IXMLNodeList.Remove Method

Removes a specified node from the list.

Pascal

```
function Remove(const Node: IXMLNode): Integer;
```

C++

```
__fastcall int Remove(const IXMLNode Node);
```

Description

Remove removes the specified node from the list.

Node is the node to remove from the list.

Remove returns the index of Node before it was removed. If node is not a node in the list, Remove returns -1.

See Also

Delete (see page 7707)

Clear (see page 7706)

81.2.4.2.17 IXMLNodeList.ReplaceNode Method

Replaces a specified node in the list with another node.

Pascal

```
function ReplaceNode(const OldNode: IXMLNode; const NewNode: IXMLNode): IXMLNode;
```

C++

```
__fastcall IXMLNode ReplaceNode(const IXMLNode OldNode, const IXMLNode NewNode);
```

Description

Call ReplaceNode to replace the node specified by OldNode with the node specified by NewNode.

OldNode is the node to replace. If OldNode does not appear in the list, then ReplaceNode adds the new node to the end of the list.

NewNode is the node to add to the list in place of OldNode.

ReplaceNode returns OldNode (even if OldNode did not appear in the list).

See Also

Remove (see page 7713)

Insert (see page 7712)

81.2.4.3 IXMLNodeList Properties

The properties of the IXMLNodeList class are listed here.

Public Properties

	Name	Description
	Count (see page 7714)	Indicates the number of nodes in the list.
	Nodes (see page 7714)	Provides access to the nodes in the list.
	UpdateCount (see page 7715)	Indicates the number of times an application has called BeginUpdate (see page 7705) without a matching call to EndUpdate (see page 7707).

81.2.4.3.1 IXMLNodeList.Count Property

Indicates the number of nodes in the list.

Pascal

```
property Count: Integer;
```

C++

```
__property int Count;
```

Description

Read (see page 3634) Count to determine the number of nodes in the list. Count provides an upper limit to the indexes that can be used with the Nodes (see page 7714) property.

To change the value of Count, add nodes using the Add (see page 7705) or Insert (see page 7712) method, or remove them using the Delete (see page 7707) or Remove (see page 7713) method.

See Also

Nodes (see page 7714)

Add (see page 7705)

Delete (see page 7707)

Remove (see page 7713)

Insert (see page 7712)

81.2.4.3.2 IXMLNodeList.Nodes Property

Provides access to the nodes in the list.

Pascal

```
property Nodes [const IndexOrName: OleVariant]: IXMLNode;
```

C++

```
__property IXMLNode Nodes[OleVariant const IndexOrName];
```

Description

Read (see page 3634) Nodes to access a specified node in the list.

IndexOrName identifies the desired node. It can be

The index of the node, where 0 is the index of the first node, 1 is the index of the second node, and so on. The Count (see page 7714) property provides an upper bound on the indexes you can specify.

The LocalName property of a node in the list.

If IndexOrName does not identify a node in the list, and if the document that contains this node list's parent includes doNodeAutoCreate in its Options property, then the node list tries to create a new node with the name specified by IndexOrName. If the node list can't create the new node, it raises an exception.

See Also

Count (see page 7714)

FindNode (see page 7708)

LocalName (see page 7692)

Options (see page 7389)

81.2.4.3.3 IXMLNodeList.UpdateCount Property

Indicates the number of times an application has called BeginUpdate (see page 7705) without a matching call to EndUpdate (see page 7707).

Pascal

```
property UpdateCount: Integer;
```

C++

```
__property int UpdateCount;
```

Description

UpdateCount keeps track of whether all update operations have been completed. When an application starts making changes to the list, it calls BeginUpdate (see page 7705), which increments UpdateCount. When those changes are complete, it calls EndUpdate (see page 7707), which decrements UpdateCount. Thus, UpdateCount allows calls to BeginUpdate (see page 7705) and EndUpdate (see page 7707) to be nested.

Applications can avoid responding to updates in a piecemeal fashion by checking UpdateCount and only responding when it is 0.

See Also

BeginUpdate (see page 7705)

EndUpdate (see page 7707)

81.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TNodeType (see page 7716)	TNodeType identifies the type of a node in an XML document.
	TParseOption (see page 7718)	TParseOption and TParseOptions (see page 7721) represent implementation-specific options that are followed when parsing the XML document.
	TXMLDocOption (see page 7718)	TXMLDocOption and TXMLDocOptions (see page 7722) indicate types of information that an XML document automatically supplies.
	TXMLEncodingType (see page 7719)	TXMLEncodingType indicates the character set used to encode characters in an XML document.

81.3.1 XMLIntf.TNodeType Enumeration

TNodeType identifies the type of a node in an XML document.

Pascal

```
TNodeType = (
  ntReserved,
  ntElement,
  ntAttribute,
  ntText,
  ntCData,
  ntEntityRef,
  ntEntity,
  ntProcessingInstr,
  ntComment,
  ntDocument,
  ntDocType,
  ntDocFragment,
  ntNotation
);
```

C++

```
enum TNodeType {
  ntReserved,
  ntElement,
  ntAttribute,
  ntText,
  ntCData,
  ntEntityRef,
  ntEntity,
  ntProcessingInstr,
  ntComment,
  ntDocument,
  ntDocType,
  ntDocFragment,
  ntNotation
};
```

File

XMLIntf

Description

TNodeType identifies the type of a node in an XML document. The following table lists the possible values:

Value	Meaning
ntReserved	Not used
ntElement	The node represents an element. Element nodes represent simple tags that have child nodes. (Note that sometimes these child nodes do not appear when using IXMLNode (see page 7669). For example child nodes of type ntText are typically hidden by IXMLNode (see page 7669) and appear only as the value of the Text property). The child nodes of an element node can have the following node types: ntElement, ntText, ntCData, ntEntityRef, ntProcessingInstr, and ntComment. Element nodes can also have attributes (ntAttribute). Element nodes can be the child of a node of type ntDocument, ntDocFragment, ntEntityRef, and ntElement.
ntAttribute	The node represents an attribute of an element. It is not the child of another node, but its value can be accessed using the Attribute property of the element node's IXMLNode (see page 7669) interface. An attribute node can have child nodes of type ntText and ntEntityRef.
ntText	The node represents the text content of a tag. A text node cannot have any child nodes, but can appear as the child node of a node of type ntAttribute, ntDocFragment, ntElement, or ntEntityRef.
ntCData	The node represents a CDATA section in the XML source. CDATA sections identify blocks of text that would otherwise be interpreted as markup. An ntCData node can't have any child nodes. They can appear as the child of an ntDocFragment, ntEntityRef, or ntElement node.
ntEntityRef	The node represents a reference to an entity in the XML document. This can be any type of entity, including character entity references. The children of an entity reference node can be of the following types: ntElement, ntProcessingInstr, ntComment, ntText, ntCData, and ntEntityRef. The entity reference node can appear as the child of an ntAttribute, ntDocFragment, ntElement, or ntEntityRef node.
ntEntity	The node represents an expanded entity. Entity nodes can have child nodes that represent the expanded entity (for example, ntText and ntEntityRef nodes). Entity nodes only appear as the child of an ntDocType node.
ntProcessingInstr	The node represents a processing instruction (PI) from the XML document. A PI node cannot have any child nodes, but can appear as the child node of a node of type ntDocument, ntDocFragment, ntElement, or ntEntityRef.
ntComment	The node represents a comment in the XML document. Comment nodes do not have child nodes. They appear as the child of an ntDocument, ntDocFragment, ntElement, or ntEntityRef node.
ntDocument	The node represents a document object, which is the root of the entire XML document. Document nodes have a single ntElement node as a child (the DocumentElement). In addition, they can have child nodes of type ntProcessingInstr, ntComment, and ntDocType. Because the document is the root of the XML document, it never appears as a child node.
ntDocType	The node represents the document type declaration, indicated by the <!DOCTYPE > tag. The document type node can child nodes of type ntNotation and ntEntity. It always appears as the child of the document node.
ntDocFragment	The node represents a document fragment. A document fragment node associates a node or subtree with a document without actually being contained in the document. Document fragment nodes can have child nodes of type ntElement, ntProcessingInstr, ntComment, ntText, ntCData, and ntEntityRef. It never appears as the child of another node.
ntNotation	A node represents a notation in the document type declaration. It always appears as the child of an ntDocType node and never has any child nodes.

81.3.2 XMLIntf.TParseOption Enumeration

TParseOption and TParseOptions (see page 7721) represent implementation-specific options that are followed when parsing the XML document.

Pascal

```
TParseOption = (
  poResolveExternals,
  poValidateOnParse,
  poPreserveWhiteSpace,
  poAsyncLoad
);
```

C++

```
enum TParseOption {
  poResolveExternals,
  poValidateOnParse,
  poPreserveWhiteSpace,
  poAsyncLoad
};
```

File

XMLIntf

Description

TParseOptions (see page 7721) is a set of TParseOption values. Each TParseOption value provides information about how a DOM parser should parse an XML document. Different DOM implementations support different options.

The following table lists the possible values:

OptionDOM VendorMeaning

Option	DOM Vendor	Meaning
poResolveExternals	MSXML	External definitions (resolvable namespaces, DTD external subsets, and external entity references) are resolved at parse time.
poValidateOnParse	MSXML	The DOM parser validates the XML document against its schema information as well as verifying that it is well-formed XML.
poPreserveWhiteSpace	MSXML	White space in the text of the XML document is not stripped off.
poAsyncLoad	MSXML	The DOM parser works asynchronously. This can speed up performance, especially when parsing a large XML document, but can also lead to exceptions if the application tries to access nodes in the document before it is completely parsed.

81.3.3 XMLIntf.TXMLDocOption Enumeration

TXMLDocOption and TXMLDocOptions (see page 7722) indicate types of information that an XML document automatically supplies.

Pascal

```
TXMLDocOption = (
```

```
    doNodeAutoCreate,
    doNodeAutoIndent,
    doAttrNull,
    doAutoPrefix,
    doNamespaceDecl,
    doAutoSave
  );
```

C++

```
  enum TXMLDocOption {
    doNodeAutoCreate,
    doNodeAutoIndent,
    doAttrNull,
    doAutoPrefix,
    doNamespaceDecl,
    doAutoSave
  };
```

File

XMLIntf

Description

TXMLDocOptions (see page 7722) is a set of TXMLDocOption values. The following table lists the TXMLDocOption values:

Value	Meaning
doNodeAutoCreate	If the application tries to read a node by name, using the Nodes property of an IXMLNodeList (see page 7703) interface, and that node does not exist, then the application creates a new node using the specified name.
doNodeAutoIndent	When formatting the XML text from the parsed set of nodes, child nodes are automatically indented from their parent nodes.
doAttrNull	When reading the value of an attribute that does not exist, the value is given as a Null Variant (see page 3421) (as opposed to a value of an empty string).
doAutoPrefix	When generating the name of a new node, the value is automatically given a namespace prefix if a namespace URI is supplied.
doNamespaceDecl	When the AddChild method must create a new namespace for the node, it also adds a namespace attribute to declare that new namespace. If doNamespaceDecl is not included, AddChild creates the new namespace but does not add an attribute to declare it.
doAutoSave	When the XML document closes, any changes are automatically saved back to the XML document file or to the XML property.

81.3.4 XMLIntf.TXMLEncodingType Enumeration

TXMLEncodingType indicates the character set used to encode characters in an XML document.

Pascal

```
  TXMLEncodingType = (
    xetUnknown,
    xetUCS_4BE,
    xetUCS_4LE,
    xetUCS_4Order2134,
    xetUCS_4Order3412,
    xetUTF_16BE,
    xetUTF_16LE,
    xetUTF_8,
```

```
    xetUCS_4Like,
    xetUTF_16BELike,
    xetUTF_16LELike,
    xetUTF_8Like,
    xetEBCDICLike
  );
```

C++

```
  enum TXMLEncodingType {
    xetUnknown,
    xetUCS_4BE,
    xetUCS_4LE,
    xetUCS_4Order2134,
    xetUCS_4Order3412,
    xetUTF_16BE,
    xetUTF_16LE,
    xetUTF_8,
    xetUCS_4Like,
    xetUTF_16BELike,
    xetUTF_16LELike,
    xetUTF_8Like,
    xetEBCDICLike
  };
```

File

XMLIntf

Description

The following table lists the possible TXMLEncodingType values:

Value	Character encoding
xetUnknown	The character set is unknown.
xetUCS_4BE	UCS-4, big-endian (1234).
xetUCS_4LE	UCS-4, little-endian (4321).
xetUCS_4Order2134	UCS-4, unusual octet order (2134)
xetUCS_4Order3412	UCS-4, unusual octet order (3412)
xetUTF_16BE	UTF-16, big-endian order
xetUTF_16LE	UTF-16, little-endian order
xetUTF_8	UTF-8
xetUCS_4Like	Any UCS-4 encoding
xetUTF_16BELike	Any UTF-16, big-endian encoding, may be without byte order mark
xetUTF_16LELike	Any UTF-16, little-endian encoding, my be without byte order mark
xetUTF_8Like	UTF-8, ISO 646, ASCII, or other 7- or 8-bit encoding, including multi-byte character sets such as Shift-JIS.
xetEBCDICLike	EBCDIC

81.4 Types

The following table lists types in this documentation.

Types

Name	Description
TParseOptions (see page 7721)	TParseOptions and TParseOption (see page 7718) represent implementation-specific options that are followed when parsing the XML document.
TXMLDocOptions (see page 7722)	This is type XMLIntf.TXMLDocOptions.

81.4.1 XMLIntf.TParseOptions Type

TParseOptions and TParseOption (see page 7718) represent implementation-specific options that are followed when parsing the XML document.

Pascal

```
TParseOptions = set of TParseOption;
```

C++

```
TParseOption TParseOptions;
```

File

XMLIntf

Description

TParseOptions is a set of TParseOption (see page 7718) values. Each TParseOption (see page 7718) value provides information about how a DOM parser should parse an XML document. Different DOM implementations support different options.

The following table lists the possible values:

OptionDOM VendorMeaning

Option	DOM Vendor	Meaning
poResolveExternals	MSXML	External definitions (resolvable namespaces, DTD external subsets, and external entity references) are resolved at parse time.
poValidateOnParse	MSXML	The DOM parser validates the XML document against its schema information as well as verifying that it is well-formed XML.
poPreserveWhiteSpace	MSXML	White space in the text of the XML document is not stripped off.
poAsyncLoad	MSXML	The DOM parser works asynchronously. This can speed up performance, especially when parsing a large XML document, but can also lead to exceptions if the application tries to access nodes in the document before it is completely parsed.

See Also

TParseOption (see page 7718)

81.4.2 XMLIntf.TXMLDocOptions Type

Pascal

```
TXMLDocOptions = set of TXMLDocOption;
```

C++

```
TXMLDocOption TXMLDocOptions;
```

File

XMLIntf

Description

This is type XMLIntf.TXMLDocOptions.

81.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
NodeTypeNames (see page 7723)	This is constant XMLIntf.NodeTypeNames.

81.5.1 XMLIntf.NodeTypeNames Constant

Pascal

```
NodeTypeNames: array[TNodeType] of string = ('', 'Element','Attribute','Text','CDATASection',
'EntityRef','Entity', 'ProcessingInstr', 'Comment','Document','DocumentType',
'DocumentFragment','Notation');
```

C++

```
array[TNodeType] of string NodeTypeNames = ('', 'Element','Attribute','Text','CDATASection',
'EntityRef','Entity', 'ProcessingInstr', 'Comment','Document','DocumentType',
'DocumentFragment','Notation');
```

File

XMLIntf

Description

This is constant XMLIntf.NodeTypeNames.

82 Xmlxform Namespace

This is namespace Xmlxform.

Topics

Name	Description
Classes (see page 7726)	The following table lists classes in this documentation.
Types (see page 7761)	The following table lists types in this documentation.

Classes

	Name	Description
	TranslateException (see page 7726)	TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.
	TXMLTransform (see page 7727)	TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.
	TXMLTransformClient (see page 7739)	TXMLTransformClient acts as an adapter between an XML document and a provider component.
	TXMLTransformProvider (see page 7747)	TXMLTransformProvider provides data from and resolves updates to an XML document.

Types

Name	Description
TRowEvent (see page 7761)	TRowEvent is the type for event handlers of events that occur at discrete points during an XML transformation.
TTranslateEvent (see page 7762)	TTranslateEvent is the type for the OnTranslate event handler.

82.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TranslateException (see page 7726)	TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.
	TXMLTransform (see page 7727)	TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.
	TXMLTransformClient (see page 7739)	TXMLTransformClient acts as an adapter between an XML document and a provider component.
	TXMLTransformProvider (see page 7747)	TXMLTransformProvider provides data from and resolves updates to an XML document.

82.1.1 TranslateException Class

TranslateException is the exception class for errors that occur when converting an XML document to a data packet or a data packet to an XML document.

Class Hierarchy

Pascal

```
TranslateException = class(Exception);
```

C++

```
class TranslateException : public Exception;
```

File

Xmlxform

Description

TranslateException is raised by the transform components (TXMLTransform (see page 7727), TXMLTransformClient (see page 7739), and TXMLTransformProvider (see page 7747)) when they can't find a file that is needed to perform a transformation. The missing file can be the XML document that is to be translated into a data packet, or a transformation file that contains instructions on how to map between a data packet and an XML document.

See Also

TXMLTransform (see page 7727)

TXMLTransformClient (see page 7739)

TXMLTransformProvider (see page 7747)

82.1.2 TXMLTransform Class

TXMLTransform converts an XML document into a data packet or an XML data packet into an XML document with another structure.

Class Hierarchy

Pascal

```
TXMLTransform = class(TComponent);
```

C++

```
class TXMLTransform : public TComponent;
```

File

Xmlxform

Description

Use TXMLTransform to convert an XML document into a data packet or an XML data packet into an XML document that is not a data packet. You can use a TXMLTransform component by itself, or specify the transformation to be performed on a TXMLTransform component that is part of another component such as TXMLTransformClient (see page 7739) or TXMLTransformProvider (see page 7747).

TXMLTransform uses a special XML document called a transformation, which indicates how to perform the conversion. This transformation is specified by the TransformationFile (see page 7736) or TransformationDocument (see page 7735) property. You can generate a transformation file using the xmlmapper.exe utility.

There are three ways to specify the source document or data packet that TXMLTransform converts:

Use the SourceXmlFile (see page 7735) property if the source document is a file on disk.

Use the SourceXml (see page 7734) property if the source document is an in-memory string of XML.

Use the SourceXmlDocument (see page 7734) property if the source document is available through a DOM interface (IDOMDocument (see page 7530)).

TXMLTransform performs its transformation when you read the Data (see page 7731) property or call the TransformXML (see page 7730) method. Once the source XML has been transformed, you can access the DOM interface for the resulting XML document or data packet using the ResultDocument (see page 7733) property, or read the Data (see page 7731) property to directly access the string of XML.

See Also

TXMLTransformClient (see page 7739)

TXMLTransformProvider (see page 7747)

Converting XML Documents into Data Packets

82.1.2.1 TXMLTransform Members

The following tables list the members exposed by TXMLTransform.

Protected Methods

	Name	Description
	DoTransform (see page 7729)	This is DoTransform, a member of class TXMLTransform.
	GetData (see page 7729)	This is GetData, a member of class TXMLTransform.
	GetResultString (see page 7729)	This is GetResultString, a member of class TXMLTransform.
	Transform (see page 7729)	This is Transform, a member of class TXMLTransform.
	Translate (see page 7730)	This is Translate, a member of class TXMLTransform.

Public Methods

	Name	Description
	TransformXML (see page 7730)	Converts an XML document using the specifications in a transformation file.

Public Properties

	Name	Description
R	Data (see page 7731)	Represents the translated string of XML.
	EmptyDestinationDocument (see page 7732)	Specifies the DOM interface for an XML document that defines the structure of the transformation result.
R	ResultDocument (see page 7733)	Specifies the DOM interface for the results of the transformation.
R	ResultString (see page 7733)	Specifies the results of the transformation.
	SourceXml (see page 7734)	Specifies a string of XML to be translated.
	SourceXmlDocument (see page 7734)	Specifies the DOM interface for the XML document to be converted.
	SourceXmlFile (see page 7735)	Specifies an XML document or XML data packet file to be translated.
	TransformationDocument (see page 7735)	Specifies the transformation that indicates how to convert the source XML document.

Published Events

	Name	Description
	AfterEachRow (see page 7737)	Occurs after TXMLTransform (see page 7727) has finished transforming a record in the XML data packet.
	AfterEachRowSet (see page 7737)	Occurs after TXMLTransform (see page 7727) has finished transforming the entire XML data packet or a nested detail set.
	BeforeEachRow (see page 7738)	Occurs immediately before TXMLTransform (see page 7727) starts transforming a record in the XML data packet.
	BeforeEachRowSet (see page 7738)	Occurs immediately before TXMLTransform (see page 7727) starts transforming the entire XML data packet or a nested detail set.
	OnTranslate (see page 7739)	Occurs when the TXMLTransform (see page 7727) component needs to translate a user-defined node.

Published Properties

	Name	Description
	TransformationFile (see page 7736)	Specifies the transformation file that indicates how to convert the source XML document.

82.1.2.2 TXMLTransform Methods

The methods of the TXMLTransform class are listed here.

Protected Methods

	Name	Description
	DoTransform (see page 7729)	This is DoTransform, a member of class TXMLTransform.
	GetData (see page 7729)	This is GetData, a member of class TXMLTransform.
	GetResultString (see page 7729)	This is GetResultString, a member of class TXMLTransform.
	Transform (see page 7729)	This is Transform, a member of class TXMLTransform.
	Translate (see page 7730)	This is Translate, a member of class TXMLTransform.

Public Methods

	Name	Description
	TransformXML (see page 7730)	Converts an XML document using the specifications in a transformation file.

82.1.2.2.1 TXMLTransform.DoTransform Method

Pascal

```
function DoTransform(const XMLSrc: IDOMDocument; const XMLExtract: IDOMDocument; const XMLOut:
IDOMDocument): string;
```

C++

```
__fastcall UnicodeString DoTransform(const IDOMDocument XMLSrc, const IDOMDocument XMLExtract,
const IDOMDocument XMLOut);
```

Description

This is DoTransform, a member of class TXMLTransform.

82.1.2.2.2 TXMLTransform.GetData Method

Pascal

```
function GetData: string;
```

C++

```
__fastcall UnicodeString GetData();
```

Description

This is GetData, a member of class TXMLTransform.

82.1.2.2.3 TXMLTransform.GetResultString Method

Pascal

```
function GetResultString: string;
```

C++

```
__fastcall UnicodeString GetResultString();
```

Description

This is GetResultString, a member of class TXMLTransform.

82.1.2.2.4 TXMLTransform.Transform Method

Pascal

```
procedure Transform(TransNode: IDOMNode; SrcNode: IDOMNode; DestNode: IDOMNode; Count:
Integer; InFromList: TStringList; InDestList: TStringList; InIdStrList: TStringList;
InValueList: TStringList; InOptionalList: TStringList; InDateFormatList: TStringList;
InDateFormatTypeList: TStringList; InMapValuesList: TStringList);
```

C++

```
__fastcall Transform(IDOMNode TransNode, IDOMNode SrcNode, IDOMNode DestNode, int Count,
TStringList InFromList, TStringList InDestList, TStringList InIdStrList, TStringList
InValueList, TStringList InOptionalList, TStringList InDateFormatList, TStringList
InDateFormatTypeList, TStringList InMapValuesList);
```

Description

This is Transform, a member of class TXMLTransform.

82.1.2.2.5 TXMLTransform.TransformXML Method

Converts an XML document using the specifications in a transformation file.

Pascal

```
function TransformXML(const SourceXml: string; const ATransformationFile: string = ''): string;
```

C++

```
__fastcall UnicodeString TransformXML(const UnicodeString SourceXml, const UnicodeString
ATransformationFile = '');
```

Description

Call TransformXML to convert an XML document when the source XML is an in-memory string.

SourceXml (see page 7734) is a string of XML to be transformed. It can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation document used to perform the transformation. If the transformation converts a specific type of XML document into a data packet, SourceXml (see page 7734) must conform to the expected schema. If the transformation converts a data packet into an XML document, SourceXml (see page 7734) must be an XML data packet with the expected metadata.

ATransformationFile is the transformation file to use when performing the translation. A transformation file is a special type of XML file with the .xtr extension. Each transformation file is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformation files.

If the ATransformationFile parameter is omitted, TransformXML uses the file specified by the TransformationFile (see page 7736) property, or the document specified by the TransformationDocument (see page 7735) property.

TransformXML returns the results of the conversion as a string of XML.

See Also

Data (see page 7731)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

82.1.2.2.6 TXMLTransform.Translate Method

Pascal

```
procedure Translate(const Id: string; const SrcNode: IDOMNode; var SrcValue: string; const
DestNode: IDOMNode); dynamic;
```

C++

```
__fastcall Translate(const UnicodeString Id, const IDOMNode SrcNode, UnicodeString SrcValue,
const IDOMNode DestNode);
```

Description

This is Translate, a member of class TXMLTransform.

82.1.2.3 TXMLTransform Properties

The properties of the TXMLTransform class are listed here.

Public Properties

	Name	Description
	Data (see page 7731)	Represents the translated string of XML.
	EmptyDestinationDocument (see page 7732)	Specifies the DOM interface for an XML document that defines the structure of the transformation result.
	ResultDocument (see page 7733)	Specifies the DOM interface for the results of the transformation.
	ResultString (see page 7733)	Specifies the results of the transformation.
	SourceXml (see page 7734)	Specifies a string of XML to be translated.
	SourceXmlDocument (see page 7734)	Specifies the DOM interface for the XML document to be converted.
	SourceXmlFile (see page 7735)	Specifies an XML document or XML data packet file to be translated.
	TransformationDocument (see page 7735)	Specifies the transformation that indicates how to convert the source XML document.

Published Properties

	Name	Description
	TransformationFile (see page 7736)	Specifies the transformation file that indicates how to convert the source XML document.

82.1.2.3.1 TXMLTransform.Data Property

Represents the translated string of XML.

Pascal

```
property Data: string;
```

C++

```
__property UnicodeString Data;
```

Description

Read (see page 3634) Data to obtain the results of the transformation. Reading the Data property causes TXMLTransform (see page 7727) to translate its source XML document or data packet into the destination format, which becomes the value of the Data property. The resulting XML string is not cached in memory: reading Data a second time causes TXMLTransform (see page 7727) to perform its transformation a second time.

If the source XML is an XML document and the transformation indicates how to convert it into a data packet, Data is the contents of a data packet in XML format.

If the source XML is an XML data packet and the transformation indicates how to convert it into another type of XML document, Data is the contents of the resulting document.

There are three ways to specify the source document or data packet that is converted to produce Data:

- Use the SourceXmlFile (see page 7735) property if the source document is a file on disk.
- Use the SourceXml (see page 7734) property if the source document is an in-memory string of XML.
- Use the SourceXmlDocument (see page 7734) property if the source document is available through a DOM interface (IDOMDocument (see page 7530)).
- If SourceXmlFile (see page 7735) is set to an invalid file name or SourceXmlFile (see page 7735) is an empty string and SourceXml (see page 7734) is not a valid string of XML, or SourceXmlFile (see page 7735) and SourceXml (see page 7734) are both empty strings and SourceXmlDocument (see page 7734) is not set, TXMLTransform (see page 7727) raises a TranslateException (see page 7726) when you try to read Data.

- There are two ways to specify the transformation that converts the source XML into Data:
- Use the TransformationFile (see page 7736) property if the transformation is stored in a file on disk.
- Use the TransformationDocument (see page 7735) method if the transformation is available through a DOM interface (IDOMDocument (see page 7530)).
- If TransformationFile (see page 7736) is set to an invalid file name or TransformationFile (see page 7736) is an empty string and TransformationDocument (see page 7735) is not set, TXMLTransform (see page 7727) raises a TranslateException (see page 7726) when you try to read Data.
- You can augment the transformation by using the EmptyDestinationDocument (see page 7732) property to supply a skeleton document that gives the form for the resulting XML document.
- When you read Data, for every user-defined node in the source XML document, TXMLTransform (see page 7727) generates an OnTranslate (see page 7739) event to convert the node.

Tip: When converting an XML document into a data packet, the value of Data can be assigned to a client dataset's XMLData property.

See Also

SourceXmlFile (see page 7735)

TransformationFile (see page 7736)

TranslateException (see page 7726)

OnTranslate (see page 7739)

XMLData

ResultDocument (see page 7733)

EmptyDestinationDocument (see page 7732)

TransformationDocument (see page 7735)

SourceXml (see page 7734)

Converting XML Documents into Data Packets

82.1.2.3.2 TXMLTransform.EmptyDestinationDocument Property

Specifies the DOM interface for an XML document that defines the structure of the transformation result.

Pascal

```
property EmptyDestinationDocument: IDOMDocument;
```

C++

```
__property IDOMDocument EmptyDestinationDocument;
```

Description

The transformation defined by TransformationFile (see page 7736) or TransformationDocument (see page 7735) contains a default structure for the results of transforming the source document in addition to the mappings between nodes in the source document and nodes in the destination. You can override this default structure, however, by setting EmptyDestinationDocument. This is the DOM interface for an XML document that has the desired structure, but with no values in the nodes that should be filled in using values in the source document. This is useful, for example, when you want to add additional information that does not come from the source document.

EmptyDestinationDocument is not changed when TXMLTransform (see page 7727) performs its transformation. To obtain a DOM interface for the resulting document, use the ResultDocument (see page 7733) property.

See Also

ResultDocument (see page 7733)

Data (see page 7731)

TransformXML (see page 7730)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

82.1.2.3.3 TXMLTransform.ResultDocument Property

Specifies the DOM interface for the results of the transformation.

Pascal

```
property ResultDocument: IDOMDocument;
```

C++

```
__property IDOMDocument ResultDocument;
```

Description

Read (see page 3634) ResultDocument to work with the results of the transformation using a DOM interface. ResultDocument is created when you call the TransformXML (see page 7730) method to perform the transformation.

See Also

SourceXmlDocument (see page 7734)

Data (see page 7731)

TransformXML (see page 7730)

EmptyDestinationDocument (see page 7732)

ResultString (see page 7733)

82.1.2.3.4 TXMLTransform.ResultString Property

Specifies the results of the transformation.

Pascal

```
property ResultString: string;
```

C++

```
__property UnicodeString ResultString;
```

Description

Read (see page 3634) ResultString to obtain the results of the transformation as a string of XML. ResultString is created when you call the TransformXML (see page 7730) method to perform the transformation.

See Also

SourceXmlDocument (see page 7734)

Data (see page 7731)

TransformXML (see page 7730)

EmptyDestinationDocument (see page 7732)

82.1.2.3.5 TXMLTransform.SourceXml Property

Specifies a string of XML to be translated.

Pascal

```
property SourceXml: string;
```

C++

```
__property UnicodeString SourceXml;
```

Description

Specify the XML to be translated as the value of SourceXml. SourceXml can be the text of any XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 7736) or TransformationDocument (see page 7735). If the transformation converts a specific type of XML document into a data packet, SourceXml must conform to the expected schema. If the transformation converts a data packet into an XML document, SourceXml is an XML data packet with the expected metadata.

SourceXmlDocument (see page 7734) is used only if the SourceXmlFile (see page 7735) property has not been set. If both SourceXmlFile (see page 7735) and SourceXml are empty strings, TXMLTransform (see page 7727) uses the SourceXmlDocument (see page 7734) property instead. SourceXml takes precedence over SourceXmlDocument (see page 7734): If SourceXmlDocument (see page 7734) is set, TXMLTransform (see page 7727) still uses SourceXml (even if its value is not a valid string of XML).

See Also

SourceXmlDocument (see page 7734)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

SourceXmlFile (see page 7735)

Converting XML Documents into Data Packets

82.1.2.3.6 TXMLTransform.SourceXmlDocument Property

Specifies the DOM interface for the XML document to be converted.

Pascal

```
property SourceXmlDocument: IDOMDocument;
```

C++

```
__property IDOMDocument SourceXmlDocument;
```

Description

Specify the document to be translated as the value of SourceXmlDocument. SourceXmlDocument can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 7736) or TransformationDocument (see page 7735). If the transformation converts a specific type of XML document into a data packet, then SourceXmlDocument is a document with the expected schema. If the transformation converts a data packet into an XML document, SourceXmlDocument is the interface to an XML data packet with the expected metadata.

SourceXmlDocument is used only if the SourceXmlFile (see page 7735) and SourceXml (see page 7734) properties have not been set.

See Also

SourceXmlFile (see page 7735)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

SourceXml (see page 7734)

Converting XML Documents into Data Packets

82.1.2.3.7 TXMLTransform.SourceXmlFile Property

Specifies an XML document or XML data packet file to be translated.

Pascal

```
property SourceXmlFile: string;
```

C++

```
__property UnicodeString SourceXmlFile;
```

Description

Specify the file to be translated as the value of SourceXmlFile. SourceXmlFile can be any type of XML document, including an XML data packet. It must, however, conform to the schema expected by the transformation specified by TransformationFile (see page 7736) or TransformationDocument (see page 7735). If the transformation converts a specific type of XML document into a data packet, SourceXmlFile is a document with the expected schema. If the transformation converts a data packet into an XML document, SourceXmlFile is an XML data packet with the expected metadata.

If SourceXmlFile is an empty string, TXMLTransform (see page 7727) uses the SourceXml (see page 7734) or SourceXmlDocument (see page 7734) property instead. SourceXmlFile takes precedence over both of these other properties. If SourceXml (see page 7734) or SourceXmlDocument (see page 7734) is set, TXMLTransform (see page 7727) still uses SourceXmlFile (even if it does not specify a valid file name).

See Also

SourceXmlDocument (see page 7734)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

SourceXml (see page 7734)

Converting XML Documents into Data Packets

82.1.2.3.8 TXMLTransform.TransformationDocument Property

Specifies the transformation that indicates how to convert the source XML document.

Pascal

```
property TransformationDocument: IDOMDocument;
```

C++

```
__property IDOMDocument TransformationDocument;
```

Description

TransformationDocument is the DOM interface for the transformation that contains instructions for converting the source XML

document into the XML that is the value of Data (see page 7731) and ResultDocument (see page 7733).

A transformation is a special type of XML document that defines the mapping between the nodes of a source document and a destination document and the structure of the destination document. Each transformation is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformations.

If the EmptyDestinationDocument (see page 7732) property is set, TXMLTransform (see page 7727) uses that to override the default structure for the destination document that is defined in TransformationDocument.

TransformationDocument is used only if the value of TransformationFile (see page 7736) is an empty string.

See Also

TransformXML (see page 7730)

TransformationFile (see page 7736)

Data (see page 7731)

EmptyDestinationDocument (see page 7732)

ResultDocument (see page 7733)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

82.1.2.3.9 TXMLTransform.TransformationFile Property

Specifies the transformation file that indicates how to convert the source XML document.

Pascal

```
property TransformationFile: string;
```

C++

```
__property UnicodeString TransformationFile;
```

Description

TransformationFile is a file that contains instructions for converting the source XML document into the XML that is the value of Data (see page 7731) and ResultDocument (see page 7733). If the source document is a general XML document, TransformationFile represents a transformation from that document's schema into a data packet. If the source document is an XML data packet, TransformationFile represents a transformation from that data packet into another type of XML document.

A transformation file is a special type of XML file with the .xtr extension. Each transformation file is specific to a particular XML schema and data packet format and represents a one-way transformation. Use the xmlmapper utility to create transformation files.

If the EmptyDestinationDocument (see page 7732) property is set, TXMLTransform (see page 7727) uses that to override the default structure for the destination document that is defined in the transformation file.

TransformationFile takes precedence over the TransformationDocument (see page 7735) property. That is, TXMLTransform (see page 7727) only uses TransformationDocument (see page 7735) if TransformationFile is an empty string.

See Also

TransformXML (see page 7730)

Data (see page 7731)

TransformationDocument (see page 7735)

ResultDocument (see page 7733)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

82.1.2.4 TXMLTransform Events

The events of the TXMLTransform class are listed here.

Published Events

	Name	Description
	AfterEachRow (see page 7737)	Occurs after TXMLTransform (see page 7727) has finished transforming a record in the XML data packet.
	AfterEachRowSet (see page 7737)	Occurs after TXMLTransform (see page 7727) has finished transforming the entire XML data packet or a nested detail set.
	BeforeEachRow (see page 7738)	Occurs immediately before TXMLTransform (see page 7727) starts transforming a record in the XML data packet.
	BeforeEachRowSet (see page 7738)	Occurs immediately before TXMLTransform (see page 7727) starts transforming the entire XML data packet or a nested detail set.
	OnTranslate (see page 7739)	Occurs when the TXMLTransform (see page 7727) component needs to translate a user-defined node.

82.1.2.4.1 TXMLTransform.AfterEachRow Event

Occurs after TXMLTransform (see page 7727) has finished transforming a record in the XML data packet.

Pascal

```
property AfterEachRow: TRowEvent;
```

C++

```
__property TRowEvent AfterEachRow;
```

Description

Write (see page 3707) an AfterEachRow event handler to take specific action after the XML transform component has translated each row in the data packet. For example, you can use AfterEachRow to read the data from the generated row and use it to compute the value of an internally calculated field.

See Also

TransformXML (see page 7730)

AfterEachRowSet (see page 7737)

BeforeEachRow (see page 7738)

Data (see page 7731)

82.1.2.4.2 TXMLTransform.AfterEachRowSet Event

Occurs after TXMLTransform (see page 7727) has finished transforming the entire XML data packet or a nested detail set.

Pascal

```
property AfterEachRowSet: TRowEvent;
```

C++

```
__property TRowEvent AfterEachRowSet;
```

Description

Write (see page 3707) an AfterEachRowSet event handler to take specific action after the XML transform component has translated an entire dataset. This can be a nested detail set, or the entire XML data packet.

For example, you can use AfterEachRowSet with the BeforeEachRowSet (see page 7738) and OnTranslate (see page 7739) methods to compute summary values: initialize a variable in the BeforeEachRowSet (see page 7738) method, update it for every OnTranslate (see page 7739) event, and use the result in the AfterEachRowSet event.

See Also

TransformXML (see page 7730)

AfterEachRow (see page 7737)

BeforeEachRow (see page 7738)

Data (see page 7731)

82.1.2.4.3 TXMLTransform.BeforeEachRow Event

Occurs immediately before TXMLTransform (see page 7727) starts transforming a record in the XML data packet.

Pascal

```
property BeforeEachRow: TRowEvent;
```

C++

```
__property TRowEvent BeforeEachRow;
```

Description

Write (see page 3707) a BeforeEachRow event handler to take specific action before the XML transform component starts translating each row in the data packet.

See Also

TransformXML (see page 7730)

BeforeEachRowSet (see page 7738)

AfterEachRow (see page 7737)

Data (see page 7731)

82.1.2.4.4 TXMLTransform.BeforeEachRowSet Event

Occurs immediately before TXMLTransform (see page 7727) starts transforming the entire XML data packet or a nested detail set.

Pascal

```
property BeforeEachRowSet: TRowEvent;
```

C++

```
__property TRowEvent BeforeEachRowSet;
```

Description

Write (see page 3707) an BeforeEachRowSet event handler to take specific action before the XML transform component

starts translating a dataset. This can be a nested detail set, or the entire XML data packet.

For example, you can use BeforeEachRowSet with the AfterEachRowSet (see page 7737) and OnTranslate (see page 7739) methods to compute summary values: initialize a variable in the BeforeEachRowSet method, update it for every OnTranslate (see page 7739) event, and use the result in the AfterEachRowSet (see page 7737) event.

See Also

TransformXML (see page 7730)

AfterEachRowSet (see page 7737)

BeforeEachRow (see page 7738)

Data (see page 7731)

82.1.2.4.5 TXMLTransform.OnTranslate Event

Occurs when the TXMLTransform (see page 7727) component needs to translate a user-defined node.

Pascal

```
property OnTranslate: TTranslateEvent;
```

C++

```
__property TTranslateEvent OnTranslate;
```

Description

Write (see page 3707) an OnTranslate event handler to provide the translation for user-defined nodes.

TXMLTransform (see page 7727) performs a translation when an application reads the Data (see page 7731) property or calls the TransformXML (see page 7730) method. If the transformation document that governs how to perform that translation indicates that a node in the XML document is user defined, TXMLTransform (see page 7727) generates an OnTranslate event when it needs to translate to or from that node. OnTranslate only occurs for nodes that are marked as user defined.

If you do not translate a user-defined node in an OnTranslate event handler, the corresponding node in the translated XML is assigned the source XML node's value, if available. Otherwise, it is left blank.

See Also

TransformXML (see page 7730)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

Data (see page 7731)

Converting XML Documents into Data Packets

Using an XML Document as the Client of a Provider

Using an XML Document as the Source for a Provider

82.1.3 TXMLTransformClient Class

TXMLTransformClient acts as an adapter between an XML document and a provider component.

Class Hierarchy

Pascal

```
TXMLTransformClient = class(TComponent);
```

C++

```
class TXMLTransformClient : public TComponent;
```

File

Xmlxform

Description

Use TXMLTransformClient to convert the data from a provider into an XML document or to apply updates in the form of an XML document to a provider component. TXMLTransformClient can act as the client of any provider component, local or remote. It uses a transformation file that is generated using the xmlmapper utility to convert between XML documents and the data packets and delta packets used by the provider.

See Also

TXMLTransform (see page 7727)

TXMLTransformProvider (see page 7747)

Using an XML Document as the Client of a Provider

82.1.3.1 TXMLTransformClient Members

The following tables list the members exposed by TXMLTransformClient.

Protected Methods

	Name	Description
	SetupAppServer (see page 7744)	This is SetupAppServer, a member of class TXMLTransformClient.

Public Methods

	Name	Description
V	ApplyUpdates (see page 7742)	Inserts or deletes the records corresponding to the data in an XML document.
V	Create (see page 7741)	Creates and initializes a TXMLTransformClient (see page 7739) object.
V	Destroy (see page 7741)	Destroys the instance of a TXMLTransformClient (see page 7739) component.
V	GetDataAsXml (see page 7742)	Fetches a data packet from the provider and converts it to an XML document.
	SetParams (see page 7743)	Sets input parameters to values supplied in an XML document.

Published Properties

	Name	Description
	ProviderName (see page 7745)	Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.
	RemoteServer (see page 7745)	Specifies the connection component that this XML transform client uses to connect to an application server.
R	TransformApplyUpdates (see page 7746)	Specifies the TXMLTransform (see page 7727) component that converts XML into delta packets that are then sent to the provider to apply updates.
R	TransformGetData (see page 7746)	Specifies the TXMLTransform (see page 7727) component that converts data packets from the provider into XML.
R	TransformSetParams (see page 7747)	Specifies the TXMLTransform (see page 7727) component that converts the XML supplied to the SetParams (see page 7743) method into a data packet format.

82.1.3.2 TXMLTransformClient.Create Constructor

Creates and initializes a TXMLTransformClient (see page 7739) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLTransformClient(TComponent * AOwner);
```

Description

Call Create to instantiate an XML transform client at runtime. XML transform clients placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (typically a form or data module) that is responsible for freeing the TXMLTransformClient (see page 7739) object.

See Also

Owner

82.1.3.3 TXMLTransformClient.Destroy Destructor

Destroys the instance of a TXMLTransformClient (see page 7739) component.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLTransformClient();
```

Description

Do not call Destroy directly in an application. Usually destruction of TXMLTransformClient (see page 7739) objects is handled automatically. If an application creates its own instances of TXMLTransformClient (see page 7739) without assigning an Owner, however, it should call Free, which verifies that the dataset component is not nil before calling Destroy.

See Also

Owner

Create (see page 7741)

System::TObject::Free (see page 3372)

82.1.3.4 TXMLTransformClient Methods

The methods of the TXMLTransformClient class are listed here.

Protected Methods

	Name	Description
	SetupAppServer (see page 7744)	This is SetupAppServer, a member of class TXMLTransformClient.

Public Methods

	Name	Description
	ApplyUpdates (see page 7742)	Inserts or deletes the records corresponding to the data in an XML document.
	GetDataAsXml (see page 7742)	Fetches a data packet from the provider and converts it to an XML document.
	SetParams (see page 7743)	Sets input parameters to values supplied in an XML document.

82.1.3.4.1 TXMLTransformClient.ApplyUpdates Method

Inserts or deletes the records corresponding to the data in an XML document.

Pascal

```
function ApplyUpdates(const UpdateXML: string; const UpdateTransformFile: string; MaxErrors:
Integer): Integer; virtual;
```

C++

```
virtual __fastcall int ApplyUpdates(const UnicodeString UpdateXML, const UnicodeString
UpdateTransformFile, int MaxErrors);
```

Description

Call ApplyUpdates to transform an XML document into a delta packet that inserts or deletes all of the resulting records and send it to the associated provider so that it can be applied. Before you can call ApplyUpdates, you must define the transformation to use by setting properties on the XML transform component specified by TransformApplyUpdates (see page 7746).

UpdateXML is the XML document that can be transformed into a delta packet. Typically, it contains the contents of an XML document file.

UpdateTransformFile is the name of a transformation file that converts UpdateXML into an insert or delete delta packet. A transformation file is a special type of XML file with the .xtr extension. The transformation file is specific to a particular XML schema, data packet format, and update operation (insert or delete). Use the xmlmapper utility to create the transformation file.

Note: If UpdateTransformFile includes any user-defined conversions, ApplyUpdates generates an OnTranslate event on TransformApplyUpdates (see page 7746) when it needs to transform user-defined nodes in the UpdateXML string into corresponding field values.

MaxErrors indicates the maximum number of errors that the provider should allow before prematurely stopping the update operation. Set MaxErrors to –1 to indicate that there is no limit to the number of errors.

ApplyUpdates returns the number of errors it encountered.

See Also

ProviderName (see page 7745)

TransformApplyUpdates (see page 7746)

Using an XML Document as the Client of a Provider

82.1.3.4.2 TXMLTransformClient.GetDataAsXml Method

Fetches a data packet from the provider and converts it to an XML document.

Pascal

```
function GetDataAsXml(const PublishTransformFile: string): string; virtual;
```

C++

```
virtual __fastcall UnicodeString GetDataAsXml(const UnicodeString PublishTransformFile);
```

Description

Call GetDataAsXml to obtain an XML document that is the translation of a data packet from the associated provider. GetDataAsXml

1. Passes any parameters supplied to the SetParams (see page 7743) method to the provider specified by ProviderName (see page 7745).
2. Obtains a data packet that the provider returns.
3. Uses the transform component specified by TransformGetData (see page 7746) to convert the data packet to an XML document. If the PublishTransformFile parameter is not an empty string, GetDataAsXml uses the file specified by this parameter as the transformation file. Otherwise, it uses the transformation file specified on the TransformGetData (see page 7746) property.
4. Generates an OnTranslate event for every user-defined conversion in its transformation. The application can use this event to perform the user-defined conversion that converts field values into XML nodes.
5. Returns the resulting XML document as a string.

Once you have obtained the resulting, translated XML document, you can assign it to a TXMLDocument component or save it to a file.

Note: Before you can call GetDataAsXml, you must supply any input parameters using the SetParams (see page 7743) method and set properties on TransformGetData (see page 7746) to indicate what transformation to perform.

See Also

ProviderName (see page 7745)

TransformGetData (see page 7746)

XML (see page 7392)

OnTranslate (see page 7739)

SetParams (see page 7743)

Using an XML Document as the Client of a Provider

82.1.3.4.3 TXMLTransformClient.SetParams Method

Sets input parameters to values supplied in an XML document.

Pascal

```
procedure SetParams(const ParamsXml: string; const ParamsTransformFile: string);
```

C++

```
__fastcall SetParams(const UnicodeString ParamsXml, const UnicodeString ParamsTransformFile);
```

Description

Call SetParams to supply input parameter values before fetching data from the provider. Parameter values can represent the input parameters to a query or stored procedure, or they can represent field values that filter the data when the provider represents the data in a TTable or TSQLTable component.

ParamsXML is the XML document that represents the parameter values to set. Typically, it contains the contents of an XML document file.

ParamsTransformFile is the name of a transformation file that converts ParamsXml into a data packet. A transformation file is a special type of XML file with the .xtr extension. The transformation file is specific to a particular XML schema and data packet format. Use the xmlmapper.exe utility to create the transformation file.

Note: If ParamsTransformFile includes any user-defined conversions, the component specified by TransformSetParams (see page 7747) generates an OnTranslate event when it needs to transform user-defined nodes in the ParamsXml string into corresponding parameter values.

The TXMLTransformClient (see page 7739) component stores parameter values internally. Each time you call SetParams, you update those internally stored values. Parameter values from previous calls are not cleared, so be sure to set every parameter value you want to change.

The next the GetDataAsXml (see page 7742) method is called, TXMLTransformClient (see page 7739) sends the currently stored parameter values to the provider.

See Also

GetDataAsXml (see page 7742)

OnTranslate (see page 7739)

ProviderName (see page 7745)

TTable

TSQLTable (see page 1680)

TransformSetParams (see page 7747)

Using an XML Document as the Client of a Provider

82.1.3.4.4 TXMLTransformClient.SetupAppServer Method

Pascal

```
procedure SetupAppServer;
```

C++

```
__fastcall SetupAppServer();
```

Description

This is SetupAppServer, a member of class TXMLTransformClient.

82.1.3.5 TXMLTransformClient Properties

The properties of the TXMLTransformClient class are listed here.

Published Properties

	Name	Description
	ProviderName (see page 7745)	Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.
	RemoteServer (see page 7745)	Specifies the connection component that this XML transform client uses to connect to an application server.
R	TransformApplyUpdates (see page 7746)	Specifies the TXMLTransform (see page 7727) component that converts XML into delta packets that are then sent to the provider to apply updates.
R	TransformGetData (see page 7746)	Specifies the TXMLTransform (see page 7727) component that converts data packets from the provider into XML.
R	TransformSetParams (see page 7747)	Specifies the TXMLTransform (see page 7727) component that converts the XML supplied to the SetParams (see page 7743) method into a data packet format.

82.1.3.5.1 TXMLTransformClient.ProviderName Property

Specifies the name of a provider component that supplies data to and applies updates from the XML transform client.

Pascal

```
property ProviderName: string;
```

C++

```
__property UnicodeString ProviderName;
```

Description

Use ProviderName to specify the provider object that supplies the XML transform client with data packets and applies updates on its behalf. This property is passed as a parameter to IAppServer interface calls when the client fetches data from or applies updates to the source dataset, its database server, or the source XML document.

The provider specified by ProviderName can reside in the same application as the XML transform client or in an application server running on another system.

If the provider is in the same application as the XML transform client, it must have the same Owner as the client.

If the provider is on a remote application server, it must be exported by that application server. If the application server contains multiple provider objects, you can set ProviderName to any of them once the XML transform client establishes an initial connection to the application server through its RemoteServer (see page 7745) property.

ProviderName can be set from a drop-down in the Object Inspector once the RemoteServer (see page 7745) property is set if

The connection component is a SOAP connection.

The connection component is a Web connection or socket connection (Windows only).

The application server is COM-based and registered on the client system using TRegsvr.exe (Windows only).

See Also

RemoteServer (see page 7745)

GetDataAsXml (see page 7742)

IAppServer

ApplyUpdates (see page 7742)

Using an XML Document as the Client of a Provider

82.1.3.5.2 TXMLTransformClient.RemoteServer Property

Specifies the connection component that this XML transform client uses to connect to an application server.

Pascal

```
property RemoteServer: TCustomRemoteServer;
```

C++

```
__property TCustomRemoteServer RemoteServer;
```

Description

If the XML transform client is in the client portion of a multi-tiered application, use RemoteServer to specify the connection component that connects to the application server. The connection component locates the application server and obtains its IAppServer interface, which the XML transform client can use for communicating with a remote provider.

If a client application uses more than connection component, the application can change which connection an XML transform client uses by changing the value for this property.

See Also

ProviderName (see page 7745)

TSocketConnection (see page 221)

TDCOMConnection

TWebConnection (see page 254)

TSoapConnection (see page 1188)

AppServer

IAppServer

Using an XML Document as the Client of a Provider

82.1.3.5.3 TXMLTransformClient.TransformApplyUpdates Property

Specifies the TXMLTransform (see page 7727) component that converts XML into delta packets that are then sent to the provider to apply updates.

Pascal

```
property TransformApplyUpdates: TXMLTransform;
```

C++

```
__property TXMLTransform TransformApplyUpdates;
```

Description

Use TransformApplyUpdates to access the subcomponent that transforms the XML passed as the UpdateXML parameter of the ApplyUpdates (see page 7742) method into a delta packet, which is then forwarded to the provider. If the transformation passed to the ApplyUpdates (see page 7742) method includes any user-defined nodes, you must supply TransformApplyUpdates with an OnTranslate event handler.

See Also

ApplyUpdates (see page 7742)

ProviderName (see page 7745)

OnTranslate (see page 7739)

82.1.3.5.4 TXMLTransformClient.TransformGetData Property

Specifies the TXMLTransform (see page 7727) component that converts data packets from the provider into XML.

Pascal

```
property TransformGetData: TXMLTransform;
```

C++

```
__property TXMLTransform TransformGetData;
```

Description

Use TransformGetData to access the subcomponent that performs the transformation of data packets from the provider into another XML format. When an application calls the GetDataAsXml (see page 7742) method, this subcomponent converts the

data packet from the provider into XML.

Before you can call GetDataAsXml (see page 7742), you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformGetData with an OnTranslate event handler.

See Also

GetDataAsXml (see page 7742)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

OnTranslate (see page 7739)

Using an XML Document as the Client of a Provider

82.1.3.5.5 TXMLTransformClient.TransformSetParams Property

Specifies the TXMLTransform (see page 7727) component that converts the XML supplied to the SetParams (see page 7743) method into a data packet format.

Pascal

```
property TransformSetParams: TXMLTransform;
```

C++

```
__property TXMLTransform TransformSetParams;
```

Description

Use TransformSetParams to directly access the subcomponent that performs the transformation of an XML document into a set of parameter values. When an application calls the SetParams (see page 7743) method, this subcomponent converts the XML supplied as the ParamsXml parameter into a data packet, which it then uses to read and store parameter values. These parameter values are later passed to the provider when you call GetDataAsXml (see page 7742).

If the transformation passed to the SetParams (see page 7743) method includes any user-defined nodes, you must supply TransformSetParams with an OnTranslate event handler.

See Also

SetParams (see page 7743)

ProviderName (see page 7745)

OnTranslate (see page 7739)

GetDataAsXml (see page 7742)

Using an XML Document as the Client of a Provider

82.1.4 TXMLTransformProvider Class

TXMLTransformProvider provides data from and resolves updates to an XML document.

Class Hierarchy

Pascal

```
TXMLTransformProvider = class(TCustomProvider);
```

C++

```
class TXMLTransformProvider : public TCustomProvider;
```

File

Xmlxform

Description

Use TXMLTransformProvider to provide data from an XML document to a client dataset or XML broker and to resolve updates from that client dataset or XML broker back to the source XML document. TXMLTransformProvider can be a part of the same application as the client dataset or XML broker, or it can be placed in the application server of a multi-tiered application. It serves as a translator between an XML document and a client dataset.

TDataSetProvider packages data from the source XML document and passes it in one or more transportable data packets to the client dataset or XML broker.

When the provider supplies data to a client dataset, the client dataset reconstructs the data in the data packet to create a local, in-memory copy for user access. When user access is complete, the client dataset repackages any changed data and sends the updates back to the provider. The provider applies the updates to the source XML document.

When the provider supplies data to an XML broker, the XML broker adds the data packet, in XML format, to an HTML document that is downloaded by a Web client. When the XML broker receives updates from the Web client, it sends the updates to the provider, which applies them to the source XML document.

Client datasets and XML brokers communicate with a provider through the IAppServer interface. In multi-tiered applications, this is the interface of the remote data module that contains the provider. To enable the remote data module to pass calls to the provider, the provider's Exported property must be true, and its Owner property must specify the remote data module.

See Also

TClientDataSet

TDataSetProvider

TXMLTransform (see page 7727)

TRemoteDataModule

TCRemoteDataModule

TSoapDataModule (see page 1206)

TXMLTransformClient (see page 7739)

TCustomProvider

Determining the Source of Data

Using Provider Components

Using an XML Document as the Source for a Provider

82.1.4.1 TXMLTransformProvider Members

The following tables list the members exposed by TXMLTransformProvider.

Protected Methods

	Name	Description
	InternalApplyUpdates (see page 7750)	Provides the underlying implementation of the ApplyUpdates method.
	InternalGetRecords (see page 7750)	Provides the underlying implementation of the GetRecords method.

Public Methods

	Name	Description
	Create (see page 7749)	Creates and initializes a TXMLTransformProvider (see page 7747) object.
	Destroy (see page 7749)	Frees an instance of a provider object.

Published Events

	Name	Description
	AfterApplyUpdates (see page 7754)	Occurs after the provider applies updates.
	AfterGetRecords (see page 7755)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 7756)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 7756)	Occurs before the provider applies updates from a client dataset.
	BeforeGetRecords (see page 7757)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 7758)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 7759)	Occurs when a provider calls its DataRequest method.

Published Properties

	Name	Description
	CacheData (see page 7751)	Specifies whether the provider caches the data packet internally in memory.
R	TransformRead (see page 7752)	Specifies the TXMLTransform (see page 7727) component that converts the source XML document into data packets.
R	TransformWrite (see page 7753)	Specifies the TXMLTransform (see page 7727) component that generates the new value of the source XML document when updates are applied.
	XMLDataFile (see page 7753)	Specifies the XML document from which the provider creates data packets and to which it applies updates.

82.1.4.2 TXMLTransformProvider.Create Constructor

Creates and initializes a TXMLTransformProvider (see page 7747) object.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXMLTransformProvider(TComponent * AOwner);
```

Description

Call Create to instantiate an XML transform provider at runtime. Providers placed in forms or data modules at design time are created automatically.

The AOwner parameter specifies another component (typically a form or data module) that is responsible for freeing the provider.

See Also

Owner

82.1.4.3 TXMLTransformProvider.Destroy Destructor

Frees an instance of a provider object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLTransformProvider();
```

Description

Do not call Destroy directly. Instead, use the delete keyword, which indirectly calls Destroy.

See Also

delete

82.1.4.4 TXMLTransformProvider Methods

The methods of the TXMLTransformProvider class are listed here.

Protected Methods

	Name	Description
	InternalApplyUpdates (see page 7750)	Provides the underlying implementation of the ApplyUpdates method.
	InternalGetRecords (see page 7750)	Provides the underlying implementation of the GetRecords method.

82.1.4.4.1 TXMLTransformProvider.InternalApplyUpdates Method

Provides the underlying implementation of the ApplyUpdates method.

Pascal

```
function InternalApplyUpdates(const Delta: OleVariant; MaxErrors: Integer; out ErrorCount:
Integer): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalApplyUpdates(const OleVariant Delta, int MaxErrors, int
ErrorCount);
```

Description

The ApplyUpdates method calls InternalApplyUpdates to apply the updates contained in a delta packet after generating the BeforeApplyUpdates event and before generating the AfterApplyUpdates event. The InternalApplyUpdates method of TCustomProvider is abstract (Delphi) or pure virtual (C++), meaning that it has no implementation. Descendant classes override this method to apply updates, resolve update errors, and return any updates that could not be applied.

Delta is the delta packet containing insertions, deletions, and modifications to be applied.

MaxErrors indicates the maximum number of errors that can be applied before stopping the update operation. If MaxErrors is –1, the provider tries to apply all updates.

ErrorCount returns the number of errors encountered during the update operation.

InternalApplyUpdates returns a data packet of updates that could not be applied to the database as an OleVariant.

See Also

ApplyUpdates

82.1.4.4.2 TXMLTransformProvider.InternalGetRecords Method

Provides the underlying implementation of the GetRecords method.

Pascal

```
function InternalGetRecords(Count: Integer; out RecsOut: Integer; Options: TGetRecordOptions;
const CommandText: WideString; var Params: OleVariant): OleVariant; override;
```

C++

```
virtual __fastcall OleVariant InternalGetRecords(int Count, int RecsOut, TGetRecordOptions
Options, const BSTR CommandText, OleVariant Params);
```

Description

The GetRecords method calls InternalGetRecords to assemble a data packet after calling DoBeforeGetRecords to apply the CommandText and Params and generate a BeforeGetRecords event. As implemented in TCustomProvider, InternalGetRecords always returns NULL. Descendant classes override this method to fetch the requested records and assemble them into a data packet.

Count indicates the number of records to retrieve. If Count is -1, all records are retrieved. If Count is 0, only metadata is retrieved. If Count is greater than 0, only the specified number of records are retrieved.

RecsOut returns the actual number of records retrieved.

Options indicates what information should be added to the data packet in addition to data. It can include grMetaData (include metadata), grReset (start providing records with the first one in the dataset), and grXML (encode the datapacket in XML rather than as an OleVariant) or grXMLUTF8 (same as grXML except extended characters are encoded using UTF8).

CommandText specifies an optional SQL statement that replaces the SQL of an associated query, or the name of a table or stored procedure that replaces the associated table or stored procedure. This parameter is ignored if the Options property does not include poAllowCommandText or if the provider does not have an Options property.

Params specifies any parameter values that should be applied to CommandText or to an associated query or stored procedure. If the provider is associated with a TTable object, Params specifies field values that filter the result set. On exit from GetRecords, Params returns any output parameters.

Records are returned as a data packet in a Variant.

See Also

GetRecords

DoBeforeGetRecords

82.1.4.5 TXMLTransformProvider Properties

The properties of the TXMLTransformProvider class are listed here.

Published Properties

	Name	Description
	CacheData (see page 7751)	Specifies whether the provider caches the data packet internally in memory.
R	TransformRead (see page 7752)	Specifies the TXMLTransform (see page 7727) component that converts the source XML document into data packets.
R	TransformWrite (see page 7753)	Specifies the TXMLTransform (see page 7727) component that generates the new value of the source XML document when updates are applied.
	XMLDataFile (see page 7753)	Specifies the XML document from which the provider creates data packets and to which it applies updates.

82.1.4.5.1 TXMLTransformProvider.CacheData Property

Specifies whether the provider caches the data packet internally in memory.

Pascal

```
property CacheData: Boolean;
```

C++

```
__property Boolean CacheData;
```

Description

Set CacheData to true to specify that the provider should cache its data packet in memory after the first time it is read from the source XML document. When CacheData is false, every time the Data property is read, the GetRecords method is called, or updates are applied, the provider must reread the source document specified by XMLDataFile (see page 7753).

Caching the data improves performance on subsequent data fetches or when applying updates, but consumes additional resources.

See Also

Data

TransformRead (see page 7752)

GetRecords

XMLDataFile (see page 7753)

82.1.4.5.2 TXMLTransformProvider.TransformRead Property

Specifies the TXMLTransform (see page 7727) component that converts the source XML document into data packets.

Pascal

```
property TransformRead: TXMLTransform;
```

C++

```
__property TXMLTransform TransformRead;
```

Description

Use TransformRead to access the subcomponent that transforms the source XML document into the data packet that is the value of the Data property. Before you can use TXMLTransformProvider (see page 7747), you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformRead with an OnTranslate event handler.

See Also

XMLDataFile (see page 7753)

TransformWrite (see page 7753)

Data

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

OnTranslate (see page 7739)

Determining the Source of Data

Using an XML Document as the Source for a Provider

82.1.4.5.3 TXMLTransformProvider.TransformWrite Property

Specifies the TXMLTransform (see page 7727) component that generates the new value of the source XML document when updates are applied.

Pascal

```
property TransformWrite: TXMLTransform;
```

C++

```
__property TXMLTransform TransformWrite;
```

Description

Use TransformWrite to access the subcomponent that transforms a data packet back into the format required by the source XML document. When a client calls the ApplyUpdates method, the delta packet is merged into the transformed data from the XML document, and then TransformWrite converts the resulting dataset into the format of the source XML document.

Before you can use TXMLTransformProvider (see page 7747) to apply updates, you must specify the transformation that this subcomponent uses by setting its TransformationFile or TransformationDocument property. In addition, if the transformation includes any user-defined nodes, you must supply TransformWrite with an OnTranslate event handler.

See Also

ApplyUpdates

TransformRead (see page 7752)

TransformationFile (see page 7736)

TransformationDocument (see page 7735)

OnTranslate (see page 7739)

Using an XML Document as the Source for a Provider

82.1.4.5.4 TXMLTransformProvider.XMLDataFile Property

Specifies the XML document from which the provider creates data packets and to which it applies updates.

Pascal

```
property XMLDataFile: string;
```

C++

```
__property UnicodeString XMLDataFile;
```

Description

Use XMLDataFile to specify the XML document whose data the provider represents. When the Data property is read or the GetRecords method is called, TXMLTransformProvider (see page 7747) reads the contents of XMLDataFile and converts it into a data packet using the transform component specified by TransformRead (see page 7752). When the ApplyUpdates method is called, TXMLTransformProvider (see page 7747) uses the transform component specified by TransformWrite (see page 7753) to convert the data into the proper format for XMLDataFile and updates this XML document accordingly.

XMLDataFile is the same as the SourceXmlFile property of TransformRead (see page 7752).

See Also

Data

GetRecords

TransformRead (see page 7752)

TransformWrite (see page 7753)

ApplyUpdates

SourceXmlFile (see page 7735)

Determining the Source of Data

Using an XML Document as the Source for a Provider

82.1.4.6 TXMLTransformProvider Events

The events of the TXMLTransformProvider class are listed here.

Published Events

	Name	Description
	AfterApplyUpdates (see page 7754)	Occurs after the provider applies updates.
	AfterGetRecords (see page 7755)	Occurs after the provider creates a data packet to send to the client dataset.
	AfterRowRequest (see page 7756)	Occurs after the provider fetches new information about the current record.
	BeforeApplyUpdates (see page 7756)	Occurs before the provider applies updates from a client dataset.
	BeforeGetRecords (see page 7757)	Occurs before the provider creates a data packet and sends it to a client dataset.
	BeforeRowRequest (see page 7758)	Occurs before the provider creates a delta packet with information about the current record.
	OnDataRequest (see page 7759)	Occurs when a provider calls its DataRequest method.

82.1.4.6.1 TXMLTransformProvider.AfterApplyUpdates Event

Occurs after the provider applies updates.

Pascal

```
property AfterApplyUpdates;
```

C++

```
__property AfterApplyUpdates;
```

Description

Write an AfterApplyUpdates event handler to return custom information to a client dataset after applying updates. AfterApplyUpdates is part of the mechanism by which a client dataset and a provider communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_ApplyUpdates method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeApplyUpdates event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnUpdateData event where it can edit the updates before they are applied.

3. The provider applies the updates, and attempts to resolve problems.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the

provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, on entry to the event handler the OwnerData parameter contains the value set in the BeforeApplyUpdates event handler and it is ignored on exit from the AfterApplyUpdates event handler.

See Also

ApplyUpdates

AS_ApplyUpdates

ApplyUpdates

AfterApplyUpdates

BeforeApplyUpdates

OnUpdateData

Responding to Client Update Requests

Communicating with the Client Dataset

82.1.4.6.2 TXMLTransformProvider.AfterGetRecords Event

Occurs after the provider creates a data packet to send to the client dataset.

Pascal

```
property AfterGetRecords;
```

C++

```
__property AfterGetRecords;
```

Description

Write an AfterGetRecords event handler to send custom information to a client dataset when providing a data packet. AfterGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_GetRecords method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeGetRecords event, where it can respond to or change custom information that the client dataset sends.

2.If it is a dataset provider, the provider receives an OnGetData event where it can edit the data before creating a data packet, and then generates the data packet.

3.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

4.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

See Also

AS_GetRecords

OnGetData

GetRecords

AfterGetRecords

BeforeGetRecords

Communicating with the Client Dataset

Responding to Client Data Requests

82.1.4.6.3 TXMLTransformProvider.AfterRowRequest Event

Occurs after the provider fetches new information about the current record.

Pascal

```
property AfterRowRequest;
```

C++

```
__property AfterRowRequest;
```

Description

Write an AfterRowRequest event handler to send custom information to a client dataset along with information about the current record. AfterRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset calls the AS_RowRequest method of the IAppServer interface, the following events occur:

1.The provider receives a BeforeRowRequest event, where it can respond to or change custom information that the client dataset sends.

2.The provider creates a delta packet that contains the requested information.

3.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

4.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

AfterRowRequest

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

82.1.4.6.4 TXMLTransformProvider.BeforeApplyUpdates Event

Occurs before the provider applies updates from a client dataset.

Pascal

```
property BeforeApplyUpdates;
```

C++

```
__property BeforeApplyUpdates;
```

Description

Write a BeforeApplyUpdates event handler to respond to custom information from a client dataset. BeforeApplyUpdates is part of the mechanism by which a provider and a client dataset communicate information about updates. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset's ApplyUpdates method is called, the following events occur:

1.The client dataset receives a BeforeApplyUpdates event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeApplyUpdates event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before applying the updates.

3.If it is a dataset provider, the provider applies the updates, and receives an OnUpdateError event for any problem records.

4.The provider receives an AfterApplyUpdates event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeApplyUpdates event handler.

5.The client dataset receives an AfterApplyUpdates event, where it can respond to the custom information returned by the provider in its AfterApplyUpdates event handler.

Note: If ApplyUpdates is called without the OwnerData parameter, the OwnerData parameter is nil (Delphi) or NULL (C++) on entry to the event handler and its value (on exit) is only passed to the AfterApplyUpdates event handler but not returned to the caller.

See Also

ApplyUpdates

AS_ApplyUpdates

ApplyUpdates

BeforeApplyUpdates

AfterApplyUpdates

Responding to Client Update Requests

Communicating with the Client Dataset

82.1.4.6.5 TXMLTransformProvider.BeforeGetRecords Event

Occurs before the provider creates a data packet and sends it to a client dataset.

Pascal

```
property BeforeGetRecords;
```

C++

```
__property BeforeGetRecords;
```

Description

Write a BeforeGetRecords event handler to respond to custom information from a client dataset. BeforeGetRecords is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches data from the provider, the following events occur:

1.The client dataset receives a BeforeGetRecords event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeGetRecords event, where the OleVariant from the client dataset appears as the OwnerData parameter. The provider can respond to or change that information before it creates a data packet.

3.The provider generates the data packet.

4.The provider receives an AfterGetRecords event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeGetRecords event handler.

5.The client dataset receives an AfterGetRecords event, where it can respond to the custom information returned by the provider in its AfterGetRecords event handler.

Delphi Examples:

```
{
This example shows the implementation of a provider BeforeGetRecords
event handler and a RemoteDataModule UpdateRegistry method.  The
complete project using this code example is in
Demos/DelphiWin32/VCLWin32/MIDAS/Pooler.
}
procedure TPooledRDM.DataSetProvider1BeforeGetRecords(Sender: TObject;
  var OwnerData: OleVariant);
begin
  try
    Query1.Close;
    Query1.SQL.Text := OwnerData[0];
    if not VarIsNull(OwnerData[1]) and not VarIsEmpty(OwnerData[1]) then
    begin
      Query1.Open;
      if not Query1.Locate(Query1.Fields[0].FieldName, OwnerData[1], []) then
        raise Exception.Create('Record not found');
      Query1.Next;
    end;
  finally
    OwnerData := NULL;
  end;
end;
```

See Also

AS_GetRecords

GetRecords

BeforeGetRecords

AfterGetRecords

Communicating with the Client Dataset

Responding to Client Data Requests

82.1.4.6.6 TXMLTransformProvider.BeforeRowRequest Event

Occurs before the provider creates a delta packet with information about the current record.

Pascal

```
property BeforeRowRequest;
```

C++

```
__property BeforeRowRequest;
```

Description

Write a BeforeRowRequest event handler to respond to custom information from a client dataset. BeforeRowRequest is part of the mechanism by which a provider and a client dataset communicate information about data fetching. When the provider is part of a stateless application server, this mechanism allows the provider and the client dataset to communicate persistent state information.

When the client dataset fetches detail information about the current record or refreshes the current record, the following events occur:

1.The client dataset receives a BeforeRowRequest event, where it can encode custom information into an OleVariant.

2.The provider receives a BeforeRowRequest event, where the custom information appears as the OwnerData parameter. The provider can respond to or change that information before it looks up the requested information.

3.The provider creates a delta packet that contains the requested information.

4.The provider receives an AfterRowRequest event, where it can encode custom information into its OwnerData parameter or respond to information from the BeforeRowRequest event handler.

5.The client dataset receives an AfterRowRequest event, where it can respond to the custom information returned by the provider in its AfterRowRequest event handler.

See Also

AS_RowRequest

RowRequest

AfterRowRequest

BeforeRowRequest

FetchBlobs

FetchParams

RefreshRecord

Communicating with the Client Dataset

82.1.4.6.7 TXMLTransformProvider.OnDataRequest Event

Occurs when a provider calls its DataRequest method.

Pascal

```
property OnDataRequest;
```

C++

```
__property OnDataRequest;
```

Description

Write an OnDataRequest event handler to do custom processing of data. OnDataRequest has no built-in use. Instead, it allows developers to augment the IAppServer interface so that clients in a multi-tiered application can communicate directly with an individual provider.

If an application does not provide an OnDataRequest event handler, calls to DataRequest return NULL.

Note: To provide the same sort of custom processing on the client side, use an automation method on the connection component.

See Also

DataRequest

IAppServer

Communicating with Providers Using Custom Events

Communicating with the Client Dataset

Responding to Client-generated Events

82.2 Types

The following table lists types in this documentation.

Types

Name	Description
TRowEvent (see page 7761)	TRowEvent is the type for event handlers of events that occur at discrete points during an XML transformation.
TTranslateEvent (see page 7762)	TTranslateEvent is the type for the OnTranslate event handler.

82.2.1 Xmlxform.TRowEvent Type

TRowEvent is the type for event handlers of events that occur at discrete points during an XML transformation.

Pascal

```
TRowEvent = procedure (Sender: TObject; Id: string; SrcNode: IDOMNode; DestNode: IDOMNode) of object;
```

C++

```
(Sender: TObject; Id: string; SrcNode: IDOMNode; DestNode: IDOMNode) ( TRowEvent)();
```

File

Xmlxform

Description

TRowEvent is the type for the event handlers of events that occur while TXMLTransform (see page 7727) is converting an XML document into an XML data packet, or vice versa.

Sender is the XML transform component that generates the event.

Id is a string that identifies the node in the XML document that is about to be translated. This identifier is assigned using the Node Repository page of the Xmlmapper utility when the transformation file is generated. If the node does not have an assigned identifier, Id is an empty string.

SrcNode is the DOM interface for the node that is about to be translated. This node may correspond to the entire XML data packet, a dataset field in the XML data packet, or a record in the XML data packet.

DestNode is the DOM interface for the translated value of SrcNode.

See Also

TXMLTransform (see page 7727)

Procedural types

Creating events

82.2.2 Xmlxform.TTranslateEvent Type

TTranslateEvent is the type for the OnTranslate event handler.

Pascal

```
TTranslateEvent = procedure (Sender: TObject; Id: string; SrcNode: IDOMNode; var Value:
string; DestNode: IDOMNode) of object;
```

C++

```
(Sender: TObject; Id: string; SrcNode: IDOMNode; var Value: string; DestNode: IDOMNode) (
TTranslateEvent)();
```

File

Xmlxform

Description

TTranslateEvent is the type of the OnTranslate event handler. OnTranslate occurs when an XML transform component needs to translate user-defined nodes. This event can occur when translating XML documents into data packets or when translating data packets into XML documents.

Sender is the XML transform component that generates the event.

Id is a string that identifies the node in the XML document. This identifier is assigned using the Node Repository page of Xmlmapper.exe when the transformation file is generated.

SrcNode is the DOM interface for the node being translated. If the current translation is from XML document to data packet, this is the node with the Id specified by Id. If the current translation is from a data packet to an XML document, this is a node in the XML form of the data packet.

Value returns the translation of SrcNode. If the current translation is from XML document to data packet, on entry to the event handler this is the value of SrcNode, if available. On exit, this is a string representation of the value to be assigned to the corresponding field in the data packet. If the current translation is from data packet to XML document, on entry to the event handler this is a string representation of the field (or parameter) value. On exit, it should be the value of the XML node that has the identifier Id (including any subnodes, if appropriate).

DestNode is the DOM interface for the translated node. If the current translation is from XML document to data packet, this is the node for the field in the XML data packet. If the current translation is from data packet to XML document, this is the XML node that has the identifier ID.

See Also

TXMLTransform (see page 7727)

Procedural types

Creating events

83 XPActnCtrls Namespace

This is namespace XPActnCtrls.

Topics

Name	Description
Classes (see page 7764)	The following table lists classes in this documentation.

Classes

	Name	Description
	TXPStyleAddRemoveItem (see page 7764)	This is class XPActnCtrls.TXPStyleAddRemoveItem.
	TXPStyleButton (see page 7767)	This is class XPActnCtrls.TXPStyleButton.
	TXPStyleCustomizePopup (see page 7769)	This is class XPActnCtrls.TXPStyleCustomizePopup.
	TXPStyleDropDownBtn (see page 7770)	This is class XPActnCtrls.TXPStyleDropDownBtn.
	TXPStyleExpandBtn (see page 7772)	This is class XPActnCtrls.TXPStyleExpandBtn.
	TXPStyleMenuButton (see page 7774)	This is class XPActnCtrls.TXPStyleMenuButton.
	TXPStyleMenuItem (see page 7775)	This is class XPActnCtrls.TXPStyleMenuItem.
	TXPStylePopupMenu (see page 7778)	This is class XPActnCtrls.TXPStylePopupMenu.
	TXPStyleToolScrollBtn (see page 7782)	This is class XPActnCtrls.TXPStyleToolScrollBtn.

83.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXPStyleAddRemoveItem (see page 7764)	This is class XPActnCtrls.TXPStyleAddRemoveItem.
	TXPStyleButton (see page 7767)	This is class XPActnCtrls.TXPStyleButton.
	TXPStyleCustomizePopup (see page 7769)	This is class XPActnCtrls.TXPStyleCustomizePopup.
	TXPStyleDropDownBtn (see page 7770)	This is class XPActnCtrls.TXPStyleDropDownBtn.
	TXPStyleExpandBtn (see page 7772)	This is class XPActnCtrls.TXPStyleExpandBtn.
	TXPStyleMenuButton (see page 7774)	This is class XPActnCtrls.TXPStyleMenuButton.
	TXPStyleMenuItem (see page 7775)	This is class XPActnCtrls.TXPStyleMenuItem.
	TXPStylePopupMenu (see page 7778)	This is class XPActnCtrls.TXPStylePopupMenu.
	TXPStyleToolScrollBtn (see page 7782)	This is class XPActnCtrls.TXPStyleToolScrollBtn.

83.1.1 TXPStyleAddRemoveItem Class

Class Hierarchy

Pascal

```
TXPStyleAddRemoveItem = class(TCustomAddRemoveItem);
```

C++

```
class TXPStyleAddRemoveItem : public TCustomAddRemoveItem;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleAddRemoveItem.

83.1.1.1 TXPStyleAddRemoveItem Members

The following tables list the members exposed by TXPStyleAddRemoveItem.

Protected Methods

	Name	Description
	DrawBackground (see page 7765)	Draws a background image on the control.
	DrawGlyph (see page 7766)	Draws a symbol on the control.
	DrawSeparator (see page 7766)	This is DrawSeparator, a member of class TXPStyleAddRemoveItem.
	DrawText (see page 7766)	This is DrawText, a member of class TXPStyleAddRemoveItem.
	DrawUnusedEdges (see page 7766)	This is DrawUnusedEdges, a member of class TXPStyleAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 7765)	Calculates the size of the add/remove menu item.

83.1.1.2 TXPStyleAddRemoveItem Methods

The methods of the TXPStyleAddRemoveItem class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7765)	Draws a background image on the control.
	DrawGlyph (see page 7766)	Draws a symbol on the control.
	DrawSeparator (see page 7766)	This is DrawSeparator, a member of class TXPStyleAddRemoveItem.
	DrawText (see page 7766)	This is DrawText, a member of class TXPStyleAddRemoveItem.
	DrawUnusedEdges (see page 7766)	This is DrawUnusedEdges, a member of class TXPStyleAddRemoveItem.

Public Methods

	Name	Description
	CalcBounds (see page 7765)	Calculates the size of the add/remove menu item.

83.1.1.2.1 TXPStyleAddRemoveItem.CalcBounds Method

Calculates the size of the add/remove menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

CalcBounds updates the Height, Width, and TextBounds properties to reflect current conditions (such as the item's caption and the current font).

See Also

Height

Width

TextBounds

83.1.1.2.2 TXPStyleAddRemoveItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

83.1.1.2.3 TXPStyleAddRemoveItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

83.1.1.2.4 TXPStyleAddRemoveItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TXPStyleAddRemoveItem.

83.1.1.2.5 TXPStyleAddRemoveItem.DrawText Method

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleAddRemoveItem.

83.1.1.2.6 TXPStyleAddRemoveItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

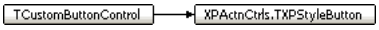
```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TXPStyleAddRemoveItem.

83.1.2 TXPStyleButton Class

Class Hierarchy



Pascal

```
TXPStyleButton = class(TCustomButtonControl);
```

C++

```
class TXPStyleButton : public TCustomButtonControl;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleButton.

83.1.2.1 TXPStyleButton Members

The following tables list the members exposed by TXPStyleButton.

Protected Methods

	Name	Description
	DrawBackground (see page 7767)	Draws a background image on the control.
	DrawFrame (see page 7768)	This is DrawFrame, a member of class TXPStyleButton.
	DrawGlyph (see page 7768)	Draws a symbol on the control.
	DrawText (see page 7768)	This is DrawText, a member of class TXPStyleButton.

83.1.2.2 TXPStyleButton Methods

The methods of the TXPStyleButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7767)	Draws a background image on the control.
	DrawFrame (see page 7768)	This is DrawFrame, a member of class TXPStyleButton.
	DrawGlyph (see page 7768)	Draws a symbol on the control.
	DrawText (see page 7768)	This is DrawText, a member of class TXPStyleButton.

83.1.2.2.1 TXPStyleButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

83.1.2.2.2 TXPStyleButton.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TXPStyleButton.

83.1.2.2.3 TXPStyleButton.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

83.1.2.2.4 TXPStyleButton.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleButton.

83.1.3 TXPStyleCustomizePopup Class

Class Hierarchy

Pascal

```
TXPStyleCustomizePopup = class(TCustomizeActionToolBar);
```

C++

```
class TXPStyleCustomizePopup : public TCustomizeActionToolBar;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleCustomizePopup.

83.1.3.1 TXPStyleCustomizePopup Members

The following tables list the members exposed by TXPStyleCustomizePopup.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 7770)	This is GetAddRemoveItemClass, a member of class TXPStyleCustomizePopup.
	GetDefaultColorMapClass (see page 7770)	This is GetDefaultColorMapClass, a member of class TXPStyleCustomizePopup.
	NCPaint (see page 7770)	This is NCPaint, a member of class TXPStyleCustomizePopup.

Public Methods

	Name	Description
	Create (see page 7769)	Creates and initializes a TCustomActionPopupMenu instance.

83.1.3.2 TXPStyleCustomizePopup.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXPStyleCustomizePopup(TComponent * AOwner);
```

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

83.1.3.3 TXPStyleCustomizePopup Methods

The methods of the TXPStyleCustomizePopup class are listed here.

Protected Methods

	Name	Description
	GetAddRemoveItemClass (see page 7770)	This is GetAddRemoveItemClass, a member of class TXPStyleCustomizePopup.
	GetDefaultColorMapClass (see page 7770)	This is GetDefaultColorMapClass, a member of class TXPStyleCustomizePopup.
	NCPaint (see page 7770)	This is NCPaint, a member of class TXPStyleCustomizePopup.

83.1.3.3.1 TXPStyleCustomizePopup.GetAddRemoveItemClass Method

Pascal

```
function GetAddRemoveItemClass: TCustomAddRemoveItemClass; override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass();
```

Description

This is GetAddRemoveItemClass, a member of class TXPStyleCustomizePopup.

83.1.3.3.2 TXPStyleCustomizePopup.GetDefaultColorMapClass Method

Pascal

```
function GetDefaultColorMapClass: TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetDefaultColorMapClass();
```

Description

This is GetDefaultColorMapClass, a member of class TXPStyleCustomizePopup.

83.1.3.3.3 TXPStyleCustomizePopup.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TXPStyleCustomizePopup.

83.1.4 TXPStyleDropDownBtn Class

Class Hierarchy

Pascal

```
TXPStyleDropDownBtn = class(TCustomDropDownButton);
```

C++

```
class TXPStyleDropDownBtn : public TCustomDropDownButton;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleDropDownBtn.

83.1.4.1 TXPStyleDropDownBtn Members

The following tables list the members exposed by TXPStyleDropDownBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 7771)	Draws a background image on the control.
	DrawFrame (see page 7772)	This is DrawFrame, a member of class TXPStyleDropDownBtn.
	DrawGlyph (see page 7772)	Draws a symbol on the control.
	GetPopupClass (see page 7772)	This is GetPopupClass, a member of class TXPStyleDropDownBtn.

83.1.4.2 TXPStyleDropDownBtn Methods

The methods of the TXPStyleDropDownBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7771)	Draws a background image on the control.
	DrawFrame (see page 7772)	This is DrawFrame, a member of class TXPStyleDropDownBtn.
	DrawGlyph (see page 7772)	Draws a symbol on the control.
	GetPopupClass (see page 7772)	This is GetPopupClass, a member of class TXPStyleDropDownBtn.

83.1.4.2.1 TXPStyleDropDownBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

83.1.4.2.2 TXPStyleDropDownBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TXPStyleDropDownBtn.

83.1.4.2.3 TXPStyleDropDownBtn.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

83.1.4.2.4 TXPStyleDropDownBtn.GetPopupClass Method

Pascal

```
function GetPopupClass: TCustomActionBarClass; override;
```

C++

```
virtual __fastcall TCustomActionBarClass GetPopupClass();
```

Description

This is GetPopupClass, a member of class TXPStyleDropDownBtn.

83.1.5 TXPStyleExpandBtn Class

Class Hierarchy

Pascal

```
TXPStyleExpandBtn = class(TCustomMenuExpandBtn);
```

C++

```
class TXPStyleExpandBtn : public TCustomMenuExpandBtn;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleExpandBtn.

83.1.5.1 TXPStyleExpandBtn Members

The following tables list the members exposed by TXPStyleExpandBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 7773)	Draws a background image on the control.
	DrawFrame (see page 7773)	This is DrawFrame, a member of class TXPStyleExpandBtn.

83.1.5.2 TXPStyleExpandBtn Methods

The methods of the TXPStyleExpandBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7773)	Draws a background image on the control.
	DrawFrame (see page 7773)	This is DrawFrame, a member of class TXPStyleExpandBtn.

83.1.5.2.1 TXPStyleExpandBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

83.1.5.2.2 TXPStyleExpandBtn.DrawFrame Method

Pascal

```
procedure DrawFrame(ARect: TRect; Down: Boolean); override;
```

C++

```
virtual __fastcall DrawFrame(TRect ARect, Boolean Down);
```

Description

This is DrawFrame, a member of class TXPStyleExpandBtn.

83.1.6 TXPStyleMenuButton Class

Class Hierarchy

Pascal

```
TXPStyleMenuButton = class(TCustomMenuButton);
```

C++

```
class TXPStyleMenuButton : public TCustomMenuButton;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleMenuButton.

83.1.6.1 TXPStyleMenuButton Members

The following tables list the members exposed by TXPStyleMenuButton.

Protected Methods

	Name	Description
	DrawBackground (see page 7774)	Draws a background image on the control.
	DrawText (see page 7775)	This is DrawText, a member of class TXPStyleMenuButton.

83.1.6.2 TXPStyleMenuButton Methods

The methods of the TXPStyleMenuButton class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7774)	Draws a background image on the control.
	DrawText (see page 7775)	This is DrawText, a member of class TXPStyleMenuButton.

83.1.6.2.1 TXPStyleMenuButton.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

83.1.6.2.2 TXPStyleMenuButton.DrawText Method

Pascal

```
procedure DrawText(var ARect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect ARect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleMenuButton.

83.1.7 TXPStyleMenuItem Class

Class Hierarchy

Pascal

```
TXPStyleMenuItem = class(TCustomMenuItem);
```

C++

```
class TXPStyleMenuItem : public TCustomMenuItem;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleMenuItem.

83.1.7.1 TXPStyleMenuItem Members

The following tables list the members exposed by TXPStyleMenuItem.

Protected Methods

	Name	Description
	CalcLayout (see page 7776)	This is CalcLayout, a member of class TXPStyleMenuItem.
	DrawBackground (see page 7777)	Draws a background image on the control.
	DrawGlyph (see page 7777)	Draws a symbol on the control.
	DrawSeparator (see page 7777)	This is DrawSeparator, a member of class TXPStyleMenuItem.
	DrawShadowedText (see page 7777)	This is DrawShadowedText, a member of class TXPStyleMenuItem.
	DrawSubMenuGlyph (see page 7778)	This is DrawSubMenuGlyph, a member of class TXPStyleMenuItem.
	DrawText (see page 7778)	This is DrawText, a member of class TXPStyleMenuItem.

	DrawUnusedEdges (see page 7778)	This is DrawUnusedEdges, a member of class TXPStyleMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7776)	Calculates the size of the menu item.

83.1.7.2 TXPStyleMenuItem Methods

The methods of the TXPStyleMenuItem class are listed here.

Protected Methods

	Name	Description
	CalcLayout (see page 7776)	This is CalcLayout, a member of class TXPStyleMenuItem.
	DrawBackground (see page 7777)	Draws a background image on the control.
	DrawGlyph (see page 7777)	Draws a symbol on the control.
	DrawSeparator (see page 7777)	This is DrawSeparator, a member of class TXPStyleMenuItem.
	DrawShadowedText (see page 7777)	This is DrawShadowedText, a member of class TXPStyleMenuItem.
	DrawSubMenuGlyph (see page 7778)	This is DrawSubMenuGlyph, a member of class TXPStyleMenuItem.
	DrawText (see page 7778)	This is DrawText, a member of class TXPStyleMenuItem.
	DrawUnusedEdges (see page 7778)	This is DrawUnusedEdges, a member of class TXPStyleMenuItem.

Public Methods

	Name	Description
	CalcBounds (see page 7776)	Calculates the size of the menu item.

83.1.7.2.1 TXPStyleMenuItem.CalcBounds Method

Calculates the size of the menu item.

Pascal

```
procedure CalcBounds; override;
```

C++

```
virtual __fastcall CalcBounds();
```

Description

The size of the control depends on many factors, such as whether or not the menu item is a separator or not. CalcBounds computes the boundaries of the menu item considering these kinds of factors. At the end, it calls SetBounds and other routines to ensure that the values are set properly.

See Also

SetBounds

83.1.7.2.2 TXPStyleMenuItem.CalcLayout Method

Pascal

```
procedure CalcLayout; override;
```

C++

```
virtual __fastcall CalcLayout();
```

Description

This is CalcLayout, a member of class TXPStyleMenuItem.

83.1.7.2.3 TXPStyleMenuItem.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

83.1.7.2.4 TXPStyleMenuItem.DrawGlyph Method

Draws a symbol on the control.

Pascal

```
procedure DrawGlyph(const Location: TPoint); override;
```

C++

```
virtual __fastcall DrawGlyph(const TPoint Location);
```

Description

Use DrawGlyph to draw a symbol on the control at the specified location.

See Also

DrawBackground

DrawText

83.1.7.2.5 TXPStyleMenuItem.DrawSeparator Method

Pascal

```
procedure DrawSeparator(const Offset: Integer); override;
```

C++

```
virtual __fastcall DrawSeparator(const int Offset);
```

Description

This is DrawSeparator, a member of class TXPStyleMenuItem.

83.1.7.2.6 TXPStyleMenuItem.DrawShadowedText Method

Pascal

```
procedure DrawShadowedText(Rect: TRect; Flags: Cardinal; Text: String; TextColor: TColor;
ShadowColor: TColor); override;
```

C++

```
virtual __fastcall DrawShadowedText(TRect Rect, unsigned Flags, UnicodeString Text, TColor
TextColor, TColor ShadowColor);
```

Description

This is DrawShadowedText, a member of class TXPStyleMenuItem.

83.1.7.2.7 TXPStyleMenuItem.DrawSubMenuGlyph Method

Pascal

```
procedure DrawSubMenuGlyph; override;
```

C++

```
virtual __fastcall DrawSubMenuGlyph();
```

Description

This is DrawSubMenuGlyph, a member of class TXPStyleMenuItem.

83.1.7.2.8 TXPStyleMenuItem.DrawText Method

Pascal

```
procedure DrawText(var Rect: TRect; var Flags: Cardinal; Text: String); override;
```

C++

```
virtual __fastcall DrawText(TRect Rect, unsigned Flags, UnicodeString Text);
```

Description

This is DrawText, a member of class TXPStyleMenuItem.

83.1.7.2.9 TXPStyleMenuItem.DrawUnusedEdges Method

Pascal

```
procedure DrawUnusedEdges; override;
```

C++

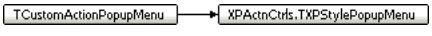
```
virtual __fastcall DrawUnusedEdges();
```

Description

This is DrawUnusedEdges, a member of class TXPStyleMenuItem.

83.1.8 TXPStylePopupMenu Class

Class Hierarchy



Pascal

```
TXPStylePopupMenu = class(TCustomActionPopupMenu);
```

C++

```
class TXPStylePopupMenu : public TCustomActionPopupMenu;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStylePopupMenu.

83.1.8.1 TXPStylePopupMenu Members

The following tables list the members exposed by TXPStylePopupMenu.

Protected Methods

	Name	Description
	DisplayShadow (see page 7780)	This is DisplayShadow, a member of class TXPStylePopupMenu.
	GetExpandBtnClass (see page 7780)	This is GetExpandBtnClass, a member of class TXPStylePopupMenu.
	HideShadow (see page 7781)	This is HideShadow, a member of class TXPStylePopupMenu.
	NCPaint (see page 7781)	This is NCPaint, a member of class TXPStylePopupMenu.
	PositionPopup (see page 7781)	This is PositionPopup, a member of class TXPStylePopupMenu.
	VisibleChanging (see page 7781)	Provides the interface for a method that responds to changes in the control's visibility.

Public Methods

	Name	Description
	Create (see page 7779)	Creates and initializes a TCustomActionPopupMenu instance.
	Destroy (see page 7779)	Destroys an instance of TCustomActionMenuBar.

83.1.8.2 TXPStylePopupMenu.Create Constructor

Creates and initializes a TCustomActionPopupMenu instance.

Pascal

```
constructor Create(AOwner: TComponent); override;
```

C++

```
virtual __fastcall TXPStylePopupMenu(TComponent * AOwner);
```

Description

Create generates a TCustomActionPopupMenu and initializes its properties.

```
__fastcall virtual TCustomActionMenuBar (Classes::TComponentTComponent_Object* AOwner);
__fastcall TCustomActionMenuBar (HWND ParentWindow) :
Controls::TWinControlTWinControl_Create(ParentWindow) { }
```

83.1.8.3 TXPStylePopupMenu.Destroy Destructor

Destroys an instance of TCustomActionMenuBar.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXPStylePopupMenu();
```

Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TCustomActionMenuBar reference is not nil, and only then calls Destroy.

See Also

System::TObject::Free (see page 3372)

Destroy

83.1.8.4 TXPStylePopupMenu Methods

The methods of the TXPStylePopupMenu class are listed here.

Protected Methods

	Name	Description
	DisplayShadow (see page 7780)	This is DisplayShadow, a member of class TXPStylePopupMenu.
	GetExpandBtnClass (see page 7780)	This is GetExpandBtnClass, a member of class TXPStylePopupMenu.
	HideShadow (see page 7781)	This is HideShadow, a member of class TXPStylePopupMenu.
	NCPaint (see page 7781)	This is NCPaint, a member of class TXPStylePopupMenu.
	PositionPopup (see page 7781)	This is PositionPopup, a member of class TXPStylePopupMenu.
	VisibleChanging (see page 7781)	Provides the interface for a method that responds to changes in the control's visibility.

83.1.8.4.1 TXPStylePopupMenu.DisplayShadow Method

Pascal

```
procedure DisplayShadow; override;
```

C++

```
virtual __fastcall DisplayShadow();
```

Description

This is DisplayShadow, a member of class TXPStylePopupMenu.

83.1.8.4.2 TXPStylePopupMenu.GetExpandBtnClass Method

Pascal

```
function GetExpandBtnClass: TCustomMenuExpandBtnClass; override;
```

C++

```
virtual __fastcall TCustomMenuExpandBtnClass GetExpandBtnClass();
```

Description

This is GetExpandBtnClass, a member of class TXPStylePopupMenu.

83.1.8.4.3 TXPStylePopupMenu.HideShadow Method

Pascal

```
procedure HideShadow; override;
```

C++

```
virtual __fastcall HideShadow();
```

Description

This is HideShadow, a member of class TXPStylePopupMenu.

83.1.8.4.4 TXPStylePopupMenu.NCPaint Method

Pascal

```
procedure NCPaint(DC: HDC); override;
```

C++

```
virtual __fastcall NCPaint(HDC DC);
```

Description

This is NCPaint, a member of class TXPStylePopupMenu.

83.1.8.4.5 TXPStylePopupMenu.PositionPopup Method

Pascal

```
procedure PositionPopup(AnOwner: TCustomActionBar; ParentItem: TCustomActionControl); override;
```

C++

```
virtual __fastcall PositionPopup(TCustomActionBar AnOwner, TCustomActionControl ParentItem);
```

Description

This is PositionPopup, a member of class TXPStylePopupMenu.

83.1.8.4.6 TXPStylePopupMenu.VisibleChanging Method

Provides the interface for a method that responds to changes in the control's visibility.

Pascal

```
procedure VisibleChanging; override;
```

C++

```
virtual __fastcall VisibleChanging();
```

Description

VisibleChanging is called before the value of a control's Visible property changes. As implemented in TControl, it does nothing. Override VisibleChanging to respond to changes in the control's visibility. For example, if a control should not change its visibility, override VisibleChanging to raise an exception when an attempt is made to alter the control's visibility.

See Also

Visible

83.1.9 TXPStyleToolScrollBtn Class

Class Hierarchy

Pascal

```
TXPStyleToolScrollBtn = class(TCustomToolScrollBtn);
```

C++

```
class TXPStyleToolScrollBtn : public TCustomToolScrollBtn;
```

File

XPActnCtrls

Description

This is class XPActnCtrls.TXPStyleToolScrollBtn.

83.1.9.1 TXPStyleToolScrollBtn Members

The following tables list the members exposed by TXPStyleToolScrollBtn.

Protected Methods

	Name	Description
	DrawBackground (see page 7782)	Draws a background image on the control.

83.1.9.2 TXPStyleToolScrollBtn Methods

The methods of the TXPStyleToolScrollBtn class are listed here.

Protected Methods

	Name	Description
	DrawBackground (see page 7782)	Draws a background image on the control.

83.1.9.2.1 TXPStyleToolScrollBtn.DrawBackground Method

Draws a background image on the control.

Pascal

```
procedure DrawBackground(var PaintRect: TRect); override;
```

C++

```
virtual __fastcall DrawBackground(TRect PaintRect);
```

Description

Use DrawBackground to draw a background image on the control within the area specified by PaintRect.

See Also

DrawGlyph

DrawText

84 XPMan Namespace

This is namespace XPMan.

Topics

Name	Description
Classes (see page 7786)	The following table lists classes in this documentation.

Classes

	Name	Description
	TXPManifest (see page 7786)	This is class XPMan.TXPManifest.

84.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXPManifest (see page 7786)	This is class XPMan.TXPManifest.

84.1.1 TXPManifest Class

Class Hierarchy

Pascal

```
[RootDesignerSerializerAttribute('', '', False)]
TXPManifest = class(TComponent);
```

C++

```
[RootDesignerSerializerAttribute('', '', False)]
class TXPManifest : public TComponent;
```

File

XPMan

Description

This is class XPMan.TXPManifest.

85 XPStyleActnCtrls Namespace

This is namespace XPStyleActnCtrls.

Topics

Name	Description
Classes (see page 7788)	The following table lists classes in this documentation.
Variables (see page 7791)	The following table lists variables in this documentation.

Classes

	Name	Description
	TXPStyleActionBars (see page 7788)	This is class XPStyleActnCtrls.TXPStyleActionBars.

Variables

Name	Description
XPStyle (see page 7791)	This is variable XPStyleActnCtrls.XPStyle.

85.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	TXPStyleActionBars (see page 7788)	This is class XPStyleActnCtrls.TXPStyleActionBars.

85.1.1 TXPStyleActionBars Class

Class Hierarchy

Pascal

```
TXPStyleActionBars = class(TActionBarStyleEx);
```

C++

```
class TXPStyleActionBars : public TActionBarStyleEx;
```

File

XPStyleActnCtrls

Description

This is class XPStyleActnCtrls.TXPStyleActionBars.

85.1.1.1 TXPStyleActionBars Members

The following tables list the members exposed by TXPStyleActionBars.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 7789)	Returns the base class for ActionBand menus for customizing action bands.
	GetColorMapClass (see page 7789)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 7789)	Returns the control class type on an action bar item.
	GetPopupClass (see page 7790)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 7790)	This is GetScrollBtnClass, a member of class TXPStyleActionBars.
	GetStyleName (see page 7790)	Returns the style associated with action manager.

85.1.1.2 TXPStyleActionBars Methods

The methods of the TXPStyleActionBars class are listed here.

Public Methods

	Name	Description
	GetAddRemoveItemClass (see page 7789)	Returns the base class for ActionBand menus for customizing action bands.

	GetColorMapClass (see page 7789)	Returns the class type of the colormap for an action bar.
	GetControlClass (see page 7789)	Returns the control class type on an action bar item.
	GetPopupClass (see page 7790)	Returns the popup class added to a menu action band.
	GetScrollBtnClass (see page 7790)	This is GetScrollBtnClass, a member of class TXPStyleActionBars.
	GetStyleName (see page 7790)	Returns the style associated with action manager.

85.1.1.2.1 TXPStyleActionBars.GetAddRemoveItemClass Method

Returns the base class for ActionBand menus for customizing action bands.

Pascal

```
function GetAddRemoveItemClass(ActionBar: TCustomActionBar): TCustomAddRemoveItemClass;
override;
```

C++

```
virtual __fastcall TCustomAddRemoveItemClass GetAddRemoveItemClass(TCustomActionBar ActionBar);
```

Description

Call GetAddRemoveItemClass to customize action bands.

See Also

TCustomAddRemoveItem

85.1.1.2.2 TXPStyleActionBars.GetColorMapClass Method

Returns the class type of the colormap for an action bar.

Pascal

```
function GetColorMapClass(ActionBar: TCustomActionBar): TCustomColorMapClass; override;
```

C++

```
virtual __fastcall TCustomColorMapClass GetColorMapClass(TCustomActionBar ActionBar);
```

Description

Returns the class type TCustomActionBarColorMap, which is a colormap associated with an action bar. Override this method to customize the appearance of your applications.

85.1.1.2.3 TXPStyleActionBars.GetControlClass Method

Returns the control class type on an action bar item.

Pascal

```
function GetControlClass(ActionBar: TCustomActionBar; AnItem: TActionClientItem):
TCustomActionControlClass; override;
```

C++

```
virtual __fastcall TCustomActionControlClass GetControlClass(TCustomActionBar ActionBar,
TActionClientItem AnItem);
```

Description

Returns the control class type on an item in the action bar, such as a button or menu item.

85.1.1.2.4 TXPStyleActionBars.GetPopupClass Method

Returns the popup class added to a menu action band.

Pascal

```
function GetPopupClass(ActionBar: TCustomActionBar): TCustomPopupClass; override;
```

C++

```
virtual __fastcall TCustomPopupClass GetPopupClass(TCustomActionBar ActionBar);
```

Description

Call GetPopupClass to find the popup menu associated with a menu action band. Override GetPopupClass to change which popup class is associated with a menu action band depending on the state of the control.

See Also

TCustomActionPopupMenu

85.1.1.2.5 TXPStyleActionBars.GetScrollBtnClass Method

Pascal

```
function GetScrollBtnClass: TCustomToolScrollBtnClass; override;
```

C++

```
virtual __fastcall TCustomToolScrollBtnClass GetScrollBtnClass();
```

Description

This is GetScrollBtnClass, a member of class TXPStyleActionBars.

85.1.1.2.6 TXPStyleActionBars.GetStyleName Method

Returns the style associated with action manager.

Pascal

```
function GetStyleName: string; override;
```

C++

```
virtual __fastcall UnicodeString GetStyleName();
```

Description

Returns the value of the Style property.

85.2 Variables

The following table lists variables in this documentation.

Variables

Name	Description
XPStyle (see page 7791)	This is variable XPStyleActnCtrls.XPStyle.

85.2.1 XPStyleActnCtrls.XPStyle Variable

Pascal

```
XPStyle: TXPStyleActionBars;
```

C++

```
TXPStyleActionBars XPStyle;
```

File

XPStyleActnCtrls

Description

This is variable XPStyleActnCtrls.XPStyle.

86 XSBuiltIns Namespace

This is namespace XSBuiltIns.

Topics

Name	Description
Classes (see page 7795)	The following table lists classes in this documentation.
Functions (see page 7861)	The following table lists functions in this documentation.
Types (see page 7865)	The following table lists types in this documentation.
Constants (see page 7866)	The following table lists constants in this documentation.

Classes

	Name	Description
	EXSDateTimeException (see page 7795)	EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.
	EXSDecimalException (see page 7796)	EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.
	EXSHexBinaryException (see page 7796)	EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.
	TXMLData (see page 7797)	This is class XSBuiltIns.TXMLData.
	TXSBoolean (see page 7801)	This is class XSBuiltIns.TXSBoolean.
	TXSCustomDateTime (see page 7803)	TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.
	TXSDate (see page 7822)	TXSDate implements the XML schema date type so that it can be used on invokable interfaces.
	TXSDateTime (see page 7828)	TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.
	TXSDecimal (see page 7832)	TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.
	TXSDuration (see page 7836)	TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.
	TXSHexBinary (see page 7840)	TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.
	TXSInteger (see page 7843)	This is class XSBuiltIns.TXSInteger.
	TXSLong (see page 7845)	This is class XSBuiltIns.TXSLong.
	TXSString (see page 7847)	This is class XSBuiltIns.TXSString.
	TXSTime (see page 7849)	TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

Constants

Name	Description
IncrementAmount (see page 7866)	This is constant XSBuiltIns.IncrementAmount.
MaxDay (see page 7867)	This is constant XSBuiltIns.MaxDay.
MaxMonth (see page 7867)	This is constant XSBuiltIns.MaxMonth.
MinDay (see page 7867)	This is constant XSBuiltIns.MinDay.
MinMonth (see page 7868)	This is constant XSBuiltIns.MinMonth.
SDecimal (see page 7868)	This is constant XSBuiltIns.SDecimal.
SHexMarker (see page 7868)	This is constant XSBuiltIns.SHexMarker.
SLocalTimeMarker (see page 7869)	This is constant XSBuiltIns.SLocalTimeMarker.
SNAN (see page 7869)	This is constant XSBuiltIns.SNAN.
SNegative (see page 7869)	This is constant XSBuiltIns.SNegative.
SoapDecimalSeparator (see page 7870)	This is constant XSBuiltIns.SoapDecimalSeparator.
SoapTimePrefix (see page 7870)	This is constant XSBuiltIns.SoapTimePrefix.
SPlus (see page 7870)	This is constant XSBuiltIns.SPlus.
SSciNotationMarker (see page 7871)	This is constant XSBuiltIns.SSciNotationMarker.
XMLDateSeparator (see page 7871)	This is constant XSBuiltIns.XMLDateSeparator.
XMLDayPos (see page 7871)	This is constant XSBuiltIns.XMLDayPos.
XMLDefaultYearDigits (see page 7872)	This is constant XSBuiltIns.XMLDefaultYearDigits.
XMLDurationDay (see page 7872)	This is constant XSBuiltIns.XMLDurationDay.
XMLDurationHour (see page 7872)	This is constant XSBuiltIns.XMLDurationHour.
XMLDurationMinute (see page 7873)	This is constant XSBuiltIns.XMLDurationMinute.
XMLDurationMonth (see page 7873)	This is constant XSBuiltIns.XMLDurationMonth.
XMLDurationSecond (see page 7873)	This is constant XSBuiltIns.XMLDurationSecond.
XMLDurationStart (see page 7874)	This is constant XSBuiltIns.XMLDurationStart.
XMLDurationYear (see page 7874)	This is constant XSBuiltIns.XMLDurationYear.
XMLHourOffsetMinusMarker (see page 7874)	This is constant XSBuiltIns.XMLHourOffsetMinusMarker.
XMLHourOffsetPlusMarker (see page 7875)	This is constant XSBuiltIns.XMLHourOffsetPlusMarker.
XMLMilSecPos (see page 7875)	This is constant XSBuiltIns.XMLMilSecPos.
XMLMonthPos (see page 7875)	This is constant XSBuiltIns.XMLMonthPos.
XMLTimeSeparator (see page 7876)	This is constant XSBuiltIns.XMLTimeSeparator.
XMLYearPos (see page 7876)	This is constant XSBuiltIns.XMLYearPos.

Functions

	Name	Description
	DateTimeToXMLTime (see page 7861)	Returns the XML string that represents a TDateTime (see page 3323) value.
	DateTimeToXSDateTime (see page 7861)	Returns a TXSDateTime (see page 7828) object that encodes a TDateTime (see page 3323) value.
	GetDataFromFile (see page 7862)	This is function XSBuiltIns.GetDataFromFile.
	InitXSTypes (see page 7862)	This is function XSBuiltIns.InitXSTypes.
	SoapFloatToStr (see page 7863)	This is function XSBuiltIns.SoapFloatToStr.
	SoapStrToFloat (see page 7863)	This is function XSBuiltIns.SoapStrToFloat.
	XMLTimeToDateTime (see page 7863)	Returns a TDateTime (see page 3323) value to represent an XML datetime value.

Types

Name	Description
TXSTimeInstant (see page 7865)	TXSTimeInstant represents the timeInstant type for Web Services applications.

86.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXSDateTimeException (see page 7795)	EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.
	EXSDecimalException (see page 7796)	EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.
	EXSHexBinaryException (see page 7796)	EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.
	TXMLData (see page 7797)	This is class XSBuiltIns.TXMLData.
	TXSBoolean (see page 7801)	This is class XSBuiltIns.TXSBoolean.
	TXSCustomDateTime (see page 7803)	TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.
	TXSDate (see page 7822)	TXSDate implements the XML schema date type so that it can be used on invokable interfaces.
	TXSDateTime (see page 7828)	TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.
	TXSDecimal (see page 7832)	TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.
	TXSDuration (see page 7836)	TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.
	TXSHexBinary (see page 7840)	TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.
	TXSInteger (see page 7843)	This is class XSBuiltIns.TXSInteger.
	TXSLong (see page 7845)	This is class XSBuiltIns.TXSLong.
	TXSString (see page 7847)	This is class XSBuiltIns.TXSString.
	TXSTime (see page 7849)	TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

86.1.1 EXSDateTimeException Class

EXSDateTimeException is the exception class for errors encountered by the remotable objects that represent XML schema date and time values.

Class Hierarchy

Pascal

```
EXSDateTimeException = class(Exception);
```

C++

```
class EXSDateTimeException : public Exception;
```

File

XSBuiltIns

Description

EXSDateTimeException is raised by the TXSDate (see page 7822), TXSTime (see page 7849), TXSDateTime (see page

7828), and TXSDuration (see page 7836) objects when

An application assigns a value that is out of range to one of their properties. Such values include Month values greater than 12, Hour values greater than 23, Minute values greater than 59, and so on.

The string that represents the object's value does not have the proper format and can't be parsed.

See Also

TXSTime (see page 7849)

TXSDate (see page 7822)

TXSDateTime (see page 7828)

TXSDuration (see page 7836)

86.1.2 EXSDecimalException Class

EXSDecimalException is the exception class for errors encountered by the remotable objects that represent decimal values.

Class Hierarchy

Pascal

```
EXSDecimalException = class(Exception);
```

C++

```
class EXSDecimalException : public Exception;
```

File

XSBuiltIns

Description

EXSDecimalException is raised by TXSDecimal (see page 7832) when it is unable to convert between its string value and a TBcd value.

See Also

TXSDecimal (see page 7832)

TBcd

86.1.3 EXSHexBinaryException Class

EXSHexBinaryException is the exception class for errors encountered by the remotable objects that represent hexadecimal values.

Class Hierarchy

Pascal

```
EXSHexBinaryException = class(Exception);
```

C++

```
class EXSHexBinaryException : public Exception;
```

File

XSBuiltIns

Description

EXSHexBinaryException is raised by TXSHexBinary (see page 7840) when it encounters a string that contains invalid characters for a hexadecimal value.

See Also

TXSDecimal (see page 7832)

TBcd

86.1.4 TXMLData Class

Class Hierarchy

Pascal

```
TXMLData = class(TRemotable);
```

C++

```
class TXMLData : public TRemotable;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXMLData.

86.1.4.1 TXMLData Members

The following tables list the members exposed by TXMLData.

Public Methods

	Name	Description
	Create (see page 7798)	Creates and initializes a TRemotable object.
	Destroy (see page 7798)	Frees the memory associated with the TRemotable object.
	LoadFromXML (see page 7799)	This is the overview for the LoadFromXML method overload.
	ObjectToSOAP (see page 7799)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 7800)	Initializes this TRemotable instance to represent the value of its SOAP representation.

Public Properties

	Name	Description
	XMLNode (see page 7801)	This is XMLNode, a member of class TXMLData.

86.1.4.2 TXMLData.Create Constructor

Creates and initializes a TRemotable object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXMLData();
```

Description

The TRemotable constructor does nothing special. It exists primarily because

The remotable type registry knows how to call it.

It is virtual, so that the remotable type registry can also instantiate descendant classes.

See Also

RemTypeRegistry

86.1.4.3 TXMLData.Destroy Destructor

Frees the memory associated with the TRemotable object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXMLData();
```

Description

Use the Free method to free a TRemotable instance. Free checks that the object is not nil before calling Destroy.

On server applications, there is typically no need to explicitly free a TRemotable instance (for example, one created as a return value or output parameter). By default, when a TRemotable descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext property). As long as the remotable object belongs to the data context, the data context handles freeing the object. Similarly, TRemotable instances that are passed in as a parameter belong to the data context.

On client applications, TRemotable instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

System::TObject::Free (see page 3372)

DataContext

86.1.4.4 TXMLData Methods

The methods of the TXMLData class are listed here.

Public Methods

	Name	Description
	LoadFromXML (see page 7799)	This is the overview for the LoadFromXML method overload.
	ObjectToSOAP (see page 7799)	Converts this remotable object to its SOAP representation.
	SOAPToObject (see page 7800)	Initializes this TRemotable instance to represent the value of its SOAP representation.

86.1.4.4.1 LoadFromXML Method

This is the overview for the LoadFromXML method overload.

Overload List

	Name	Description
	TXMLData.LoadFromXML (string) (see page 7799)	This is LoadFromXML, a member of class TXMLData.
	TXMLData.LoadFromXML (WideString) (see page 7799)	This is LoadFromXML, a member of class TXMLData.

86.1.4.4.1.1 TXMLData.LoadFromXML Method (WideString)

Pascal

```
procedure LoadFromXML(const XML: WideString); overload;
```

C++

```
__fastcall LoadFromXML(const BSTR XML);
```

Description

This is LoadFromXML, a member of class TXMLData.

86.1.4.4.1.2 TXMLData.LoadFromXML Method (string)

Pascal

```
procedure LoadFromXML(const XML: string); overload;
```

C++

```
__fastcall LoadFromXML(const UnicodeString XML);
```

Description

This is LoadFromXML, a member of class TXMLData.

86.1.4.4.2 TXMLData.ObjectToSOAP Method

Converts this remotable object to its SOAP representation.

Pascal

```
function ObjectToSOAP(RootNode: IXMLNode; ParentNode: IXMLNode; const ObjConverter:
IObjConverter; const Name: InvString; const URI: InvString; ObjConvOpts:
TObjectConvertOptions; out RefID: InvString): IXMLNode; override;
```

C++

```
virtual __fastcall IXMLNode ObjectToSOAP(IXMLNode RootNode, IXMLNode ParentNode, const
IObjConverter ObjConverter, const InvString Name, const InvString URI, TObjectConvertOptions
```

```
ObjConvOpts, InvString RefID);
```

Description

Override ObjectToSOAP to customize the way your application converts a TRemotable descendant to its SOAP representation. In TRemotable, ObjectToSOAP simply calls the ObjInstanceToSOAP method of the converter that is supplied by the ObjConverter parameter.

RootNode is an ancestor of the generated node to which information about the generated node can be added as attributes.

ParentNode is the node in the SOAP representation that should act as a parent to the node that ObjectToSOAP generates.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

Name is the name of the node that is generated.

URI is the URI for the namespace in which Name is defined.

ObjConvOpts are flags that customize the way the node is serialized.

RefID returns a reference ID for the node. This is used if the node is multi-referenced.

ObjectToSOAP returns the XML for the newly generated node.

See Also

ObjInstanceToSOAP

SOAPToObject

86.1.4.4.3 TXMLData.SOAPToObject Method

Initializes this TRemotable instance to represent the value of its SOAP representation.

Pascal

```
procedure SOAPToObject(const RootNode: IXMLNode; const Node: IXMLNode; const ObjConverter:
IObjConverter); override;
```

C++

```
virtual __fastcall SOAPToObject(const IXMLNode RootNode, const IXMLNode Node, const
IObjConverter ObjConverter);
```

Description

SOAPToObject initializes the published properties of this object instance so that they reflect the header's SOAP representation. Override this method to customize the way your application creates a specific remotable type from its XML representation.

RootNode is an ancestor node of the node for this object that can contain information about the remotable type's value.

Node is the node in the SOAP envelope that represents the remotable type's value.

ObjConverter is the interface of the default converter that translates between remotable objects and their SOAP representation. This is the interface to a TOPToSoapDomConvert instance.

In TRemotable, SOAPToObject simply calls the InitObjectFromSOAP method of the converter supplied by the ObjConverter parameter.

See Also

ObjectToSOAP

InitObjectFromSOAP

86.1.4.5 TXMLData Properties

The properties of the TXMLData class are listed here.

Public Properties

	Name	Description
	XMLNode (see page 7801)	This is XMLNode, a member of class TXMLData.

86.1.4.5.1 TXMLData.XMLNode Property

Pascal

```
property XMLNode: IXMLNode;
```

C++

```
__property IXMLNode XMLNode;
```

Description

This is XMLNode, a member of class TXMLData.

86.1.5 TXSBoolean Class

Class Hierarchy

Pascal

```
TXSBoolean = class(TRemotableXS);
```

C++

```
class TXSBoolean : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSBoolean.

86.1.5.1 TXSBoolean Members

The following tables list the members exposed by TXSBoolean.

Public Methods

	Name	Description
	NativeToXS (see page 7802)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7802)	Assigns the value of a string to the TRemotableXS instance.

Public Properties

	Name	Description
	AsBoolean (see page 7803)	This is AsBoolean, a member of class TXSBoolean.

86.1.5.2 TXSBoolean Methods

The methods of the TXSBoolean class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7802)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7802)	Assigns the value of a string to the TRemotableXS instance.

86.1.5.2.1 TXSBoolean.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

86.1.5.2.2 TXSBoolean.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

NativeToXS

Using Remotable Objects

86.1.5.3 TXSBoolean Properties

The properties of the TXSBoolean class are listed here.

Public Properties

	Name	Description
	AsBoolean (see page 7803)	This is AsBoolean, a member of class TXSBoolean.

86.1.5.3.1 TXSBoolean.AsBoolean Property

Pascal

```
property AsBoolean: Boolean;
```

C++

```
__property Boolean AsBoolean;
```

Description

This is AsBoolean, a member of class TXSBoolean.

86.1.6 TXSCustomDateTime Class

TXSCustomDateTime is the base class for classes that represent both date and time values for an invokable interface.

Class Hierarchy

Pascal

```
TXSCustomDateTime = class(TRemotableXS);
```

C++

```
class TXSCustomDateTime : public TRemotableXS;
```

File

XSBuiltIns

Description

Use TXSCustomDateTime as a base class when defining a class to represent a type that includes both date and time portions and that represents a scalar value in a WSDL document. When creating a TXSCustomDateTime descendant, override the XSToNative and NativeToXS methods to indicate how the value of the class can be converted to and from the string representation that is used to encode the value for transport.

On server applications, instances of TXSCustomDateTime descendants that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSCustomDateTime descendant is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSCustomDateTime instances that it passes as input parameters, and for freeing any TXSCustomDateTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3504)

TXSDateTime (see page 7828)

TXSDuration (see page 7836)

86.1.6.1 TXSCustomDateTime Members

The following tables list the members exposed by TXSCustomDateTime.

Protected Fields

	Name	Description
	FDateTime (see page 7806)	This is FDateTime, a member of class TXSCustomDateTime.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7807)	Returns the value of the AsDateTime (see page 7815) property.
	GetAsUTCDateTime (see page 7807)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 7807)	Returns the value of the Day (see page 7816) property.
	GetFractionalSeconds (see page 7808)	Returns the value of the FractionalSeconds (see page 7817) property.
	GetHour (see page 7808)	Returns the value of the Hour (see page 7817) property.
	GetHourOffset (see page 7808)	Returns the value of the HourOffset (see page 7818) property.
	GetMillisecond (see page 7808)	Returns the value of the Millisecond (see page 7818) property.
	GetMinute (see page 7809)	Returns the value of the Minute (see page 7819) property.
	GetMinuteOffset (see page 7809)	Returns the value of the MinuteOffset (see page 7819) property.
	GetMonth (see page 7809)	Returns the value of the Month (see page 7820) property.
	GetSecond (see page 7810)	Returns the value of the Second (see page 7820) property.
	GetUseZeroMilliseconds (see page 7810)	Returns the value of the UseZeroMilliseconds (see page 7821) property.
	GetYear (see page 7810)	Returns the value of the Year (see page 7821) property.
	SetAsDateTime (see page 7811)	Sets the value of the AsDateTime (see page 7815) property.
	SetAsUTCDateTime (see page 7811)	Sets the value of the AsUTCDateTime (see page 7816) property.
	SetDay (see page 7811)	Sets the value of the Day (see page 7816) property.
	SetFractionalSeconds (see page 7811)	Sets the value of the FractionalSeconds (see page 7817) property.
	SetHour (see page 7812)	Sets the value of the Hour (see page 7817) property.
	SetHourOffset (see page 7812)	Sets the value of the HourOffset (see page 7818) property.
	SetMillisecond (see page 7812)	Sets the value of the Millisecond (see page 7818) property.
	SetMinute (see page 7813)	Sets the value of the Minute (see page 7819) property.
	SetMinuteOffset (see page 7813)	Sets the value of the MinuteOffset (see page 7819) property.
	SetMonth (see page 7813)	Sets the value of the Month (see page 7820) property.
	SetSecond (see page 7814)	Sets the value of the Second (see page 7820) property.
	SetUseZeroMilliseconds (see page 7814)	Sets the value of the UseZeroMilliseconds (see page 7821) property.
	SetYear (see page 7814)	Sets the value of the Year (see page 7821) property.

Protected Properties

	Name	Description
	Millisecond (see page 7818)	Indicates the Millisecond of the second that the object represents.

Public Methods

	Name	Description
	Create (see page 7805)	Creates and initializes a TXSCustomDateTime (see page 7803) object.
	Destroy (see page 7805)	Frees the memory associated with the TXSCustomDateTime (see page 7803) object.

Public Properties

	Name	Description
	AsDateTime (see page 7815)	Represents the object's value as a TDateTime (see page 3323) value.
	AsUTCDateTime (see page 7816)	Represents the object's value as a TDateTime (see page 3323) value using UTC time.
	Day (see page 7816)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 7817)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7820) property.
	Hour (see page 7817)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7818)	Indicates the difference between the Hour (see page 7817) property and the hour in the local time zone.
	Minute (see page 7819)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7819)	Indicates the difference between the Minute (see page 7819) property and the minute of the hour in local time.
	Month (see page 7820)	Indicates the month of the year that the object represents.
	Second (see page 7820)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7821)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7818) is zero.
	Year (see page 7821)	Indicates the year that the object represents.

86.1.6.2 TXSCustomDateTime.Create Constructor

Creates and initializes a TXSCustomDateTime (see page 7803) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSCustomDateTime();
```

Description

Call Create to create a TXSCustomDateTime (see page 7803) object at runtime.

See Also

Destroy (see page 7805)

86.1.6.3 TXSCustomDateTime.Destroy Destructor

Frees the memory associated with the TXSCustomDateTime (see page 7803) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TXSCustomDateTime();
```

Description

Do not call Destroy to free a TXSCustomDateTime (see page 7803) object. Instead, use the Free method, which checks that the TXSCustomDateTime (see page 7803) reference is not nil before calling Destroy.

On server applications, there is typically no need to explicitly free a TXSCustomDateTime (see page 7803) instance (for example, one created as a return value or output parameter). By default, when a TXSCustomDateTime (see page 7803) descendant is created in a method that was called remotely using an invokable interface, it is added to a data context (the value of the DataContext property). As long as the remotable object belongs to the data context, the data context handles freeing the

object. Similarly, TCustomXSDateTime objects that are passed in as a parameter belong to the data context.

On client applications, TXSCustomDateTime (see page 7803) instances created and passed to an invokable interface or returned as a parameter or result of an invokable interface must be freed by the calling application.

See Also

DataContext

86.1.6.4 TXSCustomDateTime Fields

The fields of the TXSCustomDateTime class are listed here.

Protected Fields

	Name	Description
	FDateTime (see page 7806)	This is FDateTime, a member of class TXSCustomDateTime.

86.1.6.4.1 TXSCustomDateTime.FDateTime Field

Pascal

```
FDateTime: WideString;
```

C++

```
BSTR FDateTime;
```

Description

This is FDateTime, a member of class TXSCustomDateTime.

86.1.6.5 TXSCustomDateTime Methods

The methods of the TXSCustomDateTime class are listed here.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7807)	Returns the value of the AsDateTime (see page 7815) property.
	GetAsUTCDateTime (see page 7807)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 7807)	Returns the value of the Day (see page 7816) property.
	GetFractionalSeconds (see page 7808)	Returns the value of the FractionalSeconds (see page 7817) property.
	GetHour (see page 7808)	Returns the value of the Hour (see page 7817) property.
	GetHourOffset (see page 7808)	Returns the value of the HourOffset (see page 7818) property.
	GetMillisecond (see page 7808)	Returns the value of the Millisecond (see page 7818) property.
	GetMinute (see page 7809)	Returns the value of the Minute (see page 7819) property.
	GetMinuteOffset (see page 7809)	Returns the value of the MinuteOffset (see page 7819) property.
	GetMonth (see page 7809)	Returns the value of the Month (see page 7820) property.
	GetSecond (see page 7810)	Returns the value of the Second (see page 7820) property.
	GetUseZeroMilliseconds (see page 7810)	Returns the value of the UseZeroMilliseconds (see page 7821) property.
	GetYear (see page 7810)	Returns the value of the Year (see page 7821) property.
	SetAsDateTime (see page 7811)	Sets the value of the AsDateTime (see page 7815) property.
	SetAsUTCDateTime (see page 7811)	Sets the value of the AsUTCDateTime (see page 7816) property.
	SetDay (see page 7811)	Sets the value of the Day (see page 7816) property.
	SetFractionalSeconds (see page 7811)	Sets the value of the FractionalSeconds (see page 7817) property.
	SetHour (see page 7812)	Sets the value of the Hour (see page 7817) property.
	SetHourOffset (see page 7812)	Sets the value of the HourOffset (see page 7818) property.

	SetMillisecond (see page 7812)	Sets the value of the Millisecond (see page 7818) property.
	SetMinute (see page 7813)	Sets the value of the Minute (see page 7819) property.
	SetMinuteOffset (see page 7813)	Sets the value of the MinuteOffset (see page 7819) property.
	SetMonth (see page 7813)	Sets the value of the Month (see page 7820) property.
	SetSecond (see page 7814)	Sets the value of the Second (see page 7820) property.
	SetUseZeroMilliseconds (see page 7814)	Sets the value of the UseZeroMilliseconds (see page 7821) property.
	SetYear (see page 7814)	Sets the value of the Year (see page 7821) property.

86.1.6.5.1 TXSCustomDateTime.GetAsDateTime Method

Returns the value of the AsDateTime (see page 7815) property.

Pascal

```
function GetAsDateTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsDateTime();
```

Description

GetAsDateTime is the protected read implementation of the AsDateTime (see page 7815) property.

See Also

AsDateTime (see page 7815)

86.1.6.5.2 TXSCustomDateTime.GetAsUTCDateTime Method

Pascal

```
function GetAsUTCDateTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsUTCDateTime();
```

Description

This is GetAsUTCDateTime, a member of class TXSCustomDateTime.

86.1.6.5.3 TXSCustomDateTime.GetDay Method

Returns the value of the Day (see page 7816) property.

Pascal

```
function GetDay: Word;
```

C++

```
__fastcall Word GetDay();
```

Description

GetDay is the protected read implementation of the Day (see page 7816) property.

See Also

Day (see page 7816)

86.1.6.5.4 TXSCustomDateTime.GetFractionalSeconds Method

Returns the value of the FractionalSeconds (see page 7817) property.

Pascal

```
function GetFractionalSeconds: Double;
```

C++

```
__fastcall Double GetFractionalSeconds();
```

Description

GetFractionalSeconds is the protected read implementation of the FractionalSeconds (see page 7817) property.

See Also

FractionalSeconds (see page 7817)

86.1.6.5.5 TXSCustomDateTime.GetHour Method

Returns the value of the Hour (see page 7817) property.

Pascal

```
function GetHour: Word;
```

C++

```
__fastcall Word GetHour();
```

Description

GetHour is the protected read implementation of the Hour (see page 7817) property.

See Also

Hour (see page 7817)

86.1.6.5.6 TXSCustomDateTime.GetHourOffset Method

Returns the value of the HourOffset (see page 7818) property.

Pascal

```
function GetHourOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetHourOffset();
```

Description

GetHourOffset is the protected read implementation of the HourOffset (see page 7818) property.

See Also

HourOffset (see page 7818)

86.1.6.5.7 TXSCustomDateTime.GetMillisecond Method

Returns the value of the Millisecond (see page 7818) property.

Pascal

```
function GetMillisecond: Word;
```

C++

```
__fastcall Word GetMillisecond();
```

Description

GetMillisecond is the protected read implementation of the Millisecond (see page 7818) property.

See Also

Millisecond (see page 7818)

86.1.6.5.8 TXSCustomDateTime.GetMinute Method

Returns the value of the Minute (see page 7819) property.

Pascal

```
function GetMinute: Word;
```

C++

```
__fastcall Word GetMinute();
```

Description

GetMinute is the protected read implementation of the Minute (see page 7819) property.

See Also

Minute (see page 7819)

86.1.6.5.9 TXSCustomDateTime.GetMinuteOffset Method

Returns the value of the MinuteOffset (see page 7819) property.

Pascal

```
function GetMinuteOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetMinuteOffset();
```

Description

GetMinuteOffset is the protected read implementation of the MinuteOffset (see page 7819) property.

See Also

MinuteOffset (see page 7819)

86.1.6.5.10 TXSCustomDateTime.GetMonth Method

Returns the value of the Month (see page 7820) property.

Pascal

```
function GetMonth: Word;
```

C++

```
__fastcall Word GetMonth();
```

Description

GetMonth is the protected read implementation of the Month (see page 7820) property.

See Also

Month (see page 7820)

86.1.6.5.11 TXSCustomDateTime.GetSecond Method

Returns the value of the Second (see page 7820) property.

Pascal

```
function GetSecond: Word;
```

C++

```
__fastcall Word GetSecond();
```

Description

GetSecond is the protected read implementation of the Second (see page 7820) property.

See Also

Second (see page 7820)

86.1.6.5.12 TXSCustomDateTime.GetUseZeroMilliseconds Method

Returns the value of the UseZeroMilliseconds (see page 7821) property.

Pascal

```
function GetUseZeroMilliseconds: Boolean;
```

C++

```
__fastcall Boolean GetUseZeroMilliseconds();
```

Description

GetUseZeroMilliseconds is the protected read implementation of the UseZeroMilliseconds (see page 7821) property.

See Also

UseZeroMilliseconds (see page 7821)

86.1.6.5.13 TXSCustomDateTime.GetYear Method

Returns the value of the Year (see page 7821) property.

Pascal

```
function GetYear: Integer;
```

C++

```
__fastcall int GetYear();
```

Description

GetYear is the protected read implementation of the Year (see page 7821) property.

See Also

Year (see page 7821)

86.1.6.5.14 TXSCustomDateTime.SetAsDateTime Method

Sets the value of the AsDateTime (see page 7815) property.

Pascal

```
procedure SetAsDateTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsDateTime(const TDateTime Value);
```

Description

SetAsDateTime is the protected write implementation of the AsDateTime (see page 7815) property.

See Also

AsDateTime (see page 7815)

86.1.6.5.15 TXSCustomDateTime.SetAsUTCDateTime Method

Sets the value of the AsUTCDateTime (see page 7816) property.

Pascal

```
procedure SetAsUTCDateTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsUTCDateTime(const TDateTime Value);
```

Description

SetAsUTCDateTime is the protected write implementation of the AsUTCDateTime (see page 7816) property.

See Also

AsUTCDateTime (see page 7816)

86.1.6.5.16 TXSCustomDateTime.SetDay Method

Sets the value of the Day (see page 7816) property.

Pascal

```
procedure SetDay(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetDay(const Word Value);
```

Description

SetDay is the protected write implementation of the Day (see page 7816) property.

See Also

Day (see page 7816)

86.1.6.5.17 TXSCustomDateTime.SetFractionalSeconds Method

Sets the value of the FractionalSeconds (see page 7817) property.

Pascal

```
procedure SetFractionalSeconds(const Value: Double);
```

C++

```
__fastcall SetFractionalSeconds(const Double Value);
```

Description

SetDay (see page 7811) is the protected write implementation of the FractionalSeconds (see page 7817) property.

Value is the fraction of a second. It is a value between 0 and 1.

See Also

FractionalSeconds (see page 7817)

86.1.6.5.18 TXSCustomDateTime.SetHour Method

Sets the value of the Hour (see page 7817) property.

Pascal

```
procedure SetHour(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetHour(const Word Value);
```

Description

SetHour is the protected write implementation of the Hour (see page 7817) property.

See Also

Hour (see page 7817)

86.1.6.5.19 TXSCustomDateTime.SetHourOffset Method

Sets the value of the HourOffset (see page 7818) property.

Pascal

```
procedure SetHourOffset(const Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetHourOffset(const SmallInt Value);
```

Description

SetHourOffset is the protected write implementation of the HourOffset (see page 7818) property.

See Also

HourOffset (see page 7818)

86.1.6.5.20 TXSCustomDateTime.SetMillisecond Method

Sets the value of the Millisecond (see page 7818) property.

Pascal

```
procedure SetMillisecond(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMillisecond(const Word Value);
```

Description

SetMillisecond is the protected write implementation for the Millisecond (see page 7818) property.

See Also

Millisecond (see page 7818)

86.1.6.5.21 TXSCustomDateTime.SetMinute Method

Sets the value of the Minute (see page 7819) property.

Pascal

```
procedure SetMinute(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMinute(const Word Value);
```

Description

SetMinute is the protected write implementation of the Minute (see page 7819) property.

See Also

Minute (see page 7819)

86.1.6.5.22 TXSCustomDateTime.SetMinuteOffset Method

Sets the value of the MinuteOffset (see page 7819) property.

Pascal

```
procedure SetMinuteOffset(const Value: SmallInt); virtual;
```

C++

```
virtual __fastcall SetMinuteOffset(const SmallInt Value);
```

Description

SetMinuteOffest is the protected write implementation of the MinuteOffset (see page 7819) property.

See Also

MinuteOffset (see page 7819)

86.1.6.5.23 TXSCustomDateTime.SetMonth Method

Sets the value of the Month (see page 7820) property.

Pascal

```
procedure SetMonth(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetMonth(const Word Value);
```

Description

SetMonth is the protected write implementation of the Month (see page 7820) property.

See Also

Month (see page 7820)

86.1.6.5.24 TXSCustomDateTime.SetSecond Method

Sets the value of the Second (see page 7820) property.

Pascal

```
procedure SetSecond(const Value: Word); virtual;
```

C++

```
virtual __fastcall SetSecond(const Word Value);
```

Description

SetSecond is the protected write implementation of the Second (see page 7820) property.

See Also

Second (see page 7820)

86.1.6.5.25 TXSCustomDateTime.SetUseZeroMilliseconds Method

Sets the value of the UseZeroMilliseconds (see page 7821) property.

Pascal

```
procedure SetUseZeroMilliseconds(const Value: Boolean);
```

C++

```
__fastcall SetUseZeroMilliseconds(const Boolean Value);
```

Description

SetUseZeroMilliseconds is the protected write implementation of the UseZeroMilliseconds (see page 7821) property.

See Also

UseZeroMilliseconds (see page 7821)

86.1.6.5.26 TXSCustomDateTime.SetYear Method

Sets the value of the Year (see page 7821) property.

Pascal

```
procedure SetYear(const Value: Integer); virtual;
```

C++

```
virtual __fastcall SetYear(const int Value);
```

Description

SetYear is the protected write implementation of the Year (see page 7821) property.

See Also

Year (see page 7821)

86.1.6.6 TXSCustomDateTime Properties

The properties of the TXSCustomDateTime class are listed here.

Protected Properties

	Name	Description
	Millisecond (see page 7818)	Indicates the Millisecond of the second that the object represents.

Public Properties

	Name	Description
	AsDateTime (see page 7815)	Represents the object's value as a TDateTime (see page 3323) value.
	AsUTCDateTime (see page 7816)	Represents the object's value as a TDateTime (see page 3323) value using UTC time.
	Day (see page 7816)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 7817)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7820) property.
	Hour (see page 7817)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7818)	Indicates the difference between the Hour (see page 7817) property and the hour in the local time zone.
	Minute (see page 7819)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7819)	Indicates the difference between the Minute (see page 7819) property and the minute of the hour in local time.
	Month (see page 7820)	Indicates the month of the year that the object represents.
	Second (see page 7820)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7821)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7818) is zero.
	Year (see page 7821)	Indicates the year that the object represents.

86.1.6.6.1 TXSCustomDateTime.AsDateTime Property

Represents the object's value as a TDateTime (see page 3323) value.

Pascal

```
property AsDateTime: TDateTime;
```

C++

```
__property TDateTime AsDateTime;
```

Description

Use AsDateTime to read the value of the TXSCustomDateTime (see page 7803) object into a variable of type TDateTime (see page 3323) or set its value from a TDateTime (see page 3323) value. AsDateTime handles the mapping between a TDateTime (see page 3323) value and the Year (see page 7821), Month (see page 7820), Day (see page 7816), Hour (see page 7817), Minute (see page 7819), and Second (see page 7820) values.

Note: AsDateTime represents the local date-and-time, without using Coordinated Universal Time (see page 4297) (UTC). That is, AsDateTime ignores the HourOffset (see page 7818) and MinuteOffset (see page 7819) properties. To get or set the value taking the HourOffset (see page 7818) and MinuteOffset (see page 7819) properties into account, use the AsUTCDateTime (see page 7816) property instead.

See Also

Day (see page 7816)

Hour (see page 7817)

Minute (see page 7819)

Month (see page 7820)

Second (see page 7820)

Year (see page 7821)

HourOffset (see page 7818)

MinuteOffset (see page 7819)

Millisecond (see page 7818)

AsUTCDateTime (see page 7816)

86.1.6.6.2 TXSCustomDateTime.AsUTCDateTime Property

Represents the object's value as a TDateTime (see page 3323) value using UTC time.

Pascal

```
property AsUTCDateTime: TDateTime;
```

C++

```
__property TDateTime AsUTCDateTime;
```

Description

Use AsUTCDateTime to read the value of the TXSCustomDateTime (see page 7803) object into a variable of type TDateTime (see page 3323) or set its value from a TDateTime (see page 3323) value. AsUTCDateTime handles the mapping between a TDateTime (see page 3323) value and the Year (see page 7821), Month (see page 7820), Day (see page 7816), Hour (see page 7817), Minute (see page 7819), and Second (see page 7820) values. It also makes an adjustment for the HourOffset (see page 7818) and MinuteOffset (see page 7819) properties, so that the value of AsUTCDateTime is a Coordinated Universal Time (see page 4297) (UTC) value rather than a local time value.

Note: To get or set the value as a local time value ignoring the HourOffset (see page 7818) and MinuteOffset (see page 7819) properties, use the AsDateTime (see page 7815) property instead.

See Also

Day (see page 7816)

Hour (see page 7817)

Minute (see page 7819)

Month (see page 7820)

Second (see page 7820)

Year (see page 7821)

HourOffset (see page 7818)

MinuteOffset (see page 7819)

Millisecond (see page 7818)

AsDateTime (see page 7815)

86.1.6.6.3 TXSCustomDateTime.Day Property

Indicates the Day of the month that the object represents.

Pascal

```
property Day: Word;
```

C++

```
__property Word Day;
```

Description

Use Day to get or set the day portion of the date-and-time value. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month (see page 7820) value.

See Also

AsDateTime (see page 7815)

Hour (see page 7817)

Minute (see page 7819)

Month (see page 7820)

Second (see page 7820)

Year (see page 7821)

Millisecond (see page 7818)

86.1.6.6.4 TXSCustomDateTime.FractionalSeconds Property

Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7820) property.

Pascal

```
property FractionalSeconds: Double;
```

C++

```
__property Double FractionalSeconds;
```

Description

Use FractionalSeconds to get or set the fractional seconds of the time value. FractionalSeconds represents the same information as the Millisecond (see page 7818) property, but can provide greater precision.

See Also

AsDateTime (see page 7815)

Minute (see page 7819)

Second (see page 7820)

Millisecond (see page 7818)

UseZeroMilliseconds (see page 7821)

86.1.6.6.5 TXSCustomDateTime.Hour Property

Indicates the hour of the day that the object represents.

Pascal

```
property Hour: Word;
```

C++

```
__property Word Hour;
```

Description

Use Hour to get or set the hour portion of the date-and-time value. Valid Hour values are 0 through 23.

Note: Hour represents the hour using Coordinated Universal Time (see page 4297) (UTC). To convert the hour to local time, you must add the hour offset.

See Also

AsDateTime (see page 7815)

Day (see page 7816)

Minute (see page 7819)

Month (see page 7820)

Second (see page 7820)

Year (see page 7821)

HourOffset (see page 7818)

Millisecond (see page 7818)

86.1.6.6.6 TXSCustomDateTime.HourOffset Property

Indicates the difference between the Hour (see page 7817) property and the hour in the local time zone.

Pascal

```
property HourOffset: SmallInt;
```

C++

```
__property SmallInt HourOffset;
```

Description

The Hour (see page 7817) property represents the hour of the day using Coordinated Universal Time (see page 4297) (UTC). HourOffset is the correction factor you can add to Hour (see page 7817) to obtain the hour of the day in local time.

See Also

Hour (see page 7817)

86.1.6.6.7 TXSCustomDateTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the date-and-time value. Valid Millisecond values are 1 through 59.

Note: Millisecond values are not used by TXSDuration (see page 7836).

Note: You can also use the FractionalSeconds (see page 7817) property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds (see page 7817) returns a fraction of a second.

See Also

AsDateTime (see page 7815)

Day (see page 7816)

Hour (see page 7817)

Minute (see page 7819)

Second (see page 7820)

Year (see page 7821)

Month (see page 7820)

UseZeroMilliseconds (see page 7821)

FractionalSeconds (see page 7817)

86.1.6.6.8 TXSCustomDateTime.Minute Property

Indicates the minute of the hour that the object represents.

Pascal

```
property Minute: Word;
```

C++

```
__property Word Minute;
```

Description

Use Minute to get or set the minute portion of the date-and-time value. Valid Minute values are 0 through 59.

Note: Minute represents the minute using Coordinated Universal Time (see page 4297) (UTC). To convert the minutes to local time, you must add the minute offset.

See Also

AsDateTime (see page 7815)

Day (see page 7816)

Hour (see page 7817)

Month (see page 7820)

Second (see page 7820)

Year (see page 7821)

MinuteOffset (see page 7819)

Millisecond (see page 7818)

86.1.6.6.9 TXSCustomDateTime.MinuteOffset Property

Indicates the difference between the Minute (see page 7819) property and the minute of the hour in local time.

Pascal

```
property MinuteOffset: SmallInt;
```

C++

```
__property SmallInt MinuteOffset;
```

Description

The Minute (see page 7819) property represents the minute of the hour using Coordinated Universal Time (see page 4297) (UTC). MinuteOffset is the correction factor you can add to Minute (see page 7819) to obtain the minute of the hour in local time.

See Also

Minute (see page 7819)

86.1.6.6.10 TXSCustomDateTime.Month Property

Indicates the month of the year that the object represents.

Pascal

```
property Month: Word;
```

C++

```
__property Word Month;
```

Description

Use Month to get or set the month portion of the date-and-time value. Valid Month values are 1 through 12.

See Also

AsDateTime (see page 7815)

Day (see page 7816)

Hour (see page 7817)

Minute (see page 7819)

Second (see page 7820)

Year (see page 7821)

Millisecond (see page 7818)

86.1.6.6.11 TXSCustomDateTime.Second Property

Indicates the second of the minute that the object represents.

Pascal

```
property Second: Word;
```

C++

```
__property Word Second;
```

Description

Use Second to get or set the seconds portion of the date-and-time value. Valid Second values are 0 through 59.

See Also

AsDateTime (see page 7815)

Day (see page 7816)

Hour (see page 7817)

Minute (see page 7819)

Year (see page 7821)

Millisecond (see page 7818)

86.1.6.6.12 TXSCustomDateTime.UseZeroMilliseconds Property

Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7818) is zero.

Pascal

```
property UseZeroMilliseconds: Boolean;
```

C++

```
__property Boolean UseZeroMilliseconds;
```

Description

By default, the NativeToXS method does not include the milliseconds in the string encoding of this object's value when the value of the Millisecond (see page 7818) property is zero. If you set UseZeroMilliseconds, NativeToXS changes its encoding to include the milliseconds even when the value is zero. This is necessary for some Web Services that do not support the full schema defined by W3C.

See Also

Millisecond (see page 7818)

NativeToXS

86.1.6.6.13 TXSCustomDateTime.Year Property

Indicates the year that the object represents.

Pascal

```
property Year: Integer;
```

C++

```
__property int Year;
```

Description

Use Year to get or set the year portion of the date-and-time value. Valid Year values are 1 through 9999.

See Also

AsDateTime (see page 7815)

Day (see page 7816)

Hour (see page 7817)

Minute (see page 7819)

Second (see page 7820)

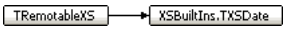
Month (see page 7820)

Millisecond (see page 7818)

86.1.7 TXSDate Class

TXSDate implements the XML schema date type so that it can be used on invokable interfaces.

Class Hierarchy



Pascal

```
TXSDate = class(TRemotableXS);
```

C++

```
class TXSDate : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSDate acts as a wrapper for the simple scalar type date that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDate enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDate value to a TDateTime (see page 3323) value, use the AsDate (see page 7827) property. Other properties give the Year (see page 7828), Month (see page 7827), and Day (see page 7827) that make up that value.

On server applications, instances of TXSDate that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDate instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDate instances that it passes as input parameters, and for freeing any TXSDate instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3504)

TXSDuration (see page 7836)

TXSTime (see page 7849)

86.1.7.1 TXSDate Members

The following tables list the members exposed by TXSDate.

Protected Methods

	Name	Description
	GetAsDate (see page 7824)	This is GetAsDate, a member of class TXSDate.

	GetDay (see page 7824)	This is GetDay, a member of class TXSDate.
	GetMonth (see page 7824)	This is GetMonth, a member of class TXSDate.
	GetYear (see page 7825)	This is GetYear, a member of class TXSDate.
	SetAsDate (see page 7825)	This is SetAsDate, a member of class TXSDate.
	SetDay (see page 7825)	This is SetDay, a member of class TXSDate.
	SetMonth (see page 7826)	This is SetMonth, a member of class TXSDate.
	SetYear (see page 7826)	This is SetYear, a member of class TXSDate.

Public Methods

	Name	Description
	Clone (see page 7824)	Returns a new TXSDate (see page 7822) object that represents the same value as this one.
V	Create (see page 7823)	Creates and initializes a Create object.
V	NativeToXS (see page 7825)	Returns a string that represents the value of a TXSDate (see page 7822) instance.
V	XSToNative (see page 7826)	Assigns the properties of the TXSDate (see page 7822) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDate (see page 7827)	Represents the object's value as a TDateTime (see page 3323) value.
	Day (see page 7827)	Indicates the Day of the month that the object represents.
	Month (see page 7827)	Indicates the month of the year that the object represents.
	Year (see page 7828)	Indicates the year that the object represents.

86.1.7.2 TXSDate.Create Constructor

Creates and initializes a Create object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSDate();
```

Description

Call Create indirectly, using the new keyword, to create a Create object at runtime.

86.1.7.3 TXSDate Methods

The methods of the TXSDate class are listed here.

Protected Methods

	Name	Description
	GetAsDate (see page 7824)	This is GetAsDate, a member of class TXSDate.
	GetDay (see page 7824)	This is GetDay, a member of class TXSDate.
	GetMonth (see page 7824)	This is GetMonth, a member of class TXSDate.
	GetYear (see page 7825)	This is GetYear, a member of class TXSDate.
	SetAsDate (see page 7825)	This is SetAsDate, a member of class TXSDate.
	SetDay (see page 7825)	This is SetDay, a member of class TXSDate.
	SetMonth (see page 7826)	This is SetMonth, a member of class TXSDate.
	SetYear (see page 7826)	This is SetYear, a member of class TXSDate.

Public Methods

	Name	Description
	Clone (see page 7824)	Returns a new TXSDate (see page 7822) object that represents the same value as this one.
V	NativeToXS (see page 7825)	Returns a string that represents the value of a TXSDate (see page 7822) instance.
V	XSToNative (see page 7826)	Assigns the properties of the TXSDate (see page 7822) instance from the string representation of its value.

86.1.7.3.1 TXSDate.Clone Method

Returns a new TXSDate (see page 7822) object that represents the same value as this one.

Pascal

```
function Clone: TXSDate;
```

C++

```
__fastcall TXSDate Clone();
```

Description

Call Clone to obtain a new TXSDate (see page 7822) instance that represents the same time as this one. Clone returns the new TXSDate (see page 7822) object.

86.1.7.3.2 TXSDate.GetAsDate Method

Pascal

```
function GetAsDate: TDateTime;
```

C++

```
__fastcall TDateTime GetAsDate();
```

Description

This is GetAsDate, a member of class TXSDate.

86.1.7.3.3 TXSDate.GetDay Method

Pascal

```
function GetDay: Word;
```

C++

```
__fastcall Word GetDay();
```

Description

This is GetDay, a member of class TXSDate.

86.1.7.3.4 TXSDate.GetMonth Method

Pascal

```
function GetMonth: Word;
```

C++

```
__fastcall Word GetMonth();
```

Description

This is GetMonth, a member of class TXSDate.

86.1.7.3.5 TXSDate.GetYear Method

Pascal

```
function GetYear: Integer;
```

C++

```
__fastcall int GetYear();
```

Description

This is GetYear, a member of class TXSDate.

86.1.7.3.6 TXSDate.NativeToXS Method

Returns a string that represents the value of a TXSDate (see page 7822) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDate (see page 7822) in a SOAP packet.

See Also

XSToNative (see page 7826)

86.1.7.3.7 TXSDate.SetAsDate Method

Pascal

```
procedure SetAsDate(const Value: TDateTime);
```

C++

```
__fastcall SetAsDate(const TDateTime Value);
```

Description

This is SetAsDate, a member of class TXSDate.

86.1.7.3.8 TXSDate.SetDay Method

Pascal

```
procedure SetDay(const Value: Word);
```

C++

```
__fastcall SetDay(const Word Value);
```

Description

This is SetDay, a member of class TXSDate.

86.1.7.3.9 TXSDate.SetMonth Method

Pascal

```
procedure SetMonth(const Value: Word);
```

C++

```
__fastcall SetMonth(const Word Value);
```

Description

This is SetMonth, a member of class TXSDate.

86.1.7.3.10 TXSDate.SetYear Method

Pascal

```
procedure SetYear(const Value: Integer);
```

C++

```
__fastcall SetYear(const int Value);
```

Description

This is SetYear, a member of class TXSDate.

86.1.7.3.11 TXSDate.XSToNative Method

Assigns the properties of the TXSDate (see page 7822) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a date value. It uses this information to set the properties of the TXSDate (see page 7822) object.

See Also

NativeToXS (see page 7825)

86.1.7.4 TXSDate Properties

The properties of the TXSDate class are listed here.

Public Properties

	Name	Description
	AsDate (see page 7827)	Represents the object's value as a TDateTime (see page 3323) value.
	Day (see page 7827)	Indicates the Day of the month that the object represents.
	Month (see page 7827)	Indicates the month of the year that the object represents.
	Year (see page 7828)	Indicates the year that the object represents.

86.1.7.4.1 TXSDate.AsDate Property

Represents the object's value as a TDateTime (see page 3323) value.

Pascal

```
property AsDate: TDateTime;
```

C++

```
__property TDateTime AsDate;
```

Description

Use AsDate to read the value of the TXSDate (see page 7822) object into a variable of type TDateTime (see page 3323) or set this objects value from a TDateTime (see page 3323) value. AsDate handles the mapping between a TDateTime (see page 3323) value and the Year (see page 7828), Month (see page 7827), and Day (see page 7827) properties. When reading AsDate, the resulting TDateTime (see page 3323) value has a time portion of 0. When setting AsDate, the time portion is ignored.

See Also

Day (see page 7827)

Month (see page 7827)

Year (see page 7828)

86.1.7.4.2 TXSDate.Day Property

Indicates the Day of the month that the object represents.

Pascal

```
property Day: Word;
```

C++

```
__property Word Day;
```

Description

Use Day to get or set the day portion of the date-and-time value. Valid Day values are 1 through 28, 29, 30, or 31, depending on the Month (see page 7827) value.

See Also

AsDate (see page 7827)

Month (see page 7827)

Year (see page 7828)

86.1.7.4.3 TXSDate.Month Property

Indicates the month of the year that the object represents.

Pascal

```
property Month: Word;
```

C++

```
__property Word Month;
```

Description

Use Month to get or set the month portion of the date-and-time value. Valid Month values are 1 through 12.

See Also

AsDate (see page 7827)

Day (see page 7827)

Year (see page 7828)

86.1.7.4.4 TXSDate.Year Property

Indicates the year that the object represents.

Pascal

```
property Year: Integer;
```

C++

```
__property int Year;
```

Description

Use Year to get or set the year portion of the date-and-time value. Valid Year values are 1 through 9999.

See Also

AsDate (see page 7827)

Day (see page 7827)

Month (see page 7827)

86.1.8 TXSDateTime Class

TXSDateTime implements the XML schema datetime type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDateTime = class(TXSCustomDateTime);
```

C++

```
class TXSDateTime : public TXSCustomDateTime;
```

File

XSBuiltIns

Description

TXSDateTime acts as a wrapper for the simple scalar type datetime that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDateTime enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDateTime value to a TDateTime (see page 3323) value, use the AsDateTime (see page 7815) property. The resulting value represents the time portion using Coordinated Universal Time (see page 4297) (UTC). To convert that to a

local time zone, adjust the hour and minute portions using the HourOffset (see page 7818) and MinuteOffset (see page 7819) properties.

On server applications, instances of TXSDateTime that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDateTime instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDateTime instances that it passes as input parameters, and for freeing any TXSDateTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3504)

TXSDate (see page 7822)

TXSDuration (see page 7836)

TXSTime (see page 7849)

86.1.8.1 TXSDateTime Members

The following tables list the members exposed by TXSDateTime.

Protected Fields

	Name	Description
	FDateTime (see page 7806)	This is FDateTime, a member of class TXSCustomDateTime.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7807)	Returns the value of the AsDateTime (see page 7815) property.
	GetAsUTCDateTime (see page 7807)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.
	GetDay (see page 7807)	Returns the value of the Day (see page 7816) property.
	GetFractionalSeconds (see page 7808)	Returns the value of the FractionalSeconds (see page 7817) property.
	GetHour (see page 7808)	Returns the value of the Hour (see page 7817) property.
	GetHourOffset (see page 7808)	Returns the value of the HourOffset (see page 7818) property.
	GetMillisecond (see page 7808)	Returns the value of the Millisecond (see page 7818) property.
	GetMinute (see page 7809)	Returns the value of the Minute (see page 7819) property.
	GetMinuteOffset (see page 7809)	Returns the value of the MinuteOffset (see page 7819) property.
	GetMonth (see page 7809)	Returns the value of the Month (see page 7820) property.
	GetSecond (see page 7810)	Returns the value of the Second (see page 7820) property.
	GetUseZeroMilliseconds (see page 7810)	Returns the value of the UseZeroMilliseconds (see page 7821) property.
	GetYear (see page 7810)	Returns the value of the Year (see page 7821) property.
	SetAsDateTime (see page 7811)	Sets the value of the AsDateTime (see page 7815) property.
	SetAsUTCDateTime (see page 7811)	Sets the value of the AsUTCDateTime (see page 7816) property.
	SetDay (see page 7811)	Sets the value of the Day (see page 7816) property.
	SetFractionalSeconds (see page 7811)	Sets the value of the FractionalSeconds (see page 7817) property.
	SetHour (see page 7812)	Sets the value of the Hour (see page 7817) property.
	SetHourOffset (see page 7812)	Sets the value of the HourOffset (see page 7818) property.
	SetMillisecond (see page 7812)	Sets the value of the Millisecond (see page 7818) property.
	SetMinute (see page 7813)	Sets the value of the Minute (see page 7819) property.
	SetMinuteOffset (see page 7813)	Sets the value of the MinuteOffset (see page 7819) property.
	SetMonth (see page 7813)	Sets the value of the Month (see page 7820) property.
	SetSecond (see page 7814)	Sets the value of the Second (see page 7820) property.

	SetUseZeroMilliseconds (see page 7814)	Sets the value of the UseZeroMilliseconds (see page 7821) property.
	SetYear (see page 7814)	Sets the value of the Year (see page 7821) property.

Public Methods

	Name	Description
	Clone (see page 7830)	Returns a new TXSDateTime (see page 7828) object that represents the same value as this one.
	CompareDateTimeParam (see page 7831)	This is CompareDateTimeParam, a member of class TXSDateTime.
	Create (see page 7805)	Creates and initializes a TXSCustomDateTime (see page 7803) object.
	Destroy (see page 7805)	Frees the memory associated with the TXSCustomDateTime (see page 7803) object.
	NativeToXS (see page 7831)	Returns a string that represents the value of a TXSDateTime (see page 7828) instance.
	XSToNative (see page 7831)	Assigns the properties of the TXSDateTime (see page 7828) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDateTime (see page 7815)	Represents the object's value as a TDateTime (see page 3323) value.
	AsUTCDateTime (see page 7816)	Represents the object's value as a TDateTime (see page 3323) value using UTC time.
	Day (see page 7816)	Indicates the Day of the month that the object represents.
	FractionalSeconds (see page 7817)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7820) property.
	Hour (see page 7817)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7818)	Indicates the difference between the Hour (see page 7817) property and the hour in the local time zone.
	Millisecond (see page 7832)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 7819)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7819)	Indicates the difference between the Minute (see page 7819) property and the minute of the hour in local time.
	Month (see page 7820)	Indicates the month of the year that the object represents.
	Second (see page 7820)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7821)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7818) is zero.
	Year (see page 7821)	Indicates the year that the object represents.

86.1.8.2 TXSDateTime Methods

The methods of the TXSDateTime class are listed here.

Public Methods

	Name	Description
	Clone (see page 7830)	Returns a new TXSDateTime (see page 7828) object that represents the same value as this one.
	CompareDateTimeParam (see page 7831)	This is CompareDateTimeParam, a member of class TXSDateTime.
	NativeToXS (see page 7831)	Returns a string that represents the value of a TXSDateTime (see page 7828) instance.
	XSToNative (see page 7831)	Assigns the properties of the TXSDateTime (see page 7828) instance from the string representation of its value.

86.1.8.2.1 TXSDateTime.Clone Method

Returns a new TXSDateTime (see page 7828) object that represents the same value as this one.

Pascal

```
function Clone: TXSDateTime;
```

C++

```
__fastcall TXSDateTime Clone();
```

Description

Call Clone to obtain a new TXSDateTime (see page 7828) instance that represents the same date and time as this one. Clone returns the new TXSDateTime (see page 7828) object.

86.1.8.2.2 TXSDateTime.CompareDateTimeParam Method

Pascal

```
function CompareDateTimeParam(const Value1: TXSDateTime; const Value2: TXSDateTime):
TXSDuration;
```

C++

```
__fastcall TXSDuration CompareDateTimeParam(const TXSDateTime Value1, const TXSDateTime
Value2);
```

Description

This is CompareDateTimeParam, a member of class TXSDateTime.

86.1.8.2.3 TXSDateTime.NativeToXS Method

Returns a string that represents the value of a TXSDateTime (see page 7828) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDateTime (see page 7828) in a SOAP packet.

See Also

XSToNative (see page 7831)

UseZeroMilliseconds (see page 7821)

86.1.8.2.4 TXSDateTime.XSToNative Method

Assigns the properties of the TXSDateTime (see page 7828) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a datetime value. It uses this information to set the properties of the TXSDateTime (see page 7828) object.

See Also

NativeToXS (see page 7831)

86.1.8.3 TXSDateTime Properties

The properties of the TXSDateTime class are listed here.

Public Properties

	Name	Description
	Millisecond (see page 7832)	Indicates the Millisecond of the second that the object represents.

86.1.8.3.1 TXSDateTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the date-and-time value. Valid Millisecond values are 1 through 59.

Note: Millisecond values are not used by TXSDuration.

Note: You can also use the FractionalSeconds property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds returns a fraction of a second.

See Also

AsDateTime (see page 7815)

Day (see page 7816)

Hour (see page 7817)

Minute (see page 7819)

Second (see page 7820)

Year (see page 7821)

Month (see page 7820)

UseZeroMilliseconds (see page 7821)

FractionalSeconds (see page 7817)

86.1.9 TXSDecimal Class

TXSDecimal implements the XML schema decimal type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDecimal = class(TRemotableXS);
```

C++

```
class TXSDecimal : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSDecimal acts as a wrapper for the simple scalar decimal type that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDecimal enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDecimal value to a TBcd value, use the AsBcd (see page 7835) property.

On server applications, instances of TXSDecimal that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDecimal instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDecimal instances that it passes as input parameters, and for freeing any TXSDecimal instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3504)

TXSHexBinary (see page 7840)

EXSDecimalException (see page 7796)

86.1.9.1 TXSDecimal Members

The following tables list the members exposed by TXSDecimal.

Public Methods

	Name	Description
	NativeToXS (see page 7834)	Returns a string that represents the value of a TXSDecimal (see page 7832) instance.
	XSToNative (see page 7834)	Assigns a string as the value of the TXSDecimal (see page 7832) instance.

Public Properties

	Name	Description
	AsBcd (see page 7835)	Represents the object's value as a TBcd value.
	DecimalString (see page 7835)	Represents the object's value as a string.

86.1.9.2 TXSDecimal Methods

The methods of the TXSDecimal class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7834)	Returns a string that represents the value of a TXSDecimal (see page 7832) instance.
	XSToNative (see page 7834)	Assigns a string as the value of the TXSDecimal (see page 7832) instance.

86.1.9.2.1 TXSDecimal.NativeToXS Method

Returns a string that represents the value of a TXSDecimal (see page 7832) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDecimal (see page 7832) object in a SOAP packet. That is, NativeToXS converts the DecimalString (see page 7835) property to a WideString (see page 3467) and returns the result.

See Also

XSToNative (see page 7834)

DecimalString (see page 7835)

86.1.9.2.2 TXSDecimal.XSToNative Method

Assigns a string as the value of the TXSDecimal (see page 7832) instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative sets the value of this TXSDecimal (see page 7832) object to a string that represents a formatted decimal value.

Data is the SOAP encoding of a decimal value. It becomes the value of the DecimalString (see page 7835) property.

See Also

NativeToXS (see page 7834)

DecimalString (see page 7835)

86.1.9.3 TXSDecimal Properties

The properties of the TXSDecimal class are listed here.

Public Properties

	Name	Description
	AsBcd (see page 7835)	Represents the object's value as a TBcd value.
	DecimalString (see page 7835)	Represents the object's value as a string.

86.1.9.3.1 TXSDecimal.AsBcd Property

Represents the object's value as a TBcd value.

Pascal

```
property AsBcd: TBcd;
```

C++

```
__property TBcd AsBcd;
```

Description

Use AsBcd to get or set this object's value as a TBcd type. TXSDecimal (see page 7832) converts between the internal string representation that is the value of the DecimalString (see page 7835) property and a TBcd value when the application gets or sets this property. Thus, if the string representation is ill-formed, reading AsBcd can cause an EXSDecimalException (see page 7796) exception. Similarly, setting AsBcd can cause an EXSDecimalException (see page 7796) exception if the TBcd value has an invalid scale or precision.

See Also

BcdToStr

TryStrToBcd

DecimalString (see page 7835)

EXSDecimalException (see page 7796)

86.1.9.3.2 TXSDecimal.DecimalString Property

Represents the object's value as a string.

Pascal

```
property DecimalString: string;
```

C++

```
__property UnicodeString DecimalString;
```

Description

Use DecimalString to obtain the value of the TXSDecimal (see page 7832) object as a string. This value is the same as the string that appears in XML documents to represent the decimal value. To work with this object's value as a TBcd value, use the AsBcd (see page 7835) property instead.

Note: If you do not need the precision of binary-coded decimal values, you can use global functions such as StrToFloat (see page 4282) and FloatToStr (see page 4180) or StrToCurr (see page 4278) and CurrToStr (see page 4086) to convert this string to a floating-point representation.

See Also

StrToCurr (see page 4278)

CurrToStr (see page 4086)

StrToFloat (see page 4282)

FloatToStr (see page 4180)

AsBcd (see page 7835)

86.1.10 TXSDuration Class

TXSDuration implements the XML schema duration type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSDuration = class(TXSCustomDateTime);
```

C++

```
class TXSDuration : public TXSCustomDateTime;
```

File

XSBuiltIns

Description

TXSDuration acts as a wrapper for the simple scalar type duration that W3C defines for XML schemas. That is, TXSDuration represents the interval between two points in time. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSDuration enables this marshaling because it is a descendant of TRemotable.

To convert a TXSDuration value to a TDateTime (see page 3323) value, use the AsDateTime (see page 7815) property.

On server applications, instances of TXSDuration that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSDuration instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSDuration instances that it passes as input parameters, and for freeing any TXSDuration instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3504)

TXSDate (see page 7822)

TXSTime (see page 7849)

86.1.10.1 TXSDuration Members

The following tables list the members exposed by TXSDuration.

Protected Fields

	Name	Description
	FDateTime (see page 7806)	This is FDateTime, a member of class TXSCustomDateTime.

Protected Methods

	Name	Description
	GetAsDateTime (see page 7807)	Returns the value of the AsDateTime (see page 7815) property.
	GetAsUTCDateTime (see page 7807)	This is GetAsUTCDateTime, a member of class TXSCustomDateTime.

	GetDay (see page 7807)	Returns the value of the Day (see page 7816) property.
	GetDecimalSecond (see page 7838)	This is GetDecimalSecond, a member of class TXSDuration.
	GetFractionalSeconds (see page 7808)	Returns the value of the FractionalSeconds (see page 7817) property.
	GetHour (see page 7808)	Returns the value of the Hour (see page 7817) property.
	GetHourOffset (see page 7808)	Returns the value of the HourOffset (see page 7818) property.
	GetMillisecond (see page 7808)	Returns the value of the Millisecond (see page 7818) property.
	GetMinute (see page 7809)	Returns the value of the Minute (see page 7819) property.
	GetMinuteOffset (see page 7809)	Returns the value of the MinuteOffset (see page 7819) property.
	GetMonth (see page 7809)	Returns the value of the Month (see page 7820) property.
	GetSecond (see page 7810)	Returns the value of the Second (see page 7820) property.
	GetUseZeroMilliseconds (see page 7810)	Returns the value of the UseZeroMilliseconds (see page 7821) property.
	GetYear (see page 7810)	Returns the value of the Year (see page 7821) property.
	SetAsDateTime (see page 7811)	Sets the value of the AsDateTime (see page 7815) property.
	SetAsUTCDateTime (see page 7811)	Sets the value of the AsUTCDateTime (see page 7816) property.
	SetDay (see page 7811)	Sets the value of the Day (see page 7816) property.
	SetDecimalSecond (see page 7839)	This is SetDecimalSecond, a member of class TXSDuration.
	SetFractionalSeconds (see page 7811)	Sets the value of the FractionalSeconds (see page 7817) property.
	SetHour (see page 7812)	Sets the value of the Hour (see page 7817) property.
	SetHourOffset (see page 7812)	Sets the value of the HourOffset (see page 7818) property.
	SetMillisecond (see page 7812)	Sets the value of the Millisecond (see page 7818) property.
	SetMinute (see page 7813)	Sets the value of the Minute (see page 7819) property.
	SetMinuteOffset (see page 7813)	Sets the value of the MinuteOffset (see page 7819) property.
	SetMonth (see page 7813)	Sets the value of the Month (see page 7820) property.
	SetSecond (see page 7814)	Sets the value of the Second (see page 7820) property.
	SetUseZeroMilliseconds (see page 7814)	Sets the value of the UseZeroMilliseconds (see page 7821) property.
	SetYear (see page 7814)	Sets the value of the Year (see page 7821) property.

Protected Properties

	Name	Description
	Millisecond (see page 7818)	Indicates the Millisecond of the second that the object represents.

Public Methods

	Name	Description
	Create (see page 7838)	Creates and initializes a TXSDuration (see page 7836) object.
	Destroy (see page 7805)	Frees the memory associated with the TXSCustomDateTime (see page 7803) object.
	NativeToXS (see page 7838)	Returns a string that represents the value of a TXSDuration (see page 7836) instance.
	XSToNative (see page 7839)	Assigns the properties of the TXSDuration (see page 7836) instance from the string representation of its value.

Public Properties

	Name	Description
	AsDateTime (see page 7815)	Represents the object's value as a TDateTime (see page 3323) value.
	AsUTCDateTime (see page 7816)	Represents the object's value as a TDateTime (see page 3323) value using UTC time.
	Day (see page 7816)	Indicates the Day of the month that the object represents.
	DecimalSecond (see page 7839)	Indicates the number of seconds, including fractional seconds, in the duration.
	FractionalSeconds (see page 7817)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7820) property.
	Hour (see page 7817)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7818)	Indicates the difference between the Hour (see page 7817) property and the hour in the local time zone.
	Minute (see page 7819)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7819)	Indicates the difference between the Minute (see page 7819) property and the minute of the hour in local time.
	Month (see page 7820)	Indicates the month of the year that the object represents.

	Second (see page 7820)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7821)	Indicates whether the string representation of this object's value includes a portion for milliseconds when the value of Millisecond (see page 7818) is zero.
	Year (see page 7821)	Indicates the year that the object represents.

86.1.10.2 TXSDuration.Create Constructor

Creates and initializes a TXSDuration (see page 7836) object.

Pascal

```
constructor Create; override;
```

C++

```
virtual __fastcall TXSDuration();
```

Description

Call Create to create a TXSDuration (see page 7836) object at runtime.

86.1.10.3 TXSDuration Methods

The methods of the TXSDuration class are listed here.

Protected Methods

	Name	Description
	GetDecimalSecond (see page 7838)	This is GetDecimalSecond, a member of class TXSDuration.
	SetDecimalSecond (see page 7839)	This is SetDecimalSecond, a member of class TXSDuration.

Public Methods

	Name	Description
	NativeToXS (see page 7838)	Returns a string that represents the value of a TXSDuration (see page 7836) instance.
	XSToNative (see page 7839)	Assigns the properties of the TXSDuration (see page 7836) instance from the string representation of its value.

86.1.10.3.1 TXSDuration.GetDecimalSecond Method

Pascal

```
function GetDecimalSecond: Double;
```

C++

```
__fastcall Double GetDecimalSecond();
```

Description

This is GetDecimalSecond, a member of class TXSDuration.

86.1.10.3.2 TXSDuration.NativeToXS Method

Returns a string that represents the value of a TXSDuration (see page 7836) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSDuration (see page 7836) in a SOAP packet.

See Also

XSToNative (see page 7839)

86.1.10.3.3 TXSDuration.SetDecimalSecond Method

Pascal

```
procedure SetDecimalSecond(const Value: Double);
```

C++

```
__fastcall SetDecimalSecond(const Double Value);
```

Description

This is SetDecimalSecond, a member of class TXSDuration.

86.1.10.3.4 TXSDuration.XSToNative Method

Assigns the properties of the TXSDuration (see page 7836) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a duration value. It uses this information to set the properties of the TXSDuration (see page 7836) object.

See Also

NativeToXS (see page 7838)

86.1.10.4 TXSDuration Properties

The properties of the TXSDuration class are listed here.

Public Properties

	Name	Description
	DecimalSecond (see page 7839)	Indicates the number of seconds, including fractional seconds, in the duration.

86.1.10.4.1 TXSDuration.DecimalSecond Property

Indicates the number of seconds, including fractional seconds, in the duration.

Pascal

```
property DecimalSecond: Double;
```

C++

```
__property Double DecimalSecond;
```

Description

Use the DecimalSecond property to determine the size of the duration with a granularity finer than the Seconds property provides. DecimalSecond represents the number of seconds in the duration (above and beyond the years, months, days, hours and minutes represented by the other TXSDuration (see page 7836) properties), including any fractional seconds.

See Also

Second (see page 7820)

FractionalSeconds (see page 7817)

86.1.11 TXSHexBinary Class

TXSHexBinary implements the XML schema hexBinary type so that it can be used on invokable interfaces.

Class Hierarchy

Pascal

```
TXSHexBinary = class(TRemotableXS);
```

C++

```
class TXSHexBinary : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSHexBinary acts as a wrapper for the simple scalar hexBinary type that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSHexBinary enables this marshaling because it is a descendant of TRemotable.

Use the AsByteArray (see page 7842) property to get or set the value of TXSHexBinary as an array of bytes. You can also set the value of TXSHexBinary from an integer using the global IntToHex (see page 4215) function to convert an integer into a hexadecimal string and setting the HexBinaryString (see page 7843) property.

On server applications, instances of TXSHexBinary that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSHexBinary instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSHexBinary instances that it passes as input parameters, and for freeing any TXSHexBinary instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3504)

TXSDecimal (see page 7832)

HexToBin

BinToHex

IntToHex (see page 4215)

EXSHexBinaryException (see page 7796)

86.1.11.1 TXSHexBinary Members

The following tables list the members exposed by TXSHexBinary.

Public Methods

	Name	Description
	NativeToXS (see page 7841)	Returns a string that represents the value of a TXSHexBinary (see page 7840) instance.
	XSToNative (see page 7842)	Assigns a string as the value of the TXSHexBinary (see page 7840) instance.

Public Properties

	Name	Description
	AsByteArray (see page 7842)	Represents the object's value as an array of bytes.
	HexBinaryString (see page 7843)	Represents the object's value as a string.

86.1.11.2 TXSHexBinary Methods

The methods of the TXSHexBinary class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7841)	Returns a string that represents the value of a TXSHexBinary (see page 7840) instance.
	XSToNative (see page 7842)	Assigns a string as the value of the TXSHexBinary (see page 7840) instance.

86.1.11.2.1 TXSHexBinary.NativeToXS Method

Returns a string that represents the value of a TXSHexBinary (see page 7840) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSHexBinary (see page 7840) object in a SOAP packet. That is, NativeToXS converts the HexBinaryString (see page 7843) property to a WideString (see page 3467) and returns the result.

See Also

XSToNative (see page 7842)

HexBinaryString (see page 7843)

86.1.11.2.2 TXSHexBinary.XSToNative Method

Assigns a string as the value of the TXSHexBinary (see page 7840) instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative sets the value of this TXSHexBinary (see page 7840) object to a string that represents a formatted hexadecimal value.

Data is the SOAP encoding of a hexadecimal value. It becomes the value of the HexBinaryString (see page 7843) property.

See Also

NativeToXS (see page 7841)

HexBinaryString (see page 7843)

86.1.11.3 TXSHexBinary Properties

The properties of the TXSHexBinary class are listed here.

Public Properties

	Name	Description
	AsByteArray (see page 7842)	Represents the object's value as an array of bytes.
	HexBinaryString (see page 7843)	Represents the object's value as a string.

86.1.11.3.1 TXSHexBinary.AsByteArray Property

Represents the object's value as an array of bytes.

Pascal

```
property AsByteArray: TByteDynArray;
```

C++

```
__property TByteDynArray AsByteArray;
```

Description

Use AsByteArray to obtain the value of the TXSHexBinary (see page 7840) object as an array of bytes. This value is obtained from the HexBinaryString (see page 7843) property using the global HexToBin function. When you set AsByteArray, TXSHexBinary (see page 7840) uses the global BinToHex function to convert the binary value to a string and sets the HexBinaryString (see page 7843) property.

See Also

BinToHex

HexToBin

HexBinaryString (see page 7843)

86.1.11.3.2 TXSHexBinary.HexBinaryString Property

Represents the object's value as a string.

Pascal

```
property HexBinaryString: string;
```

C++

```
__property UnicodeString HexBinaryString;
```

Description

Use HexBinaryString to obtain the value of the TXSHexBinary (see page 7840) object. This value is represented as the string that appears in XML documents to represent the hexadecimal value.

To work with the value as an array of bytes, use the AsByteArray (see page 7842) property instead. You can convert an integer to a string that can be assigned to this property using the global IntToHex (see page 4215) function.

See Also

IntToHex (see page 4215)

AsByteArray (see page 7842)

86.1.12 TXSInteger Class

Class Hierarchy

Pascal

```
TXSInteger = class(TRemotableXS);
```

C++

```
class TXSInteger : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSInteger.

86.1.12.1 TXSInteger Members

The following tables list the members exposed by TXSInteger.

Public Methods

	Name	Description
	NativeToXS (see page 7844)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7844)	Assigns the value of a string to the TRemotableXS instance.

Public Properties

	Name	Description
	AsInteger (see page 7845)	This is AsInteger, a member of class TXSInteger.

86.1.12.2 TXSInteger Methods

The methods of the TXSInteger class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7844)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7844)	Assigns the value of a string to the TRemotableXS instance.

86.1.12.2.1 TXSInteger.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

86.1.12.2.2 TXSInteger.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

NativeToXS

Using Remotable Objects

86.1.12.3 TXSInteger Properties

The properties of the TXSInteger class are listed here.

Public Properties

	Name	Description
	AsInteger (see page 7845)	This is AsInteger, a member of class TXSInteger.

86.1.12.3.1 TXSInteger.AsInteger Property

Pascal

```
property AsInteger: Integer;
```

C++

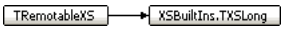
```
__property int AsInteger;
```

Description

This is AsInteger, a member of class TXSInteger.

86.1.13 TXSLong Class

Class Hierarchy



Pascal

```
TXSLong = class(TRemotableXS);
```

C++

```
class TXSLong : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSLong.

86.1.13.1 TXSLong Members

The following tables list the members exposed by TXSLong.

Public Methods

	Name	Description
	NativeToXS (see page 7846)	Returns a string that represents the value of a TRemotableXS instance.

	XSToNative (see page 7846)	Assigns the value of a string to the TRemotableXS instance.

Public Properties

	Name	Description
	AsLong (see page 7847)	This is AsLong, a member of class TXSLong.

86.1.13.2 TXSLong Methods

The methods of the TXSLong class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7846)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7846)	Assigns the value of a string to the TRemotableXS instance.

86.1.13.2.1 TXSLong.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

86.1.13.2.2 TXSLong.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

NativeToXS

Using Remotable Objects

86.1.13.3 TXSLong Properties

The properties of the TXSLong class are listed here.

Public Properties

	Name	Description
	AsLong (see page 7847)	This is AsLong, a member of class TXSLong.

86.1.13.3.1 TXSLong.AsLong Property

Pascal

```
property AsLong: Int64;
```

C++

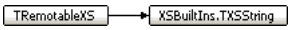
```
__property Int64 AsLong;
```

Description

This is AsLong, a member of class TXSLong.

86.1.14 TXSString Class

Class Hierarchy



Pascal

```
TXSString = class(TRemotableXS);
```

C++

```
class TXSString : public TRemotableXS;
```

File

XSBuiltIns

Description

This is class XSBuiltIns.TXSString.

86.1.14.1 TXSString Members

The following tables list the members exposed by TXSString.

Public Methods

	Name	Description
	NativeToXS (see page 7848)	Returns a string that represents the value of a TRemotableXS instance.

	XSToNative (see page 7848)	Assigns the value of a string to the TRemotableXS instance.

86.1.14.2 TXSString Methods

The methods of the TXSString class are listed here.

Public Methods

	Name	Description
	NativeToXS (see page 7848)	Returns a string that represents the value of a TRemotableXS instance.
	XSToNative (see page 7848)	Assigns the value of a string to the TRemotableXS instance.

86.1.14.2.1 TXSString.NativeToXS Method

Returns a string that represents the value of a TRemotableXS instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

Override NativeToXS when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS returns a string that represents the value of the TRemotableXS instance. This string is used when marshaling the value of the TRemotableXS descendant.

Classes that implement NativeToXS must also override the XSToNative method so that it performs the converse transformation.

See Also

XSToNative

Using Remotable Objects

86.1.14.2.2 TXSString.XSToNative Method

Assigns the value of a string to the TRemotableXS instance.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

Override XSToNative when implementing a TRemotableXS descendant that represents a non-native scalar type. NativeToXS interprets a string that represents the value of the TRemotableXS instance and uses it to make any necessary changes so that the TRemotableXS object reflects that value. It reverses the transformation that the NativeToXS method performs.

Data is a string that represents the value of the TRemotableXS instance.

See Also

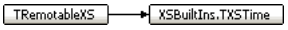
NativeToXS

Using Remotable Objects

86.1.15 TXSTime Class

TXSTime implements the XML schema time type so that it can be used on invokable interfaces.

Class Hierarchy



Pascal

```
TXSTime = class(TRemotableXS);
```

C++

```
class TXSTime : public TRemotableXS;
```

File

XSBuiltIns

Description

TXSTime acts as a wrapper for the simple scalar type time that W3C defines for XML schemas. Because this type does not correspond directly to a Delphi or C++ type, it requires special marshaling when used in an invokable interface. TXSTime enables this marshaling because it is a descendant of TRemotable.

To convert a TXSTime value to a TDateTime (see page 3323) value, use the AsTime (see page 7856) property. Other properties give the Hour (see page 7857), Minute (see page 7859), Second (see page 7859), and Millisecond (see page 7858) that make up that value. These properties represent the time using Coordinated Universal Time (see page 4297) (UTC). To convert that to a local time zone, adjust the hour and minute portions using the HourOffset (see page 7858) and MinuteOffset (see page 7859) properties.

On server applications, instances of TXSTime that are input parameters on an invokable interface are automatically created when the method call is unmarshaled and automatically freed after any output parameters or return value are marshaled so that they can be sent to the client. When a TXSTime instance is created in a method that was called remotely using an invokable interface, the instance is automatically freed after the value of the TRemotable descendant is marshaled for transport back to the client application.

The caller of an invokable interface (client) is responsible for creating any TXSTime instances that it passes as input parameters, and for freeing any TXSTime instances it creates or that are returned as output parameters or method results.

See Also

IInvokable (see page 3504)

TXSDate (see page 7822)

TXSDuration (see page 7836)

TXSDateTime (see page 7828)

86.1.15.1 TXSTime Members

The following tables list the members exposed by TXSTime.

Protected Methods

	Name	Description
	GetAsTime (see page 7851)	This is GetAsTime, a member of class TXSTime.

	GetFractionalSeconds (see page 7851)	This is GetFractionalSeconds, a member of class TXSTime.
	GetFractionalSecondString (see page 7852)	This is GetFractionalSecondString, a member of class TXSTime.
	GetHour (see page 7852)	This is GetHour, a member of class TXSTime.
	GetHourOffset (see page 7852)	This is GetHourOffset, a member of class TXSTime.
	GetMillisecond (see page 7852)	This is GetMillisecond, a member of class TXSTime.
	GetMinute (see page 7853)	This is GetMinute, a member of class TXSTime.
	GetMinuteOffset (see page 7853)	This is GetMinuteOffset, a member of class TXSTime.
	GetSecond (see page 7853)	This is GetSecond, a member of class TXSTime.
	SetAsTime (see page 7854)	This is SetAsTime, a member of class TXSTime.
	SetFractionalSeconds (see page 7854)	This is SetFractionalSeconds, a member of class TXSTime.
	SetHour (see page 7854)	This is SetHour, a member of class TXSTime.
	SetHourOffset (see page 7854)	This is SetHourOffset, a member of class TXSTime.
	SetMillisecond (see page 7854)	This is SetMillisecond, a member of class TXSTime.
	SetMinute (see page 7855)	This is SetMinute, a member of class TXSTime.
	SetMinuteOffset (see page 7855)	This is SetMinuteOffset, a member of class TXSTime.
	SetSecond (see page 7855)	This is SetSecond, a member of class TXSTime.

Protected Properties

	Name	Description
R	FractionalSecondString (see page 7857)	This is FractionalSecondString, a member of class TXSTime.

Public Methods

	Name	Description
	Clone (see page 7851)	Returns a new TXSTime (see page 7849) object that represents the same value as this one.
V	NativeToXS (see page 7853)	Returns a string that represents the value of a TXSTime (see page 7849) instance.
V	XSToNative (see page 7855)	Assigns the properties of the TXSTime (see page 7849) instance from the string representation of its value.

Public Properties

	Name	Description
	AsTime (see page 7856)	Represents the object's value as a TDateTime (see page 3323) value.
	FractionalSeconds (see page 7857)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7859) property.
	Hour (see page 7857)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7858)	Indicates the difference between the Hour (see page 7857) property and the hour in the local time zone.
	Millisecond (see page 7858)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 7859)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7859)	Indicates the difference between the Minute (see page 7859) property and the minute of the hour in local time.
	Second (see page 7859)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7860)	Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 7858) and FractionalSeconds (see page 7857) is zero.

86.1.15.2 TXSTime Methods

The methods of the TXSTime class are listed here.

Protected Methods

	Name	Description
	GetAsTime (see page 7851)	This is GetAsTime, a member of class TXSTime.
	GetFractionalSeconds (see page 7851)	This is GetFractionalSeconds, a member of class TXSTime.
	GetFractionalSecondString (see page 7852)	This is GetFractionalSecondString, a member of class TXSTime.

	GetHour (see page 7852)	This is GetHour, a member of class TXSTime.
	GetHourOffset (see page 7852)	This is GetHourOffset, a member of class TXSTime.
	GetMillisecond (see page 7852)	This is GetMillisecond, a member of class TXSTime.
	GetMinute (see page 7853)	This is GetMinute, a member of class TXSTime.
	GetMinuteOffset (see page 7853)	This is GetMinuteOffset, a member of class TXSTime.
	GetSecond (see page 7853)	This is GetSecond, a member of class TXSTime.
	SetAsTime (see page 7854)	This is SetAsTime, a member of class TXSTime.
	SetFractionalSeconds (see page 7854)	This is SetFractionalSeconds, a member of class TXSTime.
	SetHour (see page 7854)	This is SetHour, a member of class TXSTime.
	SetHourOffset (see page 7854)	This is SetHourOffset, a member of class TXSTime.
	SetMillisecond (see page 7854)	This is SetMillisecond, a member of class TXSTime.
	SetMinute (see page 7855)	This is SetMinute, a member of class TXSTime.
	SetMinuteOffset (see page 7855)	This is SetMinuteOffset, a member of class TXSTime.
	SetSecond (see page 7855)	This is SetSecond, a member of class TXSTime.

Public Methods

	Name	Description
	Clone (see page 7851)	Returns a new TXSTime (see page 7849) object that represents the same value as this one.
V	NativeToXS (see page 7853)	Returns a string that represents the value of a TXSTime (see page 7849) instance.
V	XSToNative (see page 7855)	Assigns the properties of the TXSTime (see page 7849) instance from the string representation of its value.

86.1.15.2.1 TXSTime.Clone Method

Returns a new TXSTime (see page 7849) object that represents the same value as this one.

Pascal

```
function Clone: TXSTime;
```

C++

```
__fastcall TXSTime Clone();
```

Description

Call Clone to obtain a new TXSTime (see page 7849) instance that represents the same time as this one. Clone returns the new TXSTime (see page 7849) object.

86.1.15.2.2 TXSTime.GetAsTime Method

Pascal

```
function GetAsTime: TDateTime;
```

C++

```
__fastcall TDateTime GetAsTime();
```

Description

This is GetAsTime, a member of class TXSTime.

86.1.15.2.3 TXSTime.GetFractionalSeconds Method

Pascal

```
function GetFractionalSeconds: Double;
```

C++

```
__fastcall Double GetFractionalSeconds();
```

Description

This is GetFractionalSeconds, a member of class TXSTime.

86.1.15.2.4 TXSTime.GetFractionalSecondString Method

Pascal

```
function GetFractionalSecondString: string;
```

C++

```
__fastcall UnicodeString GetFractionalSecondString();
```

Description

This is GetFractionalSecondString, a member of class TXSTime.

86.1.15.2.5 TXSTime.GetHour Method

Pascal

```
function GetHour: Word;
```

C++

```
__fastcall Word GetHour();
```

Description

This is GetHour, a member of class TXSTime.

86.1.15.2.6 TXSTime.GetHourOffset Method

Pascal

```
function GetHourOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetHourOffset();
```

Description

This is GetHourOffset, a member of class TXSTime.

86.1.15.2.7 TXSTime.GetMillisecond Method

Pascal

```
function GetMillisecond: Word;
```

C++

```
__fastcall Word GetMillisecond();
```

Description

This is GetMillisecond, a member of class TXSTime.

86.1.15.2.8 TXSTime.GetMinute Method

Pascal

```
function GetMinute: Word;
```

C++

```
__fastcall Word GetMinute();
```

Description

This is GetMinute, a member of class TXSTime.

86.1.15.2.9 TXSTime.GetMinuteOffset Method

Pascal

```
function GetMinuteOffset: SmallInt;
```

C++

```
__fastcall SmallInt GetMinuteOffset();
```

Description

This is GetMinuteOffset, a member of class TXSTime.

86.1.15.2.10 TXSTime.GetSecond Method

Pascal

```
function GetSecond: Word;
```

C++

```
__fastcall Word GetSecond();
```

Description

This is GetSecond, a member of class TXSTime.

86.1.15.2.11 TXSTime.NativeToXS Method

Returns a string that represents the value of a TXSTime (see page 7849) instance.

Pascal

```
function NativeToXS: InvString; override;
```

C++

```
virtual __fastcall InvString NativeToXS();
```

Description

NativeToXS returns the string that represents the value of this TXSTime (see page 7849) object in a SOAP packet.

See Also

XSToNative (see page 7855)

UseZeroMilliseconds (see page 7860)

86.1.15.2.12 TXSTime.SetAsTime Method

Pascal

```
procedure SetAsTime(const Value: TDateTime);
```

C++

```
__fastcall SetAsTime(const TDateTime Value);
```

Description

This is SetAsTime, a member of class TXSTime.

86.1.15.2.13 TXSTime.SetFractionalSeconds Method

Pascal

```
procedure SetFractionalSeconds(const Value: Double);
```

C++

```
__fastcall SetFractionalSeconds(const Double Value);
```

Description

This is SetFractionalSeconds, a member of class TXSTime.

86.1.15.2.14 TXSTime.SetHour Method

Pascal

```
procedure SetHour(const Value: Word);
```

C++

```
__fastcall SetHour(const Word Value);
```

Description

This is SetHour, a member of class TXSTime.

86.1.15.2.15 TXSTime.SetHourOffset Method

Pascal

```
procedure SetHourOffset(const Value: SmallInt);
```

C++

```
__fastcall SetHourOffset(const SmallInt Value);
```

Description

This is SetHourOffset, a member of class TXSTime.

86.1.15.2.16 TXSTime.SetMillisecond Method

Pascal

```
procedure SetMillisecond(const Value: Word);
```

C++

```
__fastcall SetMillisecond(const Word Value);
```

Description

This is SetMillisecond, a member of class TXSTime.

86.1.15.2.17 TXSTime.SetMinute Method

Pascal

```
procedure SetMinute(const Value: Word);
```

C++

```
__fastcall SetMinute(const Word Value);
```

Description

This is SetMinute, a member of class TXSTime.

86.1.15.2.18 TXSTime.SetMinuteOffset Method

Pascal

```
procedure SetMinuteOffset(const Value: SmallInt);
```

C++

```
__fastcall SetMinuteOffset(const SmallInt Value);
```

Description

This is SetMinuteOffset, a member of class TXSTime.

86.1.15.2.19 TXSTime.SetSecond Method

Pascal

```
procedure SetSecond(const Value: Word);
```

C++

```
__fastcall SetSecond(const Word Value);
```

Description

This is SetSecond, a member of class TXSTime.

86.1.15.2.20 TXSTime.XSToNative Method

Assigns the properties of the TXSTime (see page 7849) instance from the string representation of its value.

Pascal

```
procedure XSToNative(Value: InvString); override;
```

C++

```
virtual __fastcall XSToNative(InvString Value);
```

Description

XSToNative parses the string specified by Data, which is the SOAP encoding of a time value. It uses this information to set the properties of the TXSTime (see page 7849) object.

See Also

NativeToXS (see page 7853)

86.1.15.3 TXSTime Properties

The properties of the TXSTime class are listed here.

Protected Properties

	Name	Description
	FractionalSecondString (see page 7857)	This is FractionalSecondString, a member of class TXSTime.

Public Properties

	Name	Description
	AsTime (see page 7856)	Represents the object's value as a TDateTime (see page 3323) value.
	FractionalSeconds (see page 7857)	Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7859) property.
	Hour (see page 7857)	Indicates the hour of the day that the object represents.
	HourOffset (see page 7858)	Indicates the difference between the Hour (see page 7857) property and the hour in the local time zone.
	Millisecond (see page 7858)	Indicates the Millisecond of the second that the object represents.
	Minute (see page 7859)	Indicates the minute of the hour that the object represents.
	MinuteOffset (see page 7859)	Indicates the difference between the Minute (see page 7859) property and the minute of the hour in local time.
	Second (see page 7859)	Indicates the second of the minute that the object represents.
	UseZeroMilliseconds (see page 7860)	Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 7858) and FractionalSeconds (see page 7857) is zero.

86.1.15.3.1 TXSTime.AsTime Property

Represents the object's value as a TDateTime (see page 3323) value.

Pascal

```
property AsTime: TDateTime;
```

C++

```
__property TDateTime AsTime;
```

Description

Use AsTime to read the value of the TXSTime (see page 7849) object into a variable of type TDateTime (see page 3323) or set its value from a TDateTime (see page 3323) value. AsTime handles the mapping between a TDateTime (see page 3323) value and the Hour (see page 7857), Minute (see page 7859), and Second (see page 7859) properties. When reading AsTime, the resulting TDateTime (see page 3323) value is given a data portion of 0. When setting AsTime, the data portion is ignored.

Note: AsTime represents the time using Coordinated Universal Time (see page 4297) (UTC). To convert this to a local time, add the offsets specified by HourOffset (see page 7858) and MinuteOffset (see page 7859) to the hours and minutes.

See Also

Hour (see page 7857)

Minute (see page 7859)

Second (see page 7859)

HourOffset (see page 7858)

MinuteOffset (see page 7859)

86.1.15.3.2 TXSTime.FractionalSeconds Property

Indicates any factional portion of a second beyond the seconds specified in the Second (see page 7859) property.

Pascal

```
property FractionalSeconds: Double;
```

C++

```
__property Double FractionalSeconds;
```

Description

Use FractionalSeconds to get or set the fractional seconds of the time value. FractionalSeconds represents the same information as the Millisecond (see page 7858) property, but can provide greater precision.

See Also

AsTime (see page 7856)

Minute (see page 7859)

Second (see page 7859)

Millisecond (see page 7858)

UseZeroMilliseconds (see page 7860)

86.1.15.3.3 TXSTime.FractionalSecondString Property

Pascal

```
property FractionalSecondString: string;
```

C++

```
__property UnicodeString FractionalSecondString;
```

Description

This is FractionalSecondString, a member of class TXSTime.

86.1.15.3.4 TXSTime.Hour Property

Indicates the hour of the day that the object represents.

Pascal

```
property Hour: Word;
```

C++

```
__property Word Hour;
```

Description

Use Hour to get or set the hour portion of the time value. Valid Hour values are 0 through 23.

Note: Hour represents the hour using Coordinated Universal Time (see page 4297) (UTC). To convert the hour to local time, you must add the value of the HourOffset (see page 7858) property.

See Also

AsTime (see page 7856)

Minute (see page 7859)

Second (see page 7859)

Millisecond (see page 7858)

HourOffset (see page 7858)

86.1.15.3.5 TXSTime.HourOffset Property

Indicates the difference between the Hour (see page 7857) property and the hour in the local time zone.

Pascal

```
property HourOffset: SmallInt;
```

C++

```
__property SmallInt HourOffset;
```

Description

The Hour (see page 7857) property represents the hour of the day using Coordinated Universal Time (see page 4297) (UTC). HourOffset is the correction factor you can add to Hour (see page 7857) to obtain the hour of the day in local time.

See Also

Hour (see page 7857)

86.1.15.3.6 TXSTime.Millisecond Property

Indicates the Millisecond of the second that the object represents.

Pascal

```
property Millisecond: Word;
```

C++

```
__property Word Millisecond;
```

Description

Use Millisecond to get or set the millisecond portion of the time value. Valid Millisecond values are 1 through 59.

Note: You can also use the FractionalSeconds (see page 7857) property to obtain fractional portions of the seconds. Unlike Millisecond, which gives the value in milliseconds, FractionalSeconds (see page 7857) returns a fraction of a second.

See Also

AsTime (see page 7856)

Hour (see page 7857)

Minute (see page 7859)

Second (see page 7859)

UseZeroMilliseconds (see page 7860)

FractionalSeconds (see page 7857)

86.1.15.3.7 TXSTime.Minute Property

Indicates the minute of the hour that the object represents.

Pascal

```
property Minute: Word;
```

C++

```
__property Word Minute;
```

Description

Use Minute to get or set the minute portion of the time value. Valid Minute values are 0 through 59.

Note: Minute represents the minute using Coordinated Universal Time (see page 4297) (UTC). To convert the minutes to local time, you must add the value of the MinuteOffset (see page 7859) property.

See Also

AsTime (see page 7856)

Hour (see page 7857)

Second (see page 7859)

Millisecond (see page 7858)

MinuteOffset (see page 7859)

86.1.15.3.8 TXSTime.MinuteOffset Property

Indicates the difference between the Minute (see page 7859) property and the minute of the hour in local time.

Pascal

```
property MinuteOffset: SmallInt;
```

C++

```
__property SmallInt MinuteOffset;
```

Description

The Minute (see page 7859) property represents the minute of the hour using Coordinated Universal Time (see page 4297) (UTC). MinuteOffset is the correction factor you can add to Minute (see page 7859) to obtain the minute of the hour in local time.

See Also

Minute (see page 7859)

86.1.15.3.9 TXSTime.Second Property

Indicates the second of the minute that the object represents.

Pascal

```
property Second: Word;
```

C++

```
__property Word Second;
```

Description

Use Second to get or set the seconds portion of the time value. Valid Second values are 0 through 59.

See Also

AsTime (see page 7856)

Millisecond (see page 7858)

Hour (see page 7857)

Minute (see page 7859)

86.1.15.3.10 TXSTime.UseZeroMilliseconds Property

Indicates whether the string representation of this object's value includes a portion for fractions of a second when the value of Millisecond (see page 7858) and FractionalSeconds (see page 7857) is zero.

Pascal

```
property UseZeroMilliseconds: Boolean;
```

C++

```
__property Boolean UseZeroMilliseconds;
```

Description

By default, the NativeToXS (see page 7853) method does not include the fractional portion of a second in the string encoding of this object's value when the value represents an exact number of seconds. If you set UseZeroMilliseconds, NativeToXS (see page 7853) changes its encoding to include milliseconds even when the value is zero. This is necessary for some Web Services that do not support the full time schema defined by W3C.

See Also

Millisecond (see page 7858)

NativeToXS (see page 7853)

FractionalSeconds (see page 7857)

86.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	DateTimeToXMLTime (see page 7861)	Returns the XML string that represents a TDateTime (see page 3323) value.
	DateTimeToXSDateTime (see page 7861)	Returns a TXSDateTime (see page 7828) object that encodes a TDateTime (see page 3323) value.
	GetDataFromFile (see page 7862)	This is function XSBuiltIns.GetDataFromFile.
	InitXSTypes (see page 7862)	This is function XSBuiltIns.InitXSTypes.
	SoapFloatToStr (see page 7863)	This is function XSBuiltIns.SoapFloatToStr.
	SoapStrToFloat (see page 7863)	This is function XSBuiltIns.SoapStrToFloat.
	XMLTimeToDateTime (see page 7863)	Returns a TDateTime (see page 3323) value to represent an XML datetime value.

86.2.1 XSBuiltIns.DateTimeToXMLTime Function

Returns the XML string that represents a TDateTime (see page 3323) value.

Pascal

```
function DateTimeToXMLTime(Value: TDateTime; ApplyLocalBias: Boolean = True): WideString;
```

C++

```
BSTR DateTimeToXMLTime(TDateTime Value, Boolean ApplyLocalBias = True);
```

File

XSBuiltIns

Description

DateTimeToXMLTime returns the XML representation of a specified TDateTime (see page 3323) value.

Value is the TDateTime (see page 3323) value to convert. If the time zone information is important, this value is assumed to be given in Coordinated Universal Time (see page 4297) (UTC).

CalcLocalBias indicates whether the XML string should include offsets for converting Value to the local time.

See Also

DateTimeToXSDateTime (see page 7861)

XMLTimeToDateTime (see page 7863)

86.2.2 XSBuiltIns.DateTimeToXSDateTime Function

Returns a TXSDateTime (see page 7828) object that encodes a TDateTime (see page 3323) value.

Pascal

```
function DateTimeToXSDateTime(const Value: TDateTime; ApplyLocalBias: Boolean = False):
TXSDateTime;
```

C++

```
TXSDateTime DateTimeToXSDateTime(const TDateTime Value, Boolean ApplyLocalBias = False);
```

File

XSBuiltIns

Description

DateTimeToXSDateTime returns a TXSDateTime (see page 7828) object that corresponds to a specified TDateTime (see page 3323) value. The TXSDateTime (see page 7828) object can be used to represent a date-and-time value on an invokable interface. As a remotable object, it can encoded in SOAP packets.

Value is the TDateTime (see page 3323) value to convert. If the time zone information is important, this value is assumed to be given in Coordinated Universal Time (see page 4297) (UTC).

CalcLocalBias indicates whether the TXSDateTime (see page 7828) object should include offsets for converting Value to the local time zone. If CalcLocalBias is true, DateTimeToXSDateTime looks up the local time zone and uses it to set the HourOffset and MinuteOffset properties of the return value.

See Also

IInvokable (see page 3504)

TXSCustomDateTime (see page 7803)

86.2.3 XSBuiltIns.GetDataFromFile Function

Pascal

```
function GetDataFromFile(AFileName: string): string;
```

C++

```
UnicodeString GetDataFromFile(UnicodeString AFileName);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.GetDataFromFile.

86.2.4 XSBuiltIns.InitXSTypes Function

Pascal

```
procedure InitXSTypes;
```

C++

```
InitXSTypes();
```

File

XSBuiltIns

Description

This is function XSBuiltIns.InitXSTypes.

86.2.5 XSBuiltIns.SoapFloatToStr Function

Pascal

```
function SoapFloatToStr(Value: double): string;
```

C++

```
UnicodeString SoapFloatToStr(double Value);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.SoapFloatToStr.

86.2.6 XSBuiltIns.SoapStrToFloat Function

Pascal

```
function SoapStrToFloat(Value: string): double;
```

C++

```
double SoapStrToFloat(UnicodeString Value);
```

File

XSBuiltIns

Description

This is function XSBuiltIns.SoapStrToFloat.

86.2.7 XSBuiltIns.XMLTimeToDateTime Function

Returns a TDateTime (see page 3323) value to represent an XML datetime value.

Pascal

```
function XMLTimeToDateTime(const XMLDateTime: WideString; AsUTCTime: Boolean = False):
TDateTime;
```

C++

```
TDateTime XMLTimeToDateTime(const BSTR XMLDateTime, Boolean AsUTCTime = False);
```

File

XSBuiltIns

Description

XMLTimeToDateTime returns a TDateTime (see page 3323) value for an XML date-and time value. This function is the inverse to the DateTimeToXMLTime (see page 7861) function.

Value is the XML string to convert.

See Also

DateTimeToXSDateTime (see page 7861)

DateTimeToXMLTime (see page 7861)

86.3 Types

The following table lists types in this documentation.

Types

Name	Description
TXSTimeInstant (see page 7865)	TXSTimeInstant represents the timeInstant type for Web Services applications.

86.3.1 XSBuiltIns.TXSTimeInstant Type

TXSTimeInstant represents the timeInstant type for Web Services applications.

Pascal

```
TXSTimeInstant = TXSDateTime;
```

C++

```
TXSDateTime TXSTimeInstant;
```

File

XSBuiltIns

Description

TXSTimeInstant is a remotable class for representing the timeInstant type that W3C defines for XML schemas. It maps to the TXSDateTime (see page 7828) class, which represents W3C's datetime type. The timeInstant type is replaced by the datetime type but some Web Service applications still use the older type definition.

See Also

RemClassRegistry

TRemotable

86.4 Constants

The following table lists constants in this documentation.

Constants

Name	Description
IncrementAmount (see page 7866)	This is constant XSBuiltIns.IncrementAmount.
MaxDay (see page 7867)	This is constant XSBuiltIns.MaxDay.
MaxMonth (see page 7867)	This is constant XSBuiltIns.MaxMonth.
MinDay (see page 7867)	This is constant XSBuiltIns.MinDay.
MinMonth (see page 7868)	This is constant XSBuiltIns.MinMonth.
SDecimal (see page 7868)	This is constant XSBuiltIns.SDecimal.
SHexMarker (see page 7868)	This is constant XSBuiltIns.SHexMarker.
SLocalTimeMarker (see page 7869)	This is constant XSBuiltIns.SLocalTimeMarker.
SNAN (see page 7869)	This is constant XSBuiltIns.SNAN.
SNegative (see page 7869)	This is constant XSBuiltIns.SNegative.
SoapDecimalSeparator (see page 7870)	This is constant XSBuiltIns.SoapDecimalSeparator.
SoapTimePrefix (see page 7870)	This is constant XSBuiltIns.SoapTimePrefix.
SPlus (see page 7870)	This is constant XSBuiltIns.SPlus.
SSciNotationMarker (see page 7871)	This is constant XSBuiltIns.SSciNotationMarker.
XMLDateSeparator (see page 7871)	This is constant XSBuiltIns.XMLDateSeparator.
XMLDayPos (see page 7871)	This is constant XSBuiltIns.XMLDayPos.
XMLDefaultYearDigits (see page 7872)	This is constant XSBuiltIns.XMLDefaultYearDigits.
XMLDurationDay (see page 7872)	This is constant XSBuiltIns.XMLDurationDay.
XMLDurationHour (see page 7872)	This is constant XSBuiltIns.XMLDurationHour.
XMLDurationMinute (see page 7873)	This is constant XSBuiltIns.XMLDurationMinute.
XMLDurationMonth (see page 7873)	This is constant XSBuiltIns.XMLDurationMonth.
XMLDurationSecond (see page 7873)	This is constant XSBuiltIns.XMLDurationSecond.
XMLDurationStart (see page 7874)	This is constant XSBuiltIns.XMLDurationStart.
XMLDurationYear (see page 7874)	This is constant XSBuiltIns.XMLDurationYear.
XMLHourOffsetMinusMarker (see page 7874)	This is constant XSBuiltIns.XMLHourOffsetMinusMarker.
XMLHourOffsetPlusMarker (see page 7875)	This is constant XSBuiltIns.XMLHourOffsetPlusMarker.
XMLMilSecPos (see page 7875)	This is constant XSBuiltIns.XMLMilSecPos.
XMLMonthPos (see page 7875)	This is constant XSBuiltIns.XMLMonthPos.
XMLTimeSeparator (see page 7876)	This is constant XSBuiltIns.XMLTimeSeparator.
XMLYearPos (see page 7876)	This is constant XSBuiltIns.XMLYearPos.

86.4.1 XSBuiltIns.IncrementAmount Constant

Pascal

```
IncrementAmount: array[Boolean] of Integer = (1, -1);
```

C++

```
array[Boolean] of Integer IncrementAmount = (1, -1);
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.IncrementAmount.

86.4.2 XSBuiltIns.MaxDay Constant

Pascal

```
MaxDay = 31;
```

C++

```
MaxDay = 31;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MaxDay.

86.4.3 XSBuiltIns.MaxMonth Constant

Pascal

```
MaxMonth = 12;
```

C++

```
MaxMonth = 12;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MaxMonth.

86.4.4 XSBuiltIns.MinDay Constant

Pascal

```
MinDay = 1;
```

C++

```
MinDay = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MinDay.

86.4.5 XSBuiltIns.MinMonth Constant

Pascal

```
MinMonth = 1;
```

C++

```
MinMonth = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.MinMonth.

86.4.6 XSBuiltIns.SDecimal Constant

Pascal

```
SDecimal: Char = '.';
```

C++

```
Char SDecimal = '.';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SDecimal.

86.4.7 XSBuiltIns.SHexMarker Constant

Pascal

```
SHexMarker = '$';
```

C++

```
SHexMarker = '$';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SHexMarker.

86.4.8 XSBuiltIns.SLocalTimeMarker Constant

Pascal

```
SLocalTimeMarker = 'Z';
```

C++

```
SLocalTimeMarker = 'Z';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SLocalTimeMarker.

86.4.9 XSBuiltIns.SNAN Constant

Pascal

```
SNAN = 'NAN';
```

C++

```
SNAN = 'NAN';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SNAN.

86.4.10 XSBuiltIns.SNegative Constant

Pascal

```
SNegative: Char = '-';
```

C++

```
Char SNegative = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SNegative.

86.4.11 XSBuiltIns.SoapDecimalSeparator Constant

Pascal

```
SoapDecimalSeparator = '.';
```

C++

```
SoapDecimalSeparator = '.';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SoapDecimalSeparator.

86.4.12 XSBuiltIns.SoapTimePrefix Constant

Pascal

```
SoapTimePrefix: InvString = 'T';
```

C++

```
InvString SoapTimePrefix = 'T';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SoapTimePrefix.

86.4.13 XSBuiltIns.SPlus Constant

Pascal

```
SPlus: Char = '+';
```

C++

```
Char SPlus = '+';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SPlus.

86.4.14 XSBuiltIns.SSciNotationMarker Constant

Pascal

```
SSciNotationMarker: Char = 'E';
```

C++

```
Char SSciNotationMarker = 'E';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.SSciNotationMarker.

86.4.15 XSBuiltIns.XMLDateSeparator Constant

Pascal

```
XMLDateSeparator = '-';
```

C++

```
XMLDateSeparator = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDateSeparator.

86.4.16 XSBuiltIns.XMLDayPos Constant

Pascal

```
XMLDayPos = 9;
```

C++

```
XMLDayPos = 9;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDayPos.

86.4.17 XSBuiltIns.XMLDefaultYearDigits Constant

Pascal

```
XMLDefaultYearDigits = 4;
```

C++

```
XMLDefaultYearDigits = 4;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDefaultYearDigits.

86.4.18 XSBuiltIns.XMLDurationDay Constant

Pascal

```
XMLDurationDay = 'D';
```

C++

```
XMLDurationDay = 'D';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationDay.

86.4.19 XSBuiltIns.XMLDurationHour Constant

Pascal

```
XMLDurationHour = 'H';
```

C++

```
XMLDurationHour = 'H';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationHour.

86.4.20 XSBuiltIns.XMLDurationMinute Constant

Pascal

```
XMLDurationMinute = 'M';
```

C++

```
XMLDurationMinute = 'M';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationMinute.

86.4.21 XSBuiltIns.XMLDurationMonth Constant

Pascal

```
XMLDurationMonth = 'M';
```

C++

```
XMLDurationMonth = 'M';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationMonth.

86.4.22 XSBuiltIns.XMLDurationSecond Constant

Pascal

```
XMLDurationSecond = 'S';
```

C++

```
XMLDurationSecond = 'S';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationSecond.

86.4.23 XSBuiltIns.XMLDurationStart Constant

Pascal

```
XMLDurationStart = 'P';
```

C++

```
XMLDurationStart = 'P';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationStart.

86.4.24 XSBuiltIns.XMLDurationYear Constant

Pascal

```
XMLDurationYear = 'Y';
```

C++

```
XMLDurationYear = 'Y';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLDurationYear.

86.4.25 XSBuiltIns.XMLHourOffsetMinusMarker Constant

Pascal

```
XMLHourOffsetMinusMarker = '-';
```

C++

```
XMLHourOffsetMinusMarker = '-';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLHourOffsetMinusMarker.

86.4.26 XSBuiltIns.XMLHourOffsetPlusMarker Constant

Pascal

```
XMLHourOffsetPlusMarker = '+';
```

C++

```
XMLHourOffsetPlusMarker = '+';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLHourOffsetPlusMarker.

86.4.27 XSBuiltIns.XMLMilSecPos Constant

Pascal

```
XMLMilSecPos = 10;
```

C++

```
XMLMilSecPos = 10;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLMilSecPos.

86.4.28 XSBuiltIns.XMLMonthPos Constant

Pascal

```
XMLMonthPos = 6;
```

C++

```
XMLMonthPos = 6;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLMonthPos.

86.4.29 XSBuiltIns.XMLTimeSeparator Constant

Pascal

```
XMLTimeSeparator = ':';
```

C++

```
XMLTimeSeparator = ':';
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLTimeSeparator.

86.4.30 XSBuiltIns.XMLYearPos Constant

Pascal

```
XMLYearPos = 1;
```

C++

```
XMLYearPos = 1;
```

File

XSBuiltIns

Description

This is constant XSBuiltIns.XMLYearPos.

87 XSLProd Namespace

This is namespace XSLProd.

Topics

Name	Description
Classes (see page 7878)	The following table lists classes in this documentation.

Classes

	Name	Description
	EXSLPageProducerError (see page 7878)	EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 7893) encounters a problem.
	TCustomXSLPageProducer (see page 7878)	TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.
	TXSLPageProducer (see page 7893)	TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

87.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EXSLPageProducerError (see page 7878)	EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 7893) encounters a problem.
	TCustomXSLPageProducer (see page 7878)	TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.
	TXSLPageProducer (see page 7893)	TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

87.1.1 EXSLPageProducerError Class

EXSLPageProducerError is the exception that is raised when TXSLPageProducer (see page 7893) encounters a problem.

Class Hierarchy

Pascal

```
EXSLPageProducerError = class(EWebBrokerException);
```

C++

```
class EXSLPageProducerError : public EWebBrokerException;
```

File

XSLProd

Description

TXSLPageProducer (see page 7893) raises an EXSLPageProducerError exception when it tries to produce content and the WebSnap application has not yet specified the source XML document or the XSL template that transforms that source into the final content.

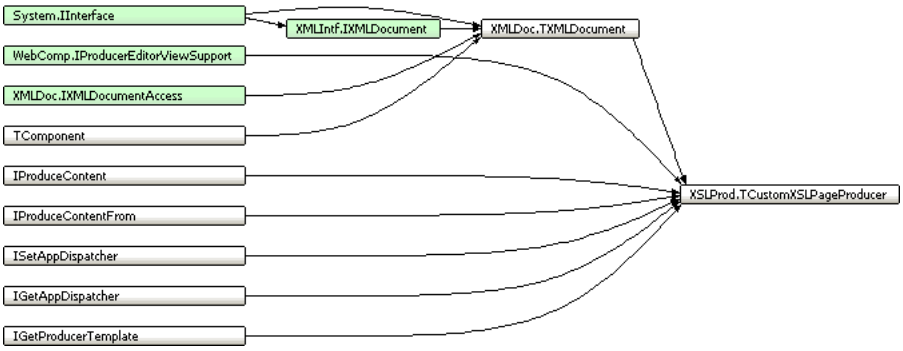
See Also

TXSLPageProducer (see page 7893)

87.1.2 TCustomXSLPageProducer Class

TCustomXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

Class Hierarchy



Pascal

```
TCustomXSLPageProducer = class(TXMLDocument, IProduceContent, IProduceContentFrom,
ISetAppDispatcher, IGetAppDispatcher, IProducerEditorViewSupport, IGetProducerTemplate);
```

C++

```
class TCustomXSLPageProducer : public TXMLDocument, public IProduceContent, public
IProduceContentFrom, public ISetAppDispatcher, public IGetAppDispatcher, public
IProducerEditorViewSupport, public IGetProducerTemplate;
```

File

XSLProd

Description

TCustomXSLPageProducer is the base class for components that transform the data in an XML document using an XSL template. This class introduces all the properties and methods necessary for specifying the XML document to transform, the XSL template that describes the transformation, and for performing the transformation.

The XMLData (see page 7892) property identifies the XML data that this component transforms. XMLData (see page 7892) must be set to a component that supports either the IXMLDocument (see page 7640) interface (for example, TXMLDocument (see page 7347) or one of its descendants) or the IGetXMLStream (see page 6083) interface (for example, TXMLBroker).

To obtain the results of the transformation, you can use any of the following methods:

The Content (see page 7885) method performs the transformation using the XSL template that is specified by the FileName or XML property. When using the Content (see page 7885) method, you can also read an XSL document by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

The ContentFromString (see page 7886) and ContentFromWideString (see page 7886) methods perform the transformation using an XSL template that is supplied as a parameter.

The ContentFromStream (see page 7885) method performs the transformation using an XSL template that it reads from a specified stream.

87.1.2.1 TCustomXSLPageProducer Members

The following tables list the members exposed by TCustomXSLPageProducer.

Protected Methods

	Name	Description
	_AddRef (see page 7354)	Called when an application uses a component interface.

	_Release (see page 7354)	Called when an application releases a component interface.
	AssignParseOptions (see page 7356)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7356)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7356)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7356)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7358)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7359)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7359)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7359)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7360)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7360)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7360)	Returns the value of the Active property.
	GetAppDispatcher (see page 7887)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetAsyncLoadState (see page 7361)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7361)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7361)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7362)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7362)	Returns the value of the Node property.
	GetDocumentObject (see page 7363)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7363)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7363)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7363)	Returns the value of the Encoding property.
	GetFileName (see page 7364)	Returns the value of the FileName property.
	GetModified (see page 7364)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7364)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7365)	Returns the value of the Options property.
	GetParseOptions (see page 7365)	Returns the value of the ParseOptions property.
	GetProducerTemplateFileName (see page 7887)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 7887)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.
	GetPrologNode (see page 7365)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7366)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7366)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7366)	Returns the value of the StandAlone property.
	GetTemplateFileType (see page 7887)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 7888)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetVersion (see page 7366)	Returns the value of the Version property.
	GetXML (see page 7367)	Returns the value of the XML property.
	GetXMLData (see page 7888)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 7888)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.
	GetXSLData (see page 7888)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 7888)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 7889)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 7889)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 7889)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	InternalSetPrologValue (see page 7367)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7368)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7368)	Initializes the component after the form file has been read into memory.
V	Notification (see page 7889)	Forwards notification messages to all owned components.
	ProduceContent (see page 7890)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 7890)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.
	ProduceContentFromString (see page 7890)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.

	ReadDOMVendor (see page 7371)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7372)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7374)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7375)	Saves the XML (see page 7392) document to an internal string list.
V	SetActive (see page 7375)	Sets the value of the Active property.
	SetAppDispatcher (see page 7891)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.
	SetDocumentElement (see page 7375)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7376)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7376)	Sets the value of the Encoding property.
	SetFileName (see page 7376)	Sets the value of the FileName property.
	SetModified (see page 7377)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7377)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7377)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7378)	Sets the value of the Options property.
	SetParseOptions (see page 7378)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7378)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7378)	Sets the value of the StandAlone property.
	SetVersion (see page 7379)	Sets the value of the Version property.
	SetXML (see page 7379)	Sets the value of the XML property.
	SetXMLStrings (see page 7379)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7380)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7380)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7383)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7383)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7385)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7386)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7391)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7354)	This is the overview for the AddChild method overload.
	AddChild (see page 7643)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7355)	Responds after the last constructor has executed.
V	Content (see page 7885)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property.
V	ContentFromStream (see page 7885)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 7886)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.
V	ContentFromWideString (see page 7886)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.
	Create (see page 7351)	This is the overview for the Create constructor overload.
	CreateElement (see page 7357)	Creates a new element node that is associated with this document.
	CreateElement (see page 7644)	Creates a new element node that is associated with this document.
	CreateNode (see page 7357)	Creates a new node that is associated with this document.
	CreateNode (see page 7645)	Creates a new node that is associated with this document.
V	Destroy (see page 7352)	Disposes of a TXMLDocument (see page 7347) object.
	GeneratePrefix (see page 7360)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 7646)	Generates a unique namespace prefix that can be used for a node in the document.

	GetActive (see page 7646)	Returns the value of the Active (see page 7661) property.
	GetAsyncLoadState (see page 7647)	Returns the value of the AsyncLoadState (see page 7662) property.
	GetChildNodes (see page 7647)	Returns the value of the ChildNodes (see page 7663) property.
	GetDocBinding (see page 7362)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7647)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7648)	Returns the value of the DocumentElement (see page 7663) property.
	GetDocumentNode (see page 7648)	Returns the value of the Node (see page 7665) property.
	GetDocumentObject (see page 7479)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7648)	Returns the value of the DOMDocument (see page 7663) property.
	GetDOMPersist (see page 7479)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7649)	Returns the value of the Encoding (see page 7664) property.
	GetFileName (see page 7649)	Returns the value of the FileName (see page 7664) property.
	GetModified (see page 7649)	Returns the value of the Modified (see page 7665) property.
	GetNodeIndentStr (see page 7650)	Returns the value of the NodeIndentStr (see page 7666) property.
	GetOptions (see page 7650)	Returns the value of the Options (see page 7666) property.
	GetParseOptions (see page 7650)	Returns the value of the ParseOptions (see page 7667) property.
	GetSchemaRef (see page 7650)	Returns the value of the SchemaRef (see page 7667) property.
	GetStandAlone (see page 7651)	Returns the value of the StandAlone (see page 7668) property.
	GetVersion (see page 7651)	Returns the value of the Version (see page 7668) property.
	GetXML (see page 7651)	Returns the value of the XML (see page 7668) property.
	IsEmptyDoc (see page 7367)	Indicates whether the TXMLDocument (see page 7347) instance represents an empty document.
	IsEmptyDoc (see page 7652)	Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.
	LoadFromFile (see page 7368)	Loads an XML (see page 7392) document and activates it.
	LoadFromFile (see page 7652)	Loads an XML (see page 7668) document from disk.
	LoadFromStream (see page 7369)	Loads an XML (see page 7392) document from a stream and activates it.
	LoadFromStream (see page 7653)	Loads an XML (see page 7668) document from a stream and activates it.
	LoadFromXML (see page 7370)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
	NewInstance (see page 7370)	Allocates memory for an instance of an object type and returns a pointer to that new instance.
	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7371)	Updates the parsed XML (see page 7392) document to reflect external changes.
	Refresh (see page 7654)	Updates the parsed XML (see page 7668) document to reflect external changes.
	RegisterDocBinding (see page 7372)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7654)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7372)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7655)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7373)	Saves the XML (see page 7392) document to disk.
	SaveToFile (see page 7655)	Saves the XML (see page 7668) document to disk.
	SaveToStream (see page 7373)	Saves the XML (see page 7392) document to a stream.
	SaveToStream (see page 7656)	Saves the XML (see page 7668) document to a stream.
	SaveToXML (see page 7374)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7657)	Sets the value of the Active (see page 7661) property.
	SetDocumentElement (see page 7657)	Sets the value of the DocumentElement (see page 7663) property.
	SetDOMDocument (see page 7658)	Sets the value of the DOMDocument (see page 7663) property.
	SetEncoding (see page 7658)	Sets the value of the Encoding (see page 7664) property.
	SetFileName (see page 7658)	Sets the value of the FileName (see page 7664) property.
	SetNodeIndentStr (see page 7659)	Sets the value of the NodeIndentStr (see page 7666) property.

	SetOnAsyncLoad (see page 7659)	Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.
	SetOptions (see page 7659)	Sets the value of the Options (see page 7666) property.
	SetParseOptions (see page 7660)	Sets the value of the ParseOptions (see page 7667) property.
	SetStandAlone (see page 7660)	Sets the value of the StandAlone (see page 7668) property.
	SetVersion (see page 7660)	Sets the value of the Version (see page 7668) property.
	SetXML (see page 7661)	Sets the value of the XML (see page 7668) property.

Public Properties

	Name	Description
	Active (see page 7661)	Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7382)	Indicates the state of the DOM parser when it parses the XML (see page 7392) document asynchronously.
R	AsyncLoadState (see page 7662)	Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.
R	ChildNodes (see page 7383)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7663)	Lists all of the child nodes of the document.
R	Dispatcher (see page 7891)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
R	DispatcherComponent (see page 7892)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DocumentElement (see page 7383)	Provides access to the root node of the XML (see page 7392) document.
	DocumentElement (see page 7663)	Provides access to the root node of the XML (see page 7668) document.
R	DocumentObject (see page 7479)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7384)	Provides access to the DOM interface for the parsed XML (see page 7392) document.
	DOMDocument (see page 7663)	Provides access to the DOM interface for the parsed XML (see page 7668) document.
	DOMImplementation (see page 7385)	Provides access to the DOM interface for document-independent functions.
R	DOMPersist (see page 7480)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7386)	Indicates the character set that the XML (see page 7392) document uses to encode characters.
	Encoding (see page 7664)	Indicates the character set that the XML (see page 7668) document uses to encode characters.
	FileName (see page 7664)	Indicates the XML (see page 7668) document file, if any, that this interface represents.
R	Modified (see page 7387)	Indicates whether the XML (see page 7392) document has been modified.
R	Modified (see page 7665)	Indicates whether the XML (see page 7668) document has been modified.
R	Node (see page 7388)	Provides access to the document node for the XML (see page 7392) document.
R	Node (see page 7665)	Provides access to the document node for the XML (see page 7668) document.
	NodeIndentStr (see page 7666)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.
	NSPrefixBase (see page 7389)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7666)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7667)	Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.
R	SchemaRef (see page 7391)	Returns the name of the schema associated with this XML (see page 7392) document.
R	SchemaRef (see page 7667)	Returns the name of the schema associated with this XML (see page 7668) document.
	StandAlone (see page 7391)	Indicates whether the XML (see page 7392) document includes any external markup declarations.
	StandAlone (see page 7668)	Indicates whether the XML (see page 7668) document includes any external markup declarations.
	Version (see page 7391)	Indicates the XML (see page 7392) version that the document uses.
	Version (see page 7668)	Indicates the XML (see page 7668) version that the document uses.

	XML (see page 7668)	Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.
	XMLData (see page 7892)	Specifies the component that supplies the XML data that this component transforms.

Published Events

	Name	Description
	AfterClose (see page 7393)	Occurs after the XML (see page 7392) document is closed.
	AfterNodeChange (see page 7393)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7394)	Occurs immediately after the XML (see page 7392) document is opened.
	BeforeClose (see page 7394)	Occurs immediately before the XML (see page 7392) document is closed.
	BeforeNodeChange (see page 7394)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7395)	Occurs immediately before the XML (see page 7392) document is opened.
	OnAsyncLoad (see page 7395)	Occurs at various stages during the asynchronous parsing of an XML (see page 7392) document.

Published Properties

	Name	Description
	Active (see page 7381)	Specifies whether the XML (see page 7392) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7386)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7392) document.
	FileName (see page 7387)	Specifies the file name for the XML (see page 7392) document associated with this TXMLDocument (see page 7347) object.
	NodeIndentStr (see page 7388)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7392) text.
	Options (see page 7389)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7390)	Specifies implementation-specific options that are followed when parsing the XML (see page 7392) document.
	XML (see page 7392)	Specifies the content of the XML document associated with this TXMLDocument (see page 7347) object.

87.1.2.2 TCustomXSLPageProducer Methods

The methods of the TCustomXSLPageProducer class are listed here.

Protected Methods

	Name	Description
	GetAppDispatcher (see page 7887)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetProducerTemplateFileName (see page 7887)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 7887)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.
	GetTemplateFileType (see page 7887)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 7888)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetXMLData (see page 7888)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 7888)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.
	GetXSLData (see page 7888)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 7888)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 7889)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 7889)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 7889)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	Notification (see page 7889)	Forwards notification messages to all owned components.
	ProduceContent (see page 7890)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 7890)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.

	ProduceContentFromString (see page 7890)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.
	SetAppDispatcher (see page 7891)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.

Public Methods

	Name	Description
	Content (see page 7885)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property.
	ContentFromStream (see page 7885)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 7886)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.
	ContentFromWideString (see page 7886)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.

87.1.2.2.1 TCustomXSLPageProducer.Content Method

Returns the transformed value of the XML supplied by the XMLData (see page 7892) property.

Pascal

```
function Content: string; virtual;
```

C++

```
virtual __fastcall UnicodeString Content();
```

Description

Call Content to retrieve the content for using in an HTTP response message. Content transforms the XML data that is supplied by the XMLData (see page 7892) property using the XSL template associated with this component. The XSL template can be specified in several ways:

If the FileName property is set, it specifies a file that contains the XSL template.

If FileName is not set, you can assign an XSL template using the XML property.

At runtime, you can supply an XSL template by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

See Also

ContentFromStream (see page 7885)

ContentFromString (see page 7886)

ContentFromWideString (see page 7886)

XMLData (see page 7892)

FileName (see page 7387)

XML (see page 7392)

LoadFromFile (see page 7368)

LoadFromStream (see page 7369)

LoadFromXML (see page 7370)

87.1.2.2.2 TCustomXSLPageProducer.ContentFromStream Method

Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using an XSL template that can be read from a specified stream.

Pascal

```
function ContentFromStream(InStream: TStream): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ContentFromStream(TStream InStream);
```

Description

Call Content (see page 7885) to retrieve the content for using in an HTTP response message. Content (see page 7885) transforms the XML data that is supplied by the XMLData (see page 7892) property using the XSL template that is reads from InStream.

See Also

Content (see page 7885)

ContentFromString (see page 7886)

ContentFromWideString (see page 7886)

XMLData (see page 7892)

87.1.2.2.3 TCustomXSLPageProducer.ContentFromString Method

Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.

Pascal

```
function ContentFromString(const S: string): string;
```

C++

```
__fastcall UnicodeString ContentFromString(const UnicodeString S);
```

Description

Call Content (see page 7885) to retrieve the content for using in an HTTP response message. Content (see page 7885) transforms the XML data that is supplied by the XMLData (see page 7892) property using the XSL template specified by S.

See Also

Content (see page 7885)

ContentFromStream (see page 7885)

ContentFromWideString (see page 7886)

XMLData (see page 7892)

87.1.2.2.4 TCustomXSLPageProducer.ContentFromWideString Method

Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.

Pascal

```
function ContentFromWideString(const S: WideString): string; virtual;
```

C++

```
virtual __fastcall UnicodeString ContentFromWideString(const BSTR S);
```

Description

Call Content (see page 7885) to retrieve the content for using in an HTTP response message. Content (see page 7885) transforms the XML data that is supplied by the XMLData (see page 7892) property using the XSL template specified by S.

See Also

Content (see page 7885)

ContentFromStream (see page 7885)

ContentFromString (see page 7886)

XMLData (see page 7892)

87.1.2.2.5 TCustomXSLPageProducer.GetAppDispatcher Method

Pascal

```
function GetAppDispatcher: TComponent;
```

C++

```
__fastcall TComponent * GetAppDispatcher();
```

Description

This is GetAppDispatcher, a member of class TCustomXSLPageProducer.

87.1.2.2.6 TCustomXSLPageProducer.GetProducerTemplateFileName Method

Pascal

```
function GetProducerTemplateFileName: string;
```

C++

```
__fastcall UnicodeString GetProducerTemplateFileName();
```

Description

This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.

87.1.2.2.7 TCustomXSLPageProducer.GetProducerTemplateStream Method

Pascal

```
function GetProducerTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetProducerTemplateStream(Boolean AOwned);
```

Description

This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.

87.1.2.2.8 TCustomXSLPageProducer.GetTemplateFileType Method

Pascal

```
function GetTemplateFileType: string;
```

C++

```
__fastcall UnicodeString GetTemplateFileType();
```

Description

This is GetTemplateFileType, a member of class TCustomXSLPageProducer.

87.1.2.2.9 TCustomXSLPageProducer.GetTemplateStream Method

Pascal

```
function GetTemplateStream(out AOwned: Boolean): TStream;
```

C++

```
__fastcall TStream GetTemplateStream(Boolean AOwned);
```

Description

This is GetTemplateStream, a member of class TCustomXSLPageProducer.

87.1.2.2.10 TCustomXSLPageProducer.GetXMLData Method

Pascal

```
function GetXMLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXMLData(Boolean Owned);
```

Description

This is GetXMLData, a member of class TCustomXSLPageProducer.

87.1.2.2.11 TCustomXSLPageProducer.GetXMLDocument Method

Pascal

```
function GetXMLDocument(out AXMLDocument: IXMLDocument): Boolean;
```

C++

```
__fastcall Boolean GetXMLDocument(IXMLDocument AXMLDocument);
```

Description

This is GetXMLDocument, a member of class TCustomXSLPageProducer.

87.1.2.2.12 TCustomXSLPageProducer.GetXSLData Method

Pascal

```
function GetXSLData(var Owned: Boolean): TStream;
```

C++

```
__fastcall TStream GetXSLData(Boolean Owned);
```

Description

This is GetXSLData, a member of class TCustomXSLPageProducer.

87.1.2.2.13 TCustomXSLPageProducer.HasHTMLBrowserView Method

Pascal

```
function HasHTMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasHTMLBrowserView();
```

Description

This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.

87.1.2.2.14 TCustomXSLPageProducer.HasScriptView Method

Pascal

```
function HasScriptView: Boolean;
```

C++

```
__fastcall Boolean HasScriptView();
```

Description

This is HasScriptView, a member of class TCustomXSLPageProducer.

87.1.2.2.15 TCustomXSLPageProducer.HasXMLBrowserView Method

Pascal

```
function HasXMLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXMLBrowserView();
```

Description

This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.

87.1.2.2.16 TCustomXSLPageProducer.HasXSLBrowserView Method

Pascal

```
function HasXSLBrowserView: Boolean;
```

C++

```
__fastcall Boolean HasXSLBrowserView();
```

Description

This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.

87.1.2.2.17 TCustomXSLPageProducer.Notification Method

Forwards notification messages to all owned components.

Pascal

```
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
```

C++

```
virtual __fastcall Notification(TComponent * AComponent, TOperation Operation);
```

Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component

has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

Note: Notification is not called for components that are freed implicitly (because their Owner is freed).

See Also

FreeNotification

InsertComponent

RemoveComponent

Owner

87.1.2.2.18 TCustomXSLPageProducer.ProduceContent Method

Returns the content that the producer creates.

Pascal

```
function ProduceContent: string;
```

C++

```
__fastcall UnicodeString ProduceContent();
```

Description

Call ProduceContent to retrieve the content for using in an HTTP response message. In most cases, this content is a string of HTML commands defining a Web page, although it can be any valid MIME type.

See Also

Content

87.1.2.2.19 TCustomXSLPageProducer.ProduceContentFromStream Method

Pascal

```
function ProduceContentFromStream(Stream: TStream): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromStream(TStream Stream);
```

Description

This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.

87.1.2.2.20 TCustomXSLPageProducer.ProduceContentFromString Method

Pascal

```
function ProduceContentFromString(const S: string): string;
```

C++

```
__fastcall UnicodeString ProduceContentFromString(const UnicodeString S);
```

Description

This is ProduceContentFromString, a member of class TCustomXSLPageProducer.

87.1.2.2.21 TCustomXSLPageProducer.SetAppDispatcher Method

Pascal

```
procedure SetAppDispatcher(const ADispatcher: TComponent);
```

C++

```
__fastcall SetAppDispatcher(const TComponent * ADispatcher);
```

Description

This is SetAppDispatcher, a member of class TCustomXSLPageProducer.

87.1.2.3 TCustomXSLPageProducer Properties

The properties of the TCustomXSLPageProducer class are listed here.

Public Properties

	Name	Description
	Dispatcher (see page 7891)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DispatcherComponent (see page 7892)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	XMLData (see page 7892)	Specifies the component that supplies the XML data that this component transforms.

87.1.2.3.1 TCustomXSLPageProducer.Dispatcher Property

Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.

Pascal

```
property Dispatcher: IWebDispatcherAccess;
```

C++

```
__property IWebDispatcherAccess Dispatcher;
```

Description

Read (see page 3634) Dispatcher to access the interface of the dispatcher. In Web Broker applications, this is the interface of a TWebDispatcher or TWebModule component. In WebSnap applications, this is the interface of a TWebAppComponents component.

The IWebDispatcherAccess interface has two methods:

Request, which returns the HTTP request message to which this page producer responds.

Response, which returns the HTTP response message for which this page producer generates content.

See Also

TWebModule

TWebDispatcher

TWebAppComponents (see page 6243)

TWebRequest

TWebResponse

DispatcherComponent (see page 7892)

87.1.2.3.2 TCustomXSLPageProducer.DispatcherComponent Property

Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.

Pascal

```
property DispatcherComponent: TComponent;
```

C++

```
__property TComponent * DispatcherComponent;
```

Description

Read (see page 3634) DispatcherComponent to access the dispatcher that handles the HTTP request and response messages for this Web Server application. In Web Broker applications, this is the interface of a TWebDispatcherComponent or TWebModule component. In WebSnap applications, this is the interface of a TWebAppComponents component.

Dispatcher (see page 7891) implements the interface that is the value of the Dispatcher (see page 7891) property. The XSL page producer uses this property so that it can respond to notifications when the component is freed. To use the dispatcher, use the Dispatcher (see page 7891) property instead.

See Also

TWebModule

TWebDispatcher

TWebAppComponents (see page 6243)

TWebRequest

TWebResponse

Dispatcher (see page 7891)

87.1.2.3.3 TCustomXSLPageProducer.XMLData Property

Specifies the component that supplies the XML data that this component transforms.

Pascal

```
property XMLData: TComponent;
```

C++

```
__property TComponent * XMLData;
```

Description

Use the XMLData property to specify the XML data that this component transforms. XMLData must be set to a component that supports either the IXMLDocument (see page 7640) interface (for example, TXMLDocument (see page 7347) or one of its descendants) or the IGetXMLStream (see page 6083) interface (for example, TXMLBroker).

See Also

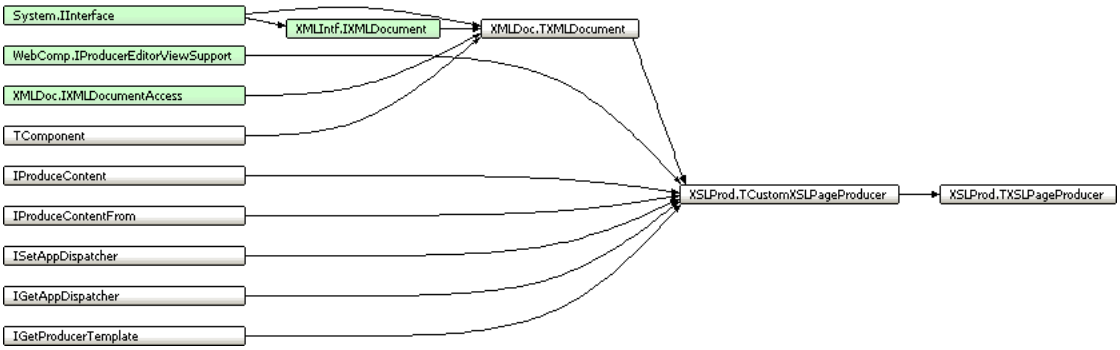
TXMLDocument (see page 7347)

TXMLBroker (see page 7303)

87.1.3 TXSLPageProducer Class

TXSLPageProducer generates web page content by transforming data described with XML (Extensible Markup Language) using an XSL (Extensible Stylesheet Language) template.

Class Hierarchy



Pascal

```
TXSLPageProducer = class(TCustomXSLPageProducer);
```

C++

```
class TXSLPageProducer : public TCustomXSLPageProducer;
```

File

XSLProd

Description

Use the Web Page Module wizard to create a web page that uses TXSLPageProducer. To produce content, TXSLPageProducer uses an XSL template to transform the data in an XML document. Typically, the XSL transforms the XML document into another XML document that is HTML compliant (that is, into a document whose tags are actually HTML tags). However, as specified by the XSL template, TXSLPageProducer can return any arbitrary output as a result of the transformation.

Use the XMLData (see page 7899) property to specify the XML data that this component transforms. XMLData (see page 7899) must be set to a component that supports either the IXMLDocument (see page 7640) interface (for example, TXMLDocument (see page 7347) or one of its descendants) or the IGetXMLStream (see page 6083) interface (for example, TXMLBroker).

To obtain the results of the transformation, you can use any of the following methods:

The Content (see page 7885) method performs the transformation using the XSL template that is specified by the FileName or XML property. When using the Content (see page 7885) method, you can also read an XSL document by calling the LoadFromFile, LoadFromStream, or LoadFromXML method first.

The ContentFromString (see page 7886) and ContentFromWideString (see page 7886) methods perform the transformation using an XSL template that is supplied as a parameter.

The ContentFromStream (see page 7885) method performs the transformation using an XSL template that it reads from a specified stream.

At design time, Web page modules that use the TXMLPageProducer have editor views called XML Tree and XSL Tree. These views provide a read only tree of the XML and XSL respectively.

See Also

TXMLDocument (see page 7347)

TXMLBroker (see page 7303)

87.1.3.1 TXSLPageProducer Members

The following tables list the members exposed by TXSLPageProducer.

Protected Methods

	Name	Description
	_AddRef (see page 7354)	Called when an application uses a component interface.
	_Release (see page 7354)	Called when an application releases a component interface.
	AssignParseOptions (see page 7356)	This is AssignParseOptions, a member of class TXMLDocument.
	CheckActive (see page 7356)	This is CheckActive, a member of class TXMLDocument.
	CheckAutoSave (see page 7356)	This is CheckAutoSave, a member of class TXMLDocument.
	CheckDOM (see page 7356)	This is CheckDOM, a member of class TXMLDocument.
V	DefineProperties (see page 7358)	Designates methods for storing an object's unpublished data on a stream such as a form file.
	DoAfterClose (see page 7359)	This is DoAfterClose, a member of class TXMLDocument.
	DoAfterOpen (see page 7359)	This is DoAfterOpen, a member of class TXMLDocument.
	DoBeforeClose (see page 7359)	This is DoBeforeClose, a member of class TXMLDocument.
	DoBeforeOpen (see page 7360)	This is DoBeforeOpen, a member of class TXMLDocument.
	DoNodeChange (see page 7360)	This is DoNodeChange, a member of class TXMLDocument.
V	GetActive (see page 7360)	Returns the value of the Active property.
	GetAppDispatcher (see page 7887)	This is GetAppDispatcher, a member of class TCustomXSLPageProducer.
	GetAsyncLoadState (see page 7361)	Returns the value of the AsyncLoadState property.
V	GetChildNodeClass (see page 7361)	This is GetChildNodeClass, a member of class TXMLDocument.
	GetChildNodes (see page 7361)	Returns the value of the ChildNodes property.
	GetDocumentElement (see page 7362)	Returns the value of the DocumentElement property.
	GetDocumentNode (see page 7362)	Returns the value of the Node property.
	GetDocumentObject (see page 7363)	This is GetDocumentObject, a member of class TXMLDocument.
	GetDOMDocument (see page 7363)	Returns the value of the DOMDocument property.
	GetDOMPersist (see page 7363)	This is GetDOMPersist, a member of class TXMLDocument.
	GetEncoding (see page 7363)	Returns the value of the Encoding property.
	GetFileName (see page 7364)	Returns the value of the FileName property.
	GetModified (see page 7364)	Returns the value of the Modified property.
	GetNodeIndentStr (see page 7364)	Returns the value of the NodeIndentStr property.
	GetOptions (see page 7365)	Returns the value of the Options property.
	GetParseOptions (see page 7365)	Returns the value of the ParseOptions property.
	GetProducerTemplateFileName (see page 7887)	This is GetProducerTemplateFileName, a member of class TCustomXSLPageProducer.
	GetProducerTemplateStream (see page 7887)	This is GetProducerTemplateStream, a member of class TCustomXSLPageProducer.
	GetPrologNode (see page 7365)	This is GetPrologNode, a member of class TXMLDocument.
	GetPrologValue (see page 7366)	This is GetPrologValue, a member of class TXMLDocument.
	GetSchemaRef (see page 7366)	Returns the value of the SchemaRef property.
	GetStandAlone (see page 7366)	Returns the value of the StandAlone property.
	GetTemplateFileType (see page 7887)	This is GetTemplateFileType, a member of class TCustomXSLPageProducer.
	GetTemplateStream (see page 7888)	This is GetTemplateStream, a member of class TCustomXSLPageProducer.
	GetVersion (see page 7366)	Returns the value of the Version property.
	GetXML (see page 7367)	Returns the value of the XML property.
	GetXMLData (see page 7888)	This is GetXMLData, a member of class TCustomXSLPageProducer.
	GetXMLDocument (see page 7888)	This is GetXMLDocument, a member of class TCustomXSLPageProducer.

	GetXSLData (see page 7888)	This is GetXSLData, a member of class TCustomXSLPageProducer.
	HasHTMLBrowserView (see page 7888)	This is HasHTMLBrowserView, a member of class TCustomXSLPageProducer.
	HasScriptView (see page 7889)	This is HasScriptView, a member of class TCustomXSLPageProducer.
	HasXMLBrowserView (see page 7889)	This is HasXMLBrowserView, a member of class TCustomXSLPageProducer.
	HasXSLBrowserView (see page 7889)	This is HasXSLBrowserView, a member of class TCustomXSLPageProducer.
	InternalSetPrologValue (see page 7367)	This is InternalSetPrologValue, a member of class TXMLDocument.
	LoadData (see page 7368)	This is LoadData, a member of class TXMLDocument.
V	Loaded (see page 7368)	Initializes the component after the form file has been read into memory.
V	Notification (see page 7889)	Forwards notification messages to all owned components.
	ProduceContent (see page 7890)	Returns the content that the producer creates.
	ProduceContentFromStream (see page 7890)	This is ProduceContentFromStream, a member of class TCustomXSLPageProducer.
	ProduceContentFromString (see page 7890)	This is ProduceContentFromString, a member of class TCustomXSLPageProducer.
	ReadDOMVendor (see page 7371)	This is ReadDOMVendor, a member of class TXMLDocument.
	ReleaseDoc (see page 7372)	This is ReleaseDoc, a member of class TXMLDocument.
	SaveToUTF8String (see page 7374)	This is SaveToUTF8String, a member of class TXMLDocument.
	SaveToXMLStrings (see page 7375)	Saves the XML (see page 7392) document to an internal string list.
V	SetActive (see page 7375)	Sets the value of the Active property.
	SetAppDispatcher (see page 7891)	This is SetAppDispatcher, a member of class TCustomXSLPageProducer.
	SetDocumentElement (see page 7375)	Sets the value of the DocumentElement property.
	SetDOMDocument (see page 7376)	Sets the value of the DOMDocument property.
	SetEncoding (see page 7376)	Sets the value of the Encoding property.
	SetFileName (see page 7376)	Sets the value of the FileName property.
	SetModified (see page 7377)	This is SetModified, a member of class TXMLDocument.
	SetNodeIndentStr (see page 7377)	Sets the value of the NodeIndentStr property.
	SetOnAsyncLoad (see page 7377)	Supplies an event handler that responds when the AsyncLoadState property changes.
	SetOptions (see page 7378)	Sets the value of the Options property.
	SetParseOptions (see page 7378)	Sets the value of the ParseOptions property.
	SetPrologValue (see page 7378)	This is SetPrologValue, a member of class TXMLDocument.
	SetStandAlone (see page 7378)	Sets the value of the StandAlone property.
	SetVersion (see page 7379)	Sets the value of the Version property.
	SetXML (see page 7379)	Sets the value of the XML property.
	SetXMLStrings (see page 7379)	This is SetXMLStrings, a member of class TXMLDocument.
	WriteDOMVendor (see page 7380)	This is WriteDOMVendor, a member of class TXMLDocument.
	XMLStringsChanging (see page 7380)	This is XMLStringsChanging, a member of class TXMLDocument.

Protected Properties

	Name	Description
R	DocBindingInfo (see page 7383)	This is DocBindingInfo, a member of class TXMLDocument.
	DocSource (see page 7383)	This is DocSource, a member of class TXMLDocument.
R	DOMParseOptions (see page 7385)	This is DOMParseOptions, a member of class TXMLDocument.
R	DOMPersist (see page 7386)	This is DOMPersist, a member of class TXMLDocument.
	PrefixID (see page 7391)	This is PrefixID, a member of class TXMLDocument.

Public Methods

	Name	Description
	_AddRef (see page 3503)	Increments the reference count for this interface.
	_Release (see page 3503)	Decrements the reference count for this interface.
	AddChild (see page 7354)	This is the overview for the AddChild method overload.
	AddChild (see page 7643)	This is the overview for the AddChild method overload.
V	AfterConstruction (see page 7355)	Responds after the last constructor has executed.
V	Content (see page 7885)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property.

	ContentFromStream (see page 7885)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using an XSL template that can be read from a specified stream.
	ContentFromString (see page 7886)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.
	ContentFromWideString (see page 7886)	Returns the transformed value of the XML supplied by the XMLData (see page 7892) property using a specified XSL template.
	Create (see page 7351)	This is the overview for the Create constructor overload.
	CreateElement (see page 7357)	Creates a new element node that is associated with this document.
	CreateElement (see page 7644)	Creates a new element node that is associated with this document.
	CreateNode (see page 7357)	Creates a new node that is associated with this document.
	CreateNode (see page 7645)	Creates a new node that is associated with this document.
	Destroy (see page 7352)	Disposes of a TXMLDocument (see page 7347) object.
	GeneratePrefix (see page 7360)	Generates a unique namespace prefix that can be used for a node in the document.
	GeneratePrefix (see page 7646)	Generates a unique namespace prefix that can be used for a node in the document.
	GetActive (see page 7646)	Returns the value of the Active (see page 7661) property.
	GetAsyncLoadState (see page 7647)	Returns the value of the AsyncLoadState (see page 7662) property.
	GetChildNodes (see page 7647)	Returns the value of the ChildNodes (see page 7663) property.
	GetDocBinding (see page 7362)	Replaces the document element with an instance of a specified implementation class.
	GetDocBinding (see page 7647)	Replaces a specified child node with an instance of a specified implementation class.
	GetDocumentElement (see page 7648)	Returns the value of the DocumentElement (see page 7663) property.
	GetDocumentNode (see page 7648)	Returns the value of the Node (see page 7665) property.
	GetDocumentObject (see page 7479)	This is GetDocumentObject, a member of class IXMLDocumentAccess.
	GetDOMDocument (see page 7648)	Returns the value of the DOMDocument (see page 7663) property.
	GetDOMPersist (see page 7479)	This is GetDOMPersist, a member of class IXMLDocumentAccess.
	GetEncoding (see page 7649)	Returns the value of the Encoding (see page 7664) property.
	GetFileName (see page 7649)	Returns the value of the FileName (see page 7664) property.
	GetModified (see page 7649)	Returns the value of the Modified (see page 7665) property.
	GetNodeIndentStr (see page 7650)	Returns the value of the NodeIndentStr (see page 7666) property.
	GetOptions (see page 7650)	Returns the value of the Options (see page 7666) property.
	GetParseOptions (see page 7650)	Returns the value of the ParseOptions (see page 7667) property.
	GetSchemaRef (see page 7650)	Returns the value of the SchemaRef (see page 7667) property.
	GetStandAlone (see page 7651)	Returns the value of the StandAlone (see page 7668) property.
	GetTemplateFileType (see page 6090)	This is GetTemplateFileType, a member of class IProducerEditorViewSupport.
	GetVersion (see page 7651)	Returns the value of the Version (see page 7668) property.
	GetXML (see page 7651)	Returns the value of the XML (see page 7668) property.
	GetXMLData (see page 6090)	This is GetXMLData, a member of class IProducerEditorViewSupport.
	GetXSLData (see page 6090)	This is GetXSLData, a member of class IProducerEditorViewSupport.
	HasHTMLBrowserView (see page 6091)	This is HasHTMLBrowserView, a member of class IProducerEditorViewSupport.
	HasScriptView (see page 6091)	This is HasScriptView, a member of class IProducerEditorViewSupport.
	HasXMLBrowserView (see page 6091)	This is HasXMLBrowserView, a member of class IProducerEditorViewSupport.
	HasXSLBrowserView (see page 6091)	This is HasXSLBrowserView, a member of class IProducerEditorViewSupport.
	IsEmptyDoc (see page 7367)	Indicates whether the TXMLDocument (see page 7347) instance represents an empty document.
	IsEmptyDoc (see page 7652)	Indicates whether the IXMLDocument (see page 7640) instance represents an empty document.
	LoadFromFile (see page 7368)	Loads an XML (see page 7392) document and activates it.
	LoadFromFile (see page 7652)	Loads an XML (see page 7668) document from disk.
	LoadFromStream (see page 7369)	Loads an XML (see page 7392) document from a stream and activates it.
	LoadFromStream (see page 7653)	Loads an XML (see page 7668) document from a stream and activates it.
	LoadFromXML (see page 7370)	This is the overview for the LoadFromXML method overload.
	LoadFromXML (see page 7653)	This is the overview for the LoadFromXML method overload.
	NewInstance (see page 7370)	Allocates memory for an instance of an object type and returns a pointer to that new instance.

	QueryInterface (see page 3504)	Returns a reference to a specified interface if the object supports that interface.
	Refresh (see page 7371)	Updates the parsed XML (see page 7392) document to reflect external changes.
	Refresh (see page 7654)	Updates the parsed XML (see page 7668) document to reflect external changes.
	RegisterDocBinding (see page 7372)	Registers an implementation class for child nodes with a specified tag name.
	RegisterDocBinding (see page 7654)	Registers an implementation class for child nodes with a specified tag name.
	Resync (see page 7372)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	Resync (see page 7655)	Forces all child nodes and attribute nodes to be reread from the underlying DOM implementation.
	SaveToFile (see page 7373)	Saves the XML (see page 7392) document to disk.
	SaveToFile (see page 7655)	Saves the XML (see page 7668) document to disk.
	SaveToStream (see page 7373)	Saves the XML (see page 7392) document to a stream.
	SaveToStream (see page 7656)	Saves the XML (see page 7668) document to a stream.
	SaveToXML (see page 7374)	This is the overview for the SaveToXML method overload.
	SaveToXML (see page 7656)	This is the overview for the SaveToXML method overload.
	SetActive (see page 7657)	Sets the value of the Active (see page 7661) property.
	SetDocumentElement (see page 7657)	Sets the value of the DocumentElement (see page 7663) property.
	SetDOMDocument (see page 7658)	Sets the value of the DOMDocument (see page 7663) property.
	SetEncoding (see page 7658)	Sets the value of the Encoding (see page 7664) property.
	SetFileName (see page 7658)	Sets the value of the FileName (see page 7664) property.
	SetNodeIndentStr (see page 7659)	Sets the value of the NodeIndentStr (see page 7666) property.
	SetOnAsyncLoad (see page 7659)	Supplies an event handler that responds when the AsyncLoadState (see page 7662) property changes.
	SetOptions (see page 7659)	Sets the value of the Options (see page 7666) property.
	SetParseOptions (see page 7660)	Sets the value of the ParseOptions (see page 7667) property.
	SetStandAlone (see page 7660)	Sets the value of the StandAlone (see page 7668) property.
	SetVersion (see page 7660)	Sets the value of the Version (see page 7668) property.
	SetXML (see page 7661)	Sets the value of the XML (see page 7668) property.

Public Properties

	Name	Description
	Active (see page 7661)	Specifies whether the XML (see page 7668) document has been parsed so that the DOM interfaces are available to examine or change its contents.
R	AsyncLoadState (see page 7382)	Indicates the state of the DOM parser when it parses the XML (see page 7392) document asynchronously.
R	AsyncLoadState (see page 7662)	Indicates the state of the DOM parser when it parses the XML (see page 7668) document asynchronously.
R	ChildNodes (see page 7383)	Lists all of the child nodes of the document.
R	ChildNodes (see page 7663)	Lists all of the child nodes of the document.
R	Dispatcher (see page 7891)	Specifies the interface of a component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
R	DispatcherComponent (see page 7892)	Specifies the component that manages the HTTP request messages to which this page producer responds and that receives the response content that this page producer creates.
	DocumentElement (see page 7383)	Provides access to the root node of the XML (see page 7392) document.
	DocumentElement (see page 7663)	Provides access to the root node of the XML (see page 7668) document.
R	DocumentObject (see page 7479)	This is DocumentObject, a member of class IXMLDocumentAccess.
	DOMDocument (see page 7384)	Provides access to the DOM interface for the parsed XML (see page 7392) document.
	DOMDocument (see page 7663)	Provides access to the DOM interface for the parsed XML (see page 7668) document.
	DOMImplementation (see page 7385)	Provides access to the DOM interface for document-independent functions.
R	DOMPersist (see page 7480)	This is DOMPersist, a member of class IXMLDocumentAccess.
	Encoding (see page 7386)	Indicates the character set that the XML (see page 7392) document uses to encode characters.

	Encoding (see page 7664)	Indicates the character set that the XML (see page 7668) document uses to encode characters.
	FileName (see page 7664)	Indicates the XML (see page 7668) document file, if any, that this interface represents.
R	Modified (see page 7387)	Indicates whether the XML (see page 7392) document has been modified.
R	Modified (see page 7665)	Indicates whether the XML (see page 7668) document has been modified.
R	Node (see page 7388)	Provides access to the document node for the XML (see page 7392) document.
R	Node (see page 7665)	Provides access to the document node for the XML (see page 7668) document.
	NodeIndentStr (see page 7666)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7668) text.
	NSPrefixBase (see page 7389)	Specifies the base string for automatically-generated namespace prefixes.
	Options (see page 7666)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7667)	Specifies implementation-specific options that are followed when parsing the XML (see page 7668) document.
R	SchemaRef (see page 7391)	Returns the name of the schema associated with this XML (see page 7392) document.
R	SchemaRef (see page 7667)	Returns the name of the schema associated with this XML (see page 7668) document.
	StandAlone (see page 7391)	Indicates whether the XML (see page 7392) document includes any external markup declarations.
	StandAlone (see page 7668)	Indicates whether the XML (see page 7668) document includes any external markup declarations.
	Version (see page 7391)	Indicates the XML (see page 7392) version that the document uses.
	Version (see page 7668)	Indicates the XML (see page 7668) version that the document uses.
	XML (see page 7668)	Specifies the content of the XML document associated with this IXMLDocument (see page 7640) interface.

Published Events

	Name	Description
	AfterClose (see page 7393)	Occurs after the XML (see page 7392) document is closed.
	AfterNodeChange (see page 7393)	Occurs after a change is made to one of the nodes in the document.
	AfterOpen (see page 7394)	Occurs immediately after the XML (see page 7392) document is opened.
	BeforeClose (see page 7394)	Occurs immediately before the XML (see page 7392) document is closed.
	BeforeNodeChange (see page 7394)	Occurs before a change is made to one of the nodes in the document.
	BeforeOpen (see page 7395)	Occurs immediately before the XML (see page 7392) document is opened.
	OnAsyncLoad (see page 7395)	Occurs at various stages during the asynchronous parsing of an XML (see page 7392) document.

Published Properties

	Name	Description
	Active (see page 7381)	Specifies whether the XML (see page 7392) document has been parsed so that the DOM interfaces are available to examine or change its contents.
	DOMVendor (see page 7386)	Specifies the DOM implementation to use for parsing and manipulating the XML (see page 7392) document.
	FileName (see page 7387)	Specifies the file name for the XML (see page 7392) document associated with this TXMLDocument (see page 7347) object.
	NodeIndentStr (see page 7388)	Indicates the string that is inserted before nested nodes in the formatted XML (see page 7392) text.
	Options (see page 7389)	Influences various aspects of the document's behavior.
	ParseOptions (see page 7390)	Specifies implementation-specific options that are followed when parsing the XML (see page 7392) document.
	XML (see page 7392)	Specifies the content of the XML document associated with this TXMLDocument (see page 7347) object.
	XMLData (see page 7899)	Specifies the component that supplies the XML data that this component transforms.

87.1.3.2 TXSLPageProducer Properties

The properties of the TXSLPageProducer class are listed here.

Published Properties

	Name	Description
	XMLData (see page 7899)	Specifies the component that supplies the XML data that this component transforms.

87.1.3.2.1 TXSLPageProducer.XMLData Property

Specifies the component that supplies the XML data that this component transforms.

Pascal

```
property XMLData: TComponent;
```

C++

```
__property TComponent * XMLData;
```

Description

Use the XMLData property to specify the XML data that this component transforms. XMLData must be set to a component that supports either the IXMLDocument interface (for example, TXMLDocument or one of its descendants) or the IGetXMLStream interface (for example, TXMLBroker).

See Also

TXMLDocument (see page 7347)

TXMLBroker (see page 7303)

88 ZLib Namespace

This is namespace ZLib.

Topics

Name	Description
Classes (see page 7904)	The following table lists classes in this documentation.
Functions (see page 7917)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 7926)	The following table lists structs, records, enums in this documentation.
Types (see page 7929)	The following table lists types in this documentation.
Constants (see page 7932)	The following table lists constants in this documentation.

Classes

	Name	Description
	EZCompressionError (see page 7904)	This is class ZLib.EZCompressionError.
	EZDecompressionError (see page 7904)	This is class ZLib.EZDecompressionError.
	EZLibError (see page 7905)	This is class ZLib.EZLibError.
	TCustomZStream (see page 7905)	Abstract ancestor class for compression streams.
	TZCompressionStream (see page 7907)	Compresses data as it is written to the stream.
	TZDecompressionStream (see page 7912)	Decompresses data that was written to a TZCompressionStream (see page 7907) object.

Constants

Name	Description
_z_errmsg (see page 7932)	This is constant ZLib._z_errmsg.
SZInvalid (see page 7933)	This is constant ZLib.SZInvalid.
Z_ASCII (see page 7933)	This is constant ZLib.Z_ASCII.
Z_BEST_COMPRESSION (see page 7933)	This is constant ZLib.Z_BEST_COMPRESSION.
Z_BEST_SPEED (see page 7934)	This is constant ZLib.Z_BEST_SPEED.
Z_BINARY (see page 7934)	This is constant ZLib.Z_BINARY.
Z_BUF_ERROR (see page 7934)	This is constant ZLib.Z_BUF_ERROR.
Z_DATA_ERROR (see page 7935)	This is constant ZLib.Z_DATA_ERROR.
Z_DEFAULT_COMPRESSION (see page 7935)	This is constant ZLib.Z_DEFAULT_COMPRESSION.
Z_DEFAULT_STRATEGY (see page 7935)	This is constant ZLib.Z_DEFAULT_STRATEGY.
Z_DEFLATED (see page 7936)	This is constant ZLib.Z_DEFLATED.
Z_ERRNO (see page 7936)	This is constant ZLib.Z_ERRNO.
Z_FILTERED (see page 7936)	This is constant ZLib.Z_FILTERED.

Z_FINISH (see page 7937)	This is constant ZLib.Z_FINISH.
Z_FULL_FLUSH (see page 7937)	This is constant ZLib.Z_FULL_FLUSH.
Z_HUFFMAN_ONLY (see page 7937)	This is constant ZLib.Z_HUFFMAN_ONLY.
Z_MEM_ERROR (see page 7938)	This is constant ZLib.Z_MEM_ERROR.
Z_NEED_DICT (see page 7938)	This is constant ZLib.Z_NEED_DICT.
Z_NO_COMPRESSION (see page 7938)	This is constant ZLib.Z_NO_COMPRESSION.
Z_NO_FLUSH (see page 7939)	This is constant ZLib.Z_NO_FLUSH.
Z_OK (see page 7939)	This is constant ZLib.Z_OK.
Z_PARTIAL_FLUSH (see page 7939)	This is constant ZLib.Z_PARTIAL_FLUSH.
Z_STREAM_END (see page 7940)	This is constant ZLib.Z_STREAM_END.
Z_STREAM_ERROR (see page 7940)	This is constant ZLib.Z_STREAM_ERROR.
Z_SYNC_FLUSH (see page 7940)	This is constant ZLib.Z_SYNC_FLUSH.
Z_UNKNOWN (see page 7941)	This is constant ZLib.Z_UNKNOWN.
Z_VERSION_ERROR (see page 7941)	This is constant ZLib.Z_VERSION_ERROR.
ZLevels (see page 7941)	This is constant ZLib.ZLevels.
ZLIB_VERSION (see page 7942)	This is constant ZLib.ZLIB_VERSION.

Functions

	Name	Description
	adler32 (see page 7917)	This is function ZLib.adler32.
	deflate (see page 7918)	This is function ZLib.deflate.
	deflateEnd (see page 7918)	This is function ZLib.deflateEnd.
	deflateInit_ (see page 7918)	This is function ZLib.deflateInit_.
	DeflateInit2_ (see page 7919)	This is function ZLib.DeflateInit2_.
	inflate (see page 7919)	This is function ZLib.inflate.
	inflateEnd (see page 7919)	This is function ZLib.inflateEnd.
	inflateInit_ (see page 7920)	This is function ZLib.inflateInit_.
	inflateInit2_ (see page 7920)	This is function ZLib.inflateInit2_.
	inflateReset (see page 7920)	This is function ZLib.inflateReset.
	MoveI32 (see page 7921)	This is function ZLib.MoveI32.
	ZCompress (see page 7921)	This is function ZLib.ZCompress.
	ZCompress (see page 7921)	This is function ZLib.ZCompress.
	ZCompressStr (see page 7922)	This is function ZLib.ZCompressStr.
	ZCompressStr (see page 7922)	This is function ZLib.ZCompressStr.
	ZCompressStream (see page 7922)	This is function ZLib.ZCompressStream.
	ZCompressStream (see page 7923)	This is function ZLib.ZCompressStream.
	ZDecompress (see page 7923)	This is function ZLib.ZDecompress.
	ZDecompressStr (see page 7923)	This is function ZLib.ZDecompressStr.
	ZDecompressStream (see page 7924)	This is function ZLib.ZDecompressStream.
	zlibAllocMem (see page 7924)	This is function ZLib.zlibAllocMem.
	zlibFreeMem (see page 7924)	This is function ZLib.zlibFreeMem.
	ZSendToBrowser (see page 7925)	This is function ZLib.ZSendToBrowser.

Structs, Records, Enums

	Name	Description
	TCompressionLevel (see page 7926)	TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.
	TZCompressionLevel (see page 7927)	This is record ZLib.TZCompressionLevel.
	TZStreamRec (see page 7927)	This is record ZLib.TZStreamRec.

Types

Name	Description
TCompressionStream (see page 7929)	Compresses data as it is written to the stream.
TCustomZLibStream (see page 7929)	This is type ZLib.TCustomZLibStream.

TDecompressionStream (see page 7930)	Decompresses data that was written to a TZCompressionStream (see page 7907) object.
TZAlloc (see page 7930)	This is type ZLib.TZAlloc.
TZFree (see page 7930)	This is type ZLib.TZFree.

88.1 Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	EZCompressionError (see page 7904)	This is class ZLib.EZCompressionError.
	EZDecompressionError (see page 7904)	This is class ZLib.EZDecompressionError.
	EZLibError (see page 7905)	This is class ZLib.EZLibError.
	TCustomZStream (see page 7905)	Abstract ancestor class for compression streams.
	TZCompressionStream (see page 7907)	Compresses data as it is written to the stream.
	TZDecompressionStream (see page 7912)	Decompresses data that was written to a TZCompressionStream (see page 7907) object.

88.1.1 EZCompressionError Class

Class Hierarchy

Pascal

```
EZCompressionError = class(EZLibError);
```

C++

```
class EZCompressionError : public EZLibError;
```

File

ZLib

Description

This is class ZLib.EZCompressionError.

88.1.2 EZDecompressionError Class

Class Hierarchy

Pascal

```
EZDecompressionError = class(EZLibError);
```

C++

```
class EZDecompressionError : public EZLibError;
```

File

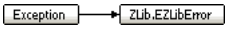
ZLib

Description

This is class ZLib.EZDecompressionError.

88.1.3 EZLibError Class

Class Hierarchy



Pascal

```
EZLibError = class(Exception);
```

C++

```
class EZLibError : public Exception;
```

File

ZLib

Description

This is class ZLib.EZLibError.

88.1.4 TCustomZStream Class

Abstract ancestor class for compression streams.

Class Hierarchy

Pascal

```
TCustomZStream = class(TStream);
```

C++

```
class TCustomZStream : public TStream;
```

File

ZLib

Description

TCustomZStream is an abstract class, which forms the base for TCompressionStream (see page 7929) and TDecompressionStream (see page 7930). Since it is an abstract class, instances of TCustomZStream cannot be directly instantiated. TCustomZStream serves as the starting point for your own custom data compression streams.

See Also

TCompressionStream (see page 7929)

TDecompressionStream (see page 7930)

TStream

88.1.4.1 TCustomZStream Members

The following tables list the members exposed by TCustomZStream.

Protected Events

	Name	Description
	OnProgress (see page 7907)	Occurs when reading or writing a large block of data in a single call.

Protected Methods

	Name	Description
	Create (see page 7906)	Initializes an instance of a TCustomZStream (see page 7905)-derived object.
	DoProgress (see page 7906)	Calls the OnDoProgress method, if assigned.

88.1.4.2 TCustomZStream.Create Constructor

Initializes an instance of a TCustomZStream (see page 7905)-derived object.

Pascal

```
constructor Create(stream: TStream);
```

C++

```
__fastcall TCustomZStream(TStream stream);
```

Description

TCustomZStream (see page 7905) is intended to be an abstract base class for your own custom data compression streams. Its constructor is protected; therefore you cannot directly instantiate an instance of this class.

A concrete class derived from TCustomZStream (see page 7905) should, in its own public constructor, call the constructor inherited from TCustomZStream (see page 7905).

88.1.4.3 TCustomZStream Methods

The methods of the TCustomZStream class are listed here.

Protected Methods

	Name	Description
	DoProgress (see page 7906)	Calls the OnDoProgress method, if assigned.

88.1.4.3.1 TCustomZStream.DoProgress Method

Calls the OnDoProgress method, if assigned.

Pascal

```
procedure DoProgress; dynamic;
```

C++

```
__fastcall DoProgress();
```

Description

Derived compression stream classes should call the DoProgress method when reading or writing a large block of data in a single call. This gives the derived class an opportunity to update a user interface control such as a progress indicator.

See Also

OnProgress (see page 7907)

88.1.4.4 TCustomZStream Events

The events of the TCustomZStream class are listed here.

Protected Events

	Name	Description
	OnProgress (see page 7907)	Occurs when reading or writing a large block of data in a single call.

88.1.4.4.1 TCustomZStream.OnProgress Event

Occurs when reading or writing a large block of data in a single call.

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

Classes derived from TCustomZStream (see page 7905) can use the OnProgress event to perform some action during a long read or write operation.

For example, the TCompressionStream (see page 7929) class calls the OnProgress event handler, after the stream's output buffer has been filled and subsequently written. Similarly, the TDecompressionStream (see page 7930) class calls the OnProgress event handler when its input buffer has been filled and is ready to be decompressed.

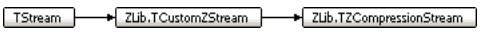
See Also

DoProgress (see page 7906)

88.1.5 TZCompressionStream Class

Compresses data as it is written to the stream.

Class Hierarchy



Pascal

```
TZCompressionStream = class(TCustomZStream);
```

C++

```
class TZCompressionStream : public TCustomZStream;
```

File

ZLib

Description

TZCompressionStream is a write-only, sequential access stream that compresses data as it is written. The compressed data is

written to a separate stream, which is passed to the TZCompressionStream constructor. TZCompressionStream does not take ownership of the output stream; you are responsible for creating, initializing, and destroying the stream when it is no longer needed.

Reading from a TZCompressionStream object will raise an exception of type ECompressionError, with the message string set to "Invalid Stream Operation." Similarly, a Seek (see page 7910) operation causes an ECompressionError exception, however, you can perform a Seek (see page 7910) operation with a zero offset from the current position. This will return the number of raw, uncompressed bytes that have been written to the stream.

You set the compression level to be used when you create the stream object. The possible compression levels are: clNone, clFastest, clDefault, and clMax.

When the stream's output buffer is filled, a Write (see page 7910) operation on TZCompressionStream will write the contents of the buffer to the output stream, and then call the OnProgress (see page 7912) event. You can use the OnProgress (see page 7912) event to update a user interface control such as a progress indicator.

See Also

TCustomZStream (see page 7905)

TZDecompressionStream (see page 7912)

TStream

88.1.5.1 TZCompressionStream Members

The following tables list the members exposed by TZCompressionStream.

Protected Methods

	Name	Description
	DoProgress (see page 7906)	Calls the OnDoProgress method, if assigned.

Public Events

	Name	Description
	OnProgress (see page 7912)	Occurs after the stream's output buffer has been written.

Public Methods

	Name	Description
	Create (see page 7908)	This is the overview for the Create constructor overload.
V	Destroy (see page 7909)	Destroys the TZCompressionStream (see page 7907) object.
V	Read (see page 7910)	Raises an exception of type ECompressionError.
V	Seek (see page 7910)	Raises an exception of type ECompressionErrror.
V	Write (see page 7910)	Writes compressed data to the destination output stream.

Public Properties

	Name	Description
R	CompressionRate (see page 7911)	Returns the percentage by which the data has been compressed.

88.1.5.2 Create Constructor

This is the overview for the Create constructor overload.

Overload List

	Name	Description
	TZCompressionStream.Create (TCompressionLevel, TStream) (see page 7909)	Creates and initializes an instance of a TZCompressionStream (see page 7907) object.
	TZCompressionStream.Create (TStream, TZCompressionLevel) (see page 7909)	Creates and initializes an instance of a TZCompressionStream (see page 7907) object.

88.1.5.2.1 TZCompressionStream.Create Constructor (TCompressionLevel, TStream)

Creates and initializes an instance of a TZCompressionStream (see page 7907) object.

Pascal

```
constructor Create(dest: TStream; compressionLevel: TZCompressionLevel = zcDefault); overload;
constructor Create(compressionLevel: TCompressionLevel; dest: TStream); overload;
```

C++

```
__fastcall TZCompressionStream(TStream dest, TZCompressionLevel compressionLevel = zcDefault);
__fastcall TZCompressionStream(TCompressionLevel compressionLevel, TStream dest);
```

Description

Call Create to instantiate a TZCompressionStream (see page 7907) object.

The CompressionLevel parameter determines the type of compression algorithm to be used, and must be one of the following: clNone, clFastest, clDefault, or clMax.

The Dest parameter is a TStream object; compressed data will be written to this stream.

88.1.5.3 TZCompressionStream.Destroy Destructor

Destroys the TZCompressionStream (see page 7907) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TZCompressionStream();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, as this will check the validity of the object before calling Destroy.

All data in the TZCompressionStream (see page 7907)'s cache is flushed to the output stream when the object is destroyed.

88.1.5.4 TZCompressionStream Methods

The methods of the TZCompressionStream class are listed here.

Public Methods

	Name	Description
	Read (see page 7910)	Raises an exception of type ECompressionError.
	Seek (see page 7910)	Raises an exception of type ECompressionErrror.
	Write (see page 7910)	Writes compressed data to the destination output stream.

88.1.5.4.1 TZCompressionStream.Read Method

Raises an exception of type ECompressionError.

Pascal

```
function Read(var buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( buffer, Longint count);
```

Description

Read operations are not permitted on a TZCompressionStream (see page 7907) object. If a Read is attempted, an exception of type ECompressionError will be raised, with the message string "Invalid Stream Operation."

See Also

Write (see page 7910)

Seek (see page 7910)

Read

88.1.5.4.2 TZCompressionStream.Seek Method

Raises an exception of type ECompressionErrror.

Pascal

```
function Seek(offset: Longint; origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint offset, Word origin);
```

Description

A TZCompressionStream (see page 7907) is a write-only, sequential access data stream. Read (see page 7910) operations are not permitted, and will raise an exception of type ECompressionError. Similarly, Seek operations will also raise an ECompressionError, with the message string "Invalid Stream Operation."

However, you can call the Seek method with an offset of zero, and the Origin set to soFromCurrent. In this special case, the Seek method will return the number of raw (uncompressed) bytes that have been written to the stream so far.

See Also

Read (see page 7910)

Write (see page 7910)

Seek

88.1.5.4.3 TZCompressionStream.Write Method

Writes compressed data to the destination output stream.

Pascal

```
function Write(const buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  buffer, Longint count);
```

Description

The Write method causes the data in the supplied buffer to be written the output stream. Buffer contains the raw (uncompressed) data to be written, and Count contains the number of bytes to write.

The TZCompressionStream (see page 7907) object caches data internally; data is flushed to the output stream when the cache becomes full, and when the object is destroyed (if pending data remains in the cache).

After data is written to the output stream, the OnProgress (see page 7912) event is called. This gives you an opportunity to update a user interface control such as a progress indicator.

The result of the Write method is the number of bytes transferred from the buffer

See Also

Read (see page 7910)

Seek (see page 7910)

Write

88.1.5.5 TZCompressionStream Properties

The properties of the TZCompressionStream class are listed here.

Public Properties

	Name	Description
	CompressionRate (see page 7911)	Returns the percentage by which the data has been compressed.

88.1.5.5.1 TZCompressionStream.CompressionRate Property

Returns the percentage by which the data has been compressed.

Pascal

```
property CompressionRate: Single;
```

C++

```
__property Single CompressionRate;
```

Description

The compression rate is a read-only property. Its value is calculated and updated as data is written to the stream. For example, if a total of 100 raw (uncompressed) bytes have been written to the stream, and only 25 compressed bytes have been written out, then reading the CompressionRate property will return a value of 75.

88.1.5.6 TZCompressionStream Events

The events of the TZCompressionStream class are listed here.

Public Events

	Name	Description
	OnProgress (see page 7912)	Occurs after the stream's output buffer has been written.

88.1.5.6.1 TZCompressionStream.OnProgress Event

Occurs after the stream's output buffer has been written.

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

The OnProgress event is automatically called from a Write (see page 7910) operation, after the object's internal buffer has been filled and written to the output stream.

The OnProgress event is of type TNotifyEvent. When your procedure is called, the Sender parameter will be set to the object itself. You can use the OnProgress event to update a user interface control, such as a progress indicator. You can use the Position property, inherited from TStream, to determine the number of raw (uncompressed) bytes written to the output stream so far.

88.1.6 TZDecompressionStream Class

Decompresses data that was written to a TZCompressionStream (see page 7907) object.

Class Hierarchy

Pascal

```
TZDecompressionStream = class(TCustomZStream);
```

C++

```
class TZDecompressionStream : public TCustomZStream;
```

File

ZLib

Description

TZDecompressionStream is the analog of the TCompressionStream (see page 7929) class. It is a read-only, unidirectional stream; you can seek forward in the stream, but not backwards. A separate input stream provides the source data. The input stream will be a stream that was previously written to using a TCompressionStream (see page 7929) object. The input stream is passed to the constructor, however, TZDecompressionStream does not take ownership of this object. You are responsible for creating, initializing, and destroying the input stream when it is no longer needed.

Writing to the stream will raise an exception, as will requesting the size of the stream, and attempting to perform a Seek (see page 7914) operation backwards or relative to the current position.

The TZDecompressionStream object's OnProgress (see page 7915) event is called when a block of data is read from the input stream. You can read the TZDecompressionStream object's Position property to determine the number of raw, uncompressed

bytes that have been read from the input stream so far.

88.1.6.1 TZDecompressionStream Members

The following tables list the members exposed by TZDecompressionStream.

Protected Methods

	Name	Description
	DoProgress (see page 7906)	Calls the OnDoProgress method, if assigned.

Public Events

	Name	Description
	OnProgress (see page 7915)	Occurs after a block of data has been read from the input stream.

Public Methods

	Name	Description
	Create (see page 7913)	Creates and initializes an instance of a TZDecompressionStream (see page 7912) object.
	Destroy (see page 7913)	Destroys the TZDecompressionStream (see page 7912) object.
	Read (see page 7914)	Reads a block of data from the input stream, and decompresses it.
	Seek (see page 7914)	Resets the input stream, or moves the position forward.
	Write (see page 7915)	Raises an exception of type ECompressionError.

88.1.6.2 TZDecompressionStream.Create Constructor

Creates and initializes an instance of a TZDecompressionStream (see page 7912) object.

Pascal

```
constructor Create(source: TStream);
```

C++

```
__fastcall TZDecompressionStream(TStream source);
```

Description

Call Create to instantiate a TZDecompressionStream (see page 7912) object.

The Source parameter is the source input stream, which was previously created with a TZCompressionStream (see page 7907) object. Subsequent Read (see page 7914) operations will read data from the Source stream, and decompress the data on the fly.

88.1.6.3 TZDecompressionStream.Destroy Destructor

Destroys the TZDecompressionStream (see page 7912) object.

Pascal

```
destructor Destroy; override;
```

C++

```
virtual __fastcall ~TZDecompressionStream();
```

Description

Do not call the Destroy method directly. Instead, use the Free method, as this will check the validity of the object before calling

Destroy.

If there is data remaining in the object's internal buffer, the input stream position is moved backwards by the number of bytes remaining.

88.1.6.4 TZDecompressionStream Methods

The methods of the TZDecompressionStream class are listed here.

Public Methods

	Name	Description
	Read (see page 7914)	Reads a block of data from the input stream, and decompresses it.
	Seek (see page 7914)	Resets the input stream, or moves the position forward.
	Write (see page 7915)	Raises an exception of type ECompressionError.

88.1.6.4.1 TZDecompressionStream.Read Method

Reads a block of data from the input stream, and decompresses it.

Pascal

```
function Read(var buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Read( buffer, Longint count);
```

Description

The Read method reads a block of data from the input stream, decompresses it, and stores it in the supplied buffer.

The Buffer parameter is the destination for the decompressed data.

Count is size of the destination buffer. The Read operation will end when the destination buffer is full, or when there are no more bytes available to be read from the input stream.

After reading a block of data, the OnProgress (see page 7915) event is fired by the object. This gives you a chance to update a user interface control such as a progress indicator during a long Read operation.

The result of the Read operation is the number of decompressed bytes placed into the destination buffer.

See Also

Write (see page 7915)

Seek (see page 7914)

Read

88.1.6.4.2 TZDecompressionStream.Seek Method

Resets the input stream, or moves the position forward.

Pascal

```
function Seek(offset: Longint; origin: Word): Longint; override;
```

C++

```
virtual __fastcall Longint Seek(Longint offset, Word origin);
```

Description

A TZDecompressionStream (see page 7912) is a read-only, unidirectional input stream. Write (see page 7915) operations are not permitted, and will raise an exception of type ECompressionError. Seek operations can be used to move the stream position forward, but a Seek operation that attempts to move the position backwards will raise an ECompressionError exception, with the message string set to "Invalid Stream Operation."

You can call the Seek method with an Offset of zero, and the Origin set to soFromBeginning. This has the effect of resetting the input stream, effectively rewinding the stream back to the beginning.

Otherwise, the Offset parameter must be a positive Longint, and the Origin parameter must be either soFromCurrent, or soFromBeginning. This has the effect of moving the input stream position forward, decompressing data as it goes.

See Also

Read (see page 7914)

Write (see page 7915)

Seek

88.1.6.4.3 TZDecompressionStream.Write Method

Raises an exception of type ECompressionError.

Pascal

```
function Write(const buffer; count: Longint): Longint; override;
```

C++

```
virtual __fastcall Longint Write(const  buffer, Longint count);
```

Description

Write operations are not permitted on a TZDecompressionStream (see page 7912) object. If a Write operation is attempted, an exception of type ECompressionError will be raised, with the message string set to "Invalid Stream Operation."

See Also

Read (see page 7914)

Seek (see page 7914)

Write

88.1.6.5 TZDecompressionStream Events

The events of the TZDecompressionStream class are listed here.

Public Events

	Name	Description
	OnProgress (see page 7915)	Occurs after a block of data has been read from the input stream.

88.1.6.5.1 TZDecompressionStream.OnProgress Event

Occurs after a block of data has been read from the input stream.

Pascal

```
property OnProgress: TNotifyEvent;
```

C++

```
__property TNotifyEvent OnProgress;
```

Description

The OnProgress event is automatically called from a Read (see page 7914) operation. The event occurs after a block of data has been read from the input stream, and before the data has been decompressed.

The OnProgress event is of type TNotifyEvent. When your procedure is called, the Sender parameter will be set to the object itself. You can use the OnProgress event to update a user interface control, such as a progress indicator. You can use the Position property, inherited from TStream, to determine the number of raw (uncompressed) bytes read from the input stream so far.

88.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	adler32 (see page 7917)	This is function ZLib.adler32.
	deflate (see page 7918)	This is function ZLib.deflate.
	deflateEnd (see page 7918)	This is function ZLib.deflateEnd.
	deflateInit_ (see page 7918)	This is function ZLib.deflateInit_.
	DeflateInit2_ (see page 7919)	This is function ZLib.DeflateInit2_.
	inflate (see page 7919)	This is function ZLib.inflate.
	inflateEnd (see page 7919)	This is function ZLib.inflateEnd.
	inflateInit_ (see page 7920)	This is function ZLib.inflateInit_.
	inflateInit2_ (see page 7920)	This is function ZLib.inflateInit2_.
	inflateReset (see page 7920)	This is function ZLib.inflateReset.
	MoveI32 (see page 7921)	This is function ZLib.MoveI32.
	ZCompress (see page 7921)	This is function ZLib.ZCompress.
	ZCompress (see page 7921)	This is function ZLib.ZCompress.
	ZCompressStr (see page 7922)	This is function ZLib.ZCompressStr.
	ZCompressStr (see page 7922)	This is function ZLib.ZCompressStr.
	ZCompressStream (see page 7922)	This is function ZLib.ZCompressStream.
	ZCompressStream (see page 7923)	This is function ZLib.ZCompressStream.
	ZDecompress (see page 7923)	This is function ZLib.ZDecompress.
	ZDecompressStr (see page 7923)	This is function ZLib.ZDecompressStr.
	ZDecompressStream (see page 7924)	This is function ZLib.ZDecompressStream.
	zlibAllocMem (see page 7924)	This is function ZLib.zlibAllocMem.
	zlibFreeMem (see page 7924)	This is function ZLib.zlibFreeMem.
	ZSendToBrowser (see page 7925)	This is function ZLib.ZSendToBrowser.

88.2.1 ZLib.adler32 Function

Pascal

```
function adler32(adler: LongInt; const buf: PChar; len: Integer): LongInt;
```

C++

```
LongInt adler32(LongInt adler, const const char * buf, int len);
```

File

ZLib

Description

This is function ZLib.adler32.

88.2.2 ZLib.deflate Function

Pascal

```
function deflate(var strm: TZStreamRec; flush: Integer): Integer;
```

C++

```
int deflate(TZStreamRec strm, int flush);
```

File

ZLib

Description

This is function ZLib.deflate.

88.2.3 ZLib.deflateEnd Function

Pascal

```
function deflateEnd(var strm: TZStreamRec): Integer;
```

C++

```
int deflateEnd(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.deflateEnd.

88.2.4 ZLib.deflateInit_ Function

Pascal

```
function deflateInit_(var strm: TZStreamRec; level: Integer; version: PChar; recsize:
Integer): Integer;
```

C++

```
int deflateInit_(TZStreamRec strm, int level, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.deflateInit_.

88.2.5 ZLib.DeflateInit2_ Function

Pascal

```
function DeflateInit2_(var strm: TZStreamRec; level: integer; method: integer; windowBits:
integer; memLevel: integer; strategy: integer; version: PChar; recsize: integer): integer;
```

C++

```
int DeflateInit2_(TZStreamRec strm, int level, int method, int windowBits, int memLevel, int
strategy, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.DeflateInit2_.

88.2.6 ZLib.inflate Function

Pascal

```
function inflate(var strm: TZStreamRec; flush: Integer): Integer;
```

C++

```
int inflate(TZStreamRec strm, int flush);
```

File

ZLib

Description

This is function ZLib.inflate.

88.2.7 ZLib.inflateEnd Function

Pascal

```
function inflateEnd(var strm: TZStreamRec): Integer;
```

C++

```
int inflateEnd(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.inflateEnd.

88.2.8 ZLib.inflateInit_ Function

Pascal

```
function inflateInit_(var strm: TZStreamRec; version: PChar; recsize: Integer): Integer;
```

C++

```
int inflateInit_(TZStreamRec strm, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.inflateInit_.

88.2.9 ZLib.inflateInit2_ Function

Pascal

```
function inflateInit2_(var strm: TZStreamRec; windowBits: integer; version: PChar; recsize:
integer): integer;
```

C++

```
int inflateInit2_(TZStreamRec strm, int windowBits, const char * version, int recsize);
```

File

ZLib

Description

This is function ZLib.inflateInit2_.

88.2.10 ZLib.inflateReset Function

Pascal

```
function inflateReset(var strm: TZStreamRec): Integer;
```

C++

```
int inflateReset(TZStreamRec strm);
```

File

ZLib

Description

This is function ZLib.inflateReset.

88.2.11 ZLib.MoveI32 Function

Pascal

```
procedure MoveI32(const Source; var Dest; Count: Integer);
```

C++

```
MoveI32(const  Source,  Dest, int Count);
```

File

ZLib

Description

This is function ZLib.MoveI32.

88.2.12 ZLib.ZCompress Function

Pascal

```
procedure ZCompress(const inBuffer: Pointer; inSize: Integer; out outBuffer: Pointer; out
outSize: Integer; level: TCompressionLevel); overload;
```

C++

```
ZCompress(const void * inBuffer, int inSize, void * outBuffer, int outSize, TCompressionLevel
level);
```

File

ZLib

Description

This is function ZLib.ZCompress.

88.2.13 ZLib.ZCompress Function

Pascal

```
procedure ZCompress(const inBuffer: Pointer; inSize: Integer; out outBuffer: Pointer; out
outSize: Integer; level: TZCompressionLevel = zcDefault); overload;
```

C++

```
ZCompress(const void * inBuffer, int inSize, void * outBuffer, int outSize, TZCompressionLevel
level = zcDefault);
```

File

ZLib

Description

This is function ZLib.ZCompress.

88.2.14 ZLib.ZCompressStr Function

Pascal

```
function ZCompressStr(const s: string; level: TCompressionLevel): TBytes; overload;
```

C++

```
TBytes ZCompressStr(const UnicodeString s, TCompressionLevel level);
```

File

ZLib

Description

This is function ZLib.ZCompressStr.

88.2.15 ZLib.ZCompressStr Function

Pascal

```
function ZCompressStr(const s: string; level: TZCompressionLevel = zcDefault): TBytes;
overload;
```

C++

```
TBytes ZCompressStr(const UnicodeString s, TZCompressionLevel level = zcDefault);
```

File

ZLib

Description

This is function ZLib.ZCompressStr.

88.2.16 ZLib.ZCompressStream Function

Pascal

```
procedure ZCompressStream(inStream: TStream; outStream: TStream; level: TCompressionLevel);
overload;
```

C++

```
ZCompressStream(TStream inStream, TStream outStream, TCompressionLevel level);
```

File

ZLib

Description

This is function ZLib.ZCompressStream.

88.2.17 ZLib.ZCompressStream Function

Pascal

```
procedure ZCompressStream(inStream: TStream; outStream: TStream; level: TZCompressionLevel =
zcDefault); overload;
```

C++

```
ZCompressStream(TStream inStream, TStream outStream, TZCompressionLevel level = zcDefault);
```

File

ZLib

Description

This is function ZLib.ZCompressStream.

88.2.18 ZLib.ZDecompress Function

Pascal

```
procedure ZDecompress(const inBuffer: Pointer; inSize: Integer; out outBuffer: Pointer; out
outSize: Integer; outEstimate: Integer = 0);
```

C++

```
ZDecompress(const void * inBuffer, int inSize, void * outBuffer, int outSize, int outEstimate
= 0);
```

File

ZLib

Description

This is function ZLib.ZDecompress.

88.2.19 ZLib.ZDecompressStr Function

Pascal

```
function ZDecompressStr(const s: TBytes): string;
```

C++

```
UnicodeString ZDecompressStr(const TBytes s);
```

File

ZLib

Description

This is function ZLib.ZDecompressStr.

88.2.20 ZLib.ZDecompressStream Function

Pascal

```
procedure ZDecompressStream(inStream: TStream; outStream: TStream);
```

C++

```
ZDecompressStream(TStream inStream, TStream outStream);
```

File

ZLib

Description

This is function ZLib.ZDecompressStream.

88.2.21 ZLib.zlibAllocMem Function

Pascal

```
function zlibAllocMem(AppData: Pointer; Items: Integer; Size: Integer): Pointer;
```

C++

```
void * zlibAllocMem(void * AppData, int Items, int Size);
```

File

ZLib

Description

This is function ZLib.zlibAllocMem.

88.2.22 ZLib.zlibFreeMem Function

Pascal

```
procedure zlibFreeMem(AppData: Pointer; Block: Pointer);
```

C++

```
zlibFreeMem(void * AppData, void * Block);
```

File

ZLib

Description

This is function ZLib.zlibFreeMem.

88.2.23 ZLib.ZSendToBrowser Function

Pascal

```
procedure ZSendToBrowser(var s: string);
```

C++

```
ZSendToBrowser(UnicodeString s);
```

File

ZLib

Description

This is function ZLib.ZSendToBrowser.

88.3 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
	TCompressionLevel (see page 7926)	TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.
	TZCompressionLevel (see page 7927)	This is record ZLib.TZCompressionLevel.

Records

	Name	Description
	TZStreamRec (see page 7927)	This is record ZLib.TZStreamRec.

88.3.1 ZLib.TCompressionLevel Enumeration

TCompressionLevel indicates the type of compression algorithm to be used when creating a compressed data stream.

Pascal

```
TCompressionLevel = (
  clNone = Integer(zcNone
);
```

C++

```
enum TCompressionLevel {
  clNone = Integer(zcNone
};
```

File

ZLib

Description

The following table lists the possible values:

Value	Meaning
clNone	Specifies no compression; data is merely copied to the output stream.
clFastest	Specifies fastest compression, resulting in a larger archive.
clDefault	Compromise between speed and amount of compression.
clMax	Specifies maximum compression, resulting in a longer time to perform the operation.

See Also

TCustomZStream (see page 7905)

TZDecompressionStream (see page 7912)

TZCompressionStream (see page 7907)

88.3.2 ZLib.TZCompressionLevel Enumeration

Pascal

```
TZCompressionLevel = (
  zcNone,
  zcFastest,
  zcDefault,
  zcMax
);
```

C++

```
enum TZCompressionLevel {
  zcNone,
  zcFastest,
  zcDefault,
  zcMax
};
```

File

ZLib

Description

This is record ZLib.TZCompressionLevel.

88.3.3 ZLib.TZStreamRec Record

Pascal

```
TZStreamRec = packed record
  next_in: PAnsiChar;
  avail_in: Longint;
  total_in: Longint;
  next_out: PAnsiChar;
  avail_out: Longint;
  total_out: Longint;
  msg: PAnsiChar;
  state: Pointer;
  zalloc: TZAlloc;
  zfree: TZFree;
  opaque: Pointer;
  data_type: Integer;
  adler: Longint;
  reserved: Longint;
end;
```

C++

```
struct TZStreamRec {
  PAnsiChar next_in;
  Longint avail_in;
  Longint total_in;
  PAnsiChar next_out;
  Longint avail_out;
  Longint total_out;
  PAnsiChar msg;
  void * state;
```

```
    TZAlloc zalloc;
    TZFree zfree;
    void * opaque;
    int data_type;
    Longint adler;
    Longint reserved;
  };
```

File

ZLib

Description

This is record ZLib.TZStreamRec.

88.4 Types

The following table lists types in this documentation.

Types

Name	Description
TCompressionStream (see page 7929)	Compresses data as it is written to the stream.
TCustomZLibStream (see page 7929)	This is type ZLib.TCustomZLibStream.
TDecompressionStream (see page 7930)	Decompresses data that was written to a TZCompressionStream (see page 7907) object.
TZAlloc (see page 7930)	This is type ZLib.TZAlloc.
TZFree (see page 7930)	This is type ZLib.TZFree.

88.4.1 ZLib.TCompressionStream Type

Compresses data as it is written to the stream.

Pascal

```
TCompressionStream = TZCompressionStream;
```

C++

```
TZCompressionStream TCompressionStream;
```

File

ZLib

Description

TCompressionStream type is just an alias for TZCompressionStream (see page 7907) class.

See Also

TZCompressionStream (see page 7907)

TZDecompressionStream (see page 7912)

88.4.2 ZLib.TCustomZLibStream Type

Pascal

```
TCustomZLibStream = TCustomZStream;
```

C++

```
TCustomZStream TCustomZLibStream;
```

File

ZLib

Description

This is type ZLib.TCustomZLibStream.

88.4.3 ZLib.TDecompressionStream Type

Decompresses data that was written to a TZCompressionStream (see page 7907) object.

Pascal

```
TDecompressionStream = TZDecompressionStream;
```

C++

```
TZDecompressionStream TDecompressionStream;
```

File

ZLib

Description

TCompressionStream (see page 7929) type is just an alias for TZDecompressionStream (see page 7912) class.

See Also

TZDecompressionStream (see page 7912)

TZCompressionStream (see page 7907)

88.4.4 ZLib.TZAlloc Type

Pascal

```
TZAlloc = function (opaque: Pointer; items, size: Integer): Pointer;
```

C++

```
(opaque: Pointer; items, size: Integer): Pointer ( TZAlloc)();
```

File

ZLib

Description

This is type ZLib.TZAlloc.

88.4.5 ZLib.TZFree Type

Pascal

```
TZFree = procedure (opaque, block: Pointer);
```

C++

```
(opaque, block: Pointer) ( TZFree)();
```

File

ZLib

Description

This is type ZLib.TZFree.

88.5 Constants

The following table lists constants in this documentation.

Constants

Name	Description
_z_errmsg (see page 7932)	This is constant ZLib._z_errmsg.
SZInvalid (see page 7933)	This is constant ZLib.SZInvalid.
Z_ASCII (see page 7933)	This is constant ZLib.Z_ASCII.
Z_BEST_COMPRESSION (see page 7933)	This is constant ZLib.Z_BEST_COMPRESSION.
Z_BEST_SPEED (see page 7934)	This is constant ZLib.Z_BEST_SPEED.
Z_BINARY (see page 7934)	This is constant ZLib.Z_BINARY.
Z_BUF_ERROR (see page 7934)	This is constant ZLib.Z_BUF_ERROR.
Z_DATA_ERROR (see page 7935)	This is constant ZLib.Z_DATA_ERROR.
Z_DEFAULT_COMPRESSION (see page 7935)	This is constant ZLib.Z_DEFAULT_COMPRESSION.
Z_DEFAULT_STRATEGY (see page 7935)	This is constant ZLib.Z_DEFAULT_STRATEGY.
Z_DEFLATED (see page 7936)	This is constant ZLib.Z_DEFLATED.
Z_ERRNO (see page 7936)	This is constant ZLib.Z_ERRNO.
Z_FILTERED (see page 7936)	This is constant ZLib.Z_FILTERED.
Z_FINISH (see page 7937)	This is constant ZLib.Z_FINISH.
Z_FULL_FLUSH (see page 7937)	This is constant ZLib.Z_FULL_FLUSH.
Z_HUFFMAN_ONLY (see page 7937)	This is constant ZLib.Z_HUFFMAN_ONLY.
Z_MEM_ERROR (see page 7938)	This is constant ZLib.Z_MEM_ERROR.
Z_NEED_DICT (see page 7938)	This is constant ZLib.Z_NEED_DICT.
Z_NO_COMPRESSION (see page 7938)	This is constant ZLib.Z_NO_COMPRESSION.
Z_NO_FLUSH (see page 7939)	This is constant ZLib.Z_NO_FLUSH.
Z_OK (see page 7939)	This is constant ZLib.Z_OK.
Z_PARTIAL_FLUSH (see page 7939)	This is constant ZLib.Z_PARTIAL_FLUSH.
Z_STREAM_END (see page 7940)	This is constant ZLib.Z_STREAM_END.
Z_STREAM_ERROR (see page 7940)	This is constant ZLib.Z_STREAM_ERROR.
Z_SYNC_FLUSH (see page 7940)	This is constant ZLib.Z_SYNC_FLUSH.
Z_UNKNOWN (see page 7941)	This is constant ZLib.Z_UNKNOWN.
Z_VERSION_ERROR (see page 7941)	This is constant ZLib.Z_VERSION_ERROR.
ZLevels (see page 7941)	This is constant ZLib.ZLevels.
ZLIB_VERSION (see page 7942)	This is constant ZLib.ZLIB_VERSION.

88.5.1 ZLib._z_errmsg Constant

Pascal

```
_z_errmsg: array[0..9] of PChar = ( 'need dictionary', 'stream end', '', 'file error', 'stream
error', 'data error', 'insufficient memory', 'buffer error', 'incompatible version', '' );
```

C++

```
array[0..9] of PChar _z_errmsg = ( 'need dictionary', 'stream end', '', 'file error', 'stream
error', 'data error', 'insufficient memory', 'buffer error', 'incompatible version', '' );
```

File

ZLib

Description

This is constant ZLib._z_errmsg.

88.5.2 ZLib.SZInvalid Constant

Pascal

```
SZInvalid = 'Invalid ZStream operation!';
```

C++

```
SZInvalid = 'Invalid ZStream operation!';
```

File

ZLib

Description

This is constant ZLib.SZInvalid.

88.5.3 ZLib.Z_ASCII Constant

Pascal

```
Z_ASCII = 1;
```

C++

```
Z_ASCII = 1;
```

File

ZLib

Description

This is constant ZLib.Z_ASCII.

88.5.4 ZLib.Z_BEST_COMPRESSION Constant

Pascal

```
Z_BEST_COMPRESSION = 9;
```

C++

```
Z_BEST_COMPRESSION = 9;
```

File

ZLib

Description

This is constant ZLib.Z_BEST_COMPRESSION.

88.5.5 ZLib.Z_BEST_SPEED Constant

Pascal

```
Z_BEST_SPEED = 1;
```

C++

```
Z_BEST_SPEED = 1;
```

File

ZLib

Description

This is constant ZLib.Z_BEST_SPEED.

88.5.6 ZLib.Z_BINARY Constant

Pascal

```
Z_BINARY = 0;
```

C++

```
Z_BINARY = 0;
```

File

ZLib

Description

This is constant ZLib.Z_BINARY.

88.5.7 ZLib.Z_BUF_ERROR Constant

Pascal

```
Z_BUF_ERROR = (-5);
```

C++

```
Z_BUF_ERROR = (-5);
```

File

ZLib

Description

This is constant ZLib.Z_BUF_ERROR.

88.5.8 ZLib.Z_DATA_ERROR Constant

Pascal

```
Z_DATA_ERROR = (-3);
```

C++

```
Z_DATA_ERROR = (-3);
```

File

ZLib

Description

This is constant ZLib.Z_DATA_ERROR.

88.5.9 ZLib.Z_DEFAULT_COMPRESSION Constant

Pascal

```
Z_DEFAULT_COMPRESSION = (-1);
```

C++

```
Z_DEFAULT_COMPRESSION = (-1);
```

File

ZLib

Description

This is constant ZLib.Z_DEFAULT_COMPRESSION.

88.5.10 ZLib.Z_DEFAULT_STRATEGY Constant

Pascal

```
Z_DEFAULT_STRATEGY = 0;
```

C++

```
Z_DEFAULT_STRATEGY = 0;
```

File

ZLib

Description

This is constant ZLib.Z_DEFAULT_STRATEGY.

88.5.11 ZLib.Z_DEFLATED Constant

Pascal

```
Z_DEFLATED = 8;
```

C++

```
Z_DEFLATED = 8;
```

File

ZLib

Description

This is constant ZLib.Z_DEFLATED.

88.5.12 ZLib.Z_ERRNO Constant

Pascal

```
Z_ERRNO = (-1);
```

C++

```
Z_ERRNO = (-1);
```

File

ZLib

Description

This is constant ZLib.Z_ERRNO.

88.5.13 ZLib.Z_FILTERED Constant

Pascal

```
Z_FILTERED = 1;
```

C++

```
Z_FILTERED = 1;
```

File

ZLib

Description

This is constant ZLib.Z_FILTERED.

88.5.14 ZLib.Z_FINISH Constant

Pascal

```
Z_FINISH = 4;
```

C++

```
Z_FINISH = 4;
```

File

ZLib

Description

This is constant ZLib.Z_FINISH.

88.5.15 ZLib.Z_FULL_FLUSH Constant

Pascal

```
Z_FULL_FLUSH = 3;
```

C++

```
Z_FULL_FLUSH = 3;
```

File

ZLib

Description

This is constant ZLib.Z_FULL_FLUSH.

88.5.16 ZLib.Z_HUFFMAN_ONLY Constant

Pascal

```
Z_HUFFMAN_ONLY = 2;
```

C++

```
Z_HUFFMAN_ONLY = 2;
```

File

ZLib

Description

This is constant ZLib.Z_HUFFMAN_ONLY.

88.5.17 ZLib.Z_MEM_ERROR Constant

Pascal

```
Z_MEM_ERROR = (-4);
```

C++

```
Z_MEM_ERROR = (-4);
```

File

ZLib

Description

This is constant ZLib.Z_MEM_ERROR.

88.5.18 ZLib.Z_NEED_DICT Constant

Pascal

```
Z_NEED_DICT = 2;
```

C++

```
Z_NEED_DICT = 2;
```

File

ZLib

Description

This is constant ZLib.Z_NEED_DICT.

88.5.19 ZLib.Z_NO_COMPRESSION Constant

Pascal

```
Z_NO_COMPRESSION = 0;
```

C++

```
Z_NO_COMPRESSION = 0;
```

File

ZLib

Description

This is constant ZLib.Z_NO_COMPRESSION.

88.5.20 ZLib.Z_NO_FLUSH Constant

Pascal

```
Z_NO_FLUSH = 0;
```

C++

```
Z_NO_FLUSH = 0;
```

File

ZLib

Description

This is constant ZLib.Z_NO_FLUSH.

88.5.21 ZLib.Z_OK Constant

Pascal

```
Z_OK = 0;
```

C++

```
Z_OK = 0;
```

File

ZLib

Description

This is constant ZLib.Z_OK.

88.5.22 ZLib.Z_PARTIAL_FLUSH Constant

Pascal

```
Z_PARTIAL_FLUSH = 1;
```

C++

```
Z_PARTIAL_FLUSH = 1;
```

File

ZLib

Description

This is constant ZLib.Z_PARTIAL_FLUSH.

88.5.23 ZLib.Z_STREAM_END Constant

Pascal

```
Z_STREAM_END = 1;
```

C++

```
Z_STREAM_END = 1;
```

File

ZLib

Description

This is constant ZLib.Z_STREAM_END.

88.5.24 ZLib.Z_STREAM_ERROR Constant

Pascal

```
Z_STREAM_ERROR = (-2);
```

C++

```
Z_STREAM_ERROR = (-2);
```

File

ZLib

Description

This is constant ZLib.Z_STREAM_ERROR.

88.5.25 ZLib.Z_SYNC_FLUSH Constant

Pascal

```
Z_SYNC_FLUSH = 2;
```

C++

```
Z_SYNC_FLUSH = 2;
```

File

ZLib

Description

This is constant ZLib.Z_SYNC_FLUSH.

88.5.26 ZLib.Z_UNKNOWN Constant

Pascal

```
Z_UNKNOWN = 2;
```

C++

```
Z_UNKNOWN = 2;
```

File

ZLib

Description

This is constant ZLib.Z_UNKNOWN.

88.5.27 ZLib.Z_VERSION_ERROR Constant

Pascal

```
Z_VERSION_ERROR = (-6);
```

C++

```
Z_VERSION_ERROR = (-6);
```

File

ZLib

Description

This is constant ZLib.Z_VERSION_ERROR.

88.5.28 ZLib.ZLevels Constant

Pascal

```
ZLevels: array[TZCompressionLevel] of Shortint = ( Z_NO_COMPRESSION, Z_BEST_SPEED,
Z_DEFAULT_COMPRESSION, Z_BEST_COMPRESSION );
```

C++

```
array[TZCompressionLevel] of Shortint ZLevels = ( Z_NO_COMPRESSION, Z_BEST_SPEED,
Z_DEFAULT_COMPRESSION, Z_BEST_COMPRESSION );
```

File

ZLib

Description

This is constant ZLib.ZLevels.

88.5.29 ZLib.ZLIB_VERSION Constant

Pascal

```
ZLIB_VERSION = '1.2.3';
```

C++

```
ZLIB_VERSION = '1.2.3';
```

File

ZLib

Description

This is constant ZLib.ZLIB_VERSION.

Index

=

-
- Currency class 3239
- TDateTime class 3334
- Variant class 3439

--
- Currency class 3240
- TDateTime class 3334

!
- Currency class 3240
- DelphiInterface template 3259
- Variant class 3439

!=
- AnsiStringBase class 3204
- AnsiStringT template 3232
- Currency class 3241
- OleVariant class 3302
- TDateTime class 3335
- UnicodeString class 3414
- Variant class 3440
- WideString class 3488

%
- Currency class 3241
- Variant class 3441

%=
- Currency class 3242
- Variant class 3441

&
- DelphiInterface template 3259
- sPointer template 3313
- Variant class 3442

&=
- Variant class 3442

*
- Currency class 3242
- DelphiInterface template 3259
- Variant class 3443

*=
- Currency class 3243
- Variant class 3443

/
- Currency class 3243
- Variant class 3444

/=
- Currency class 3244
- Variant class 3444

[]
- AnsiStringBase class 3205
- DynamicArray template 3269
- SmallString template 3309
- UnicodeString class 3414
- WideString class 3488

^
- Variant class 3445

^=
- Variant class 3445

__int64
- Variant class 3446

_AddRef
- IInterface interface 3503
- TAggregatedObject class 3316
- TCustomVariantType class 4993
- THTTPReqResp class 1279
- TInterfacedObject class 3345
- TLayoutWebContent class 6017
- TSoapDataModule class 1210
- TSOAPDOMProcessor class 1236
- TStreamedConnection class 239
- TWebComponentList class 6028
- TWebComponentsEditorHelper class 6037
- TXMLDocument class 7354
- TXMLNode class 7403

_AnsiCat
- AnsiStringBase class 3184

_AnsiFromPWChar

AnsiStringBase class 3184
_Create
TClassHelperBase class 3319
_NewEnum
ISearches interface 687
ISearchesDisp interface 689
IShellWindows interface 726
IShellWindowsDisp interface 730
TShellWindows class 498
_Release
IInterface interface 3503
TAggregatedObject class 3316
TCustomVariantType class 4993
THTTPReqResp class 1280
TInterfacedObject class 3345
TLayoutWebContent class 6017
TSoapDataModule class 1210
TSOAPDOMProcessor class 1236
TStreamedConnection class 239
TWebComponentList class 6028
TWebComponentsEditorHelper class 6037
TXMLDocument class 7354
TXMLNode class 7404
_SearchAssistantEvents interface 616
_SearchAssistantEvents Members 617
_SearchAssistantEvents methods 617
about _SearchAssistantEvents interface 616
OnNewSearch 617
OnNextMenuSelect 617
|
Variant class 3446
|=
Variant class 3446
~AnsiStringBase
AnsiStringBase class 3178
~AnsiStringT
AnsiStringT template 3211
~DelphiInterface
DelphiInterface template 3257
~DynamicArray
DynamicArray template 3263
~OleVariant
OleVariant class 3279
~sPointer
sPointer template 3312
~UnicodeString
UnicodeString class 3384
~Variant
Variant class 3424
~WideString
WideString class 3469
+
AnsiStringBase class 3205
AnsiStringT template 3232
Currency class 3245
TDateTime class 3335
UnicodeString class 3415
Variant class 3447
WideString class 3489
++
Currency class 3246
TDateTime class 3336
+=
AnsiStringT template 3232
Currency class 3246
TDateTime class 3337
UnicodeString class 3416
Variant class 3447
WideString class 3489
<
AnsiStringBase class 3205
AnsiStringT template 3233
Currency class 3247
OleVariant class 3302, 3303
TDateTime class 3337
UnicodeString class 3416
Variant class 3448
WideString class 3489

<<
Variant class 3449
<<=
Variant class 3449
<=
AnsiStringBase class 3206
AnsiStringT template 3233
Currency class 3248
OleVariant class 3303
TDateTime class 3337
UnicodeString class 3417
Variant class 3450
WideString class 3490
=
AnsiStringBase class 3206
AnsiStringT template 3233
Currency class 3248
DelphiInterface template 3260
DynamicArray template 3269
OleVariant class 3304
SmallString template 3309
TDateTime class 3338
UnicodeString class 3418
Variant class 3453
WideString class 3490
-=
Currency class 3249
TDateTime class 3339
Variant class 3454
==
AnsiStringBase class 3206
AnsiStringT template 3234
Currency class 3250
DynamicArray template 3269
OleVariant class 3305, 3306
TDateTime class 3339
UnicodeString class 3418
Variant class 3455
WideString class 3491
>
AnsiStringBase class 3207
AnsiStringT template 3234
Currency class 3250
OleVariant class 3306
TDateTime class 3339
UnicodeString class 3419
Variant class 3456
WideString class 3491
->
DelphiInterface template 3260
sPointer template 3313

>

>=
AnsiStringBase class 3207
AnsiStringT template 3234
Currency class 3251
OleVariant class 3307
TDateTime class 3340
UnicodeString class 3419
Variant class 3457
WideString class 3492
>>
Variant class 3457
>>=
Variant class 3458

A

Accept
TCustomTcpServer class 1409
TCustomWinSocket class 68
TServerWinSocket class 124
Access_Request
TMultiModuleWebAppServices class 6225
Access_Response
TMultiModuleWebAppServices class 6225
AccessRights
TWebUserItem class 6790

AccessRightsStrings
TWebUserItem class 6791
Acquire
TConditionVariableCS class 3115
TConditionVariableMutex class 3123
TCriticalSection class 3128
TMutex class 3146
TSemaphore class 3151
TSynchroObject class 3154
Action
TAdapterActionName class 6538
TButton class 2039
TCheckBox class 2119
TRadioButton class 2818
TScreenTipItem class 332
ActionChange
TButtonControl class 2110
TCustomButton class 2262
ActionList1
TSocketForm class 161
ActionName
IWebGetActionName interface 1087
TAdapterAction class 5205
TAdapterActionButton class 6318
TAdapterActionName class 6539
TBaseAdapterAction class 5474
TCustomAdapterAction class 5586
TCustomAdapterActionButton class 6438
ActionOrigValue
TAdapterNamedField class 5367
Actions
TAdapter class 5195
TApplicationAdapter class 5444
TCustomAdapter class 5572
TDefaultFieldsAdapter class 5746
TEndUserAdapter class 5770
TEndUserSessionAdapter class 5799
TLoginFormAdapter class 5830
TPagedAdapter class 5850
TStringsValuesList class 5862
ActionValue
TAdapterNamedField class 5368
Activate
TWSDLView class 7273
ActivateToolTip
TScreenTipsWindow class 353
ActivateWebModules
TWebRequestHandler class 6636
Active
IXMLDocument interface 7661
TAbstractSocket class 21
TBaseSocket class 1383
TClientSocket class 28
TCustomSQLDataSet class 1552
TServerSocket class 111
TSimpleDataSet class 904
TSQLMonitor class 1649
TSQLTable class 1687
TTcpClient class 1443
TTcpServer class 1463
TUdpSocket class 1480
TXMLDocument class 7381
ActiveChanged
TSQLDataLink class 1630
ActiveConnections
TServerWinSocket class 129
ActiveCount
TSessions class 394
TWebRequestHandler class 6638
ActiveItem
TScreenTipsWindow class 357
ActivePage
TTabbedNotebook class 4413
ActiveStatements
TSQLConnection class 1607
ActiveThreads
TServerWinSocket class 130
Adapter

TAdapterActionName class 6539
TAdapterApplicationField class 5218
TAdapterControlGroup class 6352
TAdapterEndUserField class 5251
TAdapterErrorList class 6398
TAdapterFieldGroup class 6402
TAdapterGrid class 6422
TAdapterLoginField class 5323
TAdapterNamedField class 5368
TAdapterSubComponent class 5418
TBaseAdapterAction class 5474
TBaseValuesListField class 5531
TCustomAdapterCommandGroup class 6459
TCustomAdapterErrorList class 6482
TCustomApplicationAdapterAction class 5676
TCustomEndUserAdapterAction class 5692
TCustomLoginFormAdapterAction class 5721
TCustomPagedAdapterAction class 5727
AdapterActionsListHelper
TCustomAdapterCommandColumn class 6449
TCustomAdapterCommandGroup class 6459
AdapterCanAddClass
TAdapterActions class 5212
TAdapterFields class 5278
AdapterChange
TAdapterControlGroup class 6345
TAdapterDisplayReferenceGroup class 6383
AdapterDispatcher
TCustomWebAppComponents class 6211
TWebAppComponents class 6245
AdapterMode
IAdapterMode interface 1011
TAdapterControlGroup class 6353
TAdapterFieldGroup class 6402
TAdapterGrid class 6423
AdapterRequest
TAbstractWebContext class 5971
AdapterResponse
TAbstractWebContext class 5972
AdapterVariableName
IGetAdapterVariableName interface 6515
adCantView enumeration member 6265
Add
IBindingOperations interface 7089
IBindings interface 7093
IDocumentationCollection interface 7103
IFaults interface 7109
IImports interface 7115
IIntegerTest interface 1333
IMessages interface 7122
IOperations interface 7129, 7130
IParts interface 7141
IPorts interface 7147
IPortTypes interface 7154
IServices interface 7160
IXMLNodeList interface 7705
IXMLSchemaDefs interface 7170
TAbstractNamedVariants class 6121
TAdapterHiddenFieldsList class 5305
TAssociateItemList class 304
TBindingOperations class 6922
TBindings class 6929
TDocumentationCollection class 6944
TDOMStrings class 7182
TDOMVendorList class 7517
TFaults class 6957
TFTCompTI class 1315
TGeneratedFunctions class 6507
TImports class 6970
TMessages class 6984
TOperations class 6997, 6998
TParts class 7017
TPorts class 7030
TPortTypes class 7044
TScreenTipsCollection class 335
TScriptErrors class 6683
TServices class 7057
TSoapDataList class 1173

TTabList class 4464
TWebComponentList class 6029
TWebNotifyContainer class 6602
TWideStringList class 6812
TWideStrings class 6829
TXMLNodeList class 7464
TXMLSchemaDefs class 7076
AddAction
TAllocatedActions class 299
AddAdapterModeHiddenField
TCustomAdapter class 5538
AddAttributes
TCustomAdapterDisplayField class 6470
AddChannel
IShellUIHelper interface 718
IShellUIHelperDisp interface 722
TShellUIHelper class 486
AddChild
IXMLDocument interface 7643
IXMLNode interface 7672
IXMLNodeAccess interface 7482
TXMLDocument class 7355
TXMLNode class 7404
AddClient
TSocketForm class 174
AddClientSocketThread
TServerSocketThread class 1436
AddConnectNotification
TSQLConnection class 1585
AddDefaultActions
TAdapterActionsListHelper class 6331
TDefaultFieldsAdapter class 5743
TDefaultFieldsPagedAdapter class 5753
AddDefaultCommands
TAdapterCommandColumn class 6334
TAdapterCommandGroup class 6340
TCustomAdapterCommandColumn class 6449
TCustomAdapterCommandGroup class 6459
AddDefaultFields
TAdapterControlGroup class 6353
TAdapterFieldGroup class 6403
TAdapterGrid class 6423
TDefaultFieldsAdapter class 5744
AddDesktopComponent
IShellUIHelper interface 718
IShellUIHelperDisp interface 722
TShellUIHelper class 487, 488
AddDispatch
TCustomDataBlockInterpreter class 194
TDataBlockInterpreter class 209
AddEngine
TScriptEnginesList class 6675
AddError
TAdapterErrorsList class 5263
TBaseAdapterErrorsList class 5482
AddFactory
TWebModuleFactoryList class 6624
AddFavorite
IShellUIHelper interface 718
IShellUIHelperDisp interface 722
TShellUIHelper class 488, 489
AddFieldDesc
TCustomSQLDataSet class 1527
AddGlobalObjects
TAbstractScriptObjectFactory class 6656
AddHeaderProcessor
TSOAPDOMProcessor class 1237
AddHostedNode
TXMLNode class 7405
AddIcon
TSocketForm class 174
AddIndexDefs
TCustomSQLDataSet class 1527
AddInterfaceList
TWSDLHTMLPublish class 7283
AddItem
TCustomCombo class 2291
TCustomListBox class 2402

TXMLNodeCollection class 7451
AddModule
TWebModuleList class 6628
AddModuleClass
TWebModuleList class 6629
AddModuleName
TWebModuleList class 6629
AddNamedIntf
TScriptGlobalObjects class 6690
AddNextMenuItem
ISearchAssistantOC interface 648
ISearchAssistantOC2Disp interface 660
ISearchAssistantOC3Disp interface 671
ISearchAssistantOCDisp interface 679
AddNotify
INotifyList interface 1059
TAdapterControlGroup class 6345
TNotifyList class 970
TXMLBroker class 7307
AddObject
TAbstractDesigntimeWarnings class 964
TDesigntimeWarnings class 6505
TWideStringList class 6813
TWideStrings class 6830
AddObjectFactory
TAbstractScriptEngine class 6650
TScriptEnginesList class 6675
AddOrigValueHiddenFields
TCustomAdapter class 5539
AddPopup
TCustomScreenTipsManager class 308
AddPortList
TWSDLHTMLPublish class 7283
AddProperties
TConnectionData class 1517
Addr
TCustomWinSocket class 82
AddRecordKeysHiddenFields
TCustomAdapter class 5539
Address
TAbstractSocket class 22
TClientSocket class 28
TSocketConnection class 225
TSocketTransport class 234
AddressBar
IWebBrowser2 interface 760
IWebBrowser2Disp interface 769
TInternetExplorer class 449
TWebBrowser class 520
AddSoapHeader
IMimeAttachmentHandler interface 1177
AddSpecialParams
TCustomAdapterDispatcher class 6170
AddString
TAbstractDesigntimeWarnings class 964
TDesigntimeWarnings class 6505
AddStrings
TTabList class 4467
TWideStrings class 6830
AddWebModuleFactory
TWebRequestHandler class 6636
AdjustBounds
TCustomLabel class 2385
AdjustClientRect
TCustomGroupBox class 2382
AdjustColWidths
TValueListEditor class 4789
AdjustDockRect
TToolDockObject class 4638
AdjustDropDown
TCustomCombo class 2292
AdjustRowCount
TValueListEditor class 4789
adLoginRequired enumeration member 6265
AdminEnabled
TWSDLHTMLPublish class 7287
advanceError
IScriptErrorList interface 636

IScriptErrorListDisp interface 640
AfterApplyUpdates
TSimpleDataSet class 931
TXMLTransformProvider class 7754
AfterCancel
TSimpleDataSet class 931
AfterClose
TCustomSQLDataSet class 1568
TSimpleDataSet class 933
TXMLDocument class 7393
AfterConnect
TSoapConnection class 1201
TSQLConnection class 1626
AfterConstruction
TBinding class 6908
TBindingOperation class 6915
TDefinition class 6935
TDocumented class 6950
THTTPReqResp class 1280
TInterfacedObject class 3345
TMessage class 6977
TObject class 3357
TOperation class 6990
TPortType class 7037
TService class 7050
TSoapDataModule class 1210
TSOAPDOMProcessor class 1237
TTypes class 7063
TWSDLDocument class 7069
TXMLDocument class 7355
TXMLNodeCollection class 7451
AfterDelete
TSimpleDataSet class 935
AfterDisconnect
TSoapConnection class 1201
TSQLConnection class 1626
AfterDispatch
TXMLBroker class 7329
AfterDispatchEvent
TSoapPascalInvoker class 1356
AfterEachRow
TXMLTransform class 7737
AfterEachRowSet
TXMLTransform class 7737
AfterEdit
TSimpleDataSet class 937
AfterExecute
TSimpleDataSet class 938
AfterGetParams
TSimpleDataSet class 939
AfterGetRecords
TSimpleDataSet class 940
TXMLTransformProvider class 7755
AfterInsert
TSimpleDataSet class 940
AfterInstall
TService class 3085
AfterNodeChange
TXMLDocument class 7393
AfterOpen
TCustomSQLDataSet class 1568
TFileOpenWith class 1834
TSimpleDataSet class 941
TXMLDocument class 7394
AfterPost
TSimpleDataSet class 942
AfterRefresh
TCustomSQLDataSet class 1568
TSimpleDataSet class 943
AfterRowRequest
TSimpleDataSet class 943
TXMLTransformProvider class 7756
AfterScroll
TCustomSQLDataSet class 1568
TSimpleDataSet class 944
AfterUninstall
TService class 3086
Agent

THTTPReqResp class 1288
TSoapConnection class 1196
TWebConnection class 262
Aggregates
TSimpleDataSet class 906
AggregatesActive
TSimpleDataSet class 907
Align
TAdapterActionButton class 6318
TAdapterCommandColumn class 6334
TAdapterDataDisplay class 6362
TAdapterDisplayColumn class 6366
TAdapterDisplayField class 6374
TAdapterEditColumn class 6389
TButton class 2040
TCheckBox class 2120
TComboBox class 2190
TCustomAdapterActionButton class 6438
TCustomAdapterCommandColumn class 6450
TEdit class 2467
TGroupBox class 2537
TLabel class 2610
TListBox class 2662
TMemo class 2741
TRadioButton class 2820
TScreenTipsPopup class 344
TScrollBar class 2886
TStaticText class 2948
TTabbedNotebook class 4414
TTabSet class 4481
TValueListEditor class 4802
TWebBrowser class 520
TWebBrowser_V1 class 577
AlignControls
TTabbedNotebook class 4407
Alignment
TCheckBox class 2123
TCustomCheckBox class 2281
TCustomEdit class 2365
TCustomLabel class 2388
TCustomStaticText class 2461
TEdit class 2471
TLabel class 2613
TMemo class 2745
TRadioButton class 2823
TStaticText class 2951
AllocConnection
TSimpleDataSet class 900
AllocData
DynamicArray template 3265
AllocDataSet
TSimpleDataSet class 900
AllocProvider
TSimpleDataSet class 901
AllowGrayed
TCheckBox class 2124
TCustomCheckBox class 2281
AllowPause
TService class 3078
AllowStop
TService class 3078
AllowXML
TSocketForm class 161
AllowXMLExecute
TSocketForm class 174
Anchors
TButton class 2044
TCheckBox class 2124
TComboBox class 2193
TEdit class 2471
TGroupBox class 2541
TLabel class 2614
TListBox class 2665
TMemo class 2745
TRadioButton class 2823
TScreenTipsPopup class 347
TScrollBar class 2890
TStaticText class 2952

TTabbedNotebook class 4417
TTabSet class 4485
TValueListEditor class 4805
TWebBrowser class 523
TWebBrowser_V1 class 581
AnsiCompare
AnsiStringBase class 3185
AnsiStringT template 3214
AnsiCompareIC
AnsiStringBase class 3185
AnsiStringT template 3215
AnsiLastChar
AnsiStringBase class 3186
AnsiPos
AnsiStringBase class 3186
AnsiStringT template 3215
AnsiString
SmallString template 3310
Variant class 3458
AnsiStringBase
AnsiStringBase class 3179
AnsiStringBase class 3174
!= 3204
[] 3205
[] operator 3204
_AnsiCat 3184
_AnsiCat method 3184
_AnsiFromPWChar 3184
~AnsiStringBase 3178
+ 3205
< 3205
<= 3206
= 3206
== 3206
> 3207
>= 3207
about AnsiStringBase class 3174
AnsiCompare 3185
AnsiCompareIC 3185
AnsiLastChar 3186
AnsiPos 3186
AnsiStringBase 3179
AnsiStringBase constructor 3178
AnsiStringBase data members 3182
AnsiStringBase enumerations 3180
AnsiStringBase Members 3176
AnsiStringBase methods 3182
AnsiStringBase operators 3204
AnsiStringBase structures 3179
BytesOf 3186
ByteType 3187
cat_vprintf 3187
CodePage 3188
CurrToStr 3188
CurrToStrF 3188
Data 3182
Delete 3189
ElementSize 3189
EnsureAnsi 3189
FloatToStrF 3190
FmtLoadStr 3190
Format 3191
FormatFloat 3191
GetRec 3193
GetRec method 3193
Insert 3193
IntToHex 3194
IsDelimiter 3194
IsEmpty 3194
IsLeadByte 3195
IsPathDelimiter 3195
IsTrailByte 3195
LastChar 3196
LastDelimiter 3196
Length 3196
LoadStr 3197
LoadStr method 3196
LoadString 3197

LowerCase 3198
Pos 3198
RefCount 3198
SetLength 3199
StringOfChar 3199
SubString 3199
swap 3200
ThrowIfOutOfRange 3200
ToDouble 3200
ToInt 3200
ToIntDef 3201
Trim 3201
TrimLeft 3201
TrimRight 3202
Unique 3202
UpperCase 3202
vprintf 3202
WideChar 3203
WideCharBufSize 3203
AnsiStringT
AnsiStringT template 3212
AnsiStringT template 3207
!= 3232
~AnsiStringT 3211
+ 3232
+= 3232
< 3233
<= 3233
= 3233
== 3234
> 3234
>= 3234
about AnsiStringT template 3207
AnsiCompare 3214
AnsiCompareIC 3215
AnsiPos 3215
AnsiStringT 3212
AnsiStringT constructor 3211
AnsiStringT Members 3209
AnsiStringT methods 3213
AnsiStringT operators 3231
c_str 3216
cat_printf 3216
cat_sprintf 3216
cat_vprintf 3217
CodePage 3217
CurrToStr 3218
CurrToStrF 3218
data 3218
Delete 3219
ElementSize 3219
EnsureAnsi 3220
FloatToStrF 3220
FmtLoadStr 3221
Format 3221
FormatFloat 3222
Insert 3223
IntToHex 3224
IsDelimiter 3224
LastDelimiter 3224
LoadStr 3225
LoadStr method 3224
LoadString 3225
LowerCase 3226
Pos 3226
printf 3226
RefCount 3227
SetLength 3227
sprintf 3227
StringOfChar 3228
SubString 3228
swap 3228
Trim 3229
TrimLeft 3229
TrimRight 3229
Unique 3229
UpperCase 3230
vprintf 3230

WideChar 3230
WideCharBufSize 3231
AppDispatcher
TBaseRequestHandler class 6168
Append
IStringTest interface 1335
TAbstractNamedVariants class 6122
TFTCompTI class 1315
TStringBuilder class 4014
TWideStrings class 6831
appendChild
IDOMNode interface 7559
appendData
IDOMCharacterData interface 7527
AppendFormat
TStringBuilder class 4015
AppendLine
TStringBuilder class 4016
Application
IWebBrowser interface 745
IWebBrowser2Disp interface 769
IWebBrowserAppDisp interface 790
IWebBrowserDisp interface 798
TInternetExplorer class 449
TWebBrowser class 524
TWebBrowser_V1 class 581
ApplicationAdapter
IWebApplicationInfo interface 1080
TCustomWebAppComponents class 6212
TWebAppComponents class 6246
ApplicationName
TScriptFile class 6686
ApplicationTitle
IWebApplicationInfo interface 1080
TApplicationAdapter class 5444
TCustomApplicationAdapter class 5671
ApplyAction
TSocketForm class 162
ApplyActionExecute
TSocketForm class 174
ApplyActionUpdate
TSocketForm class 174
ApplyButton
TSocketForm class 162
ApplyThemeChange
TThemeServices class 4582
ApplyUpdates
TSoapDataModule class 1211
TXMLTransformClient class 7742
ApplyXMLUpdates
TXMLBroker class 7308
AppModuleFactory
TWebModuleFactoryList class 6625
AppServer
TLocalConnection class 4553
TXMLBroker class 7322
AppServices
TCustomWebAppDataModule class 6544
TCustomWebAppPageModule class 6549
TWebAppDataModule class 6587
TWebAppPageModule class 6591
argIn enumeration member 6740
argInOut enumeration member 6740
argOut enumeration member 6740
argReturn enumeration member 6740
ArrayAsComplexContent
TWebServExp class 6731
ArrayDimCount
Variant class 3429
ArrayHighBound
Variant class 3429
ArrayLock
Variant class 3429
ArrayLowBound
Variant class 3430
ArrayRedim
Variant class 3430
ArrayUnlock

Variant class 3430
AS_ApplyUpdates
TLocalConnection class 4549
TSoapDataModule class 1211
TXMLBroker class 7308
AS_DataRequest
TLocalConnection class 4549
TSoapDataModule class 1212
AS_Execute
TLocalConnection class 4549
TSoapDataModule class 1213
AS_FetchParams
TXMLBroker class 7309
AS_GetParams
TLocalConnection class 4549
TSoapDataModule class 1214
AS_GetProviderNames
TLocalConnection class 4550
TSoapDataModule class 1215
AS_GetRecords
TLocalConnection class 4550
TSoapDataModule class 1215
TXMLBroker class 7309
AS_RowRequest
TLocalConnection class 4550
TSoapDataModule class 1216
asAccept enumeration member 145
AsBcd
TXSDecimal class 7835
AsBoolean
TXSBoolean class 7803
AsByteArray
TXSHexBinary class 7842
ASCII
TEncoding class 3985
asClose enumeration member 145
asConnect enumeration member 145
AsDate
TXSDate class 7827
AsDateTime
TXSCustomDateTime class 7815
AsFormatted
IAsFormatted interface 1014
TAdapterNamedField class 5355
AsInteger
TXSInteger class 7845
AsLong
TXSLong class 7847
asOOB enumeration member 145
ASProvider
ISearchAssistantOC interface 654
ISearchAssistantOC2Disp interface 664
ISearchAssistantOC3Disp interface 675
ISearchAssistantOCDisp interface 683
asRead enumeration member 145
ASSetting
ISearchAssistantOC interface 655
ISearchAssistantOC2Disp interface 664
ISearchAssistantOC3Disp interface 675
ISearchAssistantOCDisp interface 683
Assign
TAbstractNamedVariants class 6122
TImageMargins class 2604
TScreenTipItem class 330
TSessionItem class 380
TSoapDataList class 1173
TValueListStrings class 4884
TWebUserItem class 6789
TWideStrings class 6831
AssignClient
TButtonActionLink class 2108
AssignParseOptions
TXMLDocument class 7356
AssignString
THTMLItem class 6664
TScriptCodeItem class 6672
AssignStrings
TXMLParams class 7335

AssignTo
TAdapterActionName class 6537
TItemProp class 4780
TWideStrings class 6832
TXMLParams class 7335
Associate
TAssociateItem class 302
TCustomScreenTipsPopup class 321
TScreenTipsPopup class 348
AsTime
TXSTime class 7856
AsType
Variant class 3431
AsUTCDateTime
TXSCustomDateTime class 7816
asWrite enumeration member 145
async
IDOMParseOptions interface 7594
AsyncInitSocket
TCustomWinSocket class 68
asyncLoadState
IDOMPersist interface 7596
AsyncLoadState
IXMLDocument interface 7662
TXMLDocument class 7382
ASyncStyles
TCustomWinSocket class 83
AttributeListNotify
TXMLNode class 7406
AttributeNodes
IXMLNode interface 7688
TXMLNode class 7438
attributes
IDOMNode interface 7568
Attributes
IXMLNode interface 7688
AutoCalcFields
TSimpleDataSet class 907
AutoClone
TSQLConnection class 1608
AutoCloseUp
TComboBox class 2194
TCustomComboBox class 2332
AutoComplete
TComboBox class 2194
TCustomComboBox class 2332
TCustomListBox class 2423
TListBox class 2666
AutoCompleteAttach
IShellUIHelper interface 718
IShellUIHelperDisp interface 722
TShellUIHelper class 489
AutoCompleteDelay
TComboBox class 2195
TCustomComboBox class 2333
TCustomListBox class 2423
TListBox class 2666
AutoCompleteSaveForm
IShellUIHelper interface 719
IShellUIHelperDisp interface 723
TShellUIHelper class 490
AutoCreateModules
TWebModuleList class 6630
AutoDestroyModules
TWebModuleList class 6630
AutoDropDown
TComboBox class 2195
TCustomComboBox class 2334
AutoSave
TSQLMonitor class 1650
AutoScan
IShellUIHelper interface 719
IShellUIHelperDisp interface 723
TShellUIHelper class 490, 491
AutoScroll
TTabSet class 4485
AutoSelect
TCustomEdit class 2365

TEdit class 2472
AutoSize
TCustomEdit class 2366
TCustomLabel class 2389
TCustomStaticText class 2461
TEdit class 2472
TLabel class 2614
TStaticText class 2952

B

BackgroundColor
TTabSet class 4486
Base64ToVar
TTypeTranslator class 4682
BaseException
Exception class 3964
bCalcField
TFLDDesc class 1575
BeforeApplyUpdates
TSimpleDataSet class 944
TXMLTransformProvider class 7756
BeforeCancel
TSimpleDataSet class 945
BeforeClose
TCustomSQLDataSet class 1569
TSimpleDataSet class 946
TXMLDocument class 7394
BeforeConnect
TSoapConnection class 1202
TSQLConnection class 1626
BeforeDelete
TSimpleDataSet class 946
BeforeDestruction
TInterfacedObject class 3346
TObject class 3358
TSoapDataModule class 1217
BeforeDisconnect
TSoapConnection class 1202
TSQLConnection class 1627
BeforeDispatch
TXMLBroker class 7330
BeforeDispatchEvent
TSoapPascalInvoker class 1357
BeforeDispatchEvent2
TSoapPascalInvoker class 1357
BeforeEachRow
TXMLTransform class 7738
BeforeEachRowSet
TXMLTransform class 7738
BeforeEdit
TSimpleDataSet class 947
BeforeExecute
IWebNode interface 6617
TBrowseForFolder class 1772
TColorSelect class 1781
TCommonDialogAction class 1790
TFileOpen class 1826
TFileOpenWith class 1834
TFilePageSetup class 1842
TFilePrintSetup class 1852
TFileSaveAs class 1861
TFontEdit class 1871
THTTPReqResp class 1280
TLinkedWebNode class 1323
TPrintDlg class 1891
TSearchFind class 1906
TSearchReplace class 1929
TSimpleDataSet class 947
TWSDLClientNode class 7267
BeforeGetParams
TSimpleDataSet class 948
BeforeGetRecords
TSimpleDataSet class 949
TXMLTransformProvider class 7757
BeforeInsert
TSimpleDataSet class 950
BeforeInstall
TService class 3086

BeforeNavigate
DWebBrowserEvents interface 622
BeforeNavigate2
DWebBrowserEvents2 interface 628
BeforeNodeChange
TXMLDocument class 7394
BeforeOpen
TCustomSQLDataSet class 1569
TSimpleDataSet class 952
TXMLDocument class 7395
BeforePost
TSimpleDataSet class 952
BeforeRefresh
TCustomSQLDataSet class 1569
TSimpleDataSet class 953
BeforeRowRequest
TSimpleDataSet class 953
TXMLTransformProvider class 7758
BeforeScroll
TCustomSQLDataSet class 1569
TSimpleDataSet class 954
BeforeUninstall
TService class 3087
BeginRead
IReadWriteSync interface 4038
TMultiReadExclusiveWriteSynchronizer class 4000
TSimpleRWSync class 4005
BeginTransaction
TSQLConnection class 1586
BeginUpdate
IXMLNodeList interface 7705
TAbstractNamedVariants class 6122
TConnectionData class 1518
TWideStrings class 6832
TXMLNodeList class 7465
BeginWrite
IReadWriteSync interface 4038
TMultiReadExclusiveWriteSynchronizer class 4000
TSimpleRWSync class 4005
BevelEdges
TComboBox class 2195
TEdit class 2473
TListBox class 2666
TMemo class 2746
TStaticText class 2953
BevelInner
TComboBox class 2196
TEdit class 2474
TListBox class 2667
TMemo class 2746
TStaticText class 2953
BevelKind
TComboBox class 2196
TEdit class 2474
TListBox class 2667
TMemo class 2747
TStaticText class 2954
BevelOuter
TComboBox class 2197
TEdit class 2474
TListBox class 2668
TMemo class 2747
TStaticText class 2954
BevelWidth
TEdit class 2475
TListBox class 2668
bhoHideAlways enumeration member 6522
bhoHideOnActionNotVisible enumeration member 6522
bhoHideOnDisabledAction enumeration member 6522
bhoHideOnNoExecuteAccess enumeration member 6522
BiDiMode
TButton class 2044
TCheckBox class 2125
TComboBox class 2197
TEdit class 2475
TGroupBox class 2541
TLabel class 2615
TListBox class 2668

TMemo class 2747
TRadioButton class 2824
TScrollBar class 2890
TStaticText class 2954
TValueListEditor class 4806
bifBrowseForComputer enumeration member 1947
bifBrowseForPrinter enumeration member 1947
bifBrowseIncludeFiles enumeration member 1947
bifBrowseIncludeURLS enumeration member 1947
bifDontGoBelowDomain enumeration member 1947
bifEditBox enumeration member 1947
bifNewDialogStyle enumeration member 1947
bifNoNewFolderButton enumeration member 1947
bifNoTranslateTargets enumeration member 1947
bifReturnFSAncestors enumeration member 1947
bifReturnOnlyFSDirs enumeration member 1947
bifShareable enumeration member 1947
bifStatusText enumeration member 1947
bifUAHint enumeration member 1947
bifUseNewUI enumeration member 1947
bifValidate enumeration member 1947
BigEndianUnicode
TEncoding class 3985
BinaryOp
TCustomVariantType class 4994
Bind
TIpSocket class 1419
Binding
IBindings interface 7094
IPort interface 7145
BindingOperation
IBindingOperations interface 7090
BindingOperations
IBinding interface 7082
Bindings
IDefinition interface 7099
BindingType
TWebServExp class 6732
BlobBuffer
TCustomSQLDataSet class 1554
BlockMode
TBaseSocket class 1385
TCustomTcpServer class 1411
TTcpClient class 1445
TTcpServer class 1465
TUdpSocket class 1482
bmBlocking enumeration member 1500
bmNonBlocking enumeration member 1500
bmThreadBlocking enumeration member 1500
Body
TScreenTipsWindow class 358
bool
Variant class 3458
BorderStyle
TCustomEdit class 2367
TCustomListBox class 2423
TCustomScreenTipsManager class 313
TCustomStaticText class 2462
TEdit class 2476
TListBox class 2669
TMemo class 2748
TScreenTipsManager class 339
TStaticText class 2955
TValueListEditor class 4806
Bottom
TImageMargins class 2605
BreakButtons
TAdapterFormLayout class 6417
BrowseOptions
TBrowseForFolder class 1767
bsCommandLink enumeration member 2261
bsPushButton enumeration member 2261
bsrAsIs enumeration member 5094
bsrLower enumeration member 5094
bsrUpper enumeration member 5094
bsSplitButton enumeration member 2261
btDIME enumeration member 7185
btHttp enumeration member 7185

btMime enumeration member 7185
btSoap enumeration member 7185
btUnknown enumeration member 7185
Busy
IWebBrowser interface 745
IWebBrowser2Disp interface 769
IWebBrowserAppDisp interface 790
IWebBrowserDisp interface 799
TInternetExplorer class 449
TWebBrowser class 524
TWebBrowser_V1 class 581
BytesOf
AnsiStringBase class 3186
UnicodeString class 3391
BytesReceived
TBaseSocket class 1386
BytesReserved
IDataBlock interface 272
TDataBlock class 205
BytesSent
TBaseSocket class 1386
ByteType
AnsiStringBase class 3187
UnicodeString class 3391

C

c_bstr
WideString class 3473
c_str
AnsiStringT template 3216
UnicodeString class 3392
CacheConnections
TWebRequestHandler class 6638
CacheData
TXMLTransformProvider class 7751
CalcBounds
TThemedAddRemoveItem class 4557
TThemedMenuButton class 4568
TThemedMenuItem class 4570
TXPStyleAddRemoveItem class 7765
TXPStyleMenuItem class 7776
CalcLayout
TXPStyleMenuItem class 7776
CallCreateObject
TCustomDataBlockInterpreter class 194
TDataBlockInterpreter class 210
CallFreeObject
TCustomDataBlockInterpreter class 194
TDataBlockInterpreter class 210
CallGetIDsOfNames
TCustomDataBlockInterpreter class 194
TDataBlockInterpreter class 210
CallGetServerList
TCustomDataBlockInterpreter class 195
TDataBlockInterpreter class 210
CallInvoke
TCustomDataBlockInterpreter class 195
TDataBlockInterpreter class 211
CanAddActionClass
IAdapterEditor interface 5875
TCustomAdapter class 5539
CanAddClass
IWebComponentEditor interface 6101
TAdapterSubComponent class 5408
TBaseAdapterPageProducer class 1146
TCustomAdapterDisplayColumn class 6463
CanAddClassHelper
IWebComponentEditorHelper interface 6102
TWebComponentsEditorHelper class 6037
CanAddFieldClass
IAdapterEditor interface 5875
TCustomAdapter class 5540
canAdvanceError
IScriptErrorList interface 636
IScriptErrorListDisp interface 641
CanAutoSize
TStandardCustomizePopup class 1746
TStandardMenuPopup class 1757

Cancel
TButton class 2045
TCustomButton class 2267
CanChange
TTabSet class 4474
CanCreateObject
TCustomDataBlockInterpreter class 195
TDataBlockInterpreter class 211
CanDebugScript
IInternetScriptDebugger interface 1048
CanEditModify
TValueListEditor class 4790
CanExecuteLoginForm
TBaseEndUserAdapter class 5504
CanExecuteLogout
TBaseEndUserAdapter class 5504
CanExecuteScript
IInternetExecuteScript interface 1046
CanProduceLoginPage
TBaseEndUserAdapter class 5505
CanResize
TToolDockForm class 4635
canRetreatError
IScriptErrorList interface 637
IScriptErrorListDisp interface 641
CanShowALink
IWinHelpTester interface 6895
CanShowContext
IWinHelpTester interface 6895
CanShowTopic
IWinHelpTester interface 6895
CanUndo
TCustomEdit class 2367
Canvas
TCustomCombo class 2305
TCustomLabel class 2390
TCustomListBox class 2424
TTabSet class 4486
Capacity
TAbstractNamedVariants class 6131
TStringBuilder class 4021
TWideStrings class 6847
Caption
TAdapterActionButton class 6319
TAdapterCommandColumn class 6334
TAdapterDisplayColumn class 6367
TAdapterDisplayField class 6374
TAdapterEditColumn class 6390
TBrowseForFolder class 1767
TButton class 2045
TCheckBox class 2126
TColorSelect class 1776
TCustomAdapterCommandColumn class 6450
TCustomLabel class 2392
TFileExit class 1812
TFileOpen class 1821
TFilePageSetup class 1838
TFilePrintSetup class 1848
TFileSaveAs class 1858
TFontEdit class 1866
TGroupBox class 2542
TLabel class 2616
TPrintDlg class 1886
TRadioButton class 2824
TSearchFind class 1902
TSearchFindNext class 1915
TSearchReplace class 1925
TStaticText class 2955
TTabPage class 4445
CaptionAttributes
TAdapterCommandColumn class 6335
TAdapterDisplayColumn class 6367
TAdapterDisplayField class 6374
TAdapterEditColumn class 6390
TCustomAdapterCommandColumn class 6451
CaptionPosition
TAdapterDisplayField class 6375
TAdapterEditColumn class 6390

CaretPos
TCustomMemo class 2450
CaseSensitive
TWideStringList class 6822
Cast
TCustomVariantType class 4995
CastNativeToSoap
TTypeTranslator class 4682
CastSoapToNative
TTypeTranslator class 4683
CastSoapToVariant
TTypeTranslator class 4683
CastTo
TCustomVariantType class 4995
CastToOle
TCustomVariantType class 4996
cat_printf
AnsiStringT template 3216
UnicodeString class 3392
WideString class 3473
cat_sprintf
AnsiStringT template 3216
UnicodeString class 3393
WideString class 3474
cat_vprintf
AnsiStringBase class 3187
AnsiStringT template 3217
UnicodeString class 3394
WideString class 3474
CatchException
ECustomException class 1310
cbChecked enumeration member 2999
cbGrayed enumeration member 2999
cbUnchecked enumeration member 2999
Cells
TValueListEditor class 4806
Change
TCustomCombo class 2292
TCustomEdit class 2348
TImageMargins class 2604
TScrollBar class 2883
TScroller class 4461
TTabbedNotebook class 4407
Changed
TValueListStrings class 4887
TWebComponentList class 6029
TWideStringList class 6813
ChangeType
Variant class 3431
Changing
TValueListStrings class 4887
TWideStringList class 6814
char
Variant class 3459
CharCase
TComboBox class 2197
TCustomComboBox class 2334
TCustomEdit class 2367
TEdit class 2476
TMemo class 2748
Chars
TStringBuilder class 4022
Check
THTTPReqResp class 1281
CheckAccessRights
IWebUserList interface 1117
TCustomWebUserList class 6783
CheckActive
TXMLDocument class 7356
CheckAutoSave
TXMLDocument class 7356
CheckBrowseMode
TSQLDataLink class 1630
CheckConnection
TSQLConnection class 1586
CheckContentType
THTTPReqResp class 1282
CheckDisconnect

TSQLConnection class 1587
CheckDOM
TXMLDocument class 7356
Checked
TButtonControl class 2113
TCheckBox class 2128
TRadioButton class 2827
CheckExecuteAccess
TBaseAdapterAction class 5452
CheckInactive
TXMLBroker class 7309
CheckModifyAccess
TAdapterNamedField class 5355
TCustomAdapter class 5540
CheckNotHosted
TXMLNode class 7406
CheckOrigValue
ICheckOrigValue interface 5877
CheckPrepareError
TCustomSQLDataSet class 1527
CheckReadOnly
TXMLNode class 7407
CheckRights
TWebUserItem class 6790
CheckStatusChange
ISessionsService interface 1067
TCustomSessionsService class 6748
TSessions class 386
CheckTextNode
IXMLNodeAccess interface 7483
TXMLNode class 7407
CheckValueChange
ICheckValueChange interface 5878
TAdapterUpdateField class 5424
CheckViewAccess
TAdapterNamedField class 5356
ChildListNotify
TXMLNode class 7408
TXMLNodeCollection class 7452
ChildNodeClasses
IXMLNodeAccess interface 7487
TXMLNode class 7438
childNodes
IDOMNode interface 7568
ChildNodes
IXMLDocument interface 7663
IXMLNode interface 7689
TXMLDocument class 7383
TXMLNode class 7439
ChildTableIndent
TAdapterFormLayout class 6417
ChildValues
IXMLNode interface 7690
ClassInfo
TObject class 3358
ClassName
TObject class 3359
ClassNameIs
TObject class 3361
ClassParent
TObject class 3361
ClassType
TObject class 3364
CleanupInstance
TObject class 3367
CleanUpStackInfoProc
Exception class 3961
Clear
IDataBlock interface 267
ISessionsService interface 1067
IXMLNodeCollection interface 7700
IXMLNodeList interface 7706
TAbstractNamedVariants class 6123
TAdapterHiddenFieldsList class 5305
TAllocatedActions class 299
TCustomCombo class 2293
TCustomComboBoxStrings class 2341
TCustomEdit class 2348

TCustomListBox class 2403
TCustomSessionsService class 6749
TCustomVariantType class 4996
TDataBlock class 200
TDOMStrings class 7183
TNamedVariantsList class 6136
TSoapDataList class 1174
TTabList class 4468
TValueListStrings class 4887
TWebComponentList class 6029
TWideStringList class 6814
TWideStrings class 6833
TXMLNodeCollection class 7452
TXMLNodeList class 7465
Variant class 3431
ClearAdapterValues
IClearAdapterValues interface 5879
TCustomAdapter class 5540
ClearDefaultComponents
TAdapterDefaultActions class 5234
TAdapterDefaultFields class 5240
ClearDocumentRef
IXMLNodeAccess interface 7483
TXMLNode class 7408
ClearErrors
IAdapterErrors interface 996
TBaseAdapterErrorsList class 5482
TCustomAdapter class 5541
ClearIndexDefs
TCustomSQLDataSet class 1528
ClearModifications
TSocketForm class 175
ClearScreenTips
TCustomScreenTipsManager class 308
ClearSelection
TCustomCombo class 2293
TCustomEdit class 2349
TCustomListBox class 2403
ClearThreadPool
TServerSocketThread class 1436
ClearUndo
TCustomEdit class 2349
Click
TCustomButton class 2263
TCustomCheckBox class 2278
TTabbedNotebook class 4407
ClicksDisabled
TButtonControl class 2115
ClientConnect
TServerWinSocket class 125
ClientDisconnect
TServerWinSocket class 125
ClientErrorEvent
TServerWinSocket class 125
ClientExecute
TServerClientThread class 98
ClientRead
TServerWinSocket class 125
ClientSocket
TClientSocketThread class 1399
TServerClientThread class 102
ClientToHostWindow
DWebBrowserEvents2 interface 628
ClientToWindow
IWebBrowser2Disp interface 764
IWebBrowserApp interface 778
IWebBrowserAppDisp interface 787
TInternetExplorer class 428
TWebBrowser class 507
ClientType
TClientSocket class 29
TClientWinSocket class 38
ClientWrite
TServerWinSocket class 126
clNone enumeration member 7926
Clone
TXSDate class 7824
TXSDateTime class 7830

TXSTime class 7851
CloneConnection
TSQLConnection class 1587
cloneNode
IDOMNode interface 7560
CloneNode
IXMLNode interface 7673
TXMLNode class 7409
Close
TAbstractSocket class 16
TBaseSocket class 1373
TCustomIpClient class 1402
TCustomTcpServer class 1410
TCustomWinSocket class 69
TThreadLocalCounter class 4024
CloseCursor
TCustomSQLDataSet class 1528
CloseDataSets
TSQLConnection class 1587
CloseStatement
TCustomSQLDataSet class 1528
CloseUp
TCustomCombo class 2293
CMBorderChanged
TToolWindow class 4641
CMCtl3DChanged
TToolWindow class 4641
CMHintShow
TTabSet class 4474
CMMouseenter
TScreenTipsWindow class 353
CMMouseLeave
TScreenTipsWindow class 354
CMTabFontChanged
TTabbedNotebook class 4408
CMTextchanged
TStandardMenuItem class 1753
CoCScriptErrorList class 413
about CoCScriptErrorList class 413
CoCScriptErrorList Members 413
CoCScriptErrorList methods 414
Create 414
CreateRemote 414
code
DOMException class 7508
CodePage
AnsiStringBase class 3188
AnsiStringT template 3217
UnicodeString class 3394
CoInternetExplorer class 414
about CoInternetExplorer class 414
CoInternetExplorer Members 415
CoInternetExplorer methods 415
Create 415
CreateRemote 415
ColCount
TValueListEditor class 4807
Collection
IXMLNode interface 7690
TXMLNode class 7439
coLocalPaths enumeration member 6109
Color
TCheckBox class 2130
TComboBox class 2198
TEdit class 2476
TGroupBox class 2544
TLabel class 2618
TListBox class 2669
TMemo class 2749
TRadioButton class 2827
TScreenTipBorderStyle class 327
TStaticText class 2958
TValueListEditor class 4807
ColorToRGB
TThemeServices class 4582
coltDefault enumeration member 6523
coltImage enumeration member 6523
coltList enumeration member 6523

coltText enumeration member 6523
ColumnCount
TAdapterFormLayout class 6417
Columns
IShellNameSpace interface 707
IShellNameSpaceDisp interface 714
TCustomListBox class 2426
TListBox class 2670
TShellNameSpace class 480
ColWidthsChanged
TValueListEditor class 4790
ComboBox
TCustomComboBoxStrings class 2344
ComboWndProc
TCustomCombo class 2294
CommandLinkHint
TButton class 2048
TCustomButton class 2267
CommandStateChange
DWebBrowserEvents interface 622
DWebBrowserEvents2 interface 628
CommandText
TCustomSQLDataSet class 1554
TInternalSQLDataSet class 891
TSQLDataSet class 1637
CommandType
TCustomSQLDataSet class 1555
TInternalSQLDataSet class 892
TSQLDataSet class 1638
CommaText
TWideStrings class 6848
Commit
TSQLConnection class 1588
CommitFreeAndNil
TSQLConnection class 1588
Compare
TCustomVariantType class 4997
UnicodeString class 3394
CompareDateTimeParam
TXSDateTime class 7831
CompareIC
UnicodeString class 3395
CompareName
TWSDLItems class 7219
CompareOp
TCustomVariantType class 4998
CompareStrings
TAbstractNamedVariants class 6123
TWideStringList class 6814
TWideStrings class 6833
CompareTo
IComparable interface 3494
Component
TScriptProducer class 6714
ComponentClass
TWebComponentClassDescription class 6024
ComponentClassName
TWebComponentClassDescription class 6024
ComponentCount
IWebComponentContainer interface 6100
ComponentList
IWebComponent interface 6096
Components
IWebComponentContainer interface 6100
CompressionRate
TZCompressionStream class 7911
Connect
ISessionsServiceConnectID interface 1069
TClientWinSocket class 38
TCustomIpClient class 1403
TCustomSessionsService class 6749
TCustomWinSocket class 69
THTTPReqResp class 1282
TInternetExplorer class 428
TShellNameSpace class 470
TShellUIHelper class 491
TShellWindows class 498
ConnectActive

ISessionsServiceConnectID interface 1069
TCustomSessionsService class 6750
ConnectActiveSession
TSessions class 387
Connected
ITransport interface 281
TCustomIpClient class 1405
TCustomWinSocket class 83
TSoapConnection class 1196
TSQLConnection class 1609
TTcpClient class 1446
TXMLBroker class 7322
Connection
TSimpleDataSet class 908
ConnectionChange
INotifyConnectionChange interface 7338
ConnectionData
TSQLConnection class 1609
ConnectionList
TSocketForm class 162
ConnectionListColumnClick
TSocketForm class 175
ConnectionListCompare
TSocketForm class 175
ConnectionName
TConnectionData class 1519
TSQLConnection class 1610
ConnectionOptions
TSQLConnection class 1589
ConnectionRegistryFile
TSQLConnection class 1610
Connections
TServerWinSocket class 130
Connections1
TSocketForm class 162
ConnectionState
TSQLConnection class 1611
ConnectSession
TSessions class 387
ConnectTimeout
THTTPReqResp class 1288
ConnectTo
TInternetExplorer class 428
TShellNameSpace class 470
TShellUIHelper class 491
TShellWindows class 498
coNoExecuteScript enumeration member 6109
coNoScript enumeration member 6109
Constraints
TButton class 2048
TCheckBox class 2131
TComboBox class 2199
TEdit class 2477
TGroupBox class 2545
TLabel class 2619
TListBox class 2671
TMemo class 2749
TRadioButton class 2828
TScrollBar class 2891
TSimpleDataSet class 908
TStaticText class 2959
TTabbedNotebook class 4418
TTabSet class 4488
TValueListEditor class 4808
Container
IWebBrowser interface 746
IWebBrowser2Disp interface 769
IWebBrowserAppDisp interface 790
IWebBrowserDisp interface 799
IWebComponent interface 6096
TInternetExplorer class 450
TWebBrowser class 525
TWebBrowser_V1 class 582
ContainerList
TWebContainedComponentHelper class 6051
ContainerName
IWebVariablesContainerName interface 6105
Content

IWebContent interface 6104
TCustomAdapterErrorList class 6480
TCustomXSLPageProducer class 7885
TScriptProducer class 6715
ContentFromStream
TBaseAdapterPageProducer class 1147
TCustomXSLPageProducer class 7885
ContentFromString
TCustomXSLPageProducer class 7886
ContentFromWideString
TCustomXSLPageProducer class 7886
ContentRect
TThemeServices class 4583
Control
TDisplayOffset class 325
TEditAction class 1795
TShadowWindow class 405
ControlAttributes
TLayoutAttributes class 6013
ControlContent
TCustomAdapterDisplayColumn class 6463
TCustomAdapterDisplayField class 6470
ControlInterface
TWebBrowser class 525
TWebBrowser_V1 class 582
Controller
TAggregatedObject class 3317
TService class 3074
Convert
TEncoding class 3975
Converter
THTTPRIO class 1252
TSoapPascalInvoker class 1355
ConvertValue
TAdapterUpdateField class 5425
TCustomAdapterBooleanField class 5594
ConvertValueCount
TAdapterUpdateField class 5425
ConvertValues
TAdapterUpdateField class 5425
Copy
DynamicArray template 3265
TCustomVariantType class 4999
WideString class 3475
CopyRange
DynamicArray template 3266
CopySelection
TCustomCombo class 2294
TCustomListBox class 2404
CopyTo
TStringBuilder class 4016
CopyToClipboard
TCustomEdit class 2349
CornerSize
TScreenTipBorderStyle class 327
TScreenTipsWindow class 358
CoSearchAssistantOC class 415
about CoSearchAssistantOC class 415
CoSearchAssistantOC Members 416
CoSearchAssistantOC methods 416
Create 416
CreateRemote 416
CoShellBrowserWindow class 417
about CoShellBrowserWindow class 417
CoShellBrowserWindow Members 417
CoShellBrowserWindow methods 417
Create 417
CreateRemote 418
CoShellNameSpace class 418
about CoShellNameSpace class 418
CoShellNameSpace Members 418
CoShellNameSpace methods 418
Create 419
CreateRemote 419
CoShellUIHelper class 419
about CoShellUIHelper class 419
CoShellUIHelper Members 419
CoShellUIHelper methods 420

Create 420
CreateRemote 420
CoShellWindows class 420
about CoShellWindows class 420
CoShellWindows Members 421
CoShellWindows methods 421
Create 421
CreateRemote 421
Count
ISearches interface 688
ISearchesDisp interface 690
IShellWindows interface 729
IShellWindowsDisp interface 732
IWebFormComponentContainer interface 1086
IXMLNodeCollection interface 7702
IXMLNodeList interface 7714
TAbstractNamedVariants class 6131
TAbstractScriptObjectFactories class 6655
TAllocatedActions class 300
TCustomListBox class 2427
TDOMStrings class 7183
TDOMVendorList class 7517
TLanguages class 3989
TLineNumberMap class 6670
TNotifyList class 972
TScriptEnginesList class 6676
TScriptObjectFactories class 6706
TShellWindows class 503
TUpdateFileList class 5865
TWebComponentList class 6033
TWideStrings class 6849
TXMLNodeCollection class 7457
TXMLNodeList class 7474
CountViewTypes
IShellNameSpace interface 707
IShellNameSpaceDisp interface 714
TShellNameSpace class 480
Create
CoCScriptErrorList class 414
CoInternetExplorer class 415
CoSearchAssistantOC class 416
CoShellBrowserWindow class 417
CoShellNameSpace class 419
CoShellUIHelper class 420
CoShellWindows class 421
EAdapterFieldException class 5179
ECustomException class 1307
EDOMParseError class 7510
ESOAPHTTPException class 1274
Exception class 3952
TAbstractAdapterRequest class 5957
TAbstractAdapterResponse class 5959
TAbstractScriptEngine class 6649
TAbstractWebContext class 5962
TAdapterActionName class 6537
TAdapterActionsListHelper class 6326
TAdapterControlGroup class 6343
TAdapterDataDisplay class 6356
TAdapterDispatchParams class 6156
TAdapterDisplayReferenceGroup class 6382
TAdapterError class 5258
TAdapterErrorsList class 5262
TAdapterFormLayout class 6410
TAdapterHiddenField class 5301
TAdapterHiddenFieldsList class 5304
TAdapterItemRequestParams class 6162
TAdapterNamedDisplayField class 5346
TAdapterObjectErrorsList class 5389
TAdapterUpdateField class 5423
TAggregatedObject class 3315
TAllocatedActions class 298
TAssociateItem class 301
TAssociateItemList class 303
TBaseAdapterMultiValueField class 5492
TBaseAdapterPageProducer class 1145
TBaseSocket class 1371
TBaseWebModuleFactory class 6281
TBaseWebPageInfo class 6286

TBaseWebPageModuleFactory class 6291
TButtonControl class 2110
TClientSocket class 26
TClientSocketThread class 1397
TCommonDialogAction class 1785
TComponentFactory class 5167
TConditionVariableCS class 3114
TConditionVariableHelper class helper 3118
TConditionVariableMutex class 3122
TConnectionData class 1516
TControlRequirements class 6426
TCriticalSection class 3127
TCurrentWebPageInfo class 6659
TCustomAdapter class 5536
TCustomAdapterActionButton class 6431
TCustomAdapterCommandColumn class 6444
TCustomAdapterCommandGroup class 6456
TCustomAdapterDisplayColumn class 6463
TCustomAdapterDisplayField class 6469
TCustomAdapterErrorList class 6480
TCustomAdapterFieldGroup class 6485
TCustomAdapterForm class 6491
TCustomAdapterGrid class 6499
TCustomAdapterPageProducer class 1161
TCustomButton class 2260
TCustomCheckBox class 2277
TCustomCombo class 2285
TCustomComboBox class 2325
TCustomComboBoxStrings class 2340
TCustomEdit class 2347
TCustomGroupBox class 2382
TCustomIpClient class 1402
TCustomLabel class 2385
TCustomListBox class 2400
TCustomMemo class 2446
TCustomScreenTipsManager class 306
TCustomScreenTipsPopup class 317
TCustomSessionsService class 6747
TCustomSQLDataSet class 1524
TCustomStaticText class 2458
TCustomStringsValuesList class 5733
TCustomTcpServer class 1408
TCustomVariantType class 4991
TCustomWebAppDataModule class 6541
TCustomWebAppPageModule class 6546
TCustomWebDataModule class 6551
TCustomWebPageModule class 6559
TCustomWebUserList class 6782
TCustomWinSocket class 66
TCustomZStream class 7906
TDataBlock class 199
TDataBlockInterpreter class 208
TDataDispatch class 216
TDependencies class 3064
TDesigntimeWarnings class 6504
TDisplayOffset class 324
TDOMStrings class 7182
TEvent class 3135
TEventLogger class 3069
TFieldValuesAdapter class 5809
TFileOpen class 1819
TFileSaveAs class 1856
TGeneratedFunctions class 6507
THandleObject class 3139
THelpAction class 1875
THintAction class 1883
THTMLItem class 6664
THTMLItems class 6666
THTTPReqResp class 1278
THTTPRIO class 1249
THTTPSoapDispatcher class 5947
TInternetExplorer class 425
TInvalidWebSession class 6217
TIpSocket class 1418
TItemProp class 4779
TLanguages class 3988
TLayoutAttributes class 6013
TLayoutWebContent class 6016

TLinkedRIO class 1317
TLinkedWebNode class 1322
TLocalConnection class 4547
TLogicalPageList class 5824
TLogLinkedWebNode class 1327
TMBCSEncoding class 3993, 3994
TMultiReadExclusiveWriteSynchronizer class 3999
TMutex class 3145
TNotifyList class 969
TObject class 3355
TRadioButton class 2815
TRawSocket class 1432
TScreenTipBorderStyle class 326
TScreenTipItem class 329
TScreenTipsCollection class 335
TScreenTipsWindow class 352
TScriptComponent class 975
TScriptEnginesList class 6674
TScriptError class 6678
TScriptErrors class 6681
TScriptFile class 6684
TScriptGlobalObjects class 6688
TScriptIncludeFile class 6693
TScriptIncludeFiles class 6694
TScriptIncludeItem class 6697
TScriptItem class 6699
TScriptItems class 6701
TScriptObjectFactories class 6704
TScriptProducer class 6708
TScrollBar class 2883
TScroller class 4459
TSearchAction class 1895
TSearchFindFirst class 1911
TSearchFindNext class 1913
TSemaphore class 3149
TServerAcceptThread class 91
TServerClientThread class 97
TServerClientWinSocket class 106
TServerSocket class 110
TServerSocketThread class 1434
TServerWinSocket class 123
TServiceApplication class 3091
TServiceThread class 3096
TSessionIDGen class 377
TSessionItem class 379
TSessions class 385
TShadowWindow class 401
TShellNameSpace class 467
TShellUIHelper class 484
TShellWindows class 495
TSimpleDataSet class 899
TSimpleRWSync class 4004
TSiteModuleHelper class 6567
TSitePageModuleHelper class 6576
TSoapConnection class 1190
TSoapDataModule class 1208
TSoapPascalInvoker class 1351
TSocketConnection class 223
TSocketTransport class 231
TSQLBlobStream class 1578
TSQLConnection class 1583
TSQLDataLink class 1629
TSQLDataSet class 1635
TSQLMonitor class 1645
TSQLQuery class 1657
TSqlServerMethod class 1668
TSQLStoredProc class 1674
TSQLTable class 1684
TStreamedConnection class 237
TStringBuilder class 4011
TTabbedNotebook class 4405
TTabList class 4464
TTabPage class 4444
TTabSet class 4473
TThemedCustomizePopup class 4562
TThemedPopupMenu class 4573
TThemeServices class 4581
TToolDockForm class 4635

TToolDockObject class 4638
TToolWindow class 4640
TTransportThread class 249
TTypeTranslator class 4681
TUdpSocket class 1479
TUnicodeEncoding class 4026
TUpdateFileList class 5865
TUTF7Encoding class 4030
TUTF8Encoding class 4035
TValueListEditor class 4788
TValueListStrings class 4883
TVariableLookup class 6717
TWebAppDataModuleFactory class 6296
TWebApplication class 5937
TWebAppPageModuleFactory class 6298
TWebComponentList class 6027
TWebConnection class 256
TWebContainedComponent class 6039
TWebContainedComponentHelper class 6047
TWebContainedContainerComponent class 6054
TWebContainerComponent class 6064
TWebContainerComponentHelper class 6069
TWebContentOptions class 6077
TWebContext class 5978
TWebDataModuleFactory class 6300
TWebModuleFactoryList class 6623
TWebModuleList class 6627
TWebNotifyContainer class 6601
TWebPageInfo class 6302
TWebPageModuleFactory class 6304
TWebRequestHandler class 6635
TWebServExp class 6729
TWebSession class 6252
TWebSessionIDsImpl class 6761
TWebSessionImpl class 6765
TWebUserItem class 6788
TWebUserItems class 6793
TWideStringList class 6811
TWideStringsEnumerator class 6857
TWindowAction class 1933
TWinSocketStream class 140
TWSDLHTMLPublish class 7281
TWSDLItems class 7217
TXMLBroker class 7305
TXMLData class 7798
TXMLDocument class 7351
TXMLNode class 7400
TXMLNodeList class 7462
TXMLTransformClient class 7741
TXMLTransformProvider class 7749
TXPStyleCustomizePopup class 7769
TXPStylePopupMenu class 7779
TXSCustomDateTime class 7805
TXSDate class 7823
TXSDuration class 7838
TZCompressionStream class 7909
TZDecompressionStream class 7913
CreateActionRequestContext
ICreateActionRequestContext interface 5880
TCustomAdapter class 5541
createAttribute
IDOMDocument interface 7532
CreateAttributeList
TXMLNode class 7409
CreateAttributeNode
IXMLNodeAccess interface 7483
TXMLNode class 7410
createAttributeNS
IDOMDocument interface 7533
CreateBlobStream
TCustomSQLDataSet class 1528
createCDATASection
IDOMDocument interface 7533
CreateChildList
TXMLNode class 7410
CreateChildNode
IXMLNodeAccess interface 7483
TXMLNode class 7410

CreateCollection
IXMLNodeAccess interface 7483
TXMLNode class 7411
createComment
IDOMDocument interface 7533
CreateComObject
TComponentFactory class 5167
CreateComponentObject
TAbstractScriptObjectFactory class 6656
CreateComponentStub
TAbstractScriptEngine class 6651
CreateContext
TCustomWebAppComponents class 6204
TMultiModuleWebAppServices class 6225
CreateControl
TWebBrowser class 508
TWebBrowser_V1 class 572
CreateDefaultFields
TAdapterControlGroup class 6346
createDocument
IDOMImplementation interface 7553
createDocumentFragment
IDOMDocument interface 7533
createDocumentType
IDOMImplementation interface 7553
CreateEditor
TValueListEditor class 4790
createElement
IDOMDocument interface 7533
CreateElement
IXMLDocument interface 7644
TXMLDocument class 7357
createElementNS
IDOMDocument interface 7534
createEntityReference
IDOMDocument interface 7534
CreateErrorsList
TCustomAdapter class 5542
CreateFile
TLinkedRIO class 1318
CreateFmt
ESOAPHTTPException class 1274
Exception class 3954
CreateFmtHelp
Exception class 3955
CreateForm
TServiceApplication class 3092
TWebApplication class 5938
CreateGlobalObject
TAbstractScriptObjectFactory class 6657
TScriptObjectFactories class 6705
CreateHandle
TTabbedNotebook class 4408
CreateHeader
IDOMHeaderProcessor interface 1242
CreateHelp
Exception class 3955
CreateHosted
TXMLNode class 7401
CreateHResult
ESafeArrayError class 5113
CreateID
EAdapterException class 5178
CreateInstance
TComponentFactory class 5168
CreateItemList
TXMLNodeCollection class 7453
CreateMimeStream
IMimeAttachmentHandler interface 1178
CreateNew
TCustomWebAppDataModule class 6541
TCustomWebAppPageModule class 6547
TCustomWebDataModule class 6552
TCustomWebPageModule class 6559
TService class 3073
TSocketService class 183
CreateNode
IXMLDocument interface 7645

TXMLDocument class 7357
CreateObject
TCustomDataBlockInterpreter class 195
TDataBlockInterpreter class 211
CreatePageElements
TBaseAdapterPageProducer class 1147
CreateParams
TButtonControl class 2111
TCustomButton class 2263
TCustomCheckBox class 2278
TCustomComboBox class 2326
TCustomEdit class 2351
TCustomGroupBox class 2383
TCustomListBox class 2404
TCustomMemo class 2447
TCustomStaticText class 2459
TRadioButton class 2816
TScreenTipsWindow class 354
TScrollBar class 2884
TShadowWindow class 402
TTabbedNotebook class 4409
TTabSet class 4475
TToolDockForm class 4636
createProcessingInstruction
IDOMDocument interface 7534
CreateProducerObject
TAbstractScriptObjectFactory class 6657
CreateRemote
CoCScriptErrorList class 414
CoInternetExplorer class 415
CoSearchAssistantOC class 416
CoShellBrowserWindow class 418
CoShellNameSpace class 419
CoShellUIHelper class 420
CoShellWindows class 421
CreateRequestContext
IAdapterRequestHandler interface 1013
TBaseAdapterAction class 5452
TCustomAdapterImageField class 5628
CreateRes
Exception class 3956
CreateResFmt
Exception class 3957
CreateResFmtHelp
Exception class 3958
CreateResHelp
Exception class 3959
CreateRoundedStyle
TScreenTipsWindow class 354
CreateScreenTip
TCustomScreenTipsManager class 308
CreateSessionIDGen
TSessions class 387
CreateShadow
TShadowWindow class 401
CreateSubscriptionForSelection
IShellFavoritesNameSpace interface 692
IShellFavoritesNameSpaceDisp interface 697
IShellNameSpaceDisp interface 710
TShellNameSpace class 471
createTextNode
IDOMDocument interface 7534
CreateThread
TServerSocketThread class 1436
CreateTipBorderStyle
TScreenTipsWindow class 355
CreateTransport
TSocketConnection class 224
TStreamedConnection class 239
TWebConnection class 257
CreateWindowHandle
TCustomEdit class 2351
TCustomMemo class 2447
CreateWnd
TCustomButton class 2264
TCustomCheckBox class 2279
TCustomCombo class 2295
TCustomComboBox class 2327

TCustomEdit class 2352
TCustomListBox class 2405
TRadioButton class 2816
TScreenTipsWindow class 355
TScrollBar class 2884
TValueListEditor class 4791
crEqual enumeration member 5095
crGreaterThan enumeration member 5095
crLessThan enumeration member 5095
csAllocated enumeration member 3714
csBottom enumeration member 406
csContinuePending enumeration member 3102
csDropDown enumeration member 3000
csDropDownList enumeration member 3000
csOwnerDrawFixed enumeration member 3000
csOwnerDrawVariable enumeration member 3000
csPaused enumeration member 3102
csPausePending enumeration member 3102
csReserved enumeration member 3714
csRight enumeration member 406
csRunning enumeration member 3102
csSimple enumeration member 3000
csStartPending enumeration member 3102
csStateClosed enumeration member 1700
csStateConnecting enumeration member 1700
csStateDisconnecting enumeration member 1700
csStateExecuting enumeration member 1700
csStateFetching enumeration member 1700
csStateOpen enumeration member 1700
csStopped enumeration member 3102
csStopPending enumeration member 3102
csSysAllocated enumeration member 3714
csSysReserved enumeration member 3714
CStringEvent
TFTCompT class 1312
csUnallocated enumeration member 3714
ctAnchor enumeration member 6523
ctbLeadSurrogate enumeration member 3389
ctBlocking enumeration member 146
ctButton enumeration member 6523
ctDefault enumeration member 6523
ctEventImages enumeration member 6523
ctImage enumeration member 6523
Ctl3D
TCheckBox class 2132
TComboBox class 2199
TEdit class 2478
TGroupBox class 2546
TListBox class 2671
TMemo class 2750
TRadioButton class 2829
TScrollBar class 2892
TValueListEditor class 4809
ctNonBlocking enumeration member 146
ctNotLeadChar enumeration member 3389
ctTrailSurrogate enumeration member 3389
Currency
Currency class 3237
Variant class 3459
Currency __fastcall operator -(double lhs, const Currency& rhs)
Currency class 3253
Currency __fastcall operator -(int lhs, const Currency& rhs)
Currency class 3253
Currency __fastcall operator *(double lhs, const Currency& rhs)
Currency class 3254
Currency __fastcall operator *(int lhs, const Currency& rhs)
Currency class 3254
Currency __fastcall operator /(double lhs, const Currency& rhs)
Currency class 3254
Currency __fastcall operator /(int lhs, const Currency& rhs)
Currency class 3254
Currency __fastcall operator +(double lhs, const Currency& rhs)
Currency class 3255
Currency __fastcall operator +(int lhs, const Currency& rhs)
Currency class 3255
Currency class 3235
- 3239
-- 3240

- operator 3238
-- operator 3239
! 3240
!= 3241
!= operator 3240
% 3241
%= 3242
* 3242
* operator 3242
*= 3243
/ 3243
/ operator 3243
/= 3244
+ 3245
+ operator 3244
++ 3246
++ operator 3245
+= 3246
< 3247
< operator 3247
<= 3248
<= operator 3247
= 3248
-= 3249
= operator 3248
== 3250
== operator 3249
> 3250
> operator 3250
>= 3251
>= operator 3251
about Currency class 3235
Currency 3237
Currency __fastcall operator -(double lhs, const Currency& rhs) 3253
Currency __fastcall operator -(int lhs, const Currency& rhs) 3253
Currency __fastcall operator *(double lhs, const Currency& rhs) 3254
Currency __fastcall operator *(int lhs, const Currency& rhs) 3254
Currency __fastcall operator /(double lhs, const Currency& rhs) 3254
Currency __fastcall operator /(int lhs, const Currency& rhs) 3254
Currency __fastcall operator +(double lhs, const Currency& rhs) 3255
Currency __fastcall operator +(int lhs, const Currency& rhs) 3255
Currency constructor 3237
Currency friends 3253
Currency Members 3235
Currency operators 3238
double 3252
int 3252
System::String 3252
Currency*
Variant class 3459
CurrencyBase class 3255
about CurrencyBase class 3255
CurrencyBase data members 3256
CurrencyBase Members 3255
Val 3256
Current
IEnumerator interface 3500
TWideStringsEnumerator class 6858
CurrentBlobSize
TCustomSQLDataSet class 1556
CurrentDate
TDateTime class 3328
CurrentDateTime
TDateTime class 3328
CurrentPage
TDefaultFieldsPagedAdapter class 5760
CurrentTime
TDateTime class 3328
CurrToStr
AnsiStringBase class 3188
AnsiStringT template 3218
UnicodeString class 3395
WideString class 3475
CurrToStrF

AnsiStringBase class 3188
AnsiStringT template 3218
UnicodeString class 3396
WideString class 3475
Custom
TAdapterActionButton class 6319
TAdapterCommandColumn class 6335
TAdapterCommandGroup class 6340
TAdapterDisplayColumn class 6367
TAdapterDisplayField class 6375
TAdapterEditColumn class 6391
TAdapterFieldGroup class 6403
TAdapterForm class 6406
TCustomAdapterCommandColumn class 6451
TCustomAdapterCommandGroup class 6459
TCustomAdapterFieldGroup class 6489
TCustomAdapterForm class 6495
CustomSort
TValueListStrings class 4888
TWideStringList class 6815
CutToClipboard
TCustomEdit class 2352

D

data
AnsiStringT template 3218
IDOMCharacterData interface 7529
IDOMProcessingInstruction interface 7601
UnicodeString class 3396
WideString class 3476
Data
AnsiStringBase class 3182
SmallStringBase template 3311
TAdapter class 5196
TApplicationAdapter class 5444
TCustomAdapter class 5572
TCustomWinSocket class 84
TDefaultFieldsAdapter class 5747
TEndUserAdapter class 5770
TEndUserSessionAdapter class 5800
TLoginFormAdapter class 5831
TPagedAdapter class 5850
TServerClientThread class 103
TStringsValuesList class 5863
TXMLTransform class 7731
DataIn
IDataIntercept interface 275
DataLink
TCustomSQLDataSet class 1556
DataOut
IDataIntercept interface 276
DataRequest
TSoapDataModule class 1217
DataSet
TSimpleDataSet class 909
DataSets
TSQLConnection class 1611
DataSource
TCustomSQLDataSet class 1557
TInternalSQLDataSet class 893
TSQLDataSet class 1639
TSQLQuery class 1660
Date enumeration member 3327
DateString
TDateTime class 3329
DateTime enumeration member 3327
DateTimeString
TDateTime class 3329
Day
TXSCustomDateTime class 7816
TXSDate class 7827
DayOfWeek
TDateTime class 3329
DbxCommandType
TCustomSQLDataSet class 1557
TSQLDataSet class 1639
DBXConnection
TSQLConnection class 1611

dcChangeCurrentRecordView enumeration member 1122
dcCurrentRecordView enumeration member 1122
dcMultipleRecordView enumeration member 1122
DeactivateWebModules
TWebRequestHandler class 6636
DebugScript
IInternetScriptDebugger interface 1049
DecimalSecond
TXSDuration class 7839
DecimalString
TXSDecimal class 7835
DeclareNamespace
IXMLNode interface 7674
TXMLNode class 7412
DeclareStyles
TBaseAdapterPageProducer class 1148
DeclareWarnings
TBaseAdapterPageProducer class 1148
DecodeDate
TDateTime class 3330
DecodeTime
TDateTime class 3330
DecRefCount
DynamicArray template 3266
Default
ISearches interface 688
ISearchesDisp interface 690
TButton class 2049
TCustomButton class 2268
TEncoding class 3985
DefaultAction
IGetDefaultAction interface 1028
TCustomWebPageModule class 6561
TSitePageModuleHelper class 6583
TWebAppPageModule class 6591
TWebPageModule class 6606
DefaultColWidth
TValueListEditor class 4809
DefaultDrawing
TValueListEditor class 4811
DefaultHandler
TCustomEdit class 2354
TCustomWinSocket class 70
TObject class 3367
DefaultInterface
TInternetExplorer class 450
TShellNameSpace class 480
TShellUIHelper class 494
TShellWindows class 503
TWebBrowser class 526
TWebBrowser_V1 class 582
DefaultNamespaceURI
TXMLNodeList class 7475
DefaultPage
TCustomPageDispatcher class 6195
TPageDispatcher class 6238
DefaultPageName
IPageDispatcher interface 1064
DefaultProcessHeader
TSOAPDOMProcessor class 1238
DefaultRowHeight
TValueListEditor class 4811
DefaultTimeout
TCustomSessionsService class 6754
TSessionsService class 6758
DefaultWebComponents
TAdapterDefaultActions class 5237
TAdapterDefaultFields class 5243
TWebContainerComponentHelper class 6074
DefaultWebComponentsChange
TDefaultFieldsAdapter class 5744
DefineFunction
TAbstractControlRequirements class 6312
TAbstractFormRequirements class 6315
TControlRequirements class 6427
DefineLabel
IAdapterErrors interface 996
TBaseAdapterErrorsList class 5483

DefineProperties
TAbstractNamedVariants class 6124
TCustomSQLDataSet class 1529
TWideStrings class 6834
TXMLDocument class 7358
DefineVariable
TAbstractControlRequirements class 6312
TAbstractFormRequirements class 6315
TControlRequirements class 6428
Definition
IWSDLDocument interface 7167
TWSDLDocument class 7070
DelayInitialize
TServiceApplication class 3094
DelegateConnection
TConnectionData class 1519
Delete
AnsiStringBase class 3189
AnsiStringT template 3219
IXMLNodeCollection interface 7700
IXMLNodeList interface 7707
TAbstractNamedVariants class 6124
TAllocatedActions class 299
TCustomComboBoxStrings class 2341
TNamedVariantsList class 6136
TTabList class 4469
TThreadLocalCounter class 4024
TValueListStrings class 4888
TWideStringList class 6815
TWideStrings class 6834
TXMLNodeCollection class 7453
TXMLNodeList class 7466
UnicodeString class 3396
WideString class 3476
deleteData
IDOMCharacterData interface 7528
DeleteRecords
TSQLTable class 1685
DeleteRow
TValueListEditor class 4791
DeleteSelected
TCustomCombo class 2295
TCustomListBox class 2406
DeleteString
TCustomListBox class 2406
DeleteSubscriptionForSelection
IShellFavoritesNameSpace interface 692
IShellFavoritesNameSpaceDisp interface 697
IShellNameSpaceDisp interface 711
TShellNameSpace class 471
DelimitedText
TWideStrings class 6849
Delimiter
TWideStrings class 6850
DelphiInterface
DelphiInterface template 3258
DelphiInterface template 3256
! 3259
& 3259
* 3259
~DelphiInterface 3257
= 3260
= operator 3260
-> 3260
about DelphiInterface template 3256
DelphiInterface 3258
DelphiInterface constructor 3257
DelphiInterface Members 3257
DelphiInterface methods 3258
DelphiInterface operators 3259
Release 3258
T* 3261
Dependencies
TService class 3078
Depth
IShellNameSpace interface 707
IShellNameSpaceDisp interface 715
TShellNameSpace class 480

Description
TDOMVendor class 7515
TScreenTipItem class 332
DesignerData
TCustomSQLDataSet class 1557
Designing
TScriptProducer class 6715
DestLineToSourceLine
TLineNumberMap class 6670
Destroy
ECustomException class 1309
Exception class 3960
TAbstractNamedVariants class 6120
TAbstractScriptEngine class 6649
TAbstractWebContext class 5963
TAdapterControlGroup class 6344
TAdapterDefaultActions class 5233
TAdapterDefaultFields class 5240
TAdapterDispatchParams class 6156
TAdapterDisplayReferenceGroup class 6382
TAdapterErrorsList class 5262
TAdapterHiddenFieldsList class 5304
TAdapterItemRequestParams class 6162
TAdapterNextPageField class 5383
TAllocatedActions class 298
TAssociateItem class 301
TAssociateItemList class 303
TBaseAdapterMultiValueField class 5492
TBaseAdapterPageProducer class 1145
TBaseSocket class 1372
TClientSocket class 26
TClientSocketThread class 1397
TCommonDialogAction class 1786
TConditionVariableMutex class 3123
TConnectionData class 1517
TControlRequirements class 6427
TCriticalSection class 3127
TCriticalSectionHelper class helper 3131
TCustomAdapter class 5536
TCustomAdapterActionButton class 6432
TCustomAdapterCommandColumn class 6445
TCustomAdapterCommandGroup class 6456
TCustomAdapterErrorList class 6480
TCustomAdapterFieldGroup class 6485
TCustomAdapterForm class 6491
TCustomAdapterGrid class 6499
TCustomAdapterPageProducer class 1162
TCustomButton class 2261
TCustomCombo class 2286
TCustomComboBox class 2326
TCustomComboBoxStrings class 2340
TCustomListBox class 2401
TCustomMemo class 2446
TCustomScreenTipsManager class 307
TCustomScreenTipsPopup class 318
TCustomServerSocket class 41
TCustomSQLDataSet class 1525
TCustomStringsValuesList class 5733
TCustomTcpServer class 1408
TCustomVariantType class 4991
TCustomWebAppDataModule class 6542
TCustomWebAppPageModule class 6547
TCustomWebDataModule class 6552
TCustomWebPageModule class 6560
TCustomWebUserList class 6782
TCustomWinSocket class 67
TDataBlock class 199
TDataBlockInterpreter class 208
TDataDispatch class 217
TDesigntimeWarnings class 6504
TDOMStrings class 7182
TEditAction class 1792
TEventLogger class 3070
TGeneratedFunctions class 6507
THandleObject class 3139
THTMLItems class 6666
THTTPReqResp class 1278
THTTPRIO class 1250

THTTPSoapDispatcher class 5948
TInternetExplorer class 425
TItemProp class 4779
TLinkedRIO class 1318
TLinkedWebNode class 1322
TLocalConnection class 4548
TLogLinkedWebNode class 1328
TMultiReadExclusiveWriteSynchronizer class 3999
TNamedVariantsList class 6135
TNotifyList class 970
TObject class 3356
TScreenTipItem class 329
TScreenTipsWindow class 352
TScriptEnginesList class 6674
TScriptErrors class 6682
TScriptFile class 6684
TScriptGlobalObjects class 6689
TScriptObjectFactories class 6704
TScriptProducer class 6708
TScroller class 4460
TSearchAction class 1896
TServerClientThread class 98
TServerClientWinSocket class 107
TServerSocketThread class 1435
TServerWinSocket class 123
TService class 3073
TServiceApplication class 3091
TSessionItem class 379
TSessions class 385
TShadowWindow class 401
TShellNameSpace class 468
TShellUIHelper class 485
TShellWindows class 496
TSimpleDataSet class 900
TSimpleRWSync class 4004
TSiteModuleHelper class 6567
TSitePageModuleHelper class 6577
TSoapConnection class 1191
TSoapDataModule class 1208
TSOAPDOMProcessor class 1235
TSoapPascalInvoker class 1352
TSocketTransport class 231
TSQLBlobStream class 1579
TSQLConnection class 1584
TSQLMonitor class 1645
TSQLQuery class 1658
TSQLTable class 1685
TStreamedConnection class 238
TTabbedNotebook class 4406
TTabSet class 4473
TThemeServices class 4581
TThreadLocalCounter class 4023
TTransportThread class 249
TTypeTranslator class 4681
TValueListEditor class 4788
TValueListStrings class 4884
TWebApplication class 5938
TWebComponentList class 6027
TWebConnection class 257
TWebContainedComponent class 6040
TWebContainedComponentHelper class 6047
TWebContainedContainerComponent class 6054
TWebContainerComponent class 6064
TWebContainerComponentHelper class 6070
TWebContext class 5979
TWebModuleFactoryList class 6624
TWebModuleList class 6628
TWebNotifyContainer class 6601
TWebRequestHandler class 6635
TWebServExp class 6729
TWebSessionIDsImpl class 6761
TWebSessionImpl class 6765
TWebUserItem class 6789
TWideStringList class 6811
TWideStrings class 6828
TWinSocketStream class 140
TWSDLHTMLPublish class 7282
TXMLBroker class 7306

TXMLData class 7798
TXMLDocument class 7352
TXMLNode class 7401
TXMLNodeList class 7463
TXMLTransformClient class 7741
TXMLTransformProvider class 7749
TXPStylePopupMenu class 7779
TXSCustomDateTime class 7805
TZCompressionStream class 7909
TZDecompressionStream class 7913
DestroyWindowHandle
TCustomCombo class 2296
DestroyWnd
TCustomComboBox class 2327
TCustomEdit class 2354
TCustomListBox class 2406
dftDefault enumeration member 6525
dftImage enumeration member 6525
dftList enumeration member 6525
dftText enumeration member 6525
Dialog
TColorSelect class 1777
TFileOpen class 1821
TFilePageSetup class 1838
TFilePrintSetup class 1848
TFileSaveAs class 1858
TFontEdit class 1867
TPrintDlg class 1887
TSearchFind class 1902
TSearchReplace class 1925
DialogCaption
TBrowseForFolder class 1768
DimCount
DynamicArray template 3267
DisabledDescription
TScreenTipItem class 332
DisableDebugger
IInternetEnvOptions interface 1044
DisabledHeader
TScreenTipItem class 332
DisabledImageIndex
TButton class 2050
TCustomButton class 2269
DisableStringTrim
TSimpleDataSet class 909
Disconnect
TCustomIpClient class 1403
TCustomWinSocket class 70
TInternetExplorer class 429
TServerWinSocket class 126
TShellNameSpace class 471
TShellUIHelper class 492
TShellWindows class 499
DisconnectAction
TSocketForm class 163
DisconnectActionUpdate
TSocketForm class 175
DisconnectSession
TSessions class 387
Dispatch
TObject class 3369
DispatchActions
TCustomWebAppComponents class 6212
TWebAppComponents class 6246
DispatchedPageName
TAbstractWebContext class 5972
DispatchEnabled
THTTPSoapDispatcher class 5948
TWSDLHTMLPublish class 7283
TXMLBroker class 7309
Dispatcher
TCustomXSLPageProducer class 7891
THTTPSoapDispatchNode class 1261
DispatcherComponent
TCustomXSLPageProducer class 7892
DispatchIndex
TDataDispatch class 219
DispatchMask

THTTPSoapDispatcher class 5949
TWSDLHTMLPublish class 7283
TXMLBroker class 7309
DispatchMethodType
THTTPSoapDispatcher class 5949
TWSDLHTMLPublish class 7284
TXMLBroker class 7310
DispatchPage
IPageResult interface 1065
TSitePageModuleHelper class 6577
DispatchPageName
IPageDispatcher interface 1061
TCustomPageDispatcher class 6190
DispatchRequest
THTTPSoapDispatcher class 5949
TWSDLHTMLPublish class 7284
TXMLBroker class 7310
DispatchServiceMain
TServiceApplication class 3092
DispatchSOAP
IHTTPSoapDispatch interface 1263
THTTPSoapDispatchNode class 1259
THTTPSoapPascalInvoker class 1269
DispInvoke
TCustomVariantType class 4999
TInvokeableVariantType class 5018
DisplayColumns
TAdapterActionButton class 6320
TAdapterCommandColumn class 6335
TAdapterDisplayField class 6376
TAdapterEditColumn class 6391
TCustomAdapterActionButton class 6438
TCustomAdapterCommandColumn class 6451
TCustomAdapterDisplayField class 6472
DisplayComponent
TAdapterCommandGroup class 6340
TAdapterDisplayReferenceGroup class 6386
DisplayComponentChange
TAdapterDisplayReferenceGroup class 6384
DisplayLabel
TAdapterAction class 5205
TAdapterBooleanField class 5228
TAdapterField class 5272
TAdapterFileField class 5287
TAdapterImageField class 5314
TAdapterMemoField class 5328
TAdapterMultiValueField class 5338
TAdapterNamedDisplayField class 5350
TAdapterNextPageAction class 5376
TAdapterNextPageField class 5386
TAdapterPasswordField class 5395
TAdapterPrevPageAction class 5402
TAdapterUserNameField class 5438
TBaseAdapterAction class 5474
TEndUserAdapterLoginFormAction class 5782
TEndUserAdapterLogoutAction class 5790
TLoginFormAdapterLoginAction class 5842
DisplayName
IWebEndUser interface 5994
TService class 3079
DisplayOffset
TCustomScreenTipsManager class 313
TScreenTipsManager class 339
DisplayOptions
TValueListEditor class 4812
DisplayRows
TAdapterDisplayField class 6376
TAdapterEditColumn class 6392
TCustomAdapterDisplayField class 6472
DisplayShadow
TXPStylePopupMenu class 7780
DisplayType
TAdapterActionButton class 6320
TAdapterDisplayColumn class 6368
TAdapterDisplayField class 6376
TAdapterEditColumn class 6392
TCustomAdapterActionButton class 6439
TCustomAdapterDisplayColumn class 6466

TCustomAdapterDisplayField class 6473
DisplayWidth
TAdapterBooleanField class 5228
TAdapterDisplayField class 6377
TAdapterEditColumn class 6392
TAdapterField class 5272
TAdapterFileField class 5287
TAdapterImageField class 5314
TAdapterMemoField class 5329
TAdapterMultiValueField class 5338
TAdapterNamedDisplayField class 5350
TAdapterNextPageField class 5386
TAdapterPasswordField class 5395
TAdapterUserNameField class 5438
TCustomAdapterDisplayField class 6473
DitherBackground
TTabSet class 4489
DoAccept
TCommonDialogAction class 1787
TCustomTcpServer class 1410
DoActivate
TAbstractSocket class 16
TClientSocket class 27
TCustomServerSocket class 42
DoAdapterRequestNotHandled
TCustomAdapterDispatcher class 6171
DoAddDockClient
TToolDockForm class 4636
DoAfterClose
TXMLDocument class 7359
DoAfterDispatch
TMultiModuleWebAppServices class 6226
DoAfterDispatchAdapterRequest
TCustomAdapterDispatcher class 6172
DoAfterExecuteActionRequest
TBaseAdapterAction class 5453
DoAfterGetContent
TBaseAdapterPageProducer class 1148
DoAfterGetExecuteActionResponse
TBaseAdapterAction class 5453
DoAfterOpen
TXMLDocument class 7359
doAttrNull enumeration member 7718
doAutoColResize enumeration member 4892
doAutoPrefix enumeration member 7718
doAutoSave enumeration member 7718
DoBeforeClose
TXMLDocument class 7359
DoBeforeDispatch
TMultiModuleWebAppServices class 6226
DoBeforeDispatchAdapterRequest
TCustomAdapterDispatcher class 6172
DoBeforeExecuteActionRequest
TBaseAdapterAction class 5453
DoBeforeGetContent
TBaseAdapterPageProducer class 1149
DoBeforeGetExecuteActionResponse
TBaseAdapterAction class 5454
DoBeforeOpen
TXMLDocument class 7360
DoCallback
TBrowseForFolder class 1765
DoCancel
TCommonDialogAction class 1787
DocBindingInfo
TXMLDocument class 7383
DockSite
TGroupBox class 2546
doColumnTitles enumeration member 4892
DoConnect
TCustomIpClient class 1403
TSoapConnection class 1191
TSocketConnection class 224
TSQLConnection class 1589
TStreamedConnection class 240
TWebConnection class 258
DoContinue
TService class 3074

DoCreateHandle
TBaseSocket class 1373
DoCreateObject
TDataBlockInterpreter class 211
DoCreateScreenTip
TCustomScreenTipsManager class 309
DoCreateThread
TServerWinSocket class 126
DocSource
TXMLDocument class 7383
doctype
IDOMDocument interface 7537
Document
IWebBrowser interface 746
IWebBrowser2Disp interface 770
IWebBrowserAppDisp interface 790
IWebBrowserDisp interface 799
TInternetExplorer class 450
TWebBrowser class 526
TWebBrowser_V1 class 582
Documentation
IDocumentationCollection interface 7104
IDocumented interface 7106
DocumentComplete
DWebBrowserEvents2 interface 628
documentElement
IDOMDocument interface 7537
DocumentElement
IXMLDocument interface 7663
TXMLDocument class 7383
DocumentObject
IXMLDocumentAccess interface 7479
DoCustomAction
TDataBlockInterpreter class 211
DoCustomControl
TService class 3074
DoDestroyHandle
TBaseSocket class 1373
DoDisconnect
TCustomIpClient class 1404
TSoapConnection class 1192
TSQLConnection class 1589
TStreamedConnection class 240
DoDrawText
TCustomLabel class 2386
DoDuplicateCreateError
TCustomScreenTipsManager class 309
DoEndSession
TCustomSessionsService class 6750
DoError
TStreamedConnection class 241
DoExit
TValueListEditor class 4792
DoFindData
TCustomListBox class 2407
DoFreeObject
TDataBlockInterpreter class 212
DoFunction
IVarInvokeable interface 5024
TInvokeableVariantType class 5018
DoGetAppServerList
TDataBlockInterpreter class 212
DoGetData
TCustomListBox class 2407
DoGetDataObject
TCustomListBox class 2408
DoGetIDsOfNames
TDataBlockInterpreter class 212
DoGetScreenTipClass
TCustomScreenTipsManager class 309
DoGetServerList
TDataBlockInterpreter class 212
DoHandleError
TBaseSocket class 1373
DoHandleException
TServiceApplication class 3092
DoInterrogate
TService class 3074

DoInvoke
TDataBlockInterpreter class 213
doKeyColFixed enumeration member 4892
DoListen
TCustomWinSocket class 70
Domain
TBaseSocket class 1386
TIpSocket class 1423
DOMDocument
IXMLDocument interface 7663
TXMLDocument class 7384
DOMElement
IXMLNodeAccess interface 7484
TXMLNode class 7412
DOMException class 7506
about DOMException class 7506
code 7508
DOMException fields 7508
DOMException Members 7506
domImplementation
IDOMDocument interface 7537
DOMImplementation
TDOMVendor class 7515
TXMLDocument class 7385
DOMNode
IXMLNode interface 7691
TXMLNode class 7440
DOMParseOptions
TXMLDocument class 7385
DOMPersist
IXMLDocumentAccess interface 7480
TXMLDocument class 7386
DOMVendor
TXMLDocument class 7386
doNamespaceDecl enumeration member 7718
DoneModule
TMultiModuleWebAppServices class 6227
doNodeAutoCreate enumeration member 7718
doNodeAutoIndent enumeration member 7718
DoNodeChange
TXMLDocument class 7360
TXMLNode class 7413
DoNotify
TXMLNodeList class 7466
DoOnHasRights
TBaseEndUserAdapter class 5505
DoOnThemeChange
TThemeServices class 4583
DoOnValidate
TValueListEditor class 4792
DoOnValidateValue
TAdapterUpdateField class 5426
TCustomAdapterField class 5602
TCustomAdapterFileField class 5609
TCustomAdapterImageField class 5628
TCustomAdapterMemoField class 5642
TCustomAdapterMultiValueField class 5651
DoOpen
TCustomWinSocket class 71
DoPause
TService class 3075
DoProcedure
IVarInvokeable interface 5025
TInvokeableVariantType class 5019
DoProgress
TCustomZStream class 7906
DoReceive
TBaseSocket class 1374
DoScreenTipsLoaded
TCustomScreenTipsManager class 309
DoSend
TBaseSocket class 1374
DoSetMaxLength
TCustomEdit class 2354
DoSetTextHint
TCustomComboBox class 2328
TCustomEdit class 2355
DoShow

TCustomScreenTipsManager class 309
DoShowScreenTip
TCustomScreenTipsManager class 310
DoShutdown
TService class 3075
DoStart
TService class 3075
DoStartSession
TCustomSessionsService class 6750
DoStop
TService class 3075
DoTagEvent
TBaseAdapterPageProducer class 1149
DoTerminate
TServerClientThread class 99
DoTransform
TXMLTransform class 7729
double
Currency class 3252
TDateTime class 3340
Variant class 3460
double*
Variant class 3460
DoubleBuffered
TButton class 2050
TCheckBox class 2132
TComboBox class 2200
TEdit class 2478
TGroupBox class 2547
TListBox class 2672
TMemo class 2750
TRadioButton class 2829
TScrollBar class 2892
TStaticText class 2959
TTabbedNotebook class 4418
TTabSet class 4489
TValueListEditor class 4812
DoubleClick
DShellNameSpaceEvents interface 618
Doubler
IIntegerTest interface 1333
TFTCompTI class 1315
DoValidate
TBrowseForFolder class 1766
DownloadBegin
DWebBrowserEvents interface 622
DWebBrowserEvents2 interface 629
DownloadComplete
DWebBrowserEvents interface 623
DWebBrowserEvents2 interface 629
dpPublished enumeration member 1126
DragCanceled
TCustomListBox class 2408
DragCursor
TButton class 2052
TCheckBox class 2134
TComboBox class 2202
TEdit class 2480
TGroupBox class 2549
TLabel class 2620
TListBox class 2674
TMemo class 2752
TRadioButton class 2831
TScrollBar class 2894
TStaticText class 2961
TTabSet class 4491
TValueListEditor class 4814
TWebBrowser class 526
TWebBrowser_V1 class 582
DragKind
TButton class 2052
TCheckBox class 2134
TComboBox class 2202
TEdit class 2481
TGroupBox class 2550
TLabel class 2620
TListBox class 2674
TMemo class 2753

TRadioButton class 2831
TScrollBar class 2894
TStaticText class 2961
TTabSet class 4491
TValueListEditor class 4814
DragMode
TButton class 2053
TCheckBox class 2135
TComboBox class 2202
TEdit class 2481
TGroupBox class 2550
TLabel class 2621
TListBox class 2674
TMemo class 2753
TRadioButton class 2832
TScrollBar class 2895
TStaticText class 2962
TTabSet class 4492
TValueListEditor class 4814
TWebBrowser class 527
TWebBrowser_V1 class 583
DrawBackground
TScreenTipsWindow class 355
TStandardAddRemoveItem class 1743
TStandardMenuButton class 1749
TStandardMenuExpandBtn class 1751
TStandardMenuItem class 1753
TThemedAddRemoveItem class 4557
TThemedButtonControl class 4560
TThemedDropDownButton class 4564
TThemedExpandBtn class 4566
TThemedMenuButton class 4568
TThemedMenuItem class 4570
TThemedToolScrollBtn class 4576
TXPStyleAddRemoveItem class 7765
TXPStyleButton class 7767
TXPStyleDropDownBtn class 7771
TXPStyleExpandBtn class 7773
TXPStyleMenuButton class 7774
TXPStyleMenuItem class 7777
TXPStyleToolScrollBtn class 7782
DrawBorder
TScreenTipsWindow class 356
DrawCell
TValueListEditor class 4792
DrawDesignFocus
TStandardMenuButton class 1749
TStandardMenuItem class 1753
DrawDragDockImage
TToolDockObject class 4639
DrawEdge
TStandardMenuItem class 1754
TThemeServices class 4584
DrawElement
TThemeServices class 4585
DrawFooter
TScreenTipsWindow class 356
DrawFrame
TStandardButtonControl class 1744
TStandardDropDownButton class 1748
TStandardMenuButton class 1750
TStandardMenuExpandBtn class 1751
TStandardToolScrollBtn class 1759
TThemedButtonControl class 4560
TThemedDropDownButton class 4564
TThemedExpandBtn class 4567
TXPStyleButton class 7768
TXPStyleDropDownBtn class 7772
TXPStyleExpandBtn class 7773
DrawGlyph
TStandardAddRemoveItem class 1743
TStandardButtonControl class 1745
TStandardMenuItem class 1754
TThemedAddRemoveItem class 4558
TThemedButtonControl class 4560
TThemedDropDownButton class 4565
TThemedMenuItem class 4571
TXPStyleAddRemoveItem class 7766

TXPStyleButton class 7768
TXPStyleDropDownBtn class 7772
TXPStyleMenuItem class 7777
DrawGlyphFrame
TStandardMenuItem class 1754
DrawHeader
TScreenTipsWindow class 356
DrawIcon
TThemeServices class 4585
DrawItem
TCustomComboBox class 2328
TCustomListBox class 2408
DrawLargeGlyph
TStandardMenuItem class 1755
TThemedButtonControl class 4561
TThemedDropDownButton class 4565
TThemedMenuItem class 4571
DrawParentBackground
TThemeServices class 4586
DrawSeparator
TStandardMenuItem class 1755
TThemedAddRemoveItem class 4558
TThemedMenuItem class 4571
TXPStyleAddRemoveItem class 7766
TXPStyleMenuItem class 7777
DrawShadowedText
TThemedMenuItem class 4572
TXPStyleMenuItem class 7777
DrawSubMenuGlyph
TThemedMenuItem class 4572
TXPStyleMenuItem class 7778
DrawTab
TTabSet class 4475
DrawText
TStandardButtonControl class 1745
TStandardMenuButton class 1750
TStandardMenuItem class 1755
TThemedAddRemoveItem class 4558
TThemedButtonControl class 4561
TThemedMenuButton class 4569
TThemedMenuItem class 4572
TThemeServices class 4586
TXPStyleAddRemoveItem class 7766
TXPStyleButton class 7768
TXPStyleMenuButton class 7775
TXPStyleMenuItem class 7778
DrawUnusedEdges
TThemedAddRemoveItem class 4559
TThemedMenuItem class 4572
TXPStyleAddRemoveItem class 7766
TXPStyleMenuItem class 7778
DriverName
TConnectionData class 1520
TSQLConnection class 1612
DropDown
TCustomCombo class 2296
DropDownCount
TComboBox class 2204
TCustomCombo class 2306
DropDownMenu
TButton class 2054
TCustomButton class 2269
DropDownRows
TValueListEditor class 4816
DroppedDown
TCustomCombo class 2306
DShellNameSpaceEvents interface 618
about DShellNameSpaceEvents interface 618
DoubleClick 618
DShellNameSpaceEvents Members 618
DShellNameSpaceEvents methods 618
FavoritesSelectionChange 619
Initialized 619
SelectionChange 619
DShellWindowsEvents interface 619
about DShellWindowsEvents interface 619
DShellWindowsEvents Members 620
DShellWindowsEvents methods 620

WindowRegistered 620
WindowRevoked 620
dtHelp enumeration member 363
dtManual enumeration member 363
dtResource enumeration member 363
Duplicates
TWideStringList class 6822
DWebBrowserEvents interface 621
about DWebBrowserEvents interface 621
BeforeNavigate 622
CommandStateChange 622
DownloadBegin 622
DownloadComplete 623
DWebBrowserEvents Members 621
DWebBrowserEvents methods 621
FrameBeforeNavigate 623
FrameNavigateComplete 623
FrameNewWindow 623
NavigateComplete 624
NewWindow 624
ProgressChange 624
PropertyChange 624
Quit 624
StatusTextChange 625
TitleChange 625
WindowActivate 625
WindowMove 625
WindowResize 626
DWebBrowserEvents2 interface 626
about DWebBrowserEvents2 interface 626
BeforeNavigate2 628
ClientToHostWindow 628
CommandStateChange 628
DocumentComplete 628
DownloadBegin 629
DownloadComplete 629
DWebBrowserEvents2 Members 626
DWebBrowserEvents2 methods 627
FileDownload 629
NavigateComplete2 629
NavigateError 629
NewWindow2 630
OnFullScreen 630
OnMenuBar 630
OnQuit 630
OnStatusBar 631
OnTheaterMode 631
OnToolBar 631
OnVisible 631
PrintTemplateInstantiation 631
PrintTemplateTeardown 632
PrivacyImpactedStateChange 632
ProgressChange 632
PropertyChange 632
SetSecureLockIcon 633
StatusTextChange 633
TitleChange 633
UpdatePageStatus 633
WindowClosing 633
WindowSetHeight 634
WindowSetLeft 634
WindowSetResizable 634
WindowSetTop 634
WindowSetWidth 635
DynamicArray
DynamicArray template 3264
DynamicArray template 3261
[] 3269
[] operator 3268
~DynamicArray 3263
= 3269
== 3269
about DynamicArray template 3261
AllocData 3265
Copy 3265
Copy method 3265
CopyRange 3266
CopyRange method 3266

DecRefCount 3266
DimCount 3267
DynamicArray 3264
DynamicArray constructor 3264
DynamicArray Members 3263
DynamicArray methods 3264
DynamicArray operators 3268
DynamicArray properties 3270
FreeData 3267
get_high 3267
get_length 3267
get_low 3267
High 3270
IncRefCount 3268
Length 3270
Low 3271
set_length 3268
SetData 3268
DynArrayException class 3271
about DynArrayException class 3271
DynArrayNullData class 3272
about DynArrayNullData class 3272
DynArrayOutOfRange
DynArrayOutOfRange class 3273
DynArrayOutOfRange class 3272
about DynArrayOutOfRange class 3272
DynArrayOutOfRange 3273
DynArrayOutOfRange data members 3273
DynArrayOutOfRange Members 3272
m_Count 3273
m_Index 3273

E

EAbort class 3916
about EAbort class 3916
EAbstractError class 3916
about EAbstractError class 3916
EAccessViolation class 3917
about EAccessViolation class 3917
EAdapterDispatcherException class 6146
about EAdapterDispatcherException class 6146
EAdapterException class 5177
about EAdapterException class 5177
CreateID 5178
EAdapterException Members 5177
EAdapterException properties 5178
ID 5178
EAdapterFieldException class 5178
about EAdapterFieldException class 5178
Create 5179
EAdapterFieldException Members 5179
EAdapterFieldException properties 5180
FieldName 5180
EAllocatedActionsException class 297
about EAllocatedActionsException class 297
EArgumentException class 3918
about EArgumentException class 3918
EArgumentOutOfRangeException class 3918
about EArgumentOutOfRangeException class 3918
EAssertionFailed class 3918
about EAssertionFailed class 3918
ebBottom enumeration member 4644
ebLeft enumeration member 4644
ebRight enumeration member 4644
ebTop enumeration member 4644
EchoActionFieldValue
IEchoActionFieldValue interface 5882
TAdapterNamedField class 5368
EchoActionFieldValues
IEchoAdapterFieldValues interface 5883
TCustomAdapter class 5572
ecLowerCase enumeration member 3000
ecNormal enumeration member 3000
eConnect enumeration member 1699
EControlC class 3919
about EControlC class 3919
EConvertError class 3919
about EConvertError class 3919

ecUpperCase enumeration member 3000
ECustomException class 1305
about ECustomException class 1305
CatchException 1310
Create 1307
Destroy 1309
ECustomException Members 1305
ECustomException methods 1310
InstanceCount 1310
ThrowExcept 1310
EdgeBorders
TToolWindow class 4642
EdgeInner
TToolWindow class 4642
EdgeOuter
TToolWindow class 4643
eDisconnect enumeration member 1699
EditHandle
TCustomCombo class 2307
EditList
TValueListEditor class 4817
EditListGetItems
TValueListEditor class 4793
EditMask
TItemProp class 4781
Editor
TWebComponentClassDescription class 6025
EditStyle
TItemProp class 4782
EditWndProc
TCustomCombo class 2296
EDivByZero class 3921
about EDivByZero class 3921
EDOMParseError class 7508
about EDOMParseError class 7508
Create 7510
EDOMParseError Members 7509
EDOMParseError properties 7511
ErrorCode 7511
FilePos 7511
Line 7512
LinePos 7512
ParseError 7512
Reason 7513
SrcText 7513
URL 7513
eeAccept enumeration member 147
eeConnect enumeration member 147
eeDisconnect enumeration member 147
eeGeneral enumeration member 147
eeLookup enumeration member 147
EEncodingError class 3922
about EEncodingError class 3922
eeReceive enumeration member 147
eeSend enumeration member 147
EExternal class 3922
about EExternal class 3922
EExternal fields 3925
EExternal Members 3923
ExceptionRecord 3925
EExternalException class 3925
about EExternalException class 3925
efldrRights
TFLDDesc class 1575
efldvVchk
TFLDDesc class 1575
EHeapException class 3926
about EHeapException class 3926
EHeapException Members 3927
EHeapException methods 3928
FreeInstance 3929
RaisingException 3929
EInOutError class 3929
about EInOutError class 3929
EInOutError fields 3932
EInOutError Members 3931
ErrorCode 3933
EInterpreterError class 191

about EInterpreterError class 191
EIntError class 3933
about EIntError class 3933
EIntfCastError class 3934
about EIntfCastError class 3934
EIntOverflow class 3934
about EIntOverflow class 3934
EInvalidCast class 3935
about EInvalidCast class 3935
EInvalidContainer class 3935
about EInvalidContainer class 3935
EInvalidInsert class 3936
about EInvalidInsert class 3936
EInvalidOp class 3936
about EInvalidOp class 3936
EInvalidPointer class 3937
about EInvalidPointer class 3937
Element
IPart interface 7138
ElementSize
AnsiStringBase class 3189
AnsiStringT template 3219
UnicodeString class 3397
ElevationRequired
TButton class 2055
TCustomButton class 2270
EllipsisPosition
TCustomLabel class 2395
TLabel class 2622
EMathError class 3937
about EMathError class 3937
EMonitor class 3938
about EMonitor class 3938
EMonitorLockException class 3938
about EMonitorLockException class 3938
Empty
Variant class 3431
EmptyDestinationDocument
TXMLTransform class 7732
Enabled
TBrowseForFolder class 1768
TButton class 2055
TCheckBox class 2136
TColorSelect class 1777
TComboBox class 2204
TEdit class 2482
TFileExit class 1812
TFileOpen class 1822
TFilePageSetup class 1839
TFilePrintSetup class 1848
TFileSaveAs class 1858
TFontEdit class 1867
TGroupBox class 2551
TLabel class 2623
TListBox class 2676
TMemo class 2755
TPrintDlg class 1887
TRadioButton class 2833
TScrollBar class 2896
TSearchFind class 1903
TSearchFindNext class 1916
TSearchReplace class 1925
TStaticText class 2963
TTabbedNotebook class 4420
TTabPage class 4448
TTabSet class 4493
TValueListEditor class 4817
ENamedVariantsError class 6118
about ENamedVariantsError class 6118
EncodeActionParams
TCustomAdapter class 5542
TDefaultFieldsPagedAdapter class 5754
EncodeAdapterItemRequestAsFieldValue
IAdapterDispatcher interface 986
TCustomAdapterDispatcher class 6173
EncodeAdapterItemRequestAsHREF
IAdapterDispatcher interface 986
TCustomAdapterDispatcher class 6173

EncodeString
ISearchAssistantOC interface 648
ISearchAssistantOC2Disp interface 660
ISearchAssistantOC3Disp interface 671
ISearchAssistantOCDisp interface 679
Encoding
IDOMXMLProlog interface 7606
IXMLDocument interface 7664
TXMLDocument class 7386
EncodingStyle
IHeaderInfo interface 7251
EndButtons
TAdapterFormLayout class 6411
EndConnect
TServerClientThread class 99
EndField
TAdapterFormLayout class 6411
EndFields
TAdapterFormLayout class 6411
EndIterator
IIteratorSupport interface 1056
TAdapterHiddenFieldsList class 5305
TAdapterSubComponent class 5409
TBaseAdapterAction class 5454
TBaseAdapterErrorsList class 5483
TCustomAdapter class 5542
TSiteModuleHelper class 6568
EndLayout
TAdapterFormLayout class 6412
EndMargin
TTabSet class 4494
EndOfPage
TDefaultFieldsPagedAdapter class 5754
EndRead
IReadWriteSync interface 4038
TMultiReadExclusiveWriteSynchronizer class 4001
TSimpleRWSync class 4006
EndRow
TAdapterFormLayout class 6412
EndTable
TAdapterFormLayout class 6412
EndUpdate
IXMLNodeList interface 7707
TAbstractNamedVariants class 6125
TConnectionData class 1518
TWideStrings class 6835
TXMLNodeList class 7467
EndUser
TAbstractWebContext class 5973
TCustomLoginFormAdapter class 5707
TCustomPageDispatcher class 6190
EndUserAdapter
IWebEndUser interface 5994
TCustomWebAppComponents class 6213
TWebAppComponents class 6247
EndValueIterator
IWebGetFieldValues interface 1093
TBaseAdapterMultiValueField class 5493
EndWrite
IReadWriteSync interface 4038
TMultiReadExclusiveWriteSynchronizer class 4001
TSimpleRWSync class 4006
ENoConstructException class 3939
about ENoConstructException class 3939
ENoMonitorSupportException class 3939
about ENoMonitorSupportException class 3939
EnsureAnsi
AnsiStringBase class 3189
AnsiStringT template 3220
EnsureCapacity
TStringBuilder class 4016
EnsureUnicode
UnicodeString class 3397
Enter
TCriticalSection class 3128
TCriticalSectionHelper class helper 3132
TMonitor record 3350, 3351
entities

IDOMDocumentType interface 7541
EnumOptions
IShellNameSpace interface 707
IShellNameSpaceDisp interface 715
TShellNameSpace class 481
EOSError class 3939
about EOSError class 3939
EOSError fields 3941
EOSError Members 3940
ErrorCode 3942
EOutOfMemory class 3942
about EOutOfMemory class 3942
EOverflow class 3942
about EOverflow class 3942
EPackageError class 3943
about EPackageError class 3943
EPageDispatcherException class 6147
about EPageDispatcherException class 6147
epEndEllipsis enumeration member 3001
epNone enumeration member 3001
epPathEllipsis enumeration member 3001
EPrivilege class 3943
about EPrivilege class 3943
EProgrammerNotFound class 3944
about EProgrammerNotFound class 3944
EPropertyConvertError class 4693
about EPropertyConvertError class 4693
EPropertyError class 4693
about EPropertyError class 4693
EPropReadOnly class 3944
about EPropReadOnly class 3944
EPropWriteOnly class 3945
about EPropWriteOnly class 3945
epSelectedLeft enumeration member 4538
epSelectedRight enumeration member 4538
epUnselectedLeft enumeration member 4538
epUnselectedRight enumeration member 4538
epWordEllipsis enumeration member 3001
Equals
IEquatable interface 3501
TAbstractNamedVariants class 6125
TObject class 3371
TStringBuilder class 4017
TWideStrings class 6835
ERangeError class 3945
about ERangeError class 3945
EraseDragDockImage
TToolDockObject class 4639
eReconnect enumeration member 1699
ERecursiveIncludeFile class 6647
about ERecursiveIncludeFile class 6647
ErrCode
TService class 3079
Error
TAbstractNamedVariants class 6126
TAbstractSocket class 16
TCustomSocket class 58
TCustomWinSocket class 71
TServerClientThread class 100
TWideStrings class 6836
ErrorCheck
TBaseSocket class 1374
errorCode
IDOMParseError interface 7589
ErrorCode
EDOMParseError class 7511
EInOutError class 3933
EOSError class 3942
ESafeArrayError class 5113
ErrorCount
IAdapterErrors interface 998
TBaseAdapterErrorsList class 5487
ErrorPageName
TAdapterActionButton class 6321
TCustomAdapterActionButton class 6439
Errors
IAdapterErrors interface 998
TBaseAdapterAction class 5475

TBaseAdapterErrorsList class 5487
TCustomAdapter class 5573
TScriptProducer class 6715
ErrorSeverity
TService class 3080
ErrorText
IAdapterError interface 994
ESafeArrayBoundsError class 5110
about ESafeArrayBoundsError class 5110
ESafeArrayCreateError class 5110
about ESafeArrayCreateError class 5110
ESafeArrayError class 5111
about ESafeArrayError class 5111
CreateHResult 5113
ErrorCode 5113
ESafeArrayError Members 5112
ESafeArrayError properties 5113
ESafeArrayLockedError class 5114
about ESafeArrayLockedError class 5114
ESafecallException class 3946
about ESafecallException class 3946
EScriptEvalError class 6648
about EScriptEvalError class 6648
esCritical enumeration member 3103
ESessionException class 6147
about ESessionException class 6147
ESessionsException class 374
about ESessionsException class 374
esIgnore enumeration member 3103
esLowered enumeration member 4645
esNone enumeration member 4645
esNormal enumeration member 3103
ESOAPHTTPException class 1272
about ESOAPHTTPException class 1272
Create 1274
CreateFmt 1274
ESOAPHTTPException Members 1272
ESOAPHTTPException properties 1275
StatusCode 1275
ESocketConnectionError class 192
about ESocketConnectionError class 192
ESocketError class 13, 1369
about ESocketError class 13, 1369
esRaised enumeration member 4645
esSevere enumeration member 3103
EStackOverflow class 3947
about EStackOverflow class 3947
esTerminate enumeration member 6773
esTimeout enumeration member 6773
etANSI enumeration member 6889
etFirstIsSel enumeration member 4538
etFirstNotSel enumeration member 4538
etLastIsSel enumeration member 4538
etLastNotSel enumeration member 4538
etNone enumeration member 4538
etNotSelToNotSel enumeration member 4538
etNotSelToSel enumeration member 4538
etSelToNotSel enumeration member 4538
etUSASCII enumeration member 6889
etUTF8 enumeration member 6889
ETypeTransException class 4680
about ETypeTransException class 4680
EUnderflow class 3947
about EUnderflow class 3947
eUserDataSet enumeration member 1700
EUserIDNotFoundException class 6779
about EUserIDNotFoundException class 6779
eUserMonitor enumeration member 1700
EValidateUserException class 6779
about EValidateUserException class 6779
Evaluate
TScriptProducer class 6710
EvaluateExpression
TScriptProducer class 6710
EVariantArrayCreateError class 4978
about EVariantArrayCreateError class 4978
EVariantArrayLockedError class 4979
about EVariantArrayLockedError class 4979

EVariantBadIndexError class 4980
about EVariantBadIndexError class 4980
EVariantBadVarTypeError class 4980
about EVariantBadVarTypeError class 4980
EVariantDispatchError class 4981
about EVariantDispatchError class 4981
EVariantError class 3948
about EVariantError class 3948
EVariantInvalidArgError class 4981
about EVariantInvalidArgError class 4981
EVariantInvalidNullOpError class 4982
about EVariantInvalidNullOpError class 4982
EVariantInvalidOpError class 4983
about EVariantInvalidOpError class 4983
EVariantNotAnArrayError class 4983
about EVariantNotAnArrayError class 4983
EVariantNotImplError class 4984
about EVariantNotImplError class 4984
EVariantOutOfMemoryError class 4984
about EVariantOutOfMemoryError class 4984
EVariantOverflowError class 4985
about EVariantOverflowError class 4985
EVariantRangeCheckError class 4986
about EVariantRangeCheckError class 4986
EVariantTypeCastError class 4986
about EVariantTypeCastError class 4986
EVariantUnexpectedError class 4987
about EVariantUnexpectedError class 4987
evConst enumeration member 1336
Event
TAbstractSocket class 17
TCustomSocket class 58
TCustomWinSocket class 72
TServerClientThread class 100
EventHandled
ISearchAssistantOC interface 655
ISearchAssistantOC2Disp interface 665
ISearchAssistantOC3Disp interface 675
ISearchAssistantOCDisp interface 683
evNormal enumeration member 1336
evVar enumeration member 1336
EWebModuleException class 6535
about EWebModuleException class 6535
EWebUsersListException class 6780
about EWebUsersListException class 6780
EWin32Error class 3949
about EWin32Error class 3949
EWSDLLoadException class 7266
about EWSDLLoadException class 7266
exceptCommand enumeration member 1702
exceptConnection enumeration member 1702
exceptCursor enumeration member 1702
Exception class 3949
about Exception class 3949
BaseException 3964
CleanUpStackInfoProc 3961
Create 3952
CreateFmt 3954
CreateFmtHelp 3955
CreateHelp 3955
CreateRes 3956
CreateRes constructor 3956
CreateResFmt 3957
CreateResFmt constructor 3957
CreateResFmtHelp 3958
CreateResFmtHelp constructor 3958
CreateResHelp 3959
CreateResHelp constructor 3959
Destroy 3960
Exception fields 3960
Exception Members 3950
Exception methods 3961
Exception properties 3964
GetBaseException 3962
GetExceptionStackInfoProc 3961
GetStackInfoStringProc 3961
GetStackTrace 3962
HelpContext 3964

InnerException 3965
Message 3965
RaiseOuterException 3962
RaisingException 3962
SetInnerException 3962
SetStackInfo 3963
StackInfo 3966
StackTrace 3966
ThrowOuterException 3963
ToString 3963
ExceptionRecord
EExternal class 3925
exceptMetaData enumeration member 1702
exceptUseLast enumeration member 1702
Exchange
TAbstractNamedVariants class 6126
TValueListStrings class 4889
TWideStringList class 6816
TWideStrings class 6836
ExecProc
TSQLStoredProc class 1675
ExecSQL
TCustomSQLDataSet class 1529
TSQLDataSet class 1636
TSQLQuery class 1658
Execute
IWebNode interface 6618
TClientSocketThread class 1398
THTTPReqResp class 1283
TLinkedWebNode class 1324
TLogLinkedWebNode class 1328
TServerAcceptThread class 91
TServerClientThread class 101
TServerSocketThread class 1436
TServiceThread class 3097
TSoapDataModule class 1218
TSQLConnection class 1590
TTransportThread class 250
TWSDLClientNode class 7268, 7269
ExecuteAccess
TAdapterAction class 5206
TAdapterGotoPageAction class 5296
TAdapterNextPageAction class 5376
TAdapterPrevPageAction class 5403
TBaseAdapterAction class 5475
TCustomAdapter class 5573
ExecuteActionRequest
TBaseAdapterAction class 5455
ExecuteDirect
TSQLConnection class 1590
ExecuteLogin
TBaseEndUserAdapter class 5505
TCustomEndUserAdapter class 5683
TCustomEndUserSessionAdapter class 5699
TCustomLoginFormAdapter class 5707
ExecuteLoginForm
TBaseEndUserAdapter class 5506
ExecuteLogout
TBaseEndUserAdapter class 5506
TCustomEndUserAdapter class 5683
TCustomEndUserSessionAdapter class 5699
ExecuteMethod
TSqlServerMethod class 1669
ExecuteResult
TCommonDialogAction class 1789
ExecuteScript
IInternetExecuteScript interface 1046
ExecuteStatement
TCustomSQLDataSet class 1529
ExecuteSyncProc
TClientSocketThread class 1398
ExecuteTarget
TBrowseForFolder class 1766
TCommonDialogAction class 1787
TEditCopy class 1796
TEditCut class 1798
TEditDelete class 1800
TEditPaste class 1802

TEditSelectAll class 1804
TEditUndo class 1807
TFileExit class 1811
TFileOpen class 1820
TFileOpenWith class 1831
THelpContents class 1877
THelpContextAction class 1878
THelpOnHelp class 1880
THelpTopicSearch class 1881
TSearchAction class 1897
TSearchFindNext class 1914
TSearchReplace class 1923
TWindowArrange class 1937
TWindowCascade class 1938
TWindowClose class 1940
TWindowMinimizeAll class 1942
TWindowTileHorizontal class 1944
TWindowTileVertical class 1946
ExecuteValue
IExecuteValue interface 5884
ExecuteVerb
IWebComponentCommands interface 6098
ExecWB
IWebBrowser2 interface 752
IWebBrowser2Disp interface 765
TInternetExplorer class 429, 430
TWebBrowser class 508
Exists
TAllocatedActions class 299
TGeneratedFunctions class 6508
Exit
TMonitor record 3351
EXMLDocError class 7639
about EXMLDocError class 7639
Expand
IShellNameSpace interface 702
IShellNameSpaceDisp interface 711
TShellNameSpace class 472
ExpirationTime
IWebSessionAttributes interface 1106
TSessionItem class 381
TWebSessionImpl class 6770
Expired
IWebSessionAttributes interface 1107
TSessionItem class 381
TWebSessionImpl class 6770
Export
IShellFavoritesNameSpace interface 692
IShellFavoritesNameSpaceDisp interface 697
IShellNameSpaceDisp interface 711
TShellNameSpace class 472
ExportedObjectOnly1
TSocketForm class 163
EXSDateTimeException class 7795
about EXSDateTimeException class 7795
EXSDecimalException class 7796
about EXSDecimalException class 7796
EXSHexBinaryException class 7796
about EXSHexBinaryException class 7796
EXSLPageProducerError class 7878
about EXSLPageProducerError class 7878
Ext
TLanguages class 3989
ExtendedSelect
TCustomListBox class 2427
TListBox class 2676
ExtractName
TWideStrings class 6837
ExtractRequestParams
TCustomAdapter class 5543
TDefaultFieldsPagedAdapter class 5754
EZCompressionError class 7904
about EZCompressionError class 7904
EZDecompressionError class 7904
about EZDecompressionError class 7904
EZeroDivide class 3966
about EZeroDivide class 3966
EZLibError class 7905

about EZLibError class 7905

F

Factories
TAbstractScriptObjectFactories class 6655
TScriptObjectFactories class 6706
TWebModuleList class 6633
FailurePage
IAdapterItemRequestParams interface 1008
IGetAdapterRequestDefaultResponse interface 1026
TAdapterItemRequestParams class 6165
Fault
IFaults interface 7110
Faults
IBindingOperation interface 7086
IOperation interface 7126
FavoritesSelectionChange
DShellNameSpaceEvents interface 619
FCalcField
TFLDDesc class 1572
FCharPos
TScriptError class 6678
FClient
TButtonActionLink class 2107
FConditionVariable
TConditionVariableCS class 3115
FControl
TSearchAction class 1896
FConverter
TSoapPascalInvoker class 1352
FData
TStringBuilder class 4012
FDateTime
TXSCustomDateTime class 7806
FDefEditProc
TCustomCombo class 2286
FDefListProc
TCustomCombo class 2287
FDescription
TScriptError class 6678
FDialog
TCommonDialogAction class 1786
FDomConverter
TSoapPascalInvoker class 1352
FDropHandle
TCustomCombo class 2287
FDroppingDown
TCustomCombo class 2287
FEditHandle
TCustomCombo class 2288
FEditInstance
TCustomCombo class 2288
FErrors
TScriptProducer class 6709
FetchClientSocketThread
TServerSocketThread class 1437
FetchOnDemand
TSimpleDataSet class 910
FetchParams
TXMLBroker class 7310
ffCurrency enumeration member 4323
ffExponent enumeration member 4323
ffFixed enumeration member 4323
ffGeneral enumeration member 4323
FFileName
TScriptError class 6679
FFindFirst
TSearchAction class 1897
FFldNum
TFLDDesc class 1572
FFLDRights
TFLDDesc class 1572
FFldType
TFLDDesc class 1572
FFLDVchk
TFLDDesc class 1573
ffNumber enumeration member 4323
FFocusChanged

TCustomCombo class 2288
FHandle
THandleObject class 3140
FHeaderProcessors
TSOAPDOMProcessor class 1235
FHeadersIn
TSoapPascalInvoker class 1353
FHeadersOut
TSoapPascalInvoker class 1353
Field
IAdapterError interface 994
TFieldValuesAdapter class 5812
TFieldValuesValueField class 5817
FieldAddress
TObject class 3372
FieldConditions
TAdapterDataDisplay class 6357
FieldCount
IAdapterHiddenFields interface 1005
TAdapterHiddenFieldsList class 5308
FieldDefs
TSimpleDataSet class 911
FieldIndent
TAdapterFormLayout class 6418
FieldName
EAdapterFieldException class 5180
IWebGetFieldName interface 1091
TAdapterBooleanField class 5228
TAdapterDataDisplay class 6363
TAdapterDisplayColumn class 6368
TAdapterDisplayField class 6377
TAdapterEditColumn class 6393
TAdapterField class 5272
TAdapterFileField class 5288
TAdapterImageField class 5315
TAdapterMemoField class 5329
TAdapterMultiValueField class 5339
TAdapterNamedField class 5369
TAdapterNextPageField class 5387
TAdapterPasswordField class 5396
TAdapterUserNameField class 5438
FieldNames
TAdapterErrorsList class 5266
Fields
IAdapterHiddenFields interface 1005
TAdapterHiddenFieldsList class 5308
FileCount
TCustomAdapterFileField class 5612
FileDate
TDateTime class 3330
FileDateToDateTime
TDateTime class 3331
FileDownload
DWebBrowserEvents2 interface 629
FileExt
TAbstractInetFileType class 967
FileExtCount
TAbstractInetFileType class 967
FileName
IXMLDocument interface 7664
TFileAction class 1809
TFileOpenWith class 1832
TSessions class 395
TSimpleDataSet class 914
TSQLMonitor class 1650
TXMLDocument class 7387
FileNames
TCustomAdapterFileField class 5612
filePos
IDOMParseError interface 7590
FilePos
EDOMParseError class 7511
Files
TUpdateFileList class 5865
Filter
TSimpleDataSet class 915
Filtered
TSimpleDataSet class 917

FilterOptions
TSimpleDataSet class 917
FinalizeStream
IMimeAttachmentHandler interface 1178
Find
TAllocatedActions class 300
TAssociateItemList class 304
TDOMVendorList class 7518
TScriptEnginesList class 6675
TWideStringList class 6816
FindAction
IWebActionsList interface 1077
TAdapterActionsListHelper class 6327
TCustomAdapter class 5543
FindApplicationModule
TAbstractWebContext class 5964
TWebContext class 5980
FindAssociatedAction
TCustomAdapterActionButton class 6433
FindAssociatedField
TAdapterDataDisplay class 6357
FindComputer
ISearchAssistantOC interface 649
ISearchAssistantOC2Disp interface 660
ISearchAssistantOC3Disp interface 671
ISearchAssistantOCDisp interface 679
FindDefaultPageInfo
TAbstractWebContext class 5964
TWebContext class 5980
FindField
IWebDataFields interface 1082
TAdapterControlGroup class 6346
TCustomAdapter class 5543
FindFileExt
IInternetFileTypes interface 1047
FindFilesOrFolders
ISearchAssistantOC interface 649
ISearchAssistantOC2Disp interface 661
ISearchAssistantOC3Disp interface 671
ISearchAssistantOCDisp interface 679
FindGlobalVariable
TVariableLookup class 6717
FindGlobalVariableContainer
TVariableLookup class 6718
FindHeaderProcessor
TSOAPDOMProcessor class 1238
FindHostedNode
IXMLNodeAccess interface 7484
TXMLNode class 7413
FindIncludeFile
IFindIncludeFile interface 1017
TCustomLocateFileService class 6182
FindItemProp
TValueListStrings class 4889
FindModuleClass
TAbstractWebContext class 5965
TWebContext class 5981
TWebModuleList class 6631
FindModuleName
TAbstractWebContext class 5965
TWebContext class 5981
TWebModuleList class 6631
FindNamedAction
TCustomAdapter class 5544
FindNamedField
TCustomAdapter class 5544
FindNamedItemIntf
TScriptGlobalObjects class 6690
FindNamedVariable
TAdapterSubComponent class 5409
TCustomAdapter class 5544
TSiteModuleHelper class 6568
FindNamespaceDecl
IXMLNode interface 7674
TXMLNode class 7414
FindNamespaceURI
IXMLNode interface 7674
TXMLNode class 7414

FindNode
IXMLNodeList interface 7708
TXMLNodeList class 7468
FindOnWeb
ISearchAssistantOC interface 649
ISearchAssistantOC2Disp interface 661
ISearchAssistantOC3Disp interface 672
ISearchAssistantOCDisp interface 680
FindOrAddSchema
IWebServExp interface 6736
FindPage
TAbstractWebContext class 5966
TWebContext class 5982
FindPageInfo
TAbstractWebContext class 5966
TWebContext class 5982
FindPeople
ISearchAssistantOC interface 649
ISearchAssistantOC2Disp interface 661
ISearchAssistantOC3Disp interface 672
ISearchAssistantOCDisp interface 680
FindPrinter
ISearchAssistantOC interface 649
ISearchAssistantOC2Disp interface 661
ISearchAssistantOC3Disp interface 672
ISearchAssistantOCDisp interface 680
FindRow
TValueListEditor class 4793
FindScriptProducerClass
TScriptEnginesList class 6676
FindSibling
IXMLNodeList interface 7709
TXMLNodeList class 7468
FindStream
TCustomLocateFileService class 6183
FindTemplateFile
TCustomLocateFileService class 6183
FindUserID
TWebUserItems class 6793
FindUserName
TWebUserItems class 6793
FindVariable
TAdapterSubComponent class 5410
TCustomAdapter class 5545
TSiteModuleHelper class 6569
FindVariableContainer
TAbstractWebContext class 5967
TAdapterSubComponent class 5410
TCustomAdapter class 5545
TSiteModuleHelper class 6569
TWebContext class 5983
FindWindowSW
IShellWindows interface 726
IShellWindowsDisp interface 730
TShellWindows class 499
FInfo
TTimeZone record 1728
FinishContext
TMultiModuleWebAppServices class 6227
FInstance
TClassHelperBase class 3320
fireEvent
TFTCompT class 1312
fireEventEx
TFTCompT class 1312
First
IXMLNodeList interface 7709
TXMLNodeList class 7469
firstChild
IDOMNode interface 7569
FirstIndex
TTabSet class 4494
FIsFocused
TCustomCombo class 2288
FIsSingleByte
TEncoding class 3973
FixedColor
TValueListEditor class 4817

FixedCols
TValueListEditor class 4818
FixupIdent
TInvokeableVariantType class 5020
Flags
IShellNameSpace interface 708
IShellNameSpaceDisp interface 715
TShellNameSpace class 481
TWebContentOptions class 6077
FLastError
THandleObject class 3140
FLen
TFLDDesc class 1573
FLength
TStringBuilder class 4012
FLine
TScriptError class 6679
FLineNumberMap
TScriptFile class 6685
FList
TWebNotifyContainer class 6602
FListHandle
TCustomCombo class 2289
FListInstance
TCustomCombo class 2289
float
Variant class 3460
float*
Variant class 3460
FloatToStrF
AnsiStringBase class 3190
AnsiStringT template 3220
UnicodeString class 3397
WideString class 3477
FMaxCapacity
TStringBuilder class 4012
FMaxCharSize
TEncoding class 3974
FMoving
TCustomListBox class 2401
FmtLoadStr
AnsiStringBase class 3190
AnsiStringT template 3221
UnicodeString class 3398
FName
TFLDDesc class 1573
FNext
TTransactionItem class 1695
FNullOffset
TFLDDesc class 1573
FocusControl
TCustomLabel class 2395
TCustomStaticText class 2462
TLabel class 2623
TStaticText class 2964
Focused
TCustomCombo class 2297
FOffset
TFLDDesc class 1573
Folder
TBrowseForFolder class 1768
Font
TButton class 2056
TCheckBox class 2137
TComboBox class 2205
TCustomScreenTipsManager class 313
TEdit class 2483
TGroupBox class 2552
TLabel class 2623
TListBox class 2677
TMemo class 2755
TRadioButton class 2834
TScreenTipsManager class 339
TStaticText class 2964
TTabbedNotebook class 4420
TTabSet class 4494
TValueListEditor class 4818
Footer

TCustomScreenTipsManager class 313
TScreenTipsManager class 340
TScreenTipsWindow class 358
FooterImage
TCustomScreenTipsManager class 314
TScreenTipsManager class 340
TScreenTipsWindow class 358
Form
TWindowAction class 1935
Format
AnsiStringBase class 3191
AnsiStringT template 3221
UnicodeString class 3399
WideString class 3477
FormatCaption
TAdapterDataDisplay class 6357
FormatColumnData
TAdapterDataDisplay class 6358
TCustomAdapterCommandColumn class 6445
FormatColumnHeading
TAdapterDataDisplay class 6358
TCustomAdapterCommandColumn class 6446
FormatFloat
AnsiStringBase class 3191
AnsiStringT template 3222
UnicodeString class 3399
WideString class 3478
FormatImage
TCustomAdapterDisplayColumn class 6464
FormatListDisplay
TCustomAdapterDisplayColumn class 6464
FormatString
TDateTime class 3331
FormatTextDisplay
TCustomAdapterDisplayColumn class 6464
FormCloseQuery
TSocketForm class 176
FormCreate
TSocketForm class 176
FormDestroy
TSocketForm class 176
FormMethod
TAdapterForm class 6406
TCustomAdapterForm class 6495
FormShow
TSocketForm class 176
fpLoginRequired enumeration member 6004
fpPublished enumeration member 6004
FractionalSeconds
TXSCustomDateTime class 7817
TXSTime class 7857
FractionalSecondString
TXSTime class 7857
FrameBeforeNavigate
DWebBrowserEvents interface 623
FrameClass
TWebComponentClassDescription class 6025
FrameNavigateComplete
DWebBrowserEvents interface 623
FrameNewWindow
DWebBrowserEvents interface 623
Free
TConditionVariableHelper class helper 3119
TCriticalSectionHelper class helper 3132
TObject class 3372
FreeBuffers
TCustomSQLDataSet class 1530
FreeCommand
TCustomSQLDataSet class 1530
FreeData
DynamicArray template 3267
FreeEncodings
TEncoding class 3975
FreeInstance
EHeapException class 3929
TObject class 3374
FreeReader
TCustomSQLDataSet class 1530

FRefCount
TInterfacedObject class 3344
friend Currency __fastcall operator -(double lhs, const Currency& rhs) 3253
friend Currency __fastcall operator -(int lhs, const Currency& rhs) 3253
friend Currency __fastcall operator *(double lhs, const Currency& rhs) 3254
friend Currency __fastcall operator *(int lhs, const Currency& rhs) 3254
friend Currency __fastcall operator /(double lhs, const Currency& rhs) 3254
friend Currency __fastcall operator /(int lhs, const Currency& rhs) 3254
friend Currency __fastcall operator +(double lhs, const Currency& rhs) 3255
friend Currency __fastcall operator +(int lhs, const Currency& rhs) 3255
friend UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) 3420
friend UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) 3420
FSaveIndex
TCustomCombo class 2289
FScriptEngine
TScriptProducer class 6709
FScriptFile
TScriptProducer class 6709
FSection
TCriticalSection class 3128
FServerSocket
TCustomServerSocket class 42
FSoapDispatcher
THTTPSoapDispatchNode class 1259
FSOAPHeaders
TSoapPascalInvoker class 1353
FSourceLine
TScriptError class 6679
FSubType
TFLDDesc class 1574
ftComp enumeration member 4747
ftCurr enumeration member 4747
ftDouble enumeration member 4747
ftExtended enumeration member 4747
FTransaction
TTransactionItem class 1695
FTransactionDesc
TTransactionItem class 1695
ftSingle enumeration member 4747
FullName
IWebBrowser2Disp interface 770
IWebBrowserApp interface 783
IWebBrowserAppDisp interface 791
TInternetExplorer class 450
TWebBrowser class 528
FullScreen
IWebBrowser2Disp interface 770
IWebBrowserApp interface 783
IWebBrowserAppDisp interface 791
TInternetExplorer class 450
TWebBrowser class 529
FullyQualifyInputName
TAdapterNamedField class 5356
FullyQualifyInputNames
IFullyQualifyInputNames interface 5885
FUnits1
TFLDDesc class 1574
FUnits2
TFLDDesc class 1574
FUseCOMWait
THandleObject class 3140
FUsesDayligtSavings
TTimeZone record 1728
fvCurrency enumeration member 4325
fvExtended enumeration member 4325

G

GeneratePageElements
TBaseAdapterPageProducer class 1150
GeneratePrefix
IXMLDocument interface 7646
TXMLDocument class 7360

GenerateScreenTips
TCustomScreenTipsManager class 310
GenerateXMLSchema
IWebServExp interface 6736
TWebServExp class 6730
Get
IXMLNodeList interface 7710
TAbstractNamedVariants class 6126
TAssociateItemList class 304
TCustomComboBoxStrings class 2342
THTTPReqResp class 1283
TNamedVariantsList class 6137
TWideStringList class 6817
TWideStrings class 6837
TXMLNodeList class 7469
Get_AddressBar
IWebBrowser2 interface 753
TInternetExplorer class 430
Get_Application
IWebBrowser interface 735
TInternetExplorer class 430
TWebBrowser class 509
TWebBrowser_V1 class 572
Get_ASProvider
ISearchAssistantOC interface 650
Get_ASSetting
ISearchAssistantOC interface 650
get_async
IDOMParseOptions interface 7592
get_attributes
IDOMNode interface 7561
Get_Busy
IWebBrowser interface 735
TInternetExplorer class 430
get_childNodes
IDOMNode interface 7561
Get_Columns
IShellNameSpace interface 702
TShellNameSpace class 472
Get_Container
IWebBrowser interface 735
TInternetExplorer class 431
TWebBrowser class 509
TWebBrowser_V1 class 572
Get_Count
ISearches interface 687
IShellWindows interface 726
TShellWindows class 499
Get_CountViewTypes
IShellNameSpace interface 702
TShellNameSpace class 472
get_data
IDOMCharacterData interface 7528
IDOMProcessingInstruction interface 7601
Get_Default
ISearches interface 688
Get_Depth
IShellNameSpace interface 702
TShellNameSpace class 472
get_doctype
IDOMDocument interface 7535
Get_Document
IWebBrowser interface 736
TInternetExplorer class 431
TWebBrowser class 509
TWebBrowser_V1 class 572
get_documentElement
IDOMDocument interface 7535
get_domImplementation
IDOMDocument interface 7535
get_Encoding
IDOMXMLProlog interface 7605
get_entities
IDOMDocumentType interface 7540
Get_EnumOptions
IShellNameSpace interface 703
TShellNameSpace class 473
get_errorCode

IDOMParseError interface 7588
get_filepos
IDOMParseError interface 7588
get_firstChild
IDOMNode interface 7561
Get_Flags
IShellNameSpace interface 703
TShellNameSpace class 473
Get_FullName
IWebBrowserApp interface 778
TInternetExplorer class 431
Get_FullScreen
IWebBrowserApp interface 779
TInternetExplorer class 431
Get_Height
IWebBrowser interface 736
TInternetExplorer class 431
get_high
DynamicArray template 3267
Get_HWnd
IWebBrowserApp interface 779
TInternetExplorer class 432
Get_Id
ISearch interface 645
Get_input
IDOMXSLProcessor interface 7608
get_internalSubset
IDOMDocumentType interface 7540
Get_InWebFolder
ISearchAssistantOC interface 650
get_item
IDOMNamedNodeMap interface 7555
IDOMNodeList interface 7581
get_lastChild
IDOMNode interface 7561
Get_Left
IWebBrowser interface 736
TInternetExplorer class 432
get_length
DynamicArray template 3267
IDOMCharacterData interface 7528
IDOMNamedNodeMap interface 7555
IDOMNodeList interface 7582
get_line
IDOMParseError interface 7588
get_linepos
IDOMParseError interface 7588
get_localName
IDOMNode interface 7562
Get_LocationName
IWebBrowser interface 737
TInternetExplorer class 432
Get_LocationURL
IWebBrowser interface 737
TInternetExplorer class 432
get_low
DynamicArray template 3267
Get_MenuBar
IWebBrowserApp interface 779
TInternetExplorer class 433
Get_Mode
IShellNameSpace interface 703
TShellNameSpace class 473
get_name
IDOMAttr interface 7523
IDOMDocumentType interface 7540
Get_Name
IWebBrowserApp interface 779
TInternetExplorer class 433
get_namespaceURI
IDOMNode interface 7562
get_nextSibling
IDOMNode interface 7562
get_nodeName
IDOMNode interface 7562
get_nodeType
IDOMNode interface 7563
get_nodeValue

IDOMNode interface 7563
get_notationName
IDOMEntity interface 7550
get_notations
IDOMDocumentType interface 7540
Get_Offline
IWebBrowser2 interface 753
TInternetExplorer class 433
Get_output
IDOMXSLProcessor interface 7608
get_ownerDocument
IDOMNode interface 7563
get_ownerElement
IDOMAttr interface 7523
Get_Parent
IWebBrowser interface 737
TInternetExplorer class 433
TWebBrowser class 509
TWebBrowser_V1 class 572
get_parentNode
IDOMNode interface 7563
Get_Path
IWebBrowserApp interface 779
TInternetExplorer class 433
get_prefix
IDOMNode interface 7564
get_preserveWhiteSpace
IDOMParseOptions interface 7592
get_previousSibling
IDOMNode interface 7564
get_publicId
IDOMDocumentType interface 7540
IDOMEntity interface 7550
IDOMNotation interface 7585
Get_ReadyState
IWebBrowser2 interface 754
TInternetExplorer class 434
get_reason
IDOMParseError interface 7588
Get_RegisterAsBrowser
IWebBrowser2 interface 754
TInternetExplorer class 434
Get_RegisterAsDropTarget
IWebBrowser2 interface 754
TInternetExplorer class 434
Get_Resizable
IWebBrowser2 interface 754
TInternetExplorer class 434
get_resolveExternals
IDOMParseOptions interface 7592
Get_Root
IShellNameSpace interface 703
TShellNameSpace class 473
Get_SearchAssistantDefault
ISearchAssistantOC interface 650
Get_SearchCompanionAvailable
ISearchAssistantOC3 interface 668
Get_Searches
ISearchAssistantOC interface 651
Get_SelectedItem
IShellNameSpace interface 704
TShellNameSpace class 474
Get_ShellFeaturesEnabled
ISearchAssistantOC interface 651
Get_ShowFindPrinter
ISearchAssistantOC2 interface 658
Get_Silent
IWebBrowser2 interface 755
TInternetExplorer class 435
get_specified
IDOMAttr interface 7523
get_srcText
IDOMParseError interface 7589
get_Standalone
IDOMXMLProlog interface 7605
Get_StatusBar
IWebBrowserApp interface 780
TInternetExplorer class 435

Get_StatusText
IWebBrowserApp interface 780
TInternetExplorer class 435
Get_stylesheet
IDOMXSLProcessor interface 7609
Get_SubscriptionsEnabled
IShellFavoritesNameSpace interface 693
TShellNameSpace class 474
get_systemId
IDOMDocumentType interface 7541
IDOMEntity interface 7551
IDOMNotation interface 7586
get_tagName
IDOMElement interface 7545
get_target
IDOMProcessingInstruction interface 7601
get_text
IDOMNodeEx interface 7579
Get_TheaterMode
IWebBrowser2 interface 755
TInternetExplorer class 435
Get_Title
ISearch interface 645
Get_ToolBar
IWebBrowserApp interface 780
TInternetExplorer class 435
Get_Top
IWebBrowser interface 738
TInternetExplorer class 436
Get_TopLevelContainer
IWebBrowser interface 738
TInternetExplorer class 436
Get_TVFlags
IShellNameSpace interface 704
TShellNameSpace class 474
Get_type_
IWebBrowser interface 738
TInternetExplorer class 436
get_url
IDOMParseError interface 7589
Get_URL
ISearch interface 645
Get_UseSearchCompanion
ISearchAssistantOC3 interface 668
get_validate
IDOMParseOptions interface 7593
get_value
IDOMAttr interface 7524
get_Version
IDOMXMLProlog interface 7605
Get_Version
ISearchAssistantOC interface 651
Get_Visible
IWebBrowserApp interface 780
TInternetExplorer class 436
Get_Width
IWebBrowser interface 739
TInternetExplorer class 437
get_xml
IDOMNodeEx interface 7579
IDOMPersist interface 7596
GetActionLinkClass
TButtonControl class 2111
TCustomButton class 2264
GetActionName
IWebGetActionName interface 1087
TBaseAdapterAction class 5455
TCustomAdapterActionButton class 6433
GetActionsClass
TCustomAdapter class 5545
TDefaultFieldsAdapter class 5744
GetActionsList
IWebActionsList interface 1077
TAdapterActionsListHelper class 6327
TCustomAdapter class 5546
GetActive
IXMLDocument interface 7646
TXMLDocument class 7360

GetAdapter
IGetAdapterOfDisplay interface 6513
TAdapterControlGroup class 6346
TAdapterNamedField class 5356
TBaseAdapterAction class 5456
TBaseValuesListField class 5530
TCustomAdapterCommandGroup class 6457
TCustomPagedAdapterAction class 5727
GetAdapterActions
IGetAdapterActions interface 1018
TCustomAdapter class 5546
GetAdapterDispatcher
IGetWebAppComponents interface 1038
TCustomWebAppComponents class 6205
GetAdapterDisplayCharacteristics
IGetAdapterDisplayCharacteristics interface 1019
TAdapterDisplayReferenceGroup class 6384
TCustomAdapterCommandColumn class 6446
TCustomAdapterFieldGroup class 6486
TCustomAdapterGrid class 6500
GetAdapterErrors
IGetAdapterErrors interface 1020
TCustomAdapter class 5546
GetAdapterErrorsList
IGetAdapterErrorsList interface 5886
TCustomAdapter class 5547
GetAdapterEventImage
IActionImageProducer interface 980
GetAdapterFields
IGetAdapterFields interface 1021
TCustomAdapter class 5547
GetAdapterFormLayout
IGetAdapterFormLayout interface 6511
TAdapterFormLayout class 6412
TCustomAdapterCommandGroup class 6457
TCustomAdapterFieldGroup class 6486
TCustomAdapterForm class 6492
TCustomAdapterGrid class 6500
GetAdapterHiddenFields
IGetAdapterHiddenFields interface 1022
TCustomAdapter class 5547
GetAdapterHiddenRecordFields
IGetAdapterHiddenFields interface 1022
TCustomAdapter class 5548
GetAdapterImage
IGetAdapterImage interface 1023
IImageProducer interface 1042
TCustomAdapterImageField class 5629
GetAdapterItemRequestParams
IAdapterDispatcher interface 987
IGetAdapterItemRequestParams interface 1024
TBaseAdapterAction class 5456
TCustomAdapterDispatcher class 6174
TCustomAdapterImageField class 5629
GetAdapterMode
IAdapterMode interface 1010
GetAdapterModeNames
IGetAdapterModeNames interface 5887
GetAdapterModeOfAdapter
IGetAdapterModeOfDisplay interface 6512
TAdapterControlGroup class 6347
TAdapterDataDisplay class 6358
TCustomAdapterActionButton class 6433
TCustomAdapterCommandGroup class 6457
GetAdapterModeOfDisplay
IGetAdapterModeOfDisplay interface 6512
TAdapterControlGroup class 6347
TCustomAdapterCommandGroup class 6457
GetAdapterRequest
TAbstractWebContext class 5967
TWebContext class 5983
GetAdapterResponse
TAbstractWebContext class 5967
TWebContext class 5983
GetAdapterValuesList
IGetAdapterValuesList interface 1027
TAdapterNextPageField class 5384
TAdapterUpdateField class 5426

GetAdapterVariableName
IGetAdapterVariableName interface 6514
TAdapterControlGroup class 6348
TAdapterDataDisplay class 6358
TCustomAdapterActionButton class 6433
GetAddAllFieldsItem
IAddFieldsEditor interface 6080
TAdapterActionsListHelper class 6328
TAdapterDefaultActions class 5235
TAdapterDefaultFields class 5241
TCustomAdapterFieldGroup class 6486
TCustomAdapterGrid class 6500
GetAddFieldsDlgCaption
IAddFieldsEditor interface 6080
TAdapterActionsListHelper class 6328
TAdapterDefaultActions class 5235
TAdapterDefaultFields class 5241
TCustomAdapterFieldGroup class 6487
TCustomAdapterGrid class 6501
GetAddFieldsItem
IAddFieldsEditor interface 6080
TAdapterActionsListHelper class 6329
TAdapterDefaultActions class 5235
TAdapterDefaultFields class 5241
TCustomAdapterFieldGroup class 6487
TCustomAdapterGrid class 6501
GetAddRemoveItemClass
TStandardStyleActionBars class 3017
TThemedCustomizePopup class 4562
TXPStyleActionBars class 7789
TXPStyleCustomizePopup class 7770
getAlwaysShowLockState
IScriptErrorList interface 637
IScriptErrorListDisp interface 641
GetAppDispatcher
TCustomXSLPageProducer class 7887
GetApplicationAdapter
IWebApplicationInfo interface 1079
TCustomApplicationAdapter class 5669
GetApplicationTitle
IWebApplicationInfo interface 1080
TCustomApplicationAdapter class 5669
GetAppServer
TXMLBroker class 7310
GetAsDate
TXSDate class 7824
GetAsDateTime
TXSCustomDateTime class 7807
GetAsObject
IObjectGetter interface 1334
TFTCompTI class 1316
GetAsTime
TXSTime class 7851
GetAsUTCDateTime
TXSCustomDateTime class 7807
GetAsyncLoadState
IXMLDocument interface 7647
TXMLDocument class 7361
GetAttr
TWebUserItems class 6794
GetAttrCount
TWebUserItems class 6794
getAttribute
IDOMElement interface 7545
GetAttribute
IXMLNode interface 7675
TXMLNode class 7415
getAttributeNode
IDOMElement interface 7545
getAttributeNodeNS
IDOMElement interface 7545
GetAttributeNodes
IXMLNode interface 7675
TXMLNode class 7415
getAttributeNS
IDOMElement interface 7545
GetAttributeNS
IXMLNode interface 7676

TXMLNode class 7416
GetBaseException
Exception class 3962
GetBaseVariant
Variant class 3432
GetBinding
IBindings interface 7093
IPort interface 7144
TBindings class 6929
TPort class 7024
GetBindingForServicePort
TWSDLItems class 7220
GetBindingOfType
TWSDLItems class 7221
GetBindingOperation
IBindingOperations interface 7089
TBindingOperations class 6923
GetBindingOperationNode
TWSDLItems class 7221
GetBindingOperations
IBinding interface 7080
TBinding class 6908
GetBindings
IDefinition interface 7096
TDefinition class 6936
TWSDLItems class 7222
GetBindingType
TWSDLItems class 7222
GetBlobFieldData
TCustomSQLDataSet class 1530
GetBufferEncoding
TEncoding class 3976
GetByteCount
TEncoding class 3976
TMBCSEncoding class 3994
TUnicodeEncoding class 4026
TUTF7Encoding class 4031
GetBytes
TBigEndianUnicodeEncoding class 3968
TEncoding class 3977
TMBCSEncoding class 3995
TUnicodeEncoding class 4027
TUTF7Encoding class 4031
GetBytesReserved
IDataBlock interface 268
TDataBlock class 201
GetCacheMode
TBaseWebModuleFactory class 6282
TBaseWebPageModuleFactory class 6292
GetCanModify
TCustomSQLDataSet class 1531
GetCanView
IPageDispatcher interface 1062
TCustomPageDispatcher class 6191
GetCapacity
TAbstractNamedVariants class 6127
TNamedVariantsList class 6137
TWideStringList class 6817
TWideStrings class 6837
GetCaption
TAdapterDataDisplay class 6358
TCustomAdapterActionButton class 6434
GetCaretPos
TCustomMemo class 2448
GetCell
TValueListEditor class 4794
GetCharCount
TEncoding class 3978
TMBCSEncoding class 3995
TUnicodeEncoding class 4027
TUTF7Encoding class 4032
GetCharPos
TScriptError class 6679
GetChars
TBigEndianUnicodeEncoding class 3969
TEncoding class 3979
TMBCSEncoding class 3996
TUnicodeEncoding class 4028

TUTF7Encoding class 4032
GetChecked
TButtonControl class 2112
TCustomCheckBox class 2279
TRadioButton class 2817
GetChildNodeClass
TXMLDocument class 7361
GetChildNodeClasses
IXMLNodeAccess interface 7484
TXMLNode class 7416
GetChildNodes
IXMLDocument interface 7647
IXMLNode interface 7676
TXMLDocument class 7361
TXMLNode class 7417
GetChildOwner
TTabbedNotebook class 4409
GetChildren
TCustomAdapter class 5548
TTabbedNotebook class 4409
TTabSet class 4475
TWebContainedContainerComponent class 6055
TWebContainerComponent class 6065
TWebContainerComponentHelper class 6071
GetChildTableIndent
TAdapterFormLayout class 6413
GetChildValue
IXMLNode interface 7676
TXMLNode class 7417
GetClientSocket
TServerWinSocket class 127
GetClientThread
TServerWinSocket class 127
GetClientType
TClientSocket class 27
GetColCount
TValueListEditor class 4794
GetCollection
IXMLNode interface 7677
TXMLNode class 7418
GetColorMapClass
TStandardStyleActionBars class 3017
TXPStyleActionBars class 7789
GetCommandNames
TCustomSQLDataSet class 1531
GetCommandTypes
TSQLConnection class 1591
GetComponent
IWebComponentContainer interface 6099
TScriptComponent class 975
TWebComponentList class 6030
GetComponentClass
TBaseWebModuleFactory class 6282
TBaseWebPageModuleFactory class 6292
GetComponentCount
IWebComponentContainer interface 6099
GetComponentList
IGetWebComponentList interface 6082
TWebContainerComponentHelper class 6071
GetConnected
ITransport interface 279
TLocalConnection class 4550
TSoapConnection class 1192
TSocketTransport class 232
TSQLConnection class 1591
TXMLBroker class 7311
GetContainer
IWebComponent interface 6094
TWebContainedComponentHelper class 6048
GetContainerList
TWebContainedComponent class 6041
TWebContainedContainerComponent class 6056
GetContainerName
IWebVariablesContainerName interface 6105
GetContent
TScriptProducer class 6710
GetContentOptions
TBaseAdapterPageProducer class 1150

GetControl
TEditAction class 1793
GetControlClass
TStandardStyleActionBars class 3017
TXPStyleActionBars class 7789
GetControlRequirements
IGetControlRequirements interface 6516
TAdapterControlGroup class 6348
TAdapterDataDisplay class 6359
TCustomAdapterActionButton class 6434
GetControlsAlignment
TCustomCheckBox class 2279
TCustomEdit class 2355
TRadioButton class 2817
GetCount
IWebFormComponentContainer interface 1085
IXMLNodeCollection interface 7701
IXMLNodeList interface 7710
TAbstractNamedVariants class 6127
TAbstractScriptObjectFactories class 6654
TCustomCombo class 2297
TCustomComboBoxStrings class 2342
TCustomListBox class 2409
TNamedVariantsList class 6138
TNotifyList class 971
TScriptObjectFactories class 6705
TWebComponentList class 6030
TWebNotifyContainer class 6602
TWideStringList class 6817
TWideStrings class 6838
TXMLNodeCollection class 7454
TXMLNodeList class 7470
GetCreateMode
TBaseWebModuleFactory class 6283
TBaseWebPageModuleFactory class 6293
GetCurrent
IEnumerator interface 3499
TWideStringsEnumerator class 6857
GetCurrentIntf
IIterateIntfSupport interface 1051
TAdapterHiddenFieldsList class 5306
TBaseAdapterErrorsList class 5483
GetCurrentObject
IIterateObjectSupport interface 1052
TAdapterSubComponent class 5411
TSiteModuleHelper class 6569
GetCurrentValue
IWebGetFieldValues interface 1094
TBaseAdapterMultiValueField class 5494
GetData
TXMLTransform class 7729
GetDataAsXml
TXMLTransformClient class 7742
GetDataClass
TCustomAdapter class 5548
TDefaultFieldsAdapter class 5745
GetDataSet
TSQLConnection class 1591
GetDataSource
TCustomSQLDataSet class 1531
GetDay
TXSCustomDateTime class 7807
TXSDate class 7824
GetDecimalSecond
TXSDuration class 7838
GetDefaultAction
IGetDefaultAction interface 1028
TSitePageModuleHelper class 6578
GetDefaultActionName
TAdapterNextPageAction class 5374
TAdapterPrevPageAction class 5401
TBaseAdapterAction class 5456
TCustomAdapterGotoPageAction class 5619
TEndUserAdapterLoginFormAction class 5780
TEndUserAdapterLogoutAction class 5789
TLoginFormAdapterLoginAction class 5840
GetDefaultColorMapClass
TThemedCustomizePopup class 4563

TXPStyleCustomizePopup class 7770
GetDefaultComponentList
IGetWebComponentList interface 6082
TWebContainerComponentHelper class 6072
GetDefaultDisplayLabel
TAdapterNamedField class 5357
TAdapterNextPageAction class 5375
TAdapterPrevPageAction class 5401
TBaseAdapterAction class 5457
GetDefaultFieldName
TAdapterApplicationTitleField class 5222
TAdapterEndUserDisplayNameField class 5247
TAdapterEndUserLoggedInField class 5255
TAdapterNamedField class 5357
TFieldValuesValueField class 5816
TValuesListNameField class 5869
TValuesListValueField class 5873
GetDefaultHelpFile
IWinHelpTester interface 6895
GetDefaultPageName
IPageDispatcher interface 1062
TCustomPageDispatcher class 6191
GetDefaultSchemaName
TSQLConnection class 1592
GetDefaultTemplate
TBaseAdapterPageProducer class 1150
GetDefaultWebComponents
TAdapterDefaultActions class 5235
TAdapterDefaultFields class 5241
TAdapterSubComponent class 5411
TWebContainedContainerComponent class 6056
TWebContainerComponent class 6065
TWebContainerComponentHelper class 6072
GetDefinition
IWSDLDocument interface 7166
TWSDLDocument class 7069
GetDelta
TXMLBroker class 7311
GetDescription
TScriptError class 6680
GetDesigntimeWarnings
IGetDesigntimeWarnings interface 1029
TAdapterControlGroup class 6348
TAdapterDataDisplay class 6359
TAdapterDisplayReferenceGroup class 6384
TCustomAdapterActionButton class 6434
TCustomAdapterErrorList class 6481
GetDetailDataSet
TSQLDataLink class 1631
GetDetailLinkFields
TCustomSQLDataSet class 1531
getDetailsPaneOpen
IScriptErrorList interface 637
IScriptErrorListDisp interface 641
GetDialog
TFileAction class 1809
GetDialogClass
TBrowseForFolder class 1766
TColorSelect class 1776
TCommonDialogAction class 1788
TFileOpen class 1820
TFilePageSetup class 1837
TFilePrintSetup class 1847
TFileSaveAs class 1857
TFontEdit class 1866
TPrintDlg class 1886
TSearchFind class 1901
TSearchReplace class 1924
GetDisableDebugging
IInternetEnvOptions interface 1043
GetDispatchedPageName
TAbstractWebContext class 5968
TWebContext class 5984
GetDispatchParams
TCustomAdapterDispatcher class 6174
GetDisplayLabel
IWebGetDisplayLabel interface 1088
TAdapterNamedField class 5357

TBaseAdapterAction class 5457
GetDisplayName
IWebEndUser interface 5989
TBaseEndUserAdapter class 5506
TDependency class 3067
TScreenTipItem class 330
GetDisplayNameValue
TBaseEndUserAdapter class 5507
TCustomEndUserAdapter class 5683
TCustomEndUserSessionAdapter class 5699
GetDisplayStyle
IActionImageProducer interface 980
IGetHTMLStyle interface 1032
TAdapterNamedDisplayField class 5346
TBaseAdapterAction class 5457
GetDisplayStyleType
TAdapterNamedDisplayField class 5347
TBaseAdapterAction class 5458
TCustomAdapterBooleanField class 5594
TCustomAdapterFileField class 5609
TCustomAdapterGotoPageAction class 5619
TCustomAdapterImageField class 5630
TCustomAdapterMemoField class 5642
TCustomAdapterMultiValueField class 5651
TCustomAdapterPageAction class 5661
GetDisplayWidth
IWebGetDisplayWidth interface 1089
TAdapterNamedField class 5358
GetDocBinding
IXMLDocument interface 7647
TXMLDocument class 7362
GetDocumentation
IDocumentationCollection interface 7103
TDocumentationCollection class 6945
GetDocumentationCollection
IDocumented interface 7106
TDocumented class 6951
GetDocumentElement
IXMLDocument interface 7648
TXMLDocument class 7362
GetDocumentNode
IXMLDocument interface 7648
TXMLDocument class 7362
GetDocumentObject
IXMLDocumentAccess interface 7479
TXMLDocument class 7363
GetDomConverter
IAdapterConverter interface 1360
TSoapPascalInvoker class 1354
GetDOMDocument
IXMLDocument interface 7648
TXMLDocument class 7363
GetDOMNode
IXMLNode interface 7677
TXMLNode class 7418
GetDOMPersist
IXMLDocumentAccess interface 7479
TXMLDocument class 7363
GetDriverFunc
TSQLConnection class 1613
GetDropDownButtonWidth
TThemedDropDownButton class 4565
GetDroppedDown
TCustomCombo class 2297
GetEchoActionFieldValue
IEchoActionFieldValue interface 5881
TAdapterNamedField class 5358
GetEchoActionFieldValues
IEchoAdapterFieldValues interface 5883
TCustomAdapter class 5549
GetEditLimit
TValueListEditor class 4794
GetEditMask
TValueListEditor class 4794
GetEditStyle
TValueListEditor class 4795
GetEditText
TValueListEditor class 4795

GetElement
IPart interface 7136
TPart class 7010
Variant class 3432
getElementById
IDOMDocument interface 7535
GetElementDetails
TThemeServices class 4588
getElementsByTagName
IDOMDocument interface 7536
IDOMElement interface 7546
getElementsByTagNameNS
IDOMDocument interface 7536
IDOMElement interface 7546
GetEncoding
IXMLDocument interface 7649
TEncoding class 3980
TXMLDocument class 7363
GetEncodingStyle
IHeaderInfo interface 7248
GetEndUser
TAbstractWebContext class 5968
TWebContext class 5984
GetEndUserAdapter
IWebEndUser interface 5990
TBaseEndUserAdapter class 5507
GetEnumerator
IEnumerable interface 3498
TWideStrings class 6838
GetEOF
TDefaultFieldsPagedAdapter class 5755
GetError
IAdapterErrors interface 996
TBaseAdapterErrorsList class 5484
TScriptErrors class 6683
Variant class 3433
getErrorChar
IScriptErrorList interface 637
IScriptErrorListDisp interface 642
getErrorCode
IScriptErrorList interface 637
IScriptErrorListDisp interface 642
GetErrorCount
IAdapterErrors interface 997
TBaseAdapterErrorsList class 5484
TScriptErrors class 6683
TXMLBroker class 7311
getErrorLine
IScriptErrorList interface 638
IScriptErrorListDisp interface 642
getErrorMsg
IScriptErrorList interface 638
IScriptErrorListDisp interface 642
GetErrorResponse
TXMLBroker class 7312
GetErrors
TBaseAdapterAction class 5458
TScriptProducer class 6711
TXMLBroker class 7312
GetErrorText
IAdapterError interface 993
TAdapterError class 5258
getErrorUrl
IScriptErrorList interface 638
IScriptErrorListDisp interface 642
GetExceptionHandler
TMultiModuleWebAppServices class 6228
GetExceptionStackInfoProc
Exception class 3961
GetExecuteActionResponse
TBaseAdapterAction class 5458
GetExpandBtnClass
TStandardMenuPopup class 1757
TThemedPopupMenu class 4574
TXPStylePopupMenu class 7780
GetExpirationTime
IWebSessionAttributes interface 1104
TSessions class 388

TWebSessionImpl class 6766
GetExpired
IWebSessionAttributes interface 1105
TSessions class 388
TWebSessionImpl class 6767
GetExtCount
TAbstractInetFileType class 966
GetFactory
TAbstractScriptObjectFactories class 6654
TScriptObjectFactories class 6705
TWebModuleList class 6632
GetFactoryCount
TWebModuleList class 6632
GetFailureOptions
TBaseAdapterAction class 5459
GetFailurePage
IAdapterItemRequestParams interface 1006
IGetAdapterRequestDefaultResponse interface 1025
TAdapterDispatchParams class 6157
TAdapterItemRequestParams class 6163
GetFault
IFaults interface 7109
TFaults class 6958
GetFaults
IBindingOperation interface 7085
IOperation interface 7125
TBindingOperation class 6915
TOperation class 6990
GetField
IAdapterError interface 993
IAdapterHiddenFields interface 1004
IFieldValuesAdapter interface 1016
TAbstractFieldValuesAdapter class 5186
TAdapterError class 5258
TAdapterHiddenFieldsList class 5306
GetFieldCount
IAdapterHiddenFields interface 1004
TAdapterHiddenFieldsList class 5306
GetFieldData
TCustomSQLDataSet class 1532
GetFieldIndent
TAdapterFormLayout class 6413
GetFieldName
IWebGetFieldName interface 1090
TAdapterDataDisplay class 6359
GetFieldNames
TSQLConnection class 1592
GetFieldsList
IWebDataFields interface 1082
TAdapterControlGroup class 6349
TCustomAdapter class 5549
GetFieldValuesAdapter
IGetFieldValuesAdapter interface 1031
TBaseAdapterMultiValueField class 5494
GetFileExt
TAbstractInetFileType class 966
GetFileName
IXMLDocument interface 7649
TScriptError class 6680
TXMLDocument class 7364
GetFileNames
TCustomAdapterFileField class 5610
GetFileStream
TCustomLocateFileService class 6184
GetFirstRecord
TDefaultFieldsPagedAdapter class 5755
GetForm
TWindowAction class 1934
GetFormRequirements
IGetFormRequirements interface 6517
TCustomAdapterForm class 6492
GetFractionalSeconds
TXSCustomDateTime class 7808
TXSTime class 7851
GetFractionalSecondString
TXSTime class 7852
GetFullParams
TConnectionData class 1518

GetGetSocketEvent
TCustomServerSocket class 43
GetGetThreadEvent
TCustomServerSocket class 43
GetHandler
IAdapterDispatchParams interface 989
TAdapterDispatchParams class 6157
GetHasAppServer
TXMLBroker class 7312
GetHasChildNodes
IXMLNode interface 7678
TXMLNode class 7418
GetHashCode
TObject class 3375
GetHasViewAccess
IWebPageInfo interface 5997
TCurrentWebPageInfo class 6660
GetHelpPath
IWinHelpTester interface 6896
GetHelpStrings
IWinHelpTester interface 6896
GetHiddenFieldOptions
TCustomAdapter class 5549
GetHostedNodes
TXMLNode class 7419
GetHostNode
IXMLNodeAccess interface 7484
TXMLNode class 7419
GetHostScriptBaseURL
TWSDLHTMLPublish class 7285
GetHour
TXSCustomDateTime class 7808
TXSTime class 7852
GetHourOffset
TXSCustomDateTime class 7808
TXSTime class 7852
GetHTMLBlock
TScriptProducer class 6711
GetHTMLBlockCount
TScriptProducer class 6711
GetHTMLExt
IInternetEnvOptions interface 1044
GetHTMLFormName
TCustomAdapterForm class 6492
GetHTMLFormVarName
TCustomAdapterActionButton class 6434
TCustomAdapterForm class 6493
GetHTMLItem
THTMLItems class 6667
GetHttpBindingAttribute
TWSDLItems class 7223
GetHTTPReqResp
IHTTPReqResp interface 1297
THTTPReqResp class 1284
GetID
IAdapterError interface 993
IWebSessionID interface 1109
IWebSessionIDs interface 1110
TAdapterError class 5259
TWebSessionIDsImpl class 6762
TWebSessionImpl class 6767
GetIDCount
IWebSessionIDs interface 1111
TSessions class 388
TWebSessionIDsImpl class 6763
GetIDs
TSessions class 388
GetIDsOfNames
IDispatch interface 3495
TDataDispatch class 217
GetImageIndex
TTabSet class 4476
GetImageSize
TThemedAddRemoveItem class 4559
TThemedExpandBtn class 4567
TThemedMenuItem class 4573
GetImport
IImports interface 7116

TImports class 6971
GetImports
IDefinition interface 7096
TDefinition class 6936
TWSDLItems class 7223
GetIndex
IWebComponent interface 6094
TWebContainedComponentHelper class 6048
GetIndexForPage
TTabbedNotebook class 4410
GetIndexNames
TSQLConnection class 1593
TSQLTable class 1686
GetInput
IBindingOperation interface 7085
IOperation interface 7125
TBindingOperation class 6916
TOperation class 6990
GetInputName
IWebInputName interface 1099
TAdapterNamedField class 5358
GetInputStyle
IGetHTMLStyle interface 1033
TAdapterNamedDisplayField class 5347
TBaseAdapterAction class 5459
GetInputStyleType
TAdapterNamedDisplayField class 5347
TAdapterNextPageField class 5384
TAdapterPasswordField class 5394
TAdapterUpdateField class 5426
TCustomAdapterBooleanField class 5595
TCustomAdapterFileField class 5610
TCustomAdapterImageField class 5630
TCustomAdapterMemoField class 5643
TCustomAdapterMultiValueField class 5651
GetInstance
IVarInstanceReference interface 5023
TPublishableVariantType class 4698
GetInterceptorList
TStreamedConnection class 241
GetInterface
TObject class 3375
GetInterfaceEntry
TObject class 3376
GetInterfaceTable
TObject class 3376
GetInterpreter
TStreamedConnection class 242
TWebConnection class 258
GetInvoker
IInvokerConverter interface 1361
TSoapPascalInvoker class 1354
GetIsActive
IWebSessionAttributes interface 1105
TSessions class 389
TWebSessionImpl class 6767
GetIsAdapterActionList
IIsAdapterActionList interface 1050
TBaseAdapterAction class 5460
GetIsAppModule
TBaseWebModuleFactory class 6283
TBaseWebPageModuleFactory class 6293
TWebAppDataModuleFactory class 6296
TWebAppPageModuleFactory class 6299
GetIsLoginRequired
TBaseWebPageInfo class 6287
GetIsPublished
TBaseWebPageInfo class 6287
GetIsTextElement
IXMLNode interface 7678
TXMLNode class 7419
GetItem
IWebFormComponentContainer interface 1085
TAllocatedActions class 300
TNotifyList class 971
TWebModuleFactoryList class 6625
TWebModuleList class 6632
GetItemAttr

TWebUserItems class 6795
GetItemCount
IWebSessionValues interface 1113
TBaseAdapterAction class 5460
TCustomAdapterGotoPageAction class 5620
TCustomCombo class 2298
TCustomComboBox class 2328
TSessions class 389
TWebModuleList class 6632
TWebSessionImpl class 6767
GetItemData
TCustomListBox class 2409
GetItemHt
TCustomCombo class 2298
TCustomComboBox class 2329
GetItemIndex
TCustomCombo class 2298
TCustomListBox class 2410
GetItems
IWebSessionValues interface 1113
TSessions class 389
TWebSessionImpl class 6767
GetItemsClass
TCustomCombo class 2299
TCustomComboBox class 2329
GetIteratorIndex
IIteratorIndex interface 1053
TBaseAdapterAction class 5460
TCustomAdapter class 5550
GetKeyFieldNames
TCustomSQLDataSet class 1533
GetLabelText
TCustomLabel class 2386
GetLayoutAttributes
TCustomAdapterActionButton class 6435
TCustomAdapterCommandColumn class 6446
TCustomAdapterDisplayField class 6470
TCustomAdapterFieldGroup class 6487
TCustomAdapterForm class 6493
GetLine
TScriptError class 6680
GetList
TXMLNodeCollection class 7454
GetListImage
IValuesListAdapter interface 1074
TBaseValuesListAdapter class 5522
GetListImageOfValue
IValuesListAdapter interface 1074
TBaseValuesListAdapter class 5523
GetListName
IValuesListAdapter interface 1075
TBaseValuesListAdapter class 5523
GetListNameOfValue
IValuesListAdapter interface 1075
TBaseValuesListAdapter class 5523
GetListValue
IValuesListAdapter interface 1076
TBaseValuesListAdapter class 5524
GetLocalName
IXMLNode interface 7678
TXMLNode class 7420
GetLocateFileService
IGetWebAppComponents interface 1038
TCustomWebAppComponents class 6205
GetLocation
IImport interface 7112
TImport class 6964
GetLocationForImport
TWSDLItems class 7224
GetLoggedIn
IWebEndUser interface 5990
TBaseEndUserAdapter class 5507
GetLoggedInValue
TBaseEndUserAdapter class 5508
TCustomEndUserAdapter class 5684
TCustomEndUserSessionAdapter class 5700
GetLoginFormAction
IWebEndUser interface 5990

TBaseEndUserAdapter class 5508
GetLoginRequired
IPageDispatcher interface 1062
IWebPageInfo interface 5998
TCurrentWebPageInfo class 6660
TCustomPageDispatcher class 6191
GetLoginUsername
TSQLConnection class 1594
GetLogoutAction
IWebEndUser interface 5991
TBaseEndUserAdapter class 5508
GetMaxByteCount
TEncoding class 3980
TMBCSEncoding class 3996
TUnicodeEncoding class 4028
TUTF7Encoding class 4033
TUTF8Encoding class 4035
GetMaxCharCount
TEncoding class 3980
TMBCSEncoding class 3996
TUnicodeEncoding class 4028
TUTF7Encoding class 4033
TUTF8Encoding class 4036
GetMaxLength
IWebGetMaxLength interface 1096
TAdapterNamedField class 5359
GetMemory
IDataBlock interface 268
TDataBlock class 201
GetMessage
IMessages interface 7122
IParam interface 7133
TMessages class 6984
TParam class 7004
GetMessageNode
TWSDLItems class 7224
GetMessages
IDefinition interface 7097
TDefinition class 6936
TWSDLItems class 7224
GetMetadata
TCustomSQLDataSet class 1558
TInternalSQLDataSet class 894
TSQLQuery class 1660
GetMillisecond
TXSCustomDateTime class 7808
TXSTime class 7852
GetMimeBoundary
IWebNode interface 6619
THTTPReqResp class 1284
TLinkedWebNode class 1324
TWSDLClientNode class 7269
GetMIMEBoundary
IMimeAttachmentHandler interface 1179
GetMIMEStream
IMimeAttachmentHandler interface 1179
GetMinute
TXSCustomDateTime class 7809
TXSTime class 7853
GetMinuteOffset
TXSCustomDateTime class 7809
TXSTime class 7853
GetModified
IXMLDocument interface 7649
TXMLDocument class 7364
GetModule
TBaseWebModuleFactory class 6283
TBaseWebPageModuleFactory class 6293
GetModuleName
TBaseWebModuleFactory class 6284
TBaseWebPageModuleFactory class 6294
GetMonth
TXSCustomDateTime class 7809
TXSDate class 7824
GetName
IAdapterHiddenField interface 1002
IBinding interface 7081
IBindingOperation interface 7085

IDefinition interface 7097
IHeaderInfo interface 7248
IMessage interface 7118
IOperation interface 7125
IParam interface 7133
IPart interface 7137
IPort interface 7144
IPortType interface 7150
IQualifiedName interface 7253
IService interface 7157
TAdapterHiddenField class 5301
TBinding class 6908
TBindingOperation class 6916
TDefinition class 6936
TMessage class 6977
TOperation class 6991
TParam class 7004
TPart class 7010
TPort class 7024
TPortType class 7037
TService class 7050
TWSDLItems class 7225
getNamedItem
IDOMNamedNodeMap interface 7555
GetNamedItemCount
TScriptGlobalObjects class 6690
GetNamedItemName
TScriptGlobalObjects class 6690
getNamedItemNS
IDOMNamedNodeMap interface 7555
GetNameFieldClass
TBaseValuesListAdapter class 5524
TCustomStringsValuesList class 5734
GetNamespace
IHeaderInfo interface 7249
IImport interface 7112
IQualifiedName interface 7254
TImport class 6964
GetNamespaceURI
IXMLNode interface 7679
TXMLNode class 7421
GetNeedAdapterMode
IAdapterMode interface 1010
GetNeedPrevValue
IWebGetFieldPrevValue interface 5895
GetNewFieldPrefix
IAddFieldsEditor interface 6081
TAdapterActionsListHelper class 6329
TAdapterDefaultActions class 5236
TAdapterDefaultFields class 5242
TCustomAdapterFieldGroup class 6487
TCustomAdapterGrid class 6501
GetNewPage
TAdapterNextPageAction class 5375
TAdapterPrevPageAction class 5402
TCustomAdapterPageAction class 5661
GetNextPage
TCustomLoginFormAdapter class 5708
GetNextRecord
TDefaultFieldsPagedAdapter class 5755
GetNode
IXMLNodeCollection interface 7701
IXMLNodeList interface 7710
TXMLNodeCollection class 7454
TXMLNodeList class 7470
GetNodeIndentStr
IXMLDocument interface 7650
TXMLDocument class 7364
GetNodeName
IXMLNode interface 7679
TXMLNode class 7421
GetNodeObject
IXMLNodeAccess interface 7485
TXMLNode class 7422
GetNodeType
IXMLNode interface 7679
TXMLNode class 7422
GetNodeValue

IXMLNode interface 7680
TXMLNode class 7423
GetNotify
TXMLBroker class 7312
GetNotifyCount
TXMLBroker class 7313
GetObject
TCustomComboBoxStrings class 2342
TWideStringList class 6818
TWideStrings class 6838
GetObjectErrors
IAdapterErrors interface 997
TBaseAdapterErrorsList class 5484
GetObjectIdentifier
IAdapterDispatchParams interface 989
TAdapterDispatchParams class 6157
GetObjectList
TAbstractScriptObjectFactories class 6654
TScriptObjectFactories class 6706
GetObjectTypeNames
TCustomSQLDataSet class 1533
GetOnClientError
TCustomServerSocket class 44
GetOnClientEvent
TCustomServerSocket class 44
GetOnGetAttachmentEvent
IMimeAttachmentHandler interface 1180
GetOnModuleAdded
TWebModuleList class 6633
GetOnSendAttachmentEvent
IMimeAttachmentHandler interface 1180
GetOnThreadEnd
TCustomServerSocket class 44
GetOnThreadStart
TCustomServerSocket class 45
GetOperation
IOperations interface 7130
TOperations class 6998
GetOperationNode
TWSDLItems class 7225
GetOperations
IPortType interface 7151
TPortType class 7037
TWSDLItems class 7226
GetOperationsForBinding
TWSDLItems class 7226
GetOptions
IXMLDocument interface 7650
TXMLDocument class 7365
GetOutput
IBindingOperation interface 7085
IOperation interface 7126
TBindingOperation class 6916
TOperation class 6991
GetOutputParams
TCustomSQLDataSet class 1534
GetOwner
TDependencies class 3064
TScreenTipsCollection class 336
TWebComponentList class 6030
TWebUserItems class 6795
GetOwnerDocument
IXMLNode interface 7680
TXMLNode class 7423
GetPackageNames
TSQLConnection class 1594
GetPageCount
TAbstractWebContext class 5968
TDefaultFieldsPagedAdapter class 5756
TWebContext class 5984
GetPageDescription
IWebPageInfo interface 5998
TBaseWebPageInfo class 6287
TCurrentWebPageInfo class 6660
GetPageDispatcher
IGetWebAppComponents interface 1038
TCustomWebAppComponents class 6205
GetPageFile

TBaseWebPageInfo class 6288
GetPageHREF
IWebPageInfo interface 5998
TBaseWebPageInfo class 6288
TCurrentWebPageInfo class 6661
GetPageInfo
TAbstractWebContext class 5968
TWebContext class 5984
GetPageIsPublished
IWebPageInfo interface 5999
TCurrentWebPageInfo class 6661
GetPageName
IWebPageInfo interface 5999
TBaseWebPageInfo class 6288
TCurrentWebPageInfo class 6661
GetPageProducer
TSitePageModuleHelper class 6578
TWebAppPageModuleHelper class 6597
TWebPageModuleHelper class 6611
GetPageTitle
IWebPageInfo interface 5999
TBaseWebPageInfo class 6289
TCurrentWebPageInfo class 6662
GetParams
IAdapterDispatchParams interface 989
TAdapterDispatchParams class 6158
TSoapDataModule class 1219
GetParamValues
IAdapterItemRequestParams interface 1007
TAdapterItemRequestParams class 6163
GetParentComponent
TAdapterSubComponent class 5412
TWebContainedComponent class 6041
TWebContainedContainerComponent class 6056
GetParentNode
IXMLNode interface 7680
TXMLNode class 7424
GetParseOptions
IXMLDocument interface 7650
TXMLDocument class 7365
GetPart
IHeaderInfo interface 7249
IParts interface 7141
TParts class 7018
GetPartNode
TWSDLItems class 7227
GetParts
IMessage interface 7119
TMessage class 6977
TWSDLItems class 7227
GetPartsForOperation
TWSDLItems class 7228
getPerErrorDisplay
IScriptErrorList interface 638
IScriptErrorListDisp interface 643
GetPickList
TValueListEditor class 4795
GetPopupClass
TStandardDropDownButton class 1748
TStandardStyleActionBars class 3018
TThemedDropDownButton class 4565
TXPStyleActionBars class 7790
TXPStyleDropDownBtn class 7772
GetPort
IPorts interface 7148
TPorts class 7031
GetPorts
IService interface 7157
TService class 7050
GetPortsForService
TWSDLItems class 7229
GetPortType
IPortTypes interface 7154
TPortTypes class 7044
GetPortTypeEntries
TWSDLPublish class 7292
GetPortTypeList
IWSDLPublish interface 7296

TWSDLPublish class 7292
GetPortTypeNode
TWSDLItems class 7229
GetPortTypes
IDefinition interface 7097
TDefinition class 6937
TWSDLItems class 7229
GetPreamble
TBigEndianUnicodeEncoding class 3969
TEncoding class 3981
TMBCSEncoding class 3997
TUnicodeEncoding class 4029
TUTF8Encoding class 4036
GetPrefix
IXMLNode interface 7680
TXMLNode class 7424
GetPrefixedName
TXMLNode class 7425
GetPrevValue
IWebGetFieldPrevValue interface 5895
GetProcedureNames
TSQLConnection class 1595
GetProcedureParams
TSQLConnection class 1597
GetProducerComponent
IGetProducerComponent interface 1035
TSitePageModuleHelper class 6578
GetProducerName
TXMLBroker class 7313
GetProducerTemplateFileName
TCustomXSLPageProducer class 7887
GetProducerTemplateStream
TCustomXSLPageProducer class 7887
GetPrologNode
TXMLDocument class 7365
GetPrologValue
TXMLDocument class 7366
GetProperty
ISearchAssistantOC interface 651
ISearchAssistantOC2Disp interface 662
ISearchAssistantOC3Disp interface 672
ISearchAssistantOCDisp interface 680
IVarInvokeable interface 5025
IWebBrowser2Disp interface 765
IWebBrowserApp interface 781
IWebBrowserAppDisp interface 787
TInternetExplorer class 437
TInvokeableVariantType class 5020
TPublishableVariantType class 4698
TWebBrowser class 510
GetProvider
TLocalConnection class 4551
TSoapDataModule class 1219
GetProviderNames
TLocalConnection class 4551
TSoapConnection class 1192
TSoapDataModule class 1220
GetQuoteChar
TCustomSQLDataSet class 1534
GetReadOnly
IXMLNode interface 7681
TXMLNode class 7425
GetRec
AnsiStringBase class 3193
UnicodeString class 3401
GetRecord
TCustomSQLDataSet class 1534
GetRecordCount
TCustomSQLDataSet class 1534
TDefaultFieldsPagedAdapter class 5756
GetRecordIndex
TDefaultFieldsPagedAdapter class 5756
GetRecordKeys
IGetRecordKeys interface 5888
GetRecords
TSoapDataModule class 1220
GetRedirect
TXMLBroker class 7313

GetRedirectParams
TSitePageModuleHelper class 6578
GetRequestedPage
TCustomPageDispatcher class 6192
GetRequestedPageName
TAbstractWebContext class 5969
TWebContext class 5985
GetRequestIdentifier
IAdapterDispatchParams interface 989
TAdapterDispatchParams class 6158
GetRequired
IAdapterFieldAttributes interface 1000
IHeaderInfo interface 7249
TAdapterNamedField class 5359
GetRequiredTags
TBaseAdapterPageProducer class 1151
GetResponse
TXMLBroker class 7313
GetResponseStream
TLinkedRIO class 1319
TLinkedWebNode class 1325
TLogLinkedWebNode class 1328
GetResultString
TXMLTransform class 7729
GetRowCount
TValueListEditor class 4795
GetSampleImageFile
IInternetEnvOptions interface 1044
GetSchemaDefs
ITypes interface 7163
TTypes class 7063
GetSchemaNames
TSQLConnection class 1598
GetSchemaNode
TWSDLItems class 7230
GetSchemaRef
IXMLDocument interface 7650
TXMLDocument class 7366
GetSchemas
TWSDLItems class 7230
GetScreenTip
TCustomScreenTipsManager class 310
GetScreenTipItem
IScreenTipItem interface 362
TCustomScreenTipsPopup class 319
GetScript
TScriptCodeItem class 6673
TScriptIncludeItem class 6697
TScriptItem class 6699
GetScriptEngine
TAbstractScriptObjectFactory class 6657
TScriptGlobalObjects class 6690
GetScriptEngines
TBaseAdapterPageProducer class 1151
GetScriptIncludeFile
TScriptIncludeFiles class 6694
GetScriptItem
TScriptItems class 6701
GetScriptObject
IGetScriptObject interface 1036
GetScriptObjectClassKey
IGetScriptObjectClassKey interface 1037
GetScriptTimeoutSecs
IInternetEnvOptions interface 1044
GetScrollBtnClass
TStandardStyleActionBars class 3018
TXPStyleActionBars class 7790
GetSearchAssistantURL
ISearchAssistantOC interface 651
ISearchAssistantOC2Disp interface 662
ISearchAssistantOC3Disp interface 672
ISearchAssistantOCDisp interface 681
GetSecond
TXSCustomDateTime class 7810
TXSTime class 7853
GetSelCount
TCustomListBox class 2410
GetSelected

TCustomListBox class 2410
GetSelLength
TCustomCombo class 2299
TCustomEdit class 2356
GetSelStart
TCustomCombo class 2299
TCustomEdit class 2356
GetSelText
TCustomComboBox class 2329
TCustomEdit class 2356
GetSelTextBuf
TCustomEdit class 2357
GetServer
TLocalConnection class 4551
TSoapConnection class 1193
GetServerList
TSoapConnection class 1193
TStreamedConnection class 242
GetServerMethodNames
TSQLConnection class 1598
GetServerThread
TServerWinSocket class 127
GetServerType
TCustomServerSocket class 45
GetService
IServices interface 7160
TServices class 7057
GetServiceAndPortOfBinding
TWSDLItems class 7231
GetServiceController
TService class 3076
TSocketService class 184
GetServiceNode
TWSDLItems class 7231
GetServices
IDefinition interface 7097
TDefinition class 6937
TWSDLItems class 7232
GetSession
TAbstractWebContext class 5969
TSessions class 389
TWebContext class 5985
GetSessionID
TInvalidWebSession class 6217
TWebSession class 6253
GetSessionIDGen
TSessions class 390
GetSessionIDs
IWebGetSessionIDs interface 1097
TCustomSessionsService class 6751
GetSessionsService
IWebSessionAccess interface 6258
GetSignature
IDataBlock interface 268
TDataBlock class 201
GetSiteModuleHelper
TCustomWebDataModule class 6553
TCustomWebPageModule class 6560
GetSitePageModuleHelper
TCustomWebPageModule class 6561
GetSize
IDataBlock interface 269
TDataBlock class 202
GetSoapAction
TWSDLItems class 7232
GetSoapActionList
TWSDLItems class 7233
GetSoapAddressForServicePort
TWSDLItems class 7234
GetSoapBindingAttribute
TWSDLItems class 7234
GetSoapBindingInputNode
TWSDLItems class 7235
GetSoapBindingIONode
TWSDLItems class 7235
GetSoapBindingOutputNode
TWSDLItems class 7236
GetSoapBodyAttribute

TWSDLItems class 7236
GetSoapBodyInputAttribute
TWSDLItems class 7237
GetSoapBodyNamespace
TWSDLItems class 7238
GetSoapBodyOutputAttribute
TWSDLItems class 7238
GetSoapHeaders
TWSDLItems class 7239
GetSoapInputHeaders
TWSDLItems class 7240
GetSoapOperationAttribute
TWSDLItems class 7240
GetSoapOperationStyle
TWSDLItems class 7241
GetSoapOutputHeaders
TWSDLItems class 7241
GetSOAPServer
TSoapConnection class 1194
GetSOAPServerIID
TSoapConnection class 1194
GetSocketAddr
TIpSocket class 1419
GetSortFieldNames
TCustomSQLDataSet class 1535
GetSourceLine
TScriptError class 6680
GetStackInfoStringProc
Exception class 3961
GetStackTrace
Exception class 3962
GetStandAlone
IXMLDocument interface 7651
TXMLDocument class 7366
GetStartTime
IWebSessionAttributes interface 1105
TSessions class 390
TWebSessionImpl class 6768
GetStream
IDataBlock interface 269
TDataBlock class 202
GetString
TEncoding class 3981
GetStyleName
TStandardStyleActionBars class 3018
TXPStyleActionBars class 7790
GetSuccessOptions
TBaseAdapterAction class 5461
GetSuccessPage
IAdapterItemRequestParams interface 1007
IGetAdapterRequestDefaultResponse interface 1025
TAdapterDispatchParams class 6158
TAdapterItemRequestParams class 6164
GetTableIndent
TAdapterFormLayout class 6413
GetTableNames
TSQLConnection class 1599
GetTargetNamespace
IDefinition interface 7097
TDefinition class 6937
TWSDLHTMLPublish class 7285
TWSDLItems class 7242
GetTargetNamespacePrefix
TWSDLItems class 7243
GetTemplateFileName
TCustomAdapterPageProducer class 1162
GetTemplateFileType
IProducerEditorViewSupport interface 6090
TBaseAdapterPageProducer class 1152
TCustomXSLPageProducer class 7887
TSitePageModuleHelper class 6579
GetTemplateStream
TCustomAdapterPageProducer class 1163
TCustomLocateFileService class 6184
TCustomXSLPageProducer class 7888
TSitePageModuleHelper class 6579
GetTerminated
IWebSessionAttributes interface 1105

TSessions class 390
TWebSessionImpl class 6768
GetText
IXMLNode interface 7681
TWideStrings class 6839
TXMLNode class 7426
GetTextStr
TWideStrings class 6839
GetThreadCacheSize
TCustomServerSocket class 45
GetThreadObject
TCustomIpClient class 1404
GetTimeoutMinutes
IWebSessionAttributes interface 1105
TInvalidWebSession class 6218
TSessions class 390
TWebSession class 6253
TWebSessionImpl class 6768
GetTimeZone
TTimeZone record 1728
GetTouchTime
IWebSessionAttributes interface 1106
TSessions class 391
TWebSessionImpl class 6768
GetType_
IBinding interface 7081
IPart interface 7137
TBinding class 6909
TPart class 7011
GetTypeID
TAbstractInetFileType class 967
GetTypeInfo
IDispatch interface 3496
TDataDispatch class 218
GetTypeInfoCount
IDispatch interface 3496
TDataDispatch class 218
GetTypes
IDefinition interface 7098
TDefinition class 6937
GetTypeSystemsList
IWSDLPublish interface 7296
TWSDLPublish class 7292
GetUpdateCount
IXMLNodeList interface 7711
TXMLNodeList class 7471
GetUse
IHeaderInfo interface 7249
GetUserID
TBaseEndUserAdapter class 5509
TCustomEndUserAdapter class 5684
TCustomEndUserSessionAdapter class 5700
GetUseZeroMilliseconds
TXSCustomDateTime class 7810
GetValue
IAdapterHiddenField interface 1002
IWebGetFieldValue interface 1092
IWebSessionValues interface 1113
TAdapterHiddenField class 5301
TAdapterNamedField class 5359
TInvalidWebSession class 6218
TWebSession class 6254
TWebSessionImpl class 6769
GetValueCount
TAdapterUpdateField class 5427
TBaseAdapterMultiValueField class 5494
TCustomAdapterFileField class 5610
TCustomAdapterMultiValueField class 5652
GetValueFieldClass
TAbstractFieldValuesAdapter class 5186
TBaseValuesListAdapter class 5524
TCustomStringsValuesList class 5734
TFieldValuesAdapter class 5809
GetValueIndex
IFieldValuesAdapter interface 1016
TAbstractFieldValuesAdapter class 5187
GetValues
TAdapterUpdateField class 5427

TBaseAdapterMultiValueField class 5495
TCustomAdapterFileField class 5611
TCustomAdapterMultiValueField class 5652
GetVariable
TAdapterSubComponent class 5412
TCustomAdapter class 5550
TSiteModuleHelper class 6569
GetVariableCount
TAdapterSubComponent class 5413
TCustomAdapter class 5550
TSiteModuleHelper class 6569
GetVariableName
TAdapterNamedField class 5360
TBaseAdapterAction class 5461
TCustomAdapter class 5551
GetVariablesContainer
IGetVariablesContainerOfDisplay interface 6518
TAdapterDataDisplay class 6360
GetVariablesInUse
TAdapterDefaultActions class 5236
TAdapterDefaultFields class 5242
TAdapterSubComponent class 5413
GetVendors
TDOMVendorList class 7518
GetVerb
IWebComponentCommands interface 6098
GetVerbCount
IWebComponentCommands interface 6098
GetVersion
IXMLDocument interface 7651
TXMLDocument class 7366
GetViewAccess
TBaseWebPageInfo class 6289
GetViewMode
IGetHTMLStyle interface 1033
TAdapterNamedDisplayField class 5348
TBaseAdapterAction class 5462
GetViewModeType
TAdapterNamedDisplayField class 5348
TBaseAdapterAction class 5462
GetVisible
IAdapterActionAttributes interface 984
IAdapterFieldAttributes interface 1000
TAdapterNamedField class 5360
TBaseAdapterAction class 5462
GetVisibleActions
IWebActionsList interface 1077
TAdapterActionsListHelper class 6329
TCustomAdapter class 5551
GetVisibleFields
IWebDataFields interface 1083
TAdapterControlGroup class 6349
TCustomAdapter class 5551
GetWaitEvent
ITransport interface 279
TSocketTransport class 233
TWebConnection class 259
GetWebApplicationInfo
IGetWebAppComponents interface 1038
TCustomWebAppComponents class 6206
GetWebAppServices
TCustomWebAppDataModule class 6542
TCustomWebAppPageModule class 6548
GetWebComponentNodeName
IGetWebComponentNodeName interface 6083
TAdapterSubComponent class 5413
GetWebComponents
TWebContainedContainerComponent class 6057
TWebContainerComponent class 6065
GetWebEndUser
IGetWebAppComponents interface 1039
TCustomWebAppComponents class 6206
GetWebModuleContext
TScriptProducer class 6711
GetWebModules
TAbstractWebContext class 5969
TWebContext class 5985
GetWebRequest

TAbstractWebContext class 5969
TWebContext class 5985
GetWebResponse
TAbstractWebContext class 5970
TWebContext class 5986
GetWebServExp
IWebServExpAccess interface 6737
TWebServExp class 6731
GetWebSession
IWebSessionAccess interface 6258
GetWebSnapObjClass
TAdapterNamedField class 5360
TBaseAdapterAction class 5463
TBaseValuesListAdapter class 5524
TCustomAdapter class 5552
TCustomApplicationAdapter class 5669
TCustomWebDataModule class 6553
GetWebUserList
IGetWebAppComponents interface 1039
TCustomWebAppComponents class 6206
GetWSDLDoc
IWSDLItems interface 7257
TWSDLItems class 7243
GetWSDLForInterface
TWebServExp class 6731
GetWSDLForPortType
IWSDLPublish interface 7296
TWSDLPublish class 7293
GetWSDLItems
IWSDLItems interface 7257
TWSDLItems class 7243
GetXML
IXMLDocument interface 7651
IXMLNode interface 7681
TXMLDocument class 7367
TXMLNode class 7426
GetXMLData
IProducerEditorViewSupport interface 6090
TBaseAdapterPageProducer class 1152
TCustomXSLPageProducer class 7888
TSitePageModuleHelper class 6579
GetXMLDocument
TCustomXSLPageProducer class 7888
GetXMLRecords
TXMLBroker class 7314
GetXMLSchemaDef
IXMLSchemaDefs interface 7170
TXMLSchemaDefs class 7077
GetXMLStream
IGetXMLStream interface 6084
TXMLBroker class 7314
GetXSDForTypeSystem
IWSDLPublish interface 7296
TWSDLPublish class 7293
GetXSLData
IProducerEditorViewSupport interface 6090
TBaseAdapterPageProducer class 1152
TCustomXSLPageProducer class 7888
TSitePageModuleHelper class 6579
GetYear
TXSCustomDateTime class 7810
TXSDate class 7825
GlowSize
TCustomLabel class 2395
TLabel class 2624
Glyph
TCustomScreenTipsPopup class 321
TScreenTipsPopup class 348
GoBack
IWebBrowser interface 739
IWebBrowser2Disp interface 765
IWebBrowserAppDisp interface 787
IWebBrowserDisp interface 796
TInternetExplorer class 437
TWebBrowser class 510
TWebBrowser_V1 class 573
GoForward
IWebBrowser interface 739

IWebBrowser2Disp interface 765
IWebBrowserAppDisp interface 787
IWebBrowserDisp interface 796
TInternetExplorer class 437
TWebBrowser class 510
TWebBrowser_V1 class 573
GoHome
IWebBrowser interface 740
IWebBrowser2Disp interface 766
IWebBrowserAppDisp interface 787
IWebBrowserDisp interface 797
TInternetExplorer class 437
TWebBrowser class 511
TWebBrowser_V1 class 573
GoSearch
IWebBrowser interface 740
IWebBrowser2Disp interface 766
IWebBrowserAppDisp interface 788
IWebBrowserDisp interface 797
TInternetExplorer class 438
TWebBrowser class 511
TWebBrowser_V1 class 573
GradientEndColor
TCustomScreenTipsManager class 314
TScreenTipsManager class 340
TScreenTipsWindow class 359
GradientStartColor
TCustomScreenTipsManager class 314
TScreenTipsManager class 340
TScreenTipsWindow class 359
GridLineWidth
TValueListEditor class 4819
GUIDDesc
TSocketForm class 163

H

Handle
TBaseSocket class 1387
TCustomWinSocket class 84
THandleObject class 3142
TStreamedConnection class 246
HandleException
TMultiModuleWebAppServices class 6228
TServerClientThread class 101
TWebRequestHandler class 6637
HandleParserError
TScriptProducer class 6712
Handler
IAdapterDispatchParams interface 990
TAdapterDispatchParams class 6159
HandleRequest
IAdapterRequestHandler interface 1013
TBaseAdapterAction class 5463
TBaseRequestHandler class 6167
TCustomAdapterImageField class 5630
TMultiModuleWebAppServices class 6229
TWebRequestHandler class 6637
TWSDLHTMLPublish class 7285
HandleScriptError
TScriptProducer class 6712
HandleScriptTag
TBaseAdapterPageProducer class 1153
HandlesTarget
TCommonDialogAction class 1788
TEditAction class 1793
TFileExit class 1811
THelpAction class 1875
TSearchAction class 1898
TSearchFindNext class 1914
TWindowAction class 1934
HandleWinInetError
THTTPReqResp class 1285
HasAppServer
TXMLBroker class 7323
hasAttribute
IDOMElement interface 7546
HasAttribute
IXMLNode interface 7682

TXMLNode class 7427
hasAttributeNS
IDOMElement interface 7546
HasChildNode
IXMLNodeAccess interface 7485
TXMLNode class 7427
hasChildNodes
IDOMNode interface 7564
HasChildNodes
IXMLNode interface 7691
HasExecuteAccess
IAdapterAccess interface 982
IAdapterActionAccess interface 983
TBaseAdapterAction class 5463
TCustomAdapter class 5552
HasExecuteRights
IWebEndUser interface 5991
TBaseEndUserAdapter class 5509
hasFeature
IDOMImplementation interface 7553
HasHTMLBrowserView
IProducerEditorViewSupport interface 6091
TBaseAdapterPageProducer class 1153
TCustomXSLPageProducer class 7888
TSitePageModuleHelper class 6579
HasModifyAccess
IAdapterAccess interface 982
IAdapterFieldAccess interface 999
TAdapterNamedField class 5361
TCustomAdapter class 5552
HasModifyRights
IWebEndUser interface 5992
TBaseEndUserAdapter class 5509
HasObjectErrors
IAdapterErrors interface 997
TBaseAdapterErrorsList class 5484
HasParent
TAdapterSubComponent class 5414
TWebContainedComponent class 6041
TWebContainedContainerComponent class 6057
HasPickList
TItemProp class 4781
HasScriptView
IProducerEditorViewSupport interface 6091
TBaseAdapterPageProducer class 1154
TCustomXSLPageProducer class 7889
TSitePageModuleHelper class 6580
HasTransaction
TSQLConnection class 1599
HasTransparentParts
TThemeServices class 4589
HasTypesNode
TWSDLItems class 7244
HasValue
IWebGetFieldValues interface 1094
TBaseAdapterMultiValueField class 5495
HasViewAccess
IAdapterAccess interface 982
IAdapterFieldAccess interface 999
IWebPageInfo interface 6000
TAdapterNamedField class 5361
TCustomAdapter class 5553
HasViewRights
IWebEndUser interface 5992
TBaseEndUserAdapter class 5510
HasXMLBrowserView
IProducerEditorViewSupport interface 6091
TBaseAdapterPageProducer class 1154
TCustomXSLPageProducer class 7889
TSitePageModuleHelper class 6580
HasXSLBrowserView
IProducerEditorViewSupport interface 6091
TBaseAdapterPageProducer class 1154
TCustomXSLPageProducer class 7889
TSitePageModuleHelper class 6580
hbsNormal enumeration member 365
hbsRounded enumeration member 365
Header

TScreenTipItem class 332
HeaderControl1
TSocketForm class 163
HeadingAttributes
TAdapterGrid class 6424
TCustomAdapterGrid class 6502
Height
IWebBrowser interface 746
IWebBrowser2Disp interface 770
IWebBrowserAppDisp interface 791
IWebBrowserDisp interface 799
TInternetExplorer class 451
TScreenTipItem class 333
TTabPage class 4449
HelpContext
Exception class 3964
TBrowseForFolder class 1768
TColorSelect class 1777
TFileExit class 1813
TFileOpen class 1822
TFileOpenWith class 1832
TFilePageSetup class 1839
TFilePrintSetup class 1849
TFileSaveAs class 1859
TFontEdit class 1867
TPrintDlg class 1888
TSearchFind class 1903
TSearchFindNext class 1916
TSearchReplace class 1926
HelpKeyword
TBrowseForFolder class 1769
TColorSelect class 1778
TFileExit class 1813
TFileOpen class 1823
TFileOpenWith class 1833
TFilePageSetup class 1840
TFilePrintSetup class 1849
TFontEdit class 1868
TPrintDlg class 1888
TSearchFind class 1903
TSearchFindNext class 1917
TSearchReplace class 1926
HelpType
TBrowseForFolder class 1769
TColorSelect class 1778
TFileExit class 1814
TFileOpen class 1823
TFileOpenWith class 1833
TFilePageSetup class 1840
TFilePrintSetup class 1850
TFontEdit class 1868
TPrintDlg class 1889
TSearchFind class 1904
TSearchFindNext class 1917
TSearchReplace class 1927
HexBinaryString
TXSHexBinary class 7843
HideHint
TScreenTipsWindow class 356
HideOptions
TAdapterActionButton class 6321
TAdapterDataDisplay class 6363
TAdapterDisplayColumn class 6368
TAdapterDisplayField class 6377
TAdapterEditColumn class 6393
TCustomAdapterActionButton class 6440
HideSelection
TCustomEdit class 2368
TEdit class 2484
TMemo class 2756
HideShadow
TXPStylePopupMenu class 7781
High
DynamicArray template 3270
Hint
TBrowseForFolder class 1770
TColorSelect class 1779
TFileExit class 1814

TFileOpen class 1824
TFilePageSetup class 1840
TFilePrintSetup class 1850
TFileSaveAs class 1859
TFontEdit class 1869
THintAction class 1884
TPrintDlg class 1889
TSearchFind class 1904
TSearchFindNext class 1917
TSearchReplace class 1927
HintControl
TScreenTipsWindow class 359
hoHideAlways enumeration member 6524
hoHideOnFieldNotVisible enumeration member 6524
hoHideOnNoDisplayAccess enumeration member 6524
hoHideOnNoInputAccess enumeration member 6524
hoMode enumeration member 5901
Hook
TServiceApplication class 3093
hoOrigValues enumeration member 5901
hoRecordKeys enumeration member 5901
Host
TAbstractSocket class 22
TClientSocket class 30
TSocketConnection class 226
TSocketTransport class 235
HostedNodes
TXMLNode class 7441
HostNode
IXMLNodeAccess interface 7487
TXMLNode class 7441
HotImageIndex
TButton class 2057
TCustomButton class 2270
Hour
TXSCustomDateTime class 7817
TXSTime class 7857
HourOffset
TXSCustomDateTime class 7818
TXSTime class 7858
HREF
TAdapterImageField class 5315
TCustomAdapterImageField class 5634
HTML
THTMLItem class 6664
htmlaAnchor enumeration member 1121
htmlaButton enumeration member 1121
htmlaEventImages enumeration member 1121
htmlaImage enumeration member 1121
htmldImage enumeration member 1123
htmldList enumeration member 1123
htmldNone enumeration member 1123
HTMLDoc
TAdapterPageProducer class 1139
TCustomAdapterPageProducer class 1165
htmldText enumeration member 1123
HTMLExt
IInternetEnvOptions interface 1045
HTMLFile
TAdapterPageProducer class 1140
TCustomAdapterPageProducer class 1166
HTMLFromTemplate
TSitePageModuleHelper class 6580
htmliCheckBox enumeration member 1124
htmliFile enumeration member 1124
htmliNone enumeration member 1124
htmliPasswordInput enumeration member 1124
htmliRadio enumeration member 1124
htmliSelect enumeration member 1124
htmliSelectMultiple enumeration member 1124
htmliTextArea enumeration member 1124
htmliTextInput enumeration member 1124
HTMLSubmitFormName
TXMLBroker class 7315
htmlvmDisplay enumeration member 1123
htmlvmInput enumeration member 1123
htmlvmNone enumeration member 1123
htmlvmToggleOnAccess enumeration member 1123

HTTPWebNode
THTTPRIO class 1252
HWnd
IWebBrowser2Disp interface 771
IWebBrowserApp interface 783
IWebBrowserAppDisp interface 791
TInternetExplorer class 451
TWebBrowser class 529

I

iaBottom enumeration member 3002
iaCenter enumeration member 3002
IActionImageProducer interface 978
about IActionImageProducer interface 978
GetAdapterEventImage 980
GetDisplayStyle 980
IActionImageProducer Members 979
IActionImageProducer methods 979
IAdapterAccess interface 981
about IAdapterAccess interface 981
HasExecuteAccess 982
HasModifyAccess 982
HasViewAccess 982
IAdapterAccess Members 981
IAdapterAccess methods 981
IAdapterActionAccess interface 982
about IAdapterActionAccess interface 982
HasExecuteAccess 983
IAdapterActionAccess Members 983
IAdapterActionAccess methods 983
IAdapterActionAttributes interface 983
about IAdapterActionAttributes interface 983
GetVisible 984
IAdapterActionAttributes Members 984
IAdapterActionAttributes methods 984
IAdapterActionAttributes properties 984
Visible 984
IAdapterConverter interface 1359
about IAdapterConverter interface 1359
GetDomConverter 1360
IAdapterConverter Members 1359
IAdapterConverter methods 1359
IAdapterDispatcher interface 985
about IAdapterDispatcher interface 985
EncodeAdapterItemRequestAsFieldValue 986
EncodeAdapterItemRequestAsHREF 986
GetAdapterItemRequestParams 987
IAdapterDispatcher Members 985
IAdapterDispatcher methods 986
IAdapterDispatchParams interface 987
about IAdapterDispatchParams interface 987
GetHandler 989
GetObjectIdentifier 989
GetParams 989
GetRequestIdentifier 989
Handler 990
IAdapterDispatchParams Members 988
IAdapterDispatchParams methods 988
IAdapterDispatchParams properties 990
ObjectIdentifier 990
Params 991
RequestIdentifier 991
IAdapterDisplay interface 6510
about IAdapterDisplay interface 6510
IAdapterEditor interface 5874
about IAdapterEditor interface 5874
CanAddActionClass 5875
CanAddFieldClass 5875
IAdapterEditor Members 5875
IAdapterEditor methods 5875
IAdapterError interface 992
about IAdapterError interface 992
ErrorText 994
Field 994
GetErrorText 993
GetField 993
GetID 993
IAdapterError Members 992

IAdapterError methods 992
IAdapterError properties 993
ID 994
IAdapterErrors interface 995
about IAdapterErrors interface 995
ClearErrors 996
DefineLabel 996
ErrorCount 998
Errors 998
GetError 996
GetErrorCount 997
GetObjectErrors 997
HasObjectErrors 997
IAdapterErrors Members 995
IAdapterErrors methods 995
IAdapterErrors properties 997
IAdapterFieldAccess interface 998
about IAdapterFieldAccess interface 998
HasModifyAccess 999
HasViewAccess 999
IAdapterFieldAccess Members 999
IAdapterFieldAccess methods 999
IAdapterFieldAttributes interface 999
about IAdapterFieldAttributes interface 999
GetRequired 1000
GetVisible 1000
IAdapterFieldAttributes Members 1000
IAdapterFieldAttributes methods 1000
IAdapterFieldAttributes properties 1001
Required 1001
Visible 1001
IAdapterHiddenField interface 1001
about IAdapterHiddenField interface 1001
GetName 1002
GetValue 1002
IAdapterHiddenField Members 1002
IAdapterHiddenField methods 1002
IAdapterHiddenField properties 1003
Name 1003
Value 1003
IAdapterHiddenFields interface 1003
about IAdapterHiddenFields interface 1003
FieldCount 1005
Fields 1005
GetField 1004
GetFieldCount 1004
IAdapterHiddenFields Members 1004
IAdapterHiddenFields methods 1004
IAdapterHiddenFields properties 1004
IAdapterItemRequestParams interface 1005
about IAdapterItemRequestParams interface 1005
FailurePage 1008
GetFailurePage 1006
GetParamValues 1007
GetSuccessPage 1007
IAdapterItemRequestParams Members 1006
IAdapterItemRequestParams methods 1006
IAdapterItemRequestParams properties 1008
ParamValues 1009
SetFailurePage 1007
SetSuccessPage 1008
SuccessPage 1009
IAdapterMode interface 1009
about IAdapterMode interface 1009
AdapterMode 1011
GetAdapterMode 1010
GetNeedAdapterMode 1010
IAdapterMode Members 1010
IAdapterMode methods 1010
IAdapterMode properties 1011
NeedAdapterMode 1011
RestoreDefaultAdapterMode 1011
SetAdapterMode 1011
IAdapterNotifyAdapterChange interface 5875
about IAdapterNotifyAdapterChange interface 5875
IAdapterNotifyAdapterChange Members 5876
IAdapterNotifyAdapterChange methods 5876
NotifyAdapterChange 5876

IAdapterRequestHandler interface 1012
about IAdapterRequestHandler interface 1012
CreateRequestContext 1013
HandleRequest 1013
IAdapterRequestHandler Members 1012
IAdapterRequestHandler methods 1012
IAddFieldsEditor interface 6079
about IAddFieldsEditor interface 6079
GetAddAllFieldsItem 6080
GetAddFieldsDlgCaption 6080
GetAddFieldsItem 6080
GetNewFieldPrefix 6081
IAddFieldsEditor Members 6079
IAddFieldsEditor methods 6080
iaLeft enumeration member 363, 3002
IAppServerSOAP interface 1340
about IAppServerSOAP interface 1340
IAppServerSOAP Members 1340
IAppServerSOAP methods 1341
SAS_ApplyUpdates 1341
SAS_DataRequest 1342
SAS_Execute 1342
SAS_GetParams 1343
SAS_GetProviderNames 1344
SAS_GetRecords 1344
SAS_RowRequest 1345
iArgType
SPParamDesc class 1513
iaRight enumeration member 363, 3002
IAsFormatted interface 1013
about IAsFormatted interface 1013
AsFormatted 1014
IAsFormatted Members 1014
IAsFormatted methods 1014
iaTop enumeration member 3002
IBinding interface 7078
about IBinding interface 7078
BindingOperations 7082
GetBindingOperations 7080
GetName 7081
GetType_ 7081
IBinding Members 7079
IBinding methods 7080
IBinding properties 7081
Name 7082
SetName 7081
SetType_ 7081
Type_ 7082
IBindingInput interface 7082
about IBindingInput interface 7082
IBindingOperation interface 7083
about IBindingOperation interface 7083
Faults 7086
GetFaults 7085
GetInput 7085
GetName 7085
GetOutput 7085
IBindingOperation Members 7083
IBindingOperation methods 7085
IBindingOperation properties 7086
Input 7086
Name 7087
Output 7087
SetName 7086
IBindingOperations interface 7087
about IBindingOperations interface 7087
Add 7089
BindingOperation 7090
GetBindingOperation 7089
IBindingOperations Members 7087
IBindingOperations methods 7089
IBindingOperations properties 7090
Insert 7090
IBindingOutput interface 7090
about IBindingOutput interface 7090
IBindings interface 7091
about IBindings interface 7091
Add 7093

Binding 7094
GetBinding 7093
IBindings Members 7091
IBindings methods 7092
IBindings properties 7093
Insert 7093
ICheckOrigValue interface 5876
about ICheckOrigValue interface 5876
CheckOrigValue 5877
ICheckOrigValue Members 5877
ICheckOrigValue methods 5877
ICheckValueChange interface 5877
about ICheckValueChange interface 5877
CheckValueChange 5878
ICheckValueChange Members 5878
ICheckValueChange methods 5878
IClearAdapterValues interface 5878
about IClearAdapterValues interface 5878
ClearAdapterValues 5879
IClearAdapterValues Members 5879
IClearAdapterValues methods 5879
IComparable interface 3493
about IComparable interface 3493
CompareTo 3494
IComparable Members 3493
IComparable methods 3494
ICreateActionRequestContext interface 5879
about ICreateActionRequestContext interface 5879
CreateActionRequestContext 5880
ICreateActionRequestContext Members 5879
ICreateActionRequestContext methods 5880
Id
ISearch interface 645
ISearchDisp interface 685
ID
EAdapterException class 5178
IAdapterError interface 994
IWebSessionID interface 1109
TLanguages class 3990
TSessionItem class 382
TWebSessionImpl class 6770
IDataBlock interface 266
about IDataBlock interface 266
BytesReserved 272
Clear 267
GetBytesReserved 268
GetMemory 268
GetSignature 268
GetSize 269
GetStream 269
IDataBlock Members 267
IDataBlock methods 267
IDataBlock properties 272
IgnoreStream 269
InitData 270
Memory 272
Read 270
SetSignature 271
SetSize 271
Signature 273
Size 273
Stream 274
Write 271
IDataIntercept interface 274
about IDataIntercept interface 274
DataIn 275
DataOut 276
IDataIntercept Members 275
IDataIntercept methods 275
iDataType
SPParamDesc class 1514
IDCount
IWebSessionIDs interface 1111
IDefaultPageFileName interface 1014
about IDefaultPageFileName interface 1014
IDefaultPageFileName Members 1015
IDefaultPageFileName methods 1015
SetDefaultPageFileName 1015

IDefinition interface 7094
about IDefinition interface 7094
Bindings 7099
GetBindings 7096
GetImports 7096
GetMessages 7097
GetName 7097
GetPortTypes 7097
GetServices 7097
GetTargetNamespace 7097
GetTypes 7098
IDefinition Members 7094
IDefinition methods 7096
IDefinition properties 7098
Imports 7099
Messages 7099
Name 7099
PortTypes 7099
Services 7100
SetName 7098
SetTargetNamespace 7098
TargetNamespace 7100
Types 7100
Ident
THTMLItem class 6665
TVariableLookup class 6718
IDispatch interface 3494
about IDispatch interface 3494
GetIDsOfNames 3495
GetTypeInfo 3496
GetTypeInfoCount 3496
IDispatch Members 3495
IDispatch methods 3495
Invoke 3497
IDispatch*
Variant class 3461
IdleThreads
TServerWinSocket class 131
IDocumentation interface 7100
about IDocumentation interface 7100
IDocumentationCollection interface 7101
about IDocumentationCollection interface 7101
Add 7103
Documentation 7104
GetDocumentation 7103
IDocumentationCollection Members 7101
IDocumentationCollection methods 7103
IDocumentationCollection properties 7103
Insert 7103
IDocumented interface 7104
about IDocumented interface 7104
Documentation 7106
GetDocumentationCollection 7106
IDocumented Members 7104
IDocumented methods 7105
IDocumented properties 7106
IDOMAttr interface 7521
about IDOMAttr interface 7521
get_name 7523
get_ownerElement 7523
get_specified 7523
get_value 7524
IDOMAttr Members 7522
IDOMAttr methods 7523
IDOMAttr properties 7524
name 7524
ownerElement 7524
set_value 7524
specified 7525
value 7525
IDOMCDATASection interface 7525
about IDOMCDATASection interface 7525
IDOMCharacterData interface 7526
about IDOMCharacterData interface 7526
appendData 7527
data 7529
deleteData 7528
get_data 7528

get_length 7528
IDOMCharacterData Members 7526
IDOMCharacterData methods 7527
IDOMCharacterData properties 7529
insertData 7528
length 7529
replaceData 7528
set_data 7529
substringData 7529
IDOMComment interface 7530
about IDOMComment interface 7530
IDOMDocument interface 7530
about IDOMDocument interface 7530
createAttribute 7532
createAttributeNS 7533
createCDATASection 7533
createComment 7533
createDocumentFragment 7533
createElement 7533
createElementNS 7534
createEntityReference 7534
createProcessingInstruction 7534
createTextNode 7534
doctype 7537
documentElement 7537
domImplementation 7537
get_doctype 7535
get_documentElement 7535
get_domImplementation 7535
getElementById 7535
getElementsByTagName 7536
getElementsByTagNameNS 7536
IDOMDocument Members 7530
IDOMDocument methods 7532
IDOMDocument properties 7536
importNode 7536
set_documentElement 7536
IDOMDocumentFragment interface 7537
about IDOMDocumentFragment interface 7537
IDOMDocumentType interface 7538
about IDOMDocumentType interface 7538
entities 7541
get_entities 7540
get_internalSubset 7540
get_name 7540
get_notations 7540
get_publicId 7540
get_systemId 7541
IDOMDocumentType Members 7538
IDOMDocumentType methods 7539
IDOMDocumentType properties 7541
internalSubset 7541
name 7542
notations 7542
publicId 7542
systemId 7542
IDOMElement interface 7542
about IDOMElement interface 7542
get_tagName 7545
getAttribute 7545
getAttributeNode 7545
getAttributeNodeNS 7545
getAttributeNS 7545
getElementsByTagName 7546
getElementsByTagNameNS 7546
hasAttribute 7546
hasAttributeNS 7546
IDOMElement Members 7543
IDOMElement methods 7544
IDOMElement properties 7548
removeAttribute 7547
removeAttributeNode 7547
removeAttributeNS 7547
setAttribute 7547
setAttributeNode 7547
setAttributeNodeNS 7548
setAttributeNS 7548
tagName 7548

IDOMEntity interface 7549
about IDOMEntity interface 7549
get_notationName 7550
get_publicId 7550
get_systemId 7551
IDOMEntity Members 7549
IDOMEntity methods 7550
IDOMEntity properties 7551
notationName 7551
publicId 7551
systemId 7551
IDOMEntityReference interface 7552
about IDOMEntityReference interface 7552
IDOMHeaderProcessor interface 1241
about IDOMHeaderProcessor interface 1241
CreateHeader 1242
IDOMHeaderProcessor Members 1242
IDOMHeaderProcessor methods 1242
ProcessHeader 1242
IDOMImplementation interface 7552
about IDOMImplementation interface 7552
createDocument 7553
createDocumentType 7553
hasFeature 7553
IDOMImplementation Members 7552
IDOMImplementation methods 7553
IDOMNamedNodeMap interface 7554
about IDOMNamedNodeMap interface 7554
get_item 7555
get_length 7555
getNamedItem 7555
getNamedItemNS 7555
IDOMNamedNodeMap Members 7554
IDOMNamedNodeMap methods 7554
IDOMNamedNodeMap properties 7556
item 7557
length 7557
removeNamedItem 7556
removeNamedItemNS 7556
setNamedItem 7556
setNamedItemNS 7556
IDOMNode interface 7557
about IDOMNode interface 7557
appendChild 7559
attributes 7568
childNodes 7568
cloneNode 7560
firstChild 7569
get_attributes 7561
get_childNodes 7561
get_firstChild 7561
get_lastChild 7561
get_localName 7562
get_namespaceURI 7562
get_nextSibling 7562
get_nodeName 7562
get_nodeType 7563
get_nodeValue 7563
get_ownerDocument 7563
get_parentNode 7563
get_prefix 7564
get_previousSibling 7564
hasChildNodes 7564
IDOMNode Members 7558
IDOMNode methods 7559
IDOMNode properties 7567
insertBefore 7565
lastChild 7569
localName 7570
namespaceURI 7571
nextSibling 7571
nodeName 7572
nodeType 7572
nodeValue 7574
normalize 7565
ownerDocument 7575
parentNode 7575
prefix 7576

previousSibling 7576
removeChild 7566
replaceChild 7566
set_nodeValue 7567
supports 7567
IDOMNodeEx interface 7577
about IDOMNodeEx interface 7577
get_text 7579
get_xml 7579
IDOMNodeEx Members 7577
IDOMNodeEx methods 7578
IDOMNodeEx properties 7580
set_text 7579
text 7580
transformNode 7579, 7580
transformNode method 7579
xml 7580
IDOMNodeList interface 7581
about IDOMNodeList interface 7581
get_item 7581
get_length 7582
IDOMNodeList Members 7581
IDOMNodeList methods 7581
IDOMNodeList properties 7582
item 7582
length 7582
IDOMNodeSelect interface 7583
about IDOMNodeSelect interface 7583
IDOMNodeSelect Members 7583
IDOMNodeSelect methods 7583
selectNode 7583
selectNodes 7584
IDOMNotation interface 7584
about IDOMNotation interface 7584
get_publicId 7585
get_systemId 7586
IDOMNotation Members 7584
IDOMNotation methods 7585
IDOMNotation properties 7586
publicId 7586
systemId 7586
IDOMParseError interface 7586
about IDOMParseError interface 7586
errorCode 7589
filePos 7590
get_errorCode 7588
get_filepos 7588
get_line 7588
get_linepos 7588
get_reason 7588
get_srcText 7589
get_url 7589
IDOMParseError Members 7587
IDOMParseError methods 7587
IDOMParseError properties 7589
line 7590
linePos 7590
reason 7590
srcText 7590
url 7591
IDOMParseOptions interface 7591
about IDOMParseOptions interface 7591
async 7594
get_async 7592
get_preserveWhiteSpace 7592
get_resolveExternals 7592
get_validate 7593
IDOMParseOptions Members 7591
IDOMParseOptions methods 7592
IDOMParseOptions properties 7594
preserveWhiteSpace 7594
resolveExternals 7594
set_async 7593
set_preserveWhiteSpace 7593
set_resolveExternals 7593
set_validate 7593
validate 7595
IDOMPersist interface 7595

about IDOMPersist interface 7595
asyncLoadState 7596
get_xml 7596
IDOMPersist Members 7595
IDOMPersist methods 7596
IDOMPersist properties 7599
load 7596
loadFromStream 7597
loadFromStream method 7597
loadxml 7597
save 7597
saveToStream 7598
saveToStream method 7598
set_OnAsyncLoad 7598
xml 7599
IDOMProcessingInstruction interface 7599
about IDOMProcessingInstruction interface 7599
data 7601
get_data 7601
get_target 7601
IDOMProcessingInstruction Members 7599
IDOMProcessingInstruction methods 7600
IDOMProcessingInstruction properties 7601
set_data 7601
target 7602
IDOMText interface 7602
about IDOMText interface 7602
IDOMText Members 7602
IDOMText methods 7603
splitText 7604
IDOMXMLProlog interface 7604
about IDOMXMLProlog interface 7604
Encoding 7606
get_Encoding 7605
get_Standalone 7605
get_Version 7605
IDOMXMLProlog Members 7604
IDOMXMLProlog methods 7605
IDOMXMLProlog properties 7606
set_Encoding 7605
set_Standalone 7606
set_Version 7606
Standalone 7607
Version 7607
IDOMXSLProcessor interface 7607
about IDOMXSLProcessor interface 7607
Get_input 7608
Get_output 7608
Get_stylesheet 7609
IDOMXSLProcessor Members 7607
IDOMXSLProcessor methods 7608
IDOMXSLProcessor properties 7610
input 7610
output 7610
reset 7609
Set_input 7609
Set_output 7609
setParameter 7609
stylesheet 7611
transform 7610
IDs
IWebSessionIDs interface 1111
TAdapterErrorsList class 5266
IEchoActionFieldValue interface 5880
about IEchoActionFieldValue interface 5880
EchoActionFieldValue 5882
GetEchoActionFieldValue 5881
IEchoActionFieldValue Members 5880
IEchoActionFieldValue methods 5881
IEchoActionFieldValue properties 5881
SetEchoActionFieldValue 5881
IEchoAdapterFieldValues interface 5882
about IEchoAdapterFieldValues interface 5882
EchoActionFieldValues 5883
GetEchoActionFieldValues 5883
IEchoAdapterFieldValues Members 5882
IEchoAdapterFieldValues methods 5882
IEchoAdapterFieldValues properties 5883

SetEchoActionFieldValues 5883
IEnumerable interface 3497
about IEnumerable interface 3497
GetEnumerator 3498
IEnumerable Members 3498
IEnumerable methods 3498
IEnumerator interface 3498
about IEnumerator interface 3498
Current 3500
GetCurrent 3499
IEnumerator Members 3499
IEnumerator methods 3499
IEnumerator properties 3500
IEquatable interface 3500
about IEquatable interface 3500
Equals 3501
IEquatable Members 3501
IEquatable methods 3501
IExecuteValue interface 5883
about IExecuteValue interface 5883
ExecuteValue 5884
IExecuteValue Members 5884
IExecuteValue methods 5884
IFault interface 7106
about IFault interface 7106
IFaults interface 7107
about IFaults interface 7107
Add 7109
Fault 7110
GetFault 7109
IFaults Members 7107
IFaults methods 7109
IFaults properties 7109
Insert 7109
ifDispatch enumeration member 4751
ifDispInterface enumeration member 4751
ifHasGuid enumeration member 4751
IFieldValuesAdapter interface 1015
about IFieldValuesAdapter interface 1015
GetField 1016
GetValueIndex 1016
IFieldValuesAdapter Members 1016
IFieldValuesAdapter methods 1016
IFindIncludeFile interface 1016
about IFindIncludeFile interface 1016
FindIncludeFile 1017
IFindIncludeFile Members 1017
IFindIncludeFile methods 1017
iFldNum
TFLDDesc class 1575
iFldType
TFLDDesc class 1576
iftCheckBox enumeration member 6526
iftDefault enumeration member 6526
iftFile enumeration member 6526
iftPasswordInput enumeration member 6526
iftRadio enumeration member 6526
iftSelect enumeration member 6526
iftSelectMultiple enumeration member 6526
iftTextArea enumeration member 6526
iftTextInput enumeration member 6526
IFullyQualifyInputNames interface 5885
about IFullyQualifyInputNames interface 5885
FullyQualifyInputNames 5885
IFullyQualifyInputNames Members 5885
IFullyQualifyInputNames methods 5885
IGetAdapterActions interface 1017
about IGetAdapterActions interface 1017
GetAdapterActions 1018
IGetAdapterActions Members 1018
IGetAdapterActions methods 1018
IGetAdapterDisplayCharacteristics interface 1018
about IGetAdapterDisplayCharacteristics interface 1018
GetAdapterDisplayCharacteristics 1019
IGetAdapterDisplayCharacteristics Members 1019
IGetAdapterDisplayCharacteristics methods 1019
IGetAdapterErrors interface 1019
about IGetAdapterErrors interface 1019

GetAdapterErrors 1020
IGetAdapterErrors Members 1020
IGetAdapterErrors methods 1020
IGetAdapterErrorsList interface 5886
about IGetAdapterErrorsList interface 5886
GetAdapterErrorsList 5886
IGetAdapterErrorsList Members 5886
IGetAdapterErrorsList methods 5886
IGetAdapterFields interface 1020
about IGetAdapterFields interface 1020
GetAdapterFields 1021
IGetAdapterFields Members 1021
IGetAdapterFields methods 1021
IGetAdapterFormLayout interface 6510
about IGetAdapterFormLayout interface 6510
GetAdapterFormLayout 6511
IGetAdapterFormLayout Members 6511
IGetAdapterFormLayout methods 6511
IGetAdapterHiddenFields interface 1021
about IGetAdapterHiddenFields interface 1021
GetAdapterHiddenFields 1022
GetAdapterHiddenRecordFields 1022
IGetAdapterHiddenFields Members 1021
IGetAdapterHiddenFields methods 1022
IGetAdapterImage interface 1022
about IGetAdapterImage interface 1022
GetAdapterImage 1023
IGetAdapterImage Members 1023
IGetAdapterImage methods 1023
IGetAdapterItemRequestParams interface 1023
about IGetAdapterItemRequestParams interface 1023
GetAdapterItemRequestParams 1024
IGetAdapterItemRequestParams Members 1024
IGetAdapterItemRequestParams methods 1024
IGetAdapterModeNames interface 5887
about IGetAdapterModeNames interface 5887
GetAdapterModeNames 5887
IGetAdapterModeNames Members 5887
IGetAdapterModeNames methods 5887
IGetAdapterModeOfDisplay interface 6511
about IGetAdapterModeOfDisplay interface 6511
GetAdapterModeOfAdapter 6512
GetAdapterModeOfDisplay 6512
IGetAdapterModeOfDisplay Members 6512
IGetAdapterModeOfDisplay methods 6512
IGetAdapterOfDisplay interface 6512
about IGetAdapterOfDisplay interface 6512
GetAdapter 6513
IGetAdapterOfDisplay Members 6513
IGetAdapterOfDisplay methods 6513
IGetAdapterRequestDefaultResponse interface 1024
about IGetAdapterRequestDefaultResponse interface 1024
FailurePage 1026
GetFailurePage 1025
GetSuccessPage 1025
IGetAdapterRequestDefaultResponse Members 1025
IGetAdapterRequestDefaultResponse methods 1025
IGetAdapterRequestDefaultResponse properties 1026
SuccessPage 1026
IGetAdapterValuesList interface 1026
about IGetAdapterValuesList interface 1026
GetAdapterValuesList 1027
IGetAdapterValuesList Members 1027
IGetAdapterValuesList methods 1027
IGetAdapterVariableName interface 6513
about IGetAdapterVariableName interface 6513
AdapterVariableName 6515
GetAdapterVariableName 6514
IGetAdapterVariableName Members 6514
IGetAdapterVariableName methods 6514
IGetAdapterVariableName properties 6514
IGetControlRequirements interface 6515
about IGetControlRequirements interface 6515
GetControlRequirements 6516
IGetControlRequirements Members 6515
IGetControlRequirements methods 6515
IGetDefaultAction interface 1027
about IGetDefaultAction interface 1027

DefaultAction 1028
GetDefaultAction 1028
IGetDefaultAction Members 1027
IGetDefaultAction methods 1028
IGetDefaultAction properties 1028
IGetDesigntimeWarnings interface 1029
about IGetDesigntimeWarnings interface 1029
GetDesigntimeWarnings 1029
IGetDesigntimeWarnings Members 1029
IGetDesigntimeWarnings methods 1029
IGetFieldValuesAdapter interface 1030
about IGetFieldValuesAdapter interface 1030
GetFieldValuesAdapter 1031
IGetFieldValuesAdapter Members 1030
IGetFieldValuesAdapter methods 1030
IGetFormRequirements interface 6516
about IGetFormRequirements interface 6516
GetFormRequirements 6517
IGetFormRequirements Members 6516
IGetFormRequirements methods 6516
IGetHTMLStyle interface 1031
about IGetHTMLStyle interface 1031
GetDisplayStyle 1032
GetInputStyle 1033
GetViewMode 1033
IGetHTMLStyle Members 1031
IGetHTMLStyle methods 1032
IGetProducerComponent interface 1034
about IGetProducerComponent interface 1034
GetProducerComponent 1035
IGetProducerComponent Members 1034
IGetProducerComponent methods 1034
IGetRecordKeys interface 5888
about IGetRecordKeys interface 5888
GetRecordKeys 5888
IGetRecordKeys Members 5888
IGetRecordKeys methods 5888
IGetScriptObject interface 1035
about IGetScriptObject interface 1035
GetScriptObject 1036
IGetScriptObject Members 1035
IGetScriptObject methods 1035
IGetScriptObjectClassKey interface 1036
about IGetScriptObjectClassKey interface 1036
GetScriptObjectClassKey 1037
IGetScriptObjectClassKey Members 1036
IGetScriptObjectClassKey methods 1036
IGetVariablesContainerOfDisplay interface 6517
about IGetVariablesContainerOfDisplay interface 6517
GetVariablesContainer 6518
IGetVariablesContainerOfDisplay Members 6517
IGetVariablesContainerOfDisplay methods 6517
IGetWebAppComponents interface 1037
about IGetWebAppComponents interface 1037
GetAdapterDispatcher 1038
GetLocateFileService 1038
GetPageDispatcher 1038
GetWebApplicationInfo 1038
GetWebEndUser 1039
GetWebUserList 1039
IGetWebAppComponents Members 1037
IGetWebAppComponents methods 1038
IGetWebAppComponents properties 1039
WebApplicationInfo 1039
WebEndUser 1040
IGetWebComponentList interface 6081
about IGetWebComponentList interface 6081
GetComponentList 6082
GetDefaultComponentList 6082
IGetWebComponentList Members 6081
IGetWebComponentList methods 6081
IGetWebComponentNodeName interface 6082
about IGetWebComponentNodeName interface 6082
GetWebComponentNodeName 6083
IGetWebComponentNodeName Members 6082
IGetWebComponentNodeName methods 6083
IGetXMLStream interface 6083
about IGetXMLStream interface 6083

GetXMLStream 6084
IGetXMLStream Members 6083
IGetXMLStream methods 6084
IgnoreStream
IDataBlock interface 269
TDataBlock class 202
IHeaderInfo interface 7247
about IHeaderInfo interface 7247
EncodingStyle 7251
GetEncodingStyle 7248
GetName 7248
GetNamespace 7249
GetPart 7249
GetRequired 7249
GetUse 7249
IHeaderInfo Members 7247
IHeaderInfo methods 7248
IHeaderInfo properties 7251
Name 7251
Namespace 7251
Part 7252
Required 7252
SetEncodingStyle 7249
SetName 7250
SetNamespace 7250
SetPart 7250
SetRequired 7250
SetUse 7251
Use 7252
IHTTPReqResp interface 1296
about IHTTPReqResp interface 1296
GetHTTPReqResp 1297
IHTTPReqResp Members 1296
IHTTPReqResp methods 1296
IHTTPSoapDispatch interface 1262
about IHTTPSoapDispatch interface 1262
DispatchSOAP 1263
IHTTPSoapDispatch Members 1262
IHTTPSoapDispatch methods 1263
iiaCenter enumeration member 364
iiaLeft enumeration member 364
iiaRight enumeration member 364
IIDEFileManager interface 1040
about IIDEFileManager interface 1040
IIDEFileManager Members 1040
IIDEFileManager methods 1040
SearchFile 1041
IIdentifyAdapter interface 5889
about IIdentifyAdapter interface 5889
IImageProducer interface 1041
about IImageProducer interface 1041
GetAdapterImage 1042
IImageProducer Members 1042
IImageProducer methods 1042
IImport interface 7110
about IImport interface 7110
GetLocation 7112
GetNamespace 7112
IImport Members 7110
IImport methods 7112
IImport properties 7113
Location 7113
Namespace 7113
SetLocation 7112
SetNamespace 7112
IImports interface 7113
about IImports interface 7113
Add 7115
GetImport 7116
IImports Members 7114
IImports methods 7115
IImports properties 7116
Import 7116
Insert 7116
IIntegerTest interface 1332
about IIntegerTest interface 1332
Add 1333
Doubler 1333

IIntegerTest Members 1332
IIntegerTest methods 1332
IInterface interface 3502
_AddRef 3503
_Release 3503
about IInterface interface 3502
IInterface Members 3502
IInterface methods 3503
QueryInterface 3504
IInterface*
Variant class 3461
IInternetEnvOptions interface 1042
about IInternetEnvOptions interface 1042
DisableDebugger 1044
GetDisableDebugging 1043
GetHTMLExt 1044
GetSampleImageFile 1044
GetScriptTimeoutSecs 1044
HTMLExt 1045
IInternetEnvOptions Members 1043
IInternetEnvOptions methods 1043
IInternetEnvOptions properties 1044
SampleImageFile 1045
ScriptTimeoutSecs 1045
IInternetExecuteScript interface 1045
about IInternetExecuteScript interface 1045
CanExecuteScript 1046
ExecuteScript 1046
IInternetExecuteScript Members 1046
IInternetExecuteScript methods 1046
IInternetFileTypes interface 1047
about IInternetFileTypes interface 1047
FindFileExt 1047
IInternetFileTypes Members 1047
IInternetFileTypes methods 1047
IInternetScriptDebugger interface 1048
about IInternetScriptDebugger interface 1048
CanDebugScript 1048
DebugScript 1049
IInternetScriptDebugger Members 1048
IInternetScriptDebugger methods 1048
IInvokable interface 3504
about IInvokable interface 3504
IInvokerConverter interface 1360
about IInvokerConverter interface 1360
GetInvoker 1361
IInvokerConverter Members 1360
IInvokerConverter methods 1360
IIsAdapterActionList interface 1049
about IIsAdapterActionList interface 1049
GetIsAdapterActionList 1050
IIsAdapterActionList Members 1049
IIsAdapterActionList methods 1049
IIsAdapterActionList properties 1050
IsAdapterActionList 1050
IIterateIntfSupport interface 1050
about IIterateIntfSupport interface 1050
GetCurrentIntf 1051
IIterateIntfSupport Members 1051
IIterateIntfSupport methods 1051
IIterateObjectSupport interface 1051
about IIterateObjectSupport interface 1051
GetCurrentObject 1052
IIterateObjectSupport Members 1052
IIterateObjectSupport methods 1052
IIteratorIndex interface 1052
about IIteratorIndex interface 1052
GetIteratorIndex 1053
IIteratorIndex Members 1053
IIteratorIndex methods 1053
IIteratorIndex properties 1054
InIterator 1053
IteratorIndex 1054
IIteratorSupport interface 1054
about IIteratorSupport interface 1054
EndIterator 1056
IIteratorSupport Members 1055
IIteratorSupport methods 1055

NextIteration 1056
StartIterator 1056
ILayoutWebContent interface 6084
about ILayoutWebContent interface 6084
ILayoutWebContent Members 6085
ILayoutWebContent methods 6085
LayoutButton 6085
LayoutField 6085
LayoutLabelAndField 6086
LayoutTable 6086
iLen
SPParamDesc class 1514
TFLDDesc class 1576
Image
TScreenTipItem class 333
TScreenTipsWindow class 359
ImageAlignment
TButton class 2057
TCustomButton class 2271
ImageHeight
TAdapterActionButton class 6321
TAdapterDisplayColumn class 6369
TAdapterDisplayField class 6377
TAdapterEditColumn class 6393
TCustomAdapterActionButton class 6440
TCustomAdapterDisplayColumn class 6466
TCustomAdapterDisplayField class 6473
ImageIndex
TBrowseForFolder class 1770
TButton class 2058
TColorSelect class 1779
TCustomButton class 2271
TFileExit class 1814
TFileOpen class 1824
TFilePageSetup class 1841
TFilePrintSetup class 1850
TFileSaveAs class 1860
TFontEdit class 1869
TPrintDlg class 1889
TSearchFind class 1905
TSearchFindNext class 1918
TSearchReplace class 1927
ImageMargins
TButton class 2058
TCustomButton class 2272
ImageProducer
TAdapterActionButton class 6322
TCustomAdapterActionButton class 6441
Images
TButton class 2059
TCustomButton class 2272
TTabSet class 4496
ImageWidth
TAdapterActionButton class 6322
TAdapterDisplayColumn class 6369
TAdapterDisplayField class 6378
TAdapterEditColumn class 6393
TCustomAdapterActionButton class 6441
TCustomAdapterDisplayColumn class 6467
TCustomAdapterDisplayField class 6474
ImeMode
TComboBox class 2206
TEdit class 2484
TListBox class 2678
TMemo class 2757
ImeName
TComboBox class 2206
TEdit class 2485
TListBox class 2678
TMemo class 2757
IMessage interface 7116
about IMessage interface 7116
GetName 7118
GetParts 7119
IMessage Members 7117
IMessage methods 7118
IMessage properties 7119
Name 7119

Parts 7119
SetName 7119
IMessages interface 7120
about IMessages interface 7120
Add 7122
GetMessage 7122
IMessages Members 7120
IMessages methods 7122
IMessages properties 7122
Insert 7122
Message 7123
IMimeAttachmentHandler interface 1176
about IMimeAttachmentHandler interface 1176
AddSoapHeader 1177
CreateMimeStream 1178
FinalizeStream 1178
GetMIMEBoundary 1179
GetMIMEStream 1179
GetOnGetAttachmentEvent 1180
GetOnSendAttachmentEvent 1180
IMimeAttachmentHandler events 1183
IMimeAttachmentHandler Members 1176
IMimeAttachmentHandler methods 1177
IMimeAttachmentHandler properties 1182
MIMEBoundary 1182
OnGetAttachment 1183
OnSendAttachment 1183
ProcessMultiPartForm 1180
SetMIMEBoundary 1181
SetOnGetAttachmentEvent 1181
SetOnSendAttachmentEvent 1182
ImplAsFormatted
TAdapterNamedField class 5361
TAdapterUpdateField class 5427
TFieldValuesValueField class 5816
ImplCanAddActionClass
TBaseEndUserAdapter class 5510
TCustomAdapter class 5553
TCustomApplicationAdapter class 5669
TCustomLoginFormAdapter class 5708
TDefaultFieldsPagedAdapter class 5757
ImplCanAddClass
TAdapterActions class 5212
TAdapterFields class 5279
TAdapterSubComponent class 5414
TBaseAdapterPageProducer class 1155
TCustomAdapterCommandColumn class 6447
TCustomAdapterCommandGroup class 6458
TCustomAdapterDisplayColumn class 6465
TCustomAdapterFieldGroup class 6488
TCustomAdapterForm class 6493
TCustomAdapterGrid class 6501
ImplCanAddClassHelper
TWebComponentsEditorHelper class 6037
ImplCanAddFieldClass
TAbstractFieldValuesAdapter class 5187
TBaseEndUserAdapter class 5510
TBaseValuesListAdapter class 5525
TCustomAdapter class 5553
TCustomApplicationAdapter class 5670
TCustomLoginFormAdapter class 5709
ImplCheckValueChange
TAdapterUpdateField class 5428
TCustomAdapterFileField class 5611
TCustomAdapterImageField class 5631
TCustomAdapterMemoField class 5643
ImplClearAdapterValues
TCustomAdapter class 5554
ImplClearErrors
TAdapterErrorsList class 5264
TBaseAdapterErrorsList class 5485
ImplContent
TAdapterDataDisplay class 6360
TCustomAdapterActionButton class 6435
TCustomAdapterCommandColumn class 6447
TCustomAdapterCommandGroup class 6458
TCustomAdapterDisplayField class 6471
TCustomAdapterEditColumn class 6478

TCustomAdapterErrorList class 6481
TCustomAdapterFieldGroup class 6488
TCustomAdapterForm class 6494
TCustomAdapterGrid class 6502
ImplCreateRequestContext
TBaseAdapterAction class 5464
TCustomAdapterImageField class 5631
ImplDefineLabel
TAdapterErrorsList class 5264
TAdapterObjectErrorsList class 5390
TBaseAdapterErrorsList class 5485
ImplDispatchPage
TSitePageModuleHelper class 6581
ImplDispatchPageName
TCustomPageDispatcher class 6192
ImplEncodeAdapterItemRequestAsFieldValue
TCustomAdapterDispatcher class 6175
ImplEncodeAdapterItemRequestAsHREF
TCustomAdapterDispatcher class 6175
ImplEndIterator
TAbstractFieldValuesAdapter class 5188
TBaseAdapterAction class 5464
TCustomAdapter class 5554
TCustomStringsValuesList class 5734
TDefaultFieldsPagedAdapter class 5757
TFieldValuesAdapter class 5810
ImplExecuteActionRequest
TBaseAdapterAction class 5464
TCustomAdapterAction class 5584
TCustomAdapterGotoPageAction class 5620
TCustomAdapterPageAction class 5662
TEndUserAdapterLoginFormAction class 5781
TEndUserAdapterLogoutAction class 5789
TLoginFormAdapterLoginAction class 5840
ImplFindIncludeFile
TCustomLocateFileService class 6185
ImplFindVariable
TCustomAdapter class 5554
ImplFindVariableContainer
TCustomAdapter class 5555
ImplFinishContext
TCustomWebAppComponents class 6207
TMultiModuleWebAppServices class 6229
ImplFormatColumnData
TAdapterDataDisplay class 6360
ImplFormatColumnHeading
TAdapterDataDisplay class 6361
TCustomAdapterDisplayColumn class 6465
ImplGetActionName
TBaseAdapterAction class 5465
TCustomAdapterAction class 5584
ImplGetActionsList
TBaseEndUserAdapter class 5511
TCustomAdapter class 5555
TCustomApplicationAdapter class 5670
TCustomLoginFormAdapter class 5709
TDefaultFieldsPagedAdapter class 5757
ImplGetAdapterItemRequestParams
TBaseAdapterAction class 5465
TCustomAdapterAction class 5584
TCustomAdapterDispatcher class 6176
TCustomAdapterGotoPageAction class 5620
TCustomAdapterPageAction class 5662
TLoginFormAdapterLoginAction class 5841
ImplGetCanView
TCustomPageDispatcher class 6192
ImplGetControlRequirements
TAdapterControlGroup class 6350
TAdapterDataDisplay class 6361
TCustomAdapterActionButton class 6435
TCustomAdapterDisplayColumn class 6465
TCustomAdapterDisplayField class 6471
ImplGetCurrentValue
TBaseAdapterMultiValueField class 5495
ImplGetDefaultComponentList
TWebContainedContainerComponent class 6058
TWebContainerComponent class 6066
ImplGetDefaultPageName

TCustomPageDispatcher class 6192
ImplGetDisplayLabel
TAdapterNamedDisplayField class 5348
TAdapterNamedField class 5361
TAdapterUpdateField class 5428
TBaseAdapterAction class 5465
TCustomAdapterGotoPageAction class 5621
ImplGetDisplayName
TBaseEndUserAdapter class 5511
ImplGetDisplayWidth
TAdapterNamedDisplayField class 5349
TAdapterNamedField class 5362
TAdapterUpdateField class 5428
ImplGetEchoActionFieldValue
TAdapterNamedField class 5362
TAdapterNextPageField class 5384
TAdapterUpdateField class 5429
ImplGetEchoActionFieldValues
TCustomAdapter class 5555
ImplGetEndUserAdapter
TBaseEndUserAdapter class 5511
ImplGetError
TAdapterErrorsList class 5264
TAdapterObjectErrorsList class 5390
TBaseAdapterErrorsList class 5485
ImplGetErrorCount
TAdapterErrorsList class 5265
TAdapterObjectErrorsList class 5390
TBaseAdapterErrorsList class 5485
ImplGetExecuteActionResponse
TBaseAdapterAction class 5466
ImplGetFailureOptions
TBaseAdapterAction class 5466
ImplGetField
TAbstractFieldValuesAdapter class 5188
TFieldValuesAdapter class 5810
ImplGetFieldsList
TAbstractFieldValuesAdapter class 5188
TBaseEndUserAdapter class 5512
TBaseValuesListAdapter class 5525
TCustomAdapter class 5556
TCustomApplicationAdapter class 5670
TCustomLoginFormAdapter class 5710
ImplGetFieldValuesAdapter
TBaseAdapterMultiValueField class 5496
ImplGetInputName
TAdapterNamedField class 5362
ImplGetIsAdapterActionList
TBaseAdapterAction class 5466
TCustomAdapterGotoPageAction class 5621
ImplGetIteratorIndex
TBaseAdapterAction class 5467
TCustomAdapter class 5556
ImplGetListImage
TBaseValuesListAdapter class 5525
ImplGetListImageOfValue
TBaseValuesListAdapter class 5526
ImplGetListName
TBaseValuesListAdapter class 5526
TCustomStringsValuesList class 5735
ImplGetListNameOfValue
TBaseValuesListAdapter class 5526
TCustomStringsValuesList class 5735
ImplGetListValue
TBaseValuesListAdapter class 5527
TCustomStringsValuesList class 5735
ImplGetLoggedIn
TBaseEndUserAdapter class 5512
ImplGetLoginFormAction
TBaseEndUserAdapter class 5512
ImplGetLoginRequired
TCustomPageDispatcher class 6193
ImplGetLogoutAction
TBaseEndUserAdapter class 5513
ImplGetMaxLength
TAdapterNamedField class 5363
ImplGetObjectErrors
TAdapterErrorsList class 5265

TBaseAdapterErrorsList class 5486
ImplGetRequired
TAdapterNamedDisplayField class 5349
TAdapterNamedField class 5363
ImplGetSuccessOptions
TBaseAdapterAction class 5467
TCustomAdapterAction class 5585
ImplGetTemplateStream
TCustomLocateFileService class 6185
ImplGetValue
TAdapterApplicationTitleField class 5222
TAdapterEndUserDisplayNameField class 5247
TAdapterEndUserLoggedInField class 5255
TAdapterNamedField class 5363
TAdapterNextPageField class 5385
TAdapterPasswordField class 5395
TAdapterUserNameField class 5437
TCustomAdapterBooleanField class 5595
TCustomAdapterField class 5602
TCustomAdapterImageField class 5632
TCustomAdapterMemoField class 5644
TFieldValuesValueField class 5817
TValuesListNameField class 5869
TValuesListValueField class 5873
ImplGetValueIndex
TAbstractFieldValuesAdapter class 5189
TFieldValuesAdapter class 5811
ImplGetVariable
TCustomAdapter class 5556
ImplGetVariableCount
TCustomAdapter class 5557
ImplGetVariableName
TAdapterNamedField class 5364
TAdapterUpdateField class 5429
TBaseAdapterAction class 5467
TCustomAdapter class 5557
ImplGetVisible
TAdapterNamedField class 5364
TBaseAdapterAction class 5468
ImplGetVisibleActions
TCustomAdapter class 5557
TDefaultFieldsAdapter class 5745
ImplGetVisibleFields
TCustomAdapter class 5558
TDefaultFieldsAdapter class 5745
ImplGetWebComponentNodeName
TAdapterActions class 5213
TAdapterFields class 5280
TAdapterSubComponent class 5415
ImplHandleException
TCustomWebAppComponents class 6207
TMultiModuleWebAppServices class 6230
ImplHandleRequest
TBaseAdapterAction class 5468
TBaseRequestHandler class 6167
TCustomAdapterDispatcher class 6176
TCustomAdapterImageField class 5632
TCustomPageDispatcher class 6193
TCustomWebAppComponents class 6208
TMultiModuleWebAppServices class 6230
ImplHandleScriptError
TScriptProducer class 6712
ImplHasExecuteAccess
TBaseAdapterAction class 5468
TCustomAdapter class 5558
ImplHasExecuteRights
TBaseEndUserAdapter class 5513
ImplHasModifyAccess
TAdapterNamedField class 5365
TCustomAdapter class 5558
ImplHasModifyRights
TBaseEndUserAdapter class 5513
ImplHasObjectErrors
TAdapterErrorsList class 5265
TBaseAdapterErrorsList class 5486
ImplHasValue
TBaseAdapterMultiValueField class 5496
TCustomAdapterMultiValueField class 5653

ImplHasViewAccess
TAdapterNamedField class 5365
TCustomAdapter class 5559
ImplHasViewRights
TBaseEndUserAdapter class 5514
ImplIncludePage
TSitePageModuleHelper class 6581
ImplInitContext
TCustomWebAppComponents class 6208
TMultiModuleWebAppServices class 6231
ImplInIterator
TBaseAdapterAction class 5469
TCustomAdapter class 5559
ImplIsDefaultAction
TBaseAdapterAction class 5469
TCustomAdapterGotoPageAction class 5622
TCustomAdapterPageAction class 5663
ImplIsEqual
TCustomAdapterBooleanField class 5596
ImplLayoutButton
TAdapterFormLayout class 6413
TLayoutWebContent class 6017
ImplLayoutField
TAdapterFormLayout class 6414
TLayoutWebContent class 6018
ImplLayoutLabelAndField
TAdapterFormLayout class 6414
TLayoutWebContent class 6018
ImplLayoutTable
TAdapterFormLayout class 6415
TLayoutWebContent class 6019
ImplLogIn
TBaseEndUserAdapter class 5514
ImplNextIteration
TAbstractFieldValuesAdapter class 5189
TBaseAdapterAction class 5470
TCustomAdapter class 5560
TCustomStringsValuesList class 5736
TDefaultFieldsPagedAdapter class 5758
TFieldValuesAdapter class 5811
ImplNotifyActivate
TCustomAdapter class 5560
ImplNotifyDeactivate
TCustomAdapter class 5560
TCustomLoginFormAdapter class 5710
TCustomStringsValuesList class 5736
TDefaultFieldsPagedAdapter class 5758
ImplPageNameOfRequest
TCustomPageDispatcher class 6193
ImplProduceLoginPage
TBaseEndUserAdapter class 5514
ImplRedirectToPage
TSitePageModuleHelper class 6581
ImplRedirectToPageName
TCustomPageDispatcher class 6193
ImplRestoreDefaults
TAdapterDataDisplay class 6361
TCustomAdapterActionButton class 6436
TCustomAdapterDisplayColumn class 6465
TCustomAdapterDisplayField class 6471
ImplSetActionName
TCustomAdapterAction class 5585
ImplSetActions
TCustomAdapter class 5561
TDefaultFieldsAdapter class 5746
ImplSetData
TCustomAdapter class 5561
TDefaultFieldsAdapter class 5746
ImplSetEchoActionFieldValue
TAdapterNamedField class 5365
TAdapterNextPageField class 5385
TAdapterUpdateField class 5429
ImplSetEchoActionFieldValues
TCustomAdapter class 5561
ImplSetFieldName
TAdapterNamedField class 5366
TAdapterUpdateField class 5429
ImplSetWebContentOptions

TSitePageModuleHelper class 6581
ImplStartIterator
TAbstractFieldValuesAdapter class 5190
TBaseAdapterAction class 5470
TCustomAdapter class 5562
TCustomStringsValuesList class 5736
TDefaultFieldsPagedAdapter class 5759
TFieldValuesAdapter class 5812
ImplUpdateValue
TAdapterUpdateField class 5430
TCustomAdapterBooleanField class 5596
TCustomAdapterField class 5603
TCustomAdapterFileField class 5611
TCustomAdapterImageField class 5632
TCustomAdapterMemoField class 5644
TCustomAdapterMultiValueField class 5653
ImplValidateValue
TAdapterUpdateField class 5430
ImplWebEnabled
TBaseAdapterAction class 5470
TCustomAdapterAction class 5585
TCustomAdapterGotoPageAction class 5622
TCustomAdapterPageAction class 5663
TEndUserAdapterLoginFormAction class 5781
TEndUserAdapterLogoutAction class 5789
ImplWebImageHREF
TCustomAdapterImageField class 5633
Import
IImports interface 7116
IShellFavoritesNameSpace interface 693
IShellFavoritesNameSpaceDisp interface 697
IShellNameSpaceDisp interface 711
TShellNameSpace class 474
ImportExportFavorites
IShellUIHelper interface 719
IShellUIHelperDisp interface 723
TShellUIHelper class 492
ImportLocations
TWebServExp class 6732
ImportNames
TWebServExp class 6732
importNode
IDOMDocument interface 7536
Imports
IDefinition interface 7099
InactiveCount
TWebRequestHandler class 6639
incFile enumeration member 6721
IncludePage
IPageResult interface 1065
TSitePageModuleHelper class 6582
incPage enumeration member 6721
IncRefCount
DynamicArray template 3268
Index
IWebComponent interface 6097
IndexDefs
TCustomSQLDataSet class 1558
TSimpleDataSet class 918
IndexFieldCount
TSQLTable class 1689
IndexFieldNames
TSimpleDataSet class 919
TSQLTable class 1689
IndexFields
TSQLTable class 1690
IndexName
TSimpleDataSet class 920
TSQLTable class 1690
IndexOf
IXMLNodeList interface 7711
TCustomComboBoxStrings class 2343
TDOMStrings class 7183
TLanguages class 3988
TWebComponentList class 6030
TWideStringList class 6818
TWideStrings class 6840
TXMLNodeList class 7471

IndexOfIgnoreCase
TDOMStrings class 7183
IndexOfName
TAbstractNamedVariants class 6128
TWideStrings class 6840
IndexOfObject
TWideStrings class 6841
InheritsFrom
TObject class 3376
InitContext
TMultiModuleWebAppServices class 6231
InitControlData
TWebBrowser class 512
TWebBrowser_V1 class 574
InitData
IDataBlock interface 270
TDataBlock class 202
InIterator
IIteratorIndex interface 1053
TBaseAdapterAction class 5471
TCustomAdapter class 5562
Initialize
TCriticalSectionHelper class helper 3132
TServiceApplication class 3093
TSocketForm class 176
TWebApplication class 5939
Initialized
DShellNameSpaceEvents interface 619
InitInstance
TObject class 3377
InitModule
TCustomWebAppComponents class 6209
TMultiModuleWebAppServices class 6232
InitModules
TCustomWebAppComponents class 6210
TMultiModuleWebAppServices class 6233
InitRecord
TCustomSQLDataSet class 1535
InitServerData
TInternetExplorer class 438
TShellNameSpace class 474
TShellUIHelper class 492
TShellWindows class 499
InitSocket
TAbstractSocket class 18
TCustomWinSocket class 72
InnerException
Exception class 3965
INotifyActivateModule interface 6087
about INotifyActivateModule interface 6087
INotifyActivateModule Members 6087
INotifyActivateModule methods 6087
NotifyActivateModule 6087
NotifyDeactivateModule 6088
INotifyAdapterChange interface 1057
about INotifyAdapterChange interface 1057
INotifyAdapterChange Members 1057
INotifyAdapterChange methods 1058
NotifyAdapterChange 1058
INotifyConnectionChange interface 7337
about INotifyConnectionChange interface 7337
ConnectionChange 7338
INotifyConnectionChange Members 7337
INotifyConnectionChange methods 7337
INotifyList interface 1058
about INotifyList interface 1058
AddNotify 1059
INotifyList Members 1059
INotifyList methods 1059
RemoveNotify 1059
INotifyWebActivate interface 6088
about INotifyWebActivate interface 6088
INotifyWebActivate Members 6088
INotifyWebActivate methods 6088
NotifyActivate 6089
NotifyDeactivate 6089
input
IDOMXSLProcessor interface 7610

Input
IBindingOperation interface 7086
IOperation interface 7126
InputName
IWebInputName interface 1100
InputType
TAdapterDisplayField class 6378
TAdapterEditColumn class 6394
TCustomAdapterDisplayField class 6474
Insert
AnsiStringBase class 3193
AnsiStringT template 3223
IBindingOperations interface 7090
IBindings interface 7093
IDocumentationCollection interface 7103
IFaults interface 7109
IImports interface 7116
IMessages interface 7122
IOperations interface 7130
IParts interface 7141
IPorts interface 7148
IPortTypes interface 7154
IServices interface 7161
IXMLNodeList interface 7712
TAbstractNamedVariants class 6128
TBindingOperations class 6923
TBindings class 6929
TDocumentationCollection class 6945
TDOMStrings class 7183
TFaults class 6958
TImports class 6971
TMessages class 6984
TNamedVariantsList class 6138
TOperations class 6998
TParts class 7018
TPorts class 7031
TPortTypes class 7044
TScreenTipsCollection class 336
TServices class 7057
TSoapDataList class 1174
TStringBuilder class 4018
TTabList class 4469
TWideStringList class 6819
TWideStrings class 6841
TXMLNodeList class 7472
UnicodeString class 3401
WideString class 3479
insertBefore
IDOMNode interface 7565
insertData
IDOMCharacterData interface 7528
InsertInCollection
TXMLNodeCollection class 7455
InsertItem
TValueListStrings class 4889
TWideStringList class 6819
InsertObject
TWideStringList class 6819
TWideStrings class 6842
InsertRow
TValueListEditor class 4796
Installing
TServiceApplication class 3093
InstanceCount
ECustomException class 1310
InstanceSize
TObject class 3377
int
Currency class 3252
TDateTime class 3340
Variant class 3461
int*
Variant class 3461
IntegerExit
TSocketForm class 177
IntegralHeight
TCustomListBox class 2428
TListBox class 2679

Interactive
TService class 3080
InterceptGroup
TSocketForm class 163
InterceptGUID
TSocketConnection class 226
TSocketForm class 164
TSocketTransport class 235
TStreamedConnection class 246
InterceptName
TSocketConnection class 227
TStreamedConnection class 247
InterfaceSupportsErrorInfo
TLocalConnection class 4552
InternalAddChild
IXMLNodeAccess interface 7485
TXMLNode class 7428
InternalApplyUpdates
TXMLTransformProvider class 7750
InternalClose
TCustomSQLDataSet class 1535
TStreamedConnection class 242
InternalConnection
TCustomSQLDataSet class 1559
InternalCreateObject
TCustomDataBlockInterpreter class 195
TDataBlockInterpreter class 213
InternalFreeCommand
TCustomSQLDataSet class 1535
InternalGetItemData
TCustomListBox class 2411
InternalGetRecords
TXMLTransformProvider class 7750
InternalHandleException
TCustomSQLDataSet class 1535
InternalInitFieldDefs
TCustomSQLDataSet class 1536
InternalInsert
TXMLNodeList class 7472
InternalOpen
TCustomSQLDataSet class 1536
TStreamedConnection class 243
InternalRefresh
TCustomSQLDataSet class 1536
InternalSetItemData
TCustomListBox class 2411
InternalSetPrologValue
TXMLDocument class 7367
internalSubset
IDOMDocumentType interface 7541
InterpretData
TCustomDataBlockInterpreter class 196
TDataBlockInterpreter class 213
Interpreter
TStreamedConnection class 247
IntfInfo
TWSDLView class 7272
InTransaction
TSQLConnection class 1613
IntToHex
AnsiStringBase class 3194
AnsiStringT template 3224
UnicodeString class 3402
WideString class 3480
iNullOffset
TFLDDesc class 1576
Invoke
IDispatch interface 3497
TDataDispatch class 219
TSoapPascalInvoker class 1354
InvokeCallBack
TSQLMonitor class 1646
InvokeContextMenuCommand
IShellFavoritesNameSpace interface 693
IShellFavoritesNameSpaceDisp interface 698
IShellNameSpaceDisp interface 711
TShellNameSpace class 475
InvokeEvent

TInternetExplorer class 438
TShellNameSpace class 475
TShellWindows class 500
InvokeOptions
THTTPReqResp class 1288
Invoker
TLinkedWebNode class 1326
InWebFolder
ISearchAssistantOC interface 655
ISearchAssistantOC2Disp interface 665
ISearchAssistantOC3Disp interface 676
ISearchAssistantOCDisp interface 684
ioBeforeLoad enumeration member 7261
IObjectGetter interface 1333
about IObjectGetter interface 1333
GetAsObject 1334
IObjectGetter Members 1333
IObjectGetter methods 1334
iOffset
TFLDDesc class 1576
ioLoaded enumeration member 7261
IOperation interface 7123
about IOperation interface 7123
Faults 7126
GetFaults 7125
GetInput 7125
GetName 7125
GetOutput 7126
Input 7126
IOperation Members 7123
IOperation methods 7125
IOperation properties 7126
Name 7127
Output 7127
SetName 7126
IOperations interface 7127
about IOperations interface 7127
Add 7129, 7130
Add method 7129
GetOperation 7130
Insert 7130
IOperations Members 7127
IOperations methods 7129
IOperations properties 7130
Operation 7130
IPageDispatcher interface 1060
about IPageDispatcher interface 1060
DefaultPageName 1064
DispatchPageName 1061
GetCanView 1062
GetDefaultPageName 1062
GetLoginRequired 1062
IPageDispatcher Members 1060
IPageDispatcher methods 1061
IPageDispatcher properties 1064
PageNameOfRequest 1063
RedirectToPageName 1063
IPageResult interface 1064
about IPageResult interface 1064
DispatchPage 1065
IncludePage 1065
IPageResult Members 1065
IPageResult methods 1065
RedirectToPage 1066
IParam interface 7131
about IParam interface 7131
GetMessage 7133
GetName 7133
IParam Members 7131
IParam methods 7132
IParam properties 7134
Message 7134
Name 7134
SetMessage 7133
SetName 7133
iParamNum
SPParamDesc class 1514
IPart interface 7134

about IPart interface 7134
Element 7138
GetElement 7136
GetName 7137
GetType_ 7137
IPart Members 7135
IPart methods 7136
IPart properties 7138
Name 7138
SetElement 7137
SetName 7137
SetType_ 7137
Type_ 7138
IParts interface 7139
about IParts interface 7139
Add 7141
GetPart 7141
Insert 7141
IParts Members 7139
IParts methods 7140
IParts properties 7141
Part 7142
IPort interface 7142
about IPort interface 7142
Binding 7145
GetBinding 7144
GetName 7144
IPort Members 7142
IPort methods 7144
IPort properties 7145
Name 7145
SetBinding 7144
SetName 7144
IPorts interface 7145
about IPorts interface 7145
Add 7147
GetPort 7148
Insert 7148
IPorts Members 7146
IPorts methods 7147
IPorts properties 7148
Port 7148
IPortType interface 7148
about IPortType interface 7148
GetName 7150
GetOperations 7151
IPortType Members 7149
IPortType methods 7150
IPortType properties 7151
Name 7151
Operations 7151
SetName 7151
IPortTypes interface 7152
about IPortTypes interface 7152
Add 7154
GetPortType 7154
Insert 7154
IPortTypes Members 7152
IPortTypes methods 7154
IPortTypes properties 7154
PortType 7155
IProducerEditorViewSupport interface 6089
about IProducerEditorViewSupport interface 6089
GetTemplateFileType 6090
GetXMLData 6090
GetXSLData 6090
HasHTMLBrowserView 6091
HasScriptView 6091
HasXMLBrowserView 6091
HasXSLBrowserView 6091
IProducerEditorViewSupport Members 6089
IProducerEditorViewSupport methods 6090
IQualifiedName interface 7252
about IQualifiedName interface 7252
GetName 7253
GetNamespace 7254
IQualifiedName Members 7253
IQualifiedName methods 7253

IQualifiedName properties 7255
Name 7255
Namespace 7255
SetName 7254
SetNamespace 7254
IReadWriteSync interface 4037
about IReadWriteSync interface 4037
BeginRead 4038
BeginWrite 4038
EndRead 4038
EndWrite 4038
IReadWriteSync Members 4037
IReadWriteSync methods 4037
IsActionInUse
IWebActionsList interface 1078
TAdapterActionsListHelper class 6330
TCustomAdapter class 5563
IsActive
IWebSessionAttributes interface 1107
TSessionItem class 380
IsAdapterActionList
IIsAdapterActionList interface 1050
IsArray
Variant class 3433
IsCheckedLinked
TButtonActionLink class 2108
IsClear
TCustomVariantType class 5000
IsCollectionItem
TXMLNodeCollection class 7455
IsColumn
TCustomAdapterDisplayColumn class 6466
IScreenTipItem interface 361
about IScreenTipItem interface 361
GetScreenTipItem 362
IScreenTipItem Members 361
IScreenTipItem methods 361
IScriptErrorList interface 635
about IScriptErrorList interface 635
advanceError 636
canAdvanceError 636
canRetreatError 637
getAlwaysShowLockState 637
getDetailsPaneOpen 637
getErrorChar 637
getErrorCode 637
getErrorLine 638
getErrorMsg 638
getErrorUrl 638
getPerErrorDisplay 638
IScriptErrorList Members 635
IScriptErrorList methods 636
retreatError 639
setDetailsPaneOpen 639
setPerErrorDisplay 639
IScriptErrorListDisp interface 639
about IScriptErrorListDisp interface 639
advanceError 640
canAdvanceError 641
canRetreatError 641
getAlwaysShowLockState 641
getDetailsPaneOpen 641
getErrorChar 642
getErrorCode 642
getErrorLine 642
getErrorMsg 642
getErrorUrl 642
getPerErrorDisplay 643
IScriptErrorListDisp Members 640
IScriptErrorListDisp methods 640
retreatError 643
setDetailsPaneOpen 643
setPerErrorDisplay 643
IsCursorOpen
TCustomSQLDataSet class 1536
IsDefaultAction
IWebIsDefaultAction interface 1101
TBaseAdapterAction class 5471

IsDefaultField
IWebIsDefaultField interface 1102
TWebContainedComponent class 6042
TWebContainedComponentHelper class 6048
TWebContainedContainerComponent class 6058
IsDelimiter
AnsiStringBase class 3194
AnsiStringT template 3224
UnicodeString class 3402
WideString class 3480
ISearch interface 644
about ISearch interface 644
Get_Id 645
Get_Title 645
Get_URL 645
Id 645
ISearch Members 644
ISearch methods 644
ISearch properties 645
Title 646
URL 646
ISearchAssistantOC interface 646
about ISearchAssistantOC interface 646
AddNextMenuItem 648
ASProvider 654
ASSetting 655
EncodeString 648
EventHandled 655
FindComputer 649
FindFilesOrFolders 649
FindOnWeb 649
FindPeople 649
FindPrinter 649
Get_ASProvider 650
Get_ASSetting 650
Get_InWebFolder 650
Get_SearchAssistantDefault 650
Get_Searches 651
Get_ShellFeaturesEnabled 651
Get_Version 651
GetProperty 651
GetSearchAssistantURL 651
InWebFolder 655
ISearchAssistantOC Members 646
ISearchAssistantOC methods 647
ISearchAssistantOC properties 654
IsRestricted 652
NavigateToDefaultSearch 652
NETDetectNextNavigate 652
NotifySearchSettingsChanged 652
PutFindText 653
PutProperty 653
ResetNextMenu 653
SearchAssistantDefault 655
Searches 656
Set_ASProvider 653
Set_ASSetting 653
Set_EventHandled 654
SetDefaultSearchUrl 654
ShellFeaturesEnabled 656
Version 656
ISearchAssistantOC2 interface 656
about ISearchAssistantOC2 interface 656
Get_ShowFindPrinter 658
ISearchAssistantOC2 Members 657
ISearchAssistantOC2 methods 658
ISearchAssistantOC2 properties 658
ShowFindPrinter 658
ISearchAssistantOC2Disp interface 658
about ISearchAssistantOC2Disp interface 658
AddNextMenuItem 660
ASProvider 664
ASSetting 664
EncodeString 660
EventHandled 665
FindComputer 660
FindFilesOrFolders 661
FindOnWeb 661

FindPeople 661
FindPrinter 661
GetProperty 662
GetSearchAssistantURL 662
InWebFolder 665
ISearchAssistantOC2Disp Members 659
ISearchAssistantOC2Disp methods 659
ISearchAssistantOC2Disp properties 664
IsRestricted 662
NavigateToDefaultSearch 662
NETDetectNextNavigate 662
NotifySearchSettingsChanged 663
PutFindText 663
PutProperty 663
ResetNextMenu 663
SearchAssistantDefault 665
Searches 665
SetDefaultSearchUrl 664
ShellFeaturesEnabled 665
ShowFindPrinter 666
Version 666
ISearchAssistantOC3 interface 666
about ISearchAssistantOC3 interface 666
Get_SearchCompanionAvailable 668
Get_UseSearchCompanion 668
ISearchAssistantOC3 Members 666
ISearchAssistantOC3 methods 668
ISearchAssistantOC3 properties 668
SearchCompanionAvailable 669
Set_UseSearchCompanion 668
UseSearchCompanion 669
ISearchAssistantOC3Disp interface 669
about ISearchAssistantOC3Disp interface 669
AddNextMenuItem 671
ASProvider 675
ASSetting 675
EncodeString 671
EventHandled 675
FindComputer 671
FindFilesOrFolders 671
FindOnWeb 672
FindPeople 672
FindPrinter 672
GetProperty 672
GetSearchAssistantURL 672
InWebFolder 676
ISearchAssistantOC3Disp Members 669
ISearchAssistantOC3Disp methods 670
ISearchAssistantOC3Disp properties 674
IsRestricted 673
NavigateToDefaultSearch 673
NETDetectNextNavigate 673
NotifySearchSettingsChanged 673
PutFindText 674
PutProperty 674
ResetNextMenu 674
SearchAssistantDefault 676
SearchCompanionAvailable 676
Searches 676
SetDefaultSearchUrl 674
ShellFeaturesEnabled 676
ShowFindPrinter 677
UseSearchCompanion 677
Version 677
ISearchAssistantOCDisp interface 677
about ISearchAssistantOCDisp interface 677
AddNextMenuItem 679
ASProvider 683
ASSetting 683
EncodeString 679
EventHandled 683
FindComputer 679
FindFilesOrFolders 679
FindOnWeb 680
FindPeople 680
FindPrinter 680
GetProperty 680
GetSearchAssistantURL 681

InWebFolder 684
ISearchAssistantOCDisp Members 678
ISearchAssistantOCDisp methods 678
ISearchAssistantOCDisp properties 683
IsRestricted 681
NavigateToDefaultSearch 681
NETDetectNextNavigate 681
NotifySearchSettingsChanged 681
PutFindText 682
PutProperty 682
ResetNextMenu 682
SearchAssistantDefault 684
Searches 684
SetDefaultSearchUrl 682
ShellFeaturesEnabled 684
Version 684
ISearchDisp interface 685
about ISearchDisp interface 685
Id 685
ISearchDisp Members 685
ISearchDisp properties 685
Title 686
URL 686
ISearches interface 686
_NewEnum 687
about ISearches interface 686
Count 688
Default 688
Get_Count 687
Get_Default 688
ISearches Members 686
ISearches methods 687
ISearches properties 688
Item 688
ISearchesDisp interface 689
_NewEnum 689
about ISearchesDisp interface 689
Count 690
Default 690
ISearchesDisp Members 689
ISearchesDisp methods 689
ISearchesDisp properties 690
Item 690
IsEmpty
AnsiStringBase class 3194
UnicodeString class 3402
Variant class 3433
WideString class 3480
IsEmptyDoc
IXMLDocument interface 7652
TXMLDocument class 7367
IsEmptyRow
TValueListEditor class 4796
IsEnabled
TScreenTipItem class 331
ISendDataBlock interface 276
about ISendDataBlock interface 276
ISendDataBlock Members 277
ISendDataBlock methods 277
Send 277
IsEqual
IWebValueIsEqual interface 1119
TCustomAdapterBooleanField class 5596
IService interface 7155
about IService interface 7155
GetName 7157
GetPorts 7157
IService Members 7155
IService methods 7157
IService properties 7157
Name 7158
Ports 7158
SetName 7157
IServices interface 7158
about IServices interface 7158
Add 7160
GetService 7160
Insert 7161

IServices Members 7158
IServices methods 7160
IServices properties 7161
Service 7161
ISessionsService interface 1066
about ISessionsService interface 1066
CheckStatusChange 1067
Clear 1067
ISessionsService Members 1067
ISessionsService methods 1067
NewSession 1068
NotifyStatusChange 1068
ISessionsServiceConnectID interface 1068
about ISessionsServiceConnectID interface 1068
Connect 1069
ConnectActive 1069
ISessionsServiceConnectID Members 1069
ISessionsServiceConnectID methods 1069
ISessionsServicePersist interface 1070
about ISessionsServicePersist interface 1070
ISessionsServicePersist Members 1070
ISessionsServicePersist methods 1070
LoadFromFile 1070
LoadFromStream 1071
SaveToFile 1071
SaveToStream 1071
ISessionsServiceStoreID interface 1071
about ISessionsServiceStoreID interface 1071
ISessionsServiceStoreID Members 1072
ISessionsServiceStoreID methods 1072
ReadSessionID 1072
WriteSessionID 1072
ISetSubComponents interface 5889
about ISetSubComponents interface 5889
ISetSubComponents Members 5889
ISetSubComponents methods 5890
SetActions 5890
SetData 5890
ISetWebContentOptions interface 6091
about ISetWebContentOptions interface 6091
ISetWebContentOptions Members 6092
ISetWebContentOptions methods 6092
SetWebContentOptions 6092
ISetWebFieldValue interface 5890
about ISetWebFieldValue interface 5890
ISetWebFieldValue Members 5891
ISetWebFieldValue methods 5891
SetValue 5891
IsFieldInUse
IWebDataFields interface 1083
TAdapterControlGroup class 6350
TCustomAdapter class 5563
IsFunctionDefined
TAbstractControlRequirements class 6313
TAbstractFormRequirements class 6315
TControlRequirements class 6428
IsGroup
TDependency class 3067
IShellFavoritesNameSpace interface 691
about IShellFavoritesNameSpace interface 691
CreateSubscriptionForSelection 692
DeleteSubscriptionForSelection 692
Export 692
Get_SubscriptionsEnabled 693
Import 693
InvokeContextMenuCommand 693
IShellFavoritesNameSpace Members 691
IShellFavoritesNameSpace methods 692
IShellFavoritesNameSpace properties 695
MoveSelectionDown 693
MoveSelectionTo 694
MoveSelectionUp 694
NewFolder 694
ResetSort 694
SetRoot 694
SubscriptionsEnabled 695
Synchronize 695
IShellFavoritesNameSpaceDisp interface 695

about IShellFavoritesNameSpaceDisp interface 695
CreateSubscriptionForSelection 697
DeleteSubscriptionForSelection 697
Export 697
Import 697
InvokeContextMenuCommand 698
IShellFavoritesNameSpaceDisp Members 696
IShellFavoritesNameSpaceDisp methods 696
IShellFavoritesNameSpaceDisp properties 699
MoveSelectionDown 698
MoveSelectionTo 698
MoveSelectionUp 698
NewFolder 698
ResetSort 699
SetRoot 699
SubscriptionsEnabled 700
Synchronize 699
IShellNameSpace interface 700
about IShellNameSpace interface 700
Columns 707
CountViewTypes 707
Depth 707
EnumOptions 707
Expand 702
Flags 708
Get_Columns 702
Get_CountViewTypes 702
Get_Depth 702
Get_EnumOptions 703
Get_Flags 703
Get_Mode 703
Get_Root 703
Get_SelectedItem 704
Get_TVFlags 704
IShellNameSpace Members 700
IShellNameSpace methods 701
IShellNameSpace properties 706
Mode 708
Root 708
SelectedItem 708
SelectedItems 704
Set_Columns 704
Set_Depth 704
Set_EnumOptions 705
Set_Flags 705
Set_Mode 705
Set_Root 705
Set_SelectedItem 706
Set_TVFlags 706
SetViewType 706
TVFlags 709
UnselectAll 706
IShellNameSpaceDisp interface 709
about IShellNameSpaceDisp interface 709
Columns 714
CountViewTypes 714
CreateSubscriptionForSelection 710
DeleteSubscriptionForSelection 711
Depth 715
EnumOptions 715
Expand 711
Export 711
Flags 715
Import 711
InvokeContextMenuCommand 711
IShellNameSpaceDisp Members 709
IShellNameSpaceDisp methods 710
IShellNameSpaceDisp properties 714
Mode 715
MoveSelectionDown 712
MoveSelectionTo 712
MoveSelectionUp 712
NewFolder 712
ResetSort 713
Root 716
SelectedItem 716
SelectedItems 713
SetRoot 713

SetViewType 713
SubscriptionsEnabled 716
Synchronize 713
TVFlags 716
UnselectAll 714
IShellUIHelper interface 716
about IShellUIHelper interface 716
AddChannel 718
AddDesktopComponent 718
AddFavorite 718
AutoCompleteAttach 718
AutoCompleteSaveForm 719
AutoScan 719
ImportExportFavorites 719
IShellUIHelper Members 717
IShellUIHelper methods 717
IsSubscribed 719
NavigateAndFind 719
RefreshOfflineDesktop 720
ResetFirstBootMode 720
ResetSafeMode 720
ShowBrowserUI 720
IShellUIHelperDisp interface 721
about IShellUIHelperDisp interface 721
AddChannel 722
AddDesktopComponent 722
AddFavorite 722
AutoCompleteAttach 722
AutoCompleteSaveForm 723
AutoScan 723
ImportExportFavorites 723
IShellUIHelperDisp Members 721
IShellUIHelperDisp methods 721
IsSubscribed 723
NavigateAndFind 724
RefreshOfflineDesktop 724
ResetFirstBootMode 724
ResetSafeMode 724
ShowBrowserUI 724
IShellWindows interface 725
_NewEnum 726
about IShellWindows interface 725
Count 729
FindWindowSW 726
Get_Count 726
IShellWindows Members 725
IShellWindows methods 726
IShellWindows properties 728
Item 727
OnActivated 727
OnCreated 727
OnNavigate 727
ProcessAttachDetach 727
Register 728
RegisterPending 728
Revoke 728
IShellWindowsDisp interface 729
_NewEnum 730
about IShellWindowsDisp interface 729
Count 732
FindWindowSW 730
IShellWindowsDisp Members 729
IShellWindowsDisp methods 730
IShellWindowsDisp properties 732
Item 730
OnActivated 731
OnCreated 731
OnNavigate 731
ProcessAttachDetach 731
Register 731
RegisterPending 732
Revoke 732
IsImageIndexLinked
TPushButtonActionLink class 2812
IsLeadByte
AnsiStringBase class 3195
IsLeadSurrogate
UnicodeString class 3403

IsLoggedIn
TCustomPageDispatcher class 6194
IsModified
TConnectionData class 1520
IsMultiValueField
TAdapterUpdateField class 5430
IsNull
Variant class 3434
IsPathDelimiter
AnsiStringBase class 3195
UnicodeString class 3403
WideString class 3481
IsPortTypeHTTPBound
TWSDLItems class 7244
IsRestricted
ISearchAssistantOC interface 652
ISearchAssistantOC2Disp interface 662
ISearchAssistantOC3Disp interface 673
ISearchAssistantOCDisp interface 681
IsSequenced
TCustomSQLDataSet class 1536
IsSingleByte
TEncoding class 3986
IsStandardEncoding
TEncoding class 3984
IsSubscribed
IShellUIHelper interface 719
IShellUIHelperDisp interface 723
TShellUIHelper class 493
IsTextElement
IXMLNode interface 7692
IsTrailByte
AnsiStringBase class 3195
IsTrailSurrogate
UnicodeString class 3403
IStringTest interface 1334
about IStringTest interface 1334
Append 1335
IStringTest Members 1334
IStringTest methods 1335
UpCase 1335
iSubType
TFLDDesc class 1576
IsVariableDefined
TAbstractControlRequirements class 6313
TAbstractFormRequirements class 6316
TControlRequirements class 6429
item
IDOMNamedNodeMap interface 7557
IDOMNodeList interface 7582
Item
ISearches interface 688
ISearchesDisp interface 690
IShellWindows interface 727
IShellWindowsDisp interface 730
TShellWindows class 500, 501
ItemAtPos
TCustomListBox class 2411
TTabSet class 4476
ItemCount
TBaseAdapterAction class 5475
TCustomCombo class 2307
TGeneratedFunctions class 6508
TWebModuleFactoryList class 6625
ItemHeight
TComboBox class 2207
TCustomCombo class 2307
TCustomListBox class 2428
TListBox class 2680
ItemIndex
TBaseAdapterAction class 5476
TComboBox class 2208
TSocketForm class 181
ItemInterface
TXMLNodeCollection class 7458
ItemLength
THTMLItem class 6665
ItemNS

TXMLNodeCollection class 7458
ItemProps
TValueListEditor class 4821
TValueListStrings class 4890
ItemRect
TCustomListBox class 2413
TTabSet class 4476
Items
IWebFormComponentContainer interface 1086
TComboBox class 2210
TCustomCombo class 2308
TCustomListBox class 2429
TDependencies class 3065
THTMLItems class 6667
TLineNumberMap class 6671
TListBox class 2680
TNotifyList class 972
TScreenTipsCollection class 337
TScriptEnginesList class 6677
TScriptIncludeFiles class 6695
TScriptItems class 6701
TSessionItem class 382
TWebModuleFactoryList class 6626
TWebUserItems class 6796
ItemTag
TXMLNodeCollection class 7459
ItemValue
TSessions class 391
ItemWidth
TTabSet class 4477
itEnd enumeration member 5902
IterateImports
TWSDLItems class 7244
IteratorIndex
IIteratorIndex interface 1054
itNext enumeration member 5902
ITopLevelWebComponent interface 6092
about ITopLevelWebComponent interface 6092
ITransport interface 278
about ITransport interface 278
Connected 281
GetConnected 279
GetWaitEvent 279
ITransport Members 278
ITransport methods 279
ITransport properties 281
Receive 280
Send 280
SetConnected 280
itStart enumeration member 5902
ITypes interface 7161
about ITypes interface 7161
GetSchemaDefs 7163
ITypes Members 7162
ITypes methods 7163
ITypes properties 7163
SchemaDefs 7163
iUnits1
SPParamDesc class 1514
TFLDDesc class 1577
iUnits2
SPParamDesc class 1515
TFLDDesc class 1577
IUnknown*
Variant class 3462
IUpdateValue interface 5891
about IUpdateValue interface 5891
IUpdateValue Members 5891
IUpdateValue methods 5892
UpdateValue 5892
IValidateValue interface 5892
about IValidateValue interface 5892
IValidateValue Members 5892
IValidateValue methods 5893
ValidateValue 5893
IValuesListAdapter interface 1072
about IValuesListAdapter interface 1072
GetListImage 1074

GetListImageOfValue 1074
GetListName 1075
GetListNameOfValue 1075
GetListValue 1076
IValuesListAdapter Members 1073
IValuesListAdapter methods 1074
IVarInstanceReference interface 5022
about IVarInstanceReference interface 5022
GetInstance 5023
IVarInstanceReference Members 5022
IVarInstanceReference methods 5023
IVarInvokeable interface 5023
about IVarInvokeable interface 5023
DoFunction 5024
DoProcedure 5025
GetProperty 5025
IVarInvokeable Members 5024
IVarInvokeable methods 5024
SetProperty 5026
IWebActionsList interface 1076
about IWebActionsList interface 1076
FindAction 1077
GetActionsList 1077
GetVisibleActions 1077
IsActionInUse 1078
IWebActionsList Members 1076
IWebActionsList methods 1077
IWebActionsList properties 1078
VisibleActions 1078
IWebActionsSubcomponent interface 5893
about IWebActionsSubcomponent interface 5893
IWebApplicationInfo interface 1078
about IWebApplicationInfo interface 1078
ApplicationAdapter 1080
ApplicationTitle 1080
GetApplicationAdapter 1079
GetApplicationTitle 1080
IWebApplicationInfo Members 1079
IWebApplicationInfo methods 1079
IWebApplicationInfo properties 1080
IWebBrowser interface 733
about IWebBrowser interface 733
Application 745
Busy 745
Container 746
Document 746
Get_Application 735
Get_Busy 735
Get_Container 735
Get_Document 736
Get_Height 736
Get_Left 736
Get_LocationName 737
Get_LocationURL 737
Get_Parent 737
Get_Top 738
Get_TopLevelContainer 738
Get_type_ 738
Get_Width 739
GoBack 739
GoForward 739
GoHome 740
GoSearch 740
Height 746
IWebBrowser Members 733
IWebBrowser methods 734
IWebBrowser properties 744
Left 747
LocationName 747
LocationURL 747
Navigate 741
Parent 748
Refresh 742
Refresh2 742
Set_Height 743
Set_Left 743
Set_Top 743
Set_Width 744

Stop 744
Top 748
TopLevelContainer 748
type_ 749
Width 749
IWebBrowser2 interface 749
about IWebBrowser2 interface 749
AddressBar 760
ExecWB 752
Get_AddressBar 753
Get_Offline 753
Get_ReadyState 754
Get_RegisterAsBrowser 754
Get_RegisterAsDropTarget 754
Get_Resizable 754
Get_Silent 755
Get_TheaterMode 755
IWebBrowser2 Members 750
IWebBrowser2 methods 752
IWebBrowser2 properties 759
Navigate2 755
Offline 760
QueryStatusWB 757
ReadyState 760
RegisterAsBrowser 761
RegisterAsDropTarget 761
Resizable 762
Set_AddressBar 757
Set_Offline 757
Set_RegisterAsBrowser 758
Set_RegisterAsDropTarget 758
Set_Resizable 758
Set_Silent 759
Set_TheaterMode 759
ShowBrowserBar 759
Silent 762
TheaterMode 762
IWebBrowser2Disp interface 763
about IWebBrowser2Disp interface 763
AddressBar 769
Application 769
Busy 769
ClientToWindow 764
Container 769
Document 770
ExecWB 765
FullName 770
FullScreen 770
GetProperty 765
GoBack 765
GoForward 765
GoHome 766
GoSearch 766
Height 770
HWnd 771
IWebBrowser2Disp Members 763
IWebBrowser2Disp methods 764
IWebBrowser2Disp properties 768
Left 771
LocationName 771
LocationURL 771
MenuBar 771
Name 772
Navigate 766
Navigate2 766
Offline 772
Parent 772
Path 772
PutProperty 767
QueryStatusWB 767
Quit 767
ReadyState 773
Refresh 767
Refresh2 767
RegisterAsBrowser 773
RegisterAsDropTarget 773
Resizable 773
ShowBrowserBar 768

Silent 773
StatusBar 774
StatusText 774
Stop 768
TheaterMode 774
ToolBar 774
Top 775
TopLevelContainer 775
type_ 775
Visible 775
Width 775
IWebBrowserApp interface 776
about IWebBrowserApp interface 776
ClientToWindow 778
FullName 783
FullScreen 783
Get_FullName 778
Get_FullScreen 779
Get_HWnd 779
Get_MenuBar 779
Get_Name 779
Get_Path 779
Get_StatusBar 780
Get_StatusText 780
Get_ToolBar 780
Get_Visible 780
GetProperty 781
HWnd 783
IWebBrowserApp Members 776
IWebBrowserApp methods 778
IWebBrowserApp properties 782
MenuBar 783
Name 784
Path 784
PutProperty 781
Quit 781
Set_FullScreen 781
Set_MenuBar 781
Set_StatusBar 782
Set_StatusText 782
Set_ToolBar 782
Set_Visible 782
StatusBar 784
StatusText 784
ToolBar 785
Visible 785
IWebBrowserAppDisp interface 785
about IWebBrowserAppDisp interface 785
Application 790
Busy 790
ClientToWindow 787
Container 790
Document 790
FullName 791
FullScreen 791
GetProperty 787
GoBack 787
GoForward 787
GoHome 787
GoSearch 788
Height 791
HWnd 791
IWebBrowserAppDisp Members 785
IWebBrowserAppDisp methods 786
IWebBrowserAppDisp properties 789
Left 792
LocationName 792
LocationURL 792
MenuBar 792
Name 792
Navigate 788
Parent 793
Path 793
PutProperty 788
Quit 788
Refresh 789
Refresh2 789
StatusBar 793

StatusText 793
Stop 789
ToolBar 794
Top 794
TopLevelContainer 794
type_ 794
Visible 794
Width 795
IWebBrowserDisp interface 795
about IWebBrowserDisp interface 795
Application 798
Busy 799
Container 799
Document 799
GoBack 796
GoForward 796
GoHome 797
GoSearch 797
Height 799
IWebBrowserDisp Members 795
IWebBrowserDisp methods 796
IWebBrowserDisp properties 798
Left 799
LocationName 800
LocationURL 800
Navigate 797
Parent 800
Refresh 797
Refresh2 797
Stop 798
Top 800
TopLevelContainer 801
type_ 801
Width 801
IWebComponent interface 6093
about IWebComponent interface 6093
ComponentList 6096
Container 6096
GetContainer 6094
GetIndex 6094
Index 6097
IWebComponent Members 6093
IWebComponent methods 6094
IWebComponent properties 6096
SetComponentList 6095
SetContainer 6095
SetIndex 6095
IWebComponentCommands interface 6097
about IWebComponentCommands interface 6097
ExecuteVerb 6098
GetVerb 6098
GetVerbCount 6098
IWebComponentCommands Members 6097
IWebComponentCommands methods 6097
IWebComponentContainer interface 6098
about IWebComponentContainer interface 6098
ComponentCount 6100
Components 6100
GetComponent 6099
GetComponentCount 6099
IWebComponentContainer Members 6099
IWebComponentContainer methods 6099
IWebComponentContainer properties 6100
IWebComponentEditor interface 6100
about IWebComponentEditor interface 6100
CanAddClass 6101
IWebComponentEditor Members 6101
IWebComponentEditor methods 6101
IWebComponentEditorHelper interface 6102
about IWebComponentEditorHelper interface 6102
CanAddClassHelper 6102
IWebComponentEditorHelper Members 6102
IWebComponentEditorHelper methods 6102
IWebContainerSubComponent interface 1081
about IWebContainerSubComponent interface 1081
IWebContent interface 6103
about IWebContent interface 6103
Content 6104

IWebContent Members 6103
IWebContent methods 6103
IWebDataFields interface 1081
about IWebDataFields interface 1081
FindField 1082
GetFieldsList 1082
GetVisibleFields 1083
IsFieldInUse 1083
IWebDataFields Members 1082
IWebDataFields methods 1082
IWebDataFields properties 1083
VisibleFields 1083
IWebEnabled interface 1083
about IWebEnabled interface 1083
IWebEnabled Members 1084
IWebEnabled methods 1084
WebEnabled 1084
IWebEndUser interface 5988
about IWebEndUser interface 5988
DisplayName 5994
EndUserAdapter 5994
GetDisplayName 5989
GetEndUserAdapter 5990
GetLoggedIn 5990
GetLoginFormAction 5990
GetLogoutAction 5991
HasExecuteRights 5991
HasModifyRights 5992
HasViewRights 5992
IWebEndUser Members 5989
IWebEndUser methods 5989
IWebEndUser properties 5994
LoggedIn 5994
LogIn 5993
LoginFormAction 5995
LogoutAction 5996
ProduceLoginPage 5993
IWebFieldsSubcomponent interface 5893
about IWebFieldsSubcomponent interface 5893
IWebFormComponentContainer interface 1084
about IWebFormComponentContainer interface 1084
Count 1086
GetCount 1085
GetItem 1085
Items 1086
IWebFormComponentContainer Members 1085
IWebFormComponentContainer methods 1085
IWebFormComponentContainer properties 1086
IWebGetActionName interface 1086
about IWebGetActionName interface 1086
ActionName 1087
GetActionName 1087
IWebGetActionName Members 1087
IWebGetActionName methods 1087
IWebGetActionName properties 1087
IWebGetDisplayLabel interface 1088
about IWebGetDisplayLabel interface 1088
GetDisplayLabel 1088
IWebGetDisplayLabel Members 1088
IWebGetDisplayLabel methods 1088
IWebGetDisplayWidth interface 1089
about IWebGetDisplayWidth interface 1089
GetDisplayWidth 1089
IWebGetDisplayWidth Members 1089
IWebGetDisplayWidth methods 1089
IWebGetFieldName interface 1090
about IWebGetFieldName interface 1090
FieldName 1091
GetFieldName 1090
IWebGetFieldName Members 1090
IWebGetFieldName methods 1090
IWebGetFieldName properties 1091
IWebGetFieldPrevValue interface 5894
about IWebGetFieldPrevValue interface 5894
GetNeedPrevValue 5895
GetPrevValue 5895
IWebGetFieldPrevValue Members 5894
IWebGetFieldPrevValue methods 5894

IWebGetFieldPrevValue properties 5895
NeedPrevValue 5895
PrevValue 5895
IWebGetFieldValue interface 1091
about IWebGetFieldValue interface 1091
GetValue 1092
IWebGetFieldValue Members 1091
IWebGetFieldValue methods 1092
IWebGetFieldValue properties 1092
Value 1092
IWebGetFieldValues interface 1092
about IWebGetFieldValues interface 1092
EndValueIterator 1093
GetCurrentValue 1094
HasValue 1094
IWebGetFieldValues Members 1093
IWebGetFieldValues methods 1093
NextValueIteration 1094
StartValueIterator 1095
IWebGetMaxLength interface 1095
about IWebGetMaxLength interface 1095
GetMaxLength 1096
IWebGetMaxLength Members 1095
IWebGetMaxLength methods 1096
IWebGetSessionIDs interface 1096
about IWebGetSessionIDs interface 1096
GetSessionIDs 1097
IWebGetSessionIDs Members 1096
IWebGetSessionIDs methods 1097
IWebImageHREF interface 1097
about IWebImageHREF interface 1097
IWebImageHREF Members 1098
IWebImageHREF methods 1098
WebImageHREF 1098
IWebInputName interface 1098
about IWebInputName interface 1098
GetInputName 1099
InputName 1100
IWebInputName Members 1099
IWebInputName methods 1099
IWebInputName properties 1099
IWebIsDefaultAction interface 1100
about IWebIsDefaultAction interface 1100
IsDefaultAction 1101
IWebIsDefaultAction Members 1100
IWebIsDefaultAction methods 1100
IWebIsDefaultField interface 1101
about IWebIsDefaultField interface 1101
IsDefaultField 1102
IWebIsDefaultField Members 1101
IWebIsDefaultField methods 1101
IWebNode interface 6616
about IWebNode interface 6616
BeforeExecute 6617
Execute 6618
Execute method 6618
GetMimeBoundary 6619
IWebNode Members 6617
IWebNode methods 6617
IWebNode properties 6619
MimeBoundary 6619
SetMimeBoundary 6619
IWebPageInfo interface 5996
about IWebPageInfo interface 5996
GetHasViewAccess 5997
GetLoginRequired 5998
GetPageDescription 5998
GetPageHREF 5998
GetPageIsPublished 5999
GetPageName 5999
GetPageTitle 5999
HasViewAccess 6000
IWebPageInfo Members 5997
IWebPageInfo methods 5997
IWebPageInfo properties 5999
LoginRequired 6000
PageDescription 6000
PageHREF 6001

PageIsPublished 6001
PageName 6002
PageTitle 6002
IWebServExp interface 6735
about IWebServExp interface 6735
FindOrAddSchema 6736
GenerateXMLSchema 6736
IWebServExp Members 6735
IWebServExp methods 6736
IWebServExpAccess interface 6737
about IWebServExpAccess interface 6737
GetWebServExp 6737
IWebServExpAccess Members 6737
IWebServExpAccess methods 6737
IWebSession interface 1102
about IWebSession interface 1102
IWebSession Members 1102
IWebSession methods 1103
Terminate 1103
IWebSessionAccess interface 6257
about IWebSessionAccess interface 6257
GetSessionsService 6258
GetWebSession 6258
IWebSessionAccess Members 6257
IWebSessionAccess methods 6257
IWebSessionAttributes interface 1103
about IWebSessionAttributes interface 1103
ExpirationTime 1106
Expired 1107
GetExpirationTime 1104
GetExpired 1105
GetIsActive 1105
GetStartTime 1105
GetTerminated 1105
GetTimeoutMinutes 1105
GetTouchTime 1106
IsActive 1107
IWebSessionAttributes Members 1104
IWebSessionAttributes methods 1104
IWebSessionAttributes properties 1106
SetTimeoutMinutes 1106
StartTime 1107
Terminated 1107
TimeoutMinutes 1107
TouchTime 1108
IWebSessionID interface 1108
about IWebSessionID interface 1108
GetID 1109
ID 1109
IWebSessionID Members 1108
IWebSessionID methods 1108
IWebSessionID properties 1109
IWebSessionIDs interface 1109
about IWebSessionIDs interface 1109
GetID 1110
GetIDCount 1111
IDCount 1111
IDs 1111
IWebSessionIDs Members 1110
IWebSessionIDs methods 1110
IWebSessionIDs properties 1111
IWebSessionValues interface 1112
about IWebSessionValues interface 1112
GetItemCount 1113
GetItems 1113
GetValue 1113
IWebSessionValues Members 1112
IWebSessionValues methods 1112
IWebSessionValues properties 1113
SetValue 1113
Values 1114
IWebSetActionName interface 1114
about IWebSetActionName interface 1114
IWebSetActionName Members 1114
IWebSetActionName methods 1114
SetActionName 1115
IWebSetFieldName interface 1115
about IWebSetFieldName interface 1115

IWebSetFieldName Members 1115
IWebSetFieldName methods 1115
SetFieldName 1116
IWebUserList interface 1116
about IWebUserList interface 1116
CheckAccessRights 1117
IWebUserList Members 1116
IWebUserList methods 1117
ValidateUser 1117
IWebValueIsEqual interface 1118
about IWebValueIsEqual interface 1118
IsEqual 1119
IWebValueIsEqual Members 1118
IWebValueIsEqual methods 1118
IWebVariablesContainerName interface 6104
about IWebVariablesContainerName interface 6104
ContainerName 6105
GetContainerName 6105
IWebVariablesContainerName Members 6104
IWebVariablesContainerName methods 6105
IWebVariablesContainerName properties 6105
IWideStringsAdapter interface 6859
about IWideStringsAdapter interface 6859
IWideStringsAdapter Members 6859
IWideStringsAdapter methods 6859
ReferenceStrings 6860
ReleaseStrings 6860
IWinHelpTester interface 6894
about IWinHelpTester interface 6894
CanShowALink 6895
CanShowContext 6895
CanShowTopic 6895
GetDefaultHelpFile 6895
GetHelpPath 6896
GetHelpStrings 6896
IWinHelpTester Members 6894
IWinHelpTester methods 6894
IWSDLDocument interface 7164
about IWSDLDocument interface 7164
Definition 7167
GetDefinition 7166
IWSDLDocument Members 7164
IWSDLDocument methods 7166
IWSDLDocument properties 7167
SetDefinition 7166
IWSDLItems interface 7256
about IWSDLItems interface 7256
GetWSDLDoc 7257
GetWSDLItems 7257
IWSDLItems Members 7256
IWSDLItems methods 7256
Load 7257
IWSDLPublish interface 7295
about IWSDLPublish interface 7295
GetPortTypeList 7296
GetTypeSystemsList 7296
GetWSDLForPortType 7296
GetXSDForTypeSystem 7296
IWSDLPublish Members 7295
IWSDLPublish methods 7295
IXMLDocument interface 7640
about IXMLDocument interface 7640
Active 7661
AddChild 7643
AddChild method 7643
AsyncLoadState 7662
ChildNodes 7663
CreateElement 7644
CreateNode 7645
DocumentElement 7663
DOMDocument 7663
Encoding 7664
FileName 7664
GeneratePrefix 7646
GetActive 7646
GetAsyncLoadState 7647
GetChildNodes 7647
GetDocBinding 7647

GetDocumentElement 7648
GetDocumentNode 7648
GetDOMDocument 7648
GetEncoding 7649
GetFileName 7649
GetModified 7649
GetNodeIndentStr 7650
GetOptions 7650
GetParseOptions 7650
GetSchemaRef 7650
GetStandAlone 7651
GetVersion 7651
GetXML 7651
IsEmptyDoc 7652
IXMLDocument Members 7641
IXMLDocument methods 7642
IXMLDocument properties 7661
LoadFromFile 7652
LoadFromStream 7653
LoadFromXML 7653
LoadFromXML method 7653
Modified 7665
Node 7665
NodeIndentStr 7666
Options 7666
ParseOptions 7667
Refresh 7654
RegisterDocBinding 7654
Resync 7655
SaveToFile 7655
SaveToStream 7656
SaveToXML 7656
SaveToXML method 7656
SchemaRef 7667
SetActive 7657
SetDocumentElement 7657
SetDOMDocument 7658
SetEncoding 7658
SetFileName 7658
SetNodeIndentStr 7659
SetOnAsyncLoad 7659
SetOptions 7659
SetParseOptions 7660
SetStandAlone 7660
SetVersion 7660
SetXML 7661
StandAlone 7668
Version 7668
XML 7668
IXMLDocumentAccess interface 7478
about IXMLDocumentAccess interface 7478
DocumentObject 7479
DOMPersist 7480
GetDocumentObject 7479
GetDOMPersist 7479
IXMLDocumentAccess Members 7478
IXMLDocumentAccess methods 7478
IXMLDocumentAccess properties 7479
IXMLNode interface 7669
about IXMLNode interface 7669
AddChild 7672
AddChild method 7672
AttributeNodes 7688
Attributes 7688
ChildNodes 7689
ChildValues 7690
CloneNode 7673
Collection 7690
DeclareNamespace 7674
DOMNode 7691
FindNamespaceDecl 7674
FindNamespaceURI 7674
GetAttribute 7675
GetAttributeNodes 7675
GetAttributeNS 7676
GetChildNodes 7676
GetChildValue 7676
GetCollection 7677

GetDOMNode 7677
GetHasChildNodes 7678
GetIsTextElement 7678
GetLocalName 7678
GetNamespaceURI 7679
GetNodeName 7679
GetNodeType 7679
GetNodeValue 7680
GetOwnerDocument 7680
GetParentNode 7680
GetPrefix 7680
GetReadOnly 7681
GetText 7681
GetXML 7681
HasAttribute 7682
HasAttribute method 7682
HasChildNodes 7691
IsTextElement 7692
IXMLNode Members 7670
IXMLNode methods 7671
IXMLNode properties 7687
LocalName 7692
NamespaceURI 7693
NextSibling 7682
NodeName 7693
NodeType 7694
NodeValue 7695
Normalize 7683
OwnerDocument 7695
ParentNode 7695
Prefix 7696
PreviousSibling 7683
ReadOnly 7696
Resync 7684
SetAttribute 7684
SetAttributeNS 7684
SetChildValue 7685
SetNodeValue 7685
SetReadOnly 7686
SetText 7686
Text 7697
TransformNode 7687
TransformNode method 7686
XML 7697
IXMLNodeAccess interface 7480
about IXMLNodeAccess interface 7480
AddChild 7482
CheckTextNode 7483
ChildNodeClasses 7487
ClearDocumentRef 7483
CreateAttributeNode 7483
CreateChildNode 7483
CreateCollection 7483
DOMElement 7484
FindHostedNode 7484
GetChildNodeClasses 7484
GetHostNode 7484
GetNodeObject 7485
HasChildNode 7485
HasChildNode method 7485
HostNode 7487
InternalAddChild 7485
IXMLNodeAccess Members 7480
IXMLNodeAccess methods 7482
IXMLNodeAccess properties 7487
NestingLevel 7486
RegisterChildNode 7486
RegisterChildNodes 7486
SetCollection 7486
SetParentNode 7487
IXMLNodeCollection interface 7698
about IXMLNodeCollection interface 7698
Clear 7700
Count 7702
Delete 7700
GetCount 7701
GetNode 7701
IXMLNodeCollection Members 7698

IXMLNodeCollection methods 7700
IXMLNodeCollection properties 7702
Nodes 7703
Remove 7702
IXMLNodeList interface 7703
about IXMLNodeList interface 7703
Add 7705
BeginUpdate 7705
Clear 7706
Count 7714
Delete 7707
Delete method 7706
EndUpdate 7707
FindNode 7708
FindNode method 7708
FindSibling 7709
First 7709
Get 7710
GetCount 7710
GetNode 7710
GetUpdateCount 7711
IndexOf 7711
IndexOf method 7711
Insert 7712
IXMLNodeList Members 7704
IXMLNodeList methods 7705
IXMLNodeList properties 7714
Last 7712
Nodes 7714
Remove 7713
ReplaceNode 7713
UpdateCount 7715
IXMLSchemaDefs interface 7168
about IXMLSchemaDefs interface 7168
Add 7170
GetXMLSchemaDef 7170
IXMLSchemaDefs Members 7168
IXMLSchemaDefs methods 7169
IXMLSchemaDefs properties 7170
SchemaDef 7170

K

KeepConnection
TSQLConnection class 1614
KeepInCache
TServerClientThread class 103
keyAdd enumeration member 4893
keyDelete enumeration member 4893
KeyDesc
TItemProp class 4783
KeyDown
TTabbedNotebook class 4410
TValueListEditor class 4797
keyEdit enumeration member 4893
KeyIsValid
TValueListStrings class 4890
KeyOptions
TValueListEditor class 4821
KeyPress
TCustomComboBox class 2329
TCustomListBox class 2415
TCustomMemo class 2448
Keys
TValueListEditor class 4822
keyUnique enumeration member 4893
Kind
TScrollBar class 2897

L

Label1
TSocketForm class 164
Label4
TSocketForm class 164
Label5
TSocketForm class 164
Label7
TSocketForm class 165
LabelAttributes

TLayoutAttributes class 6013
LabelPosition
TLayoutAttributes class 6014
LargeChange
TScrollBar class 2898
Last
IXMLNodeList interface 7712
TXMLNodeList class 7473
LastChar
AnsiStringBase class 3196
UnicodeString class 3404
WideString class 3481
lastChild
IDOMNode interface 7569
LastDelimiter
AnsiStringBase class 3196
AnsiStringT template 3224
UnicodeString class 3404
WideString class 3481
LastError
THandleObject class 3142
Layout
TCustomLabel class 2396
TLabel class 2625
LayoutButton
ILayoutWebContent interface 6085
TLayoutWebContent class 6020
LayoutField
ILayoutWebContent interface 6085
TLayoutWebContent class 6020
LayoutLabelAndField
ILayoutWebContent interface 6086
TLayoutWebContent class 6020
LayoutTable
ILayoutWebContent interface 6086
TLayoutWebContent class 6021
lbOwnerDrawFixed enumeration member 3003
lbOwnerDrawVariable enumeration member 3003
lbStandard enumeration member 3003
lbVirtual enumeration member 3003
lbVirtualOwnerDraw enumeration member 3003
Leave
TCriticalSection class 3129
TCriticalSectionHelper class helper 3133
Left
IWebBrowser interface 747
IWebBrowser2Disp interface 771
IWebBrowserAppDisp interface 792
IWebBrowserDisp interface 799
TImageMargins class 2605
TInternetExplorer class 451
LeftPromotion
TCustomVariantType class 5000
Len
SmallStringBase template 3311
length
IDOMCharacterData interface 7529
IDOMNamedNodeMap interface 7557
IDOMNodeList interface 7582
Length
AnsiStringBase class 3196
DynamicArray template 3270
TStringBuilder class 4022
UnicodeString class 3404
WideString class 3481
LibraryName
TSQLConnection class 1614
line
IDOMParseError interface 7590
Line
EDOMParseError class 7512
LineBreak
TWideStrings class 6850
linePos
IDOMParseError interface 7590
LinePos
EDOMParseError class 7512
Lines

TCustomMemo class 2451
TMemo class 2758
LinkedActionLists
TCustomScreenTipsManager class 314
TScreenTipsManager class 340
LinkPopupToManager
TCustomScreenTipsPopup class 319
List
TXMLNodeCollection class 7459
TXMLNodeList class 7475
Listen
TCustomTcpServer class 1410
TCustomWinSocket class 73
TServerWinSocket class 127
Listening
TCustomTcpServer class 1412
ListHandle
TCustomCombo class 2310
ListHiddenFields
TCustomAdapter class 5563
ListHiddenRecordFields
TCustomAdapter class 5564
ListLanguageNames
TAbstractScriptEngine class 6651
TScriptEnginesList class 6676
ListRequestHandlers
TCustomWebAppComponents class 6210
ListWndProc
TCustomCombo class 2300
load
IDOMPersist interface 7596
Load
IWSDLItems interface 7257
TWSDLItems class 7245
LoadData
TWSDLDocument class 7069
TXMLDocument class 7368
Loaded
TAbstractSocket class 18
TBaseSocket class 1374
TCustomButton class 2264
TCustomCombo class 2300
TCustomLabel class 2387
TCustomMemo class 2448
TCustomScreenTipsManager class 310
TCustomSQLDataSet class 1537
TCustomStaticText class 2459
TSimpleDataSet class 901
TSQLConnection class 1600
TTabbedNotebook class 4411
TXMLDocument class 7368
LoadFile
TScriptIncludeFile class 6693
LoadFromFile
ISessionsServicePersist interface 1070
IXMLDocument interface 7652
TCustomSessionsService class 6751
TCustomWebUserList class 6783
TSessions class 391
TSQLMonitor class 1646
TWideStrings class 6842
TXMLDocument class 7368
loadFromStream
IDOMPersist interface 7597
LoadFromStream
ISessionsServicePersist interface 1071
IXMLDocument interface 7653
TCustomSessionsService class 6751
TSessions class 391
TWideStrings class 6842
TXMLDocument class 7369
LoadFromXML
IXMLDocument interface 7653
TXMLData class 7799
TXMLDocument class 7370
LoadGroup
TService class 3080
LoadParamsFromIniFile

TSQLConnection class 1600
LoadParamsOnConnect
TSQLConnection class 1615
LoadRecreateItems
TCustomListBox class 2415
LoadStr
AnsiStringBase class 3197
AnsiStringT template 3225
UnicodeString class 3405
LoadString
AnsiStringBase class 3197
AnsiStringT template 3225
UnicodeString class 3405
loadxml
IDOMPersist interface 7597
LocalAddress
TCustomWinSocket class 85
LocalDomainName
TIpSocket class 1419
LocaleCode
TSQLConnection class 1615
LocaleID
TLanguages class 3990
LocalHost
TCustomWinSocket class 85
TIpSocket class 1423
TTcpServer class 1466
TUdpSocket class 1482
LocalHostAddr
TIpSocket class 1420
LocalHostName
TIpSocket class 1420
localName
IDOMNode interface 7570
LocalName
IXMLNode interface 7692
LocalPort
TCustomWinSocket class 86
TIpSocket class 1424
TTcpServer class 1466
TUdpSocket class 1483
Locate
TCustomSQLDataSet class 1538
LocateFileService
TCustomWebAppComponents class 6213
TWebAppComponents class 6247
LocateRecord
TCustomSQLDataSet class 1538
Location
IImport interface 7113
LocationName
IWebBrowser interface 747
IWebBrowser2Disp interface 771
IWebBrowserAppDisp interface 792
IWebBrowserDisp interface 800
TInternetExplorer class 451
TWebBrowser class 529
TWebBrowser_V1 class 584
LocationURL
IWebBrowser interface 747
IWebBrowser2Disp interface 771
IWebBrowserAppDisp interface 792
IWebBrowserDisp interface 800
TInternetExplorer class 452
TWebBrowser class 530
TWebBrowser_V1 class 585
Lock
TCustomWinSocket class 74
TSoapDataModule class 1221
LockNotify
TWebComponentList class 6031
LockObject
TCustomDataBlockInterpreter class 196
TDataBlockInterpreter class 213
LoggedIn
IWebEndUser interface 5994
LogIn
IWebEndUser interface 5993

TBaseEndUserAdapter class 5515
LoginFormAction
IWebEndUser interface 5995
LoginPage
TBaseEndUserAdapter class 5516
TEndUserAdapter class 5771
TEndUserSessionAdapter class 5800
LoginPrompt
TSQLConnection class 1616
LoginRequired
IWebPageInfo interface 6000
LogMessage
TEventLogger class 3070
TService class 3076
LogoutAction
IWebEndUser interface 5996
loInvariantLocale enumeration member 4328
long
Variant class 3462
long*
Variant class 3462
LongName
TScriptFile class 6686
Lookup
TCustomSQLDataSet class 1538
LookupHostAddr
TIpSocket class 1420
LookupHostName
TIpSocket class 1421
LookupName
TCustomWinSocket class 74
LookupObject
TVariableLookup class 6718
LookupPort
TIpSocket class 1421
LookupProtocol
TIpSocket class 1421
LookupService
TCustomWinSocket class 75
LookupState
TCustomWinSocket class 86
loUserLocale enumeration member 4328
Low
DynamicArray template 3271
LowerCase
AnsiStringBase class 3198
AnsiStringT template 3226
UnicodeString class 3406
WideString class 3482
lposAbove enumeration member 6109
lposBelow enumeration member 6109
lposLeft enumeration member 6109
lposRight enumeration member 6109
lsButtons enumeration member 6527
lsFields enumeration member 6527
lsIdle enumeration member 148
lsLookupAddress enumeration member 148
lsLookupService enumeration member 148
lsUnknown enumeration member 6527

M

m_Count
DynArrayOutOfRange class 3273
m_Index
DynArrayOutOfRange class 3273
Main
TService class 3077
MainMenu1
TSocketForm class 165
MakeHeaderNodes
TSOAPDOMProcessor class 1239
MakeIdentifier
THTMLItems class 6667
Manager
TScreenTipItem class 333
MapDomain
TBaseSocket class 1375
MapSockType

TBaseSocket class 1375
MasterFields
TSimpleDataSet class 921
TSQLTable class 1691
MasterLink
TSQLTable class 1692
MasterRowSetVarName
TXMLBroker class 7315
MasterSource
TSimpleDataSet class 921
TSQLTable class 1692
Max
TScrollBar class 2900
TScroller class 4461
MaxBlobSize
TCustomSQLDataSet class 1559
TInternalSQLDataSet class 894
TSQLDataSet class 1640
TSQLQuery class 1661
TSQLStoredProc class 1677
TSQLTable class 1692
MaxCapacity
TStringBuilder class 4022
MaxConnections
TWebRequestHandler class 6640
MaxErrors
TXMLBroker class 7323
MaxLength
TAdapterDisplayField class 6378
TAdapterEditColumn class 6394
TComboBox class 2212
TCustomAdapterDisplayField class 6474
TCustomCombo class 2310
TCustomEdit class 2368
TEdit class 2486
TItemProp class 4783
TMemo class 2759
MaxRecords
TXMLBroker class 7324
MaxSessions
TCustomSessionsService class 6754
TSessions class 395
TSessionsService class 6758
MaxSinglePostSize
THTTPReqResp class 1289
MaxStmtsPerConn
TSQLConnection class 1620
MaxTraceCount
TSQLMonitor class 1651
mba16Byte enumeration member 3732
mba8Byte enumeration member 3732
mbLeadByte enumeration member 3181, 4329
mbSingleByte enumeration member 3181, 4329
mbTrailByte enumeration member 3181, 4329
MeasureItem
TCustomComboBox class 2330
TCustomListBox class 2415
MeasureTab
TTabSet class 4477
Memory
IDataBlock interface 272
TDataBlock class 205
MenuBar
IWebBrowser2Disp interface 771
IWebBrowserApp interface 783
IWebBrowserAppDisp interface 792
TInternetExplorer class 452
TWebBrowser class 530
Message
Exception class 3965
IMessages interface 7123
IParam interface 7134
Messages
IDefinition interface 7099
TAdapterErrorsList class 5266
MetaData
TSQLConnection class 1620
MethodAddress

TObject class 3378
MethodName
TObject class 3379
miAdd
TSocketForm class 165
miAddClick
TSocketForm class 177
miClose
TSocketForm class 165
miCloseClick
TSocketForm class 177
miDisconnect
TSocketForm class 165
miDisconnectClick
TSocketForm class 177
miExit
TSocketForm class 166
miExitClick
TSocketForm class 178
Millisecond
TXSCustomDateTime class 7818
TXSDateTime class 7832
TXSTime class 7858
MimeBoundary
IWebNode interface 6619
TLinkedWebNode class 1326
MIMEBoundary
IMimeAttachmentHandler interface 1182
Min
TScrollBar class 2901
TScroller class 4462
MinClientRect
TTabSet class 4477
Minute
TXSCustomDateTime class 7819
TXSTime class 7859
MinuteOffset
TXSCustomDateTime class 7819
TXSTime class 7859
miPorts
TSocketForm class 166
miProperties
TSocketForm class 166
miPropertiesClick
TSocketForm class 178
miRemove
TSocketForm class 166
miShowHostName
TSocketForm class 167
mkAmbiguous enumeration member 4323
mkClassConstructor enumeration member 4751
mkClassFunction enumeration member 4751
mkClassProcedure enumeration member 4751
mkConstructor enumeration member 4751
mkDestructor enumeration member 4751
mkExactMatch enumeration member 4323
mkFunction enumeration member 4751
mkNone enumeration member 4323
mkOperatorOverload enumeration member 4751
mkProcedure enumeration member 4751
mkSafeFunction enumeration member 4751
mkSafeProcedure enumeration member 4751
mkSingleMatch enumeration member 4323
ModalResult
TButton class 2059
TCustomButton class 2273
Mode
IShellNameSpace interface 708
IShellNameSpaceDisp interface 715
TShellNameSpace class 481
Modified
IXMLDocument interface 7665
TCustomEdit class 2369
TXMLDocument class 7387
ModifyAccess
TAdapter class 5196
TAdapterBooleanField class 5229
TAdapterField class 5273

TAdapterFileField class 5288
TAdapterImageField class 5315
TAdapterMemoField class 5329
TAdapterMultiValueField class 5339
TAdapterNamedField class 5369
TCustomAdapter class 5574
TPagedAdapter class 5851
Module
TSiteModuleHelper class 6571
ModuleAdded
TMultiModuleWebAppServices class 6233
Month
TXSCustomDateTime class 7820
TXSDate class 7827
MouseDown
TTabSet class 4478
TValueListEditor class 4797
Move
TAbstractNamedVariants class 6129
TWideStrings class 6843
MoveNext
TWideStringsEnumerator class 6857
MoveSelectionDown
IShellFavoritesNameSpace interface 693
IShellFavoritesNameSpaceDisp interface 698
IShellNameSpaceDisp interface 712
TShellNameSpace class 475
MoveSelectionTo
IShellFavoritesNameSpace interface 694
IShellFavoritesNameSpaceDisp interface 698
IShellNameSpaceDisp interface 712
TShellNameSpace class 476
MoveSelectionUp
IShellFavoritesNameSpace interface 694
IShellFavoritesNameSpaceDisp interface 698
IShellNameSpaceDisp interface 712
TShellNameSpace class 476
mtFault enumeration member 6740
mtHeaderInput enumeration member 6740
mtHeaderOutput enumeration member 6740
mtInput enumeration member 6740
mtOutput enumeration member 6740
MultipleTransactionsSupported
TSQLConnection class 1621
MultiSelect
TListBox class 2683

N

N1
TSocketForm class 167
N2
TSocketForm class 167
N3
TSocketForm class 167
name
IDOMAttr interface 7524
IDOMDocumentType interface 7542
Name
IAdapterHiddenField interface 1003
IBinding interface 7082
IBindingOperation interface 7087
IDefinition interface 7099
IHeaderInfo interface 7251
IMessage interface 7119
IOperation interface 7127
IParam interface 7134
IPart interface 7138
IPort interface 7145
IPortType interface 7151
IQualifiedName interface 7255
IService interface 7158
IWebBrowser2Disp interface 772
IWebBrowserApp interface 784
IWebBrowserAppDisp interface 792
TAbstractScriptEngine class 6651
TDependency class 3068
TInternetExplorer class 452
TLanguages class 3991

TWebBrowser class 531
NamedItemCount
TScriptGlobalObjects class 6691
NamedItemName
TScriptGlobalObjects class 6691
NameFromLCID
TLanguages class 3991
NameFromLocaleID
TLanguages class 3991
Names
TAbstractNamedVariants class 6132
TGeneratedFunctions class 6508
TWideStrings class 6850
Namespace
IHeaderInfo interface 7251
IImport interface 7113
IQualifiedName interface 7255
namespaceURI
IDOMNode interface 7571
NamespaceURI
IXMLNode interface 7693
NameValueSeparator
TWideStrings class 6851
NativeCommand
TCustomSQLDataSet class 1560
NativeToXS
TXSBoolean class 7802
TXSDate class 7825
TXSDateTime class 7831
TXSDecimal class 7834
TXSDuration class 7838
TXSHexBinary class 7841
TXSInteger class 7844
TXSLong class 7846
TXSString class 7848
TXSTime class 7853
Navigate
IWebBrowser interface 741
IWebBrowser2Disp interface 766
IWebBrowserAppDisp interface 788
IWebBrowserDisp interface 797
TInternetExplorer class 439, 440
TWebBrowser class 512
TWebBrowser_V1 class 574, 575
Navigate2
IWebBrowser2 interface 755
IWebBrowser2Disp interface 766
TInternetExplorer class 441, 442
TWebBrowser class 514
NavigateAndFind
IShellUIHelper interface 719
IShellUIHelperDisp interface 724
TShellUIHelper class 493
NavigateComplete
DWebBrowserEvents interface 624
NavigateComplete2
DWebBrowserEvents2 interface 629
NavigateError
DWebBrowserEvents2 interface 629
NavigateToDefaultSearch
ISearchAssistantOC interface 652
ISearchAssistantOC2Disp interface 662
ISearchAssistantOC3Disp interface 673
ISearchAssistantOCDisp interface 681
ncAddAttribute enumeration member 7494
ncInsertChild enumeration member 7494
NCPaint
TStandardCustomizePopup class 1746
TStandardMenuPopup class 1757
TThemedCustomizePopup class 4563
TThemedPopupMenu class 4574
TToolWindow class 4641
TXPStyleCustomizePopup class 7770
TXPStylePopupMenu class 7781
ncRemoveAttribute enumeration member 7494
ncRemoveChild enumeration member 7494
ncrError enumeration member 5094
ncrLoose enumeration member 5094

ncrStrict enumeration member 5094
ncUpdateValue enumeration member 7494
NeedAdapterMode
IAdapterMode interface 1011
NeedMultipartForm
TControlRequirements class 6429
NeedPrevValue
IWebGetFieldPrevValue interface 5895
NestingLevel
IXMLNodeAccess interface 7486
TXMLNode class 7428
NETDetectNextNavigate
ISearchAssistantOC interface 652
ISearchAssistantOC2Disp interface 662
ISearchAssistantOC3Disp interface 673
ISearchAssistantOCDisp interface 681
New
TLineNumberMap class 6670
NewComponent
TWebComponentList class 6031
NewFolder
IShellFavoritesNameSpace interface 694
IShellFavoritesNameSpaceDisp interface 698
IShellNameSpaceDisp interface 712
TShellNameSpace class 476
NewInstance
THTTPReqResp class 1285
TInterfacedObject class 3346
TObject class 3379
TSoapDataModule class 1221
TSOAPDOMProcessor class 1239
TXMLDocument class 7370
NewSession
ISessionsService interface 1068
TCustomSessionsService class 6752
NewWindow
DWebBrowserEvents interface 624
NewWindow2
DWebBrowserEvents2 interface 630
NextColumn
TAdapterFormLayout class 6415
NextID
TAbstractSessionIDGen class 375
TSessionIDGen class 378
NextIteration
IIteratorSupport interface 1056
TAdapterHiddenFieldsList class 5307
TAdapterSubComponent class 5415
TBaseAdapterAction class 5472
TBaseAdapterErrorsList class 5486
TCustomAdapter class 5564
TSiteModuleHelper class 6570
NextPage
TCustomLoginFormAdapter class 5713
TLoginFormAdapterLoginAction class 5841
NextRecordSet
TSQLStoredProc class 1676
nextSibling
IDOMNode interface 7571
NextSibling
IXMLNode interface 7682
TXMLNode class 7429
NextValueIteration
IWebGetFieldValues interface 1094
TBaseAdapterMultiValueField class 5496
nlCreateNode enumeration member 7495
nlInsert enumeration member 7495
nlRemove enumeration member 7495
Node
IXMLDocument interface 7665
TXMLDocument class 7388
NodeIndentStr
IXMLDocument interface 7666
TXMLDocument class 7388
nodeName
IDOMNode interface 7572
NodeName
IXMLNode interface 7693

Nodes
IXMLNodeCollection interface 7703
IXMLNodeList interface 7714
nodeType
IDOMNode interface 7572
NodeType
IXMLNode interface 7694
nodeValue
IDOMNode interface 7574
NodeValue
IXMLNode interface 7695
NoParam
Variant class 3434
normalize
IDOMNode interface 7565
Normalize
IXMLNode interface 7683
TXMLNode class 7429
notationName
IDOMEntity interface 7551
notations
IDOMDocumentType interface 7542
Notification
TAdapterControlGroup class 6350
TAdapterUpdateField class 5431
TCommonDialogAction class 1789
TCustomAdapter class 5565
TCustomAdapterActionButton class 6436
TCustomAdapterErrorList class 6482
TCustomButton class 2265
TCustomLabel class 2387
TCustomScreenTipsManager class 311
TCustomScreenTipsPopup class 319
TCustomStaticText class 2460
TCustomWebAppComponents class 6211
TCustomWebAppDataModule class 6543
TCustomWebAppPageModule class 6548
TCustomWebDataModule class 6553
TCustomWebPageModule class 6561
TCustomXSLPageProducer class 7889
TEditAction class 1794
THTTPRIO class 1250
THTTPSoapDispatchNode class 1260
TLocalConnection class 4552
TSearchAction class 1898
TShadowWindow class 403
TSimpleDataSet class 901
TSiteModuleHelper class 6570
TSoapConnection class 1194
TSoapPascalInvoker class 1355
TSQLConnection class 1601
TTabSet class 4479
TWindowAction class 1934
TWSDLClientNode class 7269
TXMLBroker class 7316
NotificationProc
TXMLNodeList class 7476
Notify
TXMLBroker class 7325
NotifyActivate
INotifyWebActivate interface 6089
TCustomAdapter class 5565
TSiteModuleHelper class 6570
NotifyActivateModule
INotifyActivateModule interface 6087
NotifyAdapterChange
IAdapterNotifyAdapterChange interface 5876
INotifyAdapterChange interface 1058
TAdapterControlGroup class 6351
TAdapterDisplayReferenceGroup class 6385
TAdapterSubComponent class 5416
TCustomAdapter class 5565
TCustomAdapterCommandColumn class 6448
NotifyCount
TXMLBroker class 7325
NotifyDeactivate
INotifyWebActivate interface 6089
TCustomAdapter class 5566

TSiteModuleHelper class 6570
NotifyDeactivateModule
INotifyActivateModule interface 6088
NotifyList
TCustomAdapter class 5574
NotifyLocked
TWebComponentList class 6031
NotifySearchSettingsChanged
ISearchAssistantOC interface 652
ISearchAssistantOC2Disp interface 663
ISearchAssistantOC3Disp interface 673
ISearchAssistantOCDisp interface 681
NotifyStatusChange
ISessionsService interface 1068
TCustomSessionsService class 6752
TSessions class 392
NotifyWebActivate
TWebNotifyContainer class 6603
NotifyWebDeactivate
TWebNotifyContainer class 6603
NSPrefixBase
TXMLDocument class 7389
ntAttribute enumeration member 7716
ntCData enumeration member 7716
ntComment enumeration member 7716
ntContainsUnit enumeration member 4329
ntDcpBpiName enumeration member 4329
ntDocFragment enumeration member 7716
ntDocType enumeration member 7716
ntDocument enumeration member 7716
ntElement enumeration member 7716
ntEntity enumeration member 7716
ntEntityRef enumeration member 7716
ntNotation enumeration member 7716
ntProcessingInstr enumeration member 7716
ntRequiresPackage enumeration member 4329
ntReserved enumeration member 7716
ntText enumeration member 7716
NumbersOnly
TCustomEdit class 2369
TEdit class 2486
NumericMapping
TCustomSQLDataSet class 1560

O

Obj
TScriptComponent class 975
TVariableLookup class 6718
ObjectBroker
TSocketConnection class 228
TWebConnection class 262
ObjectIdentifier
IAdapterDispatchParams interface 990
TAdapterDispatchParams class 6159
ObjectList
TAbstractScriptObjectFactories class 6655
Objects
TWideStrings class 6851
ObjectToSOAP
TXMLData class 7799
ObjectView
TCustomSQLDataSet class 1560
TSimpleDataSet class 922
OEMConvert
TCustomEdit class 2369
TEdit class 2486
TMemo class 2759
Offline
IWebBrowser2 interface 760
IWebBrowser2Disp interface 772
TInternetExplorer class 454
TWebBrowser class 531
OlePromotion
TCustomVariantType class 5001
OleVariant
OleVariant class 3281
OleVariant class 3274
!= 3302

!= operator 3301
~OleVariant 3279
< 3302, 3303
< operator 3302
<= 3303
<= operator 3303
= 3304
= operator 3303
== 3305, 3306
== operator 3305
> 3306
> operator 3306
>= 3307
>= operator 3306
about OleVariant class 3274
OleVariant 3281
OleVariant constructor 3279
OleVariant data members 3282
OleVariant Members 3275
OleVariant operators 3301
TVarData::Reserved1 3284
TVarData::Reserved2 3284
TVarData::Reserved3 3284
TVarData::VArray 3284
TVarData::VBoolean 3285
TVarData::VByte 3285
TVarData::VCurrency 3285
TVarData::VDate 3285
TVarData::VDispatch 3285
TVarData::VDouble 3286
TVarData::VError 3286
TVarData::VInteger 3286
TVarData::VOleStr 3286
TVarData::VPointer 3287
TVarData::VSingle 3287
TVarData::VSmallint 3287
TVarData::VType 3287
TVarData::VUnknown 3287
Variant::- 3288
Variant::! 3288
Variant::%= 3288
Variant::&= 3288
Variant::*= 3289
Variant::/= 3289
Variant::^= 3289
Variant::__int64 3289
Variant::|= 3289
Variant::+= 3290
Variant::<<= 3290
Variant::-= 3290
Variant::>>= 3290
Variant::AnsiString 3291
Variant::ArrayDimCount 3291
Variant::ArrayHighBound 3291
Variant::ArrayLock 3291
Variant::ArrayLowBound 3291
Variant::ArrayRedim 3292
Variant::ArrayUnlock 3292
Variant::AsType 3292
Variant::bool 3292
Variant::Byte* 3293
Variant::ChangeType 3293
Variant::char 3293
Variant::Clear 3293
Variant::Currency 3293
Variant::Currency* 3294
Variant::double 3294
Variant::double* 3294
Variant::float 3294
Variant::float* 3295
Variant::GetElement 3295
Variant::GetError 3295
Variant::IDispatch* 3295
Variant::IInterface* 3295
Variant::int 3296
Variant::int* 3296
Variant::IsArray 3296
Variant::IsEmpty 3296

Variant::IsNull 3297
Variant::IUnknown* 3297
Variant::long 3297
Variant::long* 3297
Variant::PutElement 3297
Variant::SetError 3298
Variant::short 3298
Variant::short* 3298
Variant::signed char 3298
Variant::signed char* 3299
Variant::TDateTime 3299
Variant::TDateTime* 3299
Variant::Type 3299
Variant::UnicodeString 3299
Variant::unsigned char 3300
Variant::unsigned int 3300
Variant::unsigned int* 3300
Variant::unsigned long 3300
Variant::unsigned short 3301
Variant::unsigned short* 3301
Variant::WideString 3301
OnAccept
TBrowseForFolder class 1772
TColorSelect class 1781
TCommonDialogAction class 1790
TCustomSocket class 59
TCustomTcpServer class 1413
TFileOpen class 1826
TFilePageSetup class 1843
TFilePrintSetup class 1852
TFileSaveAs class 1862
TFontEdit class 1871
TPrintDlg class 1891
TSearchFind class 1907
TSearchReplace class 1930
TServerSocket class 114
TTcpServer class 1469
OnActionRequestNotHandled
TAdapterDispatcher class 6150
TCustomAdapterDispatcher class 6177
OnActivate
TCustomWebDataModule class 6556
TSiteModuleHelper class 6573
TWebAppDataModule class 6587
TWebAppPageModule class 6592
TWebDataModule class 6598
TWebPageModule class 6607
OnActivated
IShellWindows interface 727
IShellWindowsDisp interface 731
TShellWindows class 501
OnAfterCheckAccessRights
TCustomWebUserList class 6785
TWebUserList class 6798
OnAfterDispatch
TMultiModuleWebAppServices class 6235
TWebAppComponents class 6249
OnAfterDispatchAction
TAdapterDispatcher class 6151
TCustomAdapterDispatcher class 6178
OnAfterDispatchImage
TAdapterDispatcher class 6151
TCustomAdapterDispatcher class 6178
OnAfterDispatchPage
TCustomPageDispatcher class 6196
TCustomWebPageModule class 6563
TPageDispatcher class 6239
TSitePageModuleHelper class 6583
TWebAppPageModule class 6593
TWebPageModule class 6607
OnAfterExecute
TAdapterAction class 5207
TAdapterGotoPageAction class 5297
TAdapterNextPageAction class 5377
TAdapterPrevPageAction class 5404
TBaseAdapterAction class 5477
TCustomAdapterAction class 5587
TEndUserAdapterLoginFormAction class 5782

TEndUserAdapterLogoutAction class 5791
TLoginFormAdapterLoginAction class 5842
TSoapConnection class 1203
OnAfterExecuteAction
TAdapter class 5197
TApplicationAdapter class 5445
TCustomAdapter class 5576
TEndUserAdapter class 5771
TEndUserSessionAdapter class 5800
TLoginFormAdapter class 5832
TPagedAdapter class 5852
OnAfterGetActionResponse
TAdapter class 5197
TApplicationAdapter class 5445
TCustomAdapter class 5576
TEndUserAdapter class 5772
TEndUserSessionAdapter class 5801
TLoginFormAdapter class 5832
TPagedAdapter class 5853
OnAfterGetContent
TAdapterPageProducer class 1142
TBaseAdapterPageProducer class 1158
OnAfterGetResponse
TAdapterAction class 5207
TAdapterGotoPageAction class 5297
TAdapterNextPageAction class 5378
TAdapterPrevPageAction class 5404
TBaseAdapterAction class 5477
TCustomAdapterAction class 5587
TEndUserAdapterLoginFormAction class 5783
TEndUserAdapterLogoutAction class 5791
TLoginFormAdapterLoginAction class 5843
OnAfterPublishingWSDL
TWebServExp class 6733
TWSDLHTMLPublish class 7289
TWSDLPublish class 7294
OnAfterRedirectToPage
TCustomPageDispatcher class 6197
TCustomWebPageModule class 6563
TSitePageModuleHelper class 6584
OnAfterUpdateFields
TAdapter class 5198
TCustomAdapter class 5576
TPagedAdapter class 5853
OnAfterValidateFields
TAdapter class 5198
TCustomAdapter class 5577
TPagedAdapter class 5853
OnAfterValidateUser
TCustomWebUserList class 6786
TWebUserList class 6798
OnAlignInsertBefore
TGroupBox class 2566
TTabSet class 4509
OnAlignPosition
TGroupBox class 2566
TTabSet class 4510
OnAsyncLoad
TXMLDocument class 7395
OnBeforeCheckAccessRights
TCustomWebUserList class 6786
TWebUserList class 6799
OnBeforeDispatch
TMultiModuleWebAppServices class 6235
TWebAppComponents class 6250
OnBeforeDispatchAction
TAdapterDispatcher class 6151
TCustomAdapterDispatcher class 6179
OnBeforeDispatchImage
TAdapterDispatcher class 6152
TCustomAdapterDispatcher class 6179
OnBeforeDispatchPage
TCustomPageDispatcher class 6197
TCustomWebPageModule class 6564
TPageDispatcher class 6239
TSitePageModuleHelper class 6584
TWebAppPageModule class 6593
TWebPageModule class 6608

OnBeforeExecute
TAdapterAction class 5208
TAdapterGotoPageAction class 5297
TAdapterNextPageAction class 5378
TAdapterPrevPageAction class 5404
TBaseAdapterAction class 5478
TCustomAdapterAction class 5588
TEndUserAdapterLoginFormAction class 5783
TEndUserAdapterLogoutAction class 5791
TLoginFormAdapterLoginAction class 5843
TSoapConnection class 1203
OnBeforeExecuteAction
TAdapter class 5198
TApplicationAdapter class 5446
TCustomAdapter class 5577
TEndUserAdapter class 5772
TEndUserSessionAdapter class 5801
TLoginFormAdapter class 5833
TPagedAdapter class 5854
OnBeforeGetActionResponse
TAdapter class 5199
TApplicationAdapter class 5446
TCustomAdapter class 5578
TEndUserAdapter class 5772
TEndUserSessionAdapter class 5801
TLoginFormAdapter class 5833
TPagedAdapter class 5854
OnBeforeGetContent
TAdapterPageProducer class 1142
TBaseAdapterPageProducer class 1159
OnBeforeGetResponse
TAdapterAction class 5208
TAdapterGotoPageAction class 5298
TAdapterNextPageAction class 5378
TAdapterPrevPageAction class 5405
TBaseAdapterAction class 5478
TCustomAdapterAction class 5588
TEndUserAdapterLoginFormAction class 5783
TEndUserAdapterLogoutAction class 5792
TLoginFormAdapterLoginAction class 5844
OnBeforeNavigate
TWebBrowser_V1 class 597
OnBeforeNavigate2
TInternetExplorer class 459
TWebBrowser class 545
OnBeforePost
THTTPReqResp class 1294
OnBeforePublishingTypes
TWebServExp class 6733
TWSDLHTMLPublish class 7290
TWSDLPublish class 7294
OnBeforePublishingWSDL
TWSDLHTMLPublish class 7290
TWSDLPublish class 7294
OnBeforeRedirectToPage
TCustomPageDispatcher class 6198
TCustomWebPageModule class 6564
TSitePageModuleHelper class 6584
OnBeforeUpdateFields
TAdapter class 5199
TCustomAdapter class 5578
TPagedAdapter class 5854
OnBeforeValidateFields
TAdapter class 5200
TCustomAdapter class 5578
TPagedAdapter class 5855
OnBeforeValidateUser
TCustomWebUserList class 6786
TWebUserList class 6799
OnCalcFields
TCustomSQLDataSet class 1569
TSimpleDataSet class 955
OnCallback
TBrowseForFolder class 1773
OnCancel
TBrowseForFolder class 1773
TColorSelect class 1782
TCommonDialogAction class 1791

TFileOpen class 1827
TFilePageSetup class 1843
TFilePrintSetup class 1853
TFileSaveAs class 1862
TFontEdit class 1872
TPrintDlg class 1892
TSearchFind class 1907
TSearchReplace class 1930
OnCanViewPage
TCustomPageDispatcher class 6198
TPageDispatcher class 6240
OnChange
TComboBox class 2228
TCustomCombo class 2318
TCustomEdit class 2381
TEdit class 2501
TImageMargins class 2607
TMemo class 2777
TScrollBar class 2920
TTabbedNotebook class 4435
TTabSet class 4511
TWebComponentList class 6035
TWideStringList class 6824
OnChanging
TWideStringList class 6825
OnClick
TButton class 2073
TCheckBox class 2151
TComboBox class 2230
TEdit class 2502
TGroupBox class 2567
TLabel class 2637
TListBox class 2699
TMemo class 2778
TRadioButton class 2848
TScroller class 4463
TStaticText class 2978
TTabbedNotebook class 4436
TTabSet class 4511
TValueListEditor class 4839
OnClientConnect
TCustomServerSocket class 51
TServerSocket class 115
TServerWinSocket class 133
OnClientDisconnect
TCustomServerSocket class 52
TServerSocket class 116
TServerWinSocket class 134
OnClientError
TCustomServerSocket class 52
TServerSocket class 116
TServerWinSocket class 135
OnClientRead
TCustomServerSocket class 53
TServerSocket class 117
TServerWinSocket class 135
OnClientToHostWindow
TInternetExplorer class 459
TWebBrowser class 546
OnClientWrite
TCustomServerSocket class 53
TServerSocket class 117
TServerWinSocket class 136
OnCloseUp
TComboBox class 2232
TCustomCombo class 2320
OnCommandStateChange
TInternetExplorer class 459
TWebBrowser class 546
TWebBrowser_V1 class 597
OnConnect
TClientSocket class 32
TCustomIpClient class 1405
TCustomSocket class 60
TTcpClient class 1450
TUdpSocket class 1490
OnConnecting
TClientSocket class 32

TCustomSocket class 60
OnContextPopup
TButton class 2075
TCheckBox class 2153
TComboBox class 2233
TEdit class 2504
TGroupBox class 2569
TLabel class 2639
TListBox class 2701
TMemo class 2780
TRadioButton class 2849
TScrollBar class 2920
TStaticText class 2980
TTabbedNotebook class 4436
TValueListEditor class 4841
OnContinue
TService class 3087
OnCreated
IShellWindows interface 727
IShellWindowsDisp interface 731
TShellWindows class 501
OnCreateHandle
TBaseSocket class 1388
TTcpClient class 1451
TTcpServer class 1472
TUdpSocket class 1490
OnCreateScreenTip
TCustomScreenTipsManager class 315
TScreenTipsManager class 341
OnData
TCustomListBox class 2438
TListBox class 2701
OnDataFind
TCustomListBox class 2439
TListBox class 2702
OnDataObject
TCustomListBox class 2439
TListBox class 2702
OnDataRequest
TXMLTransformProvider class 7759
OnDblClick
TComboBox class 2233
TEdit class 2504
TGroupBox class 2570
TLabel class 2640
TListBox class 2703
TMemo class 2780
TRadioButton class 2850
TStaticText class 2980
TValueListEditor class 4841
OnDeactivate
TCustomWebDataModule class 6556
TSiteModuleHelper class 6573
TWebAppDataModule class 6588
TWebAppPageModule class 6594
TWebDataModule class 6599
TWebPageModule class 6608
OnDeleteError
TSimpleDataSet class 955
OnDestroyHandle
TBaseSocket class 1390
TTcpClient class 1453
TTcpServer class 1474
TUdpSocket class 1493
OnDisconnect
TClientSocket class 33
TCustomIpClient class 1406
TCustomSocket class 61
TTcpClient class 1456
TUdpSocket class 1495
OnDisplayComponentChange
TAdapterDisplayReferenceGroup class 6385
OnDockDrop
TGroupBox class 2570
OnDockOver
TGroupBox class 2572
OnDocumentComplete
TInternetExplorer class 459

TWebBrowser class 546
OnDoubleClick
TShellNameSpace class 482
OnDownloadBegin
TInternetExplorer class 459
TWebBrowser class 547
TWebBrowser_V1 class 598
OnDownloadComplete
TInternetExplorer class 460
TWebBrowser class 547
TWebBrowser_V1 class 598
OnDragDrop
TButton class 2076
TCheckBox class 2154
TComboBox class 2234
TEdit class 2505
TGroupBox class 2574
TLabel class 2640
TListBox class 2703
TMemo class 2781
TRadioButton class 2850
TScrollBar class 2921
TStaticText class 2981
TTabSet class 4513
TValueListEditor class 4842
TWebBrowser class 548
TWebBrowser_V1 class 598
OnDragOver
TButton class 2079
TCheckBox class 2157
TComboBox class 2237
TEdit class 2508
TGroupBox class 2577
TLabel class 2643
TListBox class 2706
TMemo class 2784
TRadioButton class 2853
TScrollBar class 2924
TStaticText class 2984
TTabSet class 4516
TValueListEditor class 4845
TWebBrowser class 551
TWebBrowser_V1 class 601
OnDrawCell
TValueListEditor class 4848
OnDrawItem
TComboBox class 2240
TCustomComboBox class 2338
TCustomListBox class 2439
TListBox class 2709
OnDrawTab
TTabSet class 4519
OnDropDown
TComboBox class 2240
TCustomCombo class 2320
OnDropDownClick
TButton class 2082
TCustomButton class 2276
OnDuplicateCreateError
TCustomScreenTipsManager class 315
TScreenTipsManager class 342
OnEditButtonClick
TValueListEditor class 4849
OnEditError
TSimpleDataSet class 956
OnEndDock
TButton class 2082
TCheckBox class 2160
TComboBox class 2242
TEdit class 2511
TGroupBox class 2580
TLabel class 2646
TListBox class 2711
TMemo class 2787
TRadioButton class 2857
TScrollBar class 2927
TStaticText class 2987
TTabSet class 4519

TValueListEditor class 4850
OnEndDrag
TButton class 2083
TCheckBox class 2160
TComboBox class 2242
TEdit class 2511
TGroupBox class 2580
TLabel class 2647
TListBox class 2712
TMemo class 2787
TRadioButton class 2857
TScrollBar class 2928
TStaticText class 2987
TTabSet class 4520
TValueListEditor class 4850
TWebBrowser class 554
TWebBrowser_V1 class 604
OnEndSession
TCustomSessionsService class 6755
TSessionsService class 6759
OnEnter
TButton class 2084
TCheckBox class 2161
TComboBox class 2243
TEdit class 2512
TGroupBox class 2581
TListBox class 2713
TMemo class 2788
TRadioButton class 2858
TScrollBar class 2929
TTabbedNotebook class 4437
TTabSet class 4521
TValueListEditor class 4851
TWebBrowser class 555
TWebBrowser_V1 class 605
OnError
TBaseSocket class 1393
TClientSocket class 33
TCustomSocket class 61
TTcpClient class 1456
TUdpSocket class 1496
OnErrorEvent
TCustomWinSocket class 89
OnException
TCustomWebAppComponents class 6215
TWebAppComponents class 6250
OnExceptionEvent
TSoapPascalInvoker class 1357
OnExecute
TAdapterAction class 5208
TCustomAdapterAction class 5588
TService class 3088
OnExit
TButton class 2086
TCheckBox class 2164
TComboBox class 2246
TEdit class 2515
TGroupBox class 2584
TListBox class 2715
TMemo class 2791
TRadioButton class 2861
TScrollBar class 2931
TTabbedNotebook class 4439
TTabSet class 4523
TValueListEditor class 4854
TWebBrowser class 558
TWebBrowser_V1 class 608
OnFavoritesSelectionChange
TShellNameSpace class 483
OnFileDownload
TInternetExplorer class 460
TWebBrowser class 562
OnFilterRecord
TSimpleDataSet class 956
OnFindIncludeFile
TCustomLocateFileService class 6186
TLocateFileService class 6221
OnFindStream

TCustomLocateFileService class 6187
TLocateFileService class 6222
OnFindTemplateFile
TCustomLocateFileService class 6187
TLocateFileService class 6222
OnFrameBeforeNavigate
TWebBrowser_V1 class 612
OnFrameNavigateComplete
TWebBrowser_V1 class 612
OnFrameNewWindow
TWebBrowser_V1 class 612
OnFullScreen
DWebBrowserEvents2 interface 630
TInternetExplorer class 460
TWebBrowser class 562
OnGetActionParams
TAdapter class 5200
TApplicationAdapter class 5446
TCustomAdapter class 5579
TEndUserAdapter class 5773
TEndUserSessionAdapter class 5802
TLoginFormAdapter class 5833
TPagedAdapter class 5855
OnGetAdapterRequestParams
TAdapterDispatcher class 6152
TCustomAdapterDispatcher class 6180
OnGetAttachment
IMimeAttachmentHandler interface 1183
TSoapPascalInvoker class 1358
OnGetDefaultComponentList
TWebContainerComponentHelper class 6075
OnGetDisplayName
TCustomEndUserAdapter class 5685
TEndUserAdapter class 5773
OnGetDisplayText
TAdapterField class 5275
TAdapterUpdateField class 5433
OnGetEditMask
TValueListEditor class 4858
OnGetEditText
TValueListEditor class 4859
OnGetEnabled
TAdapterAction class 5209
TCustomAdapterAction class 5589
OnGetEOF
TDefaultFieldsPagedAdapter class 5762
TPagedAdapter class 5856
OnGetErrorResponse
TXMLBroker class 7330
OnGetFileCount
TAdapterFileField class 5289
TCustomAdapterFileField class 5613
OnGetFileNames
TAdapterFileField class 5290
TCustomAdapterFileField class 5613
OnGetFirstRecord
TDefaultFieldsPagedAdapter class 5763
TPagedAdapter class 5856
OnGetHREF
TAdapterImageField class 5317
TCustomAdapterImageField class 5635
OnGetImage
TAdapterImageField class 5318
TCustomAdapterImageField class 5636
OnGetImageIndex
TTabSet class 4527
OnGetImageName
TAdapterImageField class 5318
TCustomAdapterImageField class 5636
OnGetNextRecord
TDefaultFieldsPagedAdapter class 5763
TPagedAdapter class 5856
OnGetPageName
TAdapterGotoPageAction class 5298
TCustomAdapterGotoPageAction class 5622
OnGetParams
TAdapterAction class 5209
TAdapterGotoPageAction class 5298

TAdapterImageField class 5318
TAdapterNextPageAction class 5379
TAdapterPrevPageAction class 5405
TBaseAdapterAction class 5478
TCustomAdapterImageField class 5636
TEndUserAdapterLoginFormAction class 5784
TEndUserAdapterLogoutAction class 5792
TLoginFormAdapterLoginAction class 5844
OnGetPickList
TValueListEditor class 4860
OnGetRecordCount
TDefaultFieldsPagedAdapter class 5763
TPagedAdapter class 5856
OnGetRecordIndex
TDefaultFieldsPagedAdapter class 5763
TPagedAdapter class 5857
OnGetResponse
TXMLBroker class 7331
OnGetScreenTipWinClass
TCustomScreenTipsManager class 316
TScreenTipsManager class 342
OnGetSiteInfo
TGroupBox class 2588
OnGetSocket
TCustomServerSocket class 54
TServerSocket class 118
TServerWinSocket class 137
OnGetThread
TCustomServerSocket class 54
TCustomTcpServer class 1415
TServerSocket class 118
TServerSocketThread class 1439
TServerWinSocket class 137
TTcpServer class 1477
OnGetUserID
TCustomEndUserAdapter class 5685
OnGetValue
TAdapterBooleanField class 5230
TAdapterField class 5275
TAdapterMemoField class 5331
TCustomAdapterBooleanField class 5597
TCustomAdapterField class 5603
TCustomAdapterMemoField class 5645
OnGetValueCount
TAdapterMultiValueField class 5341
TCustomAdapterMultiValueField class 5654
OnGetValues
TAdapterMultiValueField class 5341
TCustomAdapterMultiValueField class 5655
OnHasRights
TBaseEndUserAdapter class 5516
TEndUserAdapter class 5774
TEndUserSessionAdapter class 5802
OnHasValue
TAdapterMultiValueField class 5342
TCustomAdapterMultiValueField class 5655
OnHint
TBrowseForFolder class 1774
TColorSelect class 1782
TFileExit class 1816
TFileOpen class 1827
TFilePageSetup class 1844
TFilePrintSetup class 1853
TFileSaveAs class 1863
TFontEdit class 1872
TPrintDlg class 1892
TSearchFind class 1908
TSearchFindNext class 1920
TSearchReplace class 1930
OnHostAttrNotify
TXMLNode class 7443
OnHostChildNotify
TXMLNode class 7444
OnImageRequestNotHandled
TAdapterDispatcher class 6153
TCustomAdapterDispatcher class 6180
OnInitialized
TShellNameSpace class 483

OnIsLoggedIn
TCustomEndUserAdapter class 5686
TEndUserAdapter class 5774
OnIsLoginRequired
TCustomPageDispatcher class 6199
TPageDispatcher class 6241
OnIterateRecords
TAdapter class 5200
TCustomAdapter class 5579
OnKeyDown
TButton class 2090
TCheckBox class 2168
TComboBox class 2250
TEdit class 2519
TListBox class 2719
TMemo class 2795
TRadioButton class 2865
TScrollBar class 2935
TValueListEditor class 4861
OnKeyPress
TButton class 2094
TCheckBox class 2171
TComboBox class 2253
TEdit class 2522
TListBox class 2723
TMemo class 2798
TRadioButton class 2868
TScrollBar class 2939
TValueListEditor class 4864
OnKeyUp
TButton class 2095
TCheckBox class 2172
TComboBox class 2254
TEdit class 2523
TListBox class 2724
TMemo class 2799
TRadioButton class 2869
TScrollBar class 2940
TValueListEditor class 4865
OnListen
TCustomSocket class 61
TServerSocket class 119
OnListening
TCustomTcpServer class 1416
TTcpServer class 1477
OnLogin
TCustomEndUserAdapter class 5686
TCustomLoginFormAdapter class 5715
TEndUserAdapter class 5775
TLoginFormAdapter class 5834
TSQLConnection class 1627
OnLogout
TCustomEndUserAdapter class 5687
TEndUserAdapter class 5775
OnLogTrace
TSQLMonitor class 1653
OnLookup
TClientSocket class 34
TCustomSocket class 62
OnMeasureItem
TComboBox class 2256
TCustomComboBox class 2338
TCustomListBox class 2442
TListBox class 2725
OnMeasureTab
TTabSet class 4527
OnMenuBar
DWebBrowserEvents2 interface 630
TInternetExplorer class 460
TWebBrowser class 562
OnMouseActivate
TButton class 2096
TCheckBox class 2174
TEdit class 2525
TGroupBox class 2590
TLabel class 2648
TListBox class 2727
TMemo class 2801

TRadioButton class 2871
TStaticText class 2988
TTabSet class 4528
TValueListEditor class 4867
OnMouseDown
TButton class 2097
TCheckBox class 2175
TEdit class 2526
TGroupBox class 2591
TLabel class 2649
TListBox class 2728
TMemo class 2802
TRadioButton class 2872
TStaticText class 2989
TTabSet class 4529
TValueListEditor class 4868
OnMouseEnter
TButton class 2100
TCheckBox class 2177
TComboBox class 2257
TEdit class 2528
TGroupBox class 2593
TLabel class 2651
TListBox class 2730
TMemo class 2804
TRadioButton class 2874
TScrollBar class 2942
TStaticText class 2991
TTabSet class 4531
TValueListEditor class 4870
OnMouseLeave
TButton class 2100
TCheckBox class 2178
TComboBox class 2257
TEdit class 2529
TGroupBox class 2594
TLabel class 2651
TListBox class 2731
TMemo class 2805
TRadioButton class 2874
TScrollBar class 2942
TStaticText class 2992
TTabSet class 4531
TValueListEditor class 4871
OnMouseMove
TButton class 2100
TCheckBox class 2178
TEdit class 2529
TGroupBox class 2594
TLabel class 2652
TListBox class 2731
TMemo class 2805
TRadioButton class 2875
TStaticText class 2992
TTabSet class 4532
TValueListEditor class 4871
OnMouseUp
TButton class 2103
TCheckBox class 2180
TEdit class 2531
TGroupBox class 2596
TLabel class 2654
TListBox class 2733
TMemo class 2807
TRadioButton class 2877
TStaticText class 2994
TTabSet class 4534
TValueListEditor class 4873
OnMouseWheelDown
TValueListEditor class 4875
OnMouseWheelUp
TValueListEditor class 4876
OnNavigate
IShellWindows interface 727
IShellWindowsDisp interface 731
TShellWindows class 501
OnNavigateComplete
TWebBrowser_V1 class 612

OnNavigateComplete2
TInternetExplorer class 461
TWebBrowser class 562
OnNavigateError
TInternetExplorer class 461
TWebBrowser class 563
OnNewRecord
TSimpleDataSet class 957
OnNewSearch
_SearchAssistantEvents interface 617
OnNewWindow
TWebBrowser_V1 class 613
OnNewWindow2
TInternetExplorer class 461
TWebBrowser class 563
OnNextMenuSelect
_SearchAssistantEvents interface 617
OnPageAccessDenied
TCustomPageDispatcher class 6200
TPageDispatcher class 6241
OnPageNotFound
TCustomPageDispatcher class 6200
TPageDispatcher class 6242
OnParentChanged
TWebContainedComponentHelper class 6052
OnPause
TService class 3088
OnPostError
TSimpleDataSet class 957
OnPostingData
THTTPReqResp class 1294
OnPrepareStrings
TCustomStringsValuesList class 5738
TStringsValuesList class 5864
OnPrintTemplateInstantiation
TInternetExplorer class 461
TWebBrowser class 563
OnPrintTemplateTeardown
TInternetExplorer class 461
TWebBrowser class 564
OnPrivacyImpactedStateChange
TInternetExplorer class 462
TWebBrowser class 564
OnProduceLoginPage
TBaseEndUserAdapter class 5516
TEndUserAdapter class 5776
TEndUserSessionAdapter class 5803
OnProgress
TCustomZStream class 7907
TZCompressionStream class 7912
TZDecompressionStream class 7915
OnProgressChange
TInternetExplorer class 462
TWebBrowser class 564
TWebBrowser_V1 class 613
OnPropertyChange
TInternetExplorer class 462
TWebBrowser class 565
TWebBrowser_V1 class 613
OnPublishingType
TWebServExp class 6733
TWSDLHTMLPublish class 7290
TWSDLPublish class 7294
OnQuit
DWebBrowserEvents2 interface 630
TInternetExplorer class 462
TWebBrowser class 565
TWebBrowser_V1 class 613
OnRead
TClientSocket class 34
TCustomSocket class 63
OnReceive
TBaseSocket class 1393
TTcpClient class 1456
TUdpSocket class 1496
OnReceivingData
THTTPReqResp class 1295
OnReconcileError

TSimpleDataSet class 958
OnRequestRecords
TXMLBroker class 7332
OnRequestUpdate
TXMLBroker class 7332
OnRowMoved
TValueListEditor class 4876
OnScreenTipsLoaded
TCustomScreenTipsManager class 316
TScreenTipsManager class 342
OnScroll
TScrollBar class 2942
OnSelect
TComboBox class 2257
TCustomCombo class 2322
OnSelectCell
TValueListEditor class 4877
OnSelectionChange
TShellNameSpace class 483
OnSend
TBaseSocket class 1394
TTcpClient class 1457
TUdpSocket class 1496
OnSendAttachment
IMimeAttachmentHandler interface 1183
TSoapPascalInvoker class 1358
OnSetDefaultComponentList
TWebContainerComponentHelper class 6075
OnSetEditText
TValueListEditor class 4879
OnSetSecureLockIcon
TInternetExplorer class 463
TWebBrowser class 565
OnShowScreenTip
TCustomScreenTipsManager class 316
TScreenTipsManager class 342
OnShutdown
TService class 3088
OnSocketEvent
TCustomWinSocket class 90
OnStart
TService class 3089
OnStartDock
TButton class 2105
TCheckBox class 2182
TComboBox class 2257
TEdit class 2533
TGroupBox class 2598
TLabel class 2656
TListBox class 2735
TMemo class 2809
TRadioButton class 2879
TScrollBar class 2944
TStaticText class 2996
TTabSet class 4536
TValueListEditor class 4879
OnStartDrag
TButton class 2105
TCheckBox class 2183
TComboBox class 2258
TEdit class 2534
TGroupBox class 2599
TLabel class 2657
TListBox class 2736
TMemo class 2810
TRadioButton class 2880
TScrollBar class 2944
TStaticText class 2997
TTabSet class 4537
TValueListEditor class 4880
TWebBrowser class 565
TWebBrowser_V1 class 613
OnStartSession
TCustomSessionsService class 6755
TSessionsService class 6759
OnStatusBar
DWebBrowserEvents2 interface 631
TInternetExplorer class 463

TWebBrowser class 566
OnStatusTextChange
TInternetExplorer class 463
TWebBrowser class 566
TWebBrowser_V1 class 614
OnStop
TService class 3089
OnStringsChange
TValueListEditor class 4881
OnStringsChanging
TValueListEditor class 4881
OnTheaterMode
DWebBrowserEvents2 interface 631
TInternetExplorer class 463
TWebBrowser class 567
OnThemeChange
TThemeServices class 4592
OnThreadEnd
TCustomServerSocket class 55
TServerSocket class 119
TServerWinSocket class 138
OnThreadStart
TCustomServerSocket class 55
TServerSocket class 120
TServerWinSocket class 138
OnTitleChange
TInternetExplorer class 463
TWebBrowser class 567
TWebBrowser_V1 class 614
OnToolBar
DWebBrowserEvents2 interface 631
TInternetExplorer class 464
TWebBrowser class 567
OnTopLeftChanged
TValueListEditor class 4882
OnTrace
TSQLMonitor class 1653
OnTranslate
TXMLTransform class 7739
OnUnDock
TGroupBox class 2600
OnUpdate
TColorSelect class 1782
TFileExit class 1816
TFileOpen class 1828
TFileOpenWith class 1835
TFilePageSetup class 1844
TFilePrintSetup class 1854
TFileSaveAs class 1863
TFontEdit class 1872
TPrintDlg class 1893
TSearchFind class 1908
TSearchFindNext class 1920
TSearchReplace class 1931
OnUpdatePageStatus
TInternetExplorer class 464
TWebBrowser class 567
OnUpdateValue
TAdapterBooleanField class 5230
TAdapterField class 5275
TAdapterMemoField class 5331
TCustomAdapterBooleanField class 5597
TCustomAdapterField class 5604
TCustomAdapterMemoField class 5645
OnUpdateValues
TAdapterMultiValueField class 5342
TCustomAdapterMultiValueField class 5656
OnUploadFile
TAdapterImageField class 5319
TAdapterMemoField class 5332
TCustomAdapterImageField class 5637
TCustomAdapterMemoField class 5645
OnUploadFiles
TAdapterFileField class 5290
TCustomAdapterFileField class 5613
OnUserIDNotFound
TCustomWebUserList class 6787
TWebUserList class 6799

OnValidate
TBrowseForFolder class 1774
TValueListEditor class 4882
OnValidateFile
TAdapterImageField class 5319
TCustomAdapterImageField class 5637
OnValidateFiles
TAdapterFileField class 5290
TCustomAdapterFileField class 5614
OnValidatePassword
TCustomLoginFormAdapter class 5716
TLoginFormAdapter class 5834
OnValidateUserError
TCustomWebUserList class 6787
TWebUserList class 6800
OnValidateUserName
TCustomLoginFormAdapter class 5716
TLoginFormAdapter class 5835
OnValidateValue
TAdapterBooleanField class 5231
TAdapterField class 5276
TAdapterUpdateField class 5433
OnValidateValues
TAdapterMultiValueField class 5342
TCustomAdapterMultiValueField class 5656
OnVisible
DWebBrowserEvents2 interface 631
TInternetExplorer class 464
TWebBrowser class 568
OnWindowActivate
TWebBrowser_V1 class 615
OnWindowClosing
TInternetExplorer class 464
TWebBrowser class 568
OnWindowMove
TWebBrowser_V1 class 615
OnWindowRegistered
TShellWindows class 503
OnWindowResize
TWebBrowser_V1 class 615
OnWindowRevoked
TShellWindows class 504
OnWindowSetHeight
TInternetExplorer class 465
TWebBrowser class 568
OnWindowSetLeft
TInternetExplorer class 465
TWebBrowser class 568
OnWindowSetResizable
TInternetExplorer class 465
TWebBrowser class 569
OnWindowSetTop
TInternetExplorer class 465
TWebBrowser class 569
OnWindowSetWidth
TInternetExplorer class 465
TWebBrowser class 569
OnWinInetError
THTTPReqResp class 1295
OnWrite
TClientSocket class 35
TCustomSocket class 63
Open
TAbstractSocket class 19
TBaseSocket class 1375
TCustomIpClient class 1404
TCustomTcpServer class 1411
TCustomWinSocket class 75
TThreadLocalCounter class 4024
OpenCursor
TCustomSQLDataSet class 1539
TSimpleDataSet class 902
TSQLTable class 1686
OpenSchema
TCustomSQLDataSet class 1539
TSQLConnection class 1601, 1602
Operation
IOperations interface 7130

TWSDLView class 7273
Operations
IPortType interface 7151
Options
IXMLDocument interface 7666
TCustomScreenTipsManager class 314
TScreenTipsManager class 341
TValueListEditor class 4822
TXMLDocument class 7389
otSByte enumeration member 4753
otSLong enumeration member 4753
otSWord enumeration member 4753
otUByte enumeration member 4753
otULong enumeration member 4753
otUWord enumeration member 4753
output
IDOMXSLProcessor interface 7610
Output
IBindingOperation interface 7087
IOperation interface 7127
Owner
TXMLNodeList class 7475
ownerDocument
IDOMNode interface 7575
OwnerDocument
IXMLNode interface 7695
TXMLNode class 7442
ownerElement
IDOMAttr interface 7524
OwnsObjects
TWideStringList class 6823

P

PackageName
TSQLStoredProc class 1677
PacketRecords
TSimpleDataSet class 923
Padding
TGroupBox class 2553
PageCount
TAbstractWebContext class 5973
TDefaultFieldsPagedAdapter class 5761
PageDescription
IWebPageInfo interface 6000
PageDispatcher
TCustomWebAppComponents class 6214
TWebAppComponents class 6248
PageElements
TBaseAdapterPageProducer class 1157
PageHREF
IWebPageInfo interface 6001
PageIndex
TTabbedNotebook class 4421
PageIsPublished
IWebPageInfo interface 6001
PageName
IWebPageInfo interface 6002
TAdapterActionButton class 6323
TCustomAdapterActionButton class 6441
PageNameOfRequest
IPageDispatcher interface 1063
TCustomPageDispatcher class 6194
PageProducer
TCustomWebPageModule class 6562
TWebAppPageModule class 6592
TWebPageModule class 6606
Pages
TAbstractWebContext class 5973
TSocketForm class 167
TTabbedNotebook class 4422
PageSize
TDefaultFieldsPagedAdapter class 5761
TPagedAdapter class 5851
TScrollBar class 2903
PageTitle
IWebPageInfo interface 6002
Paint
TCustomGroupBox class 2383

TCustomLabel class 2387
TCustomScreenTipsPopup class 320
TScreenTipsWindow class 356
TScroller class 4460
TShadowWindow class 403
TTabSet class 4479
PaintBorder
TThemeServices class 4590
Panel1
TSocketForm class 168
Param
TService class 3081
ParamByName
TCustomSQLDataSet class 1539
ParamCheck
TCustomSQLDataSet class 1561
TInternalSQLDataSet class 895
TSQLDataSet class 1641
TSQLQuery class 1661
TSQLStoredProc class 1678
ParamCount
TCustomSQLDataSet class 1562
TService class 3081
Params
IAdapterDispatchParams interface 991
TCustomSQLDataSet class 1562
TInternalSQLDataSet class 895
TSimpleDataSet class 924
TSQLConnection class 1621
TSQLDataSet class 1641
TSQLQuery class 1662
TSqlServerMethod class 1669
TSQLStoredProc class 1678
TXMLBroker class 7325
ParamsLoaded
TSQLConnection class 1622
ParamValues
IAdapterItemRequestParams interface 1009
TAdapterItemRequestParams class 6165
Parent
IWebBrowser interface 748
IWebBrowser2Disp interface 772
IWebBrowserAppDisp interface 793
IWebBrowserDisp interface 800
TInternetExplorer class 454
TWebBrowser class 531
TWebBrowser_V1 class 585
ParentBackground
TGroupBox class 2553
TTabSet class 4496
ParentBiDiMode
TButton class 2060
TCheckBox class 2138
TComboBox class 2212
TEdit class 2487
TGroupBox class 2554
TLabel class 2625
TListBox class 2684
TMemo class 2760
TRadioButton class 2835
TScrollBar class 2905
TStaticText class 2965
TValueListEditor class 4824
ParentChanged
TAdapterNamedField class 5366
TBaseAdapterAction class 5472
TWebContainedComponent class 6042
TWebContainedComponentHelper class 6049
TWebContainedContainerComponent class 6058
ParentColor
TCheckBox class 2138
TComboBox class 2213
TCustomCombo class 2311
TCustomEdit class 2370
TCustomListBox class 2431
TEdit class 2487
TGroupBox class 2554
TLabel class 2626

TListBox class 2684
TMemo class 2760
TRadioButton class 2835
TStaticText class 2966
TValueListEditor class 4824
ParentComponent
TWebComponentList class 6034
ParentCtl3D
TCheckBox class 2139
TComboBox class 2213
TEdit class 2488
TGroupBox class 2555
TListBox class 2685
TMemo class 2761
TRadioButton class 2836
TScrollBar class 2905
TValueListEditor class 4825
ParentDoubleBuffered
TButton class 2060
TCheckBox class 2139
TComboBox class 2214
TEdit class 2488
TGroupBox class 2555
TListBox class 2685
TMemo class 2761
TRadioButton class 2836
TScrollBar class 2906
TStaticText class 2966
TTabbedNotebook class 4422
TTabSet class 4496
TValueListEditor class 4825
ParentFont
TButton class 2061
TCheckBox class 2139
TComboBox class 2214
TEdit class 2488
TGroupBox class 2555
TLabel class 2626
TListBox class 2685
TMemo class 2761
TRadioButton class 2837
TStaticText class 2966
TValueListEditor class 4825
ParentLayout
TLayoutWebContent class 6022
parentNode
IDOMNode interface 7575
ParentNode
IXMLNode interface 7695
TXMLNode class 7443
ParentShowHint
TButton class 2061
TCheckBox class 2140
TComboBox class 2214
TEdit class 2489
TGroupBox class 2556
TLabel class 2627
TListBox class 2686
TMemo class 2762
TRadioButton class 2837
TScrollBar class 2906
TStaticText class 2967
TTabbedNotebook class 4423
TTabSet class 4496
TValueListEditor class 4826
TWebBrowser class 532
TWebBrowser_V1 class 586
ParseDeleteSql
TCustomSQLDataSet class 1569
ParseError
EDOMParseError class 7512
ParseInsertSql
TCustomSQLDataSet class 1570
ParseOptions
IXMLDocument interface 7667
TXMLDocument class 7390
ParseSelectSql
TCustomSQLDataSet class 1570

ParseStream
TScriptFile class 6685
TScriptProducer class 6712
ParseString
TScriptFile class 6685
TScriptProducer class 6712
ParseUpdateSql
TCustomSQLDataSet class 1570
Part
IHeaderInfo interface 7252
IParts interface 7142
Parts
IMessage interface 7119
Password
TCustomLoginFormAdapter class 5714
THTTPReqResp class 1289
TService class 3081
TSoapConnection class 1197
TWebConnection class 263
TWebUserItem class 6791
TWSDLView class 7274
PasswordChar
TCustomEdit class 2370
TEdit class 2489
PasswordRequired
TCustomLoginFormAdapter class 5714
TLoginFormAdapter class 5831
PasteFromClipboard
TCustomEdit class 2358
Path
IWebBrowser2Disp interface 772
IWebBrowserApp interface 784
IWebBrowserAppDisp interface 793
TInternetExplorer class 455
TWebBrowser class 532
PeekBuf
TBaseSocket class 1376
pfAddress enumeration member 4756
pfAppleTalk enumeration member 1500
pfArray enumeration member 4756
pfBan enumeration member 1500
pfCcitt enumeration member 1500
pfChaos enumeration member 1500
pfConst enumeration member 4756
pfDataKit enumeration member 1500
pfDecNet enumeration member 1500
pfDli enumeration member 1500
pfEcma enumeration member 1500
pfFireFox enumeration member 1500
pfHylink enumeration member 1500
pfImpLink enumeration member 1500
pfInet enumeration member 1500
pfIpx enumeration member 1500
pfIso enumeration member 1500
pfLat enumeration member 1500
pfMax enumeration member 1500
pfNs enumeration member 1500
pfOsi enumeration member 1500
pfOut enumeration member 4756
pfPup enumeration member 1500
pfReference enumeration member 4756
pfSna enumeration member 1500
pfUnix enumeration member 1500
pfUnknown1 enumeration member 1500
pfUnspec enumeration member 1500
pfVar enumeration member 4756
pfVoiceView enumeration member 1500
PickList
TItemProp class 4783
poAsyncLoad enumeration member 7718
poDefault enumeration member 7298
poPreserveWhiteSpace enumeration member 7718
poPublishLocationAsSecure enumeration member 7298
PopupMenu
TButton class 2062
TCheckBox class 2140
TComboBox class 2215
TEdit class 2490

TGroupBox class 2556
TLabel class 2627
TListBox class 2686
TMemo class 2762
TRadioButton class 2838
TScrollBar class 2907
TSocketForm class 168
TStaticText class 2968
TTabbedNotebook class 4423
TTabSet class 4497
TValueListEditor class 4826
TWebBrowser class 533
TWebBrowser_V1 class 587
PopupTip
TAssociateItem class 302
PopupType
TCustomScreenTipsPopup class 321
TScreenTipsPopup class 348
poResolveExternals enumeration member 7718
Port
IPorts interface 7148
TAbstractSocket class 23
TClientSocket class 30
THTTPRIO class 1253
TServerSocket class 111
TSocketConnection class 228
TSocketTransport class 235
TWSDLView class 7274
PortDesc
TSocketForm class 168
PortGroup
TSocketForm class 168
PortList
TSocketForm class 169
PortListClick
TSocketForm class 178
PortNo
TSocketForm class 169
Ports
IService interface 7158
PortType
IPortTypes interface 7155
TWSDLView class 7274
PortTypes
IDefinition interface 7099
PortUpDown
TSocketForm class 169
Pos
AnsiStringBase class 3198
AnsiStringT template 3226
UnicodeString class 3406
WideString class 3482
Position
TScrollBar class 2912
TScroller class 4462
PositionPopup
TStandardCustomizePopup class 1747
TStandardMenuPopup class 1758
TThemedPopupMenu class 4574
TXPStylePopupMenu class 7781
poUTF8ContentType enumeration member 7298
poValidateOnParse enumeration member 7718
prefix
IDOMNode interface 7576
Prefix
IXMLNode interface 7696
PrefixID
TXMLDocument class 7391
Prepared
TCustomSQLDataSet class 1563
PrepareStatement
TCustomSQLDataSet class 1540
TSQLQuery class 1659
TSQLStoredProc class 1676
TSQLTable class 1686
PrepareStrings
TCustomStringsValuesList class 5737
preserveWhiteSpace

IDOMParseOptions interface 7594
PressedImageIndex
TButton class 2067
TCustomButton class 2274
PreventDestruction
TBaseWebModuleFactory class 6284
TBaseWebPageModuleFactory class 6294
previousSibling
IDOMNode interface 7576
PreviousSibling
IXMLNode interface 7683
TXMLNode class 7430
PrevValue
IWebGetFieldPrevValue interface 5895
printf
AnsiStringT template 3226
UnicodeString class 3406
WideString class 3482
PrintTemplateInstantiation
DWebBrowserEvents2 interface 631
PrintTemplateTeardown
DWebBrowserEvents2 interface 632
PrivacyImpactedStateChange
DWebBrowserEvents2 interface 632
ProcessAttachDetach
IShellWindows interface 727
IShellWindowsDisp interface 731
TShellWindows class 502
ProcessHeader
IDOMHeaderProcessor interface 1242
ProcessMultiPartForm
IMimeAttachmentHandler interface 1180
ProcessRequests
TServiceThread class 3100
ProcParams
TCustomSQLDataSet class 1564
ProduceContent
TCustomXSLPageProducer class 7890
ProduceContentFromStream
TCustomXSLPageProducer class 7890
ProduceContentFromString
TCustomXSLPageProducer class 7890
ProduceLoginPage
IWebEndUser interface 5993
TBaseEndUserAdapter class 5515
TCustomPageDispatcher class 6194
Producer
TScriptGlobalObjects class 6691
ProgressChange
DWebBrowserEvents interface 624
DWebBrowserEvents2 interface 632
Properties
TConnectionData class 1520
PropertyChange
DWebBrowserEvents interface 624
DWebBrowserEvents2 interface 632
PropertyChanged
TCustomSQLDataSet class 1540
PropPage
TSocketForm class 169
Protocol
TBaseSocket class 1387
ProviderCount
TLocalConnection class 4554
TSoapDataModule class 1230
ProviderName
TXMLBroker class 7326
TXMLTransformClient class 7745
Providers
TLocalConnection class 4554
TSoapDataModule class 1230
Proxy
THTTPReqResp class 1289
TSoapConnection class 1197
TWebConnection class 263
TWSDLView class 7275
ProxyByPass
THTTPReqResp class 1290

TSoapConnection class 1197
TWebConnection class 263
PSEndTransaction
TCustomSQLDataSet class 1540
PSExecute
TCustomSQLDataSet class 1541
PSExecuteStatement
TCustomSQLDataSet class 1541
PSGetAttributes
TCustomSQLDataSet class 1541
PSGetCommandText
TCustomSQLDataSet class 1542
TSimpleDataSet class 902
PSGetCommandType
TCustomSQLDataSet class 1542
PSGetDefaultOrder
TCustomSQLDataSet class 1542
PSGetIndexDefs
TCustomSQLDataSet class 1543
PSGetKeyFieldsW
TCustomSQLDataSet class 1543
PSGetParams
TCustomSQLDataSet class 1544
PSGetQuoteCharW
TCustomSQLDataSet class 1544
PSGetTableNameW
TCustomSQLDataSet class 1544
PSGetUpdateException
TCustomSQLDataSet class 1544
PSInTransaction
TCustomSQLDataSet class 1545
PSIsSQLBased
TCustomSQLDataSet class 1545
PSIsSQLSupported
TCustomSQLDataSet class 1546
PSReset
TCustomSQLDataSet class 1546
PSSetCommandText
TCustomSQLDataSet class 1546
PSSetParams
TCustomSQLDataSet class 1547
PSStartTransaction
TCustomSQLDataSet class 1547
PSUpdateRecord
TCustomSQLDataSet class 1548
ptCustom enumeration member 365
ptHelp enumeration member 365
ptInfo enumeration member 365
publicId
IDOMDocumentType interface 7542
IDOMEntity interface 7551
IDOMNotation interface 7586
PublishOptions
TWSDLHTMLPublish class 7288
Pulse
TMonitor record 3352
PulseAll
TMonitor record 3352
Put
TAbstractNamedVariants class 6129
TNamedVariantsList class 6139
TTabList class 4470
TValueListStrings class 4890
TWideStringList class 6820
TWideStrings class 6843
PutElement
Variant class 3435
PutFindText
ISearchAssistantOC interface 653
ISearchAssistantOC2Disp interface 663
ISearchAssistantOC3Disp interface 674
ISearchAssistantOCDisp interface 682
PutObject
TCustomComboBoxStrings class 2343
TWideStringList class 6820
TWideStrings class 6844
PutProperty
ISearchAssistantOC interface 653

ISearchAssistantOC2Disp interface 663
ISearchAssistantOC3Disp interface 674
ISearchAssistantOCDisp interface 682
IWebBrowser2Disp interface 767
IWebBrowserApp interface 781
IWebBrowserAppDisp interface 788
TInternetExplorer class 442
TWebBrowser class 515

Q

qaNoModuleName enumeration member 1126
qaNoModulesVar enumeration member 1126
qaUseModulesVar enumeration member 1126
QueryInterface
IInterface interface 3504
TAggregatedObject class 3317
TContainedObject class 3322
TCustomVariantType class 5002
TCustomWebAppDataModule class 6543
TCustomWebAppPageModule class 6548
THTTPRIO class 1251
TInterfacedObject class 3347
TLayoutWebContent class 6021
TLinkedRIO class 1319
TStreamedConnection class 243
TWebComponentList class 6032
TWebComponentsEditorHelper class 6038
QueryStatusWB
IWebBrowser2 interface 757
IWebBrowser2Disp interface 767
TInternetExplorer class 442
TWebBrowser class 516
Quit
DWebBrowserEvents interface 624
IWebBrowser2Disp interface 767
IWebBrowserApp interface 781
IWebBrowserAppDisp interface 788
TInternetExplorer class 442
TWebBrowser class 516
QuoteChar
TSQLConnection class 1623
TWideStrings class 6852

R

RaiseBlankPassword
TCustomLoginFormAdapter class 5711
RaiseBlankUserName
TCustomLoginFormAdapter class 5711
RaiseCastError
TCustomVariantType class 5002
RaiseDispError
TCustomVariantType class 5003
RaiseInvalidOp
TCustomVariantType class 5003
RaiseOuterException
Exception class 3962
RaisingException
EHeapException class 3929
Exception class 3962
reAccessViolation enumeration member 3734
ReActivate
TServerClientThread class 101
Read
IDataBlock interface 270
TCustomWinSocket class 76
TDataBlock class 203
TWinSocketStream class 141
TZCompressionStream class 7910
TZDecompressionStream class 7914
ReadBlobData
TSQLBlobStream class 1580
ReadDOMVendor
TXMLDocument class 7371
ReadOnly
IXMLNode interface 7696
TAdapterBooleanField class 5229
TAdapterField class 5273
TAdapterFileField class 5288

TAdapterImageField class 5316
TAdapterMemoField class 5330
TAdapterMultiValueField class 5339
TAdapterNamedDisplayField class 5351
TCustomAdapterImageField class 5635
TCustomEdit class 2371
TEdit class 2495
TItemProp class 4784
TMemo class 2767
TSimpleDataSet class 929
ReadSessionID
ISessionsServiceStoreID interface 1072
ReadSettings
TSocketForm class 178
ReadState
TAdapterSubComponent class 5416
TTabbedNotebook class 4411
TTabPage class 4445
TWebContainedComponent class 6043
TWebContainedContainerComponent class 6059
ReadyState
IWebBrowser2 interface 760
IWebBrowser2Disp interface 773
TInternetExplorer class 455
TWebBrowser class 538
reason
IDOMParseError interface 7590
Reason
EDOMParseError class 7513
reAssertionFailed enumeration member 3734
Receive
ITransport interface 280
THTTPReqResp class 1285
TSocketTransport class 233
TWebConnection class 259
ReceiveBuf
TBaseSocket class 1376
TCustomWinSocket class 77
ReceiveFrom
TIpSocket class 1422
ReceiveLength
TCustomWinSocket class 77
ReceiveIn
TBaseSocket class 1379
ReceiveText
TCustomWinSocket class 78
ReceiveTimeout
THTTPReqResp class 1290
ReconcileProducer
TXMLBroker class 7327
reControlBreak enumeration member 3734
RecordChanged
TSQLDataLink class 1631
RecordCount
TCustomSQLDataSet class 1564
TDefaultFieldsPagedAdapter class 5761
RecordIndex
TDefaultFieldsPagedAdapter class 5762
RedirectOptions
TAdapterAction class 5206
TAdapterGotoPageAction class 5296
TAdapterNextPageAction class 5377
TAdapterPrevPageAction class 5403
TBaseAdapterAction class 5476
RedirectToPage
IPageResult interface 1066
TSitePageModuleHelper class 6582
RedirectToPageName
IPageDispatcher interface 1063
TCustomPageDispatcher class 6194
reDivByZero enumeration member 3734
reExternalException enumeration member 3734
RefCount
AnsiStringBase class 3198
AnsiStringT template 3227
TInterfacedObject class 3347
TSoapDataModule class 1231
UnicodeString class 3407

ReferenceStrings
IWideStringsAdapter interface 6860
Refresh
IWebBrowser interface 742
IWebBrowser2Disp interface 767
IWebBrowserAppDisp interface 789
IWebBrowserDisp interface 797
IXMLDocument interface 7654
TInternetExplorer class 442
TValueListEditor class 4798
TWebBrowser class 516
TWebBrowser_V1 class 575
TXMLDocument class 7371
Refresh2
IWebBrowser interface 742
IWebBrowser2Disp interface 767
IWebBrowserAppDisp interface 789
IWebBrowserDisp interface 797
TInternetExplorer class 443
TWebBrowser class 517
TWebBrowser_V1 class 576
RefreshOfflineDesktop
IShellUIHelper interface 720
IShellUIHelperDisp interface 724
TShellUIHelper class 493
RefreshProperties
TConnectionData class 1518
Register
IShellWindows interface 728
IShellWindowsDisp interface 731
TShellWindows class 502
RegisterAsBrowser
IWebBrowser2 interface 761
IWebBrowser2Disp interface 773
TInternetExplorer class 455
TWebBrowser class 538
RegisterAsDropTarget
IWebBrowser2 interface 761
IWebBrowser2Disp interface 773
TInternetExplorer class 455
TWebBrowser class 539
RegisterChildNode
IXMLNodeAccess interface 7486
TXMLNode class 7430
RegisterChildNodes
IXMLNodeAccess interface 7486
TXMLNode class 7431
RegisterClient
TSQLConnection class 1602
RegisterDocBinding
IXMLDocument interface 7654
TXMLDocument class 7372
RegisteredAction
TSocketForm class 169
RegisteredActionExecute
TSocketForm class 178
RegisterPending
IShellWindows interface 728
IShellWindowsDisp interface 732
TShellWindows class 502
RegisterProvider
TLocalConnection class 4552
TSoapDataModule class 1222
RegisterServices
TServiceApplication class 3093
reIntfCastError enumeration member 3734
reIntOverflow enumeration member 3734
reInvalidCast enumeration member 3734
reInvalidOp enumeration member 3734
reInvalidPtr enumeration member 3734
Release
DelphiInterface template 3258
TConditionVariableCS class 3115
TConditionVariableMutex class 3124
TCriticalSection class 3129
TMutex class 3146
TSemaphore class 3151
TSynchroObject class 3154

ReleaseAll
TConditionVariableCS class 3116
TConditionVariableMutex class 3124
ReleaseDoc
TXMLDocument class 7372
ReleaseObject
TCustomDataBlockInterpreter class 196
TDataBlockInterpreter class 214
ReleaseStrings
IWideStringsAdapter interface 6860
ReloadProperties
TConnectionData class 1518
reMonitorNotLocked enumeration member 3734
RemoteAddr
TCustomWinSocket class 87
RemoteAddress
TCustomWinSocket class 87
RemoteHost
TCustomWinSocket class 88
TIpSocket class 1426
TTcpClient class 1446
TUdpSocket class 1486
RemotePort
TCustomWinSocket class 88
TIpSocket class 1428
TTcpClient class 1448
TUdpSocket class 1488
RemoteServer
TXMLBroker class 7328
TXMLTransformClient class 7745
Remove
IXMLNodeCollection interface 7702
IXMLNodeList interface 7713
TAssociateItemList class 304
TDOMVendorList class 7519
TSoapDataList class 1174
TStringBuilder class 4019
TWebComponentList class 6032
TXMLNodeCollection class 7456
TXMLNodeList class 7473
removeAttribute
IDOMElement interface 7547
removeAttributeNode
IDOMElement interface 7547
removeAttributeNS
IDOMElement interface 7547
removeChild
IDOMNode interface 7566
RemoveClient
TSocketForm class 179
RemoveClientSocketThread
TServerSocketThread class 1437
RemoveConnectNotification
TSQLConnection class 1602
RemoveDispatch
TCustomDataBlockInterpreter class 196
TDataBlockInterpreter class 214
RemoveEngine
TScriptEnginesList class 6676
RemoveHostedNode
TXMLNode class 7431
RemoveLinkToManager
TCustomScreenTipsPopup class 320
removeNamedItem
IDOMNamedNodeMap interface 7556
removeNamedItemNS
IDOMNamedNodeMap interface 7556
RemoveNotify
INotifyList interface 1059
TAdapterControlGroup class 6351
TNotifyList class 971
TXMLBroker class 7316
RemovePopup
TCustomScreenTipsManager class 311
RemovePortAction
TSocketForm class 170
RemovePortActionExecute
TSocketForm class 179

RemovePortActionUpdate
TSocketForm class 179
RemoveScreenTip
TCustomScreenTipsManager class 312
RenderAdapterImage
TCustomAdapterImageField class 5633
reNoMonitorSupport enumeration member 3734
reNone enumeration member 3734
reOutOfMemory enumeration member 3734
reOverflow enumeration member 3734
Replace
TStringBuilder class 4020
replaceChild
IDOMNode interface 7566
replaceData
IDOMCharacterData interface 7528
ReplaceNode
IXMLNodeList interface 7713
TXMLNodeList class 7473
ReplaceTags
TScriptProducer class 6713
ReportStatus
TService class 3077
rePrivInstruction enumeration member 3734
ReqFile
TLogLinkedWebNode class 1329
Request
TAbstractAdapterRequest class 5957
TAbstractWebContext class 5974
TCustomWebDataModule class 6554
TMultiModuleWebAppServices class 6234
TSiteModuleHelper class 6571
RequestedPageName
TAbstractWebContext class 5974
RequestIdentifier
IAdapterDispatchParams interface 991
TAdapterDispatchParams class 6160
RequestRecords
TXMLBroker class 7316
RequestUpdate
TXMLBroker class 7317
Required
IAdapterFieldAttributes interface 1001
IHeaderInfo interface 7252
TAdapterField class 5273
TAdapterFileField class 5289
TAdapterImageField class 5316
TAdapterMemoField class 5330
TAdapterMultiValueField class 5340
TAdapterNamedDisplayField class 5351
RequiresMultipartForm
TAbstractControlRequirements class 6314
TControlRequirements class 6429
reRangeError enumeration member 3734
reSafeCallError enumeration member 3734
reset
IDOMXSLProcessor interface 7609
ResetContent
TCustomListBox class 2416
ResetEvent
TEvent class 3136
ResetFirstBootMode
IShellUIHelper interface 720
IShellUIHelperDisp interface 724
TShellUIHelper class 493
ResetNextMenu
ISearchAssistantOC interface 653
ISearchAssistantOC2Disp interface 663
ISearchAssistantOC3Disp interface 674
ISearchAssistantOCDisp interface 682
ResetSafeMode
IShellUIHelper interface 720
IShellUIHelperDisp interface 724
TShellUIHelper class 494
ResetSort
IShellFavoritesNameSpace interface 694
IShellFavoritesNameSpaceDisp interface 699
IShellNameSpaceDisp interface 713

TShellNameSpace class 476
Resizable
IWebBrowser2 interface 762
IWebBrowser2Disp interface 773
TInternetExplorer class 455
TWebBrowser class 539
Resize
TValueListEditor class 4798
resolveExternals
IDOMParseOptions interface 7594
RespFile
TLogLinkedWebNode class 1329
Response
TAbstractAdapterResponse class 5960
TAbstractWebContext class 5975
TCustomWebDataModule class 6554
TMultiModuleWebAppServices class 6234
TSiteModuleHelper class 6572
reStackOverflow enumeration member 3734
RestoreCurrentRow
TValueListEditor class 4798
RestoreDefaultAdapterMode
IAdapterMode interface 1011
RestoreDefaults
TAdapterDataDisplay class 6361
TCustomAdapterActionButton class 6436
ResultDocument
TXMLTransform class 7733
ResultString
TXMLTransform class 7733
Resync
IXMLDocument interface 7655
IXMLNode interface 7684
TXMLDocument class 7372
TXMLNode class 7432
retreatError
IScriptErrorList interface 639
IScriptErrorListDisp interface 643
reUnderflow enumeration member 3734
reVarArrayBounds enumeration member 3734
reVarArrayCreate enumeration member 3734
reVarDispatch enumeration member 3734
reVarInvalidOp enumeration member 3734
reVarNotArray enumeration member 3734
reVarTypeCast enumeration member 3734
RevisionLevel
TMultiReadExclusiveWriteSynchronizer class 4002
Revoke
IShellWindows interface 728
IShellWindowsDisp interface 732
TShellWindows class 502
reZeroDivide enumeration member 3734
Right
TImageMargins class 2606
RightPromotion
TCustomVariantType class 5004
RIO
TSoapConnection class 1198
rnActivateModule enumeration member 6643
rnCreateModule enumeration member 6643
rnDeactivateModule enumeration member 6643
rnFinishRequest enumeration member 6643
rnFreeModule enumeration member 6643
rnStartRequest enumeration member 6643
roExcludeID enumeration member 1125
Rollback
TSQLConnection class 1603
RollbackFreeAndNil
TSQLConnection class 1603
RollbackIncompleteFreeAndNil
TSQLConnection class 1604
Root
IShellNameSpace interface 708
IShellNameSpaceDisp interface 716
TShellNameSpace class 481
RootDir
TBrowseForFolder class 1770
RowAttributes

TAdapterGrid class 6424
TCustomAdapterGrid class 6503
RowCount
TValueListEditor class 4831
RowMoved
TValueListEditor class 4799
RowRequest
TSoapDataModule class 1222
RowsAffected
TCustomSQLDataSet class 1564
TSQLQuery class 1663
RowSetVarName
TXMLBroker class 7318
Run
TServiceApplication class 3094
TWebApplication class 5939

S

SafeCallException
TLocalConnection class 4553
TObject class 3380
TSoapDataModule class 1223
SampleImageFile
IInternetEnvOptions interface 1045
Sanitized
Variant class 3435
SAS_ApplyUpdates
IAppServerSOAP interface 1341
TSoapDataModule class 1223
SAS_DataRequest
IAppServerSOAP interface 1342
TSoapDataModule class 1224
SAS_Execute
IAppServerSOAP interface 1342
TSoapDataModule class 1225
SAS_GetParams
IAppServerSOAP interface 1343
TSoapDataModule class 1226
SAS_GetProviderNames
IAppServerSOAP interface 1344
TSoapDataModule class 1226
SAS_GetRecords
IAppServerSOAP interface 1344
TSoapDataModule class 1227
SAS_RowRequest
IAppServerSOAP interface 1345
TSoapDataModule class 1228
save
IDOMPersist interface 7597
SaveRecreateItems
TCustomListBox class 2416
SaveToFile
ISessionsServicePersist interface 1071
IXMLDocument interface 7655
TCustomSessionsService class 6753
TCustomWebUserList class 6784
TSessions class 392
TSQLMonitor class 1646
TWideStrings class 6844
TXMLDocument class 7373
saveToStream
IDOMPersist interface 7598
SaveToStream
ISessionsServicePersist interface 1071
IXMLDocument interface 7656
TCustomSessionsService class 6753
TSessions class 392
TWideStrings class 6845
TXMLDocument class 7373
SaveToUTF8String
TXMLDocument class 7374
SaveToXML
IXMLDocument interface 7656
TXMLDocument class 7374
SaveToXMLStrings
TXMLDocument class 7375
sbLeft enumeration member 4539
sbRight enumeration member 4539

sbsNone enumeration member 3006
sbsSingle enumeration member 3006
sbsSunken enumeration member 3006
scBottom enumeration member 3004
scEndScroll enumeration member 3004
SchemaDef
IXMLSchemaDefs interface 7170
SchemaDefs
ITypes interface 7163
SchemaInfo
TCustomSQLDataSet class 1565
SchemaName
TCustomSQLDataSet class 1565
SchemaRef
IXMLDocument interface 7667
TXMLDocument class 7391
SocktCnst 1
SocktCnst namespace 1
Constants 2
SocktCnst.ckInterceptGUID 2
SocktCnst.ckInterceptGUID constant 2
SocktCnst.ckPort 3
SocktCnst.ckPort constant 3
SocktCnst.ckRegistered 3
SocktCnst.ckRegistered constant 3
SocktCnst.ckShowHost 3
SocktCnst.ckShowHost constant 3
SocktCnst.ckThreadCacheSize 4
SocktCnst.ckThreadCacheSize constant 4
SocktCnst.ckTimeout 4
SocktCnst.ckTimeout constant 4
SocktCnst.csSettings 4
SocktCnst.csSettings constant 4
SocktCnst.KEY_IE 5
SocktCnst.KEY_IE constant 5
SocktCnst.KEY_SOCKETSERVER 5
SocktCnst.KEY_SOCKETSERVER constant 5
SocktCnst.SAlreadyRunning 5
SocktCnst.SAlreadyRunning constant 5
SocktCnst.SApplicationName 6
SocktCnst.SApplicationName constant 6
SocktCnst.SErrChangeSettings 6
SocktCnst.SErrChangeSettings constant 6
SocktCnst.SErrClose 6
SocktCnst.SErrClose constant 6
SocktCnst.SHostUnknown 7
SocktCnst.SHostUnknown constant 7
SocktCnst.SNotShown 7
SocktCnst.SNotShown constant 7
SocktCnst.SNotUntilRestart 7
SocktCnst.SNotUntilRestart constant 7
SocktCnst.SNoWinSock2 8
SocktCnst.SNoWinSock2 constant 8
SocktCnst.SOpenError 8
SocktCnst.SOpenError constant 8
SocktCnst.SQueryDisconnect 8
SocktCnst.SQueryDisconnect constant 8
SocktCnst.SServiceName 9
SocktCnst.SServiceName constant 9
SocktCnst.SServiceOnly 9
SocktCnst.SServiceOnly constant 9
SocktCnst.SStatusline 9
SocktCnst.SStatusline constant 9
SocktComp 11
SocktComp namespace 11
Classes 13
Constants 154
Functions 144
Structs, Records, Enums 145
Types 150
SocktComp.CM_DEFERFREE 154
SocktComp.CM_DEFERFREE constant 154
SocktComp.CM_LOOKUPCOMPLETE 154
SocktComp.CM_LOOKUPCOMPLETE constant 154
SocktComp.CM_SOCKETMESSAGE 154
SocktComp.CM_SOCKETMESSAGE constant 154
SocktComp.ESocketError 13
SocktComp.SetErrorProc 144

ScktComp.SetErrorProc function 144
ScktComp.TAbstractSocket 14
ScktComp.TAbstractSocket.Active 21
ScktComp.TAbstractSocket.Address 22
ScktComp.TAbstractSocket.Close 16
ScktComp.TAbstractSocket.DoActivate 16
ScktComp.TAbstractSocket.Error 16
ScktComp.TAbstractSocket.Event 17
ScktComp.TAbstractSocket.Host 22
ScktComp.TAbstractSocket.InitSocket 18
ScktComp.TAbstractSocket.Loaded 18
ScktComp.TAbstractSocket.Open 19
ScktComp.TAbstractSocket.Port 23
ScktComp.TAbstractSocket.Service 23
ScktComp.TAbstractSocket.SetActive 19
ScktComp.TAbstractSocket.SetAddress 20
ScktComp.TAbstractSocket.SetHost 20
ScktComp.TAbstractSocket.SetPort 20
ScktComp.TAbstractSocket.SetService 21
ScktComp.TAsyncStyle 145
ScktComp.TAsyncStyle enumeration 145
ScktComp.TAsyncStyles 150
ScktComp.TAsyncStyles type 150
ScktComp.TClientSocket 24
ScktComp.TClientSocket.Active 28
ScktComp.TClientSocket.Address 28
ScktComp.TClientSocket.ClientType 29
ScktComp.TClientSocket.Create 26
ScktComp.TClientSocket.Destroy 26
ScktComp.TClientSocket.DoActivate 27
ScktComp.TClientSocket.GetClientType 27
ScktComp.TClientSocket.Host 30
ScktComp.TClientSocket.OnConnect 32
ScktComp.TClientSocket.OnConnecting 32
ScktComp.TClientSocket.OnDisconnect 33
ScktComp.TClientSocket.OnError 33
ScktComp.TClientSocket.OnLookup 34
ScktComp.TClientSocket.OnRead 34
ScktComp.TClientSocket.OnWrite 35
ScktComp.TClientSocket.Port 30
ScktComp.TClientSocket.Service 31
ScktComp.TClientSocket.SetClientType 27
ScktComp.TClientSocket.Socket 31
ScktComp.TClientType 146
ScktComp.TClientType enumeration 146
ScktComp.TClientWinSocket 35
ScktComp.TClientWinSocket.ClientType 38
ScktComp.TClientWinSocket.Connect 38
ScktComp.TClientWinSocket.SetClientType 38
ScktComp.TCMLookupComplete 146
ScktComp.TCMLookupComplete record 146
ScktComp.TCMSocketMessage 147
ScktComp.TCMSocketMessage record 147
ScktComp.TCustomServerSocket 39
ScktComp.TCustomServerSocket.Destroy 41
ScktComp.TCustomServerSocket.DoActivate 42
ScktComp.TCustomServerSocket.FServerSocket 42
ScktComp.TCustomServerSocket.GetGetSocketEvent 43
ScktComp.TCustomServerSocket.GetGetThreadEvent 43
ScktComp.TCustomServerSocket.GetOnClientError 44
ScktComp.TCustomServerSocket.GetOnClientEvent 44
ScktComp.TCustomServerSocket.GetOnThreadEnd 44
ScktComp.TCustomServerSocket.GetOnThreadStart 45
ScktComp.TCustomServerSocket.GetServerType 45
ScktComp.TCustomServerSocket.GetThreadCacheSize 45
ScktComp.TCustomServerSocket.OnClientConnect 51
ScktComp.TCustomServerSocket.OnClientDisconnect 52
ScktComp.TCustomServerSocket.OnClientError 52
ScktComp.TCustomServerSocket.OnClientRead 53
ScktComp.TCustomServerSocket.OnClientWrite 53
ScktComp.TCustomServerSocket.OnGetSocket 54
ScktComp.TCustomServerSocket.OnGetThread 54
ScktComp.TCustomServerSocket.OnThreadEnd 55
ScktComp.TCustomServerSocket.OnThreadStart 55
ScktComp.TCustomServerSocket.ServerType 49
ScktComp.TCustomServerSocket.SetGetSocketEvent 46
ScktComp.TCustomServerSocket.SetGetThreadEvent 46
ScktComp.TCustomServerSocket.SetOnClientError 47

ScktComp.TCustomServerSocket.SetOnClientEvent 47
ScktComp.TCustomServerSocket.SetOnThreadEnd 47
ScktComp.TCustomServerSocket.SetOnThreadStart 48
ScktComp.TCustomServerSocket.SetServerType 48
ScktComp.TCustomServerSocket.SetThreadCacheSize 49
ScktComp.TCustomServerSocket.ThreadCacheSize 50
ScktComp.TCustomSocket 56
ScktComp.TCustomSocket.Error 58
ScktComp.TCustomSocket.Event 58
ScktComp.TCustomSocket.OnAccept 59
ScktComp.TCustomSocket.OnConnect 60
ScktComp.TCustomSocket.OnConnecting 60
ScktComp.TCustomSocket.OnDisconnect 61
ScktComp.TCustomSocket.OnError 61
ScktComp.TCustomSocket.OnListen 61
ScktComp.TCustomSocket.OnLookup 62
ScktComp.TCustomSocket.OnRead 63
ScktComp.TCustomSocket.OnWrite 63
ScktComp.TCustomWinSocket 64
ScktComp.TCustomWinSocket.Accept 68
ScktComp.TCustomWinSocket.Addr 82
ScktComp.TCustomWinSocket.AsyncInitSocket 68
ScktComp.TCustomWinSocket.ASyncStyles 83
ScktComp.TCustomWinSocket.Close 69
ScktComp.TCustomWinSocket.Connect 69
ScktComp.TCustomWinSocket.Connected 83
ScktComp.TCustomWinSocket.Create 66
ScktComp.TCustomWinSocket.Data 84
ScktComp.TCustomWinSocket.DefaultHandler 70
ScktComp.TCustomWinSocket.Destroy 67
ScktComp.TCustomWinSocket.Disconnect 70
ScktComp.TCustomWinSocket.DoListen 70
ScktComp.TCustomWinSocket.DoOpen 71
ScktComp.TCustomWinSocket.Error 71
ScktComp.TCustomWinSocket.Event 72
ScktComp.TCustomWinSocket.Handle 84
ScktComp.TCustomWinSocket.InitSocket 72
ScktComp.TCustomWinSocket.Listen 73
ScktComp.TCustomWinSocket.LocalAddress 85
ScktComp.TCustomWinSocket.LocalHost 85
ScktComp.TCustomWinSocket.LocalPort 86
ScktComp.TCustomWinSocket.Lock 74
ScktComp.TCustomWinSocket.LookupName 74
ScktComp.TCustomWinSocket.LookupService 75
ScktComp.TCustomWinSocket.LookupState 86
ScktComp.TCustomWinSocket.OnErrorEvent 89
ScktComp.TCustomWinSocket.OnSocketEvent 90
ScktComp.TCustomWinSocket.Open 75
ScktComp.TCustomWinSocket.Read 76
ScktComp.TCustomWinSocket.ReceiveBuf 77
ScktComp.TCustomWinSocket.ReceiveLength 77
ScktComp.TCustomWinSocket.ReceiveText 78
ScktComp.TCustomWinSocket.RemoteAddr 87
ScktComp.TCustomWinSocket.RemoteAddress 87
ScktComp.TCustomWinSocket.RemoteHost 88
ScktComp.TCustomWinSocket.RemotePort 88
ScktComp.TCustomWinSocket.SendBuf 78
ScktComp.TCustomWinSocket.SendStream 79
ScktComp.TCustomWinSocket.SendStreamThenDrop 79
ScktComp.TCustomWinSocket.SendText 80
ScktComp.TCustomWinSocket.SetAsyncStyles 81
ScktComp.TCustomWinSocket.SocketHandle 89
ScktComp.TCustomWinSocket.Unlock 81
ScktComp.TCustomWinSocket.Write 81
ScktComp.TErrorEvent 147
ScktComp.TErrorEvent enumeration 147
ScktComp.TGetSocketEvent 150
ScktComp.TGetSocketEvent type 150
ScktComp.TGetThreadEvent 151
ScktComp.TGetThreadEvent type 151
ScktComp.TLookupState 148
ScktComp.TLookupState enumeration 148
ScktComp.TServerAcceptThread 90
ScktComp.TServerAcceptThread.Create 91
ScktComp.TServerAcceptThread.Execute 91
ScktComp.TServerAcceptThread.ServerSocket 95
ScktComp.TServerClientThread 95
ScktComp.TServerClientThread.ClientExecute 98

ScktComp.TServerClientThread.ClientSocket 102
ScktComp.TServerClientThread.Create 97
ScktComp.TServerClientThread.Data 103
ScktComp.TServerClientThread.Destroy 98
ScktComp.TServerClientThread.DoTerminate 99
ScktComp.TServerClientThread.EndConnect 99
ScktComp.TServerClientThread.Error 100
ScktComp.TServerClientThread.Event 100
ScktComp.TServerClientThread.Execute 101
ScktComp.TServerClientThread.HandleException 101
ScktComp.TServerClientThread.KeepInCache 103
ScktComp.TServerClientThread.ReActivate 101
ScktComp.TServerClientThread.ServerSocket 104
ScktComp.TServerClientThread.StartConnect 102
ScktComp.TServerClientWinSocket 104
ScktComp.TServerClientWinSocket.Create 106
ScktComp.TServerClientWinSocket.Destroy 107
ScktComp.TServerClientWinSocket.ServerWinSocket 107
ScktComp.TServerSocket 108
ScktComp.TServerSocket.Active 111
ScktComp.TServerSocket.Create 110
ScktComp.TServerSocket.OnAccept 114
ScktComp.TServerSocket.OnClientConnect 115
ScktComp.TServerSocket.OnClientDisconnect 116
ScktComp.TServerSocket.OnClientError 116
ScktComp.TServerSocket.OnClientRead 117
ScktComp.TServerSocket.OnClientWrite 117
ScktComp.TServerSocket.OnGetSocket 118
ScktComp.TServerSocket.OnGetThread 118
ScktComp.TServerSocket.OnListen 119
ScktComp.TServerSocket.OnThreadEnd 119
ScktComp.TServerSocket.OnThreadStart 120
ScktComp.TServerSocket.Port 111
ScktComp.TServerSocket.ServerType 112
ScktComp.TServerSocket.Service 112
ScktComp.TServerSocket.Socket 113
ScktComp.TServerSocket.ThreadCacheSize 113
ScktComp.TServerType 148
ScktComp.TServerType enumeration 148
ScktComp.TServerWinSocket 120
ScktComp.TServerWinSocket.Accept 124
ScktComp.TServerWinSocket.ActiveConnections 129
ScktComp.TServerWinSocket.ActiveThreads 130
ScktComp.TServerWinSocket.ClientConnect 125
ScktComp.TServerWinSocket.ClientDisconnect 125
ScktComp.TServerWinSocket.ClientErrorEvent 125
ScktComp.TServerWinSocket.ClientRead 125
ScktComp.TServerWinSocket.ClientWrite 126
ScktComp.TServerWinSocket.Connections 130
ScktComp.TServerWinSocket.Create 123
ScktComp.TServerWinSocket.Destroy 123
ScktComp.TServerWinSocket.Disconnect 126
ScktComp.TServerWinSocket.DoCreateThread 126
ScktComp.TServerWinSocket.GetClientSocket 127
ScktComp.TServerWinSocket.GetClientThread 127
ScktComp.TServerWinSocket.GetServerThread 127
ScktComp.TServerWinSocket.IdleThreads 131
ScktComp.TServerWinSocket.Listen 127
ScktComp.TServerWinSocket.OnClientConnect 133
ScktComp.TServerWinSocket.OnClientDisconnect 134
ScktComp.TServerWinSocket.OnClientError 135
ScktComp.TServerWinSocket.OnClientRead 135
ScktComp.TServerWinSocket.OnClientWrite 136
ScktComp.TServerWinSocket.OnGetSocket 137
ScktComp.TServerWinSocket.OnGetThread 137
ScktComp.TServerWinSocket.OnThreadEnd 138
ScktComp.TServerWinSocket.OnThreadStart 138
ScktComp.TServerWinSocket.ServerType 131
ScktComp.TServerWinSocket.SetServerType 128
ScktComp.TServerWinSocket.SetThreadCacheSize 128
ScktComp.TServerWinSocket.ThreadCacheSize 132
ScktComp.TServerWinSocket.ThreadEnd 129
ScktComp.TServerWinSocket.ThreadStart 129
ScktComp.TSocketErrorEvent 151
ScktComp.TSocketErrorEvent type 151
ScktComp.TSocketErrorProc 152
ScktComp.TSocketErrorProc type 152
ScktComp.TSocketEvent 148

ScktComp.TSocketEvent enumeration 148
ScktComp.TSocketEventEvent 152
ScktComp.TSocketEventEvent type 152
ScktComp.TSocketNotifyEvent 152
ScktComp.TSocketNotifyEvent type 152
ScktComp.TThreadNotifyEvent 153
ScktComp.TThreadNotifyEvent type 153
ScktComp.TWinSocketStream 138
ScktComp.TWinSocketStream.Create 140
ScktComp.TWinSocketStream.Destroy 140
ScktComp.TWinSocketStream.Read 141
ScktComp.TWinSocketStream.Seek 141
ScktComp.TWinSocketStream.TimeOut 143
ScktComp.TWinSocketStream.WaitForData 142
ScktComp.TWinSocketStream.Write 142
ScktMain 157
ScktMain namespace 157
Classes 158
Constants 188
Types 186
Variables 187
ScktMain.SocketForm 187
ScktMain.SocketForm variable 187
ScktMain.SocketService 187
ScktMain.SocketService variable 187
ScktMain.TSocketForm 158
ScktMain.TSocketForm.ActionList1 161
ScktMain.TSocketForm.AddClient 174
ScktMain.TSocketForm.AddIcon 174
ScktMain.TSocketForm.AllowXML 161
ScktMain.TSocketForm.AllowXMLExecute 174
ScktMain.TSocketForm.ApplyAction 162
ScktMain.TSocketForm.ApplyActionExecute 174
ScktMain.TSocketForm.ApplyActionUpdate 174
ScktMain.TSocketForm.ApplyButton 162
ScktMain.TSocketForm.ClearModifications 175
ScktMain.TSocketForm.ConnectionList 162
ScktMain.TSocketForm.ConnectionListColumnClick 175
ScktMain.TSocketForm.ConnectionListCompare 175
ScktMain.TSocketForm.Connections1 162
ScktMain.TSocketForm.DisconnectAction 163
ScktMain.TSocketForm.DisconnectActionUpdate 175
ScktMain.TSocketForm.ExportedObjectOnly1 163
ScktMain.TSocketForm.FormCloseQuery 176
ScktMain.TSocketForm.FormCreate 176
ScktMain.TSocketForm.FormDestroy 176
ScktMain.TSocketForm.FormShow 176
ScktMain.TSocketForm.GUIDDesc 163
ScktMain.TSocketForm.HeaderControl1 163
ScktMain.TSocketForm.Initialize 176
ScktMain.TSocketForm.IntegerExit 177
ScktMain.TSocketForm.InterceptGroup 163
ScktMain.TSocketForm.InterceptGUID 164
ScktMain.TSocketForm.ItemIndex 181
ScktMain.TSocketForm.Label1 164
ScktMain.TSocketForm.Label4 164
ScktMain.TSocketForm.Label5 164
ScktMain.TSocketForm.Label7 165
ScktMain.TSocketForm.MainMenu1 165
ScktMain.TSocketForm.miAdd 165
ScktMain.TSocketForm.miAddClick 177
ScktMain.TSocketForm.miClose 165
ScktMain.TSocketForm.miCloseClick 177
ScktMain.TSocketForm.miDisconnect 165
ScktMain.TSocketForm.miDisconnectClick 177
ScktMain.TSocketForm.miExit 166
ScktMain.TSocketForm.miExitClick 178
ScktMain.TSocketForm.miPorts 166
ScktMain.TSocketForm.miProperties 166
ScktMain.TSocketForm.miPropertiesClick 178
ScktMain.TSocketForm.miRemove 166
ScktMain.TSocketForm.miShowHostName 167
ScktMain.TSocketForm.N1 167
ScktMain.TSocketForm.N2 167
ScktMain.TSocketForm.N3 167
ScktMain.TSocketForm.Pages 167
ScktMain.TSocketForm.Panel1 168
ScktMain.TSocketForm.PopupMenu 168

SocktMain.TSocketForm.PortDesc 168
SocktMain.TSocketForm.PortGroup 168
SocktMain.TSocketForm.PortList 169
SocktMain.TSocketForm.PortListClick 178
SocktMain.TSocketForm.PortNo 169
SocktMain.TSocketForm.PortUpDown 169
SocktMain.TSocketForm.PropPage 169
SocktMain.TSocketForm.ReadSettings 178
SocktMain.TSocketForm.RegisteredAction 169
SocktMain.TSocketForm.RegisteredActionExecute 178
SocktMain.TSocketForm.RemoveClient 179
SocktMain.TSocketForm.RemovePortAction 170
SocktMain.TSocketForm.RemovePortActionExecute 179
SocktMain.TSocketForm.RemovePortActionUpdate 179
SocktMain.TSocketForm.SelectedSocket 181
SocktMain.TSocketForm.ShowHostAction 170
SocktMain.TSocketForm.ShowHostActionExecute 179
SocktMain.TSocketForm.StatPage 170
SocktMain.TSocketForm.ThreadDesc 170
SocktMain.TSocketForm.ThreadGroup 171
SocktMain.TSocketForm.ThreadSize 171
SocktMain.TSocketForm.ThreadUpDown 171
SocktMain.TSocketForm.Timeout 171
SocktMain.TSocketForm.TimeoutDesc 171
SocktMain.TSocketForm.TimeoutGroup 172
SocktMain.TSocketForm.TimeoutUpDown 172
SocktMain.TSocketForm.UIInitialize 180
SocktMain.TSocketForm.UpdateTimer 172
SocktMain.TSocketForm.UpdateTimerTimer 180
SocktMain.TSocketForm.UpDownClick 180
SocktMain.TSocketForm.UserStatus 172
SocktMain.TSocketForm.WMMIDASIcon 180
SocktMain.TSocketForm.WndProc 180
SocktMain.TSocketForm.WriteSettings 181
SocktMain.TSocketForm.XMLPacket1 173
SocktMain.TSocketProc 186
SocktMain.TSocketProc type 186
SocktMain.TSocketService 182
SocktMain.TSocketService.CreateNew 183
SocktMain.TSocketService.GetServiceController 184
SocktMain.TSocketService.Start 184
SocktMain.TSocketService.Stop 185
SocktMain.UI_INITIALIZE 188
SocktMain.UI_INITIALIZE constant 188
SocktMain.WM_MIDASICON 188
SocktMain.WM_MIDASICON constant 188
scLineDown enumeration member 3004
scLineUp enumeration member 3004
SConnect 189
SConnect namespace 189
  Classes 191
  Constants 288
  Functions 282
  Interfaces 266
  Structs, Records, Enums 283
  Types 284
  Variables 286
SConnect.asCreateObject 288
SConnect.asCreateObject constant 288
SConnect.asError 288
SConnect.asError constant 288
SConnect.asFreeObject 289
SConnect.asFreeObject constant 289
SConnect.asGetAppServers 289
SConnect.asGetAppServers constant 289
SConnect.asGetGUID 289
SConnect.asGetGUID constant 289
SConnect.asGetID 290
SConnect.asGetID constant 290
SConnect.asGetServers 290
SConnect.asGetServers constant 290
SConnect.asInvoke 290
SConnect.asInvoke constant 290
SConnect.asMask 291
SConnect.asMask constant 291
SConnect.asSoapCommand 291
SConnect.asSoapCommand constant 291
SConnect.CallSig 291

SConnect.CallSig constant 291
SConnect.EInterpreterError 191
SConnect.ESocketConnectionError 192
SConnect.GetPacketInterceptorList 282
SConnect.GetPacketInterceptorList function 282
SConnect.IDataBlock 266
SConnect.IDataBlock.BytesReserved 272
SConnect.IDataBlock.Clear 267
SConnect.IDataBlock.GetBytesReserved 268
SConnect.IDataBlock.GetMemory 268
SConnect.IDataBlock.GetSignature 268
SConnect.IDataBlock.GetSize 269
SConnect.IDataBlock.GetStream 269
SConnect.IDataBlock.IgnoreStream 269
SConnect.IDataBlock.InitData 270
SConnect.IDataBlock.Memory 272
SConnect.IDataBlock.Read 270
SConnect.IDataBlock.SetSignature 271
SConnect.IDataBlock.SetSize 271
SConnect.IDataBlock.Signature 273
SConnect.IDataBlock.Size 273
SConnect.IDataBlock.Stream 274
SConnect.IDataBlock.Write 271
SConnect.IDataIntercept 274
SConnect.IDataIntercept.DataIn 275
SConnect.IDataIntercept.DataOut 276
SConnect.ISendDataBlock 276
SConnect.ISendDataBlock.Send 277
SConnect.ITransport 278
SConnect.ITransport.Connected 281
SConnect.ITransport.GetConnected 279
SConnect.ITransport.GetWaitEvent 279
SConnect.ITransport.Receive 280
SConnect.ITransport.Send 280
SConnect.ITransport.SetConnected 280
SConnect.LoadWinSock2 282
SConnect.LoadWinSock2 function 282
SConnect.PIntArray 284
SConnect.PIntArray type 284
SConnect.PVariantArray 284
SConnect.PVariantArray type 284
SConnect.ResultSig 292
SConnect.ResultSig constant 292
SConnect.TCustomDataBlockInterpreter 192
SConnect.TCustomDataBlockInterpreter.AddDispatch 194
SConnect.TCustomDataBlockInterpreter.CallCreateObject 194
SConnect.TCustomDataBlockInterpreter.CallFreeObject 194
SConnect.TCustomDataBlockInterpreter.CallGetIDsOfNames 194
SConnect.TCustomDataBlockInterpreter.CallGetServerList 195
SConnect.TCustomDataBlockInterpreter.CallInvoke 195
SConnect.TCustomDataBlockInterpreter.CanCreateObject 195
SConnect.TCustomDataBlockInterpreter.CreateObject 195
SConnect.TCustomDataBlockInterpreter.InternalCreateObject 195
SConnect.TCustomDataBlockInterpreter.InterpretData 196
SConnect.TCustomDataBlockInterpreter.LockObject 196
SConnect.TCustomDataBlockInterpreter.ReleaseObject 196
SConnect.TCustomDataBlockInterpreter.RemoveDispatch 196
SConnect.TCustomDataBlockInterpreter.StoreObject 197
SConnect.TCustomDataBlockInterpreter.UnlockObject 197
SConnect.TDataBlock 197
SConnect.TDataBlock.BytesReserved 205
SConnect.TDataBlock.Clear 200
SConnect.TDataBlock.Create 199
SConnect.TDataBlock.Destroy 199
SConnect.TDataBlock.GetBytesReserved 201
SConnect.TDataBlock.GetMemory 201
SConnect.TDataBlock.GetSignature 201
SConnect.TDataBlock.GetSize 202
SConnect.TDataBlock.GetStream 202
SConnect.TDataBlock.IgnoreStream 202
SConnect.TDataBlock.InitData 202
SConnect.TDataBlock.Memory 205
SConnect.TDataBlock.Read 203
SConnect.TDataBlock.SetSignature 203
SConnect.TDataBlock.SetSize 204
SConnect.TDataBlock.Signature 206
SConnect.TDataBlock.Size 206

SConnect.TDataBlock.Stream 206
SConnect.TDataBlock.Write 204
SConnect.TDataBlockInterpreter 207
SConnect.TDataBlockInterpreter.AddDispatch 209
SConnect.TDataBlockInterpreter.CallCreateObject 210
SConnect.TDataBlockInterpreter.CallFreeObject 210
SConnect.TDataBlockInterpreter.CallGetIDsOfNames 210
SConnect.TDataBlockInterpreter.CallGetServerList 210
SConnect.TDataBlockInterpreter.CallInvoke 211
SConnect.TDataBlockInterpreter.CanCreateObject 211
SConnect.TDataBlockInterpreter.Create 208
SConnect.TDataBlockInterpreter.CreateObject 211
SConnect.TDataBlockInterpreter.Destroy 208
SConnect.TDataBlockInterpreter.DoCreateObject 211
SConnect.TDataBlockInterpreter.DoCustomAction 211
SConnect.TDataBlockInterpreter.DoFreeObject 212
SConnect.TDataBlockInterpreter.DoGetAppServerList 212
SConnect.TDataBlockInterpreter.DoGetIDsOfNames 212
SConnect.TDataBlockInterpreter.DoGetServerList 212
SConnect.TDataBlockInterpreter.DoInvoke 213
SConnect.TDataBlockInterpreter.InternalCreateObject 213
SConnect.TDataBlockInterpreter.InterpretData 213
SConnect.TDataBlockInterpreter.LockObject 213
SConnect.TDataBlockInterpreter.ReleaseObject 214
SConnect.TDataBlockInterpreter.RemoveDispatch 214
SConnect.TDataBlockInterpreter.StoreObject 214
SConnect.TDataBlockInterpreter.UnlockObject 214
SConnect.TDataDispatch 215
SConnect.TDataDispatch.Create 216
SConnect.TDataDispatch.Destroy 217
SConnect.TDataDispatch.DispatchIndex 219
SConnect.TDataDispatch.GetIDsOfNames 217
SConnect.TDataDispatch.GetTypeInfo 218
SConnect.TDataDispatch.GetTypeInfoCount 218
SConnect.TDataDispatch.Invoke 219
SConnect.THREAD_EXCEPTION 292
SConnect.THREAD_EXCEPTION constant 292
SConnect.THREAD_RECEIVEDSTREAM 292
SConnect.THREAD_RECEIVEDSTREAM constant 292
SConnect.THREAD_REPLACETRANSPORT 293
SConnect.THREAD_REPLACETRANSPORT constant 293
SConnect.THREAD_SENDNOTIFY 293
SConnect.THREAD_SENDNOTIFY constant 293
SConnect.THREAD_SENDSTREAM 293
SConnect.THREAD_SENDSTREAM constant 293
SConnect.TIntArray 285
SConnect.TIntArray type 285
SConnect.TPacketInterceptFactory 219
SConnect.TPacketInterceptFactory.UpdateRegistry 221
SConnect.TSocketConnection 221
SConnect.TSocketConnection.Address 225
SConnect.TSocketConnection.Create 223
SConnect.TSocketConnection.CreateTransport 224
SConnect.TSocketConnection.DoConnect 224
SConnect.TSocketConnection.Host 226
SConnect.TSocketConnection.InterceptGUID 226
SConnect.TSocketConnection.InterceptName 227
SConnect.TSocketConnection.ObjectBroker 228
SConnect.TSocketConnection.Port 228
SConnect.TSocketConnection.SupportCallbacks 229
SConnect.TSocketTransport 229
SConnect.TSocketTransport.Address 234
SConnect.TSocketTransport.Create 231
SConnect.TSocketTransport.Destroy 231
SConnect.TSocketTransport.GetConnected 232
SConnect.TSocketTransport.GetWaitEvent 233
SConnect.TSocketTransport.Host 235
SConnect.TSocketTransport.InterceptGUID 235
SConnect.TSocketTransport.Port 235
SConnect.TSocketTransport.Receive 233
SConnect.TSocketTransport.Send 234
SConnect.TSocketTransport.SetConnected 234
SConnect.TSocketTransport.Socket 235
SConnect.TStreamedConnection 236
SConnect.TStreamedConnection._AddRef 239
SConnect.TStreamedConnection._Release 239
SConnect.TStreamedConnection.Create 237
SConnect.TStreamedConnection.CreateTransport 239

SConnect.TStreamedConnection.Destroy 238
SConnect.TStreamedConnection.DoConnect 240
SConnect.TStreamedConnection.DoDisconnect 240
SConnect.TStreamedConnection.DoError 241
SConnect.TStreamedConnection.GetInterceptorList 241
SConnect.TStreamedConnection.GetInterpreter 242
SConnect.TStreamedConnection.GetServerList 242
SConnect.TStreamedConnection.Handle 246
SConnect.TStreamedConnection.InterceptGUID 246
SConnect.TStreamedConnection.InterceptName 247
SConnect.TStreamedConnection.InternalClose 242
SConnect.TStreamedConnection.InternalOpen 243
SConnect.TStreamedConnection.Interpreter 247
SConnect.TStreamedConnection.QueryInterface 243
SConnect.TStreamedConnection.Send 244
SConnect.TStreamedConnection.SupportCallbacks 248
SConnect.TStreamedConnection.ThreadException 244
SConnect.TStreamedConnection.ThreadReceivedStream 244
SConnect.TStreamedConnection.WndProc 245
SConnect.TTransportThread 248
SConnect.TTransportThread.Create 249
SConnect.TTransportThread.Destroy 249
SConnect.TTransportThread.Execute 250
SConnect.TTransportThread.Semaphore 254
SConnect.TVarFlag 283
SConnect.TVarFlag enumeration 283
SConnect.TVarFlags 285
SConnect.TVarFlags type 285
SConnect.TVariantArray 285
SConnect.TVariantArray type 285
SConnect.TWebConnection 254
SConnect.TWebConnection.Agent 262
SConnect.TWebConnection.Create 256
SConnect.TWebConnection.CreateTransport 257
SConnect.TWebConnection.Destroy 257
SConnect.TWebConnection.DoConnect 258
SConnect.TWebConnection.GetInterpreter 258
SConnect.TWebConnection.GetWaitEvent 259
SConnect.TWebConnection.ObjectBroker 262
SConnect.TWebConnection.Password 263
SConnect.TWebConnection.Proxy 263
SConnect.TWebConnection.ProxyByPass 263
SConnect.TWebConnection.Receive 259
SConnect.TWebConnection.Send 260
SConnect.TWebConnection.SetConnected 260
SConnect.TWebConnection.SupportCallbacks 264
SConnect.TWebConnection.Transport_GetConnected 261
SConnect.TWebConnection.Transport_SetConnected 261
SConnect.TWebConnection.URL 264
SConnect.TWebConnection.UserName 265
SConnect.WSACloseEvent 286
SConnect.WSACloseEvent variable 286
SConnect.WSACreateEvent 286
SConnect.WSACreateEvent variable 286
SConnect.WSAEventSelect 287
SConnect.WSAEventSelect variable 287
SConnect.WSAResetEvent 287
SConnect.WSAResetEvent variable 287
scPageDown enumeration member 3004
scPageUp enumeration member 3004
scPosition enumeration member 3004
ScreenTip
  TCustomScreenTipsPopup class 322
  TScreenTipsPopup class 349
ScreenTipManager
  TCustomScreenTipsPopup class 322
  TScreenTipsPopup class 349
ScreenTips 295
  TCustomScreenTipsManager class 315
  TScreenTipsManager class 341
ScreenTips namespace 295
  Classes 297
  Constants 369
  Interfaces 361
  Structs, Records, Enums 363
  Types 366
ScreenTips.cDefaultBorderColor 369
ScreenTips.cDefaultBorderColor constant 369

ScreenTips.cDefaultEndColor 369
ScreenTips.cDefaultEndColor constant 369
ScreenTips.cDefaultStartColor 370
ScreenTips.cDefaultStartColor constant 370
ScreenTips.cScreenTipHeaderSpacing 370
ScreenTips.cScreenTipHeaderSpacing constant 370
ScreenTips.cScreenTipMinimumHeight 370
ScreenTips.cScreenTipMinimumHeight constant 370
ScreenTips.cScreenTipTextAndImageWidth 371
ScreenTips.cScreenTipTextAndImageWidth constant 371
ScreenTips.cScreenTipTextOnlyWidth 371
ScreenTips.cScreenTipTextOnlyWidth constant 371
ScreenTips.EAllocatedActionsException 297
ScreenTips.IScreenTipItem 361
ScreenTips.IScreenTipItem.GetScreenTipItem 362
ScreenTips.TAllocatedActions 297
ScreenTips.TAllocatedActions.AddAction 299
ScreenTips.TAllocatedActions.Clear 299
ScreenTips.TAllocatedActions.Count 300
ScreenTips.TAllocatedActions.Create 298
ScreenTips.TAllocatedActions.Delete 299
ScreenTips.TAllocatedActions.Destroy 298
ScreenTips.TAllocatedActions.Exists 299
ScreenTips.TAllocatedActions.Find 300
ScreenTips.TAllocatedActions.GetItem 300
ScreenTips.TAssociateItem 301
ScreenTips.TAssociateItem.Associate 302
ScreenTips.TAssociateItem.Create 301
ScreenTips.TAssociateItem.Destroy 301
ScreenTips.TAssociateItem.PopupTip 302
ScreenTips.TAssociateItemList 302
ScreenTips.TAssociateItemList.Add 304
ScreenTips.TAssociateItemList.Create 303
ScreenTips.TAssociateItemList.Destroy 303
ScreenTips.TAssociateItemList.Find 304
ScreenTips.TAssociateItemList.Get 304
ScreenTips.TAssociateItemList.Remove 304
ScreenTips.TCreateScreenTipEvent 366
ScreenTips.TCreateScreenTipEvent type 366
ScreenTips.TCustomScreenTipsManager 305
ScreenTips.TCustomScreenTipsManager.AddPopup 308
ScreenTips.TCustomScreenTipsManager.BorderStyle 313
ScreenTips.TCustomScreenTipsManager.ClearScreenTips 308
ScreenTips.TCustomScreenTipsManager.Create 306
ScreenTips.TCustomScreenTipsManager.CreateScreenTip 308
ScreenTips.TCustomScreenTipsManager.Destroy 307
ScreenTips.TCustomScreenTipsManager.DisplayOffset 313
ScreenTips.TCustomScreenTipsManager.DoCreateScreenTip 309
ScreenTips.TCustomScreenTipsManager.DoDuplicateCreateError 309
ScreenTips.TCustomScreenTipsManager.DoGetScreenTipClass 309
ScreenTips.TCustomScreenTipsManager.DoScreenTipsLoaded 309
ScreenTips.TCustomScreenTipsManager.DoShow 309
ScreenTips.TCustomScreenTipsManager.DoShowScreenTip 310
ScreenTips.TCustomScreenTipsManager.Font 313
ScreenTips.TCustomScreenTipsManager.Footer 313
ScreenTips.TCustomScreenTipsManager.FooterImage 314
ScreenTips.TCustomScreenTipsManager.GenerateScreenTips 310
ScreenTips.TCustomScreenTipsManager.GetScreenTip 310
ScreenTips.TCustomScreenTipsManager.GradientEndColor 314
ScreenTips.TCustomScreenTipsManager.GradientStartColor 314
ScreenTips.TCustomScreenTipsManager.LinkedActionLists 314
ScreenTips.TCustomScreenTipsManager.Loaded 310
ScreenTips.TCustomScreenTipsManager.Notification 311
ScreenTips.TCustomScreenTipsManager.OnCreateScreenTip 315
ScreenTips.TCustomScreenTipsManager.OnDuplicateCreateError 315
ScreenTips.TCustomScreenTipsManager.OnGetScreenTipWinClass 316
ScreenTips.TCustomScreenTipsManager.OnScreenTipsLoaded 316
ScreenTips.TCustomScreenTipsManager.OnShowScreenTip

316
ScreenTips.TCustomScreenTipsManager.Options 314
ScreenTips.TCustomScreenTipsManager.RemovePopup 311
ScreenTips.TCustomScreenTipsManager.RemoveScreenTip 312
ScreenTips.TCustomScreenTipsManager.ScreenTips 315
ScreenTips.TCustomScreenTipsManager.UpdateActionInfo 312
ScreenTips.TCustomScreenTipsManager.UpdateOptions 312
ScreenTips.TCustomScreenTipsPopup 316
ScreenTips.TCustomScreenTipsPopup.Associate 321
ScreenTips.TCustomScreenTipsPopup.Create 317
ScreenTips.TCustomScreenTipsPopup.Destroy 318
ScreenTips.TCustomScreenTipsPopup.GetScreenTipItem 319
ScreenTips.TCustomScreenTipsPopup.Glyph 321
ScreenTips.TCustomScreenTipsPopup.LinkPopupToManager 319
ScreenTips.TCustomScreenTipsPopup.Notification 319
ScreenTips.TCustomScreenTipsPopup.Paint 320
ScreenTips.TCustomScreenTipsPopup.PopupType 321
ScreenTips.TCustomScreenTipsPopup.RemoveLinkToManager 320
ScreenTips.TCustomScreenTipsPopup.ScreenTip 322
ScreenTips.TCustomScreenTipsPopup.ScreenTipManager 322
ScreenTips.TCustomScreenTipsPopup.ShowHint 322
ScreenTips.TCustomScreenTipsPopup.Transparent 324
ScreenTips.TCustomScreenTipsPopup.UpdateAssociateToManager 320
ScreenTips.TDescriptionType 363
ScreenTips.TDescriptionType enumeration 363
ScreenTips.TDisplayOffset 324
ScreenTips.TDisplayOffset.Control 325
ScreenTips.TDisplayOffset.Create 324
ScreenTips.TDisplayOffset.X 325
ScreenTips.TDisplayOffset.Y 325
ScreenTips.TDuplicateCreateErrorEvent 366
ScreenTips.TDuplicateCreateErrorEvent type 366
ScreenTips.TImageAlign 363
ScreenTips.TImageAlign enumeration 363
ScreenTips.TItemImageAlign 364
ScreenTips.TItemImageAlign enumeration 364
ScreenTips.TScreenTipBorderStyle 326
ScreenTips.TScreenTipBorderStyle.Color 327
ScreenTips.TScreenTipBorderStyle.CornerSize 327
ScreenTips.TScreenTipBorderStyle.Create 326
ScreenTips.TScreenTipBorderStyle.Style 327
ScreenTips.TScreenTipItem 328
ScreenTips.TScreenTipItem.Action 332
ScreenTips.TScreenTipItem.Assign 330
ScreenTips.TScreenTipItem.Create 329
ScreenTips.TScreenTipItem.Description 332
ScreenTips.TScreenTipItem.Destroy 329
ScreenTips.TScreenTipItem.DisabledDescription 332
ScreenTips.TScreenTipItem.DisabledHeader 332
ScreenTips.TScreenTipItem.GetDisplayName 330
ScreenTips.TScreenTipItem.Header 332
ScreenTips.TScreenTipItem.Height 333
ScreenTips.TScreenTipItem.Image 333
ScreenTips.TScreenTipItem.IsEnabled 331
ScreenTips.TScreenTipItem.Manager 333
ScreenTips.TScreenTipItem.SetManager 331
ScreenTips.TScreenTipItem.ShowFooter 333
ScreenTips.TScreenTipItem.ShowImage 334
ScreenTips.TScreenTipItem.Width 334
ScreenTips.TScreenTipOption 364
ScreenTips.TScreenTipOption enumeration 364
ScreenTips.TScreenTipOptions 367
ScreenTips.TScreenTipOptions type 367
ScreenTips.TScreenTipPopupType 365
ScreenTips.TScreenTipPopupType enumeration 365
ScreenTips.TScreenTipsCollection 334
ScreenTips.TScreenTipsCollection.Add 335
ScreenTips.TScreenTipsCollection.Create 335
ScreenTips.TScreenTipsCollection.GetOwner 336
ScreenTips.TScreenTipsCollection.Insert 336
ScreenTips.TScreenTipsCollection.Items 337
ScreenTips.TScreenTipsManager 337
ScreenTips.TScreenTipsManager.BorderStyle 339
ScreenTips.TScreenTipsManager.DisplayOffset 339
ScreenTips.TScreenTipsManager.Font 339

ScreenTips.TScreenTipsManager.Footer 340
ScreenTips.TScreenTipsManager.FooterImage 340
ScreenTips.TScreenTipsManager.GradientEndColor 340
ScreenTips.TScreenTipsManager.GradientStartColor 340
ScreenTips.TScreenTipsManager.LinkedActionLists 340
ScreenTips.TScreenTipsManager.OnCreateScreenTip 341
ScreenTips.TScreenTipsManager.OnDuplicateCreateError 342
ScreenTips.TScreenTipsManager.OnGetScreenTipWinClass 342
ScreenTips.TScreenTipsManager.OnScreenTipsLoaded 342
ScreenTips.TScreenTipsManager.OnShowScreenTip 342
ScreenTips.TScreenTipsManager.Options 341
ScreenTips.TScreenTipsManager.ScreenTips 341
ScreenTips.TScreenTipsPopup 343
ScreenTips.TScreenTipsPopup.Align 344
ScreenTips.TScreenTipsPopup.Anchors 347
ScreenTips.TScreenTipsPopup.Associate 348
ScreenTips.TScreenTipsPopup.Glyph 348
ScreenTips.TScreenTipsPopup.PopupType 348
ScreenTips.TScreenTipsPopup.ScreenTip 349
ScreenTips.TScreenTipsPopup.ScreenTipManager 349
ScreenTips.TScreenTipsPopup.Transparent 349
ScreenTips.TScreenTipsPopup.Visible 349
ScreenTips.TScreenTipsWindow 351
ScreenTips.TScreenTipsWindow.ActivateToolTip 353
ScreenTips.TScreenTipsWindow.ActiveItem 357
ScreenTips.TScreenTipsWindow.Body 358
ScreenTips.TScreenTipsWindow.CMMouseenter 353
ScreenTips.TScreenTipsWindow.CMMouseLeave 354
ScreenTips.TScreenTipsWindow.CornerSize 358
ScreenTips.TScreenTipsWindow.Create 352
ScreenTips.TScreenTipsWindow.CreateParams 354
ScreenTips.TScreenTipsWindow.CreateRoundedStyle 354
ScreenTips.TScreenTipsWindow.CreateTipBorderStyle 355
ScreenTips.TScreenTipsWindow.CreateWnd 355
ScreenTips.TScreenTipsWindow.Destroy 352
ScreenTips.TScreenTipsWindow.DrawBackground 355
ScreenTips.TScreenTipsWindow.DrawBorder 356
ScreenTips.TScreenTipsWindow.DrawFooter 356
ScreenTips.TScreenTipsWindow.DrawHeader 356
ScreenTips.TScreenTipsWindow.Footer 358
ScreenTips.TScreenTipsWindow.FooterImage 358
ScreenTips.TScreenTipsWindow.GradientEndColor 359
ScreenTips.TScreenTipsWindow.GradientStartColor 359
ScreenTips.TScreenTipsWindow.HideHint 356
ScreenTips.TScreenTipsWindow.HintControl 359
ScreenTips.TScreenTipsWindow.Image 359
ScreenTips.TScreenTipsWindow.Paint 356
ScreenTips.TScreenTipsWindow.TipBorderStyle 359
ScreenTips.TScreenTipsWindow.WMNCHitTest 357
ScreenTips.TScreenTipsWindowClass 367
ScreenTips.TScreenTipsWindowClass type 367
ScreenTips.TScreenTipWinClass 367
ScreenTips.TScreenTipWinClass type 367
ScreenTips.TShowScreenTipEvent 368
ScreenTips.TShowScreenTipEvent type 368
ScreenTips.TTipBorderStyle 365
ScreenTips.TTipBorderStyle enumeration 365
Script
- TGeneratedFunctions class 6508
- TScriptFile class 6686
- TScriptItem class 6700
- TScriptProducer class 6715

ScriptEngine
- TScriptGlobalObjects class 6692

ScriptObjectFactories
- TAbstractScriptEngine class 6652

ScriptProducerClass
- TAbstractScriptEngine class 6652

ScriptTimeoutSecs
- IInternetEnvOptions interface 1045

Scroll
- TScrollBar class 2885

ScrollBars
- TCustomMemo class 2452
- TMemo class 2768
- TValueListEditor class 4832

Scroller
- TTabSet class 4502

ScrollOrientation
TScroller class 4462
ScrollWidth
TCustomListBox class 2432
TListBox class 2691
scTop enumeration member 3004
scTrack enumeration member 3004
seAccept enumeration member 148
Search
TSearchAction class 1898
SearchAssistantDefault
ISearchAssistantOC interface 655
ISearchAssistantOC2Disp interface 665
ISearchAssistantOC3Disp interface 676
ISearchAssistantOCDisp interface 684
SearchCompanionAvailable
ISearchAssistantOC3 interface 669
ISearchAssistantOC3Disp interface 676
Searches
ISearchAssistantOC interface 656
ISearchAssistantOC2Disp interface 665
ISearchAssistantOC3Disp interface 676
ISearchAssistantOCDisp interface 684
SearchFile
IIDEFileManager interface 1041
SearchFind
TSearchFindNext class 1918
Second
TXSCustomDateTime class 7820
TXSTime class 7859
SecondaryShortCuts
TBrowseForFolder class 1771
TColorSelect class 1780
TFileExit class 1815
TFileOpen class 1824
TFilePageSetup class 1841
TFilePrintSetup class 1851
TFileSaveAs class 1860
TFontEdit class 1870
TPrintDlg class 1890
TSearchFind class 1905
TSearchFindNext class 1919
TSearchReplace class 1928
seConnect enumeration member 148
seConnecting enumeration member 148
seDisconnect enumeration member 148
Seek
TWinSocketStream class 141
TZCompressionStream class 7910
TZDecompressionStream class 7914
Select
TBaseSocket class 1379
TCustomCombo class 2301
SelectAll
TCustomCombo class 2301
TCustomEdit class 2359
TCustomListBox class 2416
SelectCell
TValueListEditor class 4799
Selected
TCustomListBox class 2432
SelectedColor
TTabSet class 4502
SelectedImageIndex
TButton class 2067
TCustomButton class 2274
SelectedItem
IShellNameSpace interface 708
IShellNameSpaceDisp interface 716
TShellNameSpace class 482
SelectedItems
IShellNameSpace interface 704
IShellNameSpaceDisp interface 713
TShellNameSpace class 477
SelectedSocket
TSocketForm class 181
SelectionChange
DShellNameSpaceEvents interface 619

SelectItem
TCustomComboBox class 2330
SelectNext
TTabSet class 4480
selectNode
IDOMNodeSelect interface 7583
selectNodes
IDOMNodeSelect interface 7584
SelectRows
TAdapterDisplayField class 6379
TAdapterEditColumn class 6394
TCustomAdapterDisplayField class 6475
seListen enumeration member 148
SelLength
TCustomCombo class 2311
TCustomEdit class 2371
seLookup enumeration member 148
SelStart
TCustomCombo class 2314
TCustomEdit class 2374
SelText
TCustomComboBox class 2335
TCustomEdit class 2377
Semaphore
TTransportThread class 254
Send
ISendDataBlock interface 277
ITransport interface 280
THTTPReqResp class 1286
TSocketTransport class 234
TStreamedConnection class 244
TWebConnection class 260
SendBuf
TBaseSocket class 1380
TCustomWinSocket class 78
SendConnectEvent
TXMLBroker class 7319
SendGet
THTTPReqResp class 1286
SendIn
TBaseSocket class 1380
SendStream
TBaseSocket class 1380
TCustomWinSocket class 79
SendStreamThenDrop
TCustomWinSocket class 79
SendText
TCustomWinSocket class 80
SendTimeout
THTTPReqResp class 1291
SendTo
TIpSocket class 1422
seRead enumeration member 148
ServerFunctionsContent
TSitePageElements class 1168
ServerMethodName
TSqlServerMethod class 1670
ServerScriptContent
TSitePageElements class 1169
ServerSocket
TServerAcceptThread class 95
TServerClientThread class 104
TServerSocketThread class 1438
ServerSocketThread
TClientSocketThread class 1399
TCustomTcpServer class 1412
ServerType
TCustomServerSocket class 49
TServerSocket class 112
TServerWinSocket class 131
ServerVariablesContent
TSitePageElements class 1169
ServerWinSocket
TServerClientWinSocket class 107
Service
IServices interface 7161
TAbstractSocket class 23
TClientSocket class 31

THTTPRIO class 1253
TServerSocket class 112
TWSDLView class 7275
ServiceAddress
TWebServExp class 6732
ServiceCount
TServiceApplication class 3094
ServiceInfo
TWSDLHTMLPublish class 7286
Services
IDefinition interface 7100
ServiceStartName
TService class 3082
ServiceThread
TService class 3082
ServiceType
TService class 3083
SessColn 373
SessColn namespace 373
Classes 374
Structs, Records, Enums 396
Types 397
Variables 398
SessColn.ESessionsException 374
SessColn.Sessions 398
SessColn.Sessions variable 398
SessColn.TAbstractSessionIDGen 374
SessColn.TAbstractSessionIDGen.NextID 375
SessColn.TAbstractSessionStatusChange 375
SessColn.TAbstractSessionStatusChange.SessionStarted 376
SessColn.TAbstractSessionStatusChange.SessionTerminated 376
SessColn.TAbstractSessionStatusChange.SessionTimeout 377
SessColn.TSessionIDGen 377
SessColn.TSessionIDGen.Create 377
SessColn.TSessionIDGen.NextID 378
SessColn.TSessionItem 378
SessColn.TSessionItem.Assign 380
SessColn.TSessionItem.Create 379
SessColn.TSessionItem.Destroy 379
SessColn.TSessionItem.ExpirationTime 381
SessColn.TSessionItem.Expired 381
SessColn.TSessionItem.ID 382
SessColn.TSessionItem.IsActive 380
SessColn.TSessionItem.Items 382
SessColn.TSessionItem.StartTime 382
SessColn.TSessionItem.StatusFlags 382
SessColn.TSessionItem.Terminated 383
SessColn.TSessionItem.TimeoutMinutes 383
SessColn.TSessionItem.TouchTime 384
SessColn.TSessions 384
SessColn.TSessions.ActiveCount 394
SessColn.TSessions.CheckStatusChange 386
SessColn.TSessions.ConnectActiveSession 387
SessColn.TSessions.ConnectSession 387
SessColn.TSessions.Create 385
SessColn.TSessions.CreateSessionIDGen 387
SessColn.TSessions.Destroy 385
SessColn.TSessions.DisconnectSession 387
SessColn.TSessions.FileName 395
SessColn.TSessions.GetExpirationTime 388
SessColn.TSessions.GetExpired 388
SessColn.TSessions.GetIDCount 388
SessColn.TSessions.GetIDs 388
SessColn.TSessions.GetIsActive 389
SessColn.TSessions.GetItemCount 389
SessColn.TSessions.GetItems 389
SessColn.TSessions.GetSession 389
SessColn.TSessions.GetSessionIDGen 390
SessColn.TSessions.GetStartTime 390
SessColn.TSessions.GetTerminated 390
SessColn.TSessions.GetTimeoutMinutes 390
SessColn.TSessions.GetTouchTime 391
SessColn.TSessions.ItemValue 391
SessColn.TSessions.LoadFromFile 391
SessColn.TSessions.LoadFromStream 391
SessColn.TSessions.MaxSessions 395
SessColn.TSessions.NotifyStatusChange 392
SessColn.TSessions.SaveToFile 392

SessColn.TSessions.SaveToStream 392
SessColn.TSessions.SessionExists 393
SessColn.TSessions.SetItemValue 393
SessColn.TSessions.SetTimeoutMinutes 393
SessColn.TSessions.StartSession 393
SessColn.TSessions.TerminateSession 394
SessColn.TSessions.TouchSession 394
SessColn.TSessionStatusFlag 396
SessColn.TSessionStatusFlag enumeration 396
SessColn.TSessionStatusFlags 397
SessColn.TSessionStatusFlags type 397
Session
TAbstractWebContext class 5975
TCustomWebDataModule class 6555
TSiteModuleHelper class 6572
SessionExists
TSessions class 393
Sessions
TCustomWebAppComponents class 6214
TWebAppComponents class 6248
SessionStarted
TAbstractSessionStatusChange class 376
SessionTerminated
TAbstractSessionStatusChange class 376
SessionTimeout
TAbstractSessionStatusChange class 377
Set_AddressBar
IWebBrowser2 interface 757
TInternetExplorer class 443
Set_ASProvider
ISearchAssistantOC interface 653
Set_ASSetting
ISearchAssistantOC interface 653
set_async
IDOMParseOptions interface 7593
Set_Columns
IShellNameSpace interface 704
TShellNameSpace class 477
set_data
IDOMCharacterData interface 7529
IDOMProcessingInstruction interface 7601
Set_Depth
IShellNameSpace interface 704
TShellNameSpace class 477
set_documentElement
IDOMDocument interface 7536
set_Encoding
IDOMXMLProlog interface 7605
Set_EnumOptions
IShellNameSpace interface 705
TShellNameSpace class 477
Set_EventHandled
ISearchAssistantOC interface 654
Set_Flags
IShellNameSpace interface 705
TShellNameSpace class 477
Set_FullScreen
IWebBrowserApp interface 781
TInternetExplorer class 444
Set_Height
IWebBrowser interface 743
TInternetExplorer class 444
Set_input
IDOMXSLProcessor interface 7609
Set_Left
IWebBrowser interface 743
TInternetExplorer class 444
set_length
DynamicArray template 3268
Set_MenuBar
IWebBrowserApp interface 781
TInternetExplorer class 444
Set_Mode
IShellNameSpace interface 705
TShellNameSpace class 478
set_nodeValue
IDOMNode interface 7567
Set_Offline

IWebBrowser2 interface 757
TInternetExplorer class 444
set_OnAsyncLoad
IDOMPersist interface 7598
Set_output
IDOMXSLProcessor interface 7609
set_preserveWhiteSpace
IDOMParseOptions interface 7593
Set_RegisterAsBrowser
IWebBrowser2 interface 758
TInternetExplorer class 445
Set_RegisterAsDropTarget
IWebBrowser2 interface 758
TInternetExplorer class 445
Set_Resizable
IWebBrowser2 interface 758
TInternetExplorer class 445
set_resolveExternals
IDOMParseOptions interface 7593
Set_Root
IShellNameSpace interface 705
TShellNameSpace class 478
Set_SelectedItem
IShellNameSpace interface 706
TShellNameSpace class 478
Set_Silent
IWebBrowser2 interface 759
TInternetExplorer class 445
set_Standalone
IDOMXMLProlog interface 7606
Set_StatusBar
IWebBrowserApp interface 782
TInternetExplorer class 446
Set_StatusText
IWebBrowserApp interface 782
TInternetExplorer class 446
set_text
IDOMNodeEx interface 7579
Set_TheaterMode
IWebBrowser2 interface 759
TInternetExplorer class 446
Set_ToolBar
IWebBrowserApp interface 782
TInternetExplorer class 446
Set_Top
IWebBrowser interface 743
TInternetExplorer class 446
Set_TVFlags
IShellNameSpace interface 706
TShellNameSpace class 478
Set_UseSearchCompanion
ISearchAssistantOC3 interface 668
set_validate
IDOMParseOptions interface 7593
set_value
IDOMAttr interface 7524
set_Version
IDOMXMLProlog interface 7606
Set_Visible
IWebBrowserApp interface 782
TInternetExplorer class 447
Set_Width
IWebBrowser interface 744
TInternetExplorer class 447
SetActionName
IWebSetActionName interface 1115
TCustomAdapterAction class 5586
TCustomAdapterActionButton class 6437
SetActions
ISetSubComponents interface 5890
TCustomAdapter class 5566
SetActive
IXMLDocument interface 7657
TAbstractSocket class 19
TSQLMonitor class 1647
TXMLDocument class 7375
SetAdapter
TAdapterControlGroup class 6352

SetAdapterMode
IAdapterMode interface 1011
SetAdapterRequest
TAbstractWebContext class 5970
TWebContext class 5986
SetAdapterResponse
TAbstractWebContext class 5970
TWebContext class 5986
SetAddress
TAbstractSocket class 20
SetAlignment
TCustomEdit class 2361
SetAppDispatcher
TBaseRequestHandler class 6167
TCustomXSLPageProducer class 7891
SetAppServer
TXMLBroker class 7319
SetAppServices
TCustomWebAppPageModule class 6549
SetAsDate
TXSDate class 7825
SetAsDateTime
TXSCustomDateTime class 7811
SetAsTime
TXSTime class 7854
SetAsUTCDateTime
TXSCustomDateTime class 7811
SetAsyncStyles
TCustomWinSocket class 81
setAttribute
IDOMElement interface 7547
SetAttribute
IXMLNode interface 7684
TXMLNode class 7432
setAttributeNode
IDOMElement interface 7547
setAttributeNodeNS
IDOMElement interface 7548
SetAttributeNodes
TXMLNode class 7432
setAttributeNS
IDOMElement interface 7548
SetAttributeNS
IXMLNode interface 7684
TXMLNode class 7433
SetAutoSize
TCustomEdit class 2361
TCustomLabel class 2388
TCustomStaticText class 2460
SetBinding
IPort interface 7144
TPort class 7024
SetBounds
TShadowWindow class 404
SetBufListSize
TCustomSQLDataSet class 1548
SetButtonStyle
TCustomButton class 2265
SetBytesReceived
TBaseSocket class 1382
SetBytesSent
TBaseSocket class 1382
SetCapacity
TAbstractNamedVariants class 6130
TNamedVariantsList class 6139
TWideStringList class 6820
TWideStrings class 6845
SetCaption
TAdapterDataDisplay class 6362
TCustomAdapterActionButton class 6437
SetCaretPos
TCustomMemo class 2449
SetCell
TValueListEditor class 4799
SetChecked
TButtonActionLink class 2108
TButtonControl class 2112
TCustomCheckBox class 2280

TRadioButton class 2817
SetChildNodes
TXMLNode class 7433
TXMLNodeCollection class 7456
SetChildOrder
TWebContainedContainerComponent class 6059
TWebContainerComponent class 6066
TWebContainerComponentHelper class 6072
SetChildValue
IXMLNode interface 7685
TXMLNode class 7434
SetClientType
TClientSocket class 27
TClientWinSocket class 38
SetCollection
IXMLNodeAccess interface 7486
TXMLNode class 7434
SetCommandText
TCustomSQLDataSet class 1549
SetComponentIndex
TWebComponentList class 6032
SetComponentList
IWebComponent interface 6095
TWebContainedComponentHelper class 6049
SetConnected
ITransport interface 280
TSocketTransport class 234
TWebConnection class 260
TXMLBroker class 7319
SetConnection
TSimpleDataSet class 903
SetContainer
IWebComponent interface 6095
TWebContainedComponentHelper class 6050
SetContent
TScriptProducer class 6713
SetData
DynamicArray template 3268
ISetSubComponents interface 5890
TCustomAdapter class 5566
SetDay
TXSCustomDateTime class 7811
TXSDate class 7825
SetDecimalSecond
TXSDuration class 7839
SetDefaultPageFileName
IDefaultPageFileName interface 1015
TSitePageModuleHelper class 6582
SetDefaultSearchUrl
ISearchAssistantOC interface 654
ISearchAssistantOC2Disp interface 664
ISearchAssistantOC3Disp interface 674
ISearchAssistantOCDisp interface 682
SetDefaultWebComponents
TAdapterDefaultActions class 5236
TAdapterDefaultFields class 5242
TAdapterSubComponent class 5416
TWebContainedContainerComponent class 6060
TWebContainerComponent class 6067
TWebContainerComponentHelper class 6073
SetDefinition
IWSDLDocument interface 7166
TWSDLDocument class 7070
SetDescription
TScriptError class 6681
SetDesignState
TWSDLView class 7273
setDetailsPaneOpen
IScriptErrorList interface 639
IScriptErrorListDisp interface 643
SetDispatchedPageName
TAbstractWebContext class 5970
TWebContext class 5986
SetDisplayLabel
TAdapterNamedDisplayField class 5349
TBaseAdapterAction class 5472
SetDocumentElement
IXMLDocument interface 7657

TXMLDocument class 7375
SetDOMDocument
IXMLDocument interface 7658
TXMLDocument class 7376
SetDropDownCount
TCustomCombo class 2301
SetDroppedDown
TCustomCombo class 2302
SetEchoActionFieldValue
IEchoActionFieldValue interface 5881
TAdapterNamedField class 5366
SetEchoActionFieldValues
IEchoAdapterFieldValues interface 5883
TCustomAdapter class 5567
SetEditText
TValueListEditor class 4800
SetElement
IPart interface 7137
TPart class 7011
SetEncoding
IXMLDocument interface 7658
TXMLDocument class 7376
SetEncodingStyle
IHeaderInfo interface 7249
SetError
Variant class 3435
SetEvent
TEvent class 3137
SetFactory
TBaseWebPageInfo class 6290
SetFailurePage
IAdapterItemRequestParams interface 1007
TAdapterItemRequestParams class 6164
SetFCommandText
TCustomSQLDataSet class 1549
SetFieldData
TCustomSQLDataSet class 1549
SetFieldName
IWebSetFieldName interface 1116
TAdapterDataDisplay class 6362
TAdapterNamedField class 5367
SetFileName
IXMLDocument interface 7658
TSQLMonitor class 1647
TXMLDocument class 7376
SetFractionalSeconds
TXSCustomDateTime class 7811
TXSTime class 7854
SetGetSocketEvent
TCustomServerSocket class 46
SetGetThreadEvent
TCustomServerSocket class 46
SetHost
TAbstractSocket class 20
SetHour
TXSCustomDateTime class 7812
TXSTime class 7854
SetHourOffset
TXSCustomDateTime class 7812
TXSTime class 7854
SetHTMLDoc
TCustomAdapterPageProducer class 1163
SetHTMLFile
TCustomAdapterPageProducer class 1164
SetImageIndex
TPushButtonActionLink class 2812
SetIndex
IWebComponent interface 6095
TWebContainedComponentHelper class 6050
SetIndexField
TSQLTable class 1687
SetInnerException
Exception class 3962
SetItemData
TCustomListBox class 2417
SetItemHeight
TCustomCombo class 2302
SetItemIndex

TCustomCombo class 2302
TCustomListBox class 2417
SetItems
TCustomCombo class 2303
SetItemValue
TSessions class 393
SetLength
AnsiStringBase class 3199
AnsiStringT template 3227
UnicodeString class 3407
WideString class 3483
SetLines
TCustomMemo class 2449
SetLocation
IImport interface 7112
TImport class 6964
SetManager
TScreenTipItem class 331
SetMaxBlobSize
TCustomSQLDataSet class 1549
SetMaxLength
TCustomCombo class 2303
SetMessage
IParam interface 7133
TParam class 7004
SetMillisecond
TXSCustomDateTime class 7812
TXSTime class 7854
SetMimeBoundary
IWebNode interface 6619
THTTPReqResp class 1287
TLinkedWebNode class 1325
TWSDLClientNode class 7270
SetMIMEBoundary
IMimeAttachmentHandler interface 1181
SetMinute
TXSCustomDateTime class 7813
TXSTime class 7855
SetMinuteOffset
TXSCustomDateTime class 7813
TXSTime class 7855
SetModified
TXMLDocument class 7377
SetMonth
TXSCustomDateTime class 7813
TXSDate class 7826
SetMultiSelect
TCustomListBox class 2417
SetName
IBinding interface 7081
IBindingOperation interface 7086
IDefinition interface 7098
IHeaderInfo interface 7250
IMessage interface 7119
IOperation interface 7126
IParam interface 7133
IPart interface 7137
IPort interface 7144
IPortType interface 7151
IQualifiedName interface 7254
IService interface 7157
TBinding class 6909
TBindingOperation class 6916
TDefinition class 6938
TMessage class 6977
TOperation class 6991
TParam class 7004
TPart class 7011
TPort class 7024
TPortType class 7037
TService class 7051
setNamedItem
IDOMNamedNodeMap interface 7556
setNamedItemNS
IDOMNamedNodeMap interface 7556
SetNamespace
IHeaderInfo interface 7250
IImport interface 7112

IQualifiedName interface 7254
TImport class 6964
SetNodeIndentStr
IXMLDocument interface 7659
TXMLDocument class 7377
SetNodeValue
IXMLNode interface 7685
TXMLNode class 7435
SetOnAsyncLoad
IXMLDocument interface 7659
TXMLDocument class 7377
SetOnClientError
TCustomServerSocket class 47
SetOnClientEvent
TCustomServerSocket class 47
SetOnGetAttachmentEvent
IMimeAttachmentHandler interface 1181
SetOnModuleAdded
TWebModuleList class 6633
SetOnSendAttachmentEvent
IMimeAttachmentHandler interface 1182
SetOnThreadEnd
TCustomServerSocket class 47
SetOnThreadStart
TCustomServerSocket class 48
SetOptions
IXMLDocument interface 7659
TXMLDocument class 7378
SetPageIndex
TTabbedNotebook class 4412
SetPageSize
TDefaultFieldsPagedAdapter class 5759
setParameter
IDOMXSLProcessor interface 7609
SetParams
TScrollBar class 2885
TXMLBroker class 7319
TXMLTransformClient class 7743
SetParamsFromCursor
TCustomSQLDataSet class 1549
SetParentComponent
TAdapterSubComponent class 5417
TWebContainedComponent class 6043
TWebContainedContainerComponent class 6060
SetParentNode
IXMLNodeAccess interface 7487
TXMLNode class 7435
SetParseOptions
IXMLDocument interface 7660
TXMLDocument class 7378
SetPart
IHeaderInfo interface 7250
setPerErrorDisplay
IScriptErrorList interface 639
IScriptErrorListDisp interface 643
SetPort
TAbstractSocket class 20
SetPrologValue
TXMLDocument class 7378
SetProperty
IVarInvokeable interface 5026
TInvokeableVariantType class 5021
TPublishableVariantType class 4699
SetProvider
TXMLBroker class 7320
SetProviderName
TXMLBroker class 7320
SetReadOnly
IXMLNode interface 7686
TXMLNode class 7436
SetReconcileProducer
TXMLBroker class 7320
SetRemoteServer
TXMLBroker class 7321
SetRequestedPageName
TAbstractWebContext class 5970
TWebContext class 5986
SetRequired

IHeaderInfo interface 7250
SetRoot
IShellFavoritesNameSpace interface 694
IShellFavoritesNameSpaceDisp interface 699
IShellNameSpaceDisp interface 713
TShellNameSpace class 479
SetSchemaInfo
TCustomSQLDataSet class 1550
SetScrollBars
TCustomMemo class 2449
SetSecond
TXSCustomDateTime class 7814
TXSTime class 7855
SetSecureLockIcon
DWebBrowserEvents2 interface 633
SetSelLength
TCustomCombo class 2303
TCustomEdit class 2361
SetSelStart
TCustomCombo class 2304
TCustomEdit class 2362
SetSelText
TCustomEdit class 2362
SetSelTextBuf
TCustomEdit class 2363
SetServerType
TCustomServerSocket class 48
TServerWinSocket class 128
SetService
TAbstractSocket class 21
SetSignature
IDataBlock interface 271
TDataBlock class 203
SetSize
IDataBlock interface 271
TDataBlock class 204
SetSOAPServerIID
TSoapConnection class 1195
SetSortFieldNames
TCustomSQLDataSet class 1551
SetSpinCount
TMonitor record 3353
SetSQLConnection
TSQLMonitor class 1647
SetStackInfo
Exception class 3963
SetStandAlone
IXMLDocument interface 7660
TXMLDocument class 7378
SetStreamedActive
TSQLMonitor class 1648
SetStrings
TValueListEditor class 4800
SetStyle
TCustomComboBox class 2331
SetStyles
TBaseAdapterPageProducer class 1155
SetStylesFile
TBaseAdapterPageProducer class 1156
SetSuccessPage
IAdapterItemRequestParams interface 1008
TAdapterItemRequestParams class 6164
SetTabIndex
TTabSet class 4480
SetTargetNamespace
IDefinition interface 7098
TDefinition class 6938
SetText
IXMLNode interface 7686
TWideStrings class 6845
TXMLNode class 7436
SetTextStr
TWideStrings class 6846
SetThreadCacheSize
TCustomServerSocket class 49
TServerWinSocket class 128
SetTimeoutMinutes
IWebSessionAttributes interface 1106

TInvalidWebSession class 6219
TSessions class 393
TWebSession class 6254
TWebSessionImpl class 6769
SetTraceEvent
TSQLConnection class 1605
SetTraceList
TSQLMonitor class 1648
SetType_
IBinding interface 7081
IPart interface 7137
TBinding class 6909
TPart class 7011
SetupAppServer
TXMLTransformClient class 7744
SetUpdateState
TAbstractNamedVariants class 6130
TCustomComboBoxStrings class 2344
TWideStringList class 6821
TWideStrings class 6846
SetupDialog
TCommonDialogAction class 1789
SetUse
IHeaderInfo interface 7251
SetUseSoapAdapter
TSoapConnection class 1195
SetUseZeroMilliseconds
TXSCustomDateTime class 7814
SetValue
ISetWebFieldValue interface 5891
IWebSessionValues interface 1113
TInvalidWebSession class 6219
TWebSession class 6254
TWebSessionImpl class 6769
SetVersion
IXMLDocument interface 7660
TXMLDocument class 7379
SetViewType
IShellNameSpace interface 706
IShellNameSpaceDisp interface 713
TShellNameSpace class 479
SetWebContentOptions
ISetWebContentOptions interface 6092
TBaseAdapterPageProducer class 1156
TSitePageModuleHelper class 6582
SetWebDispatch
TXMLBroker class 7321
SetWebDisplayComponent
TAdapterDisplayReferenceGroup class 6386
SetWebParent
TAdapterActions class 5213
TAdapterFields class 5280
TAdapterSubComponent class 5417
TWebContainedComponent class 6043
TWebContainedComponentHelper class 6050
TWebContainedContainerComponent class 6061
SetWordWrap
TCustomMemo class 2450
SetXML
IXMLDocument interface 7661
TXMLDocument class 7379
SetXMLStrings
TXMLDocument class 7379
SetYear
TXSCustomDateTime class 7814
TXSDate class 7826
seWrite enumeration member 148
sffCurrency enumeration member 3180, 3388, 3471
sffExponent enumeration member 3180, 3388, 3471
sffFixed enumeration member 3180, 3388, 3471
sffGeneral enumeration member 3180, 3388, 3471
sffNumber enumeration member 3180, 3388, 3471
ShadowWnd 399
ShadowWnd namespace 399
Classes 400
Structs, Records, Enums 406
ShadowWnd.TControlSide 406
ShadowWnd.TControlSide enumeration 406

ShadowWnd.TShadowWindow 400
ShadowWnd.TShadowWindow.Control 405
ShadowWnd.TShadowWindow.Create 401
ShadowWnd.TShadowWindow.CreateParams 402
ShadowWnd.TShadowWindow.CreateShadow 401
ShadowWnd.TShadowWindow.Destroy 401
ShadowWnd.TShadowWindow.Notification 403
ShadowWnd.TShadowWindow.Paint 403
ShadowWnd.TShadowWindow.SetBounds 404
ShadowWnd.TShadowWindow.Side 405
ShadowWnd.TShadowWindow.WMEraseBkgnd 404
SHDocVw 407
SHDocVw namespace 407
Classes 413
Constants 838
Interfaces 616
Types 802
SHDocVw._SearchAssistantEvents 616
SHDocVw._SearchAssistantEvents.OnNewSearch 617
SHDocVw._SearchAssistantEvents.OnNextMenuSelect 617
SHDocVw.BrowserNavConstants 804
SHDocVw.BrowserNavConstants type 804
SHDocVw.CGID_IWebBrowser 840
SHDocVw.CGID_IWebBrowser constant 840
SHDocVw.CLASS_CScriptErrorList 841
SHDocVw.CLASS_CScriptErrorList constant 841
SHDocVw.CLASS_InternetExplorer 841
SHDocVw.CLASS_InternetExplorer constant 841
SHDocVw.CLASS_SearchAssistantOC 841
SHDocVw.CLASS_SearchAssistantOC constant 841
SHDocVw.CLASS_ShellBrowserWindow 842
SHDocVw.CLASS_ShellBrowserWindow constant 842
SHDocVw.CLASS_ShellNameSpace 842
SHDocVw.CLASS_ShellNameSpace constant 842
SHDocVw.CLASS_ShellUIHelper 842
SHDocVw.CLASS_ShellUIHelper constant 842
SHDocVw.CLASS_ShellWindows 843
SHDocVw.CLASS_ShellWindows constant 843
SHDocVw.CLASS_WebBrowser 843
SHDocVw.CLASS_WebBrowser constant 843
SHDocVw.CLASS_WebBrowser_V1 843
SHDocVw.CLASS_WebBrowser_V1 constant 843
SHDocVw.CoCScriptErrorList 413
SHDocVw.CoCScriptErrorList.Create 414
SHDocVw.CoCScriptErrorList.CreateRemote 414
SHDocVw.CoInternetExplorer 414
SHDocVw.CoInternetExplorer.Create 415
SHDocVw.CoInternetExplorer.CreateRemote 415
SHDocVw.CommandStateChangeConstants 804
SHDocVw.CommandStateChangeConstants type 804
SHDocVw.CoSearchAssistantOC 415
SHDocVw.CoSearchAssistantOC.Create 416
SHDocVw.CoSearchAssistantOC.CreateRemote 416
SHDocVw.CoShellBrowserWindow 417
SHDocVw.CoShellBrowserWindow.Create 417
SHDocVw.CoShellBrowserWindow.CreateRemote 418
SHDocVw.CoShellNameSpace 418
SHDocVw.CoShellNameSpace.Create 419
SHDocVw.CoShellNameSpace.CreateRemote 419
SHDocVw.CoShellUIHelper 419
SHDocVw.CoShellUIHelper.Create 420
SHDocVw.CoShellUIHelper.CreateRemote 420
SHDocVw.CoShellWindows 420
SHDocVw.CoShellWindows.Create 421
SHDocVw.CoShellWindows.CreateRemote 421
SHDocVw.CSC_NAVIGATEBACK 844
SHDocVw.CSC_NAVIGATEBACK constant 844
SHDocVw.CSC_NAVIGATEFORWARD 844
SHDocVw.CSC_NAVIGATEFORWARD constant 844
SHDocVw.CSC_UPDATECOMMANDS 844
SHDocVw.CSC_UPDATECOMMANDS constant 844
SHDocVw.CScriptErrorList 804
SHDocVw.CScriptErrorList type 804
SHDocVw.DIID__SearchAssistantEvents 845
SHDocVw.DIID__SearchAssistantEvents constant 845
SHDocVw.DIID_DShellNameSpaceEvents 845
SHDocVw.DIID_DShellNameSpaceEvents constant 845
SHDocVw.DIID_DShellWindowsEvents 845

SHDocVw.DIID_DShellWindowsEvents constant 845
SHDocVw.DIID_DWebBrowserEvents 846
SHDocVw.DIID_DWebBrowserEvents constant 846
SHDocVw.DIID_DWebBrowserEvents2 846
SHDocVw.DIID_DWebBrowserEvents2 constant 846
SHDocVw.DShellNameSpaceEvents 618
SHDocVw.DShellNameSpaceEvents.DoubleClick 618
SHDocVw.DShellNameSpaceEvents.FavoritesSelectionChange 619
SHDocVw.DShellNameSpaceEvents.Initialized 619
SHDocVw.DShellNameSpaceEvents.SelectionChange 619
SHDocVw.DShellWindowsEvents 619
SHDocVw.DShellWindowsEvents.WindowRegistered 620
SHDocVw.DShellWindowsEvents.WindowRevoked 620
SHDocVw.dtlOcxPage 846
SHDocVw.dtlOcxPage constant 846
SHDocVw.dtlServerPage 847
SHDocVw.dtlServerPage constant 847
SHDocVw.DWebBrowserEvents 621
SHDocVw.DWebBrowserEvents.BeforeNavigate 622
SHDocVw.DWebBrowserEvents.CommandStateChange 622
SHDocVw.DWebBrowserEvents.DownloadBegin 622
SHDocVw.DWebBrowserEvents.DownloadComplete 623
SHDocVw.DWebBrowserEvents.FrameBeforeNavigate 623
SHDocVw.DWebBrowserEvents.FrameNavigateComplete 623
SHDocVw.DWebBrowserEvents.FrameNewWindow 623
SHDocVw.DWebBrowserEvents.NavigateComplete 624
SHDocVw.DWebBrowserEvents.NewWindow 624
SHDocVw.DWebBrowserEvents.ProgressChange 624
SHDocVw.DWebBrowserEvents.PropertyChange 624
SHDocVw.DWebBrowserEvents.Quit 624
SHDocVw.DWebBrowserEvents.StatusTextChange 625
SHDocVw.DWebBrowserEvents.TitleChange 625
SHDocVw.DWebBrowserEvents.WindowActivate 625
SHDocVw.DWebBrowserEvents.WindowMove 625
SHDocVw.DWebBrowserEvents.WindowResize 626
SHDocVw.DWebBrowserEvents2 626
SHDocVw.DWebBrowserEvents2.BeforeNavigate2 628
SHDocVw.DWebBrowserEvents2.ClientToHostWindow 628
SHDocVw.DWebBrowserEvents2.CommandStateChange 628
SHDocVw.DWebBrowserEvents2.DocumentComplete 628
SHDocVw.DWebBrowserEvents2.DownloadBegin 629
SHDocVw.DWebBrowserEvents2.DownloadComplete 629
SHDocVw.DWebBrowserEvents2.FileDownload 629
SHDocVw.DWebBrowserEvents2.NavigateComplete2 629
SHDocVw.DWebBrowserEvents2.NavigateError 629
SHDocVw.DWebBrowserEvents2.NewWindow2 630
SHDocVw.DWebBrowserEvents2.OnFullScreen 630
SHDocVw.DWebBrowserEvents2.OnMenuBar 630
SHDocVw.DWebBrowserEvents2.OnQuit 630
SHDocVw.DWebBrowserEvents2.OnStatusBar 631
SHDocVw.DWebBrowserEvents2.OnTheaterMode 631
SHDocVw.DWebBrowserEvents2.OnToolBar 631
SHDocVw.DWebBrowserEvents2.OnVisible 631
SHDocVw.DWebBrowserEvents2.PrintTemplateInstantiation 631
SHDocVw.DWebBrowserEvents2.PrintTemplateTeardown 632
SHDocVw.DWebBrowserEvents2.PrivacyImpactedStateChange 632
SHDocVw.DWebBrowserEvents2.ProgressChange 632
SHDocVw.DWebBrowserEvents2.PropertyChange 632
SHDocVw.DWebBrowserEvents2.SetSecureLockIcon 633
SHDocVw.DWebBrowserEvents2.StatusTextChange 633
SHDocVw.DWebBrowserEvents2.TitleChange 633
SHDocVw.DWebBrowserEvents2.UpdatePageStatus 633
SHDocVw.DWebBrowserEvents2.WindowClosing 633
SHDocVw.DWebBrowserEvents2.WindowSetHeight 634
SHDocVw.DWebBrowserEvents2.WindowSetLeft 634
SHDocVw.DWebBrowserEvents2.WindowSetResizable 634
SHDocVw.DWebBrowserEvents2.WindowSetTop 634
SHDocVw.DWebBrowserEvents2.WindowSetWidth 635
SHDocVw.HTMLID_FIND 847
SHDocVw.HTMLID_FIND constant 847
SHDocVw.HTMLID_OPTIONS 847
SHDocVw.HTMLID_OPTIONS constant 847
SHDocVw.HTMLID_VIEWSOURCE 848
SHDocVw.HTMLID_VIEWSOURCE constant 848
SHDocVw.IID_IScriptErrorList 848
SHDocVw.IID_IScriptErrorList constant 848

SHDocVw.IID_ISearch 848
SHDocVw.IID_ISearch constant 848
SHDocVw.IID_ISearchAssistantOC 849
SHDocVw.IID_ISearchAssistantOC constant 849
SHDocVw.IID_ISearchAssistantOC2 849
SHDocVw.IID_ISearchAssistantOC2 constant 849
SHDocVw.IID_ISearchAssistantOC3 849
SHDocVw.IID_ISearchAssistantOC3 constant 849
SHDocVw.IID_ISearches 850
SHDocVw.IID_ISearches constant 850
SHDocVw.IID_IShellFavoritesNameSpace 850
SHDocVw.IID_IShellFavoritesNameSpace constant 850
SHDocVw.IID_IShellNameSpace 850
SHDocVw.IID_IShellNameSpace constant 850
SHDocVw.IID_IShellUIHelper 851
SHDocVw.IID_IShellUIHelper constant 851
SHDocVw.IID_IShellWindows 851
SHDocVw.IID_IShellWindows constant 851
SHDocVw.IID_IWebBrowser 851
SHDocVw.IID_IWebBrowser constant 851
SHDocVw.IID_IWebBrowser2 852
SHDocVw.IID_IWebBrowser2 constant 852
SHDocVw.IID_IWebBrowserApp 852
SHDocVw.IID_IWebBrowserApp constant 852
SHDocVw.InternetExplorer 805
SHDocVw.InternetExplorer type 805
SHDocVw.IScriptErrorList 635
SHDocVw.IScriptErrorList.advanceError 636
SHDocVw.IScriptErrorList.canAdvanceError 636
SHDocVw.IScriptErrorList.canRetreatError 637
SHDocVw.IScriptErrorList.getAlwaysShowLockState 637
SHDocVw.IScriptErrorList.getDetailsPaneOpen 637
SHDocVw.IScriptErrorList.getErrorChar 637
SHDocVw.IScriptErrorList.getErrorCode 637
SHDocVw.IScriptErrorList.getErrorLine 638
SHDocVw.IScriptErrorList.getErrorMsg 638
SHDocVw.IScriptErrorList.getErrorUrl 638
SHDocVw.IScriptErrorList.getPerErrorDisplay 638
SHDocVw.IScriptErrorList.retreatError 639
SHDocVw.IScriptErrorList.setDetailsPaneOpen 639
SHDocVw.IScriptErrorList.setPerErrorDisplay 639
SHDocVw.IScriptErrorListDisp 639
SHDocVw.IScriptErrorListDisp.advanceError 640
SHDocVw.IScriptErrorListDisp.canAdvanceError 641
SHDocVw.IScriptErrorListDisp.canRetreatError 641
SHDocVw.IScriptErrorListDisp.getAlwaysShowLockState 641
SHDocVw.IScriptErrorListDisp.getDetailsPaneOpen 641
SHDocVw.IScriptErrorListDisp.getErrorChar 642
SHDocVw.IScriptErrorListDisp.getErrorCode 642
SHDocVw.IScriptErrorListDisp.getErrorLine 642
SHDocVw.IScriptErrorListDisp.getErrorMsg 642
SHDocVw.IScriptErrorListDisp.getErrorUrl 642
SHDocVw.IScriptErrorListDisp.getPerErrorDisplay 643
SHDocVw.IScriptErrorListDisp.retreatError 643
SHDocVw.IScriptErrorListDisp.setDetailsPaneOpen 643
SHDocVw.IScriptErrorListDisp.setPerErrorDisplay 643
SHDocVw.ISearch 644
SHDocVw.ISearch.Get_Id 645
SHDocVw.ISearch.Get_Title 645
SHDocVw.ISearch.Get_URL 645
SHDocVw.ISearch.Id 645
SHDocVw.ISearch.Title 646
SHDocVw.ISearch.URL 646
SHDocVw.ISearchAssistantOC 646
SHDocVw.ISearchAssistantOC.AddNextMenuItem 648
SHDocVw.ISearchAssistantOC.ASProvider 654
SHDocVw.ISearchAssistantOC.ASSetting 655
SHDocVw.ISearchAssistantOC.EncodeString 648
SHDocVw.ISearchAssistantOC.EventHandled 655
SHDocVw.ISearchAssistantOC.FindComputer 649
SHDocVw.ISearchAssistantOC.FindFilesOrFolders 649
SHDocVw.ISearchAssistantOC.FindOnWeb 649
SHDocVw.ISearchAssistantOC.FindPeople 649
SHDocVw.ISearchAssistantOC.FindPrinter 649
SHDocVw.ISearchAssistantOC.Get_ASProvider 650
SHDocVw.ISearchAssistantOC.Get_ASSetting 650
SHDocVw.ISearchAssistantOC.Get_InWebFolder 650
SHDocVw.ISearchAssistantOC.Get_SearchAssistantDefault 650

SHDocVw.ISearchAssistantOC.Get_Searches 651
SHDocVw.ISearchAssistantOC.Get_ShellFeaturesEnabled 651
SHDocVw.ISearchAssistantOC.Get_Version 651
SHDocVw.ISearchAssistantOC.GetProperty 651
SHDocVw.ISearchAssistantOC.GetSearchAssistantURL 651
SHDocVw.ISearchAssistantOC.InWebFolder 655
SHDocVw.ISearchAssistantOC.IsRestricted 652
SHDocVw.ISearchAssistantOC.NavigateToDefaultSearch 652
SHDocVw.ISearchAssistantOC.NETDetectNextNavigate 652
SHDocVw.ISearchAssistantOC.NotifySearchSettingsChanged 652
SHDocVw.ISearchAssistantOC.PutFindText 653
SHDocVw.ISearchAssistantOC.PutProperty 653
SHDocVw.ISearchAssistantOC.ResetNextMenu 653
SHDocVw.ISearchAssistantOC.SearchAssistantDefault 655
SHDocVw.ISearchAssistantOC.Searches 656
SHDocVw.ISearchAssistantOC.Set_ASProvider 653
SHDocVw.ISearchAssistantOC.Set_ASSetting 653
SHDocVw.ISearchAssistantOC.Set_EventHandled 654
SHDocVw.ISearchAssistantOC.SetDefaultSearchUrl 654
SHDocVw.ISearchAssistantOC.ShellFeaturesEnabled 656
SHDocVw.ISearchAssistantOC.Version 656
SHDocVw.ISearchAssistantOC2 656
SHDocVw.ISearchAssistantOC2.Get_ShowFindPrinter 658
SHDocVw.ISearchAssistantOC2.ShowFindPrinter 658
SHDocVw.ISearchAssistantOC2Disp 658
SHDocVw.ISearchAssistantOC2Disp.AddNextMenuItem 660
SHDocVw.ISearchAssistantOC2Disp.ASProvider 664
SHDocVw.ISearchAssistantOC2Disp.ASSetting 664
SHDocVw.ISearchAssistantOC2Disp.EncodeString 660
SHDocVw.ISearchAssistantOC2Disp.EventHandled 665
SHDocVw.ISearchAssistantOC2Disp.FindComputer 660
SHDocVw.ISearchAssistantOC2Disp.FindFilesOrFolders 661
SHDocVw.ISearchAssistantOC2Disp.FindOnWeb 661
SHDocVw.ISearchAssistantOC2Disp.FindPeople 661
SHDocVw.ISearchAssistantOC2Disp.FindPrinter 661
SHDocVw.ISearchAssistantOC2Disp.GetProperty 662
SHDocVw.ISearchAssistantOC2Disp.GetSearchAssistantURL 662
SHDocVw.ISearchAssistantOC2Disp.InWebFolder 665
SHDocVw.ISearchAssistantOC2Disp.IsRestricted 662
SHDocVw.ISearchAssistantOC2Disp.NavigateToDefaultSearch 662
SHDocVw.ISearchAssistantOC2Disp.NETDetectNextNavigate 662
SHDocVw.ISearchAssistantOC2Disp.NotifySearchSettingsChanged 663
SHDocVw.ISearchAssistantOC2Disp.PutFindText 663
SHDocVw.ISearchAssistantOC2Disp.PutProperty 663
SHDocVw.ISearchAssistantOC2Disp.ResetNextMenu 663
SHDocVw.ISearchAssistantOC2Disp.SearchAssistantDefault 665
SHDocVw.ISearchAssistantOC2Disp.Searches 665
SHDocVw.ISearchAssistantOC2Disp.SetDefaultSearchUrl 664
SHDocVw.ISearchAssistantOC2Disp.ShellFeaturesEnabled 665
SHDocVw.ISearchAssistantOC2Disp.ShowFindPrinter 666
SHDocVw.ISearchAssistantOC2Disp.Version 666
SHDocVw.ISearchAssistantOC3 666
SHDocVw.ISearchAssistantOC3.Get_SearchCompanionAvailable 668
SHDocVw.ISearchAssistantOC3.Get_UseSearchCompanion 668
SHDocVw.ISearchAssistantOC3.SearchCompanionAvailable 669
SHDocVw.ISearchAssistantOC3.Set_UseSearchCompanion 668
SHDocVw.ISearchAssistantOC3.UseSearchCompanion 669
SHDocVw.ISearchAssistantOC3Disp 669
SHDocVw.ISearchAssistantOC3Disp.AddNextMenuItem 671
SHDocVw.ISearchAssistantOC3Disp.ASProvider 675
SHDocVw.ISearchAssistantOC3Disp.ASSetting 675
SHDocVw.ISearchAssistantOC3Disp.EncodeString 671
SHDocVw.ISearchAssistantOC3Disp.EventHandled 675
SHDocVw.ISearchAssistantOC3Disp.FindComputer 671
SHDocVw.ISearchAssistantOC3Disp.FindFilesOrFolders 671
SHDocVw.ISearchAssistantOC3Disp.FindOnWeb 672
SHDocVw.ISearchAssistantOC3Disp.FindPeople 672
SHDocVw.ISearchAssistantOC3Disp.FindPrinter 672
SHDocVw.ISearchAssistantOC3Disp.GetProperty 672
SHDocVw.ISearchAssistantOC3Disp.GetSearchAssistantURL 672

SHDocVw.ISearchAssistantOC3Disp.InWebFolder 676
SHDocVw.ISearchAssistantOC3Disp.IsRestricted 673
SHDocVw.ISearchAssistantOC3Disp.NavigateToDefaultSearch 673
SHDocVw.ISearchAssistantOC3Disp.NETDetectNextNavigate 673
SHDocVw.ISearchAssistantOC3Disp.NotifySearchSettingsChanged 673
SHDocVw.ISearchAssistantOC3Disp.PutFindText 674
SHDocVw.ISearchAssistantOC3Disp.PutProperty 674
SHDocVw.ISearchAssistantOC3Disp.ResetNextMenu 674
SHDocVw.ISearchAssistantOC3Disp.SearchAssistantDefault 676
SHDocVw.ISearchAssistantOC3Disp.SearchCompanionAvailable 676
SHDocVw.ISearchAssistantOC3Disp.Searches 676
SHDocVw.ISearchAssistantOC3Disp.SetDefaultSearchUrl 674
SHDocVw.ISearchAssistantOC3Disp.ShellFeaturesEnabled 676
SHDocVw.ISearchAssistantOC3Disp.ShowFindPrinter 677
SHDocVw.ISearchAssistantOC3Disp.UseSearchCompanion 677
SHDocVw.ISearchAssistantOC3Disp.Version 677
SHDocVw.ISearchAssistantOCDisp 677
SHDocVw.ISearchAssistantOCDisp.AddNextMenuItem 679
SHDocVw.ISearchAssistantOCDisp.ASProvider 683
SHDocVw.ISearchAssistantOCDisp.ASSetting 683
SHDocVw.ISearchAssistantOCDisp.EncodeString 679
SHDocVw.ISearchAssistantOCDisp.EventHandled 683
SHDocVw.ISearchAssistantOCDisp.FindComputer 679
SHDocVw.ISearchAssistantOCDisp.FindFilesOrFolders 679
SHDocVw.ISearchAssistantOCDisp.FindOnWeb 680
SHDocVw.ISearchAssistantOCDisp.FindPeople 680
SHDocVw.ISearchAssistantOCDisp.FindPrinter 680
SHDocVw.ISearchAssistantOCDisp.GetProperty 680
SHDocVw.ISearchAssistantOCDisp.GetSearchAssistantURL 681
SHDocVw.ISearchAssistantOCDisp.InWebFolder 684
SHDocVw.ISearchAssistantOCDisp.IsRestricted 681
SHDocVw.ISearchAssistantOCDisp.NavigateToDefaultSearch 681
SHDocVw.ISearchAssistantOCDisp.NETDetectNextNavigate 681
SHDocVw.ISearchAssistantOCDisp.NotifySearchSettingsChanged 681
SHDocVw.ISearchAssistantOCDisp.PutFindText 682
SHDocVw.ISearchAssistantOCDisp.PutProperty 682
SHDocVw.ISearchAssistantOCDisp.ResetNextMenu 682
SHDocVw.ISearchAssistantOCDisp.SearchAssistantDefault 684
SHDocVw.ISearchAssistantOCDisp.Searches 684
SHDocVw.ISearchAssistantOCDisp.SetDefaultSearchUrl 682
SHDocVw.ISearchAssistantOCDisp.ShellFeaturesEnabled 684
SHDocVw.ISearchAssistantOCDisp.Version 684
SHDocVw.ISearchDisp 685
SHDocVw.ISearchDisp.Id 685
SHDocVw.ISearchDisp.Title 686
SHDocVw.ISearchDisp.URL 686
SHDocVw.ISearches 686
SHDocVw.ISearches._NewEnum 687
SHDocVw.ISearches.Count 688
SHDocVw.ISearches.Default 688
SHDocVw.ISearches.Get_Count 687
SHDocVw.ISearches.Get_Default 688
SHDocVw.ISearches.Item 688
SHDocVw.ISearchesDisp 689
SHDocVw.ISearchesDisp._NewEnum 689
SHDocVw.ISearchesDisp.Count 690
SHDocVw.ISearchesDisp.Default 690
SHDocVw.ISearchesDisp.Item 690
SHDocVw.IShellFavoritesNameSpace 691
SHDocVw.IShellFavoritesNameSpace.CreateSubscriptionForSelection 692
SHDocVw.IShellFavoritesNameSpace.DeleteSubscriptionForSelection 692
SHDocVw.IShellFavoritesNameSpace.Export 692
SHDocVw.IShellFavoritesNameSpace.Get_SubscriptionsEnabled 693
SHDocVw.IShellFavoritesNameSpace.Import 693
SHDocVw.IShellFavoritesNameSpace.InvokeContextMenuCommand 693

SHDocVw.IShellFavoritesNameSpace.MoveSelectionDown 693
SHDocVw.IShellFavoritesNameSpace.MoveSelectionTo 694
SHDocVw.IShellFavoritesNameSpace.MoveSelectionUp 694
SHDocVw.IShellFavoritesNameSpace.NewFolder 694
SHDocVw.IShellFavoritesNameSpace.ResetSort 694
SHDocVw.IShellFavoritesNameSpace.SetRoot 694
SHDocVw.IShellFavoritesNameSpace.SubscriptionsEnabled 695
SHDocVw.IShellFavoritesNameSpace.Synchronize 695
SHDocVw.IShellFavoritesNameSpaceDisp 695
SHDocVw.IShellFavoritesNameSpaceDisp.CreateSubscriptionForSelection 697
SHDocVw.IShellFavoritesNameSpaceDisp.DeleteSubscriptionForSelection 697
SHDocVw.IShellFavoritesNameSpaceDisp.Export 697
SHDocVw.IShellFavoritesNameSpaceDisp.Import 697
SHDocVw.IShellFavoritesNameSpaceDisp.InvokeContextMenuCommand 698
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionDown 698
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionTo 698
SHDocVw.IShellFavoritesNameSpaceDisp.MoveSelectionUp 698
SHDocVw.IShellFavoritesNameSpaceDisp.NewFolder 698
SHDocVw.IShellFavoritesNameSpaceDisp.ResetSort 699
SHDocVw.IShellFavoritesNameSpaceDisp.SetRoot 699
SHDocVw.IShellFavoritesNameSpaceDisp.SubscriptionsEnabled 700
SHDocVw.IShellFavoritesNameSpaceDisp.Synchronize 699
SHDocVw.IShellNameSpace 700
SHDocVw.IShellNameSpace.Columns 707
SHDocVw.IShellNameSpace.CountViewTypes 707
SHDocVw.IShellNameSpace.Depth 707
SHDocVw.IShellNameSpace.EnumOptions 707
SHDocVw.IShellNameSpace.Expand 702
SHDocVw.IShellNameSpace.Flags 708
SHDocVw.IShellNameSpace.Get_Columns 702
SHDocVw.IShellNameSpace.Get_CountViewTypes 702
SHDocVw.IShellNameSpace.Get_Depth 702
SHDocVw.IShellNameSpace.Get_EnumOptions 703
SHDocVw.IShellNameSpace.Get_Flags 703
SHDocVw.IShellNameSpace.Get_Mode 703
SHDocVw.IShellNameSpace.Get_Root 703
SHDocVw.IShellNameSpace.Get_SelectedItem 704
SHDocVw.IShellNameSpace.Get_TVFlags 704
SHDocVw.IShellNameSpace.Mode 708
SHDocVw.IShellNameSpace.Root 708
SHDocVw.IShellNameSpace.SelectedItem 708
SHDocVw.IShellNameSpace.SelectedItems 704
SHDocVw.IShellNameSpace.Set_Columns 704
SHDocVw.IShellNameSpace.Set_Depth 704
SHDocVw.IShellNameSpace.Set_EnumOptions 705
SHDocVw.IShellNameSpace.Set_Flags 705
SHDocVw.IShellNameSpace.Set_Mode 705
SHDocVw.IShellNameSpace.Set_Root 705
SHDocVw.IShellNameSpace.Set_SelectedItem 706
SHDocVw.IShellNameSpace.Set_TVFlags 706
SHDocVw.IShellNameSpace.SetViewType 706
SHDocVw.IShellNameSpace.TVFlags 709
SHDocVw.IShellNameSpace.UnselectAll 706
SHDocVw.IShellNameSpaceDisp 709
SHDocVw.IShellNameSpaceDisp.Columns 714
SHDocVw.IShellNameSpaceDisp.CountViewTypes 714
SHDocVw.IShellNameSpaceDisp.CreateSubscriptionForSelection 710
SHDocVw.IShellNameSpaceDisp.DeleteSubscriptionForSelection 711
SHDocVw.IShellNameSpaceDisp.Depth 715
SHDocVw.IShellNameSpaceDisp.EnumOptions 715
SHDocVw.IShellNameSpaceDisp.Expand 711
SHDocVw.IShellNameSpaceDisp.Export 711
SHDocVw.IShellNameSpaceDisp.Flags 715
SHDocVw.IShellNameSpaceDisp.Import 711
SHDocVw.IShellNameSpaceDisp.InvokeContextMenuCommand 711
SHDocVw.IShellNameSpaceDisp.Mode 715
SHDocVw.IShellNameSpaceDisp.MoveSelectionDown 712

SHDocVw.IShellNameSpaceDisp.MoveSelectionTo 712
SHDocVw.IShellNameSpaceDisp.MoveSelectionUp 712
SHDocVw.IShellNameSpaceDisp.NewFolder 712
SHDocVw.IShellNameSpaceDisp.ResetSort 713
SHDocVw.IShellNameSpaceDisp.Root 716
SHDocVw.IShellNameSpaceDisp.SelectedItem 716
SHDocVw.IShellNameSpaceDisp.SelectedItems 713
SHDocVw.IShellNameSpaceDisp.SetRoot 713
SHDocVw.IShellNameSpaceDisp.SetViewType 713
SHDocVw.IShellNameSpaceDisp.SubscriptionsEnabled 716
SHDocVw.IShellNameSpaceDisp.Synchronize 713
SHDocVw.IShellNameSpaceDisp.TVFlags 716
SHDocVw.IShellNameSpaceDisp.UnselectAll 714
SHDocVw.IShellUIHelper 716
SHDocVw.IShellUIHelper.AddChannel 718
SHDocVw.IShellUIHelper.AddDesktopComponent 718
SHDocVw.IShellUIHelper.AddFavorite 718
SHDocVw.IShellUIHelper.AutoCompleteAttach 718
SHDocVw.IShellUIHelper.AutoCompleteSaveForm 719
SHDocVw.IShellUIHelper.AutoScan 719
SHDocVw.IShellUIHelper.ImportExportFavorites 719
SHDocVw.IShellUIHelper.IsSubscribed 719
SHDocVw.IShellUIHelper.NavigateAndFind 719
SHDocVw.IShellUIHelper.RefreshOfflineDesktop 720
SHDocVw.IShellUIHelper.ResetFirstBootMode 720
SHDocVw.IShellUIHelper.ResetSafeMode 720
SHDocVw.IShellUIHelper.ShowBrowserUI 720
SHDocVw.IShellUIHelperDisp 721
SHDocVw.IShellUIHelperDisp.AddChannel 722
SHDocVw.IShellUIHelperDisp.AddDesktopComponent 722
SHDocVw.IShellUIHelperDisp.AddFavorite 722
SHDocVw.IShellUIHelperDisp.AutoCompleteAttach 722
SHDocVw.IShellUIHelperDisp.AutoCompleteSaveForm 723
SHDocVw.IShellUIHelperDisp.AutoScan 723
SHDocVw.IShellUIHelperDisp.ImportExportFavorites 723
SHDocVw.IShellUIHelperDisp.IsSubscribed 723
SHDocVw.IShellUIHelperDisp.NavigateAndFind 724
SHDocVw.IShellUIHelperDisp.RefreshOfflineDesktop 724
SHDocVw.IShellUIHelperDisp.ResetFirstBootMode 724
SHDocVw.IShellUIHelperDisp.ResetSafeMode 724
SHDocVw.IShellUIHelperDisp.ShowBrowserUI 724
SHDocVw.IShellWindows 725
SHDocVw.IShellWindows._NewEnum 726
SHDocVw.IShellWindows.Count 729
SHDocVw.IShellWindows.FindWindowSW 726
SHDocVw.IShellWindows.Get_Count 726
SHDocVw.IShellWindows.Item 727
SHDocVw.IShellWindows.OnActivated 727
SHDocVw.IShellWindows.OnCreated 727
SHDocVw.IShellWindows.OnNavigate 727
SHDocVw.IShellWindows.ProcessAttachDetach 727
SHDocVw.IShellWindows.Register 728
SHDocVw.IShellWindows.RegisterPending 728
SHDocVw.IShellWindows.Revoke 728
SHDocVw.IShellWindowsDisp 729
SHDocVw.IShellWindowsDisp._NewEnum 730
SHDocVw.IShellWindowsDisp.Count 732
SHDocVw.IShellWindowsDisp.FindWindowSW 730
SHDocVw.IShellWindowsDisp.Item 730
SHDocVw.IShellWindowsDisp.OnActivated 731
SHDocVw.IShellWindowsDisp.OnCreated 731
SHDocVw.IShellWindowsDisp.OnNavigate 731
SHDocVw.IShellWindowsDisp.ProcessAttachDetach 731
SHDocVw.IShellWindowsDisp.Register 731
SHDocVw.IShellWindowsDisp.RegisterPending 732
SHDocVw.IShellWindowsDisp.Revoke 732
SHDocVw.IWebBrowser 733
SHDocVw.IWebBrowser.Application 745
SHDocVw.IWebBrowser.Busy 745
SHDocVw.IWebBrowser.Container 746
SHDocVw.IWebBrowser.Document 746
SHDocVw.IWebBrowser.Get_Application 735
SHDocVw.IWebBrowser.Get_Busy 735
SHDocVw.IWebBrowser.Get_Container 735
SHDocVw.IWebBrowser.Get_Document 736
SHDocVw.IWebBrowser.Get_Height 736
SHDocVw.IWebBrowser.Get_Left 736
SHDocVw.IWebBrowser.Get_LocationName 737

SHDocVw.IWebBrowser.Get_LocationURL 737
SHDocVw.IWebBrowser.Get_Parent 737
SHDocVw.IWebBrowser.Get_Top 738
SHDocVw.IWebBrowser.Get_TopLevelContainer 738
SHDocVw.IWebBrowser.Get_type_ 738
SHDocVw.IWebBrowser.Get_Width 739
SHDocVw.IWebBrowser.GoBack 739
SHDocVw.IWebBrowser.GoForward 739
SHDocVw.IWebBrowser.GoHome 740
SHDocVw.IWebBrowser.GoSearch 740
SHDocVw.IWebBrowser.Height 746
SHDocVw.IWebBrowser.Left 747
SHDocVw.IWebBrowser.LocationName 747
SHDocVw.IWebBrowser.LocationURL 747
SHDocVw.IWebBrowser.Navigate 741
SHDocVw.IWebBrowser.Parent 748
SHDocVw.IWebBrowser.Refresh 742
SHDocVw.IWebBrowser.Refresh2 742
SHDocVw.IWebBrowser.Set_Height 743
SHDocVw.IWebBrowser.Set_Left 743
SHDocVw.IWebBrowser.Set_Top 743
SHDocVw.IWebBrowser.Set_Width 744
SHDocVw.IWebBrowser.Stop 744
SHDocVw.IWebBrowser.Top 748
SHDocVw.IWebBrowser.TopLevelContainer 748
SHDocVw.IWebBrowser.type_ 749
SHDocVw.IWebBrowser.Width 749
SHDocVw.IWebBrowser2 749
SHDocVw.IWebBrowser2.AddressBar 760
SHDocVw.IWebBrowser2.ExecWB 752
SHDocVw.IWebBrowser2.Get_AddressBar 753
SHDocVw.IWebBrowser2.Get_Offline 753
SHDocVw.IWebBrowser2.Get_ReadyState 754
SHDocVw.IWebBrowser2.Get_RegisterAsBrowser 754
SHDocVw.IWebBrowser2.Get_RegisterAsDropTarget 754
SHDocVw.IWebBrowser2.Get_Resizable 754
SHDocVw.IWebBrowser2.Get_Silent 755
SHDocVw.IWebBrowser2.Get_TheaterMode 755
SHDocVw.IWebBrowser2.Navigate2 755
SHDocVw.IWebBrowser2.Offline 760
SHDocVw.IWebBrowser2.QueryStatusWB 757
SHDocVw.IWebBrowser2.ReadyState 760
SHDocVw.IWebBrowser2.RegisterAsBrowser 761
SHDocVw.IWebBrowser2.RegisterAsDropTarget 761
SHDocVw.IWebBrowser2.Resizable 762
SHDocVw.IWebBrowser2.Set_AddressBar 757
SHDocVw.IWebBrowser2.Set_Offline 757
SHDocVw.IWebBrowser2.Set_RegisterAsBrowser 758
SHDocVw.IWebBrowser2.Set_RegisterAsDropTarget 758
SHDocVw.IWebBrowser2.Set_Resizable 758
SHDocVw.IWebBrowser2.Set_Silent 759
SHDocVw.IWebBrowser2.Set_TheaterMode 759
SHDocVw.IWebBrowser2.ShowBrowserBar 759
SHDocVw.IWebBrowser2.Silent 762
SHDocVw.IWebBrowser2.TheaterMode 762
SHDocVw.IWebBrowser2Disp 763
SHDocVw.IWebBrowser2Disp.AddressBar 769
SHDocVw.IWebBrowser2Disp.Application 769
SHDocVw.IWebBrowser2Disp.Busy 769
SHDocVw.IWebBrowser2Disp.ClientToWindow 764
SHDocVw.IWebBrowser2Disp.Container 769
SHDocVw.IWebBrowser2Disp.Document 770
SHDocVw.IWebBrowser2Disp.ExecWB 765
SHDocVw.IWebBrowser2Disp.FullName 770
SHDocVw.IWebBrowser2Disp.FullScreen 770
SHDocVw.IWebBrowser2Disp.GetProperty 765
SHDocVw.IWebBrowser2Disp.GoBack 765
SHDocVw.IWebBrowser2Disp.GoForward 765
SHDocVw.IWebBrowser2Disp.GoHome 766
SHDocVw.IWebBrowser2Disp.GoSearch 766
SHDocVw.IWebBrowser2Disp.Height 770
SHDocVw.IWebBrowser2Disp.HWnd 771
SHDocVw.IWebBrowser2Disp.Left 771
SHDocVw.IWebBrowser2Disp.LocationName 771
SHDocVw.IWebBrowser2Disp.LocationURL 771
SHDocVw.IWebBrowser2Disp.MenuBar 771
SHDocVw.IWebBrowser2Disp.Name 772
SHDocVw.IWebBrowser2Disp.Navigate 766

SHDocVw.IWebBrowser2Disp.Navigate2 766
SHDocVw.IWebBrowser2Disp.Offline 772
SHDocVw.IWebBrowser2Disp.Parent 772
SHDocVw.IWebBrowser2Disp.Path 772
SHDocVw.IWebBrowser2Disp.PutProperty 767
SHDocVw.IWebBrowser2Disp.QueryStatusWB 767
SHDocVw.IWebBrowser2Disp.Quit 767
SHDocVw.IWebBrowser2Disp.ReadyState 773
SHDocVw.IWebBrowser2Disp.Refresh 767
SHDocVw.IWebBrowser2Disp.Refresh2 767
SHDocVw.IWebBrowser2Disp.RegisterAsBrowser 773
SHDocVw.IWebBrowser2Disp.RegisterAsDropTarget 773
SHDocVw.IWebBrowser2Disp.Resizable 773
SHDocVw.IWebBrowser2Disp.ShowBrowserBar 768
SHDocVw.IWebBrowser2Disp.Silent 773
SHDocVw.IWebBrowser2Disp.StatusBar 774
SHDocVw.IWebBrowser2Disp.StatusText 774
SHDocVw.IWebBrowser2Disp.Stop 768
SHDocVw.IWebBrowser2Disp.TheaterMode 774
SHDocVw.IWebBrowser2Disp.ToolBar 774
SHDocVw.IWebBrowser2Disp.Top 775
SHDocVw.IWebBrowser2Disp.TopLevelContainer 775
SHDocVw.IWebBrowser2Disp.type_ 775
SHDocVw.IWebBrowser2Disp.Visible 775
SHDocVw.IWebBrowser2Disp.Width 775
SHDocVw.IWebBrowserApp 776
SHDocVw.IWebBrowserApp.ClientToWindow 778
SHDocVw.IWebBrowserApp.FullName 783
SHDocVw.IWebBrowserApp.FullScreen 783
SHDocVw.IWebBrowserApp.Get_FullName 778
SHDocVw.IWebBrowserApp.Get_FullScreen 779
SHDocVw.IWebBrowserApp.Get_HWnd 779
SHDocVw.IWebBrowserApp.Get_MenuBar 779
SHDocVw.IWebBrowserApp.Get_Name 779
SHDocVw.IWebBrowserApp.Get_Path 779
SHDocVw.IWebBrowserApp.Get_StatusBar 780
SHDocVw.IWebBrowserApp.Get_StatusText 780
SHDocVw.IWebBrowserApp.Get_ToolBar 780
SHDocVw.IWebBrowserApp.Get_Visible 780
SHDocVw.IWebBrowserApp.GetProperty 781
SHDocVw.IWebBrowserApp.HWnd 783
SHDocVw.IWebBrowserApp.MenuBar 783
SHDocVw.IWebBrowserApp.Name 784
SHDocVw.IWebBrowserApp.Path 784
SHDocVw.IWebBrowserApp.PutProperty 781
SHDocVw.IWebBrowserApp.Quit 781
SHDocVw.IWebBrowserApp.Set_FullScreen 781
SHDocVw.IWebBrowserApp.Set_MenuBar 781
SHDocVw.IWebBrowserApp.Set_StatusBar 782
SHDocVw.IWebBrowserApp.Set_StatusText 782
SHDocVw.IWebBrowserApp.Set_ToolBar 782
SHDocVw.IWebBrowserApp.Set_Visible 782
SHDocVw.IWebBrowserApp.StatusBar 784
SHDocVw.IWebBrowserApp.StatusText 784
SHDocVw.IWebBrowserApp.ToolBar 785
SHDocVw.IWebBrowserApp.Visible 785
SHDocVw.IWebBrowserAppDisp 785
SHDocVw.IWebBrowserAppDisp.Application 790
SHDocVw.IWebBrowserAppDisp.Busy 790
SHDocVw.IWebBrowserAppDisp.ClientToWindow 787
SHDocVw.IWebBrowserAppDisp.Container 790
SHDocVw.IWebBrowserAppDisp.Document 790
SHDocVw.IWebBrowserAppDisp.FullName 791
SHDocVw.IWebBrowserAppDisp.FullScreen 791
SHDocVw.IWebBrowserAppDisp.GetProperty 787
SHDocVw.IWebBrowserAppDisp.GoBack 787
SHDocVw.IWebBrowserAppDisp.GoForward 787
SHDocVw.IWebBrowserAppDisp.GoHome 787
SHDocVw.IWebBrowserAppDisp.GoSearch 788
SHDocVw.IWebBrowserAppDisp.Height 791
SHDocVw.IWebBrowserAppDisp.HWnd 791
SHDocVw.IWebBrowserAppDisp.Left 792
SHDocVw.IWebBrowserAppDisp.LocationName 792
SHDocVw.IWebBrowserAppDisp.LocationURL 792
SHDocVw.IWebBrowserAppDisp.MenuBar 792
SHDocVw.IWebBrowserAppDisp.Name 792
SHDocVw.IWebBrowserAppDisp.Navigate 788
SHDocVw.IWebBrowserAppDisp.Parent 793

SHDocVw.IWebBrowserAppDisp.Path 793
SHDocVw.IWebBrowserAppDisp.PutProperty 788
SHDocVw.IWebBrowserAppDisp.Quit 788
SHDocVw.IWebBrowserAppDisp.Refresh 789
SHDocVw.IWebBrowserAppDisp.Refresh2 789
SHDocVw.IWebBrowserAppDisp.StatusBar 793
SHDocVw.IWebBrowserAppDisp.StatusText 793
SHDocVw.IWebBrowserAppDisp.Stop 789
SHDocVw.IWebBrowserAppDisp.ToolBar 794
SHDocVw.IWebBrowserAppDisp.Top 794
SHDocVw.IWebBrowserAppDisp.TopLevelContainer 794
SHDocVw.IWebBrowserAppDisp.type_ 794
SHDocVw.IWebBrowserAppDisp.Visible 794
SHDocVw.IWebBrowserAppDisp.Width 795
SHDocVw.IWebBrowserDisp 795
SHDocVw.IWebBrowserDisp.Application 798
SHDocVw.IWebBrowserDisp.Busy 799
SHDocVw.IWebBrowserDisp.Container 799
SHDocVw.IWebBrowserDisp.Document 799
SHDocVw.IWebBrowserDisp.GoBack 796
SHDocVw.IWebBrowserDisp.GoForward 796
SHDocVw.IWebBrowserDisp.GoHome 797
SHDocVw.IWebBrowserDisp.GoSearch 797
SHDocVw.IWebBrowserDisp.Height 799
SHDocVw.IWebBrowserDisp.Left 799
SHDocVw.IWebBrowserDisp.LocationName 800
SHDocVw.IWebBrowserDisp.LocationURL 800
SHDocVw.IWebBrowserDisp.Navigate 797
SHDocVw.IWebBrowserDisp.Parent 800
SHDocVw.IWebBrowserDisp.Refresh 797
SHDocVw.IWebBrowserDisp.Refresh2 797
SHDocVw.IWebBrowserDisp.Stop 798
SHDocVw.IWebBrowserDisp.Top 800
SHDocVw.IWebBrowserDisp.TopLevelContainer 801
SHDocVw.IWebBrowserDisp.type_ 801
SHDocVw.IWebBrowserDisp.Width 801
SHDocVw.LIBID_SHDocVw 852
SHDocVw.LIBID_SHDocVw constant 852
SHDocVw.navAllowAutosearch 853
SHDocVw.navAllowAutosearch constant 853
SHDocVw.navBrowserBar 853
SHDocVw.navBrowserBar constant 853
SHDocVw.navNoHistory 853
SHDocVw.navNoHistory constant 853
SHDocVw.navNoReadFromCache 854
SHDocVw.navNoReadFromCache constant 854
SHDocVw.navNoWriteToCache 854
SHDocVw.navNoWriteToCache constant 854
SHDocVw.navOpenInNewWindow 854
SHDocVw.navOpenInNewWindow constant 854
SHDocVw.OLECMDEXECOPT 805
SHDocVw.OLECMDEXECOPT type 805
SHDocVw.OLECMDEXECOPT_DODEFAULT 855
SHDocVw.OLECMDEXECOPT_DODEFAULT constant 855
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER 855
SHDocVw.OLECMDEXECOPT_DONTPROMPTUSER constant 855
SHDocVw.OLECMDEXECOPT_PROMPTUSER 855
SHDocVw.OLECMDEXECOPT_PROMPTUSER constant 855
SHDocVw.OLECMDEXECOPT_SHOWHELP 856
SHDocVw.OLECMDEXECOPT_SHOWHELP constant 856
SHDocVw.OLECMDF 806
SHDocVw.OLECMDF type 806
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU 856
SHDocVw.OLECMDF_DEFHIDEONCTXTMENU constant 856
SHDocVw.OLECMDF_ENABLED 856
SHDocVw.OLECMDF_ENABLED constant 856
SHDocVw.OLECMDF_INVISIBLE 857
SHDocVw.OLECMDF_INVISIBLE constant 857
SHDocVw.OLECMDF_LATCHED 857
SHDocVw.OLECMDF_LATCHED constant 857
SHDocVw.OLECMDF_NINCHED 857
SHDocVw.OLECMDF_NINCHED constant 857
SHDocVw.OLECMDF_SUPPORTED 858
SHDocVw.OLECMDF_SUPPORTED constant 858
SHDocVw.OLECMDID 806
SHDocVw.OLECMDID type 806
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS 858
SHDocVw.OLECMDID_ALLOWUILESSSAVEAS constant 858

SHDocVw.OLECMDID_CLEARSELECTION 858
SHDocVw.OLECMDID_CLEARSELECTION constant 858
SHDocVw.OLECMDID_CLOSE 859
SHDocVw.OLECMDID_CLOSE constant 859
SHDocVw.OLECMDID_COPY 859
SHDocVw.OLECMDID_COPY constant 859
SHDocVw.OLECMDID_CUT 859
SHDocVw.OLECMDID_CUT constant 859
SHDocVw.OLECMDID_DELETE 860
SHDocVw.OLECMDID_DELETE constant 860
SHDocVw.OLECMDID_DONTDOWNLOADCSS 860
SHDocVw.OLECMDID_DONTDOWNLOADCSS constant 860
SHDocVw.OLECMDID_ENABLE_INTERACTION 860
SHDocVw.OLECMDID_ENABLE_INTERACTION constant 860
SHDocVw.OLECMDID_FIND 861
SHDocVw.OLECMDID_FIND constant 861
SHDocVw.OLECMDID_GETPRINTTEMPLATE 861
SHDocVw.OLECMDID_GETPRINTTEMPLATE constant 861
SHDocVw.OLECMDID_GETZOOMRANGE 861
SHDocVw.OLECMDID_GETZOOMRANGE constant 861
SHDocVw.OLECMDID_HIDETOOLBARS 862
SHDocVw.OLECMDID_HIDETOOLBARS constant 862
SHDocVw.OLECMDID_HTTPEQUIV 862
SHDocVw.OLECMDID_HTTPEQUIV constant 862
SHDocVw.OLECMDID_HTTPEQUIV_DONE 862
SHDocVw.OLECMDID_HTTPEQUIV_DONE constant 862
SHDocVw.OLECMDID_NEW 863
SHDocVw.OLECMDID_NEW constant 863
SHDocVw.OLECMDID_ONTOOLBARACTIVATED 863
SHDocVw.OLECMDID_ONTOOLBARACTIVATED constant 863
SHDocVw.OLECMDID_ONUNLOAD 863
SHDocVw.OLECMDID_ONUNLOAD constant 863
SHDocVw.OLECMDID_OPEN 864
SHDocVw.OLECMDID_OPEN constant 864
SHDocVw.OLECMDID_PAGESETUP 864
SHDocVw.OLECMDID_PAGESETUP constant 864
SHDocVw.OLECMDID_PASTE 864
SHDocVw.OLECMDID_PASTE constant 864
SHDocVw.OLECMDID_PASTESPECIAL 865
SHDocVw.OLECMDID_PASTESPECIAL constant 865
SHDocVw.OLECMDID_PREREFRESH 865
SHDocVw.OLECMDID_PREREFRESH constant 865
SHDocVw.OLECMDID_PRINT 865
SHDocVw.OLECMDID_PRINT constant 865
SHDocVw.OLECMDID_PRINT2 866
SHDocVw.OLECMDID_PRINT2 constant 866
SHDocVw.OLECMDID_PRINTPREVIEW 866
SHDocVw.OLECMDID_PRINTPREVIEW constant 866
SHDocVw.OLECMDID_PRINTPREVIEW2 866
SHDocVw.OLECMDID_PRINTPREVIEW2 constant 866
SHDocVw.OLECMDID_PROPERTIES 867
SHDocVw.OLECMDID_PROPERTIES constant 867
SHDocVw.OLECMDID_PROPERTYBAG2 867
SHDocVw.OLECMDID_PROPERTYBAG2 constant 867
SHDocVw.OLECMDID_REDO 867
SHDocVw.OLECMDID_REDO constant 867
SHDocVw.OLECMDID_REFRESH 868
SHDocVw.OLECMDID_REFRESH constant 868
SHDocVw.OLECMDID_SAVE 868
SHDocVw.OLECMDID_SAVE constant 868
SHDocVw.OLECMDID_SAVEAS 868
SHDocVw.OLECMDID_SAVEAS constant 868
SHDocVw.OLECMDID_SAVECOPYAS 869
SHDocVw.OLECMDID_SAVECOPYAS constant 869
SHDocVw.OLECMDID_SELECTALL 869
SHDocVw.OLECMDID_SELECTALL constant 869
SHDocVw.OLECMDID_SETDOWNLOADSTATE 869
SHDocVw.OLECMDID_SETDOWNLOADSTATE constant 869
SHDocVw.OLECMDID_SETPRINTTEMPLATE 870
SHDocVw.OLECMDID_SETPRINTTEMPLATE constant 870
SHDocVw.OLECMDID_SETPROGRESSMAX 870
SHDocVw.OLECMDID_SETPROGRESSMAX constant 870
SHDocVw.OLECMDID_SETPROGRESSPOS 870
SHDocVw.OLECMDID_SETPROGRESSPOS constant 870
SHDocVw.OLECMDID_SETPROGRESSTEXT 871
SHDocVw.OLECMDID_SETPROGRESSTEXT constant 871
SHDocVw.OLECMDID_SETTITLE 871
SHDocVw.OLECMDID_SETTITLE constant 871

SHDocVw.OLECMDID_SHOWFIND 871
SHDocVw.OLECMDID_SHOWFIND constant 871
SHDocVw.OLECMDID_SHOWMESSAGE 872
SHDocVw.OLECMDID_SHOWMESSAGE constant 872
SHDocVw.OLECMDID_SHOWPAGESETUP 872
SHDocVw.OLECMDID_SHOWPAGESETUP constant 872
SHDocVw.OLECMDID_SHOWPRINT 872
SHDocVw.OLECMDID_SHOWPRINT constant 872
SHDocVw.OLECMDID_SHOWSCRIPTERROR 873
SHDocVw.OLECMDID_SHOWSCRIPTERROR constant 873
SHDocVw.OLECMDID_SPELL 873
SHDocVw.OLECMDID_SPELL constant 873
SHDocVw.OLECMDID_STOP 873
SHDocVw.OLECMDID_STOP constant 873
SHDocVw.OLECMDID_STOPDOWNLOAD 874
SHDocVw.OLECMDID_STOPDOWNLOAD constant 874
SHDocVw.OLECMDID_UNDO 874
SHDocVw.OLECMDID_UNDO constant 874
SHDocVw.OLECMDID_UPDATECOMMANDS 874
SHDocVw.OLECMDID_UPDATECOMMANDS constant 874
SHDocVw.OLECMDID_UPDATEPAGESTATUS 875
SHDocVw.OLECMDID_UPDATEPAGESTATUS constant 875
SHDocVw.OLECMDID_ZOOM 875
SHDocVw.OLECMDID_ZOOM constant 875
SHDocVw.POleVariant1 807
SHDocVw.POleVariant1 type 807
SHDocVw.READYSTATE_COMPLETE 875
SHDocVw.READYSTATE_COMPLETE constant 875
SHDocVw.READYSTATE_INTERACTIVE 876
SHDocVw.READYSTATE_INTERACTIVE constant 876
SHDocVw.READYSTATE_LOADED 876
SHDocVw.READYSTATE_LOADED constant 876
SHDocVw.READYSTATE_LOADING 876
SHDocVw.READYSTATE_LOADING constant 876
SHDocVw.READYSTATE_UNINITIALIZED 877
SHDocVw.READYSTATE_UNINITIALIZED constant 877
SHDocVw.REFRESH_COMPLETELY 877
SHDocVw.REFRESH_COMPLETELY constant 877
SHDocVw.REFRESH_CONTINUE 877
SHDocVw.REFRESH_CONTINUE constant 877
SHDocVw.REFRESH_IFEXPIRED 878
SHDocVw.REFRESH_IFEXPIRED constant 878
SHDocVw.REFRESH_NORMAL 878
SHDocVw.REFRESH_NORMAL constant 878
SHDocVw.RefreshConstants 807
SHDocVw.RefreshConstants type 807
SHDocVw.SearchAssistantOC 808
SHDocVw.SearchAssistantOC type 808
SHDocVw.SecureLockIconConstants 808
SHDocVw.SecureLockIconConstants type 808
SHDocVw.secureLockIconMixed 878
SHDocVw.secureLockIconMixed constant 878
SHDocVw.secureLockIconSecure128Bit 879
SHDocVw.secureLockIconSecure128Bit constant 879
SHDocVw.secureLockIconSecure40Bit 879
SHDocVw.secureLockIconSecure40Bit constant 879
SHDocVw.secureLockIconSecure56Bit 879
SHDocVw.secureLockIconSecure56Bit constant 879
SHDocVw.secureLockIconSecureFortezza 880
SHDocVw.secureLockIconSecureFortezza constant 880
SHDocVw.secureLockIconSecureUnknownBits 880
SHDocVw.secureLockIconSecureUnknownBits constant 880
SHDocVw.secureLockIconUnsecure 880
SHDocVw.secureLockIconUnsecure constant 880
SHDocVw.SHDocVwMajorVersion 881
SHDocVw.SHDocVwMajorVersion constant 881
SHDocVw.SHDocVwMinorVersion 881
SHDocVw.SHDocVwMinorVersion constant 881
SHDocVw.ShellBrowserWindow 808
SHDocVw.ShellBrowserWindow type 808
SHDocVw.ShellNameSpace 809
SHDocVw.ShellNameSpace type 809
SHDocVw.ShellUIHelper 809
SHDocVw.ShellUIHelper type 809
SHDocVw.ShellWindowFindWindowOptions 809
SHDocVw.ShellWindowFindWindowOptions type 809
SHDocVw.ShellWindows 810
SHDocVw.ShellWindows type 810

SHDocVw.ShellWindowTypeConstants 810
SHDocVw.ShellWindowTypeConstants type 810
SHDocVw.SWC_3RDPARTY 881
SHDocVw.SWC_3RDPARTY constant 881
SHDocVw.SWC_BROWSER 882
SHDocVw.SWC_BROWSER constant 882
SHDocVw.SWC_CALLBACK 882
SHDocVw.SWC_CALLBACK constant 882
SHDocVw.SWC_EXPLORER 882
SHDocVw.SWC_EXPLORER constant 882
SHDocVw.SWFO_COOKIEPASSED 883
SHDocVw.SWFO_COOKIEPASSED constant 883
SHDocVw.SWFO_INCLUDEPENDING 883
SHDocVw.SWFO_INCLUDEPENDING constant 883
SHDocVw.SWFO_NEEDDISPATCH 883
SHDocVw.SWFO_NEEDDISPATCH constant 883
SHDocVw.tagREADYSTATE 810
SHDocVw.tagREADYSTATE type 810
SHDocVw.TInternetExplorer 421
SHDocVw.TInternetExplorer.AddressBar 449
SHDocVw.TInternetExplorer.Application 449
SHDocVw.TInternetExplorer.Busy 449
SHDocVw.TInternetExplorer.ClientToWindow 428
SHDocVw.TInternetExplorer.Connect 428
SHDocVw.TInternetExplorer.ConnectTo 428
SHDocVw.TInternetExplorer.Container 450
SHDocVw.TInternetExplorer.Create 425
SHDocVw.TInternetExplorer.DefaultInterface 450
SHDocVw.TInternetExplorer.Destroy 425
SHDocVw.TInternetExplorer.Disconnect 429
SHDocVw.TInternetExplorer.Document 450
SHDocVw.TInternetExplorer.ExecWB 429, 430
SHDocVw.TInternetExplorer.FullName 450
SHDocVw.TInternetExplorer.FullScreen 450
SHDocVw.TInternetExplorer.Get_AddressBar 430
SHDocVw.TInternetExplorer.Get_Application 430
SHDocVw.TInternetExplorer.Get_Busy 430
SHDocVw.TInternetExplorer.Get_Container 431
SHDocVw.TInternetExplorer.Get_Document 431
SHDocVw.TInternetExplorer.Get_FullName 431
SHDocVw.TInternetExplorer.Get_FullScreen 431
SHDocVw.TInternetExplorer.Get_Height 431
SHDocVw.TInternetExplorer.Get_HWnd 432
SHDocVw.TInternetExplorer.Get_Left 432
SHDocVw.TInternetExplorer.Get_LocationName 432
SHDocVw.TInternetExplorer.Get_LocationURL 432
SHDocVw.TInternetExplorer.Get_MenuBar 433
SHDocVw.TInternetExplorer.Get_Name 433
SHDocVw.TInternetExplorer.Get_Offline 433
SHDocVw.TInternetExplorer.Get_Parent 433
SHDocVw.TInternetExplorer.Get_Path 433
SHDocVw.TInternetExplorer.Get_ReadyState 434
SHDocVw.TInternetExplorer.Get_RegisterAsBrowser 434
SHDocVw.TInternetExplorer.Get_RegisterAsDropTarget 434
SHDocVw.TInternetExplorer.Get_Resizable 434
SHDocVw.TInternetExplorer.Get_Silent 435
SHDocVw.TInternetExplorer.Get_StatusBar 435
SHDocVw.TInternetExplorer.Get_StatusText 435
SHDocVw.TInternetExplorer.Get_TheaterMode 435
SHDocVw.TInternetExplorer.Get_ToolBar 435
SHDocVw.TInternetExplorer.Get_Top 436
SHDocVw.TInternetExplorer.Get_TopLevelContainer 436
SHDocVw.TInternetExplorer.Get_type_ 436
SHDocVw.TInternetExplorer.Get_Visible 436
SHDocVw.TInternetExplorer.Get_Width 437
SHDocVw.TInternetExplorer.GetProperty 437
SHDocVw.TInternetExplorer.GoBack 437
SHDocVw.TInternetExplorer.GoForward 437
SHDocVw.TInternetExplorer.GoHome 437
SHDocVw.TInternetExplorer.GoSearch 438
SHDocVw.TInternetExplorer.Height 451
SHDocVw.TInternetExplorer.HWnd 451
SHDocVw.TInternetExplorer.InitServerData 438
SHDocVw.TInternetExplorer.InvokeEvent 438
SHDocVw.TInternetExplorer.Left 451
SHDocVw.TInternetExplorer.LocationName 451
SHDocVw.TInternetExplorer.LocationURL 452
SHDocVw.TInternetExplorer.MenuBar 452

SHDocVw.TInternetExplorer.Name 452
SHDocVw.TInternetExplorer.Navigate 439, 440
SHDocVw.TInternetExplorer.Navigate2 441, 442
SHDocVw.TInternetExplorer.Offline 454
SHDocVw.TInternetExplorer.OnBeforeNavigate2 459
SHDocVw.TInternetExplorer.OnClientToHostWindow 459
SHDocVw.TInternetExplorer.OnCommandStateChange 459
SHDocVw.TInternetExplorer.OnDocumentComplete 459
SHDocVw.TInternetExplorer.OnDownloadBegin 459
SHDocVw.TInternetExplorer.OnDownloadComplete 460
SHDocVw.TInternetExplorer.OnFileDownload 460
SHDocVw.TInternetExplorer.OnFullScreen 460
SHDocVw.TInternetExplorer.OnMenuBar 460
SHDocVw.TInternetExplorer.OnNavigateComplete2 461
SHDocVw.TInternetExplorer.OnNavigateError 461
SHDocVw.TInternetExplorer.OnNewWindow2 461
SHDocVw.TInternetExplorer.OnPrintTemplateInstantiation 461
SHDocVw.TInternetExplorer.OnPrintTemplateTeardown 461
SHDocVw.TInternetExplorer.OnPrivacyImpactedStateChange 462
SHDocVw.TInternetExplorer.OnProgressChange 462
SHDocVw.TInternetExplorer.OnPropertyChange 462
SHDocVw.TInternetExplorer.OnQuit 462
SHDocVw.TInternetExplorer.OnSetSecureLockIcon 463
SHDocVw.TInternetExplorer.OnStatusBar 463
SHDocVw.TInternetExplorer.OnStatusTextChange 463
SHDocVw.TInternetExplorer.OnTheaterMode 463
SHDocVw.TInternetExplorer.OnTitleChange 463
SHDocVw.TInternetExplorer.OnToolBar 464
SHDocVw.TInternetExplorer.OnUpdatePageStatus 464
SHDocVw.TInternetExplorer.OnVisible 464
SHDocVw.TInternetExplorer.OnWindowClosing 464
SHDocVw.TInternetExplorer.OnWindowSetHeight 465
SHDocVw.TInternetExplorer.OnWindowSetLeft 465
SHDocVw.TInternetExplorer.OnWindowSetResizable 465
SHDocVw.TInternetExplorer.OnWindowSetTop 465
SHDocVw.TInternetExplorer.OnWindowSetWidth 465
SHDocVw.TInternetExplorer.Parent 454
SHDocVw.TInternetExplorer.Path 455
SHDocVw.TInternetExplorer.PutProperty 442
SHDocVw.TInternetExplorer.QueryStatusWB 442
SHDocVw.TInternetExplorer.Quit 442
SHDocVw.TInternetExplorer.ReadyState 455
SHDocVw.TInternetExplorer.Refresh 442
SHDocVw.TInternetExplorer.Refresh2 443
SHDocVw.TInternetExplorer.RegisterAsBrowser 455
SHDocVw.TInternetExplorer.RegisterAsDropTarget 455
SHDocVw.TInternetExplorer.Resizable 455
SHDocVw.TInternetExplorer.Set_AddressBar 443
SHDocVw.TInternetExplorer.Set_FullScreen 444
SHDocVw.TInternetExplorer.Set_Height 444
SHDocVw.TInternetExplorer.Set_Left 444
SHDocVw.TInternetExplorer.Set_MenuBar 444
SHDocVw.TInternetExplorer.Set_Offline 444
SHDocVw.TInternetExplorer.Set_RegisterAsBrowser 445
SHDocVw.TInternetExplorer.Set_RegisterAsDropTarget 445
SHDocVw.TInternetExplorer.Set_Resizable 445
SHDocVw.TInternetExplorer.Set_Silent 445
SHDocVw.TInternetExplorer.Set_StatusBar 446
SHDocVw.TInternetExplorer.Set_StatusText 446
SHDocVw.TInternetExplorer.Set_TheaterMode 446
SHDocVw.TInternetExplorer.Set_ToolBar 446
SHDocVw.TInternetExplorer.Set_Top 446
SHDocVw.TInternetExplorer.Set_Visible 447
SHDocVw.TInternetExplorer.Set_Width 447
SHDocVw.TInternetExplorer.ShowBrowserBar 447, 448
SHDocVw.TInternetExplorer.Silent 456
SHDocVw.TInternetExplorer.StatusBar 456
SHDocVw.TInternetExplorer.StatusText 456
SHDocVw.TInternetExplorer.Stop 448
SHDocVw.TInternetExplorer.TheaterMode 456
SHDocVw.TInternetExplorer.ToolBar 457
SHDocVw.TInternetExplorer.Top 457
SHDocVw.TInternetExplorer.TopLevelContainer 457
SHDocVw.TInternetExplorer.type_ 457
SHDocVw.TInternetExplorer.Visible 457
SHDocVw.TInternetExplorer.Width 458
SHDocVw.TInternetExplorerBeforeNavigate2 811
SHDocVw.TInternetExplorerBeforeNavigate2 type 811

SHDocVw.TInternetExplorerClientToHostWindow 811
SHDocVw.TInternetExplorerClientToHostWindow type 811
SHDocVw.TInternetExplorerCommandStateChange 811
SHDocVw.TInternetExplorerCommandStateChange type 811
SHDocVw.TInternetExplorerDocumentComplete 812
SHDocVw.TInternetExplorerDocumentComplete type 812
SHDocVw.TInternetExplorerFileDownload 812
SHDocVw.TInternetExplorerFileDownload type 812
SHDocVw.TInternetExplorerNavigateComplete2 812
SHDocVw.TInternetExplorerNavigateComplete2 type 812
SHDocVw.TInternetExplorerNavigateError 813
SHDocVw.TInternetExplorerNavigateError type 813
SHDocVw.TInternetExplorerNewWindow2 813
SHDocVw.TInternetExplorerNewWindow2 type 813
SHDocVw.TInternetExplorerOnFullScreen 813
SHDocVw.TInternetExplorerOnFullScreen type 813
SHDocVw.TInternetExplorerOnMenuBar 814
SHDocVw.TInternetExplorerOnMenuBar type 814
SHDocVw.TInternetExplorerOnStatusBar 814
SHDocVw.TInternetExplorerOnStatusBar type 814
SHDocVw.TInternetExplorerOnTheaterMode 814
SHDocVw.TInternetExplorerOnTheaterMode type 814
SHDocVw.TInternetExplorerOnToolBar 815
SHDocVw.TInternetExplorerOnToolBar type 815
SHDocVw.TInternetExplorerOnVisible 815
SHDocVw.TInternetExplorerOnVisible type 815
SHDocVw.TInternetExplorerPrintTemplateInstantiation 815
SHDocVw.TInternetExplorerPrintTemplateInstantiation type 815
SHDocVw.TInternetExplorerPrintTemplateTeardown 816
SHDocVw.TInternetExplorerPrintTemplateTeardown type 816
SHDocVw.TInternetExplorerPrivacyImpactedStateChange 816
SHDocVw.TInternetExplorerPrivacyImpactedStateChange type 816
SHDocVw.TInternetExplorerProgressChange 816
SHDocVw.TInternetExplorerProgressChange type 816
SHDocVw.TInternetExplorerPropertyChange 817
SHDocVw.TInternetExplorerPropertyChange type 817
SHDocVw.TInternetExplorerSetSecureLockIcon 817
SHDocVw.TInternetExplorerSetSecureLockIcon type 817
SHDocVw.TInternetExplorerStatusTextChange 817
SHDocVw.TInternetExplorerStatusTextChange type 817
SHDocVw.TInternetExplorerTitleChange 818
SHDocVw.TInternetExplorerTitleChange type 818
SHDocVw.TInternetExplorerUpdatePageStatus 818
SHDocVw.TInternetExplorerUpdatePageStatus type 818
SHDocVw.TInternetExplorerWindowClosing 818
SHDocVw.TInternetExplorerWindowClosing type 818
SHDocVw.TInternetExplorerWindowSetHeight 819
SHDocVw.TInternetExplorerWindowSetHeight type 819
SHDocVw.TInternetExplorerWindowSetLeft 819
SHDocVw.TInternetExplorerWindowSetLeft type 819
SHDocVw.TInternetExplorerWindowSetResizable 819
SHDocVw.TInternetExplorerWindowSetResizable type 819
SHDocVw.TInternetExplorerWindowSetTop 820
SHDocVw.TInternetExplorerWindowSetTop type 820
SHDocVw.TInternetExplorerWindowSetWidth 820
SHDocVw.TInternetExplorerWindowSetWidth type 820
SHDocVw.TShellNameSpace 466
SHDocVw.TShellNameSpace.Columns 480
SHDocVw.TShellNameSpace.Connect 470
SHDocVw.TShellNameSpace.ConnectTo 470
SHDocVw.TShellNameSpace.CountViewTypes 480
SHDocVw.TShellNameSpace.Create 467
SHDocVw.TShellNameSpace.CreateSubscriptionForSelection 471
SHDocVw.TShellNameSpace.DefaultInterface 480
SHDocVw.TShellNameSpace.DeleteSubscriptionForSelection 471
SHDocVw.TShellNameSpace.Depth 480
SHDocVw.TShellNameSpace.Destroy 468
SHDocVw.TShellNameSpace.Disconnect 471
SHDocVw.TShellNameSpace.EnumOptions 481
SHDocVw.TShellNameSpace.Expand 472
SHDocVw.TShellNameSpace.Export 472
SHDocVw.TShellNameSpace.Flags 481
SHDocVw.TShellNameSpace.Get_Columns 472
SHDocVw.TShellNameSpace.Get_CountViewTypes 472
SHDocVw.TShellNameSpace.Get_Depth 472
SHDocVw.TShellNameSpace.Get_EnumOptions 473
SHDocVw.TShellNameSpace.Get_Flags 473

SHDocVw.TShellNameSpace.Get_Mode 473
SHDocVw.TShellNameSpace.Get_Root 473
SHDocVw.TShellNameSpace.Get_SelectedItem 474
SHDocVw.TShellNameSpace.Get_SubscriptionsEnabled 474
SHDocVw.TShellNameSpace.Get_TVFlags 474
SHDocVw.TShellNameSpace.Import 474
SHDocVw.TShellNameSpace.InitServerData 474
SHDocVw.TShellNameSpace.InvokeContextMenuCommand 475
SHDocVw.TShellNameSpace.InvokeEvent 475
SHDocVw.TShellNameSpace.Mode 481
SHDocVw.TShellNameSpace.MoveSelectionDown 475
SHDocVw.TShellNameSpace.MoveSelectionTo 476
SHDocVw.TShellNameSpace.MoveSelectionUp 476
SHDocVw.TShellNameSpace.NewFolder 476
SHDocVw.TShellNameSpace.OnDoubleClick 482
SHDocVw.TShellNameSpace.OnFavoritesSelectionChange 483
SHDocVw.TShellNameSpace.OnInitialized 483
SHDocVw.TShellNameSpace.OnSelectionChange 483
SHDocVw.TShellNameSpace.ResetSort 476
SHDocVw.TShellNameSpace.Root 481
SHDocVw.TShellNameSpace.SelectedItem 482
SHDocVw.TShellNameSpace.SelectedItems 477
SHDocVw.TShellNameSpace.Set_Columns 477
SHDocVw.TShellNameSpace.Set_Depth 477
SHDocVw.TShellNameSpace.Set_EnumOptions 477
SHDocVw.TShellNameSpace.Set_Flags 477
SHDocVw.TShellNameSpace.Set_Mode 478
SHDocVw.TShellNameSpace.Set_Root 478
SHDocVw.TShellNameSpace.Set_SelectedItem 478
SHDocVw.TShellNameSpace.Set_TVFlags 478
SHDocVw.TShellNameSpace.SetRoot 479
SHDocVw.TShellNameSpace.SetViewType 479
SHDocVw.TShellNameSpace.SubscriptionsEnabled 482
SHDocVw.TShellNameSpace.Synchronize 479
SHDocVw.TShellNameSpace.TVFlags 482
SHDocVw.TShellNameSpace.UnselectAll 479
SHDocVw.TShellNameSpaceFavoritesSelectionChange 820
SHDocVw.TShellNameSpaceFavoritesSelectionChange type 820
SHDocVw.TShellUIHelper 483
SHDocVw.TShellUIHelper.AddChannel 486
SHDocVw.TShellUIHelper.AddDesktopComponent 487, 488
SHDocVw.TShellUIHelper.AddFavorite 488, 489
SHDocVw.TShellUIHelper.AutoCompleteAttach 489
SHDocVw.TShellUIHelper.AutoCompleteSaveForm 490
SHDocVw.TShellUIHelper.AutoScan 490, 491
SHDocVw.TShellUIHelper.Connect 491
SHDocVw.TShellUIHelper.ConnectTo 491
SHDocVw.TShellUIHelper.Create 484
SHDocVw.TShellUIHelper.DefaultInterface 494
SHDocVw.TShellUIHelper.Destroy 485
SHDocVw.TShellUIHelper.Disconnect 492
SHDocVw.TShellUIHelper.ImportExportFavorites 492
SHDocVw.TShellUIHelper.InitServerData 492
SHDocVw.TShellUIHelper.IsSubscribed 493
SHDocVw.TShellUIHelper.NavigateAndFind 493
SHDocVw.TShellUIHelper.RefreshOfflineDesktop 493
SHDocVw.TShellUIHelper.ResetFirstBootMode 493
SHDocVw.TShellUIHelper.ResetSafeMode 494
SHDocVw.TShellUIHelper.ShowBrowserUI 494
SHDocVw.TShellWindows 494
SHDocVw.TShellWindows._NewEnum 498
SHDocVw.TShellWindows.Connect 498
SHDocVw.TShellWindows.ConnectTo 498
SHDocVw.TShellWindows.Count 503
SHDocVw.TShellWindows.Create 495
SHDocVw.TShellWindows.DefaultInterface 503
SHDocVw.TShellWindows.Destroy 496
SHDocVw.TShellWindows.Disconnect 499
SHDocVw.TShellWindows.FindWindowSW 499
SHDocVw.TShellWindows.Get_Count 499
SHDocVw.TShellWindows.InitServerData 499
SHDocVw.TShellWindows.InvokeEvent 500
SHDocVw.TShellWindows.Item 500, 501
SHDocVw.TShellWindows.OnActivated 501
SHDocVw.TShellWindows.OnCreated 501
SHDocVw.TShellWindows.OnNavigate 501
SHDocVw.TShellWindows.OnWindowRegistered 503

SHDocVw.TShellWindows.OnWindowRevoked 504
SHDocVw.TShellWindows.ProcessAttachDetach 502
SHDocVw.TShellWindows.Register 502
SHDocVw.TShellWindows.RegisterPending 502
SHDocVw.TShellWindows.Revoke 502
SHDocVw.TShellWindowsWindowRegistered 821
SHDocVw.TShellWindowsWindowRegistered type 821
SHDocVw.TShellWindowsWindowRevoked 821
SHDocVw.TShellWindowsWindowRevoked type 821
SHDocVw.TWebBrowser 504
SHDocVw.TWebBrowser.AddressBar 520
SHDocVw.TWebBrowser.Align 520
SHDocVw.TWebBrowser.Anchors 523
SHDocVw.TWebBrowser.Application 524
SHDocVw.TWebBrowser.Busy 524
SHDocVw.TWebBrowser.ClientToWindow 507
SHDocVw.TWebBrowser.Container 525
SHDocVw.TWebBrowser.ControlInterface 525
SHDocVw.TWebBrowser.CreateControl 508
SHDocVw.TWebBrowser.DefaultInterface 526
SHDocVw.TWebBrowser.Document 526
SHDocVw.TWebBrowser.DragCursor 526
SHDocVw.TWebBrowser.DragMode 527
SHDocVw.TWebBrowser.ExecWB 508
SHDocVw.TWebBrowser.FullName 528
SHDocVw.TWebBrowser.FullScreen 529
SHDocVw.TWebBrowser.Get_Application 509
SHDocVw.TWebBrowser.Get_Container 509
SHDocVw.TWebBrowser.Get_Document 509
SHDocVw.TWebBrowser.Get_Parent 509
SHDocVw.TWebBrowser.GetProperty 510
SHDocVw.TWebBrowser.GoBack 510
SHDocVw.TWebBrowser.GoForward 510
SHDocVw.TWebBrowser.GoHome 511
SHDocVw.TWebBrowser.GoSearch 511
SHDocVw.TWebBrowser.HWnd 529
SHDocVw.TWebBrowser.InitControlData 512
SHDocVw.TWebBrowser.LocationName 529
SHDocVw.TWebBrowser.LocationURL 530
SHDocVw.TWebBrowser.MenuBar 530
SHDocVw.TWebBrowser.Name 531
SHDocVw.TWebBrowser.Navigate 512
SHDocVw.TWebBrowser.Navigate2 514
SHDocVw.TWebBrowser.Offline 531
SHDocVw.TWebBrowser.OnBeforeNavigate2 545
SHDocVw.TWebBrowser.OnClientToHostWindow 546
SHDocVw.TWebBrowser.OnCommandStateChange 546
SHDocVw.TWebBrowser.OnDocumentComplete 546
SHDocVw.TWebBrowser.OnDownloadBegin 547
SHDocVw.TWebBrowser.OnDownloadComplete 547
SHDocVw.TWebBrowser.OnDragDrop 548
SHDocVw.TWebBrowser.OnDragOver 551
SHDocVw.TWebBrowser.OnEndDrag 554
SHDocVw.TWebBrowser.OnEnter 555
SHDocVw.TWebBrowser.OnExit 558
SHDocVw.TWebBrowser.OnFileDownload 562
SHDocVw.TWebBrowser.OnFullScreen 562
SHDocVw.TWebBrowser.OnMenuBar 562
SHDocVw.TWebBrowser.OnNavigateComplete2 562
SHDocVw.TWebBrowser.OnNavigateError 563
SHDocVw.TWebBrowser.OnNewWindow2 563
SHDocVw.TWebBrowser.OnPrintTemplateInstantiation 563
SHDocVw.TWebBrowser.OnPrintTemplateTeardown 564
SHDocVw.TWebBrowser.OnPrivacyImpactedStateChange 564
SHDocVw.TWebBrowser.OnProgressChange 564
SHDocVw.TWebBrowser.OnPropertyChange 565
SHDocVw.TWebBrowser.OnQuit 565
SHDocVw.TWebBrowser.OnSetSecureLockIcon 565
SHDocVw.TWebBrowser.OnStartDrag 565
SHDocVw.TWebBrowser.OnStatusBar 566
SHDocVw.TWebBrowser.OnStatusTextChange 566
SHDocVw.TWebBrowser.OnTheaterMode 567
SHDocVw.TWebBrowser.OnTitleChange 567
SHDocVw.TWebBrowser.OnToolBar 567
SHDocVw.TWebBrowser.OnUpdatePageStatus 567
SHDocVw.TWebBrowser.OnVisible 568
SHDocVw.TWebBrowser.OnWindowClosing 568
SHDocVw.TWebBrowser.OnWindowSetHeight 568

SHDocVw.TWebBrowser.OnWindowSetLeft 568
SHDocVw.TWebBrowser.OnWindowSetResizable 569
SHDocVw.TWebBrowser.OnWindowSetTop 569
SHDocVw.TWebBrowser.OnWindowSetWidth 569
SHDocVw.TWebBrowser.Parent 531
SHDocVw.TWebBrowser.ParentShowHint 532
SHDocVw.TWebBrowser.Path 532
SHDocVw.TWebBrowser.PopupMenu 533
SHDocVw.TWebBrowser.PutProperty 515
SHDocVw.TWebBrowser.QueryStatusWB 516
SHDocVw.TWebBrowser.Quit 516
SHDocVw.TWebBrowser.ReadyState 538
SHDocVw.TWebBrowser.Refresh 516
SHDocVw.TWebBrowser.Refresh2 517
SHDocVw.TWebBrowser.RegisterAsBrowser 538
SHDocVw.TWebBrowser.RegisterAsDropTarget 539
SHDocVw.TWebBrowser.Resizable 539
SHDocVw.TWebBrowser.ShowBrowserBar 518
SHDocVw.TWebBrowser.ShowHint 539
SHDocVw.TWebBrowser.Silent 541
SHDocVw.TWebBrowser.StatusBar 541
SHDocVw.TWebBrowser.StatusText 541
SHDocVw.TWebBrowser.Stop 518
SHDocVw.TWebBrowser.TabOrder 542
SHDocVw.TWebBrowser.TabStop 542
SHDocVw.TWebBrowser.TheaterMode 543
SHDocVw.TWebBrowser.ToolBar 543
SHDocVw.TWebBrowser.TopLevelContainer 543
SHDocVw.TWebBrowser.type_ 544
SHDocVw.TWebBrowser.Visible 544
SHDocVw.TWebBrowser_V1 569
SHDocVw.TWebBrowser_V1.Align 577
SHDocVw.TWebBrowser_V1.Anchors 581
SHDocVw.TWebBrowser_V1.Application 581
SHDocVw.TWebBrowser_V1.Busy 581
SHDocVw.TWebBrowser_V1.Container 582
SHDocVw.TWebBrowser_V1.ControlInterface 582
SHDocVw.TWebBrowser_V1.CreateControl 572
SHDocVw.TWebBrowser_V1.DefaultInterface 582
SHDocVw.TWebBrowser_V1.Document 582
SHDocVw.TWebBrowser_V1.DragCursor 582
SHDocVw.TWebBrowser_V1.DragMode 583
SHDocVw.TWebBrowser_V1.Get_Application 572
SHDocVw.TWebBrowser_V1.Get_Container 572
SHDocVw.TWebBrowser_V1.Get_Document 572
SHDocVw.TWebBrowser_V1.Get_Parent 572
SHDocVw.TWebBrowser_V1.GoBack 573
SHDocVw.TWebBrowser_V1.GoForward 573
SHDocVw.TWebBrowser_V1.GoHome 573
SHDocVw.TWebBrowser_V1.GoSearch 573
SHDocVw.TWebBrowser_V1.InitControlData 574
SHDocVw.TWebBrowser_V1.LocationName 584
SHDocVw.TWebBrowser_V1.LocationURL 585
SHDocVw.TWebBrowser_V1.Navigate 574, 575
SHDocVw.TWebBrowser_V1.OnBeforeNavigate 597
SHDocVw.TWebBrowser_V1.OnCommandStateChange 597
SHDocVw.TWebBrowser_V1.OnDownloadBegin 598
SHDocVw.TWebBrowser_V1.OnDownloadComplete 598
SHDocVw.TWebBrowser_V1.OnDragDrop 598
SHDocVw.TWebBrowser_V1.OnDragOver 601
SHDocVw.TWebBrowser_V1.OnEndDrag 604
SHDocVw.TWebBrowser_V1.OnEnter 605
SHDocVw.TWebBrowser_V1.OnExit 608
SHDocVw.TWebBrowser_V1.OnFrameBeforeNavigate 612
SHDocVw.TWebBrowser_V1.OnFrameNavigateComplete 612
SHDocVw.TWebBrowser_V1.OnFrameNewWindow 612
SHDocVw.TWebBrowser_V1.OnNavigateComplete 612
SHDocVw.TWebBrowser_V1.OnNewWindow 613
SHDocVw.TWebBrowser_V1.OnProgressChange 613
SHDocVw.TWebBrowser_V1.OnPropertyChange 613
SHDocVw.TWebBrowser_V1.OnQuit 613
SHDocVw.TWebBrowser_V1.OnStartDrag 613
SHDocVw.TWebBrowser_V1.OnStatusTextChange 614
SHDocVw.TWebBrowser_V1.OnTitleChange 614
SHDocVw.TWebBrowser_V1.OnWindowActivate 615
SHDocVw.TWebBrowser_V1.OnWindowMove 615
SHDocVw.TWebBrowser_V1.OnWindowResize 615
SHDocVw.TWebBrowser_V1.Parent 585

SHDocVw.TWebBrowser_V1.ParentShowHint 586
SHDocVw.TWebBrowser_V1.PopupMenu 587
SHDocVw.TWebBrowser_V1.Refresh 575
SHDocVw.TWebBrowser_V1.Refresh2 576
SHDocVw.TWebBrowser_V1.ShowHint 592
SHDocVw.TWebBrowser_V1.Stop 576
SHDocVw.TWebBrowser_V1.TabOrder 593
SHDocVw.TWebBrowser_V1.TabStop 594
SHDocVw.TWebBrowser_V1.TopLevelContainer 595
SHDocVw.TWebBrowser_V1.type_ 595
SHDocVw.TWebBrowser_V1.Visible 595
SHDocVw.TWebBrowser_V1BeforeNavigate 821
SHDocVw.TWebBrowser_V1BeforeNavigate type 821
SHDocVw.TWebBrowser_V1CommandStateChange 822
SHDocVw.TWebBrowser_V1CommandStateChange type 822
SHDocVw.TWebBrowser_V1FrameBeforeNavigate 822
SHDocVw.TWebBrowser_V1FrameBeforeNavigate type 822
SHDocVw.TWebBrowser_V1FrameNavigateComplete 823
SHDocVw.TWebBrowser_V1FrameNavigateComplete type 823
SHDocVw.TWebBrowser_V1FrameNewWindow 823
SHDocVw.TWebBrowser_V1FrameNewWindow type 823
SHDocVw.TWebBrowser_V1NavigateComplete 823
SHDocVw.TWebBrowser_V1NavigateComplete type 823
SHDocVw.TWebBrowser_V1NewWindow 824
SHDocVw.TWebBrowser_V1NewWindow type 824
SHDocVw.TWebBrowser_V1ProgressChange 824
SHDocVw.TWebBrowser_V1ProgressChange type 824
SHDocVw.TWebBrowser_V1PropertyChange 824
SHDocVw.TWebBrowser_V1PropertyChange type 824
SHDocVw.TWebBrowser_V1Quit 825
SHDocVw.TWebBrowser_V1Quit type 825
SHDocVw.TWebBrowser_V1StatusTextChange 825
SHDocVw.TWebBrowser_V1StatusTextChange type 825
SHDocVw.TWebBrowser_V1TitleChange 825
SHDocVw.TWebBrowser_V1TitleChange type 825
SHDocVw.TWebBrowserBeforeNavigate2 826
SHDocVw.TWebBrowserBeforeNavigate2 type 826
SHDocVw.TWebBrowserClientToHostWindow 826
SHDocVw.TWebBrowserClientToHostWindow type 826
SHDocVw.TWebBrowserCommandStateChange 827
SHDocVw.TWebBrowserCommandStateChange type 827
SHDocVw.TWebBrowserDocumentComplete 827
SHDocVw.TWebBrowserDocumentComplete type 827
SHDocVw.TWebBrowserFileDownload 828
SHDocVw.TWebBrowserFileDownload type 828
SHDocVw.TWebBrowserNavigateComplete2 828
SHDocVw.TWebBrowserNavigateComplete2 type 828
SHDocVw.TWebBrowserNavigateError 829
SHDocVw.TWebBrowserNavigateError type 829
SHDocVw.TWebBrowserNewWindow2 829
SHDocVw.TWebBrowserNewWindow2 type 829
SHDocVw.TWebBrowserOnFullScreen 830
SHDocVw.TWebBrowserOnFullScreen type 830
SHDocVw.TWebBrowserOnMenuBar 830
SHDocVw.TWebBrowserOnMenuBar type 830
SHDocVw.TWebBrowserOnStatusBar 830
SHDocVw.TWebBrowserOnStatusBar type 830
SHDocVw.TWebBrowserOnTheaterMode 831
SHDocVw.TWebBrowserOnTheaterMode type 831
SHDocVw.TWebBrowserOnToolBar 831
SHDocVw.TWebBrowserOnToolBar type 831
SHDocVw.TWebBrowserOnVisible 831
SHDocVw.TWebBrowserOnVisible type 831
SHDocVw.TWebBrowserPrintTemplateInstantiation 832
SHDocVw.TWebBrowserPrintTemplateInstantiation type 832
SHDocVw.TWebBrowserPrintTemplateTeardown 832
SHDocVw.TWebBrowserPrintTemplateTeardown type 832
SHDocVw.TWebBrowserPrivacyImpactedStateChange 832
SHDocVw.TWebBrowserPrivacyImpactedStateChange type 832
SHDocVw.TWebBrowserProgressChange 833
SHDocVw.TWebBrowserProgressChange type 833
SHDocVw.TWebBrowserPropertyChange 833
SHDocVw.TWebBrowserPropertyChange type 833
SHDocVw.TWebBrowserSetSecureLockIcon 834
SHDocVw.TWebBrowserSetSecureLockIcon type 834
SHDocVw.TWebBrowserStatusTextChange 834
SHDocVw.TWebBrowserStatusTextChange type 834
SHDocVw.TWebBrowserTitleChange 834

SHDocVw.TWebBrowserTitleChange type 834
SHDocVw.TWebBrowserUpdatePageStatus 835
SHDocVw.TWebBrowserUpdatePageStatus type 835
SHDocVw.TWebBrowserWindowClosing 835
SHDocVw.TWebBrowserWindowClosing type 835
SHDocVw.TWebBrowserWindowSetHeight 835
SHDocVw.TWebBrowserWindowSetHeight type 835
SHDocVw.TWebBrowserWindowSetLeft 836
SHDocVw.TWebBrowserWindowSetLeft type 836
SHDocVw.TWebBrowserWindowSetResizable 836
SHDocVw.TWebBrowserWindowSetResizable type 836
SHDocVw.TWebBrowserWindowSetTop 836
SHDocVw.TWebBrowserWindowSetTop type 836
SHDocVw.TWebBrowserWindowSetWidth 837
SHDocVw.TWebBrowserWindowSetWidth type 837
SHDocVw.WebBrowser 837
SHDocVw.WebBrowser type 837
SHDocVw.WebBrowser_V1 837
SHDocVw.WebBrowser_V1 type 837
ShellAnimations 885
ShellAnimations namespace 885
  Classes 886
ShellAnimations.TShellResources 886
ShellFeaturesEnabled
  ISearchAssistantOC interface 656
  ISearchAssistantOC2Disp interface 665
  ISearchAssistantOC3Disp interface 676
  ISearchAssistantOCDisp interface 684
short
  Variant class 3462
short*
  Variant class 3463
ShortCut
  TBrowseForFolder class 1771
  TColorSelect class 1780
  TFileExit class 1815
  TFileOpen class 1825
  TFilePageSetup class 1842
  TFilePrintSetup class 1851
  TFileSaveAs class 1860
  TFontEdit class 1870
  TPrintDlg class 1890
  TSearchFind class 1905
  TSearchFindNext class 1919
  TSearchReplace class 1928
ShortName
  TScriptFile class 6687
ShowAccelChar
  TCustomLabel class 2396
  TCustomStaticText class 2462
  TLabel class 2632
  TStaticText class 2973
ShowBrowserBar
  IWebBrowser2 interface 759
  IWebBrowser2Disp interface 768
  TInternetExplorer class 447, 448
  TWebBrowser class 518
ShowBrowserUI
  IShellUIHelper interface 720
  IShellUIHelperDisp interface 724
  TShellUIHelper class 494
ShowControl
  TTabbedNotebook class 4412
ShowFindPrinter
  ISearchAssistantOC2 interface 658
  ISearchAssistantOC2Disp interface 666
  ISearchAssistantOC3Disp interface 677
ShowFooter
  TScreenTipItem class 333
ShowHint
  TButton class 2068
  TCheckBox class 2145
  TComboBox class 2220
  TCustomScreenTipsPopup class 322
  TEdit class 2495
  TGroupBox class 2561
  TLabel class 2633
  TListBox class 2692

TMemo class 2770
TRadioButton class 2843
TScrollBar class 2914
TStaticText class 2973
TTabbedNotebook class 4428
TTabSet class 4502
TValueListEditor class 4832
TWebBrowser class 539
TWebBrowser_V1 class 592
ShowHostAction
TSocketForm class 170
ShowHostActionExecute
TSocketForm class 179
ShowImage
TScreenTipItem class 334
ShrinkToFit
TTabSet class 4504
Side
TShadowWindow class 405
Signature
IDataBlock interface 273
TDataBlock class 206
signed char
Variant class 3463
signed char*
Variant class 3463
Silent
IWebBrowser2 interface 762
IWebBrowser2Disp interface 773
TInternetExplorer class 456
TWebBrowser class 541
SimpleDS 887
SimpleDS namespace 887
Classes 888
SimpleDS.TInternalSQLDataSet 888
SimpleDS.TInternalSQLDataSet.CommandText 891
SimpleDS.TInternalSQLDataSet.CommandType 892
SimpleDS.TInternalSQLDataSet.DataSource 893
SimpleDS.TInternalSQLDataSet.GetMetadata 894
SimpleDS.TInternalSQLDataSet.MaxBlobSize 894
SimpleDS.TInternalSQLDataSet.ParamCheck 895
SimpleDS.TInternalSQLDataSet.Params 895
SimpleDS.TInternalSQLDataSet.SortFieldNames 896
SimpleDS.TSimpleDataSet 896
SimpleDS.TSimpleDataSet.Active 904
SimpleDS.TSimpleDataSet.AfterApplyUpdates 931
SimpleDS.TSimpleDataSet.AfterCancel 931
SimpleDS.TSimpleDataSet.AfterClose 933
SimpleDS.TSimpleDataSet.AfterDelete 935
SimpleDS.TSimpleDataSet.AfterEdit 937
SimpleDS.TSimpleDataSet.AfterExecute 938
SimpleDS.TSimpleDataSet.AfterGetParams 939
SimpleDS.TSimpleDataSet.AfterGetRecords 940
SimpleDS.TSimpleDataSet.AfterInsert 940
SimpleDS.TSimpleDataSet.AfterOpen 941
SimpleDS.TSimpleDataSet.AfterPost 942
SimpleDS.TSimpleDataSet.AfterRefresh 943
SimpleDS.TSimpleDataSet.AfterRowRequest 943
SimpleDS.TSimpleDataSet.AfterScroll 944
SimpleDS.TSimpleDataSet.Aggregates 906
SimpleDS.TSimpleDataSet.AggregatesActive 907
SimpleDS.TSimpleDataSet.AllocConnection 900
SimpleDS.TSimpleDataSet.AllocDataSet 900
SimpleDS.TSimpleDataSet.AllocProvider 901
SimpleDS.TSimpleDataSet.AutoCalcFields 907
SimpleDS.TSimpleDataSet.BeforeApplyUpdates 944
SimpleDS.TSimpleDataSet.BeforeCancel 945
SimpleDS.TSimpleDataSet.BeforeClose 946
SimpleDS.TSimpleDataSet.BeforeDelete 946
SimpleDS.TSimpleDataSet.BeforeEdit 947
SimpleDS.TSimpleDataSet.BeforeExecute 947
SimpleDS.TSimpleDataSet.BeforeGetParams 948
SimpleDS.TSimpleDataSet.BeforeGetRecords 949
SimpleDS.TSimpleDataSet.BeforeInsert 950
SimpleDS.TSimpleDataSet.BeforeOpen 952
SimpleDS.TSimpleDataSet.BeforePost 952
SimpleDS.TSimpleDataSet.BeforeRefresh 953
SimpleDS.TSimpleDataSet.BeforeRowRequest 953

SimpleDS.TSimpleDataSet.BeforeScroll 954
SimpleDS.TSimpleDataSet.Connection 908
SimpleDS.TSimpleDataSet.Constraints 908
SimpleDS.TSimpleDataSet.Create 899
SimpleDS.TSimpleDataSet.DataSet 909
SimpleDS.TSimpleDataSet.Destroy 900
SimpleDS.TSimpleDataSet.DisableStringTrim 909
SimpleDS.TSimpleDataSet.FetchOnDemand 910
SimpleDS.TSimpleDataSet.FieldDefs 911
SimpleDS.TSimpleDataSet.FileName 914
SimpleDS.TSimpleDataSet.Filter 915
SimpleDS.TSimpleDataSet.Filtered 917
SimpleDS.TSimpleDataSet.FilterOptions 917
SimpleDS.TSimpleDataSet.IndexDefs 918
SimpleDS.TSimpleDataSet.IndexFieldNames 919
SimpleDS.TSimpleDataSet.IndexName 920
SimpleDS.TSimpleDataSet.Loaded 901
SimpleDS.TSimpleDataSet.MasterFields 921
SimpleDS.TSimpleDataSet.MasterSource 921
SimpleDS.TSimpleDataSet.Notification 901
SimpleDS.TSimpleDataSet.ObjectView 922
SimpleDS.TSimpleDataSet.OnCalcFields 955
SimpleDS.TSimpleDataSet.OnDeleteError 955
SimpleDS.TSimpleDataSet.OnEditError 956
SimpleDS.TSimpleDataSet.OnFilterRecord 956
SimpleDS.TSimpleDataSet.OnNewRecord 957
SimpleDS.TSimpleDataSet.OnPostError 957
SimpleDS.TSimpleDataSet.OnReconcileError 958
SimpleDS.TSimpleDataSet.OpenCursor 902
SimpleDS.TSimpleDataSet.PacketRecords 923
SimpleDS.TSimpleDataSet.Params 924
SimpleDS.TSimpleDataSet.PSGetCommandText 902
SimpleDS.TSimpleDataSet.ReadOnly 929
SimpleDS.TSimpleDataSet.SetConnection 903
SimpleDS.TSimpleDataSet.StoreDefs 929
SimplisticClear
TCustomVariantType class 5005
SimplisticCopy
TCustomVariantType class 5005
SiteComp 959
SiteComp namespace 959
Classes 963
Constants 1133
Functions 1120
Interfaces 977
Structs, Records, Enums 1121
Types 1128
Variables 1130
SiteComp.AdapterActionHTMLElementTypeNames 1133
SiteComp.AdapterActionHTMLElementTypeNames constant 1133
SiteComp.AdapterDisplayHTMLElementTypeNames 1133
SiteComp.AdapterDisplayHTMLElementTypeNames constant 1133
SiteComp.AdapterDisplayViewModeTypeNames 1134
SiteComp.AdapterDisplayViewModeTypeNames constant 1134
SiteComp.AdapterInputHTMLElementTypeNames 1134
SiteComp.AdapterInputHTMLElementTypeNames constant 1134
SiteComp.ExecuteScript 1130
SiteComp.ExecuteScript variable 1130
SiteComp.FullyQualifiedFieldName 1120
SiteComp.FullyQualifiedFieldName function 1120
SiteComp.FullyQualifiedName 1120
SiteComp.FullyQualifiedName function 1120
SiteComp.GetHTMLSampleImage 1130
SiteComp.GetHTMLSampleImage variable 1130
SiteComp.IActionImageProducer 978
SiteComp.IActionImageProducer.GetAdapterEventImage 980
SiteComp.IActionImageProducer.GetDisplayStyle 980
SiteComp.IAdapterAccess 981
SiteComp.IAdapterAccess.HasExecuteAccess 982
SiteComp.IAdapterAccess.HasModifyAccess 982
SiteComp.IAdapterAccess.HasViewAccess 982
SiteComp.IAdapterActionAccess 982
SiteComp.IAdapterActionAccess.HasExecuteAccess 983
SiteComp.IAdapterActionAttributes 983
SiteComp.IAdapterActionAttributes.GetVisible 984
SiteComp.IAdapterActionAttributes.Visible 984

SiteComp.IAdapterDispatcher 985
SiteComp.IAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue 986
SiteComp.IAdapterDispatcher.EncodeAdapterItemRequestAsHREF 986
SiteComp.IAdapterDispatcher.GetAdapterItemRequestParams 987
SiteComp.IAdapterDispatchParams 987
SiteComp.IAdapterDispatchParams.GetHandler 989
SiteComp.IAdapterDispatchParams.GetObjectIdentifier 989
SiteComp.IAdapterDispatchParams.GetParams 989
SiteComp.IAdapterDispatchParams.GetRequestIdentifier 989
SiteComp.IAdapterDispatchParams.Handler 990
SiteComp.IAdapterDispatchParams.ObjectIdentifier 990
SiteComp.IAdapterDispatchParams.Params 991
SiteComp.IAdapterDispatchParams.RequestIdentifier 991
SiteComp.IAdapterError 992
SiteComp.IAdapterError.ErrorText 994
SiteComp.IAdapterError.Field 994
SiteComp.IAdapterError.GetErrorText 993
SiteComp.IAdapterError.GetField 993
SiteComp.IAdapterError.GetID 993
SiteComp.IAdapterError.ID 994
SiteComp.IAdapterErrors 995
SiteComp.IAdapterErrors.ClearErrors 996
SiteComp.IAdapterErrors.DefineLabel 996
SiteComp.IAdapterErrors.ErrorCount 998
SiteComp.IAdapterErrors.Errors 998
SiteComp.IAdapterErrors.GetError 996
SiteComp.IAdapterErrors.GetErrorCount 997
SiteComp.IAdapterErrors.GetObjectErrors 997
SiteComp.IAdapterErrors.HasObjectErrors 997
SiteComp.IAdapterFieldAccess 998
SiteComp.IAdapterFieldAccess.HasModifyAccess 999
SiteComp.IAdapterFieldAccess.HasViewAccess 999
SiteComp.IAdapterFieldAttributes 999
SiteComp.IAdapterFieldAttributes.GetRequired 1000
SiteComp.IAdapterFieldAttributes.GetVisible 1000
SiteComp.IAdapterFieldAttributes.Required 1001
SiteComp.IAdapterFieldAttributes.Visible 1001
SiteComp.IAdapterHiddenField 1001
SiteComp.IAdapterHiddenField.GetName 1002
SiteComp.IAdapterHiddenField.GetValue 1002
SiteComp.IAdapterHiddenField.Name 1003
SiteComp.IAdapterHiddenField.Value 1003
SiteComp.IAdapterHiddenFields 1003
SiteComp.IAdapterHiddenFields.FieldCount 1005
SiteComp.IAdapterHiddenFields.Fields 1005
SiteComp.IAdapterHiddenFields.GetField 1004
SiteComp.IAdapterHiddenFields.GetFieldCount 1004
SiteComp.IAdapterItemRequestParams 1005
SiteComp.IAdapterItemRequestParams.FailurePage 1008
SiteComp.IAdapterItemRequestParams.GetFailurePage 1006
SiteComp.IAdapterItemRequestParams.GetParamValues 1007
SiteComp.IAdapterItemRequestParams.GetSuccessPage 1007
SiteComp.IAdapterItemRequestParams.ParamValues 1009
SiteComp.IAdapterItemRequestParams.SetFailurePage 1007
SiteComp.IAdapterItemRequestParams.SetSuccessPage 1008
SiteComp.IAdapterItemRequestParams.SuccessPage 1009
SiteComp.IAdapterMode 1009
SiteComp.IAdapterMode.AdapterMode 1011
SiteComp.IAdapterMode.GetAdapterMode 1010
SiteComp.IAdapterMode.GetNeedAdapterMode 1010
SiteComp.IAdapterMode.NeedAdapterMode 1011
SiteComp.IAdapterMode.RestoreDefaultAdapterMode 1011
SiteComp.IAdapterMode.SetAdapterMode 1011
SiteComp.IAdapterRequestHandler 1012
SiteComp.IAdapterRequestHandler.CreateRequestContext 1013
SiteComp.IAdapterRequestHandler.HandleRequest 1013
SiteComp.IAsFormatted 1013
SiteComp.IAsFormatted.AsFormatted 1014
SiteComp.IDefaultPageFileName 1014
SiteComp.IDefaultPageFileName.SetDefaultPageFileName 1015
SiteComp.IDEFileManager 1131
SiteComp.IDEFileManager variable 1131
SiteComp.IFieldValuesAdapter 1015
SiteComp.IFieldValuesAdapter.GetField 1016
SiteComp.IFieldValuesAdapter.GetValueIndex 1016

SiteComp.IFindIncludeFile 1016
SiteComp.IFindIncludeFile.FindIncludeFile 1017
SiteComp.IGetAdapterActions 1017
SiteComp.IGetAdapterActions.GetAdapterActions 1018
SiteComp.IGetAdapterDisplayCharacteristics 1018
SiteComp.IGetAdapterDisplayCharacteristics.GetAdapterDisplayCharacteristics 1019
SiteComp.IGetAdapterErrors 1019
SiteComp.IGetAdapterErrors.GetAdapterErrors 1020
SiteComp.IGetAdapterFields 1020
SiteComp.IGetAdapterFields.GetAdapterFields 1021
SiteComp.IGetAdapterHiddenFields 1021
SiteComp.IGetAdapterHiddenFields.GetAdapterHiddenFields 1022
SiteComp.IGetAdapterHiddenFields.GetAdapterHiddenRecordFields 1022
SiteComp.IGetAdapterImage 1022
SiteComp.IGetAdapterImage.GetAdapterImage 1023
SiteComp.IGetAdapterItemRequestParams 1023
SiteComp.IGetAdapterItemRequestParams.GetAdapterItemRequestParams 1024
SiteComp.IGetAdapterRequestDefaultResponse 1024
SiteComp.IGetAdapterRequestDefaultResponse.FailurePage 1026
SiteComp.IGetAdapterRequestDefaultResponse.GetFailurePage 1025
SiteComp.IGetAdapterRequestDefaultResponse.GetSuccessPage 1025
SiteComp.IGetAdapterRequestDefaultResponse.SuccessPage 1026
SiteComp.IGetAdapterValuesList 1026
SiteComp.IGetAdapterValuesList.GetAdapterValuesList 1027
SiteComp.IGetDefaultAction 1027
SiteComp.IGetDefaultAction.DefaultAction 1028
SiteComp.IGetDefaultAction.GetDefaultAction 1028
SiteComp.IGetDesigntimeWarnings 1029
SiteComp.IGetDesigntimeWarnings.GetDesigntimeWarnings 1029
SiteComp.IGetFieldValuesAdapter 1030
SiteComp.IGetFieldValuesAdapter.GetFieldValuesAdapter 1031
SiteComp.IGetHTMLStyle 1031
SiteComp.IGetHTMLStyle.GetDisplayStyle 1032
SiteComp.IGetHTMLStyle.GetInputStyle 1033
SiteComp.IGetHTMLStyle.GetViewMode 1033
SiteComp.IGetProducerComponent 1034
SiteComp.IGetProducerComponent.GetProducerComponent 1035
SiteComp.IGetScriptObject 1035
SiteComp.IGetScriptObject.GetScriptObject 1036
SiteComp.IGetScriptObjectClassKey 1036
SiteComp.IGetScriptObjectClassKey.GetScriptObjectClassKey 1037
SiteComp.IGetWebAppComponents 1037
SiteComp.IGetWebAppComponents.GetAdapterDispatcher 1038
SiteComp.IGetWebAppComponents.GetLocateFileService 1038
SiteComp.IGetWebAppComponents.GetPageDispatcher 1038
SiteComp.IGetWebAppComponents.GetWebApplicationInfo 1038
SiteComp.IGetWebAppComponents.GetWebEndUser 1039
SiteComp.IGetWebAppComponents.GetWebUserList 1039
SiteComp.IGetWebAppComponents.WebApplicationInfo 1039
SiteComp.IGetWebAppComponents.WebEndUser 1040
SiteComp.IIDEFileManager 1040
SiteComp.IIDEFileManager.SearchFile 1041
SiteComp.IImageProducer 1041
SiteComp.IImageProducer.GetAdapterImage 1042
SiteComp.IInternetEnvOptions 1042
SiteComp.IInternetEnvOptions.DisableDebugger 1044
SiteComp.IInternetEnvOptions.GetDisableDebugging 1043
SiteComp.IInternetEnvOptions.GetHTMLExt 1044
SiteComp.IInternetEnvOptions.GetSampleImageFile 1044
SiteComp.IInternetEnvOptions.GetScriptTimeoutSecs 1044
SiteComp.IInternetEnvOptions.HTMLExt 1045
SiteComp.IInternetEnvOptions.SampleImageFile 1045
SiteComp.IInternetEnvOptions.ScriptTimeoutSecs 1045
SiteComp.IInternetExecuteScript 1045
SiteComp.IInternetExecuteScript.CanExecuteScript 1046
SiteComp.IInternetExecuteScript.ExecuteScript 1046
SiteComp.IInternetFileTypes 1047

SiteComp.IInternetFileTypes.FindFileExt 1047
SiteComp.IInternetScriptDebugger 1048
SiteComp.IInternetScriptDebugger.CanDebugScript 1048
SiteComp.IInternetScriptDebugger.DebugScript 1049
SiteComp.IIsAdapterActionList 1049
SiteComp.IIsAdapterActionList.GetIsAdapterActionList 1050
SiteComp.IIsAdapterActionList.IsAdapterActionList 1050
SiteComp.IIterateIntfSupport 1050
SiteComp.IIterateIntfSupport.GetCurrentIntf 1051
SiteComp.IIterateObjectSupport 1051
SiteComp.IIterateObjectSupport.GetCurrentObject 1052
SiteComp.IIteratorIndex 1052
SiteComp.IIteratorIndex.GetIteratorIndex 1053
SiteComp.IIteratorIndex.InIterator 1053
SiteComp.IIteratorIndex.IteratorIndex 1054
SiteComp.IIteratorSupport 1054
SiteComp.IIteratorSupport.EndIterator 1056
SiteComp.IIteratorSupport.NextIteration 1056
SiteComp.IIteratorSupport.StartIterator 1056
SiteComp.InetFileTypes 1131
SiteComp.InetFileTypes variable 1131
SiteComp.INotifyAdapterChange 1057
SiteComp.INotifyAdapterChange.NotifyAdapterChange 1058
SiteComp.INotifyList 1058
SiteComp.INotifyList.AddNotify 1059
SiteComp.INotifyList.RemoveNotify 1059
SiteComp.InternetEnvOptions 1131
SiteComp.InternetEnvOptions variable 1131
SiteComp.IPageDispatcher 1060
SiteComp.IPageDispatcher.DefaultPageName 1064
SiteComp.IPageDispatcher.DispatchPageName 1061
SiteComp.IPageDispatcher.GetCanView 1062
SiteComp.IPageDispatcher.GetDefaultPageName 1062
SiteComp.IPageDispatcher.GetLoginRequired 1062
SiteComp.IPageDispatcher.PageNameOfRequest 1063
SiteComp.IPageDispatcher.RedirectToPageName 1063
SiteComp.IPageResult 1064
SiteComp.IPageResult.DispatchPage 1065
SiteComp.IPageResult.IncludePage 1065
SiteComp.IPageResult.RedirectToPage 1066
SiteComp.ISessionsService 1066
SiteComp.ISessionsService.CheckStatusChange 1067
SiteComp.ISessionsService.Clear 1067
SiteComp.ISessionsService.NewSession 1068
SiteComp.ISessionsService.NotifyStatusChange 1068
SiteComp.ISessionsServiceConnectID 1068
SiteComp.ISessionsServiceConnectID.Connect 1069
SiteComp.ISessionsServiceConnectID.ConnectActive 1069
SiteComp.ISessionsServicePersist 1070
SiteComp.ISessionsServicePersist.LoadFromFile 1070
SiteComp.ISessionsServicePersist.LoadFromStream 1071
SiteComp.ISessionsServicePersist.SaveToFile 1071
SiteComp.ISessionsServicePersist.SaveToStream 1071
SiteComp.ISessionsServiceStoreID 1071
SiteComp.ISessionsServiceStoreID.ReadSessionID 1072
SiteComp.ISessionsServiceStoreID.WriteSessionID 1072
SiteComp.IValuesListAdapter 1072
SiteComp.IValuesListAdapter.GetListImage 1074
SiteComp.IValuesListAdapter.GetListImageOfValue 1074
SiteComp.IValuesListAdapter.GetListName 1075
SiteComp.IValuesListAdapter.GetListNameOfValue 1075
SiteComp.IValuesListAdapter.GetListValue 1076
SiteComp.IWebActionsList 1076
SiteComp.IWebActionsList.FindAction 1077
SiteComp.IWebActionsList.GetActionsList 1077
SiteComp.IWebActionsList.GetVisibleActions 1077
SiteComp.IWebActionsList.IsActionInUse 1078
SiteComp.IWebActionsList.VisibleActions 1078
SiteComp.IWebApplicationInfo 1078
SiteComp.IWebApplicationInfo.ApplicationAdapter 1080
SiteComp.IWebApplicationInfo.ApplicationTitle 1080
SiteComp.IWebApplicationInfo.GetApplicationAdapter 1079
SiteComp.IWebApplicationInfo.GetApplicationTitle 1080
SiteComp.IWebContainerSubComponent 1081
SiteComp.IWebDataFields 1081
SiteComp.IWebDataFields.FindField 1082
SiteComp.IWebDataFields.GetFieldsList 1082
SiteComp.IWebDataFields.GetVisibleFields 1083

SiteComp.IWebDataFields.IsFieldInUse 1083
SiteComp.IWebDataFields.VisibleFields 1083
SiteComp.IWebEnabled 1083
SiteComp.IWebEnabled.WebEnabled 1084
SiteComp.IWebFormComponentContainer 1084
SiteComp.IWebFormComponentContainer.Count 1086
SiteComp.IWebFormComponentContainer.GetCount 1085
SiteComp.IWebFormComponentContainer.GetItem 1085
SiteComp.IWebFormComponentContainer.Items 1086
SiteComp.IWebGetActionName 1086
SiteComp.IWebGetActionName.ActionName 1087
SiteComp.IWebGetActionName.GetActionName 1087
SiteComp.IWebGetDisplayLabel 1088
SiteComp.IWebGetDisplayLabel.GetDisplayLabel 1088
SiteComp.IWebGetDisplayWidth 1089
SiteComp.IWebGetDisplayWidth.GetDisplayWidth 1089
SiteComp.IWebGetFieldName 1090
SiteComp.IWebGetFieldName.FieldName 1091
SiteComp.IWebGetFieldName.GetFieldName 1090
SiteComp.IWebGetFieldValue 1091
SiteComp.IWebGetFieldValue.GetValue 1092
SiteComp.IWebGetFieldValue.Value 1092
SiteComp.IWebGetFieldValues 1092
SiteComp.IWebGetFieldValues.EndValueIterator 1093
SiteComp.IWebGetFieldValues.GetCurrentValue 1094
SiteComp.IWebGetFieldValues.HasValue 1094
SiteComp.IWebGetFieldValues.NextValueIteration 1094
SiteComp.IWebGetFieldValues.StartValueIterator 1095
SiteComp.IWebGetMaxLength 1095
SiteComp.IWebGetMaxLength.GetMaxLength 1096
SiteComp.IWebGetSessionIDs 1096
SiteComp.IWebGetSessionIDs.GetSessionIDs 1097
SiteComp.IWebImageHREF 1097
SiteComp.IWebImageHREF.WebImageHREF 1098
SiteComp.IWebInputName 1098
SiteComp.IWebInputName.GetInputName 1099
SiteComp.IWebInputName.InputName 1100
SiteComp.IWebIsDefaultAction 1100
SiteComp.IWebIsDefaultAction.IsDefaultAction 1101
SiteComp.IWebIsDefaultField 1101
SiteComp.IWebIsDefaultField.IsDefaultField 1102
SiteComp.IWebSession 1102
SiteComp.IWebSession.Terminate 1103
SiteComp.IWebSessionAttributes 1103
SiteComp.IWebSessionAttributes.ExpirationTime 1106
SiteComp.IWebSessionAttributes.Expired 1107
SiteComp.IWebSessionAttributes.GetExpirationTime 1104
SiteComp.IWebSessionAttributes.GetExpired 1105
SiteComp.IWebSessionAttributes.GetIsActive 1105
SiteComp.IWebSessionAttributes.GetStartTime 1105
SiteComp.IWebSessionAttributes.GetTerminated 1105
SiteComp.IWebSessionAttributes.GetTimeoutMinutes 1105
SiteComp.IWebSessionAttributes.GetTouchTime 1106
SiteComp.IWebSessionAttributes.IsActive 1107
SiteComp.IWebSessionAttributes.SetTimeoutMinutes 1106
SiteComp.IWebSessionAttributes.StartTime 1107
SiteComp.IWebSessionAttributes.Terminated 1107
SiteComp.IWebSessionAttributes.TimeoutMinutes 1107
SiteComp.IWebSessionAttributes.TouchTime 1108
SiteComp.IWebSessionID 1108
SiteComp.IWebSessionID.GetID 1109
SiteComp.IWebSessionID.ID 1109
SiteComp.IWebSessionIDs 1109
SiteComp.IWebSessionIDs.GetID 1110
SiteComp.IWebSessionIDs.GetIDCount 1111
SiteComp.IWebSessionIDs.IDCount 1111
SiteComp.IWebSessionIDs.IDs 1111
SiteComp.IWebSessionValues 1112
SiteComp.IWebSessionValues.GetItemCount 1113
SiteComp.IWebSessionValues.GetItems 1113
SiteComp.IWebSessionValues.GetValue 1113
SiteComp.IWebSessionValues.SetValue 1113
SiteComp.IWebSessionValues.Values 1114
SiteComp.IWebSetActionName 1114
SiteComp.IWebSetActionName.SetActionName 1115
SiteComp.IWebSetFieldName 1115
SiteComp.IWebSetFieldName.SetFieldName 1116
SiteComp.IWebUserList 1116

SiteComp.IWebUserList.CheckAccessRights 1117
SiteComp.IWebUserList.ValidateUser 1117
SiteComp.IWebValueIsEqual 1118
SiteComp.IWebValueIsEqual.IsEqual 1119
SiteComp.ScriptDebugger 1132
SiteComp.ScriptDebugger variable 1132
SiteComp.TAbstractDesigntimeWarnings 963
SiteComp.TAbstractDesigntimeWarnings.AddObject 964
SiteComp.TAbstractDesigntimeWarnings.AddString 964
SiteComp.TAbstractInetFileType 965
SiteComp.TAbstractInetFileType.FileExt 967
SiteComp.TAbstractInetFileType.FileExtCount 967
SiteComp.TAbstractInetFileType.GetExtCount 966
SiteComp.TAbstractInetFileType.GetFileExt 966
SiteComp.TAbstractInetFileType.GetTypeID 967
SiteComp.TAbstractInetFileType.TypeID 968
SiteComp.TAdapterActionHTMLElementType 1121
SiteComp.TAdapterActionHTMLElementType enumeration 1121
SiteComp.TAdapterDisplayCharacteristic 1122
SiteComp.TAdapterDisplayCharacteristic enumeration 1122
SiteComp.TAdapterDisplayCharacteristics 1128
SiteComp.TAdapterDisplayCharacteristics type 1128
SiteComp.TAdapterDisplayHTMLElementType 1123
SiteComp.TAdapterDisplayHTMLElementType enumeration 1123
SiteComp.TAdapterDisplayViewModeType 1123
SiteComp.TAdapterDisplayViewModeType enumeration 1123
SiteComp.TAdapterInputHTMLElementType 1124
SiteComp.TAdapterInputHTMLElementType enumeration 1124
SiteComp.TAdapterItemRequestParamOption 1125
SiteComp.TAdapterItemRequestParamOption enumeration 1125
SiteComp.TAdapterItemRequestParamOptions 1128
SiteComp.TAdapterItemRequestParamOptions type 1128
SiteComp.TDispatchPageFlag 1126
SiteComp.TDispatchPageFlag enumeration 1126
SiteComp.TDispatchPageFlags 1129
SiteComp.TDispatchPageFlags type 1129
SiteComp.TNotifyList 968
SiteComp.TNotifyList.AddNotify 970
SiteComp.TNotifyList.Count 972
SiteComp.TNotifyList.Create 969
SiteComp.TNotifyList.Destroy 970
SiteComp.TNotifyList.GetCount 971
SiteComp.TNotifyList.GetItem 971
SiteComp.TNotifyList.Items 972
SiteComp.TNotifyList.RemoveNotify 971
SiteComp.TQualifyOption 1126
SiteComp.TQualifyOption enumeration 1126
SiteComp.TScriptComponent 973
SiteComp.TScriptComponent.Create 975
SiteComp.TScriptComponent.GetComponent 975
SiteComp.TScriptComponent.Obj 975
SiteComp.TScriptObject 976
SiteComp.TSessionID 1129
SiteComp.TSessionID type 1129
SiteModuleHelper
    TCustomWebDataModule class 6555
SitePageModuleHelper
    TCustomWebPageModule class 6562
SiteProd 1135
SiteProd namespace 1135
    Classes 1136
SiteProd.TAdapterPageProducer 1136
SiteProd.TAdapterPageProducer.HTMLDoc 1139
SiteProd.TAdapterPageProducer.HTMLFile 1140
SiteProd.TAdapterPageProducer.OnAfterGetContent 1142
SiteProd.TAdapterPageProducer.OnBeforeGetContent 1142
SiteProd.TAdapterPageProducer.Styles 1140
SiteProd.TAdapterPageProducer.StylesFile 1141
SiteProd.TAdapterPageProducer.WebPageItems 1141
SiteProd.TBaseAdapterPageProducer 1143
SiteProd.TBaseAdapterPageProducer.CanAddClass 1146
SiteProd.TBaseAdapterPageProducer.ContentFromStream 1147
SiteProd.TBaseAdapterPageProducer.Create 1145
SiteProd.TBaseAdapterPageProducer.CreatePageElements 1147
SiteProd.TBaseAdapterPageProducer.DeclareStyles 1148
SiteProd.TBaseAdapterPageProducer.DeclareWarnings 1148

SiteProd.TBaseAdapterPageProducer.Destroy 1145
SiteProd.TBaseAdapterPageProducer.DoAfterGetContent 1148
SiteProd.TBaseAdapterPageProducer.DoBeforeGetContent 1149
SiteProd.TBaseAdapterPageProducer.DoTagEvent 1149
SiteProd.TBaseAdapterPageProducer.GeneratePageElements 1150
SiteProd.TBaseAdapterPageProducer.GetContentOptions 1150
SiteProd.TBaseAdapterPageProducer.GetDefaultTemplate 1150
SiteProd.TBaseAdapterPageProducer.GetRequiredTags 1151
SiteProd.TBaseAdapterPageProducer.GetScriptEngines 1151
SiteProd.TBaseAdapterPageProducer.GetTemplateFileType 1152
SiteProd.TBaseAdapterPageProducer.GetXMLData 1152
SiteProd.TBaseAdapterPageProducer.GetXSLData 1152
SiteProd.TBaseAdapterPageProducer.HandleScriptTag 1153
SiteProd.TBaseAdapterPageProducer.HasHTMLBrowserView 1153
SiteProd.TBaseAdapterPageProducer.HasScriptView 1154
SiteProd.TBaseAdapterPageProducer.HasXMLBrowserView 1154
SiteProd.TBaseAdapterPageProducer.HasXSLBrowserView 1154
SiteProd.TBaseAdapterPageProducer.ImplCanAddClass 1155
SiteProd.TBaseAdapterPageProducer.OnAfterGetContent 1158
SiteProd.TBaseAdapterPageProducer.OnBeforeGetContent 1159
SiteProd.TBaseAdapterPageProducer.PageElements 1157
SiteProd.TBaseAdapterPageProducer.SetStyles 1155
SiteProd.TBaseAdapterPageProducer.SetStylesFile 1156
SiteProd.TBaseAdapterPageProducer.SetWebContentOptions 1156
SiteProd.TBaseAdapterPageProducer.Styles 1157
SiteProd.TBaseAdapterPageProducer.StylesFile 1158
SiteProd.TCustomAdapterPageProducer 1159
SiteProd.TCustomAdapterPageProducer.Create 1161
SiteProd.TCustomAdapterPageProducer.Destroy 1162
SiteProd.TCustomAdapterPageProducer.GetTemplateFileName 1162
SiteProd.TCustomAdapterPageProducer.GetTemplateStream 1163
SiteProd.TCustomAdapterPageProducer.HTMLDoc 1165
SiteProd.TCustomAdapterPageProducer.HTMLFile 1166
SiteProd.TCustomAdapterPageProducer.SetHTMLDoc 1163
SiteProd.TCustomAdapterPageProducer.SetHTMLFile 1164
SiteProd.TCustomAdapterPageProducer.WebContent 1164
SiteProd.TSitePageElements 1167
SiteProd.TSitePageElements.ServerFunctionsContent 1168
SiteProd.TSitePageElements.ServerScriptContent 1169
SiteProd.TSitePageElements.ServerVariablesContent 1169
SiteProd.TSitePageElements.StylesContent 1169
SiteProd.TSitePageElements.WarningsContent 1169
Size
    IDataBlock interface 273
    TDataBlock class 206
SkipToPage
    TDefaultFieldsPagedAdapter class 5760
SleepCS
    TConditionVariableHelper class helper 3119
SmallChange
    TScrollBar class 2915
SmallString
    SmallString template 3308, 3309
SmallString template 3307
    [] 3309
    = 3309
    about SmallString template 3307
    AnsiString 3310
    SmallString 3308, 3309
    SmallString constructor 3308
    SmallString Members 3307
    SmallString operators 3309
SmallStringBase template 3310
    about SmallStringBase template 3310
    Data 3311
    Len 3311
    SmallStringBase data members 3310
    SmallStringBase Members 3310
SoapAction
    THTTPReqResp class 1291
SOAPAttachIntf 1171

SOAPAttachIntf namespace 1171
Classes 1172
Interfaces 1176
Types 1185
SOAPAttachIntf.IMimeAttachmentHandler 1176
SOAPAttachIntf.IMimeAttachmentHandler.AddSoapHeader 1177
SOAPAttachIntf.IMimeAttachmentHandler.CreateMimeStream 1178
SOAPAttachIntf.IMimeAttachmentHandler.FinalizeStream 1178
SOAPAttachIntf.IMimeAttachmentHandler.GetMIMEBoundary 1179
SOAPAttachIntf.IMimeAttachmentHandler.GetMIMEStream 1179
SOAPAttachIntf.IMimeAttachmentHandler.GetOnGetAttachmentEvent 1180
SOAPAttachIntf.IMimeAttachmentHandler.GetOnSendAttachmentEvent 1180
SOAPAttachIntf.IMimeAttachmentHandler.MIMEBoundary 1182
SOAPAttachIntf.IMimeAttachmentHandler.OnGetAttachment 1183
SOAPAttachIntf.IMimeAttachmentHandler.OnSendAttachment 1183
SOAPAttachIntf.IMimeAttachmentHandler.ProcessMultiPartForm 1180
SOAPAttachIntf.IMimeAttachmentHandler.SetMIMEBoundary 1181
SOAPAttachIntf.IMimeAttachmentHandler.SetOnGetAttachmentEvent 1181
SOAPAttachIntf.IMimeAttachmentHandler.SetOnSendAttachmentEvent 1182
SOAPAttachIntf.TOnGetAttachmentEvent 1185
SOAPAttachIntf.TOnGetAttachmentEvent type 1185
SOAPAttachIntf.TOnSendAttachmentEvent 1185
SOAPAttachIntf.TOnSendAttachmentEvent type 1185
SOAPAttachIntf.TSoapDataList 1172
SOAPAttachIntf.TSoapDataList.Add 1173
SOAPAttachIntf.TSoapDataList.Assign 1173
SOAPAttachIntf.TSoapDataList.Clear 1174
SOAPAttachIntf.TSoapDataList.Insert 1174
SOAPAttachIntf.TSoapDataList.Remove 1174
SOAPConn 1187
SOAPConn namespace 1187
Classes 1188
SOAPConn.TSoapConnection 1188
SOAPConn.TSoapConnection.AfterConnect 1201
SOAPConn.TSoapConnection.AfterDisconnect 1201
SOAPConn.TSoapConnection.Agent 1196
SOAPConn.TSoapConnection.BeforeConnect 1202
SOAPConn.TSoapConnection.BeforeDisconnect 1202
SOAPConn.TSoapConnection.Connected 1196
SOAPConn.TSoapConnection.Create 1190
SOAPConn.TSoapConnection.Destroy 1191
SOAPConn.TSoapConnection.DoConnect 1191
SOAPConn.TSoapConnection.DoDisconnect 1192
SOAPConn.TSoapConnection.GetConnected 1192
SOAPConn.TSoapConnection.GetProviderNames 1192
SOAPConn.TSoapConnection.GetServer 1193
SOAPConn.TSoapConnection.GetServerList 1193
SOAPConn.TSoapConnection.GetSOAPServer 1194
SOAPConn.TSoapConnection.GetSOAPServerIID 1194
SOAPConn.TSoapConnection.Notification 1194
SOAPConn.TSoapConnection.OnAfterExecute 1203
SOAPConn.TSoapConnection.OnBeforeExecute 1203
SOAPConn.TSoapConnection.Password 1197
SOAPConn.TSoapConnection.Proxy 1197
SOAPConn.TSoapConnection.ProxyByPass 1197
SOAPConn.TSoapConnection.RIO 1198
SOAPConn.TSoapConnection.SetSOAPServerIID 1195
SOAPConn.TSoapConnection.SetUseSoapAdapter 1195
SOAPConn.TSoapConnection.SOAPServerIID 1198
SOAPConn.TSoapConnection.URL 1199
SOAPConn.TSoapConnection.UserName 1200
SOAPConn.TSoapConnection.UseSOAPAdapter 1200
SOAPDm 1205
SOAPDm namespace 1205
Classes 1206
SOAPDm.TSoapDataModule 1206
SOAPDm.TSoapDataModule._AddRef 1210

SOAPDm.TSoapDataModule._Release 1210
SOAPDm.TSoapDataModule.AfterConstruction 1210
SOAPDm.TSoapDataModule.ApplyUpdates 1211
SOAPDm.TSoapDataModule.AS_ApplyUpdates 1211
SOAPDm.TSoapDataModule.AS_DataRequest 1212
SOAPDm.TSoapDataModule.AS_Execute 1213
SOAPDm.TSoapDataModule.AS_GetParams 1214
SOAPDm.TSoapDataModule.AS_GetProviderNames 1215
SOAPDm.TSoapDataModule.AS_GetRecords 1215
SOAPDm.TSoapDataModule.AS_RowRequest 1216
SOAPDm.TSoapDataModule.BeforeDestruction 1217
SOAPDm.TSoapDataModule.Create 1208
SOAPDm.TSoapDataModule.DataRequest 1217
SOAPDm.TSoapDataModule.Destroy 1208
SOAPDm.TSoapDataModule.Execute 1218
SOAPDm.TSoapDataModule.GetParams 1219
SOAPDm.TSoapDataModule.GetProvider 1219
SOAPDm.TSoapDataModule.GetProviderNames 1220
SOAPDm.TSoapDataModule.GetRecords 1220
SOAPDm.TSoapDataModule.Lock 1221
SOAPDm.TSoapDataModule.NewInstance 1221
SOAPDm.TSoapDataModule.ProviderCount 1230
SOAPDm.TSoapDataModule.Providers 1230
SOAPDm.TSoapDataModule.RefCount 1231
SOAPDm.TSoapDataModule.RegisterProvider 1222
SOAPDm.TSoapDataModule.RowRequest 1222
SOAPDm.TSoapDataModule.SafeCallException 1223
SOAPDm.TSoapDataModule.SAS_ApplyUpdates 1223
SOAPDm.TSoapDataModule.SAS_DataRequest 1224
SOAPDm.TSoapDataModule.SAS_Execute 1225
SOAPDm.TSoapDataModule.SAS_GetParams 1226
SOAPDm.TSoapDataModule.SAS_GetProviderNames 1226
SOAPDm.TSoapDataModule.SAS_GetRecords 1227
SOAPDm.TSoapDataModule.SAS_RowRequest 1228
SOAPDm.TSoapDataModule.Unlock 1229
SOAPDm.TSoapDataModule.UnRegisterProvider 1229
SOAPDomConv 1233
SOAPDomConv namespace 1233
Classes 1234
Interfaces 1241
Structs, Records, Enums 1244
Types 1245
SOAPDomConv.IDOMHeaderProcessor 1241
SOAPDomConv.IDOMHeaderProcessor.CreateHeader 1242
SOAPDomConv.IDOMHeaderProcessor.ProcessHeader 1242
SOAPDomConv.TDOMHeaderProcessorArray 1245
SOAPDomConv.TDOMHeaderProcessorArray type 1245
SOAPDomConv.TDOMHeaderProcessorEntry 1244
SOAPDomConv.TDOMHeaderProcessorEntry record 1244
SOAPDomConv.TSOAPDOMProcessor 1234
SOAPDomConv.TSOAPDOMProcessor._AddRef 1236
SOAPDomConv.TSOAPDOMProcessor._Release 1236
SOAPDomConv.TSOAPDOMProcessor.AddHeaderProcessor 1237
SOAPDomConv.TSOAPDOMProcessor.AfterConstruction 1237
SOAPDomConv.TSOAPDOMProcessor.DefaultProcessHeader 1238
SOAPDomConv.TSOAPDOMProcessor.Destroy 1235
SOAPDomConv.TSOAPDOMProcessor.FHeaderProcessors 1235
SOAPDomConv.TSOAPDOMProcessor.FindHeaderProcessor 1238
SOAPDomConv.TSOAPDOMProcessor.MakeHeaderNodes 1239
SOAPDomConv.TSOAPDOMProcessor.NewInstance 1239
SOAPHTTPClient 1247
SOAPHTTPClient namespace 1247
Classes 1248
SOAPHTTPClient.THTTPRIO 1248
SOAPHTTPClient.THTTPRIO.Converter 1252
SOAPHTTPClient.THTTPRIO.Create 1249
SOAPHTTPClient.THTTPRIO.Destroy 1250
SOAPHTTPClient.THTTPRIO.HTTPWebNode 1252
SOAPHTTPClient.THTTPRIO.Notification 1250
SOAPHTTPClient.THTTPRIO.Port 1253
SOAPHTTPClient.THTTPRIO.QueryInterface 1251
SOAPHTTPClient.THTTPRIO.Service 1253
SOAPHTTPClient.THTTPRIO.URL 1254
SOAPHTTPClient.THTTPRIO.WSDLItems 1255
SOAPHTTPClient.THTTPRIO.WSDLLocation 1255

SOAPHTTPDisp 1257
SOAPHTTPDisp namespace 1257
Classes 1258
Interfaces 1262
SOAPHTTPDisp.IHTTPSoapDispatch 1262
SOAPHTTPDisp.IHTTPSoapDispatch.DispatchSOAP 1263
SOAPHTTPDisp.THTTPSoapDispatchNode 1258
SOAPHTTPDisp.THTTPSoapDispatchNode.Dispatcher 1261
SOAPHTTPDisp.THTTPSoapDispatchNode.DispatchSOAP 1259
SOAPHTTPDisp.THTTPSoapDispatchNode.FSoapDispatcher 1259
SOAPHTTPDisp.THTTPSoapDispatchNode.Notification 1260
SOAPHTTPPasInv 1265
SOAPHTTPPasInv namespace 1265
Classes 1266
SOAPHTTPPasInv.THTTPSoapCppInvoker 1266
SOAPHTTPPasInv.THTTPSoapPascalInvoker 1267
SOAPHTTPPasInv.THTTPSoapPascalInvoker.DispatchSOAP 1269
SOAPHTTPTrans 1271
SOAPHTTPTrans namespace 1271
Classes 1272
Interfaces 1296
Structs, Records, Enums 1298
Types 1299
SOAPHTTPTrans.ESOAPHTTPException 1272
SOAPHTTPTrans.ESOAPHTTPException.Create 1274
SOAPHTTPTrans.ESOAPHTTPException.CreateFmt 1274
SOAPHTTPTrans.ESOAPHTTPException.StatusCode 1275
SOAPHTTPTrans.IHTTPReqResp 1296
SOAPHTTPTrans.IHTTPReqResp.GetHTTPReqResp 1297
SOAPHTTPTrans.SOAPInvokeOptions 1298
SOAPHTTPTrans.SOAPInvokeOptions enumeration 1298
SOAPHTTPTrans.TBeforePostEvent 1299
SOAPHTTPTrans.TBeforePostEvent type 1299
SOAPHTTPTrans.THTTPReqResp 1276
SOAPHTTPTrans.THTTPReqResp._AddRef 1279
SOAPHTTPTrans.THTTPReqResp._Release 1280
SOAPHTTPTrans.THTTPReqResp.AfterConstruction 1280
SOAPHTTPTrans.THTTPReqResp.Agent 1288
SOAPHTTPTrans.THTTPReqResp.BeforeExecute 1280
SOAPHTTPTrans.THTTPReqResp.Check 1281
SOAPHTTPTrans.THTTPReqResp.CheckContentType 1282
SOAPHTTPTrans.THTTPReqResp.Connect 1282
SOAPHTTPTrans.THTTPReqResp.ConnectTimeout 1288
SOAPHTTPTrans.THTTPReqResp.Create 1278
SOAPHTTPTrans.THTTPReqResp.Destroy 1278
SOAPHTTPTrans.THTTPReqResp.Execute 1283
SOAPHTTPTrans.THTTPReqResp.Get 1283
SOAPHTTPTrans.THTTPReqResp.GetHTTPReqResp 1284
SOAPHTTPTrans.THTTPReqResp.GetMimeBoundary 1284
SOAPHTTPTrans.THTTPReqResp.HandleWinInetError 1285
SOAPHTTPTrans.THTTPReqResp.InvokeOptions 1288
SOAPHTTPTrans.THTTPReqResp.MaxSinglePostSize 1289
SOAPHTTPTrans.THTTPReqResp.NewInstance 1285
SOAPHTTPTrans.THTTPReqResp.OnBeforePost 1294
SOAPHTTPTrans.THTTPReqResp.OnPostingData 1294
SOAPHTTPTrans.THTTPReqResp.OnReceivingData 1295
SOAPHTTPTrans.THTTPReqResp.OnWinInetError 1295
SOAPHTTPTrans.THTTPReqResp.Password 1289
SOAPHTTPTrans.THTTPReqResp.Proxy 1289
SOAPHTTPTrans.THTTPReqResp.ProxyByPass 1290
SOAPHTTPTrans.THTTPReqResp.Receive 1285
SOAPHTTPTrans.THTTPReqResp.ReceiveTimeout 1290
SOAPHTTPTrans.THTTPReqResp.Send 1286
SOAPHTTPTrans.THTTPReqResp.SendGet 1286
SOAPHTTPTrans.THTTPReqResp.SendTimeout 1291
SOAPHTTPTrans.THTTPReqResp.SetMimeBoundary 1287
SOAPHTTPTrans.THTTPReqResp.SoapAction 1291
SOAPHTTPTrans.THTTPReqResp.UDDIBindingKey 1291
SOAPHTTPTrans.THTTPReqResp.UDDIOperator 1292
SOAPHTTPTrans.THTTPReqResp.URL 1292
SOAPHTTPTrans.THTTPReqResp.UserName 1293
SOAPHTTPTrans.THTTPReqResp.UseUTF8InHeader 1293
SOAPHTTPTrans.THTTPReqResp.WSDLView 1293
SOAPHTTPTrans.TPostingDataEvent 1299
SOAPHTTPTrans.TPostingDataEvent type 1299
SOAPHTTPTrans.TReceivingDataEvent 1300

SOAPHTTPTrans.TReceivingDataEvent type 1300
SOAPHTTPTrans.TSOAPInvokeOptions 1300
SOAPHTTPTrans.TSOAPInvokeOptions type 1300
SOAPHTTPTrans.TWinInetErrorEvent 1301
SOAPHTTPTrans.TWinInetErrorEvent type 1301
SOAPLinked 1303
SOAPLinked namespace 1303
Classes 1305
Interfaces 1332
Structs, Records, Enums 1336
Types 1337
SOAPLinked.ECustomException 1305
SOAPLinked.ECustomException.CatchException 1310
SOAPLinked.ECustomException.Create 1307
SOAPLinked.ECustomException.Destroy 1309
SOAPLinked.ECustomException.InstanceCount 1310
SOAPLinked.ECustomException.ThrowExcept 1310
SOAPLinked.EventKind 1336
SOAPLinked.EventKind enumeration 1336
SOAPLinked.IIntegerTest 1332
SOAPLinked.IIntegerTest.Add 1333
SOAPLinked.IIntegerTest.Doubler 1333
SOAPLinked.IObjectGetter 1333
SOAPLinked.IObjectGetter.GetAsObject 1334
SOAPLinked.IStringTest 1334
SOAPLinked.IStringTest.Append 1335
SOAPLinked.IStringTest.UpCase 1335
SOAPLinked.TCStringEvent 1337
SOAPLinked.TCStringEvent type 1337
SOAPLinked.TCustExCallback 1337
SOAPLinked.TCustExCallback type 1337
SOAPLinked.TFTCompT 1311
SOAPLinked.TFTCompT.CStringEvent 1312
SOAPLinked.TFTCompT.fireEvent 1312
SOAPLinked.TFTCompT.fireEventEx 1312
SOAPLinked.TFTCompT.StringEvent 1312
SOAPLinked.TFTCompT.VStringEvent 1313
SOAPLinked.TFTCompTI 1313
SOAPLinked.TFTCompTI.Add 1315
SOAPLinked.TFTCompTI.Append 1315
SOAPLinked.TFTCompTI.Doubler 1315
SOAPLinked.TFTCompTI.GetAsObject 1316
SOAPLinked.TFTCompTI.UpCase 1316
SOAPLinked.TLinkedRIO 1316
SOAPLinked.TLinkedRIO.Create 1317
SOAPLinked.TLinkedRIO.CreateFile 1318
SOAPLinked.TLinkedRIO.Destroy 1318
SOAPLinked.TLinkedRIO.GetResponseStream 1319
SOAPLinked.TLinkedRIO.QueryInterface 1319
SOAPLinked.TLinkedRIO.WebNode 1320
SOAPLinked.TLinkedWebNode 1321
SOAPLinked.TLinkedWebNode.BeforeExecute 1323
SOAPLinked.TLinkedWebNode.Create 1322
SOAPLinked.TLinkedWebNode.Destroy 1322
SOAPLinked.TLinkedWebNode.Execute 1324
SOAPLinked.TLinkedWebNode.GetMimeBoundary 1324
SOAPLinked.TLinkedWebNode.GetResponseStream 1325
SOAPLinked.TLinkedWebNode.Invoker 1326
SOAPLinked.TLinkedWebNode.MimeBoundary 1326
SOAPLinked.TLinkedWebNode.SetMimeBoundary 1325
SOAPLinked.TLogLinkedWebNode 1326
SOAPLinked.TLogLinkedWebNode.Create 1327
SOAPLinked.TLogLinkedWebNode.Destroy 1328
SOAPLinked.TLogLinkedWebNode.Execute 1328
SOAPLinked.TLogLinkedWebNode.GetResponseStream 1328
SOAPLinked.TLogLinkedWebNode.ReqFile 1329
SOAPLinked.TLogLinkedWebNode.RespFile 1329
SOAPLinked.TStringEvent 1338
SOAPLinked.TStringEvent type 1338
SOAPLinked.TVStringEvent 1338
SOAPLinked.TVStringEvent type 1338
SOAPLinked.TWinHandleUser 1329
SOAPLinked.TWinHandleUser.UseHandle1 1331
SOAPLinked.TWinHandleUser.UseHandle2 1331
SOAPMidas 1339
SOAPMidas namespace 1339
Functions 1347
Interfaces 1340

SOAPMidas.IAppServerSOAP 1340
SOAPMidas.IAppServerSOAP.SAS_ApplyUpdates 1341
SOAPMidas.IAppServerSOAP.SAS_DataRequest 1342
SOAPMidas.IAppServerSOAP.SAS_Execute 1342
SOAPMidas.IAppServerSOAP.SAS_GetParams 1343
SOAPMidas.IAppServerSOAP.SAS_GetProviderNames 1344
SOAPMidas.IAppServerSOAP.SAS_GetRecords 1344
SOAPMidas.IAppServerSOAP.SAS_RowRequest 1345
SOAPMidas.RegDefIAppServerInvClass 1347
SOAPMidas.RegDefIAppServerInvClass function 1347
SOAPPasInv 1349
SOAPPasInv namespace 1349
Classes 1350
Functions 1362
Interfaces 1359
Types 1363
SOAPPasInv.GetSOAPHeaders 1362
SOAPPasInv.GetSOAPHeaders function 1362
SOAPPasInv.IAdapterConverter 1359
SOAPPasInv.IAdapterConverter.GetDomConverter 1360
SOAPPasInv.IInvokerConverter 1360
SOAPPasInv.IInvokerConverter.GetInvoker 1361
SOAPPasInv.TAfterDispatchEvent 1363
SOAPPasInv.TAfterDispatchEvent type 1363
SOAPPasInv.TBeforeDispatchEvent 1363
SOAPPasInv.TBeforeDispatchEvent type 1363
SOAPPasInv.TBeforeDispatchEvent2 1364
SOAPPasInv.TBeforeDispatchEvent2 type 1364
SOAPPasInv.TOnExceptionEvent 1364
SOAPPasInv.TOnExceptionEvent type 1364
SOAPPasInv.TSoapPascalInvoker 1350
SOAPPasInv.TSoapPascalInvoker.AfterDispatchEvent 1356
SOAPPasInv.TSoapPascalInvoker.BeforeDispatchEvent 1357
SOAPPasInv.TSoapPascalInvoker.BeforeDispatchEvent2 1357
SOAPPasInv.TSoapPascalInvoker.Converter 1355
SOAPPasInv.TSoapPascalInvoker.Create 1351
SOAPPasInv.TSoapPascalInvoker.Destroy 1352
SOAPPasInv.TSoapPascalInvoker.FConverter 1352
SOAPPasInv.TSoapPascalInvoker.FDomConverter 1352
SOAPPasInv.TSoapPascalInvoker.FHeadersIn 1353
SOAPPasInv.TSoapPascalInvoker.FHeadersOut 1353
SOAPPasInv.TSoapPascalInvoker.FSOAPHeaders 1353
SOAPPasInv.TSoapPascalInvoker.GetDomConverter 1354
SOAPPasInv.TSoapPascalInvoker.GetInvoker 1354
SOAPPasInv.TSoapPascalInvoker.Invoke 1354
SOAPPasInv.TSoapPascalInvoker.Notification 1355
SOAPPasInv.TSoapPascalInvoker.OnExceptionEvent 1357
SOAPPasInv.TSoapPascalInvoker.OnGetAttachment 1358
SOAPPasInv.TSoapPascalInvoker.OnSendAttachment 1358
SOAPServerIID
TSoapConnection class 1198
SOAPToObject
TXMLData class 7800
soAutoCheckAccessPointViaUDDI enumeration member 1298
Socket
TClientSocket class 31
TServerSocket class 113
TSocketTransport class 235
SocketHandle
TCustomWinSocket class 89
Sockets 1367
Sockets namespace 1367
Classes 1369
Constants 1508
Structs, Records, Enums 1499
Types 1504
Sockets.CRLF 1508
Sockets.CRLF constant 1508
Sockets.ESocketError 1369
Sockets.TBaseSocket 1370
Sockets.TBaseSocket.Active 1383
Sockets.TBaseSocket.BlockMode 1385
Sockets.TBaseSocket.BytesReceived 1386
Sockets.TBaseSocket.BytesSent 1386
Sockets.TBaseSocket.Close 1373
Sockets.TBaseSocket.Create 1371
Sockets.TBaseSocket.Destroy 1372
Sockets.TBaseSocket.DoCreateHandle 1373

Sockets.TBaseSocket.DoDestroyHandle 1373
Sockets.TBaseSocket.DoHandleError 1373
Sockets.TBaseSocket.Domain 1386
Sockets.TBaseSocket.DoReceive 1374
Sockets.TBaseSocket.DoSend 1374
Sockets.TBaseSocket.ErrorCheck 1374
Sockets.TBaseSocket.Handle 1387
Sockets.TBaseSocket.Loaded 1374
Sockets.TBaseSocket.MapDomain 1375
Sockets.TBaseSocket.MapSockType 1375
Sockets.TBaseSocket.OnCreateHandle 1388
Sockets.TBaseSocket.OnDestroyHandle 1390
Sockets.TBaseSocket.OnError 1393
Sockets.TBaseSocket.OnReceive 1393
Sockets.TBaseSocket.OnSend 1394
Sockets.TBaseSocket.Open 1375
Sockets.TBaseSocket.PeekBuf 1376
Sockets.TBaseSocket.Protocol 1387
Sockets.TBaseSocket.ReceiveBuf 1376
Sockets.TBaseSocket.Receiveln 1379
Sockets.TBaseSocket.Select 1379
Sockets.TBaseSocket.SendBuf 1380
Sockets.TBaseSocket.Sendln 1380
Sockets.TBaseSocket.SendStream 1380
Sockets.TBaseSocket.SetBytesReceived 1382
Sockets.TBaseSocket.SetBytesSent 1382
Sockets.TBaseSocket.SockType 1387
Sockets.TBaseSocket.WaitForData 1383
Sockets.TClientSocketThread 1396
Sockets.TClientSocketThread.ClientSocket 1399
Sockets.TClientSocketThread.Create 1397
Sockets.TClientSocketThread.Destroy 1397
Sockets.TClientSocketThread.Execute 1398
Sockets.TClientSocketThread.ExecuteSyncProc 1398
Sockets.TClientSocketThread.ServerSocketThread 1399
Sockets.TClientSocketThread.SyncProc 1398
Sockets.TCustomIpClient 1400
Sockets.TCustomIpClient.Close 1402
Sockets.TCustomIpClient.Connect 1403
Sockets.TCustomIpClient.Connected 1405
Sockets.TCustomIpClient.Create 1402
Sockets.TCustomIpClient.Disconnect 1403
Sockets.TCustomIpClient.DoConnect 1403
Sockets.TCustomIpClient.DoDisconnect 1404
Sockets.TCustomIpClient.GetThreadObject 1404
Sockets.TCustomIpClient.OnConnect 1405
Sockets.TCustomIpClient.OnDisconnect 1406
Sockets.TCustomIpClient.Open 1404
Sockets.TCustomTcpServer 1406
Sockets.TCustomTcpServer.Accept 1409
Sockets.TCustomTcpServer.BlockMode 1411
Sockets.TCustomTcpServer.Close 1410
Sockets.TCustomTcpServer.Create 1408
Sockets.TCustomTcpServer.Destroy 1408
Sockets.TCustomTcpServer.DoAccept 1410
Sockets.TCustomTcpServer.Listen 1410
Sockets.TCustomTcpServer.Listening 1412
Sockets.TCustomTcpServer.OnAccept 1413
Sockets.TCustomTcpServer.OnGetThread 1415
Sockets.TCustomTcpServer.OnListening 1416
Sockets.TCustomTcpServer.Open 1411
Sockets.TCustomTcpServer.ServerSocketThread 1412
Sockets.TCustomTcpServer.WaitForConnection 1411
Sockets.TGetThreadEvent 1504
Sockets.TGetThreadEvent type 1504
Sockets.TIPHeader 1499
Sockets.TIPHeader record 1499
Sockets.TIpSocket 1416
Sockets.TIpSocket.Bind 1419
Sockets.TIpSocket.Create 1418
Sockets.TIpSocket.Domain 1423
Sockets.TIpSocket.GetSocketAddr 1419
Sockets.TIpSocket.LocalDomainName 1419
Sockets.TIpSocket.LocalHost 1423
Sockets.TIpSocket.LocalHostAddr 1420
Sockets.TIpSocket.LocalHostName 1420
Sockets.TIpSocket.LocalPort 1424
Sockets.TIpSocket.LookupHostAddr 1420

Sockets.TIpSocket.LookupHostName 1421
Sockets.TIpSocket.LookupPort 1421
Sockets.TIpSocket.LookupProtocol 1421
Sockets.TIpSocket.ReceiveFrom 1422
Sockets.TIpSocket.RemoteHost 1426
Sockets.TIpSocket.RemotePort 1428
Sockets.TIpSocket.SendTo 1422
Sockets.TRawSocket 1431
Sockets.TRawSocket.Create 1432
Sockets.TRawSocket.SockType 1433
Sockets.TServerSocketBlockMode 1500
Sockets.TServerSocketBlockMode enumeration 1500
Sockets.TServerSocketThread 1433
Sockets.TServerSocketThread.AddClientSocketThread 1436
Sockets.TServerSocketThread.ClearThreadPool 1436
Sockets.TServerSocketThread.Create 1434
Sockets.TServerSocketThread.CreateThread 1436
Sockets.TServerSocketThread.Destroy 1435
Sockets.TServerSocketThread.Execute 1436
Sockets.TServerSocketThread.FetchClientSocketThread 1437
Sockets.TServerSocketThread.OnGetThread 1439
Sockets.TServerSocketThread.RemoveClientSocketThread 1437
Sockets.TServerSocketThread.ServerSocket 1438
Sockets.TServerSocketThread.ThreadCacheSize 1438
Sockets.TServerSocketThread.ThreadPool 1438
Sockets.TSocketAcceptEvent 1504
Sockets.TSocketAcceptEvent type 1504
Sockets.TSocketBlockMode 1505
Sockets.TSocketBlockMode type 1505
Sockets.TSocketDataEvent 1505
Sockets.TSocketDataEvent type 1505
Sockets.TSocketDomain 1500
Sockets.TSocketDomain enumeration 1500
Sockets.TSocketErrorEvent 1505
Sockets.TSocketErrorEvent type 1505
Sockets.TSocketHost 1506
Sockets.TSocketHost type 1506
Sockets.TSocketNotifyEvent 1506
Sockets.TSocketNotifyEvent type 1506
Sockets.TSocketPort 1506
Sockets.TSocketPort type 1506
Sockets.TSocketProtocol 1507
Sockets.TSocketProtocol type 1507
Sockets.TSocketType 1502
Sockets.TSocketType enumeration 1502
Sockets.TTcpClient 1439
Sockets.TTcpClient.Active 1443
Sockets.TTcpClient.BlockMode 1445
Sockets.TTcpClient.Connected 1446
Sockets.TTcpClient.OnConnect 1450
Sockets.TTcpClient.OnCreateHandle 1451
Sockets.TTcpClient.OnDestroyHandle 1453
Sockets.TTcpClient.OnDisconnect 1456
Sockets.TTcpClient.OnError 1456
Sockets.TTcpClient.OnReceive 1456
Sockets.TTcpClient.OnSend 1457
Sockets.TTcpClient.RemoteHost 1446
Sockets.TTcpClient.RemotePort 1448
Sockets.TTcpServer 1459
Sockets.TTcpServer.Active 1463
Sockets.TTcpServer.BlockMode 1465
Sockets.TTcpServer.LocalHost 1466
Sockets.TTcpServer.LocalPort 1466
Sockets.TTcpServer.OnAccept 1469
Sockets.TTcpServer.OnCreateHandle 1472
Sockets.TTcpServer.OnDestroyHandle 1474
Sockets.TTcpServer.OnGetThread 1477
Sockets.TTcpServer.OnListening 1477
Sockets.TUdpSocket 1477
Sockets.TUdpSocket.Active 1480
Sockets.TUdpSocket.BlockMode 1482
Sockets.TUdpSocket.Create 1479
Sockets.TUdpSocket.LocalHost 1482
Sockets.TUdpSocket.LocalPort 1483
Sockets.TUdpSocket.OnConnect 1490
Sockets.TUdpSocket.OnCreateHandle 1490
Sockets.TUdpSocket.OnDestroyHandle 1493
Sockets.TUdpSocket.OnDisconnect 1495

Sockets.TUdpSocket.OnError 1496
Sockets.TUdpSocket.OnReceive 1496
Sockets.TUdpSocket.OnSend 1496
Sockets.TUdpSocket.RemoteHost 1486
Sockets.TUdpSocket.RemotePort 1488
SockType
TBaseSocket class 1387
TRawSocket class 1433
soDown enumeration member 3055
SoftTop
TTabSet class 4504
soHorizontal enumeration member 4645
soIgnoreInvalidCerts enumeration member 1298
soLeftRight enumeration member 4539
soMatchCase enumeration member 3055
soNone enumeration member 4645
soNoSOAPActionHeader enumeration member 1298
soNoValueForEmptySOAPAction enumeration member 1298
Sort
TWideStringList class 6821
Sorted
TComboBox class 2221
TCustomComboBox class 2335
TCustomListBox class 2434
TListBox class 2693
TWideStringList class 6823
SortFieldNames
TCustomSQLDataSet class 1566
TInternalSQLDataSet class 896
TSQLDataSet class 1642
soShowHeader enumeration member 364
soShowShortCut enumeration member 364
soUpDown enumeration member 4539
Source
TScriptFile class 6687
SourceXml
TXMLTransform class 7734
SourceXmlDocument
TXMLTransform class 7734
SourceXmlFile
TXMLTransform class 7735
soVertical enumeration member 4645
soWholeWord enumeration member 3055
specified
IDOMAttr interface 7525
splitText
IDOMText interface 7604
sPointer
sPointer template 3312
sPointer template 3311
& 3313
~sPointer 3312
-> 3313
about sPointer template 3311
sPointer 3312
sPointer constructor 3312
sPointer Members 3311
sPointer operators 3313
SPParamDesc class 1512
about SPParamDesc class 1512
iArgType 1513
iDataType 1514
iLen 1514
iParamNum 1514
iUnits1 1514
iUnits2 1515
SPParamDesc fields 1513
SPParamDesc Members 1513
szName 1515
sprintf
AnsiStringT template 3227
UnicodeString class 3408
WideString class 3483
SQL
TSQLQuery class 1663
SQLConnection
TCustomSQLDataSet class 1566
TSQLDataSet class 1642

TSQLMonitor class 1651
TSQLQuery class 1664
TSqlServerMethod class 1670
TSQLStoredProc class 1679
TSQLTable class 1693
SQLDllHandle
TSQLConnection class 1623
SQLError
TSQLConnection class 1605
SqlExpr 1509
SqlExpr namespace 1509
Classes 1512
Constants 1711
Functions 1696
Structs, Records, Enums 1699
Types 1706
SqlExpr.BlobTypeMap 1711
SqlExpr.BlobTypeMap constant 1711
SqlExpr.DataTypeMap 1712
SqlExpr.DataTypeMap constant 1712
SqlExpr.DefaultCursor 1712
SqlExpr.DefaultCursor constant 1712
SqlExpr.DefaultMaxBlobSize 1713
SqlExpr.DefaultMaxBlobSize constant 1713
SqlExpr.DefaultRowsetSize 1713
SqlExpr.DefaultRowsetSize constant 1713
SqlExpr.EConnectFlag 1699
SqlExpr.EConnectFlag enumeration 1699
SqlExpr.eSQLAutoIncr 1713
SqlExpr.eSQLAutoIncr constant 1713
SqlExpr.eSQLDefault 1714
SqlExpr.eSQLDefault constant 1714
SqlExpr.eSQLFunction 1714
SqlExpr.eSQLFunction constant 1714
SqlExpr.eSQLLocal 1714
SqlExpr.eSQLLocal constant 1714
SqlExpr.eSQLNonUnique 1715
SqlExpr.eSQLNonUnique constant 1715
SqlExpr.eSQLPackage 1715
SqlExpr.eSQLPackage constant 1715
SqlExpr.eSQLPrimaryKey 1715
SqlExpr.eSQLPrimaryKey constant 1715
SqlExpr.eSQLProcedure 1716
SqlExpr.eSQLProcedure constant 1716
SqlExpr.eSQLRowId 1716
SqlExpr.eSQLRowId constant 1716
SqlExpr.eSQLRowVersion 1716
SqlExpr.eSQLRowVersion constant 1716
SqlExpr.eSQLSynonym 1717
SqlExpr.eSQLSynonym constant 1717
SqlExpr.eSQLSysProcedure 1717
SqlExpr.eSQLSysProcedure constant 1717
SqlExpr.eSQLSystemTable 1717
SqlExpr.eSQLSystemTable constant 1717
SqlExpr.eSQLTable 1718
SqlExpr.eSQLTable constant 1718
SqlExpr.eSQLTempTable 1718
SqlExpr.eSQLTempTable constant 1718
SqlExpr.eSQLUnique 1718
SqlExpr.eSQLUnique constant 1718
SqlExpr.eSQLView 1719
SqlExpr.eSQLView constant 1719
SqlExpr.FldSubTypeMap 1719
SqlExpr.FldSubTypeMap constant 1719
SqlExpr.FldTypeMap 1719
SqlExpr.FldTypeMap constant 1719
SqlExpr.FreeProcParams 1696
SqlExpr.FreeProcParams function 1696
SqlExpr.GetConnectionNames 1696
SqlExpr.GetConnectionNames function 1696
SqlExpr.GetConnectionRegistryFile 1697
SqlExpr.GetConnectionRegistryFile function 1697
SqlExpr.GetDelegateDriverNames 1697
SqlExpr.GetDelegateDriverNames function 1697
SqlExpr.GetDriverNames 1697
SqlExpr.GetDriverNames function 1697
SqlExpr.GetDriverRegistryFile 1698
SqlExpr.GetDriverRegistryFile function 1698

SqlExpr.HourGlassCursor 1720
SqlExpr.HourGlassCursor constant 1720
SqlExpr.LoadParamListItems 1698
SqlExpr.LoadParamListItems function 1698
SqlExpr.SAnd 1721
SqlExpr.SAnd constant 1721
SqlExpr.SOrderBy 1721
SqlExpr.SOrderBy constant 1721
SqlExpr.SParam 1721
SqlExpr.SParam constant 1721
SqlExpr.SPParamDesc 1512
SqlExpr.SPParamDesc.iArgType 1513
SqlExpr.SPParamDesc.iDataType 1514
SqlExpr.SPParamDesc.iLen 1514
SqlExpr.SPParamDesc.iParamNum 1514
SqlExpr.SPParamDesc.iUnits1 1514
SqlExpr.SPParamDesc.iUnits2 1515
SqlExpr.SPParamDesc.szName 1515
SqlExpr.SQLResult 1706
SqlExpr.SQLResult type 1706
SqlExpr.SSelect 1722
SqlExpr.SSelect constant 1722
SqlExpr.SSelectSpaces 1722
SqlExpr.SSelectSpaces constant 1722
SqlExpr.SSelectStar 1722
SqlExpr.SSelectStar constant 1722
SqlExpr.SSelectStarFrom 1723
SqlExpr.SSelectStarFrom constant 1723
SqlExpr.SUB_TYPE_MEMO 1723
SqlExpr.SUB_TYPE_MEMO constant 1723
SqlExpr.SWhere 1723
SqlExpr.SWhere constant 1723
SqlExpr.TConnectionData 1515
SqlExpr.TConnectionData.AddProperties 1517
SqlExpr.TConnectionData.BeginUpdate 1518
SqlExpr.TConnectionData.ConnectionName 1519
SqlExpr.TConnectionData.Create 1516
SqlExpr.TConnectionData.DelegateConnection 1519
SqlExpr.TConnectionData.Destroy 1517
SqlExpr.TConnectionData.DriverName 1520
SqlExpr.TConnectionData.EndUpdate 1518
SqlExpr.TConnectionData.GetFullParams 1518
SqlExpr.TConnectionData.IsModified 1520
SqlExpr.TConnectionData.Properties 1520
SqlExpr.TConnectionData.RefreshProperties 1518
SqlExpr.TConnectionData.ReloadProperties 1518
SqlExpr.TConnectionData.UpdateProperties 1519
SqlExpr.TConnectionState 1700
SqlExpr.TConnectionState enumeration 1700
SqlExpr.TConnectionUserType 1700
SqlExpr.TConnectionUserType enumeration 1700
SqlExpr.TCustomSQLDataSet 1520
SqlExpr.TCustomSQLDataSet.Active 1552
SqlExpr.TCustomSQLDataSet.AddFieldDesc 1527
SqlExpr.TCustomSQLDataSet.AddIndexDefs 1527
SqlExpr.TCustomSQLDataSet.AfterClose 1568
SqlExpr.TCustomSQLDataSet.AfterOpen 1568
SqlExpr.TCustomSQLDataSet.AfterRefresh 1568
SqlExpr.TCustomSQLDataSet.AfterScroll 1568
SqlExpr.TCustomSQLDataSet.BeforeClose 1569
SqlExpr.TCustomSQLDataSet.BeforeOpen 1569
SqlExpr.TCustomSQLDataSet.BeforeRefresh 1569
SqlExpr.TCustomSQLDataSet.BeforeScroll 1569
SqlExpr.TCustomSQLDataSet.BlobBuffer 1554
SqlExpr.TCustomSQLDataSet.CheckPrepareError 1527
SqlExpr.TCustomSQLDataSet.ClearIndexDefs 1528
SqlExpr.TCustomSQLDataSet.CloseCursor 1528
SqlExpr.TCustomSQLDataSet.CloseStatement 1528
SqlExpr.TCustomSQLDataSet.CommandText 1554
SqlExpr.TCustomSQLDataSet.CommandType 1555
SqlExpr.TCustomSQLDataSet.Create 1524
SqlExpr.TCustomSQLDataSet.CreateBlobStream 1528
SqlExpr.TCustomSQLDataSet.CurrentBlobSize 1556
SqlExpr.TCustomSQLDataSet.DataLink 1556
SqlExpr.TCustomSQLDataSet.DataSource 1557
SqlExpr.TCustomSQLDataSet.DbxCommandType 1557
SqlExpr.TCustomSQLDataSet.DefineProperties 1529
SqlExpr.TCustomSQLDataSet.DesignerData 1557

SqlExpr.TCustomSQLDataSet.Destroy 1525
SqlExpr.TCustomSQLDataSet.ExecSQL 1529
SqlExpr.TCustomSQLDataSet.ExecuteStatement 1529
SqlExpr.TCustomSQLDataSet.FreeBuffers 1530
SqlExpr.TCustomSQLDataSet.FreeCommand 1530
SqlExpr.TCustomSQLDataSet.FreeReader 1530
SqlExpr.TCustomSQLDataSet.GetBlobFieldData 1530
SqlExpr.TCustomSQLDataSet.GetCanModify 1531
SqlExpr.TCustomSQLDataSet.GetCommandNames 1531
SqlExpr.TCustomSQLDataSet.GetDataSource 1531
SqlExpr.TCustomSQLDataSet.GetDetailLinkFields 1531
SqlExpr.TCustomSQLDataSet.GetFieldData 1532
SqlExpr.TCustomSQLDataSet.GetKeyFieldNames 1533
SqlExpr.TCustomSQLDataSet.GetMetadata 1558
SqlExpr.TCustomSQLDataSet.GetObjectTypeNames 1533
SqlExpr.TCustomSQLDataSet.GetOutputParams 1534
SqlExpr.TCustomSQLDataSet.GetQuoteChar 1534
SqlExpr.TCustomSQLDataSet.GetRecord 1534
SqlExpr.TCustomSQLDataSet.GetRecordCount 1534
SqlExpr.TCustomSQLDataSet.GetSortFieldNames 1535
SqlExpr.TCustomSQLDataSet.IndexDefs 1558
SqlExpr.TCustomSQLDataSet.InitRecord 1535
SqlExpr.TCustomSQLDataSet.InternalClose 1535
SqlExpr.TCustomSQLDataSet.InternalConnection 1559
SqlExpr.TCustomSQLDataSet.InternalFreeCommand 1535
SqlExpr.TCustomSQLDataSet.InternalHandleException 1535
SqlExpr.TCustomSQLDataSet.InternalInitFieldDefs 1536
SqlExpr.TCustomSQLDataSet.InternalOpen 1536
SqlExpr.TCustomSQLDataSet.InternalRefresh 1536
SqlExpr.TCustomSQLDataSet.IsCursorOpen 1536
SqlExpr.TCustomSQLDataSet.IsSequenced 1536
SqlExpr.TCustomSQLDataSet.Loaded 1537
SqlExpr.TCustomSQLDataSet.Locate 1538
SqlExpr.TCustomSQLDataSet.LocateRecord 1538
SqlExpr.TCustomSQLDataSet.Lookup 1538
SqlExpr.TCustomSQLDataSet.MaxBlobSize 1559
SqlExpr.TCustomSQLDataSet.NativeCommand 1560
SqlExpr.TCustomSQLDataSet.NumericMapping 1560
SqlExpr.TCustomSQLDataSet.ObjectView 1560
SqlExpr.TCustomSQLDataSet.OnCalcFields 1569
SqlExpr.TCustomSQLDataSet.OpenCursor 1539
SqlExpr.TCustomSQLDataSet.OpenSchema 1539
SqlExpr.TCustomSQLDataSet.ParamByName 1539
SqlExpr.TCustomSQLDataSet.ParamCheck 1561
SqlExpr.TCustomSQLDataSet.ParamCount 1562
SqlExpr.TCustomSQLDataSet.Params 1562
SqlExpr.TCustomSQLDataSet.ParseDeleteSql 1569
SqlExpr.TCustomSQLDataSet.ParseInsertSql 1570
SqlExpr.TCustomSQLDataSet.ParseSelectSql 1570
SqlExpr.TCustomSQLDataSet.ParseUpdateSql 1570
SqlExpr.TCustomSQLDataSet.Prepared 1563
SqlExpr.TCustomSQLDataSet.PrepareStatement 1540
SqlExpr.TCustomSQLDataSet.ProcParams 1564
SqlExpr.TCustomSQLDataSet.PropertyChanged 1540
SqlExpr.TCustomSQLDataSet.PSEndTransaction 1540
SqlExpr.TCustomSQLDataSet.PSExecute 1541
SqlExpr.TCustomSQLDataSet.PSExecuteStatement 1541
SqlExpr.TCustomSQLDataSet.PSGetAttributes 1541
SqlExpr.TCustomSQLDataSet.PSGetCommandText 1542
SqlExpr.TCustomSQLDataSet.PSGetCommandType 1542
SqlExpr.TCustomSQLDataSet.PSGetDefaultOrder 1542
SqlExpr.TCustomSQLDataSet.PSGetIndexDefs 1543
SqlExpr.TCustomSQLDataSet.PSGetKeyFieldsW 1543
SqlExpr.TCustomSQLDataSet.PSGetParams 1544
SqlExpr.TCustomSQLDataSet.PSGetQuoteCharW 1544
SqlExpr.TCustomSQLDataSet.PSGetTableNameW 1544
SqlExpr.TCustomSQLDataSet.PSGetUpdateException 1544
SqlExpr.TCustomSQLDataSet.PSInTransaction 1545
SqlExpr.TCustomSQLDataSet.PSIsSQLBased 1545
SqlExpr.TCustomSQLDataSet.PSIsSQLSupported 1546
SqlExpr.TCustomSQLDataSet.PSReset 1546
SqlExpr.TCustomSQLDataSet.PSSetCommandText 1546
SqlExpr.TCustomSQLDataSet.PSSetParams 1547
SqlExpr.TCustomSQLDataSet.PSStartTransaction 1547
SqlExpr.TCustomSQLDataSet.PSUpdateRecord 1548
SqlExpr.TCustomSQLDataSet.RecordCount 1564
SqlExpr.TCustomSQLDataSet.RowsAffected 1564
SqlExpr.TCustomSQLDataSet.SchemaInfo 1565

SqlExpr.TCustomSQLDataSet.SchemaName 1565
SqlExpr.TCustomSQLDataSet.SetBufListSize 1548
SqlExpr.TCustomSQLDataSet.SetCommandText 1549
SqlExpr.TCustomSQLDataSet.SetFCommandText 1549
SqlExpr.TCustomSQLDataSet.SetFieldData 1549
SqlExpr.TCustomSQLDataSet.SetMaxBlobSize 1549
SqlExpr.TCustomSQLDataSet.SetParamsFromCursor 1549
SqlExpr.TCustomSQLDataSet.SetSchemaInfo 1550
SqlExpr.TCustomSQLDataSet.SetSortFieldNames 1551
SqlExpr.TCustomSQLDataSet.SortFieldNames 1566
SqlExpr.TCustomSQLDataSet.SQLConnection 1566
SqlExpr.TCustomSQLDataSet.TransactionLevel 1567
SqlExpr.TCustomSQLDataSet.UpdateIndexDefs 1551
SqlExpr.TErrorMessageSize 1724
SqlExpr.TErrorMessageSize constant 1724
SqlExpr.TFieldDescList 1707
SqlExpr.TFieldDescList type 1707
SqlExpr.TFieldList 1707
SqlExpr.TFieldList type 1707
SqlExpr.TFLDDesc 1570
SqlExpr.TFLDDesc.bCalcField 1575
SqlExpr.TFLDDesc.efldrRights 1575
SqlExpr.TFLDDesc.efldvVchk 1575
SqlExpr.TFLDDesc.FCalcField 1572
SqlExpr.TFLDDesc.FFldNum 1572
SqlExpr.TFLDDesc.FFLDRights 1572
SqlExpr.TFLDDesc.FFldType 1572
SqlExpr.TFLDDesc.FFLDVchk 1573
SqlExpr.TFLDDesc.FLen 1573
SqlExpr.TFLDDesc.FName 1573
SqlExpr.TFLDDesc.FNullOffset 1573
SqlExpr.TFLDDesc.FOffset 1573
SqlExpr.TFLDDesc.FSubType 1574
SqlExpr.TFLDDesc.FUnits1 1574
SqlExpr.TFLDDesc.FUnits2 1574
SqlExpr.TFLDDesc.iFldNum 1575
SqlExpr.TFLDDesc.iFldType 1576
SqlExpr.TFLDDesc.iLen 1576
SqlExpr.TFLDDesc.iNullOffset 1576
SqlExpr.TFLDDesc.iOffset 1576
SqlExpr.TFLDDesc.iSubType 1576
SqlExpr.TFLDDesc.iUnits1 1577
SqlExpr.TFLDDesc.iUnits2 1577
SqlExpr.TFLDDesc.szName 1577
SqlExpr.TGetDriverFunc 1708
SqlExpr.TGetDriverFunc type 1708
SqlExpr.TLocale 1708
SqlExpr.TLocale type 1708
SqlExpr.TLocaleCode 1708
SqlExpr.TLocaleCode type 1708
SqlExpr.TParseInsertSqlEvent 1709
SqlExpr.TParseInsertSqlEvent type 1709
SqlExpr.TParseSqlEvent 1709
SqlExpr.TParseSqlEvent type 1709
SqlExpr.TSchemaType 1701
SqlExpr.TSchemaType enumeration 1701
SqlExpr.TSQLBlobStream 1577
SqlExpr.TSQLBlobStream.Create 1578
SqlExpr.TSQLBlobStream.Destroy 1579
SqlExpr.TSQLBlobStream.ReadBlobData 1580
SqlExpr.TSQLConnection 1580
SqlExpr.TSQLConnection.ActiveStatements 1607
SqlExpr.TSQLConnection.AddConnectNotification 1585
SqlExpr.TSQLConnection.AfterConnect 1626
SqlExpr.TSQLConnection.AfterDisconnect 1626
SqlExpr.TSQLConnection.AutoClone 1608
SqlExpr.TSQLConnection.BeforeConnect 1626
SqlExpr.TSQLConnection.BeforeDisconnect 1627
SqlExpr.TSQLConnection.BeginTransaction 1586
SqlExpr.TSQLConnection.CheckConnection 1586
SqlExpr.TSQLConnection.CheckDisconnect 1587
SqlExpr.TSQLConnection.CloneConnection 1587
SqlExpr.TSQLConnection.CloseDataSets 1587
SqlExpr.TSQLConnection.Commit 1588
SqlExpr.TSQLConnection.CommitFreeAndNil 1588
SqlExpr.TSQLConnection.Connected 1609
SqlExpr.TSQLConnection.ConnectionData 1609
SqlExpr.TSQLConnection.ConnectionName 1610

SqlExpr.TSQLConnection.ConnectionOptions 1589
SqlExpr.TSQLConnection.ConnectionRegistryFile 1610
SqlExpr.TSQLConnection.ConnectionState 1611
SqlExpr.TSQLConnection.Create 1583
SqlExpr.TSQLConnection.DataSets 1611
SqlExpr.TSQLConnection.DBXConnection 1611
SqlExpr.TSQLConnection.Destroy 1584
SqlExpr.TSQLConnection.DoConnect 1589
SqlExpr.TSQLConnection.DoDisconnect 1589
SqlExpr.TSQLConnection.DriverName 1612
SqlExpr.TSQLConnection.Execute 1590
SqlExpr.TSQLConnection.ExecuteDirect 1590
SqlExpr.TSQLConnection.GetCommandTypes 1591
SqlExpr.TSQLConnection.GetConnected 1591
SqlExpr.TSQLConnection.GetDataSet 1591
SqlExpr.TSQLConnection.GetDefaultSchemaName 1592
SqlExpr.TSQLConnection.GetDriverFunc 1613
SqlExpr.TSQLConnection.GetFieldNames 1592
SqlExpr.TSQLConnection.GetIndexNames 1593
SqlExpr.TSQLConnection.GetLoginUsername 1594
SqlExpr.TSQLConnection.GetPackageNames 1594
SqlExpr.TSQLConnection.GetProcedureNames 1595
SqlExpr.TSQLConnection.GetProcedureParams 1597
SqlExpr.TSQLConnection.GetSchemaNames 1598
SqlExpr.TSQLConnection.GetServerMethodNames 1598
SqlExpr.TSQLConnection.GetTableNames 1599
SqlExpr.TSQLConnection.HasTransaction 1599
SqlExpr.TSQLConnection.InTransaction 1613
SqlExpr.TSQLConnection.KeepConnection 1614
SqlExpr.TSQLConnection.LibraryName 1614
SqlExpr.TSQLConnection.Loaded 1600
SqlExpr.TSQLConnection.LoadParamsFromIniFile 1600
SqlExpr.TSQLConnection.LoadParamsOnConnect 1615
SqlExpr.TSQLConnection.LocaleCode 1615
SqlExpr.TSQLConnection.LoginPrompt 1616
SqlExpr.TSQLConnection.MaxStmtsPerConn 1620
SqlExpr.TSQLConnection.MetaData 1620
SqlExpr.TSQLConnection.MultipleTransactionsSupported 1621
SqlExpr.TSQLConnection.Notification 1601
SqlExpr.TSQLConnection.OnLogin 1627
SqlExpr.TSQLConnection.OpenSchema 1601, 1602
SqlExpr.TSQLConnection.Params 1621
SqlExpr.TSQLConnection.ParamsLoaded 1622
SqlExpr.TSQLConnection.QuoteChar 1623
SqlExpr.TSQLConnection.RegisterClient 1602
SqlExpr.TSQLConnection.RemoveConnectNotification 1602
SqlExpr.TSQLConnection.Rollback 1603
SqlExpr.TSQLConnection.RollbackFreeAndNil 1603
SqlExpr.TSQLConnection.RollbackIncompleteFreeAndNil 1604
SqlExpr.TSQLConnection.SetTraceEvent 1605
SqlExpr.TSQLConnection.SQLDllHandle 1623
SqlExpr.TSQLConnection.SQLError 1605
SqlExpr.TSQLConnection.SQLHourGlass 1623
SqlExpr.TSQLConnection.StartTransaction 1605
SqlExpr.TSQLConnection.TableScope 1623
SqlExpr.TSQLConnection.TraceCallbackEvent 1628
SqlExpr.TSQLConnection.TransactionsSupported 1624
SqlExpr.TSQLConnection.UnRegisterClient 1606
SqlExpr.TSQLConnection.VendorLib 1625
SqlExpr.TSQLConnectionLoginEvent 1709
SqlExpr.TSQLConnectionLoginEvent type 1709
SqlExpr.TSQLDataLink 1628
SqlExpr.TSQLDataLink.ActiveChanged 1630
SqlExpr.TSQLDataLink.CheckBrowseMode 1630
SqlExpr.TSQLDataLink.Create 1629
SqlExpr.TSQLDataLink.GetDetailDataSet 1631
SqlExpr.TSQLDataLink.RecordChanged 1631
SqlExpr.TSQLDataSet 1631
SqlExpr.TSQLDataSet.CommandText 1637
SqlExpr.TSQLDataSet.CommandType 1638
SqlExpr.TSQLDataSet.Create 1635
SqlExpr.TSQLDataSet.DataSource 1639
SqlExpr.TSQLDataSet.DbxCommandType 1639
SqlExpr.TSQLDataSet.ExecSQL 1636
SqlExpr.TSQLDataSet.MaxBlobSize 1640
SqlExpr.TSQLDataSet.ParamCheck 1641
SqlExpr.TSQLDataSet.Params 1641
SqlExpr.TSQLDataSet.SortFieldNames 1642

SqlExpr.TSQLDataSet.SQLConnection 1642
SqlExpr.TSQLExceptionType 1702
SqlExpr.TSQLExceptionType enumeration 1702
SqlExpr.TSQLMonitor 1643
SqlExpr.TSQLMonitor.Active 1649
SqlExpr.TSQLMonitor.AutoSave 1650
SqlExpr.TSQLMonitor.Create 1645
SqlExpr.TSQLMonitor.Destroy 1645
SqlExpr.TSQLMonitor.FileName 1650
SqlExpr.TSQLMonitor.InvokeCallBack 1646
SqlExpr.TSQLMonitor.LoadFromFile 1646
SqlExpr.TSQLMonitor.MaxTraceCount 1651
SqlExpr.TSQLMonitor.OnLogTrace 1653
SqlExpr.TSQLMonitor.OnTrace 1653
SqlExpr.TSQLMonitor.SaveToFile 1646
SqlExpr.TSQLMonitor.SetActive 1647
SqlExpr.TSQLMonitor.SetFileName 1647
SqlExpr.TSQLMonitor.SetSQLConnection 1647
SqlExpr.TSQLMonitor.SetStreamedActive 1648
SqlExpr.TSQLMonitor.SetTraceList 1648
SqlExpr.TSQLMonitor.SQLConnection 1651
SqlExpr.TSQLMonitor.SwitchConnection 1648
SqlExpr.TSQLMonitor.Trace 1648
SqlExpr.TSQLMonitor.TraceCount 1651
SqlExpr.TSQLMonitor.TraceList 1652
SqlExpr.TSQLMonitor.UpdateTraceCallBack 1648
SqlExpr.TSQLQuery 1654
SqlExpr.TSQLQuery.Create 1657
SqlExpr.TSQLQuery.DataSource 1660
SqlExpr.TSQLQuery.Destroy 1658
SqlExpr.TSQLQuery.ExecSQL 1658
SqlExpr.TSQLQuery.GetMetadata 1660
SqlExpr.TSQLQuery.MaxBlobSize 1661
SqlExpr.TSQLQuery.ParamCheck 1661
SqlExpr.TSQLQuery.Params 1662
SqlExpr.TSQLQuery.PrepareStatement 1659
SqlExpr.TSQLQuery.RowsAffected 1663
SqlExpr.TSQLQuery.SQL 1663
SqlExpr.TSQLQuery.SQLConnection 1664
SqlExpr.TSQLQuery.Text 1665
SqlExpr.TSQLSchemaInfo 1703
SqlExpr.TSQLSchemaInfo record 1703
SqlExpr.TSqlServerMethod 1665
SqlExpr.TSqlServerMethod.Create 1668
SqlExpr.TSqlServerMethod.ExecuteMethod 1669
SqlExpr.TSqlServerMethod.Params 1669
SqlExpr.TSqlServerMethod.ServerMethodName 1670
SqlExpr.TSqlServerMethod.SQLConnection 1670
SqlExpr.TSQLStoredProc 1671
SqlExpr.TSQLStoredProc.Create 1674
SqlExpr.TSQLStoredProc.ExecProc 1675
SqlExpr.TSQLStoredProc.MaxBlobSize 1677
SqlExpr.TSQLStoredProc.NextRecordSet 1676
SqlExpr.TSQLStoredProc.PackageName 1677
SqlExpr.TSQLStoredProc.ParamCheck 1678
SqlExpr.TSQLStoredProc.Params 1678
SqlExpr.TSQLStoredProc.PrepareStatement 1676
SqlExpr.TSQLStoredProc.SQLConnection 1679
SqlExpr.TSQLStoredProc.StoredProcName 1680
SqlExpr.TSQLTable 1680
SqlExpr.TSQLTable.Active 1687
SqlExpr.TSQLTable.Create 1684
SqlExpr.TSQLTable.DeleteRecords 1685
SqlExpr.TSQLTable.Destroy 1685
SqlExpr.TSQLTable.GetIndexNames 1686
SqlExpr.TSQLTable.IndexFieldCount 1689
SqlExpr.TSQLTable.IndexFieldNames 1689
SqlExpr.TSQLTable.IndexFields 1690
SqlExpr.TSQLTable.IndexName 1690
SqlExpr.TSQLTable.MasterFields 1691
SqlExpr.TSQLTable.MasterLink 1692
SqlExpr.TSQLTable.MasterSource 1692
SqlExpr.TSQLTable.MaxBlobSize 1692
SqlExpr.TSQLTable.OpenCursor 1686
SqlExpr.TSQLTable.PrepareStatement 1686
SqlExpr.TSQLTable.SetIndexField 1687
SqlExpr.TSQLTable.SQLConnection 1693
SqlExpr.TSQLTable.TableName 1693

SqlExpr.TTableScope 1703
SqlExpr.TTableScope enumeration 1703
SqlExpr.TTableScopes 1710
SqlExpr.TTableScopes type 1710
SqlExpr.TTraceEvent 1710
SqlExpr.TTraceEvent type 1710
SqlExpr.TTraceLogEvent 1710
SqlExpr.TTraceLogEvent type 1710
SqlExpr.TTransactionDesc 1704
SqlExpr.TTransactionDesc record 1704
SqlExpr.TTransactionItem 1694
SqlExpr.TTransactionItem.FNext 1695
SqlExpr.TTransactionItem.FTransaction 1695
SqlExpr.TTransactionItem.FTransactionDesc 1695
SqlExpr.TTransIsolationLevel 1704
SqlExpr.TTransIsolationLevel enumeration 1704
SQLHourGlass
    TSQLConnection class 1623
SqlTimSt 1725
SqlTimSt namespace 1725
    Classes 1727
    Constants 1740
    Functions 1729
    Structs, Records, Enums 1737
    Types 1739
SqlTimSt.CheckSqlTimeStamp 1729
SqlTimSt.CheckSqlTimeStamp function 1729
SqlTimSt.DateTimeToSQLTimeStamp 1730
SqlTimSt.DateTimeToSQLTimeStamp function 1730
SqlTimSt.LocalToUTC 1730
SqlTimSt.LocalToUTC function 1730
SqlTimSt.NullSQLTimeStamp 1740
SqlTimSt.NullSQLTimeStamp constant 1740
SqlTimSt.PSQLTimeStamp 1739
SqlTimSt.PSQLTimeStamp type 1739
SqlTimSt.SQLDayOfWeek 1731
SqlTimSt.SQLDayOfWeek function 1731
SqlTimSt.SQLTimeStampToDateTime 1731
SqlTimSt.SQLTimeStampToDateTime function 1731
SqlTimSt.SQLTimeStampToStr 1732
SqlTimSt.SQLTimeStampToStr function 1732
SqlTimSt.StrToSQLTimeStamp 1732
SqlTimSt.StrToSQLTimeStamp function 1732
SqlTimSt.TryStrToSQLTimeStamp 1733
SqlTimSt.TryStrToSQLTimeStamp function 1733
SqlTimSt.TSQLTimeStamp 1737
SqlTimSt.TSQLTimeStamp record 1737
SqlTimSt.TTimeZone 1727
SqlTimSt.TTimeZone.FInfo 1728
SqlTimSt.TTimeZone.FUsesDayligtSavings 1728
SqlTimSt.TTimeZone.GetTimeZone 1728
SqlTimSt.UTCToLocal 1733
SqlTimSt.UTCToLocal function 1733
SqlTimSt.VarIsSQLTimeStamp 1734
SqlTimSt.VarIsSQLTimeStamp function 1734
SqlTimSt.VarSQLTimeStamp 1734
SqlTimSt.VarSQLTimeStamp function 1734
SqlTimSt.VarSQLTimeStampCreate 1735
SqlTimSt.VarSQLTimeStampCreate function 1735
SqlTimSt.VarToSQLTimeStamp 1736
SqlTimSt.VarToSQLTimeStamp function 1736
srBottom enumeration member 4662
srcText
    IDOMParseError interface 7590
SrcText
    EDOMParseError class 7513
srLeft enumeration member 4662
srRight enumeration member 4662
srTop enumeration member 4662
ssBoth enumeration member 3005
ssHorizontal enumeration member 3005
ssNone enumeration member 3005
ssStarted enumeration member 396
ssTerminated enumeration member 396
ssTimeOut enumeration member 396
ssVertical enumeration member 3005
StackInfo
    Exception class 3966

StackTrace
Exception class 3966
Standalone
IDOMXMLProlog interface 7607
StandAlone
IXMLDocument interface 7668
TXMLDocument class 7391
Start
TSocketService class 184
StartButtons
TAdapterFormLayout class 6415
StartConnect
TServerClientThread class 102
StartFields
TAdapterFormLayout class 6416
StartIterator
IIteratorSupport interface 1056
TAdapterHiddenFieldsList class 5307
TAdapterSubComponent class 5418
TBaseAdapterAction class 5473
TBaseAdapterErrorsList class 5486
TCustomAdapter class 5567
TSiteModuleHelper class 6571
StartMargin
TTabSet class 4504
StartRow
TAdapterFormLayout class 6416
StartSession
TSessions class 393
StartTable
TAdapterFormLayout class 6416
StartTime
IWebSessionAttributes interface 1107
TSessionItem class 382
TWebSessionImpl class 6771
StartTransaction
TSQLConnection class 1605
StartType
TService class 3083
StartValueIterator
IWebGetFieldValues interface 1095
TBaseAdapterMultiValueField class 5497
State
TCheckBox class 2147
TCustomCheckBox class 2282
StatPage
TSocketForm class 170
Status
TService class 3083
StatusBar
IWebBrowser2Disp interface 774
IWebBrowserApp interface 784
IWebBrowserAppDisp interface 793
TInternetExplorer class 456
TWebBrowser class 541
StatusCode
ESOAPHTTPException class 1275
StatusFlags
TSessionItem class 382
StatusText
IWebBrowser2Disp interface 774
IWebBrowserApp interface 784
IWebBrowserAppDisp interface 793
TInternetExplorer class 456
TWebBrowser class 541
StatusTextChange
DWebBrowserEvents interface 625
DWebBrowserEvents2 interface 633
stAuto enumeration member 3104
stBoot enumeration member 3104
stColumns enumeration member 1701
StdActnMenus 1741
StdActnMenus namespace 1741
Classes 1742
Functions 1760
StdActnMenus.RegisterStandardMenus 1760
StdActnMenus.RegisterStandardMenus function 1760
StdActnMenus.TStandardAddRemoveItem 1742

StdActnMenus.TStandardAddRemoveItem.DrawBackground 1743
StdActnMenus.TStandardAddRemoveItem.DrawGlyph 1743
StdActnMenus.TStandardButtonControl 1744
StdActnMenus.TStandardButtonControl.DrawFrame 1744
StdActnMenus.TStandardButtonControl.DrawGlyph 1745
StdActnMenus.TStandardButtonControl.DrawText 1745
StdActnMenus.TStandardCustomizePopup 1745
StdActnMenus.TStandardCustomizePopup.CanAutoSize 1746
StdActnMenus.TStandardCustomizePopup.NCPaint 1746
StdActnMenus.TStandardCustomizePopup.PositionPopup 1747
StdActnMenus.TStandardDropDownButton 1747
StdActnMenus.TStandardDropDownButton.DrawFrame 1748
StdActnMenus.TStandardDropDownButton.GetPopupClass 1748
StdActnMenus.TStandardMenuButton 1748
StdActnMenus.TStandardMenuButton.DrawBackground 1749
StdActnMenus.TStandardMenuButton.DrawDesignFocus 1749
StdActnMenus.TStandardMenuButton.DrawFrame 1750
StdActnMenus.TStandardMenuButton.DrawText 1750
StdActnMenus.TStandardMenuExpandBtn 1750
StdActnMenus.TStandardMenuExpandBtn.DrawBackground 1751
StdActnMenus.TStandardMenuExpandBtn.DrawFrame 1751
StdActnMenus.TStandardMenuItem 1751
StdActnMenus.TStandardMenuItem.CMTextchanged 1753
StdActnMenus.TStandardMenuItem.DrawBackground 1753
StdActnMenus.TStandardMenuItem.DrawDesignFocus 1753
StdActnMenus.TStandardMenuItem.DrawEdge 1754
StdActnMenus.TStandardMenuItem.DrawGlyph 1754
StdActnMenus.TStandardMenuItem.DrawGlyphFrame 1754
StdActnMenus.TStandardMenuItem.DrawLargeGlyph 1755
StdActnMenus.TStandardMenuItem.DrawSeparator 1755
StdActnMenus.TStandardMenuItem.DrawText 1755
StdActnMenus.TStandardMenuPopup 1756
StdActnMenus.TStandardMenuPopup.CanAutoSize 1757
StdActnMenus.TStandardMenuPopup.GetExpandBtnClass 1757
StdActnMenus.TStandardMenuPopup.NCPaint 1757
StdActnMenus.TStandardMenuPopup.PositionPopup 1758
StdActnMenus.TStandardToolScrollBtn 1758
StdActnMenus.TStandardToolScrollBtn.DrawFrame 1759
StdActns 1761
StdActns namespace 1761
Classes 1763
Structs, Records, Enums 1947
Types 1949
StdActns.TBrowseForFolder 1764
StdActns.TBrowseForFolder.BeforeExecute 1772
StdActns.TBrowseForFolder.BrowseOptions 1767
StdActns.TBrowseForFolder.Caption 1767
StdActns.TBrowseForFolder.DialogCaption 1768
StdActns.TBrowseForFolder.DoCallback 1765
StdActns.TBrowseForFolder.DoValidate 1766
StdActns.TBrowseForFolder.Enabled 1768
StdActns.TBrowseForFolder.ExecuteTarget 1766
StdActns.TBrowseForFolder.Folder 1768
StdActns.TBrowseForFolder.GetDialogClass 1766
StdActns.TBrowseForFolder.HelpContext 1768
StdActns.TBrowseForFolder.HelpKeyword 1769
StdActns.TBrowseForFolder.HelpType 1769
StdActns.TBrowseForFolder.Hint 1770
StdActns.TBrowseForFolder.ImageIndex 1770
StdActns.TBrowseForFolder.OnAccept 1772
StdActns.TBrowseForFolder.OnCallback 1773
StdActns.TBrowseForFolder.OnCancel 1773
StdActns.TBrowseForFolder.OnHint 1774
StdActns.TBrowseForFolder.OnValidate 1774
StdActns.TBrowseForFolder.RootDir 1770
StdActns.TBrowseForFolder.SecondaryShortCuts 1771
StdActns.TBrowseForFolder.ShortCut 1771
StdActns.TBrowseForFolder.Visible 1771
StdActns.TBrowseForFolderCallbackEvent 1949
StdActns.TBrowseForFolderCallbackEvent type 1949
StdActns.TBrowseForFolderOption 1947
StdActns.TBrowseForFolderOption enumeration 1947
StdActns.TBrowseOptions 1949
StdActns.TBrowseOptions type 1949
StdActns.TColorSelect 1774
StdActns.TColorSelect.BeforeExecute 1781

StdActns.TColorSelect.Caption 1776
StdActns.TColorSelect.Dialog 1777
StdActns.TColorSelect.Enabled 1777
StdActns.TColorSelect.GetDialogClass 1776
StdActns.TColorSelect.HelpContext 1777
StdActns.TColorSelect.HelpKeyword 1778
StdActns.TColorSelect.HelpType 1778
StdActns.TColorSelect.Hint 1779
StdActns.TColorSelect.ImageIndex 1779
StdActns.TColorSelect.OnAccept 1781
StdActns.TColorSelect.OnCancel 1782
StdActns.TColorSelect.OnHint 1782
StdActns.TColorSelect.OnUpdate 1782
StdActns.TColorSelect.SecondaryShortCuts 1780
StdActns.TColorSelect.ShortCut 1780
StdActns.TColorSelect.Visible 1780
StdActns.TCommonDialogAction 1784
StdActns.TCommonDialogAction.BeforeExecute 1790
StdActns.TCommonDialogAction.Create 1785
StdActns.TCommonDialogAction.Destroy 1786
StdActns.TCommonDialogAction.DoAccept 1787
StdActns.TCommonDialogAction.DoCancel 1787
StdActns.TCommonDialogAction.ExecuteResult 1789
StdActns.TCommonDialogAction.ExecuteTarget 1787
StdActns.TCommonDialogAction.FDialog 1786
StdActns.TCommonDialogAction.GetDialogClass 1788
StdActns.TCommonDialogAction.HandlesTarget 1788
StdActns.TCommonDialogAction.Notification 1789
StdActns.TCommonDialogAction.OnAccept 1790
StdActns.TCommonDialogAction.OnCancel 1791
StdActns.TCommonDialogAction.SetupDialog 1789
StdActns.TCommonDialogClass 1950
StdActns.TCommonDialogClass type 1950
StdActns.TEditAction 1791
StdActns.TEditAction.Control 1795
StdActns.TEditAction.Destroy 1792
StdActns.TEditAction.GetControl 1793
StdActns.TEditAction.HandlesTarget 1793
StdActns.TEditAction.Notification 1794
StdActns.TEditAction.UpdateTarget 1794
StdActns.TEditCopy 1795
StdActns.TEditCopy.ExecuteTarget 1796
StdActns.TEditCut 1797
StdActns.TEditCut.ExecuteTarget 1798
StdActns.TEditDelete 1799
StdActns.TEditDelete.ExecuteTarget 1800
StdActns.TEditDelete.UpdateTarget 1800
StdActns.TEditPaste 1801
StdActns.TEditPaste.ExecuteTarget 1802
StdActns.TEditPaste.UpdateTarget 1803
StdActns.TEditSelectAll 1803
StdActns.TEditSelectAll.ExecuteTarget 1804
StdActns.TEditSelectAll.UpdateTarget 1805
StdActns.TEditUndo 1805
StdActns.TEditUndo.ExecuteTarget 1807
StdActns.TEditUndo.UpdateTarget 1807
StdActns.TFileAction 1807
StdActns.TFileAction.FileName 1809
StdActns.TFileAction.GetDialog 1809
StdActns.TFileExit 1810
StdActns.TFileExit.Caption 1812
StdActns.TFileExit.Enabled 1812
StdActns.TFileExit.ExecuteTarget 1811
StdActns.TFileExit.HandlesTarget 1811
StdActns.TFileExit.HelpContext 1813
StdActns.TFileExit.HelpKeyword 1813
StdActns.TFileExit.HelpType 1814
StdActns.TFileExit.Hint 1814
StdActns.TFileExit.ImageIndex 1814
StdActns.TFileExit.OnHint 1816
StdActns.TFileExit.OnUpdate 1816
StdActns.TFileExit.SecondaryShortCuts 1815
StdActns.TFileExit.ShortCut 1815
StdActns.TFileExit.Visible 1815
StdActns.TFileOpen 1817
StdActns.TFileOpen.BeforeExecute 1826
StdActns.TFileOpen.Caption 1821
StdActns.TFileOpen.Create 1819

StdActns.TFileOpen.Dialog 1821
StdActns.TFileOpen.Enabled 1822
StdActns.TFileOpen.ExecuteTarget 1820
StdActns.TFileOpen.GetDialogClass 1820
StdActns.TFileOpen.HelpContext 1822
StdActns.TFileOpen.HelpKeyword 1823
StdActns.TFileOpen.HelpType 1823
StdActns.TFileOpen.Hint 1824
StdActns.TFileOpen.ImageIndex 1824
StdActns.TFileOpen.OnAccept 1826
StdActns.TFileOpen.OnCancel 1827
StdActns.TFileOpen.OnHint 1827
StdActns.TFileOpen.OnUpdate 1828
StdActns.TFileOpen.SecondaryShortCuts 1824
StdActns.TFileOpen.ShortCut 1825
StdActns.TFileOpen.UseDefaultApp 1825
StdActns.TFileOpen.Visible 1825
StdActns.TFileOpenWith 1829
StdActns.TFileOpenWith.AfterOpen 1834
StdActns.TFileOpenWith.BeforeExecute 1834
StdActns.TFileOpenWith.ExecuteTarget 1831
StdActns.TFileOpenWith.FileName 1832
StdActns.TFileOpenWith.HelpContext 1832
StdActns.TFileOpenWith.HelpKeyword 1833
StdActns.TFileOpenWith.HelpType 1833
StdActns.TFileOpenWith.OnUpdate 1835
StdActns.TFilePageSetup 1836
StdActns.TFilePageSetup.BeforeExecute 1842
StdActns.TFilePageSetup.Caption 1838
StdActns.TFilePageSetup.Dialog 1838
StdActns.TFilePageSetup.Enabled 1839
StdActns.TFilePageSetup.GetDialogClass 1837
StdActns.TFilePageSetup.HelpContext 1839
StdActns.TFilePageSetup.HelpKeyword 1840
StdActns.TFilePageSetup.HelpType 1840
StdActns.TFilePageSetup.Hint 1840
StdActns.TFilePageSetup.ImageIndex 1841
StdActns.TFilePageSetup.OnAccept 1843
StdActns.TFilePageSetup.OnCancel 1843
StdActns.TFilePageSetup.OnHint 1844
StdActns.TFilePageSetup.OnUpdate 1844
StdActns.TFilePageSetup.SecondaryShortCuts 1841
StdActns.TFilePageSetup.ShortCut 1842
StdActns.TFilePageSetup.Visible 1842
StdActns.TFilePrintSetup 1845
StdActns.TFilePrintSetup.BeforeExecute 1852
StdActns.TFilePrintSetup.Caption 1848
StdActns.TFilePrintSetup.Dialog 1848
StdActns.TFilePrintSetup.Enabled 1848
StdActns.TFilePrintSetup.GetDialogClass 1847
StdActns.TFilePrintSetup.HelpContext 1849
StdActns.TFilePrintSetup.HelpKeyword 1849
StdActns.TFilePrintSetup.HelpType 1850
StdActns.TFilePrintSetup.Hint 1850
StdActns.TFilePrintSetup.ImageIndex 1850
StdActns.TFilePrintSetup.OnAccept 1852
StdActns.TFilePrintSetup.OnCancel 1853
StdActns.TFilePrintSetup.OnHint 1853
StdActns.TFilePrintSetup.OnUpdate 1854
StdActns.TFilePrintSetup.SecondaryShortCuts 1851
StdActns.TFilePrintSetup.ShortCut 1851
StdActns.TFilePrintSetup.Visible 1851
StdActns.TFileSaveAs 1855
StdActns.TFileSaveAs.BeforeExecute 1861
StdActns.TFileSaveAs.Caption 1858
StdActns.TFileSaveAs.Create 1856
StdActns.TFileSaveAs.Dialog 1858
StdActns.TFileSaveAs.Enabled 1858
StdActns.TFileSaveAs.GetDialogClass 1857
StdActns.TFileSaveAs.HelpContext 1859
StdActns.TFileSaveAs.Hint 1859
StdActns.TFileSaveAs.ImageIndex 1860
StdActns.TFileSaveAs.OnAccept 1862
StdActns.TFileSaveAs.OnCancel 1862
StdActns.TFileSaveAs.OnHint 1863
StdActns.TFileSaveAs.OnUpdate 1863
StdActns.TFileSaveAs.SecondaryShortCuts 1860
StdActns.TFileSaveAs.ShortCut 1860

StdActns.TFileSaveAs.Visible 1861
StdActns.TFontEdit 1864
StdActns.TFontEdit.BeforeExecute 1871
StdActns.TFontEdit.Caption 1866
StdActns.TFontEdit.Dialog 1867
StdActns.TFontEdit.Enabled 1867
StdActns.TFontEdit.GetDialogClass 1866
StdActns.TFontEdit.HelpContext 1867
StdActns.TFontEdit.HelpKeyword 1868
StdActns.TFontEdit.HelpType 1868
StdActns.TFontEdit.Hint 1869
StdActns.TFontEdit.ImageIndex 1869
StdActns.TFontEdit.OnAccept 1871
StdActns.TFontEdit.OnCancel 1872
StdActns.TFontEdit.OnHint 1872
StdActns.TFontEdit.OnUpdate 1872
StdActns.TFontEdit.SecondaryShortCuts 1870
StdActns.TFontEdit.ShortCut 1870
StdActns.TFontEdit.Visible 1870
StdActns.THelpAction 1874
StdActns.THelpAction.Create 1875
StdActns.THelpAction.HandlesTarget 1875
StdActns.THelpAction.UpdateTarget 1875
StdActns.THelpContents 1876
StdActns.THelpContents.ExecuteTarget 1877
StdActns.THelpContextAction 1877
StdActns.THelpContextAction.ExecuteTarget 1878
StdActns.THelpContextAction.UpdateTarget 1879
StdActns.THelpOnHelp 1879
StdActns.THelpOnHelp.ExecuteTarget 1880
StdActns.THelpTopicSearch 1881
StdActns.THelpTopicSearch.ExecuteTarget 1881
StdActns.THintAction 1882
StdActns.THintAction.Create 1883
StdActns.THintAction.Hint 1884
StdActns.TPrintDlg 1884
StdActns.TPrintDlg.BeforeExecute 1891
StdActns.TPrintDlg.Caption 1886
StdActns.TPrintDlg.Dialog 1887
StdActns.TPrintDlg.Enabled 1887
StdActns.TPrintDlg.GetDialogClass 1886
StdActns.TPrintDlg.HelpContext 1888
StdActns.TPrintDlg.HelpKeyword 1888
StdActns.TPrintDlg.HelpType 1889
StdActns.TPrintDlg.Hint 1889
StdActns.TPrintDlg.ImageIndex 1889
StdActns.TPrintDlg.OnAccept 1891
StdActns.TPrintDlg.OnCancel 1892
StdActns.TPrintDlg.OnHint 1892
StdActns.TPrintDlg.OnUpdate 1893
StdActns.TPrintDlg.SecondaryShortCuts 1890
StdActns.TPrintDlg.ShortCut 1890
StdActns.TPrintDlg.Visible 1890
StdActns.TSearchAction 1894
StdActns.TSearchAction.Create 1895
StdActns.TSearchAction.Destroy 1896
StdActns.TSearchAction.ExecuteTarget 1897
StdActns.TSearchAction.FControl 1896
StdActns.TSearchAction.FFindFirst 1897
StdActns.TSearchAction.HandlesTarget 1898
StdActns.TSearchAction.Notification 1898
StdActns.TSearchAction.Search 1898
StdActns.TSearchAction.UpdateTarget 1899
StdActns.TSearchFind 1899
StdActns.TSearchFind.BeforeExecute 1906
StdActns.TSearchFind.Caption 1902
StdActns.TSearchFind.Dialog 1902
StdActns.TSearchFind.Enabled 1903
StdActns.TSearchFind.GetDialogClass 1901
StdActns.TSearchFind.HelpContext 1903
StdActns.TSearchFind.HelpKeyword 1903
StdActns.TSearchFind.HelpType 1904
StdActns.TSearchFind.Hint 1904
StdActns.TSearchFind.ImageIndex 1905
StdActns.TSearchFind.OnAccept 1907
StdActns.TSearchFind.OnCancel 1907
StdActns.TSearchFind.OnHint 1908
StdActns.TSearchFind.OnUpdate 1908

StdActns.TSearchFind.SecondaryShortCuts 1905
StdActns.TSearchFind.ShortCut 1905
StdActns.TSearchFind.Visible 1906
StdActns.TSearchFindFirst 1909
StdActns.TSearchFindFirst.Create 1911
StdActns.TSearchFindNext 1912
StdActns.TSearchFindNext.Caption 1915
StdActns.TSearchFindNext.Create 1913
StdActns.TSearchFindNext.Enabled 1916
StdActns.TSearchFindNext.ExecuteTarget 1914
StdActns.TSearchFindNext.HandlesTarget 1914
StdActns.TSearchFindNext.HelpContext 1916
StdActns.TSearchFindNext.HelpKeyword 1917
StdActns.TSearchFindNext.HelpType 1917
StdActns.TSearchFindNext.Hint 1917
StdActns.TSearchFindNext.ImageIndex 1918
StdActns.TSearchFindNext.OnHint 1920
StdActns.TSearchFindNext.OnUpdate 1920
StdActns.TSearchFindNext.SearchFind 1918
StdActns.TSearchFindNext.SecondaryShortCuts 1919
StdActns.TSearchFindNext.ShortCut 1919
StdActns.TSearchFindNext.UpdateTarget 1915
StdActns.TSearchFindNext.Visible 1919
StdActns.TSearchReplace 1921
StdActns.TSearchReplace.BeforeExecute 1929
StdActns.TSearchReplace.Caption 1925
StdActns.TSearchReplace.Dialog 1925
StdActns.TSearchReplace.Enabled 1925
StdActns.TSearchReplace.ExecuteTarget 1923
StdActns.TSearchReplace.GetDialogClass 1924
StdActns.TSearchReplace.HelpContext 1926
StdActns.TSearchReplace.HelpKeyword 1926
StdActns.TSearchReplace.HelpType 1927
StdActns.TSearchReplace.Hint 1927
StdActns.TSearchReplace.ImageIndex 1927
StdActns.TSearchReplace.OnAccept 1930
StdActns.TSearchReplace.OnCancel 1930
StdActns.TSearchReplace.OnHint 1930
StdActns.TSearchReplace.OnUpdate 1931
StdActns.TSearchReplace.SecondaryShortCuts 1928
StdActns.TSearchReplace.ShortCut 1928
StdActns.TSearchReplace.Visible 1929
StdActns.TWindowAction 1932
StdActns.TWindowAction.Create 1933
StdActns.TWindowAction.Form 1935
StdActns.TWindowAction.GetForm 1934
StdActns.TWindowAction.HandlesTarget 1934
StdActns.TWindowAction.Notification 1934
StdActns.TWindowAction.UpdateTarget 1935
StdActns.TWindowArrange 1936
StdActns.TWindowArrange.ExecuteTarget 1937
StdActns.TWindowCascade 1937
StdActns.TWindowCascade.ExecuteTarget 1938
StdActns.TWindowClose 1939
StdActns.TWindowClose.ExecuteTarget 1940
StdActns.TWindowClose.UpdateTarget 1941
StdActns.TWindowMinimizeAll 1941
StdActns.TWindowMinimizeAll.ExecuteTarget 1942
StdActns.TWindowTileHorizontal 1943
StdActns.TWindowTileHorizontal.ExecuteTarget 1944
StdActns.TWindowTileVertical 1944
StdActns.TWindowTileVertical.ExecuteTarget 1946
StdConvs 1953
StdConvs namespace 1953
Constants 2010
Functions 1958
Variables 1960
StdConvs.auAcres 1962
StdConvs.auAcres variable 1962
StdConvs.auAres 1963
StdConvs.auAres variable 1963
StdConvs.auCentares 1963
StdConvs.auCentares variable 1963
StdConvs.auHectares 1963
StdConvs.auHectares variable 1963
StdConvs.auSquareCentimeters 1964
StdConvs.auSquareCentimeters variable 1964
StdConvs.auSquareDecameters 1964

StdConvs.auSquareDecameters variable 1964
StdConvs.auSquareDecimeters 1964
StdConvs.auSquareDecimeters variable 1964
StdConvs.auSquareFeet 1965
StdConvs.auSquareFeet variable 1965
StdConvs.auSquareHectometers 1965
StdConvs.auSquareHectometers variable 1965
StdConvs.auSquareInches 1965
StdConvs.auSquareInches variable 1965
StdConvs.auSquareKilometers 1966
StdConvs.auSquareKilometers variable 1966
StdConvs.auSquareMeters 1966
StdConvs.auSquareMeters variable 1966
StdConvs.auSquareMiles 1966
StdConvs.auSquareMiles variable 1966
StdConvs.auSquareMillimeters 1967
StdConvs.auSquareMillimeters variable 1967
StdConvs.auSquareRods 1967
StdConvs.auSquareRods variable 1967
StdConvs.auSquareYards 1967
StdConvs.auSquareYards variable 1967
StdConvs.cbArea 1968
StdConvs.cbArea variable 1968
StdConvs.cbDistance 1968
StdConvs.cbDistance variable 1968
StdConvs.cbMass 1969
StdConvs.cbMass variable 1969
StdConvs.cbTemperature 1969
StdConvs.cbTemperature variable 1969
StdConvs.cbTime 1969
StdConvs.cbTime variable 1969
StdConvs.cbVolume 1970
StdConvs.cbVolume variable 1970
StdConvs.CelsiusToFahrenheit 1958
StdConvs.CelsiusToFahrenheit function 1958
StdConvs.CubicMetersPerAcreFoot 2011
StdConvs.CubicMetersPerAcreFoot constant 2011
StdConvs.CubicMetersPerAcreInch 2011
StdConvs.CubicMetersPerAcreInch constant 2011
StdConvs.CubicMetersPerCord 2012
StdConvs.CubicMetersPerCord constant 2012
StdConvs.CubicMetersPerCordFoot 2012
StdConvs.CubicMetersPerCordFoot constant 2012
StdConvs.CubicMetersPerCubicFoot 2012
StdConvs.CubicMetersPerCubicFoot constant 2012
StdConvs.CubicMetersPerCubicInch 2013
StdConvs.CubicMetersPerCubicInch constant 2013
StdConvs.CubicMetersPerCubicMile 2013
StdConvs.CubicMetersPerCubicMile constant 2013
StdConvs.CubicMetersPerCubicYard 2013
StdConvs.CubicMetersPerCubicYard constant 2013
StdConvs.CubicMetersPerUKBucket 2014
StdConvs.CubicMetersPerUKBucket constant 2014
StdConvs.CubicMetersPerUKBushel 2014
StdConvs.CubicMetersPerUKBushel constant 2014
StdConvs.CubicMetersPerUKGallon 2014
StdConvs.CubicMetersPerUKGallon constant 2014
StdConvs.CubicMetersPerUKGill 2015
StdConvs.CubicMetersPerUKGill constant 2015
StdConvs.CubicMetersPerUKOunce 2015
StdConvs.CubicMetersPerUKOunce constant 2015
StdConvs.CubicMetersPerUKPeck 2015
StdConvs.CubicMetersPerUKPeck constant 2015
StdConvs.CubicMetersPerUKPint 2016
StdConvs.CubicMetersPerUKPint constant 2016
StdConvs.CubicMetersPerUKPottle 2016
StdConvs.CubicMetersPerUKPottle constant 2016
StdConvs.CubicMetersPerUKQuart 2016
StdConvs.CubicMetersPerUKQuart constant 2016
StdConvs.CubicMetersPerUSDryBucket 2017
StdConvs.CubicMetersPerUSDryBucket constant 2017
StdConvs.CubicMetersPerUSDryBushel 2017
StdConvs.CubicMetersPerUSDryBushel constant 2017
StdConvs.CubicMetersPerUSDryGallon 2017
StdConvs.CubicMetersPerUSDryGallon constant 2017
StdConvs.CubicMetersPerUSDryPeck 2018
StdConvs.CubicMetersPerUSDryPeck constant 2018
StdConvs.CubicMetersPerUSDryPint 2018

StdConvs.CubicMetersPerUSDryPint constant 2018
StdConvs.CubicMetersPerUSDryQuart 2018
StdConvs.CubicMetersPerUSDryQuart constant 2018
StdConvs.CubicMetersPerUSFluidCup 2019
StdConvs.CubicMetersPerUSFluidCup constant 2019
StdConvs.CubicMetersPerUSFluidGallon 2019
StdConvs.CubicMetersPerUSFluidGallon constant 2019
StdConvs.CubicMetersPerUSFluidGill 2019
StdConvs.CubicMetersPerUSFluidGill constant 2019
StdConvs.CubicMetersPerUSFluidOunce 2020
StdConvs.CubicMetersPerUSFluidOunce constant 2020
StdConvs.CubicMetersPerUSFluidPint 2020
StdConvs.CubicMetersPerUSFluidPint constant 2020
StdConvs.CubicMetersPerUSFluidQuart 2020
StdConvs.CubicMetersPerUSFluidQuart constant 2020
StdConvs.CubicMetersPerUSFluidTablespoon 2021
StdConvs.CubicMetersPerUSFluidTablespoon constant 2021
StdConvs.CubicMetersPerUSFluidTeaspoon 2021
StdConvs.CubicMetersPerUSFluidTeaspoon constant 2021
StdConvs.duAngstroms 1971
StdConvs.duAngstroms variable 1971
StdConvs.duAstronomicalUnits 1971
StdConvs.duAstronomicalUnits variable 1971
StdConvs.duCentimeters 1972
StdConvs.duCentimeters variable 1972
StdConvs.duChains 1972
StdConvs.duChains variable 1972
StdConvs.duCubits 1972
StdConvs.duCubits variable 1972
StdConvs.duDecameters 1973
StdConvs.duDecameters variable 1973
StdConvs.duDecimeters 1973
StdConvs.duDecimeters variable 1973
StdConvs.duFathoms 1973
StdConvs.duFathoms variable 1973
StdConvs.duFeet 1974
StdConvs.duFeet variable 1974
StdConvs.duFurlongs 1974
StdConvs.duFurlongs variable 1974
StdConvs.duGigameters 1974
StdConvs.duGigameters variable 1974
StdConvs.duHands 1975
StdConvs.duHands variable 1975
StdConvs.duHectometers 1975
StdConvs.duHectometers variable 1975
StdConvs.duInches 1975
StdConvs.duInches variable 1975
StdConvs.duKilometers 1976
StdConvs.duKilometers variable 1976
StdConvs.duLightYears 1976
StdConvs.duLightYears variable 1976
StdConvs.duLinks 1976
StdConvs.duLinks variable 1976
StdConvs.duMegameters 1977
StdConvs.duMegameters variable 1977
StdConvs.duMeters 1977
StdConvs.duMeters variable 1977
StdConvs.duMicromicrons 1977
StdConvs.duMicromicrons variable 1977
StdConvs.duMicrons 1978
StdConvs.duMicrons variable 1978
StdConvs.duMiles 1978
StdConvs.duMiles variable 1978
StdConvs.duMillimeters 1978
StdConvs.duMillimeters variable 1978
StdConvs.duMillimicrons 1979
StdConvs.duMillimicrons variable 1979
StdConvs.duNauticalMiles 1979
StdConvs.duNauticalMiles variable 1979
StdConvs.duPaces 1979
StdConvs.duPaces variable 1979
StdConvs.duParsecs 1980
StdConvs.duParsecs variable 1980
StdConvs.duPicas 1980
StdConvs.duPicas variable 1980
StdConvs.duPoints 1980
StdConvs.duPoints variable 1980
StdConvs.duRods 1981

StdConvs.duRods variable 1981
StdConvs.duYards 1981
StdConvs.duYards variable 1981
StdConvs.FahrenheitToCelsius 1958
StdConvs.FahrenheitToCelsius function 1958
StdConvs.GramsPerDrams 2021
StdConvs.GramsPerDrams constant 2021
StdConvs.GramsPerGrains 2022
StdConvs.GramsPerGrains constant 2022
StdConvs.GramsPerLongTons 2022
StdConvs.GramsPerLongTons constant 2022
StdConvs.GramsPerOunces 2022
StdConvs.GramsPerOunces constant 2022
StdConvs.GramsPerPound 2023
StdConvs.GramsPerPound constant 2023
StdConvs.GramsPerStones 2023
StdConvs.GramsPerStones constant 2023
StdConvs.GramsPerTons 2023
StdConvs.GramsPerTons constant 2023
StdConvs.InitStdConvs 1959
StdConvs.InitStdConvs function 1959
StdConvs.MetersPerAstronomicalUnit 2024
StdConvs.MetersPerAstronomicalUnit constant 2024
StdConvs.MetersPerChain 2024
StdConvs.MetersPerChain constant 2024
StdConvs.MetersPerCubit 2024
StdConvs.MetersPerCubit constant 2024
StdConvs.MetersPerFathom 2025
StdConvs.MetersPerFathom constant 2025
StdConvs.MetersPerFoot 2025
StdConvs.MetersPerFoot constant 2025
StdConvs.MetersPerFurlong 2025
StdConvs.MetersPerFurlong constant 2025
StdConvs.MetersPerHand 2026
StdConvs.MetersPerHand constant 2026
StdConvs.MetersPerInch 2026
StdConvs.MetersPerInch constant 2026
StdConvs.MetersPerLightSecond 2026
StdConvs.MetersPerLightSecond constant 2026
StdConvs.MetersPerLightYear 2027
StdConvs.MetersPerLightYear constant 2027
StdConvs.MetersPerLink 2027
StdConvs.MetersPerLink constant 2027
StdConvs.MetersPerMile 2027
StdConvs.MetersPerMile constant 2027
StdConvs.MetersPerNauticalMiles 2028
StdConvs.MetersPerNauticalMiles constant 2028
StdConvs.MetersPerPace 2028
StdConvs.MetersPerPace constant 2028
StdConvs.MetersPerParsec 2028
StdConvs.MetersPerParsec constant 2028
StdConvs.MetersPerPica 2029
StdConvs.MetersPerPica constant 2029
StdConvs.MetersPerPoint 2029
StdConvs.MetersPerPoint constant 2029
StdConvs.MetersPerRod 2029
StdConvs.MetersPerRod constant 2029
StdConvs.MetersPerYard 2030
StdConvs.MetersPerYard constant 2030
StdConvs.muCentigrams 1981
StdConvs.muCentigrams variable 1981
StdConvs.muDecagrams 1982
StdConvs.muDecagrams variable 1982
StdConvs.muDecigrams 1982
StdConvs.muDecigrams variable 1982
StdConvs.muDrams 1982
StdConvs.muDrams variable 1982
StdConvs.muGrains 1983
StdConvs.muGrains variable 1983
StdConvs.muGrams 1983
StdConvs.muGrams variable 1983
StdConvs.muHectograms 1983
StdConvs.muHectograms variable 1983
StdConvs.muKilograms 1984
StdConvs.muKilograms variable 1984
StdConvs.muLongTons 1984
StdConvs.muLongTons variable 1984
StdConvs.muMetricTons 1984

StdConvs.muMetricTons variable 1984
StdConvs.muMicrograms 1985
StdConvs.muMicrograms variable 1985
StdConvs.muMilligrams 1985
StdConvs.muMilligrams variable 1985
StdConvs.muNanograms 1985
StdConvs.muNanograms variable 1985
StdConvs.muOunces 1986
StdConvs.muOunces variable 1986
StdConvs.muPounds 1986
StdConvs.muPounds variable 1986
StdConvs.muStones 1986
StdConvs.muStones variable 1986
StdConvs.muTons 1987
StdConvs.muTons variable 1987
StdConvs.SquareMetersPerAcre 2030
StdConvs.SquareMetersPerAcre constant 2030
StdConvs.SquareMetersPerSquareFoot 2030
StdConvs.SquareMetersPerSquareFoot constant 2030
StdConvs.SquareMetersPerSquareInch 2031
StdConvs.SquareMetersPerSquareInch constant 2031
StdConvs.SquareMetersPerSquareMile 2031
StdConvs.SquareMetersPerSquareMile constant 2031
StdConvs.SquareMetersPerSquareRod 2031
StdConvs.SquareMetersPerSquareRod constant 2031
StdConvs.SquareMetersPerSquareYard 2032
StdConvs.SquareMetersPerSquareYard constant 2032
StdConvs.tuCelsius 1987
StdConvs.tuCelsius variable 1987
StdConvs.tuCenturies 1987
StdConvs.tuCenturies variable 1987
StdConvs.tuDateTime 1988
StdConvs.tuDateTime variable 1988
StdConvs.tuDays 1988
StdConvs.tuDays variable 1988
StdConvs.tuDecades 1988
StdConvs.tuDecades variable 1988
StdConvs.tuFahrenheit 1989
StdConvs.tuFahrenheit variable 1989
StdConvs.tuFortnights 1989
StdConvs.tuFortnights variable 1989
StdConvs.tuHours 1989
StdConvs.tuHours variable 1989
StdConvs.tuJulianDate 1990
StdConvs.tuJulianDate variable 1990
StdConvs.tuKelvin 1990
StdConvs.tuKelvin variable 1990
StdConvs.tuMillennia 1990
StdConvs.tuMillennia variable 1990
StdConvs.tuMilliSeconds 1991
StdConvs.tuMilliSeconds variable 1991
StdConvs.tuMinutes 1991
StdConvs.tuMinutes variable 1991
StdConvs.tuModifiedJulianDate 1991
StdConvs.tuModifiedJulianDate variable 1991
StdConvs.tuMonths 1992
StdConvs.tuMonths variable 1992
StdConvs.tuRankine 1992
StdConvs.tuRankine variable 1992
StdConvs.tuReaumur 1992
StdConvs.tuReaumur variable 1992
StdConvs.tuSeconds 1993
StdConvs.tuSeconds variable 1993
StdConvs.tuWeeks 1993
StdConvs.tuWeeks variable 1993
StdConvs.tuYears 1993
StdConvs.tuYears variable 1993
StdConvs.vuAcreFeet 1994
StdConvs.vuAcreFeet variable 1994
StdConvs.vuAcreInches 1994
StdConvs.vuAcreInches variable 1994
StdConvs.vuCentiLiters 1994
StdConvs.vuCentiLiters variable 1994
StdConvs.vuCordFeet 1995
StdConvs.vuCordFeet variable 1995
StdConvs.vuCords 1995
StdConvs.vuCords variable 1995
StdConvs.vuCubicCentimeters 1995

StdConvs.vuCubicCentimeters variable 1995
StdConvs.vuCubicDecameters 1996
StdConvs.vuCubicDecameters variable 1996
StdConvs.vuCubicDecimeters 1996
StdConvs.vuCubicDecimeters variable 1996
StdConvs.vuCubicFeet 1996
StdConvs.vuCubicFeet variable 1996
StdConvs.vuCubicHectometers 1997
StdConvs.vuCubicHectometers variable 1997
StdConvs.vuCubicInches 1997
StdConvs.vuCubicInches variable 1997
StdConvs.vuCubicKilometers 1997
StdConvs.vuCubicKilometers variable 1997
StdConvs.vuCubicMeters 1998
StdConvs.vuCubicMeters variable 1998
StdConvs.vuCubicMiles 1998
StdConvs.vuCubicMiles variable 1998
StdConvs.vuCubicMillimeters 1998
StdConvs.vuCubicMillimeters variable 1998
StdConvs.vuCubicYards 1999
StdConvs.vuCubicYards variable 1999
StdConvs.vuDecaLiters 1999
StdConvs.vuDecaLiters variable 1999
StdConvs.vuDecasteres 1999
StdConvs.vuDecasteres variable 1999
StdConvs.vuDeciLiters 2000
StdConvs.vuDeciLiters variable 2000
StdConvs.vuDecisteres 2000
StdConvs.vuDecisteres variable 2000
StdConvs.vuDryBuckets 2000
StdConvs.vuDryBuckets variable 2000
StdConvs.vuDryBushels 2001
StdConvs.vuDryBushels variable 2001
StdConvs.vuDryGallons 2001
StdConvs.vuDryGallons variable 2001
StdConvs.vuDryPecks 2001
StdConvs.vuDryPecks variable 2001
StdConvs.vuDryPints 2002
StdConvs.vuDryPints variable 2002
StdConvs.vuDryQuarts 2002
StdConvs.vuDryQuarts variable 2002
StdConvs.vuFluidCups 2002
StdConvs.vuFluidCups variable 2002
StdConvs.vuFluidGallons 2003
StdConvs.vuFluidGallons variable 2003
StdConvs.vuFluidGills 2003
StdConvs.vuFluidGills variable 2003
StdConvs.vuFluidOunces 2003
StdConvs.vuFluidOunces variable 2003
StdConvs.vuFluidPints 2004
StdConvs.vuFluidPints variable 2004
StdConvs.vuFluidQuarts 2004
StdConvs.vuFluidQuarts variable 2004
StdConvs.vuFluidTablespoons 2004
StdConvs.vuFluidTablespoons variable 2004
StdConvs.vuFluidTeaspoons 2005
StdConvs.vuFluidTeaspoons variable 2005
StdConvs.vuHectoLiters 2005
StdConvs.vuHectoLiters variable 2005
StdConvs.vuKiloLiters 2005
StdConvs.vuKiloLiters variable 2005
StdConvs.vuLiters 2006
StdConvs.vuLiters variable 2006
StdConvs.vuMilliLiters 2006
StdConvs.vuMilliLiters variable 2006
StdConvs.vuSteres 2006
StdConvs.vuSteres variable 2006
StdConvs.vuUKBuckets 2007
StdConvs.vuUKBuckets variable 2007
StdConvs.vuUKBushels 2007
StdConvs.vuUKBushels variable 2007
StdConvs.vuUKGallons 2007
StdConvs.vuUKGallons variable 2007
StdConvs.vuUKGills 2008
StdConvs.vuUKGills variable 2008
StdConvs.vuUKOunces 2008
StdConvs.vuUKOunces variable 2008
StdConvs.vuUKPecks 2008

StdConvs.vuUKPecks variable 2008
StdConvs.vuUKPints 2009
StdConvs.vuUKPints variable 2009
StdConvs.vuUKPottles 2009
StdConvs.vuUKPottles variable 2009
StdConvs.vuUKQuarts 2009
StdConvs.vuUKQuarts variable 2009
StdCtrls 2033
StdCtrls namespace 2033
Classes 2035
Structs, Records, Enums 2999
Types 3008
StdCtrls.TButton 2035
StdCtrls.TButton.Action 2039
StdCtrls.TButton.Align 2040
StdCtrls.TButton.Anchors 2044
StdCtrls.TButton.BiDiMode 2044
StdCtrls.TButton.Cancel 2045
StdCtrls.TButton.Caption 2045
StdCtrls.TButton.CommandLinkHint 2048
StdCtrls.TButton.Constraints 2048
StdCtrls.TButton.Default 2049
StdCtrls.TButton.DisabledImageIndex 2050
StdCtrls.TButton.DoubleBuffered 2050
StdCtrls.TButton.DragCursor 2052
StdCtrls.TButton.DragKind 2052
StdCtrls.TButton.DragMode 2053
StdCtrls.TButton.DropDownMenu 2054
StdCtrls.TButton.ElevationRequired 2055
StdCtrls.TButton.Enabled 2055
StdCtrls.TButton.Font 2056
StdCtrls.TButton.HotImageIndex 2057
StdCtrls.TButton.ImageAlignment 2057
StdCtrls.TButton.ImageIndex 2058
StdCtrls.TButton.ImageMargins 2058
StdCtrls.TButton.Images 2059
StdCtrls.TButton.ModalResult 2059
StdCtrls.TButton.OnClick 2073
StdCtrls.TButton.OnContextPopup 2075
StdCtrls.TButton.OnDragDrop 2076
StdCtrls.TButton.OnDragOver 2079
StdCtrls.TButton.OnDropDownClick 2082
StdCtrls.TButton.OnEndDock 2082
StdCtrls.TButton.OnEndDrag 2083
StdCtrls.TButton.OnEnter 2084
StdCtrls.TButton.OnExit 2086
StdCtrls.TButton.OnKeyDown 2090
StdCtrls.TButton.OnKeyPress 2094
StdCtrls.TButton.OnKeyUp 2095
StdCtrls.TButton.OnMouseActivate 2096
StdCtrls.TButton.OnMouseDown 2097
StdCtrls.TButton.OnMouseEnter 2100
StdCtrls.TButton.OnMouseLeave 2100
StdCtrls.TButton.OnMouseMove 2100
StdCtrls.TButton.OnMouseUp 2103
StdCtrls.TButton.OnStartDock 2105
StdCtrls.TButton.OnStartDrag 2105
StdCtrls.TButton.ParentBiDiMode 2060
StdCtrls.TButton.ParentDoubleBuffered 2060
StdCtrls.TButton.ParentFont 2061
StdCtrls.TButton.ParentShowHint 2061
StdCtrls.TButton.PopupMenu 2062
StdCtrls.TButton.PressedImageIndex 2067
StdCtrls.TButton.SelectedImageIndex 2067
StdCtrls.TButton.ShowHint 2068
StdCtrls.TButton.Style 2069
StdCtrls.TButton.TabOrder 2069
StdCtrls.TButton.TabStop 2070
StdCtrls.TButton.Visible 2070
StdCtrls.TButton.WordWrap 2072
StdCtrls.TButtonActionLink 2106
StdCtrls.TButtonActionLink.AssignClient 2108
StdCtrls.TButtonActionLink.FClient 2107
StdCtrls.TButtonActionLink.IsCheckedLinked 2108
StdCtrls.TButtonActionLink.SetChecked 2108
StdCtrls.TButtonActionLinkClass 3008
StdCtrls.TButtonActionLinkClass type 3008
StdCtrls.TButtonControl 2109

StdCtrls.TButtonControl.ActionChange 2110
StdCtrls.TButtonControl.Checked 2113
StdCtrls.TButtonControl.ClicksDisabled 2115
StdCtrls.TButtonControl.Create 2110
StdCtrls.TButtonControl.CreateParams 2111
StdCtrls.TButtonControl.GetActionLinkClass 2111
StdCtrls.TButtonControl.GetChecked 2112
StdCtrls.TButtonControl.SetChecked 2112
StdCtrls.TButtonControl.WndProc 2112
StdCtrls.TButtonControl.WordWrap 2115
StdCtrls.TCheckBox 2116
StdCtrls.TCheckBox.Action 2119
StdCtrls.TCheckBox.Align 2120
StdCtrls.TCheckBox.Alignment 2123
StdCtrls.TCheckBox.AllowGrayed 2124
StdCtrls.TCheckBox.Anchors 2124
StdCtrls.TCheckBox.BiDiMode 2125
StdCtrls.TCheckBox.Caption 2126
StdCtrls.TCheckBox.Checked 2128
StdCtrls.TCheckBox.Color 2130
StdCtrls.TCheckBox.Constraints 2131
StdCtrls.TCheckBox.Ctl3D 2132
StdCtrls.TCheckBox.DoubleBuffered 2132
StdCtrls.TCheckBox.DragCursor 2134
StdCtrls.TCheckBox.DragKind 2134
StdCtrls.TCheckBox.DragMode 2135
StdCtrls.TCheckBox.Enabled 2136
StdCtrls.TCheckBox.Font 2137
StdCtrls.TCheckBox.OnClick 2151
StdCtrls.TCheckBox.OnContextPopup 2153
StdCtrls.TCheckBox.OnDragDrop 2154
StdCtrls.TCheckBox.OnDragOver 2157
StdCtrls.TCheckBox.OnEndDock 2160
StdCtrls.TCheckBox.OnEndDrag 2160
StdCtrls.TCheckBox.OnEnter 2161
StdCtrls.TCheckBox.OnExit 2164
StdCtrls.TCheckBox.OnKeyDown 2168
StdCtrls.TCheckBox.OnKeyPress 2171
StdCtrls.TCheckBox.OnKeyUp 2172
StdCtrls.TCheckBox.OnMouseActivate 2174
StdCtrls.TCheckBox.OnMouseDown 2175
StdCtrls.TCheckBox.OnMouseEnter 2177
StdCtrls.TCheckBox.OnMouseLeave 2178
StdCtrls.TCheckBox.OnMouseMove 2178
StdCtrls.TCheckBox.OnMouseUp 2180
StdCtrls.TCheckBox.OnStartDock 2182
StdCtrls.TCheckBox.OnStartDrag 2183
StdCtrls.TCheckBox.ParentBiDiMode 2138
StdCtrls.TCheckBox.ParentColor 2138
StdCtrls.TCheckBox.ParentCtl3D 2139
StdCtrls.TCheckBox.ParentDoubleBuffered 2139
StdCtrls.TCheckBox.ParentFont 2139
StdCtrls.TCheckBox.ParentShowHint 2140
StdCtrls.TCheckBox.PopupMenu 2140
StdCtrls.TCheckBox.ShowHint 2145
StdCtrls.TCheckBox.State 2147
StdCtrls.TCheckBox.TabOrder 2148
StdCtrls.TCheckBox.TabStop 2148
StdCtrls.TCheckBox.Visible 2149
StdCtrls.TCheckBox.WordWrap 2150
StdCtrls.TCheckBoxState 2999
StdCtrls.TCheckBoxState enumeration 2999
StdCtrls.TComboBox 2184
StdCtrls.TComboBox.Align 2190
StdCtrls.TComboBox.Anchors 2193
StdCtrls.TComboBox.AutoCloseUp 2194
StdCtrls.TComboBox.AutoComplete 2194
StdCtrls.TComboBox.AutoCompleteDelay 2195
StdCtrls.TComboBox.AutoDropDown 2195
StdCtrls.TComboBox.BevelEdges 2195
StdCtrls.TComboBox.BevelInner 2196
StdCtrls.TComboBox.BevelKind 2196
StdCtrls.TComboBox.BevelOuter 2197
StdCtrls.TComboBox.BiDiMode 2197
StdCtrls.TComboBox.CharCase 2197
StdCtrls.TComboBox.Color 2198
StdCtrls.TComboBox.Constraints 2199
StdCtrls.TComboBox.Ctl3D 2199

StdCtrls.TComboBox.DoubleBuffered 2200
StdCtrls.TComboBox.DragCursor 2202
StdCtrls.TComboBox.DragKind 2202
StdCtrls.TComboBox.DragMode 2202
StdCtrls.TComboBox.DropDownCount 2204
StdCtrls.TComboBox.Enabled 2204
StdCtrls.TComboBox.Font 2205
StdCtrls.TComboBox.ImeMode 2206
StdCtrls.TComboBox.ImeName 2206
StdCtrls.TComboBox.ItemHeight 2207
StdCtrls.TComboBox.ItemIndex 2208
StdCtrls.TComboBox.Items 2210
StdCtrls.TComboBox.MaxLength 2212
StdCtrls.TComboBox.OnChange 2228
StdCtrls.TComboBox.OnClick 2230
StdCtrls.TComboBox.OnCloseUp 2232
StdCtrls.TComboBox.OnContextPopup 2233
StdCtrls.TComboBox.OnDblClick 2233
StdCtrls.TComboBox.OnDragDrop 2234
StdCtrls.TComboBox.OnDragOver 2237
StdCtrls.TComboBox.OnDrawItem 2240
StdCtrls.TComboBox.OnDropDown 2240
StdCtrls.TComboBox.OnEndDock 2242
StdCtrls.TComboBox.OnEndDrag 2242
StdCtrls.TComboBox.OnEnter 2243
StdCtrls.TComboBox.OnExit 2246
StdCtrls.TComboBox.OnKeyDown 2250
StdCtrls.TComboBox.OnKeyPress 2253
StdCtrls.TComboBox.OnKeyUp 2254
StdCtrls.TComboBox.OnMeasureItem 2256
StdCtrls.TComboBox.OnMouseEnter 2257
StdCtrls.TComboBox.OnMouseLeave 2257
StdCtrls.TComboBox.OnSelect 2257
StdCtrls.TComboBox.OnStartDock 2257
StdCtrls.TComboBox.OnStartDrag 2258
StdCtrls.TComboBox.ParentBiDiMode 2212
StdCtrls.TComboBox.ParentColor 2213
StdCtrls.TComboBox.ParentCtl3D 2213
StdCtrls.TComboBox.ParentDoubleBuffered 2214
StdCtrls.TComboBox.ParentFont 2214
StdCtrls.TComboBox.ParentShowHint 2214
StdCtrls.TComboBox.PopupMenu 2215
StdCtrls.TComboBox.ShowHint 2220
StdCtrls.TComboBox.Sorted 2221
StdCtrls.TComboBox.Style 2223
StdCtrls.TComboBox.TabOrder 2223
StdCtrls.TComboBox.TabStop 2224
StdCtrls.TComboBox.Text 2224
StdCtrls.TComboBox.TextHint 2226
StdCtrls.TComboBox.Visible 2226
StdCtrls.TComboBoxStyle 3000
StdCtrls.TComboBoxStyle enumeration 3000
StdCtrls.TCustomButton 2259
StdCtrls.TCustomButton.ActionChange 2262
StdCtrls.TCustomButton.Cancel 2267
StdCtrls.TCustomButton.Click 2263
StdCtrls.TCustomButton.CommandLinkHint 2267
StdCtrls.TCustomButton.Create 2260
StdCtrls.TCustomButton.CreateParams 2263
StdCtrls.TCustomButton.CreateWnd 2264
StdCtrls.TCustomButton.Default 2268
StdCtrls.TCustomButton.Destroy 2261
StdCtrls.TCustomButton.DisabledImageIndex 2269
StdCtrls.TCustomButton.DropDownMenu 2269
StdCtrls.TCustomButton.ElevationRequired 2270
StdCtrls.TCustomButton.GetActionLinkClass 2264
StdCtrls.TCustomButton.HotImageIndex 2270
StdCtrls.TCustomButton.ImageAlignment 2271
StdCtrls.TCustomButton.ImageIndex 2271
StdCtrls.TCustomButton.ImageMargins 2272
StdCtrls.TCustomButton.Images 2272
StdCtrls.TCustomButton.Loaded 2264
StdCtrls.TCustomButton.ModalResult 2273
StdCtrls.TCustomButton.Notification 2265
StdCtrls.TCustomButton.OnDropDownClick 2276
StdCtrls.TCustomButton.PressedImageIndex 2274
StdCtrls.TCustomButton.SelectedImageIndex 2274
StdCtrls.TCustomButton.SetButtonStyle 2265

StdCtrls.TCustomButton.Style 2275
StdCtrls.TCustomButton.TabStop 2275
StdCtrls.TCustomButton.TButtonStyle 2261
StdCtrls.TCustomButton.TButtonStyle enumeration 2261
StdCtrls.TCustomButton.UpdateImageList 2266
StdCtrls.TCustomButton.UpdateImages 2266
StdCtrls.TCustomButton.UseRightToLeftAlignment 2266
StdCtrls.TCustomCheckBox 2276
StdCtrls.TCustomCheckBox.Alignment 2281
StdCtrls.TCustomCheckBox.AllowGrayed 2281
StdCtrls.TCustomCheckBox.Click 2278
StdCtrls.TCustomCheckBox.Create 2277
StdCtrls.TCustomCheckBox.CreateParams 2278
StdCtrls.TCustomCheckBox.CreateWnd 2279
StdCtrls.TCustomCheckBox.GetChecked 2279
StdCtrls.TCustomCheckBox.GetControlsAlignment 2279
StdCtrls.TCustomCheckBox.SetChecked 2280
StdCtrls.TCustomCheckBox.State 2282
StdCtrls.TCustomCheckBox.TabStop 2282
StdCtrls.TCustomCheckBox.Toggle 2280
StdCtrls.TCustomCombo 2283
StdCtrls.TCustomCombo.AddItem 2291
StdCtrls.TCustomCombo.AdjustDropDown 2292
StdCtrls.TCustomCombo.Canvas 2305
StdCtrls.TCustomCombo.Change 2292
StdCtrls.TCustomCombo.Clear 2293
StdCtrls.TCustomCombo.ClearSelection 2293
StdCtrls.TCustomCombo.CloseUp 2293
StdCtrls.TCustomCombo.ComboWndProc 2294
StdCtrls.TCustomCombo.CopySelection 2294
StdCtrls.TCustomCombo.Create 2285
StdCtrls.TCustomCombo.CreateWnd 2295
StdCtrls.TCustomCombo.DeleteSelected 2295
StdCtrls.TCustomCombo.Destroy 2286
StdCtrls.TCustomCombo.DestroyWindowHandle 2296
StdCtrls.TCustomCombo.DropDown 2296
StdCtrls.TCustomCombo.DropDownCount 2306
StdCtrls.TCustomCombo.DroppedDown 2306
StdCtrls.TCustomCombo.EditHandle 2307
StdCtrls.TCustomCombo.EditWndProc 2296
StdCtrls.TCustomCombo.FDefEditProc 2286
StdCtrls.TCustomCombo.FDefListProc 2287
StdCtrls.TCustomCombo.FDropHandle 2287
StdCtrls.TCustomCombo.FDroppingDown 2287
StdCtrls.TCustomCombo.FEditHandle 2288
StdCtrls.TCustomCombo.FEditInstance 2288
StdCtrls.TCustomCombo.FFocusChanged 2288
StdCtrls.TCustomCombo.FIsFocused 2288
StdCtrls.TCustomCombo.FListHandle 2289
StdCtrls.TCustomCombo.FListInstance 2289
StdCtrls.TCustomCombo.Focused 2297
StdCtrls.TCustomCombo.FSaveIndex 2289
StdCtrls.TCustomCombo.GetCount 2297
StdCtrls.TCustomCombo.GetDroppedDown 2297
StdCtrls.TCustomCombo.GetItemCount 2298
StdCtrls.TCustomCombo.GetItemHt 2298
StdCtrls.TCustomCombo.GetItemIndex 2298
StdCtrls.TCustomCombo.GetItemsClass 2299
StdCtrls.TCustomCombo.GetSelLength 2299
StdCtrls.TCustomCombo.GetSelStart 2299
StdCtrls.TCustomCombo.ItemCount 2307
StdCtrls.TCustomCombo.ItemHeight 2307
StdCtrls.TCustomCombo.Items 2308
StdCtrls.TCustomCombo.ListHandle 2310
StdCtrls.TCustomCombo.ListWndProc 2300
StdCtrls.TCustomCombo.Loaded 2300
StdCtrls.TCustomCombo.MaxLength 2310
StdCtrls.TCustomCombo.OnChange 2318
StdCtrls.TCustomCombo.OnCloseUp 2320
StdCtrls.TCustomCombo.OnDropDown 2320
StdCtrls.TCustomCombo.OnSelect 2322
StdCtrls.TCustomCombo.ParentColor 2311
StdCtrls.TCustomCombo.Select 2301
StdCtrls.TCustomCombo.SelectAll 2301
StdCtrls.TCustomCombo.SelLength 2311
StdCtrls.TCustomCombo.SelStart 2314
StdCtrls.TCustomCombo.SetDropDownCount 2301
StdCtrls.TCustomCombo.SetDroppedDown 2302

StdCtrls.TCustomCombo.SetItemHeight 2302
StdCtrls.TCustomCombo.SetItemIndex 2302
StdCtrls.TCustomCombo.SetItems 2303
StdCtrls.TCustomCombo.SetMaxLength 2303
StdCtrls.TCustomCombo.SetSelLength 2303
StdCtrls.TCustomCombo.SetSelStart 2304
StdCtrls.TCustomCombo.TabStop 2317
StdCtrls.TCustomCombo.WndProc 2304
StdCtrls.TCustomComboBox 2322
StdCtrls.TCustomComboBox.AutoCloseUp 2332
StdCtrls.TCustomComboBox.AutoComplete 2332
StdCtrls.TCustomComboBox.AutoCompleteDelay 2333
StdCtrls.TCustomComboBox.AutoDropDown 2334
StdCtrls.TCustomComboBox.CharCase 2334
StdCtrls.TCustomComboBox.Create 2325
StdCtrls.TCustomComboBox.CreateParams 2326
StdCtrls.TCustomComboBox.CreateWnd 2327
StdCtrls.TCustomComboBox.Destroy 2326
StdCtrls.TCustomComboBox.DestroyWnd 2327
StdCtrls.TCustomComboBox.DoSetTextHint 2328
StdCtrls.TCustomComboBox.DrawItem 2328
StdCtrls.TCustomComboBox.GetItemCount 2328
StdCtrls.TCustomComboBox.GetItemHt 2329
StdCtrls.TCustomComboBox.GetItemsClass 2329
StdCtrls.TCustomComboBox.GetSelText 2329
StdCtrls.TCustomComboBox.KeyPress 2329
StdCtrls.TCustomComboBox.MeasureItem 2330
StdCtrls.TCustomComboBox.OnDrawItem 2338
StdCtrls.TCustomComboBox.OnMeasureItem 2338
StdCtrls.TCustomComboBox.SelectItem 2330
StdCtrls.TCustomComboBox.SelText 2335
StdCtrls.TCustomComboBox.SetStyle 2331
StdCtrls.TCustomComboBox.Sorted 2335
StdCtrls.TCustomComboBox.Style 2337
StdCtrls.TCustomComboBox.TextHint 2337
StdCtrls.TCustomComboBox.WndProc 2331
StdCtrls.TCustomComboBoxStrings 2339
StdCtrls.TCustomComboBoxStrings.Clear 2341
StdCtrls.TCustomComboBoxStrings.ComboBox 2344
StdCtrls.TCustomComboBoxStrings.Create 2340
StdCtrls.TCustomComboBoxStrings.Delete 2341
StdCtrls.TCustomComboBoxStrings.Destroy 2340
StdCtrls.TCustomComboBoxStrings.Get 2342
StdCtrls.TCustomComboBoxStrings.GetCount 2342
StdCtrls.TCustomComboBoxStrings.GetObject 2342
StdCtrls.TCustomComboBoxStrings.IndexOf 2343
StdCtrls.TCustomComboBoxStrings.PutObject 2343
StdCtrls.TCustomComboBoxStrings.SetUpdateState 2344
StdCtrls.TCustomComboBoxStringsClass 3008
StdCtrls.TCustomComboBoxStringsClass type 3008
StdCtrls.TCustomEdit 2345
StdCtrls.TCustomEdit.Alignment 2365
StdCtrls.TCustomEdit.AutoSelect 2365
StdCtrls.TCustomEdit.AutoSize 2366
StdCtrls.TCustomEdit.BorderStyle 2367
StdCtrls.TCustomEdit.CanUndo 2367
StdCtrls.TCustomEdit.Change 2348
StdCtrls.TCustomEdit.CharCase 2367
StdCtrls.TCustomEdit.Clear 2348
StdCtrls.TCustomEdit.ClearSelection 2349
StdCtrls.TCustomEdit.ClearUndo 2349
StdCtrls.TCustomEdit.CopyToClipboard 2349
StdCtrls.TCustomEdit.Create 2347
StdCtrls.TCustomEdit.CreateParams 2351
StdCtrls.TCustomEdit.CreateWindowHandle 2351
StdCtrls.TCustomEdit.CreateWnd 2352
StdCtrls.TCustomEdit.CutToClipboard 2352
StdCtrls.TCustomEdit.DefaultHandler 2354
StdCtrls.TCustomEdit.DestroyWnd 2354
StdCtrls.TCustomEdit.DoSetMaxLength 2354
StdCtrls.TCustomEdit.DoSetTextHint 2355
StdCtrls.TCustomEdit.GetControlsAlignment 2355
StdCtrls.TCustomEdit.GetSelLength 2356
StdCtrls.TCustomEdit.GetSelStart 2356
StdCtrls.TCustomEdit.GetSelText 2356
StdCtrls.TCustomEdit.GetSelTextBuf 2357
StdCtrls.TCustomEdit.HideSelection 2368
StdCtrls.TCustomEdit.MaxLength 2368

StdCtrls.TCustomEdit.Modified 2369
StdCtrls.TCustomEdit.NumbersOnly 2369
StdCtrls.TCustomEdit.OEMConvert 2369
StdCtrls.TCustomEdit.OnChange 2381
StdCtrls.TCustomEdit.ParentColor 2370
StdCtrls.TCustomEdit.PasswordChar 2370
StdCtrls.TCustomEdit.PasteFromClipboard 2358
StdCtrls.TCustomEdit.ReadOnly 2371
StdCtrls.TCustomEdit.SelectAll 2359
StdCtrls.TCustomEdit.SelLength 2371
StdCtrls.TCustomEdit.SelStart 2374
StdCtrls.TCustomEdit.SelText 2377
StdCtrls.TCustomEdit.SetAlignment 2361
StdCtrls.TCustomEdit.SetAutoSize 2361
StdCtrls.TCustomEdit.SetSelLength 2361
StdCtrls.TCustomEdit.SetSelStart 2362
StdCtrls.TCustomEdit.SetSelText 2362
StdCtrls.TCustomEdit.SetSelTextBuf 2363
StdCtrls.TCustomEdit.TabStop 2379
StdCtrls.TCustomEdit.Text 2379
StdCtrls.TCustomEdit.TextHint 2380
StdCtrls.TCustomEdit.Undo 2364
StdCtrls.TCustomGroupBox 2381
StdCtrls.TCustomGroupBox.AdjustClientRect 2382
StdCtrls.TCustomGroupBox.Create 2382
StdCtrls.TCustomGroupBox.CreateParams 2383
StdCtrls.TCustomGroupBox.Paint 2383
StdCtrls.TCustomLabel 2384
StdCtrls.TCustomLabel.AdjustBounds 2385
StdCtrls.TCustomLabel.Alignment 2388
StdCtrls.TCustomLabel.AutoSize 2389
StdCtrls.TCustomLabel.Canvas 2390
StdCtrls.TCustomLabel.Caption 2392
StdCtrls.TCustomLabel.Create 2385
StdCtrls.TCustomLabel.DoDrawText 2386
StdCtrls.TCustomLabel.EllipsisPosition 2395
StdCtrls.TCustomLabel.FocusControl 2395
StdCtrls.TCustomLabel.GetLabelText 2386
StdCtrls.TCustomLabel.GlowSize 2395
StdCtrls.TCustomLabel.Layout 2396
StdCtrls.TCustomLabel.Loaded 2387
StdCtrls.TCustomLabel.Notification 2387
StdCtrls.TCustomLabel.Paint 2387
StdCtrls.TCustomLabel.SetAutoSize 2388
StdCtrls.TCustomLabel.ShowAccelChar 2396
StdCtrls.TCustomLabel.Transparent 2397
StdCtrls.TCustomLabel.WordWrap 2397
StdCtrls.TCustomListBox 2398
StdCtrls.TCustomListBox.AddItem 2402
StdCtrls.TCustomListBox.AutoComplete 2423
StdCtrls.TCustomListBox.AutoCompleteDelay 2423
StdCtrls.TCustomListBox.BorderStyle 2423
StdCtrls.TCustomListBox.Canvas 2424
StdCtrls.TCustomListBox.Clear 2403
StdCtrls.TCustomListBox.ClearSelection 2403
StdCtrls.TCustomListBox.Columns 2426
StdCtrls.TCustomListBox.CopySelection 2404
StdCtrls.TCustomListBox.Count 2427
StdCtrls.TCustomListBox.Create 2400
StdCtrls.TCustomListBox.CreateParams 2404
StdCtrls.TCustomListBox.CreateWnd 2405
StdCtrls.TCustomListBox.DeleteSelected 2406
StdCtrls.TCustomListBox.DeleteString 2406
StdCtrls.TCustomListBox.Destroy 2401
StdCtrls.TCustomListBox.DestroyWnd 2406
StdCtrls.TCustomListBox.DoFindData 2407
StdCtrls.TCustomListBox.DoGetData 2407
StdCtrls.TCustomListBox.DoGetDataObject 2408
StdCtrls.TCustomListBox.DragCanceled 2408
StdCtrls.TCustomListBox.DrawItem 2408
StdCtrls.TCustomListBox.ExtendedSelect 2427
StdCtrls.TCustomListBox.FMoving 2401
StdCtrls.TCustomListBox.GetCount 2409
StdCtrls.TCustomListBox.GetItemData 2409
StdCtrls.TCustomListBox.GetItemIndex 2410
StdCtrls.TCustomListBox.GetSelCount 2410
StdCtrls.TCustomListBox.GetSelected 2410
StdCtrls.TCustomListBox.IntegralHeight 2428

StdCtrls.TCustomListBox.InternalGetItemData 2411
StdCtrls.TCustomListBox.InternalSetItemData 2411
StdCtrls.TCustomListBox.ItemAtPos 2411
StdCtrls.TCustomListBox.ItemHeight 2428
StdCtrls.TCustomListBox.ItemRect 2413
StdCtrls.TCustomListBox.Items 2429
StdCtrls.TCustomListBox.KeyPress 2415
StdCtrls.TCustomListBox.LoadRecreateItems 2415
StdCtrls.TCustomListBox.MeasureItem 2415
StdCtrls.TCustomListBox.OnData 2438
StdCtrls.TCustomListBox.OnDataFind 2439
StdCtrls.TCustomListBox.OnDataObject 2439
StdCtrls.TCustomListBox.OnDrawItem 2439
StdCtrls.TCustomListBox.OnMeasureItem 2442
StdCtrls.TCustomListBox.ParentColor 2431
StdCtrls.TCustomListBox.ResetContent 2416
StdCtrls.TCustomListBox.SaveRecreateItems 2416
StdCtrls.TCustomListBox.ScrollWidth 2432
StdCtrls.TCustomListBox.SelectAll 2416
StdCtrls.TCustomListBox.Selected 2432
StdCtrls.TCustomListBox.SetItemData 2417
StdCtrls.TCustomListBox.SetItemIndex 2417
StdCtrls.TCustomListBox.SetMultiSelect 2417
StdCtrls.TCustomListBox.Sorted 2434
StdCtrls.TCustomListBox.Style 2436
StdCtrls.TCustomListBox.TabStop 2436
StdCtrls.TCustomListBox.TabWidth 2437
StdCtrls.TCustomListBox.TopIndex 2437
StdCtrls.TCustomListBox.WndProc 2418
StdCtrls.TCustomMemo 2443
StdCtrls.TCustomMemo.CaretPos 2450
StdCtrls.TCustomMemo.Create 2446
StdCtrls.TCustomMemo.CreateParams 2447
StdCtrls.TCustomMemo.CreateWindowHandle 2447
StdCtrls.TCustomMemo.Destroy 2446
StdCtrls.TCustomMemo.GetCaretPos 2448
StdCtrls.TCustomMemo.KeyPress 2448
StdCtrls.TCustomMemo.Lines 2451
StdCtrls.TCustomMemo.Loaded 2448
StdCtrls.TCustomMemo.ScrollBars 2452
StdCtrls.TCustomMemo.SetCaretPos 2449
StdCtrls.TCustomMemo.SetLines 2449
StdCtrls.TCustomMemo.SetScrollBars 2449
StdCtrls.TCustomMemo.SetWordWrap 2450
StdCtrls.TCustomMemo.WantReturns 2454
StdCtrls.TCustomMemo.WantTabs 2454
StdCtrls.TCustomMemo.WordWrap 2455
StdCtrls.TCustomStaticText 2457
StdCtrls.TCustomStaticText.Alignment 2461
StdCtrls.TCustomStaticText.AutoSize 2461
StdCtrls.TCustomStaticText.BorderStyle 2462
StdCtrls.TCustomStaticText.Create 2458
StdCtrls.TCustomStaticText.CreateParams 2459
StdCtrls.TCustomStaticText.FocusControl 2462
StdCtrls.TCustomStaticText.Loaded 2459
StdCtrls.TCustomStaticText.Notification 2460
StdCtrls.TCustomStaticText.SetAutoSize 2460
StdCtrls.TCustomStaticText.ShowAccelChar 2462
StdCtrls.TCustomStaticText.Transparent 2463
StdCtrls.TDrawItemEvent 3009
StdCtrls.TDrawItemEvent type 3009
StdCtrls.TEdit 2463
StdCtrls.TEdit.Align 2467
StdCtrls.TEdit.Alignment 2471
StdCtrls.TEdit.Anchors 2471
StdCtrls.TEdit.AutoSelect 2472
StdCtrls.TEdit.AutoSize 2472
StdCtrls.TEdit.BevelEdges 2473
StdCtrls.TEdit.BevelInner 2474
StdCtrls.TEdit.BevelKind 2474
StdCtrls.TEdit.BevelOuter 2474
StdCtrls.TEdit.BevelWidth 2475
StdCtrls.TEdit.BiDiMode 2475
StdCtrls.TEdit.BorderStyle 2476
StdCtrls.TEdit.CharCase 2476
StdCtrls.TEdit.Color 2476
StdCtrls.TEdit.Constraints 2477
StdCtrls.TEdit.Ctl3D 2478

StdCtrls.TEdit.DoubleBuffered 2478
StdCtrls.TEdit.DragCursor 2480
StdCtrls.TEdit.DragKind 2481
StdCtrls.TEdit.DragMode 2481
StdCtrls.TEdit.Enabled 2482
StdCtrls.TEdit.Font 2483
StdCtrls.TEdit.HideSelection 2484
StdCtrls.TEdit.ImeMode 2484
StdCtrls.TEdit.ImeName 2485
StdCtrls.TEdit.MaxLength 2486
StdCtrls.TEdit.NumbersOnly 2486
StdCtrls.TEdit.OEMConvert 2486
StdCtrls.TEdit.OnChange 2501
StdCtrls.TEdit.OnClick 2502
StdCtrls.TEdit.OnContextPopup 2504
StdCtrls.TEdit.OnDblClick 2504
StdCtrls.TEdit.OnDragDrop 2505
StdCtrls.TEdit.OnDragOver 2508
StdCtrls.TEdit.OnEndDock 2511
StdCtrls.TEdit.OnEndDrag 2511
StdCtrls.TEdit.OnEnter 2512
StdCtrls.TEdit.OnExit 2515
StdCtrls.TEdit.OnKeyDown 2519
StdCtrls.TEdit.OnKeyPress 2522
StdCtrls.TEdit.OnKeyUp 2523
StdCtrls.TEdit.OnMouseActivate 2525
StdCtrls.TEdit.OnMouseDown 2526
StdCtrls.TEdit.OnMouseEnter 2528
StdCtrls.TEdit.OnMouseLeave 2529
StdCtrls.TEdit.OnMouseMove 2529
StdCtrls.TEdit.OnMouseUp 2531
StdCtrls.TEdit.OnStartDock 2533
StdCtrls.TEdit.OnStartDrag 2534
StdCtrls.TEdit.ParentBiDiMode 2487
StdCtrls.TEdit.ParentColor 2487
StdCtrls.TEdit.ParentCtl3D 2488
StdCtrls.TEdit.ParentDoubleBuffered 2488
StdCtrls.TEdit.ParentFont 2488
StdCtrls.TEdit.ParentShowHint 2489
StdCtrls.TEdit.PasswordChar 2489
StdCtrls.TEdit.PopupMenu 2490
StdCtrls.TEdit.ReadOnly 2495
StdCtrls.TEdit.ShowHint 2495
StdCtrls.TEdit.TabOrder 2497
StdCtrls.TEdit.TabStop 2497
StdCtrls.TEdit.Text 2498
StdCtrls.TEdit.TextHint 2499
StdCtrls.TEdit.Visible 2499
StdCtrls.TEditCharCase 3000
StdCtrls.TEditCharCase enumeration 3000
StdCtrls.TEllipsisPosition 3001
StdCtrls.TEllipsisPosition enumeration 3001
StdCtrls.TFNDrawText 3010
StdCtrls.TFNDrawText type 3010
StdCtrls.TGroupBox 2535
StdCtrls.TGroupBox.Align 2537
StdCtrls.TGroupBox.Anchors 2541
StdCtrls.TGroupBox.BiDiMode 2541
StdCtrls.TGroupBox.Caption 2542
StdCtrls.TGroupBox.Color 2544
StdCtrls.TGroupBox.Constraints 2545
StdCtrls.TGroupBox.Ctl3D 2546
StdCtrls.TGroupBox.DockSite 2546
StdCtrls.TGroupBox.DoubleBuffered 2547
StdCtrls.TGroupBox.DragCursor 2549
StdCtrls.TGroupBox.DragKind 2550
StdCtrls.TGroupBox.DragMode 2550
StdCtrls.TGroupBox.Enabled 2551
StdCtrls.TGroupBox.Font 2552
StdCtrls.TGroupBox.OnAlignInsertBefore 2566
StdCtrls.TGroupBox.OnAlignPosition 2566
StdCtrls.TGroupBox.OnClick 2567
StdCtrls.TGroupBox.OnContextPopup 2569
StdCtrls.TGroupBox.OnDblClick 2570
StdCtrls.TGroupBox.OnDockDrop 2570
StdCtrls.TGroupBox.OnDockOver 2572
StdCtrls.TGroupBox.OnDragDrop 2574
StdCtrls.TGroupBox.OnDragOver 2577

StdCtrls.TGroupBox.OnEndDock 2580
StdCtrls.TGroupBox.OnEndDrag 2580
StdCtrls.TGroupBox.OnEnter 2581
StdCtrls.TGroupBox.OnExit 2584
StdCtrls.TGroupBox.OnGetSiteInfo 2588
StdCtrls.TGroupBox.OnMouseActivate 2590
StdCtrls.TGroupBox.OnMouseDown 2591
StdCtrls.TGroupBox.OnMouseEnter 2593
StdCtrls.TGroupBox.OnMouseLeave 2594
StdCtrls.TGroupBox.OnMouseMove 2594
StdCtrls.TGroupBox.OnMouseUp 2596
StdCtrls.TGroupBox.OnStartDock 2598
StdCtrls.TGroupBox.OnStartDrag 2599
StdCtrls.TGroupBox.OnUnDock 2600
StdCtrls.TGroupBox.Padding 2553
StdCtrls.TGroupBox.ParentBackground 2553
StdCtrls.TGroupBox.ParentBiDiMode 2554
StdCtrls.TGroupBox.ParentColor 2554
StdCtrls.TGroupBox.ParentCtl3D 2555
StdCtrls.TGroupBox.ParentDoubleBuffered 2555
StdCtrls.TGroupBox.ParentFont 2555
StdCtrls.TGroupBox.ParentShowHint 2556
StdCtrls.TGroupBox.PopupMenu 2556
StdCtrls.TGroupBox.ShowHint 2561
StdCtrls.TGroupBox.TabOrder 2563
StdCtrls.TGroupBox.TabStop 2563
StdCtrls.TGroupBox.Visible 2564
StdCtrls.TImageAlignment 3002
StdCtrls.TImageAlignment enumeration 3002
StdCtrls.TImageMargins 2602
StdCtrls.TImageMargins.Assign 2604
StdCtrls.TImageMargins.Bottom 2605
StdCtrls.TImageMargins.Change 2604
StdCtrls.TImageMargins.Left 2605
StdCtrls.TImageMargins.OnChange 2607
StdCtrls.TImageMargins.Right 2606
StdCtrls.TImageMargins.Top 2606
StdCtrls.TLabel 2607
StdCtrls.TLabel.Align 2610
StdCtrls.TLabel.Alignment 2613
StdCtrls.TLabel.Anchors 2614
StdCtrls.TLabel.AutoSize 2614
StdCtrls.TLabel.BiDiMode 2615
StdCtrls.TLabel.Caption 2616
StdCtrls.TLabel.Color 2618
StdCtrls.TLabel.Constraints 2619
StdCtrls.TLabel.DragCursor 2620
StdCtrls.TLabel.DragKind 2620
StdCtrls.TLabel.DragMode 2621
StdCtrls.TLabel.EllipsisPosition 2622
StdCtrls.TLabel.Enabled 2623
StdCtrls.TLabel.FocusControl 2623
StdCtrls.TLabel.Font 2623
StdCtrls.TLabel.GlowSize 2624
StdCtrls.TLabel.Layout 2625
StdCtrls.TLabel.OnClick 2637
StdCtrls.TLabel.OnContextPopup 2639
StdCtrls.TLabel.OnDblClick 2640
StdCtrls.TLabel.OnDragDrop 2640
StdCtrls.TLabel.OnDragOver 2643
StdCtrls.TLabel.OnEndDock 2646
StdCtrls.TLabel.OnEndDrag 2647
StdCtrls.TLabel.OnMouseActivate 2648
StdCtrls.TLabel.OnMouseDown 2649
StdCtrls.TLabel.OnMouseEnter 2651
StdCtrls.TLabel.OnMouseLeave 2651
StdCtrls.TLabel.OnMouseMove 2652
StdCtrls.TLabel.OnMouseUp 2654
StdCtrls.TLabel.OnStartDock 2656
StdCtrls.TLabel.OnStartDrag 2657
StdCtrls.TLabel.ParentBiDiMode 2625
StdCtrls.TLabel.ParentColor 2626
StdCtrls.TLabel.ParentFont 2626
StdCtrls.TLabel.ParentShowHint 2627
StdCtrls.TLabel.PopupMenu 2627
StdCtrls.TLabel.ShowAccelChar 2632
StdCtrls.TLabel.ShowHint 2633
StdCtrls.TLabel.Transparent 2635

StdCtrls.TLabel.Visible 2635
StdCtrls.TLabel.WordWrap 2636
StdCtrls.TLBFindDataEvent 3010
StdCtrls.TLBFindDataEvent type 3010
StdCtrls.TLBGetDataEvent 3011
StdCtrls.TLBGetDataEvent type 3011
StdCtrls.TLBGetDataObjectEvent 3011
StdCtrls.TLBGetDataObjectEvent type 3011
StdCtrls.TListBox 2657
StdCtrls.TListBox.Align 2662
StdCtrls.TListBox.Anchors 2665
StdCtrls.TListBox.AutoComplete 2666
StdCtrls.TListBox.AutoCompleteDelay 2666
StdCtrls.TListBox.BevelEdges 2666
StdCtrls.TListBox.BevelInner 2667
StdCtrls.TListBox.BevelKind 2667
StdCtrls.TListBox.BevelOuter 2668
StdCtrls.TListBox.BevelWidth 2668
StdCtrls.TListBox.BiDiMode 2668
StdCtrls.TListBox.BorderStyle 2669
StdCtrls.TListBox.Color 2669
StdCtrls.TListBox.Columns 2670
StdCtrls.TListBox.Constraints 2671
StdCtrls.TListBox.Ctl3D 2671
StdCtrls.TListBox.DoubleBuffered 2672
StdCtrls.TListBox.DragCursor 2674
StdCtrls.TListBox.DragKind 2674
StdCtrls.TListBox.DragMode 2674
StdCtrls.TListBox.Enabled 2676
StdCtrls.TListBox.ExtendedSelect 2676
StdCtrls.TListBox.Font 2677
StdCtrls.TListBox.ImeMode 2678
StdCtrls.TListBox.ImeName 2678
StdCtrls.TListBox.IntegralHeight 2679
StdCtrls.TListBox.ItemHeight 2680
StdCtrls.TListBox.Items 2680
StdCtrls.TListBox.MultiSelect 2683
StdCtrls.TListBox.OnClick 2699
StdCtrls.TListBox.OnContextPopup 2701
StdCtrls.TListBox.OnData 2701
StdCtrls.TListBox.OnDataFind 2702
StdCtrls.TListBox.OnDataObject 2702
StdCtrls.TListBox.OnDblClick 2703
StdCtrls.TListBox.OnDragDrop 2703
StdCtrls.TListBox.OnDragOver 2706
StdCtrls.TListBox.OnDrawItem 2709
StdCtrls.TListBox.OnEndDock 2711
StdCtrls.TListBox.OnEndDrag 2712
StdCtrls.TListBox.OnEnter 2713
StdCtrls.TListBox.OnExit 2715
StdCtrls.TListBox.OnKeyDown 2719
StdCtrls.TListBox.OnKeyPress 2723
StdCtrls.TListBox.OnKeyUp 2724
StdCtrls.TListBox.OnMeasureItem 2725
StdCtrls.TListBox.OnMouseActivate 2727
StdCtrls.TListBox.OnMouseDown 2728
StdCtrls.TListBox.OnMouseEnter 2730
StdCtrls.TListBox.OnMouseLeave 2731
StdCtrls.TListBox.OnMouseMove 2731
StdCtrls.TListBox.OnMouseUp 2733
StdCtrls.TListBox.OnStartDock 2735
StdCtrls.TListBox.OnStartDrag 2736
StdCtrls.TListBox.ParentBiDiMode 2684
StdCtrls.TListBox.ParentColor 2684
StdCtrls.TListBox.ParentCtl3D 2685
StdCtrls.TListBox.ParentDoubleBuffered 2685
StdCtrls.TListBox.ParentFont 2685
StdCtrls.TListBox.ParentShowHint 2686
StdCtrls.TListBox.PopupMenu 2686
StdCtrls.TListBox.ScrollWidth 2691
StdCtrls.TListBox.ShowHint 2692
StdCtrls.TListBox.Sorted 2693
StdCtrls.TListBox.Style 2695
StdCtrls.TListBox.TabOrder 2695
StdCtrls.TListBox.TabStop 2696
StdCtrls.TListBox.TabWidth 2696
StdCtrls.TListBox.Visible 2697
StdCtrls.TListBoxStyle 3003

StdCtrls.TListBoxStyle enumeration 3003
StdCtrls.TMeasureItemEvent 3012
StdCtrls.TMeasureItemEvent type 3012
StdCtrls.TMemo 2737
StdCtrls.TMemo.Align 2741
StdCtrls.TMemo.Alignment 2745
StdCtrls.TMemo.Anchors 2745
StdCtrls.TMemo.BevelEdges 2746
StdCtrls.TMemo.BevelInner 2746
StdCtrls.TMemo.BevelKind 2747
StdCtrls.TMemo.BevelOuter 2747
StdCtrls.TMemo.BiDiMode 2747
StdCtrls.TMemo.BorderStyle 2748
StdCtrls.TMemo.CharCase 2748
StdCtrls.TMemo.Color 2749
StdCtrls.TMemo.Constraints 2749
StdCtrls.TMemo.Ctl3D 2750
StdCtrls.TMemo.DoubleBuffered 2750
StdCtrls.TMemo.DragCursor 2752
StdCtrls.TMemo.DragKind 2753
StdCtrls.TMemo.DragMode 2753
StdCtrls.TMemo.Enabled 2755
StdCtrls.TMemo.Font 2755
StdCtrls.TMemo.HideSelection 2756
StdCtrls.TMemo.ImeMode 2757
StdCtrls.TMemo.ImeName 2757
StdCtrls.TMemo.Lines 2758
StdCtrls.TMemo.MaxLength 2759
StdCtrls.TMemo.OEMConvert 2759
StdCtrls.TMemo.OnChange 2777
StdCtrls.TMemo.OnClick 2778
StdCtrls.TMemo.OnContextPopup 2780
StdCtrls.TMemo.OnDblClick 2780
StdCtrls.TMemo.OnDragDrop 2781
StdCtrls.TMemo.OnDragOver 2784
StdCtrls.TMemo.OnEndDock 2787
StdCtrls.TMemo.OnEndDrag 2787
StdCtrls.TMemo.OnEnter 2788
StdCtrls.TMemo.OnExit 2791
StdCtrls.TMemo.OnKeyDown 2795
StdCtrls.TMemo.OnKeyPress 2798
StdCtrls.TMemo.OnKeyUp 2799
StdCtrls.TMemo.OnMouseActivate 2801
StdCtrls.TMemo.OnMouseDown 2802
StdCtrls.TMemo.OnMouseEnter 2804
StdCtrls.TMemo.OnMouseLeave 2805
StdCtrls.TMemo.OnMouseMove 2805
StdCtrls.TMemo.OnMouseUp 2807
StdCtrls.TMemo.OnStartDock 2809
StdCtrls.TMemo.OnStartDrag 2810
StdCtrls.TMemo.ParentBiDiMode 2760
StdCtrls.TMemo.ParentColor 2760
StdCtrls.TMemo.ParentCtl3D 2761
StdCtrls.TMemo.ParentDoubleBuffered 2761
StdCtrls.TMemo.ParentFont 2761
StdCtrls.TMemo.ParentShowHint 2762
StdCtrls.TMemo.PopupMenu 2762
StdCtrls.TMemo.ReadOnly 2767
StdCtrls.TMemo.ScrollBars 2768
StdCtrls.TMemo.ShowHint 2770
StdCtrls.TMemo.TabOrder 2771
StdCtrls.TMemo.TabStop 2772
StdCtrls.TMemo.Visible 2772
StdCtrls.TMemo.WantReturns 2774
StdCtrls.TMemo.WantTabs 2774
StdCtrls.TMemo.WordWrap 2775
StdCtrls.TOwnerDrawState 3012
StdCtrls.TOwnerDrawState type 3012
StdCtrls.TPushButtonActionLink 2811
StdCtrls.TPushButtonActionLink.IsImageIndexLinked 2812
StdCtrls.TPushButtonActionLink.SetImageIndex 2812
StdCtrls.TPushButtonActionLinkClass 3013
StdCtrls.TPushButtonActionLinkClass type 3013
StdCtrls.TRadioButton 2813
StdCtrls.TRadioButton.Action 2818
StdCtrls.TRadioButton.Align 2820
StdCtrls.TRadioButton.Alignment 2823
StdCtrls.TRadioButton.Anchors 2823

StdCtrls.TRadioButton.BiDiMode 2824
StdCtrls.TRadioButton.Caption 2824
StdCtrls.TRadioButton.Checked 2827
StdCtrls.TRadioButton.Color 2827
StdCtrls.TRadioButton.Constraints 2828
StdCtrls.TRadioButton.Create 2815
StdCtrls.TRadioButton.CreateParams 2816
StdCtrls.TRadioButton.CreateWnd 2816
StdCtrls.TRadioButton.Ctl3D 2829
StdCtrls.TRadioButton.DoubleBuffered 2829
StdCtrls.TRadioButton.DragCursor 2831
StdCtrls.TRadioButton.DragKind 2831
StdCtrls.TRadioButton.DragMode 2832
StdCtrls.TRadioButton.Enabled 2833
StdCtrls.TRadioButton.Font 2834
StdCtrls.TRadioButton.GetChecked 2817
StdCtrls.TRadioButton.GetControlsAlignment 2817
StdCtrls.TRadioButton.OnClick 2848
StdCtrls.TRadioButton.OnContextPopup 2849
StdCtrls.TRadioButton.OnDblClick 2850
StdCtrls.TRadioButton.OnDragDrop 2850
StdCtrls.TRadioButton.OnDragOver 2853
StdCtrls.TRadioButton.OnEndDock 2857
StdCtrls.TRadioButton.OnEndDrag 2857
StdCtrls.TRadioButton.OnEnter 2858
StdCtrls.TRadioButton.OnExit 2861
StdCtrls.TRadioButton.OnKeyDown 2865
StdCtrls.TRadioButton.OnKeyPress 2868
StdCtrls.TRadioButton.OnKeyUp 2869
StdCtrls.TRadioButton.OnMouseActivate 2871
StdCtrls.TRadioButton.OnMouseDown 2872
StdCtrls.TRadioButton.OnMouseEnter 2874
StdCtrls.TRadioButton.OnMouseLeave 2874
StdCtrls.TRadioButton.OnMouseMove 2875
StdCtrls.TRadioButton.OnMouseUp 2877
StdCtrls.TRadioButton.OnStartDock 2879
StdCtrls.TRadioButton.OnStartDrag 2880
StdCtrls.TRadioButton.ParentBiDiMode 2835
StdCtrls.TRadioButton.ParentColor 2835
StdCtrls.TRadioButton.ParentCtl3D 2836
StdCtrls.TRadioButton.ParentDoubleBuffered 2836
StdCtrls.TRadioButton.ParentFont 2837
StdCtrls.TRadioButton.ParentShowHint 2837
StdCtrls.TRadioButton.PopupMenu 2838
StdCtrls.TRadioButton.SetChecked 2817
StdCtrls.TRadioButton.ShowHint 2843
StdCtrls.TRadioButton.TabOrder 2844
StdCtrls.TRadioButton.TabStop 2845
StdCtrls.TRadioButton.Visible 2845
StdCtrls.TRadioButton.WordWrap 2847
StdCtrls.TScrollBar 2881
StdCtrls.TScrollBar.Align 2886
StdCtrls.TScrollBar.Anchors 2890
StdCtrls.TScrollBar.BiDiMode 2890
StdCtrls.TScrollBar.Change 2883
StdCtrls.TScrollBar.Constraints 2891
StdCtrls.TScrollBar.Create 2883
StdCtrls.TScrollBar.CreateParams 2884
StdCtrls.TScrollBar.CreateWnd 2884
StdCtrls.TScrollBar.Ctl3D 2892
StdCtrls.TScrollBar.DoubleBuffered 2892
StdCtrls.TScrollBar.DragCursor 2894
StdCtrls.TScrollBar.DragKind 2894
StdCtrls.TScrollBar.DragMode 2895
StdCtrls.TScrollBar.Enabled 2896
StdCtrls.TScrollBar.Kind 2897
StdCtrls.TScrollBar.LargeChange 2898
StdCtrls.TScrollBar.Max 2900
StdCtrls.TScrollBar.Min 2901
StdCtrls.TScrollBar.OnChange 2920
StdCtrls.TScrollBar.OnContextPopup 2920
StdCtrls.TScrollBar.OnDragDrop 2921
StdCtrls.TScrollBar.OnDragOver 2924
StdCtrls.TScrollBar.OnEndDock 2927
StdCtrls.TScrollBar.OnEndDrag 2928
StdCtrls.TScrollBar.OnEnter 2929
StdCtrls.TScrollBar.OnExit 2931
StdCtrls.TScrollBar.OnKeyDown 2935

StdCtrls.TScrollBar.OnKeyPress 2939
StdCtrls.TScrollBar.OnKeyUp 2940
StdCtrls.TScrollBar.OnMouseEnter 2942
StdCtrls.TScrollBar.OnMouseLeave 2942
StdCtrls.TScrollBar.OnScroll 2942
StdCtrls.TScrollBar.OnStartDock 2944
StdCtrls.TScrollBar.OnStartDrag 2944
StdCtrls.TScrollBar.PageSize 2903
StdCtrls.TScrollBar.ParentBiDiMode 2905
StdCtrls.TScrollBar.ParentCtl3D 2905
StdCtrls.TScrollBar.ParentDoubleBuffered 2906
StdCtrls.TScrollBar.ParentShowHint 2906
StdCtrls.TScrollBar.PopupMenu 2907
StdCtrls.TScrollBar.Position 2912
StdCtrls.TScrollBar.Scroll 2885
StdCtrls.TScrollBar.SetParams 2885
StdCtrls.TScrollBar.ShowHint 2914
StdCtrls.TScrollBar.SmallChange 2915
StdCtrls.TScrollBar.TabOrder 2917
StdCtrls.TScrollBar.TabStop 2918
StdCtrls.TScrollBar.Visible 2918
StdCtrls.TScrollCode 3004
StdCtrls.TScrollCode enumeration 3004
StdCtrls.TScrollEvent 3013
StdCtrls.TScrollEvent type 3013
StdCtrls.TScrollStyle 3005
StdCtrls.TScrollStyle enumeration 3005
StdCtrls.TSelection 3006
StdCtrls.TSelection record 3006
StdCtrls.TStaticBorderStyle 3006
StdCtrls.TStaticBorderStyle enumeration 3006
StdCtrls.TStaticText 2945
StdCtrls.TStaticText.Align 2948
StdCtrls.TStaticText.Alignment 2951
StdCtrls.TStaticText.Anchors 2952
StdCtrls.TStaticText.AutoSize 2952
StdCtrls.TStaticText.BevelEdges 2953
StdCtrls.TStaticText.BevelInner 2953
StdCtrls.TStaticText.BevelKind 2954
StdCtrls.TStaticText.BevelOuter 2954
StdCtrls.TStaticText.BiDiMode 2954
StdCtrls.TStaticText.BorderStyle 2955
StdCtrls.TStaticText.Caption 2955
StdCtrls.TStaticText.Color 2958
StdCtrls.TStaticText.Constraints 2959
StdCtrls.TStaticText.DoubleBuffered 2959
StdCtrls.TStaticText.DragCursor 2961
StdCtrls.TStaticText.DragKind 2961
StdCtrls.TStaticText.DragMode 2962
StdCtrls.TStaticText.Enabled 2963
StdCtrls.TStaticText.FocusControl 2964
StdCtrls.TStaticText.Font 2964
StdCtrls.TStaticText.OnClick 2978
StdCtrls.TStaticText.OnContextPopup 2980
StdCtrls.TStaticText.OnDblClick 2980
StdCtrls.TStaticText.OnDragDrop 2981
StdCtrls.TStaticText.OnDragOver 2984
StdCtrls.TStaticText.OnEndDock 2987
StdCtrls.TStaticText.OnEndDrag 2987
StdCtrls.TStaticText.OnMouseActivate 2988
StdCtrls.TStaticText.OnMouseDown 2989
StdCtrls.TStaticText.OnMouseEnter 2991
StdCtrls.TStaticText.OnMouseLeave 2992
StdCtrls.TStaticText.OnMouseMove 2992
StdCtrls.TStaticText.OnMouseUp 2994
StdCtrls.TStaticText.OnStartDock 2996
StdCtrls.TStaticText.OnStartDrag 2997
StdCtrls.TStaticText.ParentBiDiMode 2965
StdCtrls.TStaticText.ParentColor 2966
StdCtrls.TStaticText.ParentDoubleBuffered 2966
StdCtrls.TStaticText.ParentFont 2966
StdCtrls.TStaticText.ParentShowHint 2967
StdCtrls.TStaticText.PopupMenu 2968
StdCtrls.TStaticText.ShowAccelChar 2973
StdCtrls.TStaticText.ShowHint 2973
StdCtrls.TStaticText.TabOrder 2974
StdCtrls.TStaticText.TabStop 2975
StdCtrls.TStaticText.Transparent 2975

StdCtrls.TStaticText.Visible 2976
StdCtrls.TTextLayout 3007
StdCtrls.TTextLayout enumeration 3007
stDevice enumeration member 3103
stDgram enumeration member 1502
stDisabled enumeration member 3104
StdStyleActnCtrls 3015
StdStyleActnCtrls namespace 3015
Classes 3016
Variables 3019
StdStyleActnCtrls.StandardStyle 3019
StdStyleActnCtrls.StandardStyle variable 3019
StdStyleActnCtrls.TStandardStyleActionBars 3016
StdStyleActnCtrls.TStandardStyleActionBars.GetAddRemoveItemClass 3017
StdStyleActnCtrls.TStandardStyleActionBars.GetColorMapClass 3017
StdStyleActnCtrls.TStandardStyleActionBars.GetControlClass 3017
StdStyleActnCtrls.TStandardStyleActionBars.GetPopupClass 3018
StdStyleActnCtrls.TStandardStyleActionBars.GetScrollBtnClass 3018
StdStyleActnCtrls.TStandardStyleActionBars.GetStyleName 3018
stFileSystem enumeration member 3103
stIndexes enumeration member 1701
stManual enumeration member 3104
stNonBlocking enumeration member 148
stNoSchema enumeration member 1701
Stop
IWebBrowser interface 744
IWebBrowser2Disp interface 768
IWebBrowserAppDisp interface 789
IWebBrowserDisp interface 798
TInternetExplorer class 448
TSocketService class 185
TWebBrowser class 518
TWebBrowser_V1 class 576
StoreDefs
TSimpleDataSet class 929
StoredProcName
TSQLStoredProc class 1680
StoreObject
TCustomDataBlockInterpreter class 197
TDataBlockInterpreter class 214
stPackages enumeration member 1701
stProcedureParams enumeration member 1701
stProcedures enumeration member 1701
stRaw enumeration member 1502
stRdm enumeration member 1502
Stream
IDataBlock interface 274
TDataBlock class 206
StreamLoader
TWSDLItems class 7245
StrictDelimiter
TWideStrings class 6852
StringEvent
TFTCompT class 1312
StringOfChar
AnsiStringBase class 3199
AnsiStringT template 3228
UnicodeString class 3408
WideString class 3484
Strings
TCustomStringsValuesList class 5737
TDesigntimeWarnings class 6506
TDOMStrings class 7184
TStringsValuesList class 5863
TValueListEditor class 4834
TWideStrings class 6853
StringsAdapter
TWideStrings class 6854
StringsChanging
TValueListEditor class 4800
StringsPrepared
TCustomStringsValuesList class 5737
StrUtils 3021

StrUtils namespace 3021
Constants 3060
Functions 3024
Structs, Records, Enums 3055
Types 3056
Variables 3058
StrUtils.AnsiContainsStr 3025
StrUtils.AnsiContainsStr function 3025
StrUtils.AnsiContainsText 3026
StrUtils.AnsiContainsText function 3026
StrUtils.AnsiEndsStr 3026
StrUtils.AnsiEndsStr function 3026
StrUtils.AnsiEndsText 3027
StrUtils.AnsiEndsText function 3027
StrUtils.AnsiIndexStr 3028
StrUtils.AnsiIndexStr function 3028
StrUtils.AnsiIndexText 3028
StrUtils.AnsiIndexText function 3028
StrUtils.AnsiLeftStr 3029
StrUtils.AnsiLeftStr function 3029
StrUtils.AnsiMatchStr 3029
StrUtils.AnsiMatchStr function 3029
StrUtils.AnsiMatchText 3030
StrUtils.AnsiMatchText function 3030
StrUtils.AnsiMidStr 3030
StrUtils.AnsiMidStr function 3030
StrUtils.AnsiReplaceStr 3031
StrUtils.AnsiReplaceStr function 3031
StrUtils.AnsiReplaceText 3032
StrUtils.AnsiReplaceText function 3032
StrUtils.AnsiResemblesProc 3058
StrUtils.AnsiResemblesProc variable 3058
StrUtils.AnsiResemblesText 3032
StrUtils.AnsiResemblesText function 3032
StrUtils.AnsiReverseString 3033
StrUtils.AnsiReverseString function 3033
StrUtils.AnsiRightStr 3033
StrUtils.AnsiRightStr function 3033
StrUtils.AnsiStartsStr 3034
StrUtils.AnsiStartsStr function 3034
StrUtils.AnsiStartsText 3034
StrUtils.AnsiStartsText function 3034
StrUtils.ContainsStr 3035
StrUtils.ContainsStr function 3035
StrUtils.ContainsText 3035
StrUtils.ContainsText function 3035
StrUtils.DecodeSoundexInt 3036
StrUtils.DecodeSoundexInt function 3036
StrUtils.DecodeSoundexWord 3036
StrUtils.DecodeSoundexWord function 3036
StrUtils.DupeString 3037
StrUtils.DupeString function 3037
StrUtils.EndsStr 3037
StrUtils.EndsStr function 3037
StrUtils.EndsText 3038
StrUtils.EndsText function 3038
StrUtils.IfThen 3038
StrUtils.IfThen function 3038
StrUtils.IndexStr 3039
StrUtils.IndexStr function 3039
StrUtils.IndexText 3039
StrUtils.IndexText function 3039
StrUtils.LeftBStr 3040
StrUtils.LeftBStr function 3040
StrUtils.LeftStr 3040
StrUtils.LeftStr function 3040
StrUtils.MatchStr 3041
StrUtils.MatchStr function 3041
StrUtils.MatchText 3041
StrUtils.MatchText function 3041
StrUtils.MidBStr 3042
StrUtils.MidBStr function 3042
StrUtils.MidStr 3043
StrUtils.MidStr function 3043
StrUtils.PosEx 3043
StrUtils.PosEx function 3043
StrUtils.RandomFrom 3044
StrUtils.RandomFrom function 3044

StrUtils.ReplaceStr 3044
StrUtils.ReplaceStr function 3044
StrUtils.ReplaceText 3045
StrUtils.ReplaceText function 3045
StrUtils.ResemblesProc 3058
StrUtils.ResemblesProc variable 3058
StrUtils.ResemblesText 3046
StrUtils.ResemblesText function 3046
StrUtils.ReverseString 3046
StrUtils.ReverseString function 3046
StrUtils.RightBStr 3046
StrUtils.RightBStr function 3046
StrUtils.RightStr 3047
StrUtils.RightStr function 3047
StrUtils.SearchBuf 3048
StrUtils.SearchBuf function 3048
StrUtils.Soundex 3048
StrUtils.Soundex function 3048
StrUtils.SoundexCompare 3049
StrUtils.SoundexCompare function 3049
StrUtils.SoundexInt 3050
StrUtils.SoundexInt function 3050
StrUtils.SoundexProc 3050
StrUtils.SoundexProc function 3050
StrUtils.SoundexSimilar 3051
StrUtils.SoundexSimilar function 3051
StrUtils.SoundexWord 3052
StrUtils.SoundexWord function 3052
StrUtils.StartsStr 3052
StrUtils.StartsStr function 3052
StrUtils.StartsText 3053
StrUtils.StartsText function 3053
StrUtils.StuffString 3053
StrUtils.StuffString function 3053
StrUtils.TCompareTextProc 3056
StrUtils.TCompareTextProc type 3056
StrUtils.TSoundexIntLength 3056
StrUtils.TSoundexIntLength type 3056
StrUtils.TSoundexLength 3057
StrUtils.TSoundexLength type 3057
StrUtils.TStringSeachOption 3055
StrUtils.TStringSeachOption enumeration 3055
StrUtils.TStringSearchOptions 3057
StrUtils.TStringSearchOptions type 3057
StrUtils.WordDelimiters 3060
StrUtils.WordDelimiters constant 3060
stSeqPacket enumeration member 1502
stStream enumeration member 1502
stSysTables enumeration member 1701
stSystem enumeration member 3104
stTables enumeration member 1701
stThreadBlocking enumeration member 148
stUserNames enumeration member 1701
stWin32 enumeration member 3103
Style
    TAdapterActionButton class 6323
    TAdapterCommandColumn class 6336
    TAdapterCommandGroup class 6341
    TAdapterDisplayColumn class 6369
    TAdapterDisplayField class 6379
    TAdapterEditColumn class 6395
    TAdapterFieldGroup class 6404
    TAdapterForm class 6407
    TButton class 2069
    TComboBox class 2223
    TCustomAdapterCommandColumn class 6452
    TCustomAdapterCommandGroup class 6460
    TCustomAdapterFieldGroup class 6489
    TCustomAdapterForm class 6495
    TCustomButton class 2275
    TCustomComboBox class 2337
    TCustomListBox class 2436
    TListBox class 2695
    TScreenTipBorderStyle class 327
    TTabSet class 4505
StyleRule
    TAdapterActionButton class 6323
    TAdapterCommandColumn class 6336

TAdapterCommandGroup class 6341
TAdapterDisplayColumn class 6370
TAdapterDisplayField class 6379
TAdapterEditColumn class 6395
TAdapterFieldGroup class 6404
TAdapterForm class 6407
TCustomAdapterCommandColumn class 6452
TCustomAdapterCommandGroup class 6460
TCustomAdapterFieldGroup class 6489
TCustomAdapterForm class 6496
Styles
TAdapterPageProducer class 1140
TBaseAdapterPageProducer class 1157
StylesContent
TSitePageElements class 1169
StylesFile
TAdapterPageProducer class 1141
TBaseAdapterPageProducer class 1158
stylesheet
IDOMXSLProcessor interface 7611
SubmitFormVarName
TXMLBroker class 7321
SubscriptionsEnabled
IShellFavoritesNameSpace interface 695
IShellFavoritesNameSpaceDisp interface 700
IShellNameSpaceDisp interface 716
TShellNameSpace class 482
SubString
AnsiStringBase class 3199
AnsiStringT template 3228
UnicodeString class 3408
WideString class 3484
substringData
IDOMCharacterData interface 7529
SuccessPage
IAdapterItemRequestParams interface 1009
IGetAdapterRequestDefaultResponse interface 1026
TAdapterItemRequestParams class 6165
SupportCallbacks
TSocketConnection class 229
TStreamedConnection class 248
TWebConnection class 264
supports
IDOMNode interface 7567
SupportsLanguage
TAbstractScriptEngine class 6651
svApplication enumeration member 6720
SvcMgr 3061
SvcMgr namespace 3061
Classes 3063
Constants 3110
Structs, Records, Enums 3102
Types 3106
Variables 3109
SvcMgr.Application 3109
SvcMgr.Application variable 3109
SvcMgr.CM_SERVICE_CONTROL_CODE 3110
SvcMgr.CM_SERVICE_CONTROL_CODE constant 3110
SvcMgr.TContinueEvent 3106
SvcMgr.TContinueEvent type 3106
SvcMgr.TCurrentStatus 3102
SvcMgr.TCurrentStatus enumeration 3102
SvcMgr.TDependencies 3063
SvcMgr.TDependencies.Create 3064
SvcMgr.TDependencies.GetOwner 3064
SvcMgr.TDependencies.Items 3065
SvcMgr.TDependency 3066
SvcMgr.TDependency.GetDisplayName 3067
SvcMgr.TDependency.IsGroup 3067
SvcMgr.TDependency.Name 3068
SvcMgr.TErrorSeverity 3103
SvcMgr.TErrorSeverity enumeration 3103
SvcMgr.TEventLogger 3068
SvcMgr.TEventLogger.Create 3069
SvcMgr.TEventLogger.Destroy 3070
SvcMgr.TEventLogger.LogMessage 3070
SvcMgr.TPauseEvent 3106
SvcMgr.TPauseEvent type 3106

SvcMgr.TService 3071
SvcMgr.TService.AfterInstall 3085
SvcMgr.TService.AfterUninstall 3086
SvcMgr.TService.AllowPause 3078
SvcMgr.TService.AllowStop 3078
SvcMgr.TService.BeforeInstall 3086
SvcMgr.TService.BeforeUninstall 3087
SvcMgr.TService.Controller 3074
SvcMgr.TService.CreateNew 3073
SvcMgr.TService.Dependencies 3078
SvcMgr.TService.Destroy 3073
SvcMgr.TService.DisplayName 3079
SvcMgr.TService.DoContinue 3074
SvcMgr.TService.DoCustomControl 3074
SvcMgr.TService.DoInterrogate 3074
SvcMgr.TService.DoPause 3075
SvcMgr.TService.DoShutdown 3075
SvcMgr.TService.DoStart 3075
SvcMgr.TService.DoStop 3075
SvcMgr.TService.ErrCode 3079
SvcMgr.TService.ErrorSeverity 3080
SvcMgr.TService.GetServiceController 3076
SvcMgr.TService.Interactive 3080
SvcMgr.TService.LoadGroup 3080
SvcMgr.TService.LogMessage 3076
SvcMgr.TService.Main 3077
SvcMgr.TService.OnContinue 3087
SvcMgr.TService.OnExecute 3088
SvcMgr.TService.OnPause 3088
SvcMgr.TService.OnShutdown 3088
SvcMgr.TService.OnStart 3089
SvcMgr.TService.OnStop 3089
SvcMgr.TService.Param 3081
SvcMgr.TService.ParamCount 3081
SvcMgr.TService.Password 3081
SvcMgr.TService.ReportStatus 3077
SvcMgr.TService.ServiceStartName 3082
SvcMgr.TService.ServiceThread 3082
SvcMgr.TService.ServiceType 3083
SvcMgr.TService.StartType 3083
SvcMgr.TService.Status 3083
SvcMgr.TService.TagID 3084
SvcMgr.TService.Terminated 3084
SvcMgr.TService.WaitHint 3084
SvcMgr.TService.Win32ErrCode 3085
SvcMgr.TServiceApplication 3089
SvcMgr.TServiceApplication.Create 3091
SvcMgr.TServiceApplication.CreateForm 3092
SvcMgr.TServiceApplication.DelayInitialize 3094
SvcMgr.TServiceApplication.Destroy 3091
SvcMgr.TServiceApplication.DispatchServiceMain 3092
SvcMgr.TServiceApplication.DoHandleException 3092
SvcMgr.TServiceApplication.Hook 3093
SvcMgr.TServiceApplication.Initialize 3093
SvcMgr.TServiceApplication.Installing 3093
SvcMgr.TServiceApplication.RegisterServices 3093
SvcMgr.TServiceApplication.Run 3094
SvcMgr.TServiceApplication.ServiceCount 3094
SvcMgr.TServiceApplication.Title 3095
SvcMgr.TServiceController 3107
SvcMgr.TServiceController type 3107
SvcMgr.TServiceEvent 3107
SvcMgr.TServiceEvent type 3107
SvcMgr.TServiceThread 3095
SvcMgr.TServiceThread.Create 3096
SvcMgr.TServiceThread.Execute 3097
SvcMgr.TServiceThread.ProcessRequests 3100
SvcMgr.TServiceType 3103
SvcMgr.TServiceType enumeration 3103
SvcMgr.TStartEvent 3108
SvcMgr.TStartEvent type 3108
SvcMgr.TStartType 3104
SvcMgr.TStartType enumeration 3104
SvcMgr.TStopEvent 3108
SvcMgr.TStopEvent type 3108
svEndUser enumeration member 6720
svHTMLItems enumeration member 6720
svModules enumeration member 6720

svPage enumeration member 6720
svPages enumeration member 6720
svProducer enumeration member 6720
svRequest enumeration member 6720
svResponse enumeration member 6720
svSession enumeration member 6720
swap
AnsiStringBase class 3200
AnsiStringT template 3228
UnicodeString class 3409
WideString class 3484
SwitchConnection
TSQLMonitor class 1648
Synchronize
IShellFavoritesNameSpace interface 695
IShellFavoritesNameSpaceDisp interface 699
IShellNameSpaceDisp interface 713
TShellNameSpace class 479
SyncObjs 3111
SyncObjs namespace 3111
Classes 3112
Structs, Records, Enums 3156
Types 3157
SyncObjs.TConditionVariableCS 3112
SyncObjs.TConditionVariableCS.Acquire 3115
SyncObjs.TConditionVariableCS.Create 3114
SyncObjs.TConditionVariableCS.FConditionVariable 3115
SyncObjs.TConditionVariableCS.Release 3115
SyncObjs.TConditionVariableCS.ReleaseAll 3116
SyncObjs.TConditionVariableCS.WaitFor 3116
SyncObjs.TConditionVariableHelper 3117
SyncObjs.TConditionVariableHelper.Create 3118
SyncObjs.TConditionVariableHelper.Free 3119
SyncObjs.TConditionVariableHelper.SleepCS 3119
SyncObjs.TConditionVariableHelper.Wake 3120
SyncObjs.TConditionVariableHelper.WakeAll 3120
SyncObjs.TConditionVariableMutex 3120
SyncObjs.TConditionVariableMutex.Acquire 3123
SyncObjs.TConditionVariableMutex.Create 3122
SyncObjs.TConditionVariableMutex.Destroy 3123
SyncObjs.TConditionVariableMutex.Release 3124
SyncObjs.TConditionVariableMutex.ReleaseAll 3124
SyncObjs.TConditionVariableMutex.WaitFor 3124
SyncObjs.TCriticalSection 3125
SyncObjs.TCriticalSection.Acquire 3128
SyncObjs.TCriticalSection.Create 3127
SyncObjs.TCriticalSection.Destroy 3127
SyncObjs.TCriticalSection.Enter 3128
SyncObjs.TCriticalSection.FSection 3128
SyncObjs.TCriticalSection.Leave 3129
SyncObjs.TCriticalSection.Release 3129
SyncObjs.TCriticalSection.TryEnter 3130
SyncObjs.TCriticalSectionHelper 3130
SyncObjs.TCriticalSectionHelper.Destroy 3131
SyncObjs.TCriticalSectionHelper.Enter 3132
SyncObjs.TCriticalSectionHelper.Free 3132
SyncObjs.TCriticalSectionHelper.Initialize 3132
SyncObjs.TCriticalSectionHelper.Leave 3133
SyncObjs.TCriticalSectionHelper.TryEnter 3133
SyncObjs.TEvent 3133
SyncObjs.TEvent.Create 3135
SyncObjs.TEvent.ResetEvent 3136
SyncObjs.TEvent.SetEvent 3137
SyncObjs.THandleObject 3137
SyncObjs.THandleObject.Create 3139
SyncObjs.THandleObject.Destroy 3139
SyncObjs.THandleObject.FHandle 3140
SyncObjs.THandleObject.FLastError 3140
SyncObjs.THandleObject.FUseCOMWait 3140
SyncObjs.THandleObject.Handle 3142
SyncObjs.THandleObject.LastError 3142
SyncObjs.THandleObject.WaitFor 3141
SyncObjs.THandleObject.WaitForMultiple 3141
SyncObjs.THandleObjectArray 3157
SyncObjs.THandleObjectArray type 3157
SyncObjs.TMutex 3143
SyncObjs.TMutex.Acquire 3146
SyncObjs.TMutex.Create 3145

SyncObjs.TMutex.Release 3146
SyncObjs.TSemaphore 3147
SyncObjs.TSemaphore.Acquire 3151
SyncObjs.TSemaphore.Create 3149
SyncObjs.TSemaphore.Release 3151
SyncObjs.TSimpleEvent 3152
SyncObjs.TSynchroObject 3152
SyncObjs.TSynchroObject.Acquire 3154
SyncObjs.TSynchroObject.Release 3154
SyncObjs.TWaitResult 3156
SyncObjs.TWaitResult enumeration 3156
SyncProc
TClientSocketThread class 1398
System 3159
System namespace 3159
Classes 3174
Constants 3842
Functions 3506
Interfaces 3493
Macros 3892
Structs, Records, Enums 3712
Types 3750
Variables 3806
System.Abs 3511
System.Abs function 3511
System.AbstractErrorProc 3807
System.AbstractErrorProc variable 3807
System.AcquireExceptionObject 3512
System.AcquireExceptionObject function 3512
System.AddModuleUnloadProc 3514
System.AddModuleUnloadProc function 3514
System.Addr 3514
System.Addr function 3514
System.AllocMem 3515
System.AllocMem function 3515
System.AllocMemCount 3807
System.AllocMemCount variable 3807
System.AllocMemSize 3809
System.AllocMemSize variable 3809
System.AnsiToUtf8 3517
System.AnsiToUtf8 function 3517
System.Append 3520
System.Append function 3520
System.ArcTan 3521
System.ArcTan function 3521
System.Assert 3522
System.Assert function 3522
System.AssertErrorProc 3810
System.AssertErrorProc variable 3810
System.Assigned 3524
System.Assigned function 3524
System.AssignFile 3525
System.AssignFile function 3525
System.AttemptToUseSharedMemoryManager 3527
System.AttemptToUseSharedMemoryManager function 3527
System.BeginThread 3527
System.BeginThread function 3527
System.BlockRead 3528
System.BlockRead function 3528
System.BlockWrite 3530
System.BlockWrite function 3530
System.Break 3531
System.Break function 3531
System.ChDir 3534
System.ChDir function 3534
System.Chr 3536
System.Chr function 3536
System.Close 3538
System.Close function 3538
System.CloseFile 3539
System.CloseFile function 3539
System.CmdLine 3812
System.CmdLine variable 3812
System.CmdShow 3812
System.CmdShow variable 3812
System.CompilerVersion 3844
System.CompilerVersion constant 3844
System.CompToCurrency 3542

System.CompToCurrency function 3542
System.CompToDouble 3543
System.CompToDouble function 3543
System.Concat 3545
System.Concat function 3545
System.Continue 3546
System.Continue function 3546
System.Copy 3547
System.Copy function 3547
System.Cos 3549
System.Cos function 3549
System.CppLongInt 3752
System.CppLongInt type 3752
System.CppULongInt 3753
System.CppULongInt type 3753
System.CPUCount 3813
System.CPUCount variable 3813
System.CPUi386 3844
System.CPUi386 constant 3844
System.CPUi486 3845
System.CPUi486 constant 3845
System.CPUPentium 3845
System.CPUPentium constant 3845
System.CurrencyToComp 3550
System.CurrencyToComp function 3550
System.DebugHook 3813
System.DebugHook variable 3813
System.Dec 3550
System.Dec function 3550
System.Default8087CW 3813
System.Default8087CW variable 3813
System.DefaultSystemCodePage 3815
System.DefaultSystemCodePage variable 3815
System.DefaultTextLineBreakStyle 3815
System.DefaultTextLineBreakStyle variable 3815
System.DefaultUnicodeCodePage 3816
System.DefaultUnicodeCodePage variable 3816
System.Delete 3552
System.Delete function 3552
System.DispCallByIDProc 3816
System.DispCallByIDProc variable 3816
System.Dispose 3553
System.Dispose function 3553
System.DLLProc 3816
System.DLLProc variable 3816
System.DoubleToComp 3555
System.DoubleToComp function 3555
System.DynArrayClear 3555
System.DynArrayClear function 3555
System.DynArrayDim 3555
System.DynArrayDim function 3555
System.DynArraySetLength 3556
System.DynArraySetLength function 3556
System.E_NOINTERFACE 3845
System.E_NOINTERFACE constant 3845
System.E_NOTIMPL 3846
System.E_NOTIMPL constant 3846
System.E_UNEXPECTED 3846
System.E_UNEXPECTED constant 3846
System.EndThread 3556
System.EndThread function 3556
System.EnumModules 3556
System.EnumModules function 3556
System.EnumResourceModules 3557
System.EnumResourceModules function 3557
System.envp 3817
System.envp variable 3817
System.Eof 3558
System.Eof function 3558
System.Eoln 3559
System.Eoln function 3559
System.Erase 3560
System.Erase function 3560
System.Error 3560
System.Error function 3560
System.ErrorAddr 3817
System.ErrorAddr variable 3817
System.ErrorProc 3818

System.ErrorProc variable 3818
System.ErrOutput 3820
System.ErrOutput variable 3820
System.ExceptAddr 3562
System.ExceptAddr function 3562
System.ExceptClsProc 3820
System.ExceptClsProc variable 3820
System.ExceptionAcquired 3821
System.ExceptionAcquired variable 3821
System.ExceptionClass 3821
System.ExceptionClass variable 3821
System.ExceptObject 3562
System.ExceptObject function 3562
System.ExceptObjProc 3821
System.ExceptObjProc variable 3821
System.ExceptProc 3822
System.ExceptProc variable 3822
System.Exclude 3564
System.Exclude function 3564
System.Exit 3565
System.Exit function 3565
System.ExitCode 3823
System.ExitCode variable 3823
System.ExitProc 3823
System.ExitProc variable 3823
System.ExitProcessProc 3824
System.ExitProcessProc variable 3824
System.Exp 3567
System.Exp function 3567
System.FileMode 3824
System.FileMode variable 3824
System.FilePos 3567
System.FilePos function 3567
System.FileSize 3569
System.FileSize function 3569
System.FillChar 3570
System.FillChar function 3570
System.Finalize 3572
System.Finalize function 3572
System.FindClassHInstance 3573
System.FindClassHInstance function 3573
System.FindHInstance 3574
System.FindHInstance function 3574
System.FindResourceHInstance 3574
System.FindResourceHInstance function 3574
System.Flush 3575
System.Flush function 3575
System.fmClosed 3846
System.fmClosed constant 3846
System.fmInOut 3847
System.fmInOut constant 3847
System.fmInput 3847
System.fmInput constant 3847
System.fmOutput 3847
System.fmOutput constant 3847
System.FPower10 3576
System.FPower10 function 3576
System.Frac 3576
System.Frac function 3576
System.FreeMem 3577
System.FreeMem function 3577
System.FreeMemory 3579
System.FreeMemory function 3579
System.Get8087CW 3580
System.Get8087CW function 3580
System.GetDir 3580
System.GetDir function 3580
System.GetHeapStatus 3581
System.GetHeapStatus function 3581
System.GetLastError 3582
System.GetLastError function 3582
System.GetMem 3584
System.GetMem function 3584
System.GetMemory 3585
System.GetMemory function 3585
System.GetMemoryManager 3587
System.GetMemoryManager function 3587
System.GetMemoryManagerState 3591

System.GetMemoryManagerState function 3591
System.GetMemoryMap 3592
System.GetMemoryMap function 3592
System.GetMinimumBlockAlignment 3592
System.GetMinimumBlockAlignment function 3592
System.GetModuleFileName 3593
System.GetModuleFileName function 3593
System.GetPackageInfoTable 3753
System.GetPackageInfoTable type 3753
System.GetVariantManager 3594
System.GetVariantManager function 3594
System.Halt 3594
System.Halt function 3594
System.HeapAllocFlags 3825
System.HeapAllocFlags variable 3825
System.hfFieldSize 3848
System.hfFieldSize constant 3848
System.hfMonitorOffset 3848
System.hfMonitorOffset constant 3848
System.HGLOBAL 3753
System.HGLOBAL type 3753
System.Hi 3595
System.Hi function 3595
System.High 3596
System.High function 3596
System.HINST 3754
System.HINST type 3754
System.HMODULE 3754
System.HMODULE type 3754
System.HPrevInst 3826
System.HPrevInst variable 3826
System.HRESULT 3754
System.HRESULT type 3754
System.HRSRC 3755
System.HRSRC type 3755
System.IComparable 3493
System.IComparable.CompareTo 3494
System.IDispatch 3494
System.IDispatch.GetIDsOfNames 3495
System.IDispatch.GetTypeInfo 3496
System.IDispatch.GetTypeInfoCount 3496
System.IDispatch.Invoke 3497
System.IEnumerable 3497
System.IEnumerable.GetEnumerator 3498
System.IEnumerator 3498
System.IEnumerator.Current 3500
System.IEnumerator.GetCurrent 3499
System.IEquatable 3500
System.IEquatable.Equals 3501
System.IInterface 3502
System.IInterface._AddRef 3503
System.IInterface._Release 3503
System.IInterface.QueryInterface 3504
System.IInvokable 3504
System.Inc 3597
System.Inc function 3597
System.Include 3598
System.Include function 3598
System.INFINITE 3848
System.INFINITE constant 3848
System.Initialize 3599
System.Initialize function 3599
System.InitProc 3826
System.InitProc variable 3826
System.Input 3827
System.Input variable 3827
System.Insert 3601
System.Insert function 3601
System.Int 3602
System.Int function 3602
System.Int16 3755
System.Int16 type 3755
System.Int32 3755
System.Int32 type 3755
System.Int8 3756
System.Int8 type 3756
System.IntegerArray 3756
System.IntegerArray type 3756

System.IOResult 3602
System.IOResult function 3602
System.IsConsole 3828
System.IsConsole variable 3828
System.IsLibrary 3828
System.IsLibrary variable 3828
System.IsMemoryManagerSet 3604
System.IsMemoryManagerSet function 3604
System.IsMultiThread 3829
System.IsMultiThread variable 3829
System.IsVariantManagerSet 3606
System.IsVariantManagerSet function 3606
System.IUnknown 3756
System.IUnknown type 3756
System.JITEnable 3829
System.JITEnable variable 3829
System.Length 3607
System.Length function 3607
System.LibModuleList 3830
System.LibModuleList variable 3830
System.Ln 3608
System.Ln function 3608
System.Lo 3609
System.Lo function 3609
System.LoadResourceModule 3609
System.LoadResourceModule function 3609
System.LoadResString 3610
System.LoadResString function 3610
System.Low 3610
System.Low function 3610
System.MainInstance 3830
System.MainInstance variable 3830
System.MainThreadID 3831
System.MainThreadID variable 3831
System.Mark 3612
System.Mark function 3612
System.MaxInt 3849
System.MaxInt constant 3849
System.MaxLongint 3849
System.MaxLongint constant 3849
System.MemoryBarrier 3612
System.MemoryBarrier function 3612
System.MkDir 3612
System.MkDir function 3612
System.ModuleUnloadList 3831
System.ModuleUnloadList variable 3831
System.MonitorEnter 3614
System.MonitorEnter function 3614
System.MonitorExit 3614
System.MonitorExit function 3614
System.MonitorPulse 3615
System.MonitorPulse function 3615
System.MonitorPulseAll 3615
System.MonitorPulseAll function 3615
System.MonitorSupport 3831
System.MonitorSupport variable 3831
System.MonitorTryEnter 3616
System.MonitorTryEnter function 3616
System.MonitorWait 3616
System.MonitorWait function 3616
System.Move 3617
System.Move function 3617
System.MoveChars 3618
System.MoveChars function 3618
System.NeverSleepOnMMThreadContention 3832
System.NeverSleepOnMMThreadContention variable 3832
System.New 3619
System.New function 3619
System.NoErrMsg 3832
System.NoErrMsg variable 3832
System.NumSmallBlockTypes 3849
System.NumSmallBlockTypes constant 3849
System.Odd 3621
System.Odd function 3621
System.OleStrToString 3622
System.OleStrToString function 3622
System.OleStrToStrVar 3622
System.OleStrToStrVar function 3622

System.opAdd 3850
System.opAdd constant 3850
System.opAnd 3850
System.opAnd constant 3850
System.opCmpEQ 3851
System.opCmpEQ constant 3851
System.opCmpGE 3851
System.opCmpGE constant 3851
System.opCmpGT 3852
System.opCmpGT constant 3852
System.opCmpLE 3852
System.opCmpLE constant 3852
System.opCmpLT 3853
System.opCmpLT constant 3853
System.opCmpNE 3853
System.opCmpNE constant 3853
System.opCompare 3854
System.opCompare constant 3854
System.opDivide 3854
System.opDivide constant 3854
System.opIntDivide 3855
System.opIntDivide constant 3855
System.opModulus 3855
System.opModulus constant 3855
System.opMultiply 3855
System.opMultiply constant 3855
System.opNegate 3856
System.opNegate constant 3856
System.opNot 3856
System.opNot constant 3856
System.opOr 3857
System.opOr constant 3857
System.opShiftLeft 3857
System.opShiftLeft constant 3857
System.opShiftRight 3858
System.opShiftRight constant 3858
System.opSubtract 3858
System.opSubtract constant 3858
System.opXor 3859
System.opXor constant 3859
System.Ord 3623
System.Ord function 3623
System.Output 3833
System.Output variable 3833
System.PackageInfo 3757
System.PackageInfo type 3757
System.PackageInfoTable 3713
System.PackageInfoTable record 3713
System.PackageUnitEntry 3713
System.PackageUnitEntry record 3713
System.PAnsiString 3757
System.PAnsiString type 3757
System.ParamCount 3623
System.ParamCount function 3623
System.ParamStr 3625
System.ParamStr function 3625
System.PBoolean 3758
System.PBoolean type 3758
System.PByte 3758
System.PByte type 3758
System.PCardinal 3759
System.PCardinal type 3759
System.PComp 3759
System.PComp type 3759
System.PCurrency 3759
System.PCurrency type 3759
System.PCVModInfo 3760
System.PCVModInfo type 3760
System.PDate 3760
System.PDate type 3760
System.PDateTime 3761
System.PDateTime type 3761
System.PDispatch 3761
System.PDispatch type 3761
System.PDouble 3762
System.PDouble type 3762
System.PDynArrayTypeInfo 3762
System.PDynArrayTypeInfo type 3762

System.PError 3762
System.PError type 3762
System.PExtended 3763
System.PExtended type 3763
System.PGUID 3763
System.PGUID type 3763
System.Pi 3626
System.Pi function 3626
System.PInitContext 3765
System.PInitContext type 3765
System.PInt64 3765
System.PInt64 type 3765
System.PInteger 3765
System.PInteger type 3765
System.PIntegerArray 3766
System.PIntegerArray type 3766
System.PInterfaceEntry 3766
System.PInterfaceEntry type 3766
System.PInterfaceTable 3766
System.PInterfaceTable type 3766
System.PLibModule 3767
System.PLibModule type 3767
System.PLongint 3767
System.PLongint type 3767
System.PLongWord 3768
System.PLongWord type 3768
System.PMonitor 3769
System.PMonitor type 3769
System.PMonitorSupport 3769
System.PMonitorSupport type 3769
System.PointerArray 3770
System.PointerArray type 3770
System.POleVariant 3770
System.POleVariant type 3770
System.Pos 3627
System.Pos function 3627
System.PPAnsiChar 3770
System.PPAnsiChar type 3770
System.PPChar 3771
System.PPChar type 3771
System.PPCharArray 3771
System.PPCharArray type 3771
System.PPDispatch 3771
System.PPDispatch type 3771
System.PPMonitor 3772
System.PPMonitor type 3772
System.PPointer 3772
System.PPointer type 3772
System.PPointerArray 3772
System.PPointerArray type 3772
System.PPUnknown 3773
System.PPUnknown type 3773
System.PPWideChar 3773
System.PPWideChar type 3773
System.PRawByteString 3773
System.PRawByteString type 3773
System.Pred 3628
System.Pred function 3628
System.PShortInt 3774
System.PShortInt type 3774
System.PShortString 3774
System.PShortString type 3774
System.PSingle 3775
System.PSingle type 3775
System.PSmallInt 3775
System.PSmallInt type 3775
System.PString 3776
System.PString type 3776
System.PTextBuf 3776
System.PTextBuf type 3776
System.Ptr 3630
System.Ptr function 3630
System.PUCS2Char 3776
System.PUCS2Char type 3776
System.PUCS4Char 3777
System.PUCS4Char type 3777
System.PUCS4CharArray 3777
System.PUCS4CharArray type 3777

System.PUCS4Chars 3632
System.PUCS4Chars function 3632
System.PUInt64 3777
System.PUInt64 type 3777
System.PUnicodeString 3778
System.PUnicodeString type 3778
System.PUnitEntryTable 3778
System.PUnitEntryTable type 3778
System.PUnknown 3778
System.PUnknown type 3778
System.PUTF8String 3779
System.PUTF8String type 3779
System.PVarArray 3779
System.PVarArray type 3779
System.PVarArrayBoundArray 3779
System.PVarArrayBoundArray type 3779
System.PVarArrayCoorArray 3780
System.PVarArrayCoorArray type 3780
System.PVarData 3780
System.PVarData type 3780
System.PVariant 3780
System.PVariant type 3780
System.PWideString 3782
System.PWideString type 3782
System.PWord 3782
System.PWord type 3782
System.PWordBool 3782
System.PWordBool type 3782
System.RaiseExceptionProc 3833
System.RaiseExceptionProc variable 3833
System.RaiseExceptObjProc 3834
System.RaiseExceptObjProc variable 3834
System.RaiseList 3632
System.RaiseList function 3632
System.Random 3632
System.Random function 3632
System.Randomize 3633
System.Randomize function 3633
System.RandSeed 3834
System.RandSeed variable 3834
System.RawByteString 3783
System.RawByteString type 3783
System.Read 3634
System.Read function 3634
System.ReadLn 3636
System.ReadLn function 3636
System.ReallocMem 3637
System.ReallocMem function 3637
System.ReallocMemory 3638
System.ReallocMemory function 3638
System.RegisterExpectedMemoryLeak 3640
System.RegisterExpectedMemoryLeak function 3640
System.RegisterModule 3640
System.RegisterModule function 3640
System.Release 3641
System.Release function 3641
System.ReleaseExceptionObject 3641
System.ReleaseExceptionObject function 3641
System.RemoveModuleUnloadProc 3641
System.RemoveModuleUnloadProc function 3641
System.Rename 3642
System.Rename function 3642
System.ReportMemoryLeaksOnShutdown 3835
System.ReportMemoryLeaksOnShutdown variable 3835
System.Reset 3643
System.Reset function 3643
System.Rewrite 3645
System.Rewrite function 3645
System.RmDir 3647
System.RmDir function 3647
System.Round 3648
System.Round function 3648
System.RTLUnwindProc 3835
System.RTLUnwindProc variable 3835
System.RTLVersion 3859
System.RTLVersion constant 3859
System.RunError 3649
System.RunError function 3649

System.S_FALSE 3859
System.S_FALSE constant 3859
System.S_OK 3860
System.S_OK constant 3860
System.SafeCallErrorProc 3836
System.SafeCallErrorProc variable 3836
System.Seek 3650
System.Seek function 3650
System.SeekEof 3651
System.SeekEof function 3651
System.SeekEoln 3652
System.SeekEoln function 3652
System.Set8087CW 3653
System.Set8087CW function 3653
System.SetCodePage 3655
System.SetCodePage function 3655
System.SetInOutRes 3655
System.SetInOutRes function 3655
System.SetLastError 3656
System.SetLastError function 3656
System.SetLength 3657
System.SetLength function 3657
System.SetLineBreakStyle 3659
System.SetLineBreakStyle function 3659
System.SetMemoryManager 3660
System.SetMemoryManager function 3660
System.SetMinimumBlockAlignment 3664
System.SetMinimumBlockAlignment function 3664
System.SetMultiByteConversionCodePage 3665
System.SetMultiByteConversionCodePage function 3665
System.SetRaiseList 3665
System.SetRaiseList function 3665
System.SetString 3665
System.SetString function 3665
System.SetTextBuf 3667
System.SetTextBuf function 3667
System.SetVariantManager 3668
System.SetVariantManager function 3668
System.ShareMemoryManager 3668
System.ShareMemoryManager function 3668
System.Sin 3669
System.Sin function 3669
System.SizeOf 3670
System.SizeOf function 3670
System.Slice 3671
System.Slice function 3671
System.sLineBreak 3860
System.sLineBreak constant 3860
System.Sqr 3674
System.Sqr function 3674
System.Sqrt 3675
System.Sqrt function 3675
System.Str 3676
System.Str function 3676
System.StringCodePage 3676
System.StringCodePage function 3676
System.StringElementSize 3677
System.StringElementSize function 3677
System.StringOfChar 3677
System.StringOfChar function 3677
System.StringRefCount 3678
System.StringRefCount function 3678
System.StringToOleStr 3678
System.StringToOleStr function 3678
System.StringToWideChar 3679
System.StringToWideChar function 3679
System.Succ 3680
System.Succ function 3680
System.Swap 3680
System.Swap function 3680
System.SysAllocMem 3681
System.SysAllocMem function 3681
System.SysFreeMem 3681
System.SysFreeMem function 3681
System.SysGetMem 3682
System.SysGetMem function 3682
System.SysReallocMem 3683
System.SysReallocMem function 3683

System.SysRegisterExpectedMemoryLeak 3683
System.SysRegisterExpectedMemoryLeak function 3683
System.SystemThreadEndProc 3836
System.SystemThreadEndProc variable 3836
System.SystemThreadFuncProc 3837
System.SystemThreadFuncProc variable 3837
System.SysUnregisterExpectedMemoryLeak 3684
System.SysUnregisterExpectedMemoryLeak function 3684
System.TAssertErrorProc 3783
System.TAssertErrorProc type 3783
System.TBoundArray 3784
System.TBoundArray type 3784
System.TCallDesc 3714
System.TCallDesc record 3714
System.TChunkStatus 3714
System.TChunkStatus enumeration 3714
System.TClass 3785
System.TClass type 3785
System.TClassHelperBase 3318
System.TClassHelperBase._Create 3319
System.TClassHelperBase.FInstance 3320
System.TClassHelperBaseClass 3786
System.TClassHelperBaseClass type 3786
System.TCVModInfo 3715
System.TCVModInfo record 3715
System.TDate 3786
System.TDate type 3786
System.TDateTime 3787
System.TDateTime type 3787
System.TDispatchMessage 3716
System.TDispatchMessage record 3716
System.TDispDesc 3716
System.TDispDesc record 3716
System.TDLLProc 3788
System.TDLLProc type 3788
System.TDLLProcEx 3788
System.TDLLProcEx type 3788
System.TDynArrayTypeInfo 3716
System.TDynArrayTypeInfo record 3716
System.TEnumModuleFunc 3789
System.TEnumModuleFunc type 3789
System.TEnumModuleFuncLW 3789
System.TEnumModuleFuncLW type 3789
System.Test8086 3837
System.Test8086 variable 3837
System.Test8087 3837
System.Test8087 variable 3837
System.TestFDIV 3838
System.TestFDIV variable 3838
System.TextStart 3684
System.TextStart function 3684
System.tfCRLF 3861
System.tfCRLF constant 3861
System.TFileIOFunc 3789
System.TFileIOFunc type 3789
System.TFileRec 3717
System.TFileRec record 3717
System.TGUID 3718
System.TGUID record 3718
System.THandle 3790
System.THandle type 3790
System.TInitContext 3721
System.TInitContext record 3721
System.TInterfacedClass 3791
System.TInterfacedClass type 3791
System.TInterfaceEntry 3722
System.TInterfaceEntry record 3722
System.TInterfaceTable 3723
System.TInterfaceTable record 3723
System.TMemoryMap 3791
System.TMemoryMap type 3791
System.TMethod 3731
System.TMethod record 3731
System.TMinimumBlockAlignment 3732
System.TMinimumBlockAlignment enumeration 3732
System.TModuleUnloadProc 3792
System.TModuleUnloadProc type 3792
System.TModuleUnloadProcLW 3793

System.TModuleUnloadProcLW type 3793
System.TMonitor 3349
System.TMonitor.Enter 3350, 3351
System.TMonitor.Exit 3351
System.TMonitor.Pulse 3352
System.TMonitor.PulseAll 3352
System.TMonitor.SetSpinCount 3353
System.TMonitor.TryEnter 3353
System.TMonitor.Wait 3353
System.TMonitorSupport 3733
System.TMonitorSupport record 3733
System.TPCharArray 3793
System.TPCharArray type 3793
System.TResourceHandle 3793
System.TResourceHandle type 3793
System.Trunc 3684
System.Trunc function 3684
System.Truncate 3685
System.Truncate function 3685
System.TRuntimeError 3734
System.TRuntimeError enumeration 3734
System.TSafeCallErrorProc 3794
System.TSafeCallErrorProc type 3794
System.TSystemThreadEndProc 3796
System.TSystemThreadEndProc type 3796
System.TSystemThreadFuncProc 3796
System.TSystemThreadFuncProc type 3796
System.TTextBuf 3797
System.TTextBuf type 3797
System.TTextIOFunc 3797
System.TTextIOFunc type 3797
System.TTextLineBreakStyle 3737
System.TTextLineBreakStyle enumeration 3737
System.TTextRec 3738
System.TTextRec record 3738
System.TTime 3798
System.TTime type 3798
System.TUCS4CharArray 3798
System.TUCS4CharArray type 3798
System.TVarArrayBoundArray 3798
System.TVarArrayBoundArray type 3798
System.TVarArrayCoorArray 3799
System.TVarArrayCoorArray type 3799
System.TVarData 3740
System.TVarData record 3740
System.TVarOp 3799
System.TVarOp type 3799
System.TVarType 3801
System.TVarType type 3801
System.UCS2Char 3802
System.UCS2Char type 3802
System.UCS4Char 3802
System.UCS4Char type 3802
System.UCS4String 3802
System.UCS4String type 3802
System.UCS4StringToUnicodeString 3686
System.UCS4StringToUnicodeString function 3686
System.UCS4StringToWideString 3687
System.UCS4StringToWideString function 3687
System.UInt16 3803
System.UInt16 type 3803
System.UInt32 3803
System.UInt32 type 3803
System.UInt8 3804
System.UInt8 type 3804
System.UnicodeStringToUCS4String 3689
System.UnicodeStringToUCS4String function 3689
System.UnicodeToUtf8 3690
System.UnicodeToUtf8 function 3690
System.UniqueString 3693
System.UniqueString function 3693
System.UnitEntryTable 3804
System.UnitEntryTable type 3804
System.UnregisterExpectedMemoryLeak 3694
System.UnregisterExpectedMemoryLeak function 3694
System.UnregisterModule 3694
System.UnregisterModule function 3694
System.UpCase 3695

System.UpCase function 3695
System.UTF8CompareLocale 3838
System.UTF8CompareLocale variable 3838
System.UTF8Decode 3696
System.UTF8Decode function 3696
System.UTF8Encode 3696
System.UTF8Encode function 3696
System.UTF8EncodeToShortString 3697
System.UTF8EncodeToShortString function 3697
System.UTF8String 3804
System.UTF8String type 3804
System.Utf8ToAnsi 3698
System.Utf8ToAnsi function 3698
System.UTF8ToString 3698
System.UTF8ToString function 3698
System.Utf8ToUnicode 3699
System.Utf8ToUnicode function 3699
System.UTF8ToUnicodeString 3700
System.UTF8ToUnicodeString function 3700
System.UTF8ToWideString 3700
System.UTF8ToWideString function 3700
System.Val 3701
System.Val function 3701
System.VarAddRefProc 3838
System.VarAddRefProc variable 3838
System.varAny 3861
System.varAny constant 3861
System.varArray 3861
System.varArray constant 3861
System.VarArrayRedim 3702
System.VarArrayRedim function 3702
System.varBoolean 3862
System.varBoolean constant 3862
System.varByRef 3862
System.varByRef constant 3862
System.varByte 3863
System.varByte constant 3863
System.VarClear 3704
System.VarClear function 3704
System.VarClearProc 3839
System.VarClearProc variable 3839
System.VarCopyProc 3839
System.VarCopyProc variable 3839
System.varCurrency 3863
System.varCurrency constant 3863
System.varDate 3864
System.varDate constant 3864
System.varDispatch 3864
System.varDispatch constant 3864
System.varDouble 3865
System.varDouble constant 3865
System.varEmpty 3865
System.varEmpty constant 3865
System.varError 3866
System.varError constant 3866
System.varInt64 3867
System.varInt64 constant 3867
System.varInteger 3867
System.varInteger constant 3867
System.varLongWord 3868
System.varLongWord constant 3868
System.varNull 3868
System.varNull constant 3868
System.varOleStr 3869
System.varOleStr constant 3869
System.varShortInt 3869
System.varShortInt constant 3869
System.varSingle 3870
System.varSingle constant 3870
System.varSmallint 3870
System.varSmallint constant 3870
System.varStrArg 3871
System.varStrArg constant 3871
System.varString 3871
System.varString constant 3871
System.VarToLStrProc 3840
System.VarToLStrProc variable 3840
System.VarToWStrProc 3840

System.VarToWStrProc variable 3840
System.varTypeMask 3872
System.varTypeMask constant 3872
System.varUInt64 3872
System.varUInt64 constant 3872
System.varUnknown 3873
System.varUnknown constant 3873
System.varUString 3873
System.varUString constant 3873
System.varVariant 3873
System.varVariant constant 3873
System.varWord 3874
System.varWord constant 3874
System.vmtAddRef 3874
System.vmtAddRef constant 3874
System.vmtAfterConstruction 3875
System.vmtAfterConstruction constant 3875
System.vmtAutoTable 3875
System.vmtAutoTable constant 3875
System.vmtBeforeDestruction 3875
System.vmtBeforeDestruction constant 3875
System.vmtClassName 3876
System.vmtClassName constant 3876
System.vmtCreateObject 3876
System.vmtCreateObject constant 3876
System.vmtDefaultHandler 3876
System.vmtDefaultHandler constant 3876
System.vmtDestroy 3877
System.vmtDestroy constant 3877
System.vmtDispatch 3877
System.vmtDispatch constant 3877
System.vmtDynamicTable 3877
System.vmtDynamicTable constant 3877
System.vmtEquals 3878
System.vmtEquals constant 3878
System.vmtFieldTable 3878
System.vmtFieldTable constant 3878
System.vmtFreeInstance 3878
System.vmtFreeInstance constant 3878
System.vmtGetHashCode 3879
System.vmtGetHashCode constant 3879
System.vmtInitTable 3879
System.vmtInitTable constant 3879
System.vmtInstanceSize 3879
System.vmtInstanceSize constant 3879
System.vmtIntfTable 3880
System.vmtIntfTable constant 3880
System.vmtMethodTable 3880
System.vmtMethodTable constant 3880
System.vmtNewInstance 3880
System.vmtNewInstance constant 3880
System.vmtParent 3881
System.vmtParent constant 3881
System.vmtQueryInterface 3881
System.vmtQueryInterface constant 3881
System.vmtRelease 3881
System.vmtRelease constant 3881
System.vmtSafeCallException 3882
System.vmtSafeCallException constant 3882
System.vmtSelfPtr 3882
System.vmtSelfPtr constant 3882
System.vmtToString 3882
System.vmtToString constant 3882
System.vmtTypeInfo 3883
System.vmtTypeInfo constant 3883
System.vtAnsiString 3883
System.vtAnsiString constant 3883
System.vtBoolean 3883
System.vtBoolean constant 3883
System.vtChar 3884
System.vtChar constant 3884
System.vtClass 3884
System.vtClass constant 3884
System.vtCurrency 3885
System.vtCurrency constant 3885
System.vtExtended 3885
System.vtExtended constant 3885
System.vtInt64 3886

System.vtInt64 constant 3886
System.vtInteger 3886
System.vtInteger constant 3886
System.vtInterface 3887
System.vtInterface constant 3887
System.vtObject 3887
System.vtObject constant 3887
System.vtPChar 3888
System.vtPChar constant 3888
System.vtPointer 3888
System.vtPointer constant 3888
System.vtPWideChar 3889
System.vtPWideChar constant 3889
System.vtString 3889
System.vtString constant 3889
System.vtUnicodeString 3890
System.vtUnicodeString constant 3890
System.vtVariant 3890
System.vtVariant constant 3890
System.vtWideChar 3890
System.vtWideChar constant 3890
System.vtWideString 3891
System.vtWideString constant 3891
System.WideCharLenToString 3705
System.WideCharLenToString function 3705
System.WideCharLenToStrVar 3706
System.WideCharLenToStrVar function 3706
System.WideCharToString 3706
System.WideCharToString function 3706
System.WideCharToStrVar 3707
System.WideCharToStrVar function 3707
System.WideStringToUCS4String 3707
System.WideStringToUCS4String function 3707
System.Write 3707
System.Write function 3707
System.WriteLn 3710
System.WriteLn function 3710
System.YieldProcessor 3711
System.YieldProcessor function 3711
System::AnsiStringBase 3174
System::AnsiStringBase::!= 3204
System::AnsiStringBase::[] 3205
System::AnsiStringBase::_AnsiCat 3184
System::AnsiStringBase::_AnsiFromPWChar 3184
System::AnsiStringBase::~AnsiStringBase 3178
System::AnsiStringBase::+ 3205
System::AnsiStringBase::< 3205
System::AnsiStringBase::<= 3206
System::AnsiStringBase::= 3206
System::AnsiStringBase::== 3206
System::AnsiStringBase::> 3207
System::AnsiStringBase::>= 3207
System::AnsiStringBase::AnsiCompare 3185
System::AnsiStringBase::AnsiCompareIC 3185
System::AnsiStringBase::AnsiLastChar 3186
System::AnsiStringBase::AnsiPos 3186
System::AnsiStringBase::AnsiStringBase 3179
System::AnsiStringBase::BytesOf 3186
System::AnsiStringBase::ByteType 3187
System::AnsiStringBase::cat_vprintf 3187
System::AnsiStringBase::CodePage 3188
System::AnsiStringBase::CurrToStr 3188
System::AnsiStringBase::CurrToStrF 3188
System::AnsiStringBase::Data 3182
System::AnsiStringBase::Delete 3189
System::AnsiStringBase::ElementSize 3189
System::AnsiStringBase::EnsureAnsi 3189
System::AnsiStringBase::FloatToStrF 3190
System::AnsiStringBase::FmtLoadStr 3190
System::AnsiStringBase::Format 3191
System::AnsiStringBase::FormatFloat 3191
System::AnsiStringBase::GetRec 3193
System::AnsiStringBase::Insert 3193
System::AnsiStringBase::IntToHex 3194
System::AnsiStringBase::IsDelimiter 3194
System::AnsiStringBase::IsEmpty 3194
System::AnsiStringBase::IsLeadByte 3195
System::AnsiStringBase::IsPathDelimiter 3195

System::AnsiStringBase::IsTrailByte 3195
System::AnsiStringBase::LastChar 3196
System::AnsiStringBase::LastDelimiter 3196
System::AnsiStringBase::Length 3196
System::AnsiStringBase::LoadStr 3197
System::AnsiStringBase::LoadString 3197
System::AnsiStringBase::LowerCase 3198
System::AnsiStringBase::Pos 3198
System::AnsiStringBase::RefCount 3198
System::AnsiStringBase::SetLength 3199
System::AnsiStringBase::StringOfChar 3199
System::AnsiStringBase::StrRec 3180
System::AnsiStringBase::StrRec structure 3180
System::AnsiStringBase::SubString 3199
System::AnsiStringBase::swap 3200
System::AnsiStringBase::ThrowIfOutOfRange 3200
System::AnsiStringBase::ToDouble 3200
System::AnsiStringBase::ToInt 3200
System::AnsiStringBase::ToIntDef 3201
System::AnsiStringBase::Trim 3201
System::AnsiStringBase::TrimLeft 3201
System::AnsiStringBase::TrimRight 3202
System::AnsiStringBase::TStringFloatFormat 3180
System::AnsiStringBase::TStringFloatFormat enumeration 3180
System::AnsiStringBase::TStringMbcsByteType 3181
System::AnsiStringBase::TStringMbcsByteType enumeration 3181
System::AnsiStringBase::Unique 3202
System::AnsiStringBase::UpperCase 3202
System::AnsiStringBase::vprintf 3202
System::AnsiStringBase::WideChar 3203
System::AnsiStringBase::WideCharBufSize 3203
System::AnsiStringT 3207
System::AnsiStringT::!= 3232
System::AnsiStringT::~AnsiStringT 3211
System::AnsiStringT::+ 3232
System::AnsiStringT::+= 3232
System::AnsiStringT::< 3233
System::AnsiStringT::<= 3233
System::AnsiStringT::= 3233
System::AnsiStringT::== 3234
System::AnsiStringT::> 3234
System::AnsiStringT::>= 3234
System::AnsiStringT::AnsiCompare 3214
System::AnsiStringT::AnsiCompareIC 3215
System::AnsiStringT::AnsiPos 3215
System::AnsiStringT::AnsiStringT 3212
System::AnsiStringT::c_str 3216
System::AnsiStringT::cat_printf 3216
System::AnsiStringT::cat_sprintf 3216
System::AnsiStringT::cat_vprintf 3217
System::AnsiStringT::CodePage 3217
System::AnsiStringT::CurrToStr 3218
System::AnsiStringT::CurrToStrF 3218
System::AnsiStringT::data 3218
System::AnsiStringT::Delete 3219
System::AnsiStringT::ElementSize 3219
System::AnsiStringT::EnsureAnsi 3220
System::AnsiStringT::FloatToStrF 3220
System::AnsiStringT::FmtLoadStr 3221
System::AnsiStringT::Format 3221
System::AnsiStringT::FormatFloat 3222
System::AnsiStringT::Insert 3223
System::AnsiStringT::IntToHex 3224
System::AnsiStringT::IsDelimiter 3224
System::AnsiStringT::LastDelimiter 3224
System::AnsiStringT::LoadStr 3225
System::AnsiStringT::LoadString 3225
System::AnsiStringT::LowerCase 3226
System::AnsiStringT::Pos 3226
System::AnsiStringT::printf 3226
System::AnsiStringT::RefCount 3227
System::AnsiStringT::SetLength 3227
System::AnsiStringT::sprintf 3227
System::AnsiStringT::StringOfChar 3228
System::AnsiStringT::SubString 3228
System::AnsiStringT::swap 3228
System::AnsiStringT::Trim 3229
System::AnsiStringT::TrimLeft 3229

System::AnsiStringT::TrimRight 3229
System::AnsiStringT::Unique 3229
System::AnsiStringT::UpperCase 3230
System::AnsiStringT::vprintf 3230
System::AnsiStringT::WideChar 3230
System::AnsiStringT::WideCharBufSize 3231
System::CheckSafecallResult 3536
System::CheckSafecallResult function 3536
System::Currency 3235
System::Currency::- 3239
System::Currency::-- 3240
System::Currency::! 3240
System::Currency::!= 3241
System::Currency::% 3241
System::Currency::%= 3242
System::Currency::* 3242
System::Currency::*= 3243
System::Currency::/ 3243
System::Currency::/= 3244
System::Currency::+ 3245
System::Currency::++ 3246
System::Currency::+= 3246
System::Currency::< 3247
System::Currency::<= 3248
System::Currency::= 3248
System::Currency::-= 3249
System::Currency::== 3250
System::Currency::> 3250
System::Currency::>= 3251
System::Currency::Currency 3237
System::Currency::double 3252
System::Currency::int 3252
System::Currency::System::String 3252
System::CurrencyBase 3255
System::CurrencyBase::Val 3256
System::DelphiInterface 3256
System::DelphiInterface::! 3259
System::DelphiInterface::& 3259
System::DelphiInterface::* 3259
System::DelphiInterface::~DelphiInterface 3257
System::DelphiInterface::= 3260
System::DelphiInterface::-> 3260
System::DelphiInterface::DelphiInterface 3258
System::DelphiInterface::Release 3258
System::DelphiInterface::T* 3261
System::DynamicArray 3261
System::DynamicArray::[] 3269
System::DynamicArray::~DynamicArray 3263
System::DynamicArray::= 3269
System::DynamicArray::== 3269
System::DynamicArray::AllocData 3265
System::DynamicArray::Copy 3265
System::DynamicArray::CopyRange 3266
System::DynamicArray::DecRefCount 3266
System::DynamicArray::DimCount 3267
System::DynamicArray::DynamicArray 3264
System::DynamicArray::FreeData 3267
System::DynamicArray::get_high 3267
System::DynamicArray::get_length 3267
System::DynamicArray::get_low 3267
System::DynamicArray::High 3270
System::DynamicArray::IncRefCount 3268
System::DynamicArray::Length 3270
System::DynamicArray::Low 3271
System::DynamicArray::set_length 3268
System::DynamicArray::SetData 3268
System::DynArrayException 3271
System::DynArrayNullData 3272
System::DynArrayOutOfRange 3272
System::DynArrayOutOfRange::DynArrayOutOfRange 3273
System::DynArrayOutOfRange::m_Count 3273
System::DynArrayOutOfRange::m_Index 3273
System::FindDynaInst 3573
System::FindDynaInst function 3573
System::LoadResourceString 3610
System::LoadResourceString function 3610
System::OleVariant 3274
System::OleVariant::!= 3302

System::OleVariant::~OleVariant 3279
System::OleVariant::< 3302, 3303
System::OleVariant::<= 3303
System::OleVariant::= 3304
System::OleVariant::== 3305, 3306
System::OleVariant::> 3306
System::OleVariant::>= 3307
System::OleVariant::OleVariant 3281
System::OleVariant::TVarData::Reserved1 3284
System::OleVariant::TVarData::Reserved2 3284
System::OleVariant::TVarData::Reserved3 3284
System::OleVariant::TVarData::VArray 3284
System::OleVariant::TVarData::VBoolean 3285
System::OleVariant::TVarData::VByte 3285
System::OleVariant::TVarData::VCurrency 3285
System::OleVariant::TVarData::VDate 3285
System::OleVariant::TVarData::VDispatch 3285
System::OleVariant::TVarData::VDouble 3286
System::OleVariant::TVarData::VError 3286
System::OleVariant::TVarData::VInteger 3286
System::OleVariant::TVarData::VOleStr 3286
System::OleVariant::TVarData::VPointer 3287
System::OleVariant::TVarData::VSingle 3287
System::OleVariant::TVarData::VSmallint 3287
System::OleVariant::TVarData::VType 3287
System::OleVariant::TVarData::VUnknown 3287
System::OleVariant::Variant::- 3288
System::OleVariant::Variant::! 3288
System::OleVariant::Variant::%= 3288
System::OleVariant::Variant::&= 3288
System::OleVariant::Variant::*= 3289
System::OleVariant::Variant::/= 3289
System::OleVariant::Variant::^= 3289
System::OleVariant::Variant::__int64 3289
System::OleVariant::Variant::|= 3289
System::OleVariant::Variant::+= 3290
System::OleVariant::Variant::<<= 3290
System::OleVariant::Variant::-= 3290
System::OleVariant::Variant::>>= 3290
System::OleVariant::Variant::AnsiString 3291
System::OleVariant::Variant::ArrayDimCount 3291
System::OleVariant::Variant::ArrayHighBound 3291
System::OleVariant::Variant::ArrayLock 3291
System::OleVariant::Variant::ArrayLowBound 3291
System::OleVariant::Variant::ArrayRedim 3292
System::OleVariant::Variant::ArrayUnlock 3292
System::OleVariant::Variant::AsType 3292
System::OleVariant::Variant::bool 3292
System::OleVariant::Variant::Byte* 3293
System::OleVariant::Variant::ChangeType 3293
System::OleVariant::Variant::char 3293
System::OleVariant::Variant::Clear 3293
System::OleVariant::Variant::Currency 3293
System::OleVariant::Variant::Currency* 3294
System::OleVariant::Variant::double 3294
System::OleVariant::Variant::double* 3294
System::OleVariant::Variant::float 3294
System::OleVariant::Variant::float* 3295
System::OleVariant::Variant::GetElement 3295
System::OleVariant::Variant::GetError 3295
System::OleVariant::Variant::IDispatch* 3295
System::OleVariant::Variant::IInterface* 3295
System::OleVariant::Variant::int 3296
System::OleVariant::Variant::int* 3296
System::OleVariant::Variant::IsArray 3296
System::OleVariant::Variant::IsEmpty 3296
System::OleVariant::Variant::IsNull 3297
System::OleVariant::Variant::IUnknown* 3297
System::OleVariant::Variant::long 3297
System::OleVariant::Variant::long* 3297
System::OleVariant::Variant::PutElement 3297
System::OleVariant::Variant::SetError 3298
System::OleVariant::Variant::short 3298
System::OleVariant::Variant::short* 3298
System::OleVariant::Variant::signed char 3298
System::OleVariant::Variant::signed char* 3299
System::OleVariant::Variant::TDateTime 3299
System::OleVariant::Variant::TDateTime* 3299

System::OleVariant::Variant::Type 3299
System::OleVariant::Variant::UnicodeString 3299
System::OleVariant::Variant::unsigned char 3300
System::OleVariant::Variant::unsigned int 3300
System::OleVariant::Variant::unsigned int* 3300
System::OleVariant::Variant::unsigned long 3300
System::OleVariant::Variant::unsigned short 3301
System::OleVariant::Variant::unsigned short* 3301
System::OleVariant::Variant::WideString 3301
System::PCallDesc 3758
System::PCallDesc type 3758
System::PDispDesc 3761
System::PDispDesc type 3761
System::PMemoryManager 3768
System::PMemoryManager type 3768
System::PMemoryManagerEx 3768
System::PMemoryManagerEx type 3768
System::PModuleUnloadRec 3769
System::PModuleUnloadRec type 3769
System::PResStringRec 3774
System::PResStringRec type 3774
System::PSmallBlockTypeState 3775
System::PSmallBlockTypeState type 3775
System::PVariantManager 3781
System::PVariantManager type 3781
System::PVarRec 3781
System::PVarRec type 3781
System::ResourceString 3783
System::ResourceString type 3783
System::SmallString 3307
System::SmallString::[] 3309
System::SmallString::= 3309
System::SmallString::AnsiString 3310
System::SmallString::SmallString 3308, 3309
System::SmallStringBase 3310
System::SmallStringBase::Data 3311
System::SmallStringBase::Len 3311
System::sPointer 3311
System::sPointer::& 3313
System::sPointer::~sPointer 3312
System::sPointer::-> 3313
System::sPointer::sPointer 3312
System::String
  Currency class 3252
  TDateTime class 3341
System::TAggregatedObject 3313
System::TAggregatedObject::_AddRef 3316
System::TAggregatedObject::_Release 3316
System::TAggregatedObject::Controller 3317
System::TAggregatedObject::Create 3315
System::TAggregatedObject::QueryInterface 3317
System::TBooleanDynArray 3784
System::TBooleanDynArray type 3784
System::TByteDynArray 3784
System::TByteDynArray type 3784
System::TCallDesc 3749
System::TCallDesc structure 3749
System::TCardinalDynArray 3785
System::TCardinalDynArray type 3785
System::TContainedObject 3320
System::TContainedObject::QueryInterface 3322
System::TDateTime 3323
System::TDateTime::- 3334
System::TDateTime::-- 3334
System::TDateTime::!= 3335
System::TDateTime::+ 3335
System::TDateTime::++ 3336
System::TDateTime::+= 3337
System::TDateTime::< 3337
System::TDateTime::<= 3337
System::TDateTime::= 3338
System::TDateTime::-= 3339
System::TDateTime::== 3339
System::TDateTime::> 3339
System::TDateTime::>= 3340
System::TDateTime::CurrentDate 3328
System::TDateTime::CurrentDateTime 3328
System::TDateTime::CurrentTime 3328

System::TDateTime::DateString 3329
System::TDateTime::DateTimeString 3329
System::TDateTime::DayOfWeek 3329
System::TDateTime::DecodeDate 3330
System::TDateTime::DecodeTime 3330
System::TDateTime::double 3340
System::TDateTime::FileDate 3330
System::TDateTime::FileDateToDateTime 3331
System::TDateTime::FormatString 3331
System::TDateTime::int 3340
System::TDateTime::System::String 3341
System::TDateTime::TDateTime 3325
System::TDateTime::TDateTimeFlag 3327
System::TDateTime::TDateTimeFlag enumeration 3327
System::TDateTime::TimeString 3333
System::TDateTimeBase 3341
System::TDateTimeBase::Val 3342
System::TDispDesc 3749
System::TDispDesc structure 3749
System::TDoubleDynArray 3788
System::TDoubleDynArray type 3788
System::THeapBlock 3720
System::THeapBlock structure 3720
System::THeapStatus 3720
System::THeapStatus record 3720
System::TInt64DynArray 3790
System::TInt64DynArray type 3790
System::TIntegerDynArray 3790
System::TIntegerDynArray type 3790
System::TInterfacedObject 3342
System::TInterfacedObject::_AddRef 3345
System::TInterfacedObject::_Release 3345
System::TInterfacedObject::AfterConstruction 3345
System::TInterfacedObject::BeforeDestruction 3346
System::TInterfacedObject::FRefCount 3344
System::TInterfacedObject::NewInstance 3346
System::TInterfacedObject::QueryInterface 3347
System::TInterfacedObject::RefCount 3347
System::TLibModule 3723
System::TLibModule record 3723
System::TLongWordDynArray 3791
System::TLongWordDynArray type 3791
System::TMemoryManager 3724
System::TMemoryManager record 3724
System::TMemoryManagerEx 3725
System::TMemoryManagerEx record 3725
System::TMemoryManagerState 3730
System::TMemoryManagerState record 3730
System::TMetaClass 3348
System::TModuleUnloadRec 3732
System::TModuleUnloadRec record 3732
System::TObject 3354
System::TObject::AfterConstruction 3357
System::TObject::BeforeDestruction 3358
System::TObject::ClassInfo 3358
System::TObject::ClassName 3359
System::TObject::ClassNameIs 3361
System::TObject::ClassParent 3361
System::TObject::ClassType 3364
System::TObject::CleanupInstance 3367
System::TObject::Create 3355
System::TObject::DefaultHandler 3367
System::TObject::Destroy 3356
System::TObject::Dispatch 3369
System::TObject::Equals 3371
System::TObject::FieldAddress 3372
System::TObject::Free 3372
System::TObject::FreeInstance 3374
System::TObject::GetHashCode 3375
System::TObject::GetInterface 3375
System::TObject::GetInterfaceEntry 3376
System::TObject::GetInterfaceTable 3376
System::TObject::InheritsFrom 3376
System::TObject::InitInstance 3377
System::TObject::InstanceSize 3377
System::TObject::MethodAddress 3378
System::TObject::MethodName 3379
System::TObject::NewInstance 3379

System::TObject::SafeCallException 3380
System::TObject::ToString 3380
System::TObject::UnitName 3380
System::TResStringRec 3733
System::TResStringRec record 3733
System::TShortIntDynArray 3794
System::TShortIntDynArray type 3794
System::TSingleDynArray 3794
System::TSingleDynArray type 3794
System::TSmallBlockTypeState 3736
System::TSmallBlockTypeState record 3736
System::TSmallBlockTypeStates 3795
System::TSmallBlockTypeStates type 3795
System::TSmallIntDynArray 3795
System::TSmallIntDynArray type 3795
System::TStringDynArray 3796
System::TStringDynArray type 3796
System::TThreadFunc 3797
System::TThreadFunc type 3797
System::TVarArray 3739
System::TVarArray record 3739
System::TVarArrayBound 3740
System::TVarArrayBound record 3740
System::TVariantManager 3745
System::TVariantManager record 3745
System::TVarRec 3747
System::TVarRec record 3747
System::TWideStringDynArray 3801
System::TWideStringDynArray type 3801
System::TWordDynArray 3801
System::TWordDynArray type 3801
System::UnicodeString 3381
System::UnicodeString::!= 3414
System::UnicodeString::[] 3414
System::UnicodeString::~UnicodeString 3384
System::UnicodeString::+ 3415
System::UnicodeString::+= 3416
System::UnicodeString::< 3416
System::UnicodeString::<= 3417
System::UnicodeString::= 3418
System::UnicodeString::== 3418
System::UnicodeString::> 3419
System::UnicodeString::>= 3419
System::UnicodeString::BytesOf 3391
System::UnicodeString::ByteType 3391
System::UnicodeString::c_str 3392
System::UnicodeString::cat_printf 3392
System::UnicodeString::cat_sprintf 3393
System::UnicodeString::cat_vprintf 3394
System::UnicodeString::CodePage 3394
System::UnicodeString::Compare 3394
System::UnicodeString::CompareIC 3395
System::UnicodeString::CurrToStr 3395
System::UnicodeString::CurrToStrF 3396
System::UnicodeString::data 3396
System::UnicodeString::Delete 3396
System::UnicodeString::ElementSize 3397
System::UnicodeString::EnsureUnicode 3397
System::UnicodeString::FloatToStrF 3397
System::UnicodeString::FmtLoadStr 3398
System::UnicodeString::Format 3399
System::UnicodeString::FormatFloat 3399
System::UnicodeString::GetRec 3401
System::UnicodeString::Insert 3401
System::UnicodeString::IntToHex 3402
System::UnicodeString::IsDelimiter 3402
System::UnicodeString::IsEmpty 3402
System::UnicodeString::IsLeadSurrogate 3403
System::UnicodeString::IsPathDelimiter 3403
System::UnicodeString::IsTrailSurrogate 3403
System::UnicodeString::LastChar 3404
System::UnicodeString::LastDelimiter 3404
System::UnicodeString::Length 3404
System::UnicodeString::LoadStr 3405
System::UnicodeString::LoadString 3405
System::UnicodeString::LowerCase 3406
System::UnicodeString::Pos 3406
System::UnicodeString::printf 3406

System::UnicodeString::RefCount 3407
System::UnicodeString::SetLength 3407
System::UnicodeString::sprintf 3408
System::UnicodeString::StringOfChar 3408
System::UnicodeString::StrRec 3387
System::UnicodeString::StrRec structure 3387
System::UnicodeString::SubString 3408
System::UnicodeString::swap 3409
System::UnicodeString::t_str 3409
System::UnicodeString::ThrowIfOutOfRange 3410
System::UnicodeString::ToDouble 3410
System::UnicodeString::ToInt 3410
System::UnicodeString::ToIntDef 3411
System::UnicodeString::Trim 3411
System::UnicodeString::TrimLeft 3411
System::UnicodeString::TrimRight 3412
System::UnicodeString::TStringFloatFormat 3388
System::UnicodeString::TStringFloatFormat enumeration 3388
System::UnicodeString::TStringLeadCharType 3389
System::UnicodeString::TStringLeadCharType enumeration 3389
System::UnicodeString::UnicodeString 3386
System::UnicodeString::Unique 3412
System::UnicodeString::UpperCase 3412
System::UnicodeString::vprintf 3413
System::UnicodeString::w_str 3413
System::USE 3892
System::USE macro 3892
System::USEADDITIONALFILES 3892
System::USEADDITIONALFILES macro 3892
System::USEASM 3893
System::USEASM macro 3893
System::USEDATAMODULE 3893
System::USEDATAMODULE macro 3893
System::USEDATAMODULENS 3893
System::USEDATAMODULENS macro 3893
System::USEDEF 3894
System::USEDEF macro 3894
System::USEFILE 3894
System::USEFILE macro 3894
System::USEFORM 3895
System::USEFORM macro 3895
System::USEFORMNS 3895
System::USEFORMNS macro 3895
System::USEFORMRES 3895
System::USEFORMRES macro 3895
System::USEIDL 3896
System::USEIDL macro 3896
System::USELIB 3896
System::USELIB macro 3896
System::USEOBJ 3896
System::USEOBJ macro 3896
System::USEPACKAGE 3897
System::USEPACKAGE macro 3897
System::USERC 3897
System::USERC macro 3897
System::USERES 3897
System::USERES macro 3897
System::USETLB 3898
System::USETLB macro 3898
System::USEUNIT 3898
System::USEUNIT macro 3898
System::Variant 3421
System::Variant::- 3439
System::Variant::! 3439
System::Variant::!= 3440
System::Variant::% 3441
System::Variant::%= 3441
System::Variant::& 3442
System::Variant::&= 3442
System::Variant::* 3443
System::Variant::*= 3443
System::Variant::/ 3444
System::Variant::/= 3444
System::Variant::^ 3445
System::Variant::^= 3445
System::Variant::__int64 3446
System::Variant::| 3446
System::Variant::|= 3446

System::Variant::~Variant 3424
System::Variant::+ 3447
System::Variant::+= 3447
System::Variant::< 3448
System::Variant::<< 3449
System::Variant::<<= 3449
System::Variant::<= 3450
System::Variant::= 3453
System::Variant::-= 3454
System::Variant::== 3455
System::Variant::> 3456
System::Variant::>= 3457
System::Variant::>> 3457
System::Variant::>>= 3458
System::Variant::AnsiString 3458
System::Variant::ArrayDimCount 3429
System::Variant::ArrayHighBound 3429
System::Variant::ArrayLock 3429
System::Variant::ArrayLowBound 3430
System::Variant::ArrayRedim 3430
System::Variant::ArrayUnlock 3430
System::Variant::AsType 3431
System::Variant::bool 3458
System::Variant::ChangeType 3431
System::Variant::char 3459
System::Variant::Clear 3431
System::Variant::Currency 3459
System::Variant::Currency* 3459
System::Variant::double 3460
System::Variant::double* 3460
System::Variant::Empty 3431
System::Variant::float 3460
System::Variant::float* 3460
System::Variant::GetBaseVariant 3432
System::Variant::GetElement 3432
System::Variant::GetError 3433
System::Variant::IDispatch* 3461
System::Variant::IInterface* 3461
System::Variant::int 3461
System::Variant::int* 3461
System::Variant::IsArray 3433
System::Variant::IsEmpty 3433
System::Variant::IsNull 3434
System::Variant::IUnknown* 3462
System::Variant::long 3462
System::Variant::long* 3462
System::Variant::NoParam 3434
System::Variant::PutElement 3435
System::Variant::Sanitized 3435
System::Variant::SetError 3435
System::Variant::short 3462
System::Variant::short* 3463
System::Variant::signed char 3463
System::Variant::signed char* 3463
System::Variant::TDateTime 3463
System::Variant::TDateTime* 3464
System::Variant::Type 3436
System::Variant::UnicodeString 3464
System::Variant::unsigned char 3464
System::Variant::unsigned char* 3465
System::Variant::unsigned int 3465
System::Variant::unsigned int* 3465
System::Variant::unsigned long 3465
System::Variant::unsigned long* 3466
System::Variant::unsigned short 3466
System::Variant::unsigned short* 3466
System::Variant::Variant 3426
System::Variant::WideString 3466
System::WideString 3467
System::WideString::!= 3488
System::WideString::[] 3488
System::WideString::~WideString 3469
System::WideString::+ 3489
System::WideString::+= 3489
System::WideString::< 3489
System::WideString::<= 3490
System::WideString::= 3490
System::WideString::== 3491

System::WideString::> 3491
System::WideString::>= 3492
System::WideString::c_bstr 3473
System::WideString::cat_printf 3473
System::WideString::cat_sprintf 3474
System::WideString::cat_vprintf 3474
System::WideString::Copy 3475
System::WideString::CurrToStr 3475
System::WideString::CurrToStrF 3475
System::WideString::data 3476
System::WideString::Delete 3476
System::WideString::FloatToStrF 3477
System::WideString::Format 3477
System::WideString::FormatFloat 3478
System::WideString::Insert 3479
System::WideString::IntToHex 3480
System::WideString::IsDelimiter 3480
System::WideString::IsEmpty 3480
System::WideString::IsPathDelimiter 3481
System::WideString::LastChar 3481
System::WideString::LastDelimiter 3481
System::WideString::Length 3481
System::WideString::LowerCase 3482
System::WideString::Pos 3482
System::WideString::printf 3482
System::WideString::SetLength 3483
System::WideString::sprintf 3483
System::WideString::StringOfChar 3484
System::WideString::SubString 3484
System::WideString::swap 3484
System::WideString::ToDouble 3485
System::WideString::ToInt 3485
System::WideString::ToIntDef 3485
System::WideString::Trim 3486
System::WideString::TrimLeft 3486
System::WideString::TrimRight 3486
System::WideString::TStringFloatFormat 3471
System::WideString::TStringFloatFormat enumeration 3471
System::WideString::UpperCase 3486
System::WideString::vprintf 3487
System::WideString::WideString 3470
systemId
IDOMDocumentType interface 7542
IDOMEntity interface 7551
IDOMNotation interface 7586
Sysutils 3899
SysUtils 3901
Sysutils namespace 3899
Functions 3900
SysUtils namespace 3901
Classes 3915
Constants 4368
Functions 4039
Interfaces 4037
Structs, Records, Enums 4320
Types 4336
Variables 4347
SysUtils.Abort 4047
SysUtils.Abort function 4047
SysUtils.AddExitProc 4048
SysUtils.AddExitProc function 4048
SysUtils.AddTerminateProc 4048
SysUtils.AddTerminateProc function 4048
SysUtils.AdjustLineBreaks 4049
SysUtils.AdjustLineBreaks function 4049
SysUtils.AnsiCompareFileName 4049
SysUtils.AnsiCompareFileName function 4049
SysUtils.AnsiCompareStr 4050
SysUtils.AnsiCompareStr function 4050
SysUtils.AnsiCompareText 4051
SysUtils.AnsiCompareText function 4051
SysUtils.AnsiDequotedStr 4051
SysUtils.AnsiDequotedStr function 4051
SysUtils.AnsiExtractQuotedStr 4052
SysUtils.AnsiExtractQuotedStr function 4052
SysUtils.AnsiLastChar 4052
SysUtils.AnsiLastChar function 4052
SysUtils.AnsiLowerCase 4053

SysUtils.AnsiLowerCase function 4053
SysUtils.AnsiLowerCaseFileName 4053
SysUtils.AnsiLowerCaseFileName function 4053
SysUtils.AnsiPos 4054
SysUtils.AnsiPos function 4054
SysUtils.AnsiQuotedStr 4054
SysUtils.AnsiQuotedStr function 4054
SysUtils.AnsiSameStr 4055
SysUtils.AnsiSameStr function 4055
SysUtils.AnsiSameText 4055
SysUtils.AnsiSameText function 4055
SysUtils.AnsiStrAlloc 4056
SysUtils.AnsiStrAlloc function 4056
SysUtils.AnsiStrComp 4056
SysUtils.AnsiStrComp function 4056
SysUtils.AnsiStrIComp 4057
SysUtils.AnsiStrIComp function 4057
SysUtils.AnsiStrLastChar 4057
SysUtils.AnsiStrLastChar function 4057
SysUtils.AnsiStrLComp 4058
SysUtils.AnsiStrLComp function 4058
SysUtils.AnsiStrLIComp 4058
SysUtils.AnsiStrLIComp function 4058
SysUtils.AnsiStrLower 4059
SysUtils.AnsiStrLower function 4059
SysUtils.AnsiStrPos 4060
SysUtils.AnsiStrPos function 4060
SysUtils.AnsiStrRScan 4060
SysUtils.AnsiStrRScan function 4060
SysUtils.AnsiStrScan 4061
SysUtils.AnsiStrScan function 4061
SysUtils.AnsiStrUpper 4061
SysUtils.AnsiStrUpper function 4061
SysUtils.AnsiUpperCase 4062
SysUtils.AnsiUpperCase function 4062
SysUtils.AnsiUpperCaseFileName 4062
SysUtils.AnsiUpperCaseFileName function 4062
SysUtils.AppendStr 4063
SysUtils.AppendStr function 4063
SysUtils.AssignStr 4063
SysUtils.AssignStr function 4063
SysUtils.Beep 4063
SysUtils.Beep function 4063
SysUtils.BoolToStr 4066
SysUtils.BoolToStr function 4066
SysUtils.ByteLength 4068
SysUtils.ByteLength function 4068
SysUtils.BytesOf 4068
SysUtils.BytesOf function 4068
SysUtils.ByteToCharIndex 4069
SysUtils.ByteToCharIndex function 4069
SysUtils.ByteToCharLen 4070
SysUtils.ByteToCharLen function 4070
SysUtils.ByteType 4070
SysUtils.ByteType function 4070
SysUtils.CallTerminateProcs 4071
SysUtils.CallTerminateProcs function 4071
SysUtils.ChangeFileExt 4071
SysUtils.ChangeFileExt function 4071
SysUtils.ChangeFilePath 4073
SysUtils.ChangeFilePath function 4073
SysUtils.CharInSet 4073
SysUtils.CharInSet function 4073
SysUtils.CharLength 4074
SysUtils.CharLength function 4074
SysUtils.CharToByteIndex 4075
SysUtils.CharToByteIndex function 4075
SysUtils.CharToByteLen 4076
SysUtils.CharToByteLen function 4076
SysUtils.CharToElementIndex 4076
SysUtils.CharToElementIndex function 4076
SysUtils.CharToElementLen 4077
SysUtils.CharToElementLen function 4077
SysUtils.CheckWin32Version 4077
SysUtils.CheckWin32Version function 4077
SysUtils.CompareMem 4078
SysUtils.CompareMem function 4078
SysUtils.CompareStr 4078

SysUtils.CompareStr function 4078
SysUtils.CompareText 4079
SysUtils.CompareText function 4079
SysUtils.CreateDir 4083
SysUtils.CreateDir function 4083
SysUtils.CreateGUID 4084
SysUtils.CreateGUID function 4084
SysUtils.CurrencyDecimals 4348
SysUtils.CurrencyDecimals variable 4348
SysUtils.CurrencyFormat 4348
SysUtils.CurrencyFormat variable 4348
SysUtils.CurrencyString 4349
SysUtils.CurrencyString variable 4349
SysUtils.CurrentYear 4086
SysUtils.CurrentYear function 4086
SysUtils.CurrToStr 4086
SysUtils.CurrToStr function 4086
SysUtils.CurrToStrF 4086
SysUtils.CurrToStrF function 4086
SysUtils.Date 4087
SysUtils.Date function 4087
SysUtils.DateDelta 4369
SysUtils.DateDelta constant 4369
SysUtils.DateSeparator 4349
SysUtils.DateSeparator variable 4349
SysUtils.DateTimeToFileDate 4088
SysUtils.DateTimeToFileDate function 4088
SysUtils.DateTimeToStr 4089
SysUtils.DateTimeToStr function 4089
SysUtils.DateTimeToString 4089
SysUtils.DateTimeToString function 4089
SysUtils.DateTimeToSystemTime 4091
SysUtils.DateTimeToSystemTime function 4091
SysUtils.DateTimeToTimeStamp 4092
SysUtils.DateTimeToTimeStamp function 4092
SysUtils.DateToStr 4092
SysUtils.DateToStr function 4092
SysUtils.DayOfWeek 4094
SysUtils.DayOfWeek function 4094
SysUtils.DecimalSeparator 4350
SysUtils.DecimalSeparator variable 4350
SysUtils.DecodeDate 4095
SysUtils.DecodeDate function 4095
SysUtils.DecodeDateFully 4096
SysUtils.DecodeDateFully function 4096
SysUtils.DecodeTime 4097
SysUtils.DecodeTime function 4097
SysUtils.DefaultFalseBoolStr 4370
SysUtils.DefaultFalseBoolStr constant 4370
SysUtils.DefaultTrueBoolStr 4370
SysUtils.DefaultTrueBoolStr constant 4370
SysUtils.DelegatesEqual 4098
SysUtils.DelegatesEqual function 4098
SysUtils.DeleteFile 4098
SysUtils.DeleteFile function 4098
SysUtils.DirectoryExists 4100
SysUtils.DirectoryExists function 4100
SysUtils.DiskFree 4101
SysUtils.DiskFree function 4101
SysUtils.DiskSize 4102
SysUtils.DiskSize function 4102
SysUtils.DisposeStr 4103
SysUtils.DisposeStr function 4103
SysUtils.DriveDelim 4371
SysUtils.DriveDelim constant 4371
SysUtils.EAbort 3916
SysUtils.EAbstractError 3916
SysUtils.EAccessViolation 3917
SysUtils.EArgumentException 3918
SysUtils.EArgumentOutOfRangeException 3918
SysUtils.EAssertionFailed 3918
SysUtils.EControlC 3919
SysUtils.EConvertError 3919
SysUtils.EDivByZero 3921
SysUtils.EEncodingError 3922
SysUtils.EExternal 3922
SysUtils.EExternal.ExceptionRecord 3925
SysUtils.EExternalException 3925

SysUtils.EHeapException 3926
SysUtils.EHeapException.FreeInstance 3929
SysUtils.EHeapException.RaisingException 3929
SysUtils.EInOutError 3929
SysUtils.EInOutError.ErrorCode 3933
SysUtils.EIntError 3933
SysUtils.EIntfCastError 3934
SysUtils.EIntOverflow 3934
SysUtils.EInvalidCast 3935
SysUtils.EInvalidContainer 3935
SysUtils.EInvalidInsert 3936
SysUtils.EInvalidOp 3936
SysUtils.EInvalidPointer 3937
SysUtils.ElementToCharIndex 4104
SysUtils.ElementToCharIndex function 4104
SysUtils.ElementToCharLen 4104, 4105
SysUtils.ElementToCharLen function 4104, 4105
SysUtils.EMathError 3937
SysUtils.EMonitor 3938
SysUtils.EMonitorLockException 3938
SysUtils.EmptyAnsiStr 4351
SysUtils.EmptyAnsiStr variable 4351
SysUtils.EmptyStr 4351
SysUtils.EmptyStr variable 4351
SysUtils.EmptyWideStr 4351
SysUtils.EmptyWideStr variable 4351
SysUtils.EncodeDate 4105
SysUtils.EncodeDate function 4105
SysUtils.EncodeTime 4106
SysUtils.EncodeTime function 4106
SysUtils.ENoConstructException 3939
SysUtils.ENoMonitorSupportException 3939
SysUtils.EOSError 3939
SysUtils.EOSError.ErrorCode 3942
SysUtils.EOutOfMemory 3942
SysUtils.EOverflow 3942
SysUtils.EPackageError 3943
SysUtils.EPrivilege 3943
SysUtils.EProgrammerNotFound 3944
SysUtils.EPropReadOnly 3944
SysUtils.EPropWriteOnly 3945
SysUtils.EraNames 4352
SysUtils.EraNames variable 4352
SysUtils.ERangeError 3945
SysUtils.EraYearOffsets 4352
SysUtils.EraYearOffsets variable 4352
SysUtils.ESafecallException 3946
SysUtils.EStackOverflow 3947
SysUtils.EUnderflow 3947
SysUtils.EVariantError 3948
SysUtils.EWin32Error 3949
SysUtils.ExceptClass 4336
SysUtils.ExceptClass type 4336
SysUtils.Exception 3949
SysUtils.Exception.BaseException 3964
SysUtils.Exception.CleanUpStackInfoProc 3961
SysUtils.Exception.Create 3952
SysUtils.Exception.CreateFmt 3954
SysUtils.Exception.CreateFmtHelp 3955
SysUtils.Exception.CreateHelp 3955
SysUtils.Exception.CreateRes 3956
SysUtils.Exception.CreateResFmt 3957
SysUtils.Exception.CreateResFmtHelp 3958
SysUtils.Exception.CreateResHelp 3959
SysUtils.Exception.Destroy 3960
SysUtils.Exception.GetBaseException 3962
SysUtils.Exception.GetExceptionStackInfoProc 3961
SysUtils.Exception.GetStackInfoStringProc 3961
SysUtils.Exception.GetStackTrace 3962
SysUtils.Exception.HelpContext 3964
SysUtils.Exception.InnerException 3965
SysUtils.Exception.Message 3965
SysUtils.Exception.RaiseOuterException 3962
SysUtils.Exception.RaisingException 3962
SysUtils.Exception.SetInnerException 3962
SysUtils.Exception.SetStackInfo 3963
SysUtils.Exception.StackInfo 3966
SysUtils.Exception.StackTrace 3966

SysUtils.Exception.ThrowOuterException 3963
SysUtils.Exception.ToString 3963
SysUtils.ExceptionErrorMessage 4107
SysUtils.ExceptionErrorMessage function 4107
SysUtils.ExcludeTrailingBackslash 4108
SysUtils.ExcludeTrailingBackslash function 4108
SysUtils.ExcludeTrailingPathDelimiter 4109
SysUtils.ExcludeTrailingPathDelimiter function 4109
SysUtils.ExpandFileName 4109
SysUtils.ExpandFileName function 4109
SysUtils.ExpandFileNameCase 4110
SysUtils.ExpandFileNameCase function 4110
SysUtils.ExpandUNCFileName 4111
SysUtils.ExpandUNCFileName function 4111
SysUtils.ExtractFileDir 4112
SysUtils.ExtractFileDir function 4112
SysUtils.ExtractFileDrive 4112
SysUtils.ExtractFileDrive function 4112
SysUtils.ExtractFileExt 4113
SysUtils.ExtractFileExt function 4113
SysUtils.ExtractFileName 4113
SysUtils.ExtractFileName function 4113
SysUtils.ExtractFilePath 4115
SysUtils.ExtractFilePath function 4115
SysUtils.ExtractRelativePath 4117
SysUtils.ExtractRelativePath function 4117
SysUtils.ExtractShortPathName 4118
SysUtils.ExtractShortPathName function 4118
SysUtils.EZeroDivide 3966
SysUtils.faAnyFile 4371
SysUtils.faAnyFile constant 4371
SysUtils.faArchive 4372
SysUtils.faArchive constant 4372
SysUtils.faDirectory 4372
SysUtils.faDirectory constant 4372
SysUtils.faHidden 4373
SysUtils.faHidden constant 4373
SysUtils.FalseBoolStrs 4352
SysUtils.FalseBoolStrs variable 4352
SysUtils.faNormal 4374
SysUtils.faNormal constant 4374
SysUtils.faReadOnly 4374
SysUtils.faReadOnly constant 4374
SysUtils.faSymLink 4375
SysUtils.faSymLink constant 4375
SysUtils.faSysFile 4375
SysUtils.faSysFile constant 4375
SysUtils.faTemporary 4376
SysUtils.faTemporary constant 4376
SysUtils.faVolumeID 4376
SysUtils.faVolumeID constant 4376
SysUtils.FileAge 4119
SysUtils.FileAge function 4119
SysUtils.FileClose 4121
SysUtils.FileClose function 4121
SysUtils.FileCreate 4128
SysUtils.FileCreate function 4128
SysUtils.FileDateToDateTime 4130
SysUtils.FileDateToDateTime function 4130
SysUtils.FileExists 4133
SysUtils.FileExists function 4133
SysUtils.FileGetAttr 4139
SysUtils.FileGetAttr function 4139
SysUtils.FileGetDate 4142
SysUtils.FileGetDate function 4142
SysUtils.FileIsReadOnly 4142
SysUtils.FileIsReadOnly function 4142
SysUtils.FileOpen 4143
SysUtils.FileOpen function 4143
SysUtils.FileRead 4150
SysUtils.FileRead function 4150
SysUtils.FileSearch 4156
SysUtils.FileSearch function 4156
SysUtils.FileSeek 4157
SysUtils.FileSeek function 4157
SysUtils.FileSetAttr 4162
SysUtils.FileSetAttr function 4162
SysUtils.FileSetDate 4164

SysUtils.FileSetDate function 4164
SysUtils.FileSetReadOnly 4165
SysUtils.FileSetReadOnly function 4165
SysUtils.FileWrite 4166
SysUtils.FileWrite function 4166
SysUtils.FinalizePackage 4172
SysUtils.FinalizePackage function 4172
SysUtils.FindClose 4172
SysUtils.FindClose function 4172
SysUtils.FindCmdLineSwitch 4174
SysUtils.FindCmdLineSwitch function 4174
SysUtils.FindFirst 4175
SysUtils.FindFirst function 4175
SysUtils.FindNext 4177
SysUtils.FindNext function 4177
SysUtils.FloatToCurr 4179
SysUtils.FloatToCurr function 4179
SysUtils.FloatToDateTime 4179
SysUtils.FloatToDateTime function 4179
SysUtils.FloatToDecimal 4180
SysUtils.FloatToDecimal function 4180
SysUtils.FloatToStr 4180
SysUtils.FloatToStr function 4180
SysUtils.FloatToStrF 4182
SysUtils.FloatToStrF function 4182
SysUtils.FloatToText 4183
SysUtils.FloatToText function 4183
SysUtils.FloatToTextFmt 4184
SysUtils.FloatToTextFmt function 4184
SysUtils.fmOpenRead 4377
SysUtils.fmOpenRead constant 4377
SysUtils.fmOpenReadWrite 4378
SysUtils.fmOpenReadWrite constant 4378
SysUtils.fmOpenWrite 4378
SysUtils.fmOpenWrite constant 4378
SysUtils.fmShareCompat 4379
SysUtils.fmShareCompat constant 4379
SysUtils.fmShareDenyNone 4380
SysUtils.fmShareDenyNone constant 4380
SysUtils.fmShareDenyRead 4380
SysUtils.fmShareDenyRead constant 4380
SysUtils.fmShareDenyWrite 4381
SysUtils.fmShareDenyWrite constant 4381
SysUtils.fmShareExclusive 4382
SysUtils.fmShareExclusive constant 4382
SysUtils.FmtLoadStr 4185
SysUtils.FmtLoadStr function 4185
SysUtils.FmtStr 4185
SysUtils.FmtStr function 4185
SysUtils.ForceDirectories 4186
SysUtils.ForceDirectories function 4186
SysUtils.Format 4188
SysUtils.Format function 4188
SysUtils.FormatBuf 4193
SysUtils.FormatBuf function 4193
SysUtils.FormatCurr 4194
SysUtils.FormatCurr function 4194
SysUtils.FormatDateTime 4195
SysUtils.FormatDateTime function 4195
SysUtils.FormatFloat 4197
SysUtils.FormatFloat function 4197
SysUtils.FreeAndNil 4201
SysUtils.FreeAndNil function 4201
SysUtils.GDAL 4202
SysUtils.GDAL function 4202
SysUtils.GetCurrentDir 4202
SysUtils.GetCurrentDir function 4202
SysUtils.GetDiskFreeSpaceEx 4353
SysUtils.GetDiskFreeSpaceEx variable 4353
SysUtils.GetEnvironmentVariable 4204
SysUtils.GetEnvironmentVariable function 4204
SysUtils.GetFileVersion 4205
SysUtils.GetFileVersion function 4205
SysUtils.GetFormatSettings 4206
SysUtils.GetFormatSettings function 4206
SysUtils.GetLocaleChar 4206
SysUtils.GetLocaleChar function 4206
SysUtils.GetLocaleFormatSettings 4206

SysUtils.GetLocaleFormatSettings function 4206
SysUtils.GetLocaleStr 4207
SysUtils.GetLocaleStr function 4207
SysUtils.GetModuleName 4207
SysUtils.GetModuleName function 4207
SysUtils.GetPackageDescription 4208
SysUtils.GetPackageDescription function 4208
SysUtils.GetPackageInfo 4209
SysUtils.GetPackageInfo function 4209
SysUtils.GetTime 4209
SysUtils.GetTime function 4209
SysUtils.GUIDToString 4210
SysUtils.GUIDToString function 4210
SysUtils.HashName 4212
SysUtils.HashName function 4212
SysUtils.HexDisplayPrefix 4353
SysUtils.HexDisplayPrefix variable 4353
SysUtils.HoursPerDay 4382
SysUtils.HoursPerDay constant 4382
SysUtils.IncAMonth 4212
SysUtils.IncAMonth function 4212
SysUtils.IncludeTrailingBackslash 4213
SysUtils.IncludeTrailingBackslash function 4213
SysUtils.IncludeTrailingPathDelimiter 4213
SysUtils.IncludeTrailingPathDelimiter function 4213
SysUtils.IncMonth 4214
SysUtils.IncMonth function 4214
SysUtils.InitializePackage 4215
SysUtils.InitializePackage function 4215
SysUtils.Int64Rec 4320
SysUtils.Int64Rec record 4320
SysUtils.IntToHex 4215
SysUtils.IntToHex function 4215
SysUtils.IntToStr 4216
SysUtils.IntToStr function 4216
SysUtils.IReadWriteSync 4037
SysUtils.IReadWriteSync.BeginRead 4038
SysUtils.IReadWriteSync.BeginWrite 4038
SysUtils.IReadWriteSync.EndRead 4038
SysUtils.IReadWriteSync.EndWrite 4038
SysUtils.IsAssembly 4217
SysUtils.IsAssembly function 4217
SysUtils.IsDelimiter 4218
SysUtils.IsDelimiter function 4218
SysUtils.IsEqualGUID 4218
SysUtils.IsEqualGUID function 4218
SysUtils.IsLeadChar 4220
SysUtils.IsLeadChar function 4220
SysUtils.IsLeapYear 4220
SysUtils.IsLeapYear function 4220
SysUtils.IsPathDelimiter 4222
SysUtils.IsPathDelimiter function 4222
SysUtils.IsValidIdent 4222
SysUtils.IsValidIdent function 4222
SysUtils.Languages 4223
SysUtils.Languages function 4223
SysUtils.LastDelimiter 4223
SysUtils.LastDelimiter function 4223
SysUtils.LeadBytes 4354
SysUtils.LeadBytes variable 4354
SysUtils.ListSeparator 4354
SysUtils.ListSeparator variable 4354
SysUtils.LoadPackage 4224
SysUtils.LoadPackage function 4224
SysUtils.LoadStr 4224
SysUtils.LoadStr function 4224
SysUtils.LongDateFormat 4354
SysUtils.LongDateFormat variable 4354
SysUtils.LongDayNames 4355
SysUtils.LongDayNames variable 4355
SysUtils.LongMonthNames 4355
SysUtils.LongMonthNames variable 4355
SysUtils.LongRec 4321
SysUtils.LongRec record 4321
SysUtils.LongTimeFormat 4356
SysUtils.LongTimeFormat variable 4356
SysUtils.LowerCase 4225
SysUtils.LowerCase function 4225

SysUtils.MaxCurrency 4383
SysUtils.MaxCurrency constant 4383
SysUtils.MaxDateTime 4383
SysUtils.MaxDateTime constant 4383
SysUtils.MaxEraCount 4384
SysUtils.MaxEraCount constant 4384
SysUtils.MinCurrency 4384
SysUtils.MinCurrency constant 4384
SysUtils.MinDateTime 4384
SysUtils.MinDateTime constant 4384
SysUtils.MinsPerDay 4385
SysUtils.MinsPerDay constant 4385
SysUtils.MinsPerHour 4386
SysUtils.MinsPerHour constant 4386
SysUtils.MonthDays 4386
SysUtils.MonthDays constant 4386
SysUtils.MSecsPerDay 4387
SysUtils.MSecsPerDay constant 4387
SysUtils.MSecsPerSec 4387
SysUtils.MSecsPerSec constant 4387
SysUtils.MSecsToTimeStamp 4226
SysUtils.MSecsToTimeStamp function 4226
SysUtils.NegCurrFormat 4356
SysUtils.NegCurrFormat variable 4356
SysUtils.NewStr 4227
SysUtils.NewStr function 4227
SysUtils.NextCharIndex 4227
SysUtils.NextCharIndex function 4227
SysUtils.Now 4227
SysUtils.Now function 4227
SysUtils.NullAnsiStr 4357
SysUtils.NullAnsiStr variable 4357
SysUtils.NullStr 4357
SysUtils.NullStr variable 4357
SysUtils.NullWideStr 4358
SysUtils.NullWideStr variable 4358
SysUtils.OutOfMemoryError 4228
SysUtils.OutOfMemoryError function 4228
SysUtils.PathDelim 4388
SysUtils.PathDelim constant 4388
SysUtils.PathSep 4388
SysUtils.PathSep constant 4388
SysUtils.PByteArray 4337
SysUtils.PByteArray type 4337
SysUtils.PDayTable 4337
SysUtils.PDayTable type 4337
SysUtils.PExceptionRecord 4338
SysUtils.PExceptionRecord type 4338
SysUtils.pfBCB4Produced 4389
SysUtils.pfBCB4Produced constant 4389
SysUtils.pfConsumerBCB 4389
SysUtils.pfConsumerBCB constant 4389
SysUtils.pfConsumerCompat 4390
SysUtils.pfConsumerCompat constant 4390
SysUtils.pfConsumerDelphi 4390
SysUtils.pfConsumerDelphi constant 4390
SysUtils.pfConsumerMask 4391
SysUtils.pfConsumerMask constant 4391
SysUtils.pfDelphi4Produced 4391
SysUtils.pfDelphi4Produced constant 4391
SysUtils.pfDesignOnly 4392
SysUtils.pfDesignOnly constant 4392
SysUtils.pfExeModule 4392
SysUtils.pfExeModule constant 4392
SysUtils.pfIgnoreDupUnits 4392
SysUtils.pfIgnoreDupUnits constant 4392
SysUtils.pfLibraryModule 4393
SysUtils.pfLibraryModule constant 4393
SysUtils.pfModuleTypeMask 4393
SysUtils.pfModuleTypeMask constant 4393
SysUtils.pfNeverBuild 4394
SysUtils.pfNeverBuild constant 4394
SysUtils.pfPackageModule 4394
SysUtils.pfPackageModule constant 4394
SysUtils.pfProducerMask 4395
SysUtils.pfProducerMask constant 4395
SysUtils.pfProducerUndefined 4395
SysUtils.pfProducerUndefined constant 4395

SysUtils.pfRunOnly 4396
SysUtils.pfRunOnly constant 4396
SysUtils.pfV3Produced 4396
SysUtils.pfV3Produced constant 4396
SysUtils.PlatformBytesOf 4229
SysUtils.PlatformBytesOf function 4229
SysUtils.PlatformStringOf 4229
SysUtils.PlatformStringOf function 4229
SysUtils.PThreadInfo 4338
SysUtils.PThreadInfo type 4338
SysUtils.PWordArray 4338
SysUtils.PWordArray type 4338
SysUtils.QuotedStr 4229
SysUtils.QuotedStr function 4229
SysUtils.RaiseLastOSError 4230
SysUtils.RaiseLastOSError function 4230
SysUtils.RaiseLastWin32Error 4231
SysUtils.RaiseLastWin32Error function 4231
SysUtils.RCS 4232
SysUtils.RCS function 4232
SysUtils.RemoveDir 4232
SysUtils.RemoveDir function 4232
SysUtils.RenameFile 4233
SysUtils.RenameFile function 4233
SysUtils.ReplaceDate 4236
SysUtils.ReplaceDate function 4236
SysUtils.ReplaceTime 4236
SysUtils.ReplaceTime function 4236
SysUtils.RPR 4237
SysUtils.RPR function 4237
SysUtils.SafeLoadLibrary 4238
SysUtils.SafeLoadLibrary function 4238
SysUtils.SameFileName 4238
SysUtils.SameFileName function 4238
SysUtils.SameStr 4239
SysUtils.SameStr function 4239
SysUtils.SameText 4239
SysUtils.SameText function 4239
SysUtils.SecsPerDay 4397
SysUtils.SecsPerDay constant 4397
SysUtils.SecsPerHour 4397
SysUtils.SecsPerHour constant 4397
SysUtils.SecsPerMin 4398
SysUtils.SecsPerMin constant 4398
SysUtils.SetCurrentDir 4240
SysUtils.SetCurrentDir function 4240
SysUtils.ShortDateFormat 4358
SysUtils.ShortDateFormat variable 4358
SysUtils.ShortDayNames 4359
SysUtils.ShortDayNames variable 4359
SysUtils.ShortMonthNames 4359
SysUtils.ShortMonthNames variable 4359
SysUtils.ShortTimeFormat 4360
SysUtils.ShortTimeFormat variable 4360
SysUtils.ShowException 4240
SysUtils.ShowException function 4240
SysUtils.Sleep 4241
SysUtils.Sleep function 4241
SysUtils.StrAlloc 4241
SysUtils.StrAlloc function 4241
SysUtils.StrBufSize 4242
SysUtils.StrBufSize function 4242
SysUtils.StrByteType 4242
SysUtils.StrByteType function 4242
SysUtils.StrCat 4243
SysUtils.StrCat function 4243
SysUtils.StrCharLength 4244
SysUtils.StrCharLength function 4244
SysUtils.StrComp 4244
SysUtils.StrComp function 4244
SysUtils.StrCopy 4246
SysUtils.StrCopy function 4246
SysUtils.StrDispose 4247
SysUtils.StrDispose function 4247
SysUtils.StrECopy 4248
SysUtils.StrECopy function 4248
SysUtils.StrEnd 4249
SysUtils.StrEnd function 4249

SysUtils.StrFmt 4250
SysUtils.StrFmt function 4250
SysUtils.StrIComp 4251
SysUtils.StrIComp function 4251
SysUtils.StringOf 4253
SysUtils.StringOf function 4253
SysUtils.StringReplace 4253
SysUtils.StringReplace function 4253
SysUtils.StringToGUID 4254
SysUtils.StringToGUID function 4254
SysUtils.StrLCat 4255
SysUtils.StrLCat function 4255
SysUtils.StrLComp 4257
SysUtils.StrLComp function 4257
SysUtils.StrLCopy 4258
SysUtils.StrLCopy function 4258
SysUtils.StrLen 4260
SysUtils.StrLen function 4260
SysUtils.StrLFmt 4261
SysUtils.StrLFmt function 4261
SysUtils.StrLIComp 4262
SysUtils.StrLIComp function 4262
SysUtils.StrLower 4264
SysUtils.StrLower function 4264
SysUtils.StrMove 4265
SysUtils.StrMove function 4265
SysUtils.StrNew 4266
SysUtils.StrNew function 4266
SysUtils.StrNextChar 4268
SysUtils.StrNextChar function 4268
SysUtils.StrPas 4268, 4269
SysUtils.StrPas function 4268, 4269
SysUtils.StrPCopy 4271
SysUtils.StrPCopy function 4271
SysUtils.StrPLCopy 4272
SysUtils.StrPLCopy function 4272
SysUtils.StrPos 4272, 4273
SysUtils.StrPos function 4272, 4273
SysUtils.StrRScan 4274
SysUtils.StrRScan function 4274
SysUtils.StrScan 4275
SysUtils.StrScan function 4275
SysUtils.StrToBool 4276
SysUtils.StrToBool function 4276
SysUtils.StrToBoolDef 4277
SysUtils.StrToBoolDef function 4277
SysUtils.StrToCurr 4278
SysUtils.StrToCurr function 4278
SysUtils.StrToCurrDef 4278
SysUtils.StrToCurrDef function 4278
SysUtils.StrToDate 4279
SysUtils.StrToDate function 4279
SysUtils.StrToDateDef 4280
SysUtils.StrToDateDef function 4280
SysUtils.StrToDateTime 4281
SysUtils.StrToDateTime function 4281
SysUtils.StrToDateTimeDef 4282
SysUtils.StrToDateTimeDef function 4282
SysUtils.StrToFloat 4282
SysUtils.StrToFloat function 4282
SysUtils.StrToFloatDef 4286
SysUtils.StrToFloatDef function 4286
SysUtils.StrToInt 4286
SysUtils.StrToInt function 4286
SysUtils.StrToInt64 4287
SysUtils.StrToInt64 function 4287
SysUtils.StrToInt64Def 4288
SysUtils.StrToInt64Def function 4288
SysUtils.StrToIntDef 4288
SysUtils.StrToIntDef function 4288
SysUtils.StrToTime 4291
SysUtils.StrToTime function 4291
SysUtils.StrToTimeDef 4292
SysUtils.StrToTimeDef function 4292
SysUtils.StrUpper 4292
SysUtils.StrUpper function 4292
SysUtils.Supports 4294
SysUtils.Supports function 4294

SysUtils.SwitchChars 4398
SysUtils.SwitchChars constant 4398
SysUtils.SysErrorMessage 4294
SysUtils.SysErrorMessage function 4294
SysUtils.SysLocale 4360
SysUtils.SysLocale variable 4360
SysUtils.SystemTimeToDateTime 4295
SysUtils.SystemTimeToDateTime function 4295
SysUtils.TBigEndianUnicodeEncoding 3967
SysUtils.TBigEndianUnicodeEncoding.GetBytes 3968
SysUtils.TBigEndianUnicodeEncoding.GetChars 3969
SysUtils.TBigEndianUnicodeEncoding.GetPreamble 3969
SysUtils.TByteArray 4339
SysUtils.TByteArray type 4339
SysUtils.TBytes 4339
SysUtils.TBytes type 4339
SysUtils.TCharArray 4340
SysUtils.TCharArray type 4340
SysUtils.TCharSearch 4321
SysUtils.TCharSearch record 4321
SysUtils.TDayTable 4340
SysUtils.TDayTable type 4340
SysUtils.TEncoding 3969
SysUtils.TEncoding.ASCII 3985
SysUtils.TEncoding.BigEndianUnicode 3985
SysUtils.TEncoding.Convert 3975
SysUtils.TEncoding.Default 3985
SysUtils.TEncoding.FIsSingleByte 3973
SysUtils.TEncoding.FMaxCharSize 3974
SysUtils.TEncoding.FreeEncodings 3975
SysUtils.TEncoding.GetBufferEncoding 3976
SysUtils.TEncoding.GetByteCount 3976
SysUtils.TEncoding.GetBytes 3977
SysUtils.TEncoding.GetCharCount 3978
SysUtils.TEncoding.GetChars 3979
SysUtils.TEncoding.GetEncoding 3980
SysUtils.TEncoding.GetMaxByteCount 3980
SysUtils.TEncoding.GetMaxCharCount 3980
SysUtils.TEncoding.GetPreamble 3981
SysUtils.TEncoding.GetString 3981
SysUtils.TEncoding.IsSingleByte 3986
SysUtils.TEncoding.IsStandardEncoding 3984
SysUtils.TEncoding.Unicode 3986
SysUtils.TEncoding.UTF7 3986
SysUtils.TEncoding.UTF8 3987
SysUtils.TExceptionRecord 4322
SysUtils.TExceptionRecord record 4322
SysUtils.TextPos 4296
SysUtils.TextPos function 4296
SysUtils.TextToFloat 4296
SysUtils.TextToFloat function 4296
SysUtils.TFileName 4340
SysUtils.TFileName type 4340
SysUtils.TFilenameCaseMatch 4323
SysUtils.TFilenameCaseMatch enumeration 4323
SysUtils.TFloatFormat 4323
SysUtils.TFloatFormat enumeration 4323
SysUtils.TFloatRec 4325
SysUtils.TFloatRec record 4325
SysUtils.TFloatValue 4325
SysUtils.TFloatValue enumeration 4325
SysUtils.TFormatSettings 4326
SysUtils.TFormatSettings record 4326
SysUtils.TFunc 4341
SysUtils.TFunc type 4341
SysUtils.ThousandSeparator 4362
SysUtils.ThousandSeparator variable 4362
SysUtils.Time 4297
SysUtils.Time function 4297
SysUtils.TimeAMString 4362
SysUtils.TimeAMString variable 4362
SysUtils.TimePMString 4362
SysUtils.TimePMString variable 4362
SysUtils.TimeSeparator 4363
SysUtils.TimeSeparator variable 4363
SysUtils.TimeStampToDateTime 4298
SysUtils.TimeStampToDateTime function 4298
SysUtils.TimeStampToMSecs 4299

SysUtils.TimeStampToMSecs function 4299
SysUtils.TimeToStr 4299
SysUtils.TimeToStr function 4299
SysUtils.TIntegerSet 4341
SysUtils.TIntegerSet type 4341
SysUtils.TLangRec 4327
SysUtils.TLangRec record 4327
SysUtils.TLanguages 3987
SysUtils.TLanguages.Count 3989
SysUtils.TLanguages.Create 3988
SysUtils.TLanguages.Ext 3989
SysUtils.TLanguages.ID 3990
SysUtils.TLanguages.IndexOf 3988
SysUtils.TLanguages.LocaleID 3990
SysUtils.TLanguages.Name 3991
SysUtils.TLanguages.NameFromLCID 3991
SysUtils.TLanguages.NameFromLocaleID 3991
SysUtils.TLocaleOptions 4328
SysUtils.TLocaleOptions enumeration 4328
SysUtils.TMbcsByteType 4329
SysUtils.TMbcsByteType enumeration 4329
SysUtils.TMBCSEncoding 3992
SysUtils.TMBCSEncoding.Create 3993, 3994
SysUtils.TMBCSEncoding.GetByteCount 3994
SysUtils.TMBCSEncoding.GetBytes 3995
SysUtils.TMBCSEncoding.GetCharCount 3995
SysUtils.TMBCSEncoding.GetChars 3996
SysUtils.TMBCSEncoding.GetMaxByteCount 3996
SysUtils.TMBCSEncoding.GetMaxCharCount 3996
SysUtils.TMBCSEncoding.GetPreamble 3997
SysUtils.TMREWSync 4341
SysUtils.TMREWSync type 4341
SysUtils.TMultiReadExclusiveWriteSynchronizer 3997
SysUtils.TMultiReadExclusiveWriteSynchronizer.BeginRead 4000
SysUtils.TMultiReadExclusiveWriteSynchronizer.BeginWrite 4000
SysUtils.TMultiReadExclusiveWriteSynchronizer.Create 3999
SysUtils.TMultiReadExclusiveWriteSynchronizer.Destroy 3999
SysUtils.TMultiReadExclusiveWriteSynchronizer.EndRead 4001
SysUtils.TMultiReadExclusiveWriteSynchronizer.EndWrite 4001
SysUtils.TMultiReadExclusiveWriteSynchronizer.RevisionLevel 4002
SysUtils.TNameType 4329
SysUtils.TNameType enumeration 4329
SysUtils.TPackageInfoProc 4342
SysUtils.TPackageInfoProc type 4342
SysUtils.TPredicate 4342
SysUtils.TPredicate type 4342
SysUtils.TProc 4343
SysUtils.TProc type 4343
SysUtils.TProcedure 4343
SysUtils.TProcedure type 4343
SysUtils.TReplaceFlags 4343
SysUtils.TReplaceFlags type 4343
SysUtils.Trim 4301
SysUtils.Trim function 4301
SysUtils.TrimLeft 4301
SysUtils.TrimLeft function 4301
SysUtils.TrimRight 4302
SysUtils.TrimRight function 4302
SysUtils.TrueBoolStrs 4363
SysUtils.TrueBoolStrs variable 4363
SysUtils.TryEncodeDate 4302
SysUtils.TryEncodeDate function 4302
SysUtils.TryEncodeTime 4303
SysUtils.TryEncodeTime function 4303
SysUtils.TryFloatToCurr 4304
SysUtils.TryFloatToCurr function 4304
SysUtils.TryFloatToDateTime 4304
SysUtils.TryFloatToDateTime function 4304
SysUtils.TryStrToBool 4305
SysUtils.TryStrToBool function 4305
SysUtils.TryStrToCurr 4305
SysUtils.TryStrToCurr function 4305
SysUtils.TryStrToDate 4306
SysUtils.TryStrToDate function 4306
SysUtils.TryStrToDateTime 4307
SysUtils.TryStrToDateTime function 4307
SysUtils.TryStrToFloat 4307

SysUtils.TryStrToFloat function 4307
SysUtils.TryStrToInt 4308
SysUtils.TryStrToInt function 4308
SysUtils.TryStrToInt64 4309
SysUtils.TryStrToInt64 function 4309
SysUtils.TryStrToTime 4309
SysUtils.TryStrToTime function 4309
SysUtils.TSearchRec 4330
SysUtils.TSearchRec record 4330
SysUtils.TSimpleRWSync 4002
SysUtils.TSimpleRWSync.BeginRead 4005
SysUtils.TSimpleRWSync.BeginWrite 4005
SysUtils.TSimpleRWSync.Create 4004
SysUtils.TSimpleRWSync.Destroy 4004
SysUtils.TSimpleRWSync.EndRead 4006
SysUtils.TSimpleRWSync.EndWrite 4006
SysUtils.TStringBuilder 4006
SysUtils.TStringBuilder.Append 4014
SysUtils.TStringBuilder.AppendFormat 4015
SysUtils.TStringBuilder.AppendLine 4016
SysUtils.TStringBuilder.Capacity 4021
SysUtils.TStringBuilder.Chars 4022
SysUtils.TStringBuilder.CopyTo 4016
SysUtils.TStringBuilder.Create 4011
SysUtils.TStringBuilder.EnsureCapacity 4016
SysUtils.TStringBuilder.Equals 4017
SysUtils.TStringBuilder.FData 4012
SysUtils.TStringBuilder.FLength 4012
SysUtils.TStringBuilder.FMaxCapacity 4012
SysUtils.TStringBuilder.Insert 4018
SysUtils.TStringBuilder.Length 4022
SysUtils.TStringBuilder.MaxCapacity 4022
SysUtils.TStringBuilder.Remove 4019
SysUtils.TStringBuilder.Replace 4020
SysUtils.TStringBuilder.ToString 4021
SysUtils.TSysCharSet 4344
SysUtils.TSysCharSet type 4344
SysUtils.TSysLocale 4333
SysUtils.TSysLocale record 4333
SysUtils.TTerminateProc 4344
SysUtils.TTerminateProc type 4344
SysUtils.TThreadInfo 4334
SysUtils.TThreadInfo record 4334
SysUtils.TThreadLocalCounter 4023
SysUtils.TThreadLocalCounter.Close 4024
SysUtils.TThreadLocalCounter.Delete 4024
SysUtils.TThreadLocalCounter.Destroy 4023
SysUtils.TThreadLocalCounter.Open 4024
SysUtils.TTimeStamp 4334
SysUtils.TTimeStamp record 4334
SysUtils.TUnicodeEncoding 4024
SysUtils.TUnicodeEncoding.Create 4026
SysUtils.TUnicodeEncoding.GetByteCount 4026
SysUtils.TUnicodeEncoding.GetBytes 4027
SysUtils.TUnicodeEncoding.GetCharCount 4027
SysUtils.TUnicodeEncoding.GetChars 4028
SysUtils.TUnicodeEncoding.GetMaxByteCount 4028
SysUtils.TUnicodeEncoding.GetMaxCharCount 4028
SysUtils.TUnicodeEncoding.GetPreamble 4029
SysUtils.TUTF7Encoding 4029
SysUtils.TUTF7Encoding.Create 4030
SysUtils.TUTF7Encoding.GetByteCount 4031
SysUtils.TUTF7Encoding.GetBytes 4031
SysUtils.TUTF7Encoding.GetCharCount 4032
SysUtils.TUTF7Encoding.GetChars 4032
SysUtils.TUTF7Encoding.GetMaxByteCount 4033
SysUtils.TUTF7Encoding.GetMaxCharCount 4033
SysUtils.TUTF8Encoding 4034
SysUtils.TUTF8Encoding.Create 4035
SysUtils.TUTF8Encoding.GetMaxByteCount 4035
SysUtils.TUTF8Encoding.GetMaxCharCount 4036
SysUtils.TUTF8Encoding.GetPreamble 4036
SysUtils.TValidatePackageProc 4345
SysUtils.TValidatePackageProc type 4345
SysUtils.TwoDigitYearCenturyWindow 4364
SysUtils.TwoDigitYearCenturyWindow variable 4364
SysUtils.TWordArray 4345
SysUtils.TWordArray type 4345

SysUtils.ufImplicitUnit 4399
SysUtils.ufImplicitUnit constant 4399
SysUtils.ufMainUnit 4399
SysUtils.ufMainUnit constant 4399
SysUtils.ufOrgWeakUnit 4400
SysUtils.ufOrgWeakUnit constant 4400
SysUtils.ufPackageUnit 4400
SysUtils.ufPackageUnit constant 4400
SysUtils.ufWeakPackageUnit 4400
SysUtils.ufWeakPackageUnit constant 4400
SysUtils.ufWeakUnit 4401
SysUtils.ufWeakUnit constant 4401
SysUtils.UIntToStr 4310
SysUtils.UIntToStr function 4310
SysUtils.UnixDateDelta 4401
SysUtils.UnixDateDelta constant 4401
SysUtils.UnloadPackage 4310
SysUtils.UnloadPackage function 4310
SysUtils.UpperCase 4311
SysUtils.UpperCase function 4311
SysUtils.WideBytesOf 4312
SysUtils.WideBytesOf function 4312
SysUtils.WideCompareStr 4313
SysUtils.WideCompareStr function 4313
SysUtils.WideCompareText 4313
SysUtils.WideCompareText function 4313
SysUtils.WideFmtStr 4314
SysUtils.WideFmtStr function 4314
SysUtils.WideFormat 4314
SysUtils.WideFormat function 4314
SysUtils.WideFormatBuf 4315
SysUtils.WideFormatBuf function 4315
SysUtils.WideLowerCase 4316
SysUtils.WideLowerCase function 4316
SysUtils.WideSameStr 4316
SysUtils.WideSameStr function 4316
SysUtils.WideSameText 4316
SysUtils.WideSameText function 4316
SysUtils.WideStrAlloc 4317
SysUtils.WideStrAlloc function 4317
SysUtils.WideStringOf 4317
SysUtils.WideStringOf function 4317
SysUtils.WideUpperCase 4318
SysUtils.WideUpperCase function 4318
SysUtils.Win32BuildNumber 4364
SysUtils.Win32BuildNumber variable 4364
SysUtils.Win32Check 4318
SysUtils.Win32Check function 4318
SysUtils.Win32CSDVersion 4365
SysUtils.Win32CSDVersion variable 4365
SysUtils.Win32MajorVersion 4365
SysUtils.Win32MajorVersion variable 4365
SysUtils.Win32MinorVersion 4366
SysUtils.Win32MinorVersion variable 4366
SysUtils.Win32Platform 4366
SysUtils.Win32Platform variable 4366
SysUtils.WordRec 4335
SysUtils.WordRec record 4335
SysUtils.WrapText 4318
SysUtils.WrapText function 4318
Sysutils::IntToStr 3900
Sysutils::IntToStr function 3900
szName
SPParamDesc class 1515
TFLDDesc class 1577

T

T*
DelphiInterface template 3261
t_str
UnicodeString class 3409
TabFont
TTabbedNotebook class 4430
TabFontChanged
TTabbedNotebook class 4412
TabHeight
TTabSet class 4505
TabIndex

TTabSet class 4506
TableAttributes
TAdapterGrid class 6425
TCustomAdapterGrid class 6503
TableHeader
TAdapterFormLayout class 6418
TableIndent
TAdapterFormLayout class 6419
TableName
TSQLTable class 1693
TableScope
TSQLConnection class 1623
TabNotBk 4403
TabNotBk namespace 4403
Classes 4404
Constants 4455
Types 4454
TabNotBk.CM_TABFONTCHANGED 4455
TabNotBk.CM_TABFONTCHANGED constant 4455
TabNotBk.TPageChangeEvent 4454
TabNotBk.TPageChangeEvent type 4454
TabNotBk.TTabbedNotebook 4404
TabNotBk.TTabbedNotebook.ActivePage 4413
TabNotBk.TTabbedNotebook.Align 4414
TabNotBk.TTabbedNotebook.AlignControls 4407
TabNotBk.TTabbedNotebook.Anchors 4417
TabNotBk.TTabbedNotebook.Change 4407
TabNotBk.TTabbedNotebook.Click 4407
TabNotBk.TTabbedNotebook.CMTabFontChanged 4408
TabNotBk.TTabbedNotebook.Constraints 4418
TabNotBk.TTabbedNotebook.Create 4405
TabNotBk.TTabbedNotebook.CreateHandle 4408
TabNotBk.TTabbedNotebook.CreateParams 4409
TabNotBk.TTabbedNotebook.Destroy 4406
TabNotBk.TTabbedNotebook.DoubleBuffered 4418
TabNotBk.TTabbedNotebook.Enabled 4420
TabNotBk.TTabbedNotebook.Font 4420
TabNotBk.TTabbedNotebook.GetChildOwner 4409
TabNotBk.TTabbedNotebook.GetChildren 4409
TabNotBk.TTabbedNotebook.GetIndexForPage 4410
TabNotBk.TTabbedNotebook.KeyDown 4410
TabNotBk.TTabbedNotebook.Loaded 4411
TabNotBk.TTabbedNotebook.OnChange 4435
TabNotBk.TTabbedNotebook.OnClick 4436
TabNotBk.TTabbedNotebook.OnContextPopup 4436
TabNotBk.TTabbedNotebook.OnEnter 4437
TabNotBk.TTabbedNotebook.OnExit 4439
TabNotBk.TTabbedNotebook.PageIndex 4421
TabNotBk.TTabbedNotebook.Pages 4422
TabNotBk.TTabbedNotebook.ParentDoubleBuffered 4422
TabNotBk.TTabbedNotebook.ParentShowHint 4423
TabNotBk.TTabbedNotebook.PopupMenu 4423
TabNotBk.TTabbedNotebook.ReadState 4411
TabNotBk.TTabbedNotebook.SetPageIndex 4412
TabNotBk.TTabbedNotebook.ShowControl 4412
TabNotBk.TTabbedNotebook.ShowHint 4428
TabNotBk.TTabbedNotebook.TabFont 4430
TabNotBk.TTabbedNotebook.TabFontChanged 4412
TabNotBk.TTabbedNotebook.TabOrder 4430
TabNotBk.TTabbedNotebook.Tabs 4431
TabNotBk.TTabbedNotebook.TabsPerRow 4432
TabNotBk.TTabbedNotebook.TabStop 4433
TabNotBk.TTabbedNotebook.TopFont 4433
TabNotBk.TTabbedNotebook.Visible 4433
TabNotBk.TTabPage 4443
TabNotBk.TTabPage.Caption 4445
TabNotBk.TTabPage.Create 4444
TabNotBk.TTabPage.Enabled 4448
TabNotBk.TTabPage.Height 4449
TabNotBk.TTabPage.ReadState 4445
TabNotBk.TTabPage.TabOrder 4450
TabNotBk.TTabPage.Visible 4450
TabNotBk.TTabPage.Width 4452
TabOrder
TButton class 2069
TCheckBox class 2148
TComboBox class 2223
TEdit class 2497

TGroupBox class 2563
TListBox class 2695
TMemo class 2771
TRadioButton class 2844
TScrollBar class 2917
TStaticText class 2974
TTabbedNotebook class 4430
TTabPage class 4450
TValueListEditor class 4834
TWebBrowser class 542
TWebBrowser_V1 class 593
TabPosition
TTabSet class 4506
Tabs 4457
TTabbedNotebook class 4431
TTabSet class 4506
Tabs namespace 4457
Classes 4458
Structs, Records, Enums 4538
Types 4541
Tabs.TDrawTabEvent 4541
Tabs.TDrawTabEvent type 4541
Tabs.TEdgePart 4538
Tabs.TEdgePart enumeration 4538
Tabs.TEdgeType 4538
Tabs.TEdgeType enumeration 4538
Tabs.TMeasureTabEvent 4541
Tabs.TMeasureTabEvent type 4541
Tabs.TScrollBtn 4539
Tabs.TScrollBtn enumeration 4539
Tabs.TScroller 4458
Tabs.TScroller.Change 4461
Tabs.TScroller.Create 4459
Tabs.TScroller.Destroy 4460
Tabs.TScroller.Max 4461
Tabs.TScroller.Min 4462
Tabs.TScroller.OnClick 4463
Tabs.TScroller.Paint 4460
Tabs.TScroller.Position 4462
Tabs.TScroller.ScrollOrientation 4462
Tabs.TScrollOrientation 4539
Tabs.TScrollOrientation enumeration 4539
Tabs.TTabChangeEvent 4542
Tabs.TTabChangeEvent type 4542
Tabs.TTabList 4463
Tabs.TTabList.Add 4464
Tabs.TTabList.AddStrings 4467
Tabs.TTabList.Clear 4468
Tabs.TTabList.Create 4464
Tabs.TTabList.Delete 4469
Tabs.TTabList.Insert 4469
Tabs.TTabList.Put 4470
Tabs.TTabSet 4470
Tabs.TTabSet.Align 4481
Tabs.TTabSet.Anchors 4485
Tabs.TTabSet.AutoScroll 4485
Tabs.TTabSet.BackgroundColor 4486
Tabs.TTabSet.CanChange 4474
Tabs.TTabSet.Canvas 4486
Tabs.TTabSet.CMHintShow 4474
Tabs.TTabSet.Constraints 4488
Tabs.TTabSet.Create 4473
Tabs.TTabSet.CreateParams 4475
Tabs.TTabSet.Destroy 4473
Tabs.TTabSet.DitherBackground 4489
Tabs.TTabSet.DoubleBuffered 4489
Tabs.TTabSet.DragCursor 4491
Tabs.TTabSet.DragKind 4491
Tabs.TTabSet.DragMode 4492
Tabs.TTabSet.DrawTab 4475
Tabs.TTabSet.Enabled 4493
Tabs.TTabSet.EndMargin 4494
Tabs.TTabSet.FirstIndex 4494
Tabs.TTabSet.Font 4494
Tabs.TTabSet.GetChildren 4475
Tabs.TTabSet.GetImageIndex 4476
Tabs.TTabSet.Images 4496
Tabs.TTabSet.ItemAtPos 4476

Tabs.TTabSet.ItemRect 4476
Tabs.TTabSet.ItemWidth 4477
Tabs.TTabSet.MeasureTab 4477
Tabs.TTabSet.MinClientRect 4477
Tabs.TTabSet.MouseDown 4478
Tabs.TTabSet.Notification 4479
Tabs.TTabSet.OnAlignInsertBefore 4509
Tabs.TTabSet.OnAlignPosition 4510
Tabs.TTabSet.OnChange 4511
Tabs.TTabSet.OnClick 4511
Tabs.TTabSet.OnDragDrop 4513
Tabs.TTabSet.OnDragOver 4516
Tabs.TTabSet.OnDrawTab 4519
Tabs.TTabSet.OnEndDock 4519
Tabs.TTabSet.OnEndDrag 4520
Tabs.TTabSet.OnEnter 4521
Tabs.TTabSet.OnExit 4523
Tabs.TTabSet.OnGetImageIndex 4527
Tabs.TTabSet.OnMeasureTab 4527
Tabs.TTabSet.OnMouseActivate 4528
Tabs.TTabSet.OnMouseDown 4529
Tabs.TTabSet.OnMouseEnter 4531
Tabs.TTabSet.OnMouseLeave 4531
Tabs.TTabSet.OnMouseMove 4532
Tabs.TTabSet.OnMouseUp 4534
Tabs.TTabSet.OnStartDock 4536
Tabs.TTabSet.OnStartDrag 4537
Tabs.TTabSet.Paint 4479
Tabs.TTabSet.ParentBackground 4496
Tabs.TTabSet.ParentDoubleBuffered 4496
Tabs.TTabSet.ParentShowHint 4496
Tabs.TTabSet.PopupMenu 4497
Tabs.TTabSet.Scroller 4502
Tabs.TTabSet.SelectedColor 4502
Tabs.TTabSet.SelectNext 4480
Tabs.TTabSet.SetTabIndex 4480
Tabs.TTabSet.ShowHint 4502
Tabs.TTabSet.ShrinkToFit 4504
Tabs.TTabSet.SoftTop 4504
Tabs.TTabSet.StartMargin 4504
Tabs.TTabSet.Style 4505
Tabs.TTabSet.TabHeight 4505
Tabs.TTabSet.TabIndex 4506
Tabs.TTabSet.TabPosition 4506
Tabs.TTabSet.Tabs 4506
Tabs.TTabSet.UnselectedColor 4507
Tabs.TTabSet.Visible 4507
Tabs.TTabSet.VisibleTabs 4508
Tabs.TTabSetDrawTabEvent 4542
Tabs.TTabSetDrawTabEvent type 4542
Tabs.TTabSetTabStyle 4540
Tabs.TTabSetTabStyle enumeration 4540
Tabs.TTabStyle 4543
Tabs.TTabStyle type 4543
TabsPerRow
  TTabbedNotebook class 4432
TabStop
  TButton class 2070
  TCheckBox class 2148
  TComboBox class 2224
  TCustomButton class 2275
  TCustomCheckBox class 2282
  TCustomCombo class 2317
  TCustomEdit class 2379
  TCustomListBox class 2436
  TEdit class 2497
  TGroupBox class 2563
  TListBox class 2696
  TMemo class 2772
  TRadioButton class 2845
  TScrollBar class 2918
  TStaticText class 2975
  TTabbedNotebook class 4433
  TWebBrowser class 542
  TWebBrowser_V1 class 594
TAbstractAdapterRequest class 5955
  about TAbstractAdapterRequest class 5955
  Create 5957

Request 5957
TAbstractAdapterRequest Members 5956
TAbstractAdapterRequest properties 5957
TAbstractAdapterResponse class 5958
about TAbstractAdapterResponse class 5958
Create 5959
Response 5960
TAbstractAdapterResponse Members 5958
TAbstractAdapterResponse properties 5960
TAbstractControlRequirements class 6311
about TAbstractControlRequirements class 6311
DefineFunction 6312
DefineVariable 6312
IsFunctionDefined 6313
IsVariableDefined 6313
RequiresMultipartForm 6314
TAbstractControlRequirements Members 6311
TAbstractControlRequirements methods 6312
TAbstractDesigntimeWarnings class 963
about TAbstractDesigntimeWarnings class 963
AddObject 964
AddString 964
TAbstractDesigntimeWarnings Members 963
TAbstractDesigntimeWarnings methods 964
TAbstractFieldValuesAdapter class 5180
about TAbstractFieldValuesAdapter class 5180
GetField 5186
GetValueFieldClass 5186
GetValueIndex 5187
ImplCanAddFieldClass 5187
ImplEndIterator 5188
ImplGetField 5188
ImplGetFieldsList 5188
ImplGetValueIndex 5189
ImplNextIteration 5189
ImplStartIterator 5190
TAbstractFieldValuesAdapter Members 5182
TAbstractFieldValuesAdapter methods 5185
TAbstractFormRequirements class 6314
about TAbstractFormRequirements class 6314
DefineFunction 6315
DefineVariable 6315
IsFunctionDefined 6315
IsVariableDefined 6316
TAbstractFormRequirements Members 6314
TAbstractFormRequirements methods 6315
TAbstractInetFileType class 965
about TAbstractInetFileType class 965
FileExt 967
FileExtCount 967
GetExtCount 966
GetFileExt 966
GetTypeID 967
TAbstractInetFileType Members 965
TAbstractInetFileType methods 966
TAbstractInetFileType properties 967
TypeID 968
TAbstractNamedVariants class 6118
about TAbstractNamedVariants class 6118
Add 6121
Append 6122
Assign 6122
BeginUpdate 6122
Capacity 6131
Clear 6123
CompareStrings 6123
Count 6131
DefineProperties 6124
Delete 6124
Destroy 6120
EndUpdate 6125
Equals 6125
Error 6126
Exchange 6126
Get 6126
GetCapacity 6127
GetCount 6127
IndexOfName 6128

Insert 6128
Move 6129
Names 6132
Put 6129
SetCapacity 6130
SetUpdateState 6130
TAbstractNamedVariants Members 6119
TAbstractNamedVariants methods 6120
TAbstractNamedVariants properties 6131
UpdateCount 6132
Values 6133
Variants 6133
TAbstractScriptEngine class 6648
about TAbstractScriptEngine class 6648
AddObjectFactory 6650
Create 6649
CreateComponentStub 6651
Destroy 6649
ListLanguageNames 6651
Name 6651
ScriptObjectFactories 6652
ScriptProducerClass 6652
SupportsLanguage 6651
TAbstractScriptEngine Members 6648
TAbstractScriptEngine methods 6650
TAbstractScriptEngine properties 6651
TAbstractScriptObjectFactories class 6652
about TAbstractScriptObjectFactories class 6652
Count 6655
Factories 6655
GetCount 6654
GetFactory 6654
GetObjectList 6654
ObjectList 6655
TAbstractScriptObjectFactories Members 6652
TAbstractScriptObjectFactories methods 6653
TAbstractScriptObjectFactories properties 6654
TAbstractScriptObjectFactory class 6655
about TAbstractScriptObjectFactory class 6655
AddGlobalObjects 6656
CreateComponentObject 6656
CreateGlobalObject 6657
CreateProducerObject 6657
GetScriptEngine 6657
TAbstractScriptObjectFactory Members 6656
TAbstractScriptObjectFactory methods 6656
TAbstractSessionIDGen class 374
about TAbstractSessionIDGen class 374
NextID 375
TAbstractSessionIDGen Members 375
TAbstractSessionIDGen methods 375
TAbstractSessionStatusChange class 375
about TAbstractSessionStatusChange class 375
SessionStarted 376
SessionTerminated 376
SessionTimeout 377
TAbstractSessionStatusChange Members 376
TAbstractSessionStatusChange methods 376
TAbstractSocket class 14
about TAbstractSocket class 14
Active 21
Address 22
Close 16
DoActivate 16
Error 16
Event 17
Host 22
InitSocket 18
Loaded 18
Open 19
Port 23
Service 23
SetActive 19
SetAddress 20
SetHost 20
SetPort 20
SetService 21
TAbstractSocket Members 15

TAbstractSocket methods 15
TAbstractSocket properties 21
TAbstractWebContext class 5960
about TAbstractWebContext class 5960
AdapterRequest 5971
AdapterResponse 5972
Create 5962
Destroy 5963
DispatchedPageName 5972
EndUser 5973
FindApplicationModule 5964
FindDefaultPageInfo 5964
FindModuleClass 5965
FindModuleName 5965
FindPage 5966
FindPageInfo 5966
FindVariableContainer 5967
GetAdapterRequest 5967
GetAdapterResponse 5967
GetDispatchedPageName 5968
GetEndUser 5968
GetPageCount 5968
GetPageInfo 5968
GetRequestedPageName 5969
GetSession 5969
GetWebModules 5969
GetWebRequest 5969
GetWebResponse 5970
PageCount 5973
Pages 5973
Request 5974
RequestedPageName 5974
Response 5975
Session 5975
SetAdapterRequest 5970
SetAdapterResponse 5970
SetDispatchedPageName 5970
SetRequestedPageName 5970
TAbstractWebContext Members 5961
TAbstractWebContext methods 5963
TAbstractWebContext properties 5971
WebModules 5975
TabWidth
TCustomListBox class 2437
TListBox class 2696
TAdapter class 5190
about TAdapter class 5190
Actions 5195
Data 5196
ModifyAccess 5196
OnAfterExecuteAction 5197
OnAfterGetActionResponse 5197
OnAfterUpdateFields 5198
OnAfterValidateFields 5198
OnBeforeExecuteAction 5198
OnBeforeGetActionResponse 5199
OnBeforeUpdateFields 5199
OnBeforeValidateFields 5200
OnGetActionParams 5200
OnIterateRecords 5200
TAdapter events 5197
TAdapter Members 5192
TAdapter properties 5195
ViewAccess 5196
TAdapterAction class 5201
about TAdapterAction class 5201
ActionName 5205
DisplayLabel 5205
ExecuteAccess 5206
OnAfterExecute 5207
OnAfterGetResponse 5207
OnBeforeExecute 5208
OnBeforeGetResponse 5208
OnExecute 5208
OnGetEnabled 5209
OnGetParams 5209
RedirectOptions 5206
TAdapterAction events 5206

TAdapterAction Members 5202
TAdapterAction properties 5205
TAdapterActionButton class 6316
about TAdapterActionButton class 6316
ActionName 6318
Align 6318
Caption 6319
Custom 6319
DisplayColumns 6320
DisplayType 6320
ErrorPageName 6321
HideOptions 6321
ImageHeight 6321
ImageProducer 6322
ImageWidth 6322
PageName 6323
Style 6323
StyleRule 6323
TAdapterActionButton Members 6317
TAdapterActionButton properties 6318
VAlign 6324
TAdapterActionName class 6536
about TAdapterActionName class 6536
Action 6538
ActionName 6539
Adapter 6539
AssignTo 6537
Create 6537
TAdapterActionName Members 6536
TAdapterActionName methods 6537
TAdapterActionName properties 6538
TAdapterActions class 5209
about TAdapterActions class 5209
AdapterCanAddClass 5212
ImplCanAddClass 5212
ImplGetWebComponentNodeName 5213
SetWebParent 5213
TAdapterActions Members 5210
TAdapterActions methods 5212
TAdapterActions properties 5214
WebActions 5214
TAdapterActionsListHelper class 6324
about TAdapterActionsListHelper class 6324
AddDefaultActions 6331
Create 6326
FindAction 6327
GetActionsList 6327
GetAddAllFieldsItem 6328
GetAddFieldsDlgCaption 6328
GetAddFieldsItem 6329
GetNewFieldPrefix 6329
GetVisibleActions 6329
IsActionInUse 6330
TAdapterActionsListHelper Members 6325
TAdapterActionsListHelper methods 6327
TAdapterActionsListHelper properties 6330
TAdapterApplicationField class 5214
about TAdapterApplicationField class 5214
Adapter 5218
TAdapterApplicationField Members 5215
TAdapterApplicationField properties 5218
TAdapterApplicationTitleField class 5218
about TAdapterApplicationTitleField class 5218
GetDefaultFieldName 5222
ImplGetValue 5222
TAdapterApplicationTitleField Members 5219
TAdapterApplicationTitleField methods 5222
TAdapterBooleanField class 5223
about TAdapterBooleanField class 5223
DisplayLabel 5228
DisplayWidth 5228
FieldName 5228
ModifyAccess 5229
OnGetValue 5230
OnUpdateValue 5230
OnValidateValue 5231
ReadOnly 5229
TAdapterBooleanField events 5230

TAdapterBooleanField Members 5224
TAdapterBooleanField properties 5227
ValuesList 5229
ViewAccess 5230
TAdapterCommandColumn class 6331
about TAdapterCommandColumn class 6331
AddDefaultCommands 6334
Align 6334
Caption 6334
CaptionAttributes 6335
Custom 6335
DisplayColumns 6335
Style 6336
StyleRule 6336
TAdapterCommandColumn Members 6332
TAdapterCommandColumn properties 6333
VAlign 6337
TAdapterCommandGroup class 6337
about TAdapterCommandGroup class 6337
AddDefaultCommands 6340
Custom 6340
DisplayComponent 6340
Style 6341
StyleRule 6341
TAdapterCommandGroup Members 6338
TAdapterCommandGroup properties 6339
TAdapterControlGroup class 6341
about TAdapterControlGroup class 6341
Adapter 6352
AdapterChange 6345
AdapterMode 6353
AddDefaultFields 6353
AddNotify 6345
Create 6343
CreateDefaultFields 6346
Destroy 6344
FindField 6346
GetAdapter 6346
GetAdapterModeOfAdapter 6347
GetAdapterModeOfDisplay 6347
GetAdapterVariableName 6348
GetControlRequirements 6348
GetDesigntimeWarnings 6348
GetFieldsList 6349
GetVisibleFields 6349
ImplGetControlRequirements 6350
IsFieldInUse 6350
Notification 6350
NotifyAdapterChange 6351
RemoveNotify 6351
SetAdapter 6352
TAdapterControlGroup Members 6342
TAdapterControlGroup methods 6344
TAdapterControlGroup properties 6352
VisibleFields 6354
TAdapterDataDisplay class 6354
about TAdapterDataDisplay class 6354
Align 6362
Create 6356
FieldConditions 6357
FieldName 6363
FindAssociatedField 6357
FormatCaption 6357
FormatColumnData 6358
FormatColumnHeading 6358
GetAdapterModeOfAdapter 6358
GetAdapterVariableName 6358
GetCaption 6358
GetControlRequirements 6359
GetDesigntimeWarnings 6359
GetFieldName 6359
GetVariablesContainer 6360
HideOptions 6363
ImplContent 6360
ImplFormatColumnData 6360
ImplFormatColumnHeading 6361
ImplGetControlRequirements 6361
ImplRestoreDefaults 6361

RestoreDefaults 6361
SetCaption 6362
SetFieldName 6362
TAdapterDataDisplay Members 6355
TAdapterDataDisplay methods 6356
TAdapterDataDisplay properties 6362
VAlign 6363
TAdapterDefaultActions class 5231
about TAdapterDefaultActions class 5231
ClearDefaultComponents 5234
DefaultWebComponents 5237
Destroy 5233
GetAddAllFieldsItem 5235
GetAddFieldsDlgCaption 5235
GetAddFieldsItem 5235
GetDefaultWebComponents 5235
GetNewFieldPrefix 5236
GetVariablesInUse 5236
SetDefaultWebComponents 5236
TAdapterDefaultActions Members 5232
TAdapterDefaultActions methods 5234
TAdapterDefaultActions properties 5237
TAdapterDefaultFields class 5237
about TAdapterDefaultFields class 5237
ClearDefaultComponents 5240
DefaultWebComponents 5243
Destroy 5240
GetAddAllFieldsItem 5241
GetAddFieldsDlgCaption 5241
GetAddFieldsItem 5241
GetDefaultWebComponents 5241
GetNewFieldPrefix 5242
GetVariablesInUse 5242
SetDefaultWebComponents 5242
TAdapterDefaultFields Members 5238
TAdapterDefaultFields methods 5240
TAdapterDefaultFields properties 5243
TAdapterDispatcher class 6148
about TAdapterDispatcher class 6148
OnActionRequestNotHandled 6150
OnAfterDispatchAction 6151
OnAfterDispatchImage 6151
OnBeforeDispatchAction 6151
OnBeforeDispatchImage 6152
OnGetAdapterRequestParams 6152
OnImageRequestNotHandled 6153
TAdapterDispatcher events 6150
TAdapterDispatcher Members 6149
TAdapterDispatchParams class 6153
about TAdapterDispatchParams class 6153
Create 6156
Destroy 6156
GetFailurePage 6157
GetHandler 6157
GetObjectIdentifier 6157
GetParams 6158
GetRequestIdentifier 6158
GetSuccessPage 6158
Handler 6159
ObjectIdentifier 6159
RequestIdentifier 6160
TAdapterDispatchParams Members 6154
TAdapterDispatchParams methods 6157
TAdapterDispatchParams properties 6159
TAdapterDisplayColumn class 6364
about TAdapterDisplayColumn class 6364
Align 6366
Caption 6367
CaptionAttributes 6367
Custom 6367
DisplayType 6368
FieldName 6368
HideOptions 6368
ImageHeight 6369
ImageWidth 6369
Style 6369
StyleRule 6370
TAdapterDisplayColumn Members 6364

TAdapterDisplayColumn properties 6366
VAlign 6370
TAdapterDisplayField class 6371
about TAdapterDisplayField class 6371
Align 6374
Caption 6374
CaptionAttributes 6374
CaptionPosition 6375
Custom 6375
DisplayColumns 6376
DisplayRows 6376
DisplayType 6376
DisplayWidth 6377
FieldName 6377
HideOptions 6377
ImageHeight 6377
ImageWidth 6378
InputType 6378
MaxLength 6378
SelectRows 6379
Style 6379
StyleRule 6379
TAdapterDisplayField Members 6372
TAdapterDisplayField properties 6373
TextAreaWrap 6380
VAlign 6380
ViewMode 6381
TAdapterDisplayReferenceGroup class 6381
about TAdapterDisplayReferenceGroup class 6381
AdapterChange 6383
Create 6382
Destroy 6382
DisplayComponent 6386
DisplayComponentChange 6384
GetAdapterDisplayCharacteristics 6384
GetDesigntimeWarnings 6384
NotifyAdapterChange 6385
OnDisplayComponentChange 6385
SetWebDisplayComponent 6386
TAdapterDisplayReferenceGroup Members 6382
TAdapterDisplayReferenceGroup methods 6383
TAdapterDisplayReferenceGroup properties 6386
TAdapterEditColumn class 6387
about TAdapterEditColumn class 6387
Align 6389
Caption 6390
CaptionAttributes 6390
CaptionPosition 6390
Custom 6391
DisplayColumns 6391
DisplayRows 6392
DisplayType 6392
DisplayWidth 6392
FieldName 6393
HideOptions 6393
ImageHeight 6393
ImageWidth 6393
InputType 6394
MaxLength 6394
SelectRows 6394
Style 6395
StyleRule 6395
TAdapterEditColumn Members 6387
TAdapterEditColumn properties 6389
TextAreaWrap 6396
VAlign 6396
ViewMode 6396
TAdapterEndUserDisplayNameField class 5244
about TAdapterEndUserDisplayNameField class 5244
GetDefaultFieldName 5247
ImplGetValue 5247
TAdapterEndUserDisplayNameField Members 5244
TAdapterEndUserDisplayNameField methods 5247
TAdapterEndUserField class 5248
about TAdapterEndUserField class 5248
Adapter 5251
TAdapterEndUserField Members 5249
TAdapterEndUserField properties 5251

TAdapterEndUserLoggedInField class 5252
about TAdapterEndUserLoggedInField class 5252
GetDefaultFieldName 5255
ImplGetValue 5255
TAdapterEndUserLoggedInField Members 5252
TAdapterEndUserLoggedInField methods 5255
TAdapterError class 5256
about TAdapterError class 5256
Create 5258
GetErrorText 5258
GetField 5258
GetID 5259
TAdapterError Members 5256
TAdapterError methods 5258
TAdapterErrorList class 6397
about TAdapterErrorList class 6397
Adapter 6398
TAdapterErrorList Members 6397
TAdapterErrorList properties 6398
TAdapterErrorsList class 5259
about TAdapterErrorsList class 5259
AddError 5263
AddError method 5263
Create 5262
Destroy 5262
FieldNames 5266
IDs 5266
ImplClearErrors 5264
ImplDefineLabel 5264
ImplGetError 5264
ImplGetErrorCount 5265
ImplGetObjectErrors 5265
ImplHasObjectErrors 5265
Messages 5266
TAdapterErrorsList Members 5260
TAdapterErrorsList methods 5263
TAdapterErrorsList properties 5265
TAdapterField class 5267
about TAdapterField class 5267
DisplayLabel 5272
DisplayWidth 5272
FieldName 5272
ModifyAccess 5273
OnGetDisplayText 5275
OnGetValue 5275
OnUpdateValue 5275
OnValidateValue 5276
ReadOnly 5273
Required 5273
TAdapterField events 5275
TAdapterField Members 5268
TAdapterField properties 5271
ValuesList 5274
ViewAccess 5274
TAdapterFieldGroup class 6399
about TAdapterFieldGroup class 6399
Adapter 6402
AdapterMode 6402
AddDefaultFields 6403
Custom 6403
Style 6404
StyleRule 6404
TAdapterFieldGroup Members 6400
TAdapterFieldGroup properties 6401
TAdapterFields class 5276
about TAdapterFields class 5276
AdapterCanAddClass 5278
ImplCanAddClass 5279
ImplGetWebComponentNodeName 5280
SetWebParent 5280
TAdapterFields Members 5277
TAdapterFields methods 5278
TAdapterFields properties 5280
WebFields 5281
TAdapterFileField class 5281
about TAdapterFileField class 5281
DisplayLabel 5287
DisplayWidth 5287

FieldName 5288
ModifyAccess 5288
OnGetFileCount 5289
OnGetFileNames 5290
OnUploadFiles 5290
OnValidateFiles 5290
ReadOnly 5288
Required 5289
TAdapterFileField events 5289
TAdapterFileField Members 5283
TAdapterFileField properties 5287
ViewAccess 5289
TAdapterForm class 6404
about TAdapterForm class 6404
Custom 6406
FormMethod 6406
Style 6407
StyleRule 6407
TAdapterForm Members 6405
TAdapterForm properties 6406
TAdapterFormLayout class 6408
about TAdapterFormLayout class 6408
BreakButtons 6417
ChildTableIndent 6417
ColumnCount 6417
Create 6410
EndButtons 6411
EndField 6411
EndFields 6411
EndLayout 6412
EndRow 6412
EndTable 6412
FieldIndent 6418
GetAdapterFormLayout 6412
GetChildTableIndent 6413
GetFieldIndent 6413
GetTableIndent 6413
ImplLayoutButton 6413
ImplLayoutField 6414
ImplLayoutLabelAndField 6414
ImplLayoutTable 6415
NextColumn 6415
StartButtons 6415
StartFields 6416
StartRow 6416
StartTable 6416
TableHeader 6418
TableIndent 6419
TAdapterFormLayout Members 6408
TAdapterFormLayout methods 6410
TAdapterFormLayout properties 6416
TAdapterGotoPageAction class 5291
about TAdapterGotoPageAction class 5291
ExecuteAccess 5296
OnAfterExecute 5297
OnAfterGetResponse 5297
OnBeforeExecute 5297
OnBeforeGetResponse 5298
OnGetPageName 5298
OnGetParams 5298
RedirectOptions 5296
TAdapterGotoPageAction events 5296
TAdapterGotoPageAction Members 5292
TAdapterGotoPageAction properties 5295
TAdapterGrid class 6419
about TAdapterGrid class 6419
Adapter 6422
AdapterMode 6423
AddDefaultFields 6423
HeadingAttributes 6424
RowAttributes 6424
TableAttributes 6425
TAdapterGrid Members 6421
TAdapterGrid properties 6422
TAdapterHiddenField class 5299
about TAdapterHiddenField class 5299
Create 5301
GetName 5301

GetValue 5301
TAdapterHiddenField Members 5299
TAdapterHiddenField methods 5301
TAdapterHiddenFieldsList class 5301
about TAdapterHiddenFieldsList class 5301
Add 5305
Clear 5305
Create 5304
Destroy 5304
EndIterator 5305
FieldCount 5308
Fields 5308
GetCurrentIntf 5306
GetField 5306
GetFieldCount 5306
NextIteration 5307
StartIterator 5307
TAdapterHiddenFieldsList Members 5302
TAdapterHiddenFieldsList methods 5304
TAdapterHiddenFieldsList properties 5307
TAdapterImageField class 5308
about TAdapterImageField class 5308
DisplayLabel 5314
DisplayWidth 5314
FieldName 5315
HREF 5315
ModifyAccess 5315
OnGetHREF 5317
OnGetImage 5318
OnGetImageName 5318
OnGetParams 5318
OnUploadFile 5319
OnValidateFile 5319
ReadOnly 5316
Required 5316
TAdapterImageField events 5317
TAdapterImageField Members 5310
TAdapterImageField properties 5314
ValuesList 5316
ViewAccess 5317
TAdapterItemRequestParams class 6160
about TAdapterItemRequestParams class 6160
Create 6162
Destroy 6162
FailurePage 6165
GetFailurePage 6163
GetParamValues 6163
GetSuccessPage 6164
ParamValues 6165
SetFailurePage 6164
SetSuccessPage 6164
SuccessPage 6165
TAdapterItemRequestParams Members 6161
TAdapterItemRequestParams methods 6163
TAdapterItemRequestParams properties 6165
TAdapterLoginField class 5320
about TAdapterLoginField class 5320
Adapter 5323
TAdapterLoginField Members 5321
TAdapterLoginField properties 5323
TAdapterMemoField class 5323
about TAdapterMemoField class 5323
DisplayLabel 5328
DisplayWidth 5329
FieldName 5329
ModifyAccess 5329
OnGetValue 5331
OnUpdateValue 5331
OnUploadFile 5332
ReadOnly 5330
Required 5330
TAdapterMemoField events 5331
TAdapterMemoField Members 5325
TAdapterMemoField properties 5328
ViewAccess 5330
TAdapterMultiValueField class 5332
about TAdapterMultiValueField class 5332
DisplayLabel 5338

DisplayWidth 5338
FieldName 5339
ModifyAccess 5339
OnGetValueCount 5341
OnGetValues 5341
OnHasValue 5342
OnUpdateValues 5342
OnValidateValues 5342
ReadOnly 5339
Required 5340
TAdapterMultiValueField events 5341
TAdapterMultiValueField Members 5334
TAdapterMultiValueField properties 5338
ValuesList 5340
ViewAccess 5340
TAdapterNamedDisplayField class 5343
about TAdapterNamedDisplayField class 5343
Create 5346
DisplayLabel 5350
DisplayWidth 5350
GetDisplayStyle 5346
GetDisplayStyleType 5347
GetInputStyle 5347
GetInputStyleType 5347
GetViewMode 5348
GetViewModeType 5348
ImplGetDisplayLabel 5348
ImplGetDisplayWidth 5349
ImplGetRequired 5349
ReadOnly 5351
Required 5351
SetDisplayLabel 5349
TAdapterNamedDisplayField Members 5344
TAdapterNamedDisplayField methods 5346
TAdapterNamedDisplayField properties 5350
TAdapterNamedField class 5351
about TAdapterNamedField class 5351
ActionOrigValue 5367
ActionValue 5368
Adapter 5368
AsFormatted 5355
CheckModifyAccess 5355
CheckViewAccess 5356
EchoActionFieldValue 5368
FieldName 5369
FullyQualifyInputName 5356
GetAdapter 5356
GetDefaultDisplayLabel 5357
GetDefaultFieldName 5357
GetDisplayLabel 5357
GetDisplayWidth 5358
GetEchoActionFieldValue 5358
GetInputName 5358
GetMaxLength 5359
GetRequired 5359
GetValue 5359
GetVariableName 5360
GetVisible 5360
GetWebSnapObjClass 5360
HasModifyAccess 5361
HasViewAccess 5361
ImplAsFormatted 5361
ImplGetDisplayLabel 5361
ImplGetDisplayWidth 5362
ImplGetEchoActionFieldValue 5362
ImplGetInputName 5362
ImplGetMaxLength 5363
ImplGetRequired 5363
ImplGetValue 5363
ImplGetVariableName 5364
ImplGetVisible 5364
ImplHasModifyAccess 5365
ImplHasViewAccess 5365
ImplSetEchoActionFieldValue 5365
ImplSetFieldName 5366
ModifyAccess 5369
ParentChanged 5366
SetEchoActionFieldValue 5366

SetFieldName 5367
TAdapterNamedField Members 5352
TAdapterNamedField methods 5354
TAdapterNamedField properties 5367
Value 5369
ViewAccess 5370
TAdapterNextPageAction class 5370
about TAdapterNextPageAction class 5370
DisplayLabel 5376
ExecuteAccess 5376
GetDefaultActionName 5374
GetDefaultDisplayLabel 5375
GetNewPage 5375
OnAfterExecute 5377
OnAfterGetResponse 5378
OnBeforeExecute 5378
OnBeforeGetResponse 5378
OnGetParams 5379
RedirectOptions 5377
TAdapterNextPageAction events 5377
TAdapterNextPageAction Members 5371
TAdapterNextPageAction methods 5374
TAdapterNextPageAction properties 5376
TAdapterNextPageField class 5379
about TAdapterNextPageField class 5379
Destroy 5383
DisplayLabel 5386
DisplayWidth 5386
FieldName 5387
GetAdapterValuesList 5384
GetInputStyleType 5384
ImplGetEchoActionFieldValue 5384
ImplGetValue 5385
ImplSetEchoActionFieldValue 5385
TAdapterNextPageField Members 5381
TAdapterNextPageField methods 5383
TAdapterNextPageField properties 5386
UpdateValue 5386
TAdapterObjectErrorsList class 5387
about TAdapterObjectErrorsList class 5387
Create 5389
ImplDefineLabel 5390
ImplGetError 5390
ImplGetErrorCount 5390
TAdapterObjectErrorsList Members 5388
TAdapterObjectErrorsList methods 5390
TAdapterPageProducer class 1136
about TAdapterPageProducer class 1136
HTMLDoc 1139
HTMLFile 1140
OnAfterGetContent 1142
OnBeforeGetContent 1142
Styles 1140
StylesFile 1141
TAdapterPageProducer events 1142
TAdapterPageProducer Members 1137
TAdapterPageProducer properties 1138
WebPageItems 1141
TAdapterPasswordField class 5391
about TAdapterPasswordField class 5391
DisplayLabel 5395
DisplayWidth 5395
FieldName 5396
GetInputStyleType 5394
ImplGetValue 5395
TAdapterPasswordField Members 5392
TAdapterPasswordField methods 5394
TAdapterPasswordField properties 5395
TAdapterPrevPageAction class 5396
about TAdapterPrevPageAction class 5396
DisplayLabel 5402
ExecuteAccess 5403
GetDefaultActionName 5401
GetDefaultDisplayLabel 5401
GetNewPage 5402
OnAfterExecute 5404
OnAfterGetResponse 5404
OnBeforeExecute 5404

OnBeforeGetResponse 5405
OnGetParams 5405
RedirectOptions 5403
TAdapterPrevPageAction events 5403
TAdapterPrevPageAction Members 5397
TAdapterPrevPageAction methods 5401
TAdapterPrevPageAction properties 5402
TAdapterSubComponent class 5406
about TAdapterSubComponent class 5406
Adapter 5418
CanAddClass 5408
EndIterator 5409
FindNamedVariable 5409
FindVariable 5410
FindVariableContainer 5410
GetCurrentObject 5411
GetDefaultWebComponents 5411
GetParentComponent 5412
GetVariable 5412
GetVariableCount 5413
GetVariablesInUse 5413
GetWebComponentNodeName 5413
HasParent 5414
ImplCanAddClass 5414
ImplGetWebComponentNodeName 5415
NextIteration 5415
NotifyAdapterChange 5416
ReadState 5416
SetDefaultWebComponents 5416
SetParentComponent 5417
SetWebParent 5417
StartIterator 5418
TAdapterSubComponent Members 5407
TAdapterSubComponent methods 5408
TAdapterSubComponent properties 5418
VariablesInUse 5419
TAdapterUpdateField class 5419
about TAdapterUpdateField class 5419
CheckValueChange 5424
ConvertValue 5425
ConvertValueCount 5425
ConvertValues 5425
Create 5423
DoOnValidateValue 5426
GetAdapterValuesList 5426
GetInputStyleType 5426
GetValueCount 5427
GetValues 5427
ImplAsFormatted 5427
ImplCheckValueChange 5428
ImplGetDisplayLabel 5428
ImplGetDisplayWidth 5428
ImplGetEchoActionFieldValue 5429
ImplGetVariableName 5429
ImplSetEchoActionFieldValue 5429
ImplSetFieldName 5429
ImplUpdateValue 5430
ImplValidateValue 5430
IsMultiValueField 5430
Notification 5431
OnGetDisplayText 5433
OnValidateValue 5433
TAdapterUpdateField events 5433
TAdapterUpdateField Members 5421
TAdapterUpdateField methods 5424
TAdapterUpdateField properties 5432
UpdateValue 5431
ValidateValue 5431
Value 5432
ValuesList 5432
TAdapterUserNameField class 5433
about TAdapterUserNameField class 5433
DisplayLabel 5438
DisplayWidth 5438
FieldName 5438
ImplGetValue 5437
TAdapterUserNameField Members 5434
TAdapterUserNameField methods 5437

TAdapterUserNameField properties 5437
TAggregatedObject class 3313
_AddRef 3316
_Release 3316
about TAggregatedObject class 3313
Controller 3317
Create 3315
QueryInterface 3317
TAggregatedObject Members 3314
TAggregatedObject methods 3315
TAggregatedObject properties 3317
TagID
TService class 3084
tagName
IDOMElement interface 7548
TagToken
THTMLItem class 6665
TAllocatedActions class 297
about TAllocatedActions class 297
AddAction 299
Clear 299
Count 300
Create 298
Delete 299
Destroy 298
Exists 299
Find 300
GetItem 300
TAllocatedActions Members 298
TAllocatedActions methods 299
TAllocatedActions properties 300
TApplicationAdapter class 5439
about TApplicationAdapter class 5439
Actions 5444
ApplicationTitle 5444
Data 5444
OnAfterExecuteAction 5445
OnAfterGetActionResponse 5445
OnBeforeExecuteAction 5446
OnBeforeGetActionResponse 5446
OnGetActionParams 5446
TApplicationAdapter events 5445
TApplicationAdapter Members 5440
TApplicationAdapter properties 5444
target
IDOMProcessingInstruction interface 7602
TargetNamespace
IDefinition interface 7100
TWSDLHTMLPublish class 7288
TWSDLItems class 7246
TWSDLPublish class 7293
TargetNameSpace
TWebServExp class 6732
TAssociateItem class 301
about TAssociateItem class 301
Associate 302
Create 301
Destroy 301
PopupTip 302
TAssociateItem Members 301
TAssociateItem properties 302
TAssociateItemList class 302
about TAssociateItemList class 302
Add 304
Create 303
Destroy 303
Find 304
Get 304
Remove 304
TAssociateItemList Members 303
TAssociateItemList methods 303
TBaseAdapterAction class 5447
about TBaseAdapterAction class 5447
ActionName 5474
Adapter 5474
CheckExecuteAccess 5452
CreateRequestContext 5452
DisplayLabel 5474

DoAfterExecuteActionRequest 5453
DoAfterGetExecuteActionResponse 5453
DoBeforeExecuteActionRequest 5453
DoBeforeGetExecuteActionResponse 5454
EndIterator 5454
Errors 5475
ExecuteAccess 5475
ExecuteActionRequest 5455
GetActionName 5455
GetAdapter 5456
GetAdapterItemRequestParams 5456
GetDefaultActionName 5456
GetDefaultDisplayLabel 5457
GetDisplayLabel 5457
GetDisplayStyle 5457
GetDisplayStyleType 5458
GetErrors 5458
GetExecuteActionResponse 5458
GetFailureOptions 5459
GetInputStyle 5459
GetIsAdapterActionList 5460
GetItemCount 5460
GetIteratorIndex 5460
GetSuccessOptions 5461
GetVariableName 5461
GetViewMode 5462
GetViewModeType 5462
GetVisible 5462
GetWebSnapObjClass 5463
HandleRequest 5463
HasExecuteAccess 5463
ImplCreateRequestContext 5464
ImplEndIterator 5464
ImplExecuteActionRequest 5464
ImplGetActionName 5465
ImplGetAdapterItemRequestParams 5465
ImplGetDisplayLabel 5465
ImplGetExecuteActionResponse 5466
ImplGetFailureOptions 5466
ImplGetIsAdapterActionList 5466
ImplGetIteratorIndex 5467
ImplGetSuccessOptions 5467
ImplGetVariableName 5467
ImplGetVisible 5468
ImplHandleRequest 5468
ImplHasExecuteAccess 5468
ImplInIterator 5469
ImplIsDefaultAction 5469
ImplNextIteration 5470
ImplStartIterator 5470
ImplWebEnabled 5470
InIterator 5471
IsDefaultAction 5471
ItemCount 5475
ItemIndex 5476
NextIteration 5472
OnAfterExecute 5477
OnAfterGetResponse 5477
OnBeforeExecute 5478
OnBeforeGetResponse 5478
OnGetParams 5478
ParentChanged 5472
RedirectOptions 5476
SetDisplayLabel 5472
StartIterator 5473
TBaseAdapterAction events 5477
TBaseAdapterAction Members 5448
TBaseAdapterAction methods 5450
TBaseAdapterAction properties 5473
WebEnabled 5473
TBaseAdapterErrorsList class 5479
about TBaseAdapterErrorsList class 5479
AddError 5482
AddError method 5481
ClearErrors 5482
DefineLabel 5483
EndIterator 5483
ErrorCount 5487

Errors 5487
GetCurrentIntf 5483
GetError 5484
GetErrorCount 5484
GetObjectErrors 5484
HasObjectErrors 5484
ImplClearErrors 5485
ImplDefineLabel 5485
ImplGetError 5485
ImplGetErrorCount 5485
ImplGetObjectErrors 5486
ImplHasObjectErrors 5486
NextIteration 5486
StartIterator 5486
TBaseAdapterErrorsList Members 5479
TBaseAdapterErrorsList methods 5481
TBaseAdapterErrorsList properties 5487
TBaseAdapterMultiValueField class 5488
about TBaseAdapterMultiValueField class 5488
Create 5492
Destroy 5492
EndValueIterator 5493
GetCurrentValue 5494
GetFieldValuesAdapter 5494
GetValueCount 5494
GetValues 5495
HasValue 5495
ImplGetCurrentValue 5495
ImplGetFieldValuesAdapter 5496
ImplHasValue 5496
NextValueIteration 5496
StartValueIterator 5497
TBaseAdapterMultiValueField Members 5489
TBaseAdapterMultiValueField methods 5493
TBaseAdapterPageProducer class 1143
about TBaseAdapterPageProducer class 1143
CanAddClass 1146
ContentFromStream 1147
Create 1145
CreatePageElements 1147
DeclareStyles 1148
DeclareWarnings 1148
Destroy 1145
DoAfterGetContent 1148
DoBeforeGetContent 1149
DoTagEvent 1149
GeneratePageElements 1150
GetContentOptions 1150
GetDefaultTemplate 1150
GetRequiredTags 1151
GetScriptEngines 1151
GetTemplateFileType 1152
GetXMLData 1152
GetXSLData 1152
HandleScriptTag 1153
HasHTMLBrowserView 1153
HasScriptView 1154
HasXMLBrowserView 1154
HasXSLBrowserView 1154
ImplCanAddClass 1155
OnAfterGetContent 1158
OnBeforeGetContent 1159
PageElements 1157
SetStyles 1155
SetStylesFile 1156
SetWebContentOptions 1156
Styles 1157
StylesFile 1158
TBaseAdapterPageProducer events 1158
TBaseAdapterPageProducer Members 1144
TBaseAdapterPageProducer methods 1145
TBaseAdapterPageProducer properties 1156
TBaseEndUserAdapter class 5497
about TBaseEndUserAdapter class 5497
CanExecuteLoginForm 5504
CanExecuteLogout 5504
CanProduceLoginPage 5505
DoOnHasRights 5505

ExecuteLogin 5505
ExecuteLoginForm 5506
ExecuteLogout 5506
GetDisplayName 5506
GetDisplayNameValue 5507
GetEndUserAdapter 5507
GetLoggedIn 5507
GetLoggedInValue 5508
GetLoginFormAction 5508
GetLogoutAction 5508
GetUserID 5509
HasExecuteRights 5509
HasModifyRights 5509
HasViewRights 5510
ImplCanAddActionClass 5510
ImplCanAddFieldClass 5510
ImplGetActionsList 5511
ImplGetDisplayName 5511
ImplGetEndUserAdapter 5511
ImplGetFieldsList 5512
ImplGetLoggedIn 5512
ImplGetLoginFormAction 5512
ImplGetLogoutAction 5513
ImplHasExecuteRights 5513
ImplHasModifyRights 5513
ImplHasViewRights 5514
ImplLogIn 5514
ImplProduceLoginPage 5514
LogIn 5515
LoginPage 5516
OnHasRights 5516
OnProduceLoginPage 5516
ProduceLoginPage 5515
TBaseEndUserAdapter events 5516
TBaseEndUserAdapter Members 5499
TBaseEndUserAdapter methods 5503
TBaseEndUserAdapter properties 5515
UserList 5515
TBaseRequestHandler class 6166
about TBaseRequestHandler class 6166
AppDispatcher 6168
HandleRequest 6167
ImplHandleRequest 6167
SetAppDispatcher 6167
TBaseRequestHandler Members 6166
TBaseRequestHandler methods 6167
TBaseRequestHandler properties 6168
TBaseSocket class 1370
about TBaseSocket class 1370
Active 1383
BlockMode 1385
BytesReceived 1386
BytesSent 1386
Close 1373
Create 1371
Destroy 1372
DoCreateHandle 1373
DoDestroyHandle 1373
DoHandleError 1373
Domain 1386
DoReceive 1374
DoSend 1374
ErrorCheck 1374
Handle 1387
Loaded 1374
MapDomain 1375
MapSockType 1375
OnCreateHandle 1388
OnDestroyHandle 1390
OnError 1393
OnReceive 1393
OnSend 1394
Open 1375
PeekBuf 1376
Protocol 1387
ReceiveBuf 1376
ReceiveIn 1379
Select 1379

SendBuf 1380
Sendln 1380
SendStream 1380
SetBytesReceived 1382
SetBytesSent 1382
SockType 1387
TBaseSocket events 1388
TBaseSocket Members 1370
TBaseSocket methods 1372
TBaseSocket properties 1383
WaitForData 1383
TBaseValuesListAdapter class 5517
about TBaseValuesListAdapter class 5517
GetListImage 5522
GetListImageOfValue 5523
GetListName 5523
GetListNameOfValue 5523
GetListValue 5524
GetNameFieldClass 5524
GetValueFieldClass 5524
GetWebSnapObjClass 5524
ImplCanAddFieldClass 5525
ImplGetFieldsList 5525
ImplGetListImage 5525
ImplGetListImageOfValue 5526
ImplGetListName 5526
ImplGetListNameOfValue 5526
ImplGetListValue 5527
TBaseValuesListAdapter Members 5518
TBaseValuesListAdapter methods 5522
TBaseValuesListField class 5527
about TBaseValuesListField class 5527
Adapter 5531
GetAdapter 5530
TBaseValuesListField Members 5528
TBaseValuesListField methods 5530
TBaseValuesListField properties 5530
TBaseWebModuleFactory class 6280
about TBaseWebModuleFactory class 6280
Create 6281
GetCacheMode 6282
GetComponentClass 6282
GetCreateMode 6283
GetIsAppModule 6283
GetModule 6283
GetModuleName 6284
PreventDestruction 6284
TBaseWebModuleFactory Members 6281
TBaseWebModuleFactory methods 6282
TBaseWebPageInfo class 6285
about TBaseWebPageInfo class 6285
Create 6286
GetIsLoginRequired 6287
GetIsPublished 6287
GetPageDescription 6287
GetPageFile 6288
GetPageHREF 6288
GetPageName 6288
GetPageTitle 6289
GetViewAccess 6289
SetFactory 6290
TBaseWebPageInfo Members 6285
TBaseWebPageInfo methods 6286
TBaseWebPageModuleFactory class 6290
about TBaseWebPageModuleFactory class 6290
Create 6291
GetCacheMode 6292
GetComponentClass 6292
GetCreateMode 6293
GetIsAppModule 6293
GetModule 6293
GetModuleName 6294
PreventDestruction 6294
TBaseWebPageModuleFactory Members 6291
TBaseWebPageModuleFactory methods 6292
tbButtonDontCare enumeration member 4595
tbButtonRoot enumeration member 4595
tbCheckBoxCheckedDisabled enumeration member 4595

tbCheckBoxCheckedHot enumeration member 4595
tbCheckBoxCheckedNormal enumeration member 4595
tbCheckBoxCheckedPressed enumeration member 4595
tbCheckBoxMixedDisabled enumeration member 4595
tbCheckBoxMixedHot enumeration member 4595
tbCheckBoxMixedNormal enumeration member 4595
tbCheckBoxMixedPressed enumeration member 4595
tbCheckBoxUncheckedDisabled enumeration member 4595
tbCheckBoxUncheckedHot enumeration member 4595
tbCheckBoxUncheckedNormal enumeration member 4595
tbCheckBoxUncheckedPressed enumeration member 4595
tbGroupBoxDisabled enumeration member 4595
tbGroupBoxNormal enumeration member 4595
TBigEndianUnicodeEncoding class 3967
about TBigEndianUnicodeEncoding class 3967
GetBytes 3968
GetChars 3969
GetPreamble 3969
TBigEndianUnicodeEncoding Members 3967
TBigEndianUnicodeEncoding methods 3968
TBinding class 6902
about TBinding class 6902
AfterConstruction 6908
GetBindingOperations 6908
GetName 6908
GetType_ 6909
SetName 6909
SetType_ 6909
TBinding Members 6903
TBinding methods 6907
TBindingOperation class 6909
about TBindingOperation class 6909
AfterConstruction 6915
GetFaults 6915
GetInput 6916
GetName 6916
GetOutput 6916
SetName 6916
TBindingOperation Members 6910
TBindingOperation methods 6915
TBindingOperations class 6917
about TBindingOperations class 6917
Add 6922
GetBindingOperation 6923
Insert 6923
TBindingOperations Members 6917
TBindingOperations methods 6922
TBindings class 6923
about TBindings class 6923
Add 6929
GetBinding 6929
Insert 6929
TBindings Members 6924
TBindings methods 6929
tbPushButtonDefaulted enumeration member 4595
tbPushButtonDisabled enumeration member 4595
tbPushButtonHot enumeration member 4595
tbPushButtonNormal enumeration member 4595
tbPushButtonPressed enumeration member 4595
tbRadioButtonCheckedDisabled enumeration member 4595
tbRadioButtonCheckedHot enumeration member 4595
tbRadioButtonCheckedNormal enumeration member 4595
tbRadioButtonCheckedPressed enumeration member 4595
tbRadioButtonUncheckedDisabled enumeration member 4595
tbRadioButtonUncheckedHot enumeration member 4595
tbRadioButtonUncheckedNormal enumeration member 4595
tbRadioButtonUncheckedPressed enumeration member 4595
TBrowseForFolder class 1764
about TBrowseForFolder class 1764
BeforeExecute 1772
BrowseOptions 1767
Caption 1767
DialogCaption 1768
DoCallback 1765
DoValidate 1766
Enabled 1768
ExecuteTarget 1766
Folder 1768

GetDialogClass 1766
HelpContext 1768
HelpKeyword 1769
HelpType 1769
Hint 1770
ImageIndex 1770
OnAccept 1772
OnCallback 1773
OnCancel 1773
OnHint 1774
OnValidate 1774
RootDir 1770
SecondaryShortCuts 1771
ShortCut 1771
TBrowseForFolder events 1772
TBrowseForFolder Members 1764
TBrowseForFolder methods 1765
TBrowseForFolder properties 1767
Visible 1771
tbUserButton enumeration member 4595
TButton class 2035
about TButton class 2035
Action 2039
Align 2040
Anchors 2044
BiDiMode 2044
Cancel 2045
Caption 2045
CommandLinkHint 2048
Constraints 2048
Default 2049
DisabledImageIndex 2050
DoubleBuffered 2050
DragCursor 2052
DragKind 2052
DragMode 2053
DropDownMenu 2054
ElevationRequired 2055
Enabled 2055
Font 2056
HotImageIndex 2057
ImageAlignment 2057
ImageIndex 2058
ImageMargins 2058
Images 2059
ModalResult 2059
OnClick 2073
OnContextPopup 2075
OnDragDrop 2076
OnDragOver 2079
OnDropDownClick 2082
OnEndDock 2082
OnEndDrag 2083
OnEnter 2084
OnExit 2086
OnKeyDown 2090
OnKeyPress 2094
OnKeyUp 2095
OnMouseActivate 2096
OnMouseDown 2097
OnMouseEnter 2100
OnMouseLeave 2100
OnMouseMove 2100
OnMouseUp 2103
OnStartDock 2105
OnStartDrag 2105
ParentBiDiMode 2060
ParentDoubleBuffered 2060
ParentFont 2061
ParentShowHint 2061
PopupMenu 2062
PressedImageIndex 2067
SelectedImageIndex 2067
ShowHint 2068
Style 2069
TabOrder 2069
TabStop 2070
TButton events 2072

TButton Members 2036
TButton properties 2038
Visible 2070
WordWrap 2072
TButtonActionLink class 2106
about TButtonActionLink class 2106
AssignClient 2108
FClient 2107
IsCheckedLinked 2108
SetChecked 2108
TButtonActionLink fields 2107
TButtonActionLink Members 2107
TButtonActionLink methods 2107
TButtonControl class 2109
about TButtonControl class 2109
ActionChange 2110
Checked 2113
ClicksDisabled 2115
Create 2110
CreateParams 2111
GetActionLinkClass 2111
GetChecked 2112
SetChecked 2112
TButtonControl Members 2109
TButtonControl methods 2110
TButtonControl properties 2112
WndProc 2112
WordWrap 2115
tcClockDontCare enumeration member 4598
tcClockRoot enumeration member 4598
tcComboBoxDontCare enumeration member 4598
tcComboBoxRoot enumeration member 4598
tcDropDownButtonDisabled enumeration member 4598
tcDropDownButtonHot enumeration member 4598
tcDropDownButtonNormal enumeration member 4598
tcDropDownButtonPressed enumeration member 4598
TCheckBox class 2116
about TCheckBox class 2116
Action 2119
Align 2120
Alignment 2123
AllowGrayed 2124
Anchors 2124
BiDiMode 2125
Caption 2126
Checked 2128
Color 2130
Constraints 2131
Ctl3D 2132
DoubleBuffered 2132
DragCursor 2134
DragKind 2134
DragMode 2135
Enabled 2136
Font 2137
OnClick 2151
OnContextPopup 2153
OnDragDrop 2154
OnDragOver 2157
OnEndDock 2160
OnEndDrag 2160
OnEnter 2161
OnExit 2164
OnKeyDown 2168
OnKeyPress 2171
OnKeyUp 2172
OnMouseActivate 2174
OnMouseDown 2175
OnMouseEnter 2177
OnMouseLeave 2178
OnMouseMove 2178
OnMouseUp 2180
OnStartDock 2182
OnStartDrag 2183
ParentBiDiMode 2138
ParentColor 2138
ParentCtl3D 2139
ParentDoubleBuffered 2139

ParentFont 2139
ParentShowHint 2140
PopupMenu 2140
ShowHint 2145
State 2147
TabOrder 2148
TabStop 2148
TCheckBox events 2151
TCheckBox Members 2116
TCheckBox properties 2118
Visible 2149
WordWrap 2150
TClassHelperBase class 3318
_Create 3319
about TClassHelperBase class 3318
FInstance 3320
TClassHelperBase fields 3319
TClassHelperBase Members 3318
TClientSocket class 24
about TClientSocket class 24
Active 28
Address 28
ClientType 29
Create 26
Destroy 26
DoActivate 27
GetClientType 27
Host 30
OnConnect 32
OnConnecting 32
OnDisconnect 33
OnError 33
OnLookup 34
OnRead 34
OnWrite 35
Port 30
Service 31
SetClientType 27
Socket 31
TClientSocket events 32
TClientSocket Members 25
TClientSocket methods 27
TClientSocket properties 28
TClientSocketThread class 1396
about TClientSocketThread class 1396
ClientSocket 1399
Create 1397
Destroy 1397
Execute 1398
ExecuteSyncProc 1398
ServerSocketThread 1399
SyncProc 1398
TClientSocketThread Members 1396
TClientSocketThread methods 1397
TClientSocketThread properties 1399
TClientWinSocket class 35
about TClientWinSocket class 35
ClientType 38
Connect 38
SetClientType 38
TClientWinSocket Members 36
TClientWinSocket methods 37
TClientWinSocket properties 38
TColorSelect class 1774
about TColorSelect class 1774
BeforeExecute 1781
Caption 1776
Dialog 1777
Enabled 1777
GetDialogClass 1776
HelpContext 1777
HelpKeyword 1778
HelpType 1778
Hint 1779
ImageIndex 1779
OnAccept 1781
OnCancel 1782
OnHint 1782

OnUpdate 1782
SecondaryShortCuts 1780
ShortCut 1780
TColorSelect events 1781
TColorSelect Members 1775
TColorSelect methods 1776
TColorSelect properties 1776
Visible 1780
TComboBox class 2184
about TComboBox class 2184
Align 2190
Anchors 2193
AutoCloseUp 2194
AutoComplete 2194
AutoCompleteDelay 2195
AutoDropDown 2195
BevelEdges 2195
BevelInner 2196
BevelKind 2196
BevelOuter 2197
BiDiMode 2197
CharCase 2197
Color 2198
Constraints 2199
Ctl3D 2199
DoubleBuffered 2200
DragCursor 2202
DragKind 2202
DragMode 2202
DropDownCount 2204
Enabled 2204
Font 2205
ImeMode 2206
ImeName 2206
ItemHeight 2207
ItemIndex 2208
Items 2210
MaxLength 2212
OnChange 2228
OnClick 2230
OnCloseUp 2232
OnContextPopup 2233
OnDblClick 2233
OnDragDrop 2234
OnDragOver 2237
OnDrawItem 2240
OnDropDown 2240
OnEndDock 2242
OnEndDrag 2242
OnEnter 2243
OnExit 2246
OnKeyDown 2250
OnKeyPress 2253
OnKeyUp 2254
OnMeasureItem 2256
OnMouseEnter 2257
OnMouseLeave 2257
OnSelect 2257
OnStartDock 2257
OnStartDrag 2258
ParentBiDiMode 2212
ParentColor 2213
ParentCtl3D 2213
ParentDoubleBuffered 2214
ParentFont 2214
ParentShowHint 2214
PopupMenu 2215
ShowHint 2220
Sorted 2221
Style 2223
TabOrder 2223
TabStop 2224
TComboBox events 2227
TComboBox Members 2185
TComboBox properties 2189
Text 2224
TextHint 2226
Visible 2226

TCommonDialogAction class 1784
about TCommonDialogAction class 1784
BeforeExecute 1790
Create 1785
Destroy 1786
DoAccept 1787
DoCancel 1787
ExecuteResult 1789
ExecuteTarget 1787
FDialog 1786
GetDialogClass 1788
HandlesTarget 1788
Notification 1789
OnAccept 1790
OnCancel 1791
SetupDialog 1789
TCommonDialogAction events 1790
TCommonDialogAction fields 1786
TCommonDialogAction Members 1785
TCommonDialogAction methods 1786
TCommonDialogAction properties 1789
TComponentFactory class 5166
about TComponentFactory class 5166
Create 5167
CreateComObject 5167
CreateInstance 5168
TComponentFactory Members 5166
TComponentFactory methods 5167
UpdateRegistry 5168
TConditionVariableCS class 3112
about TConditionVariableCS class 3112
Acquire 3115
Create 3114
FConditionVariable 3115
Release 3115
ReleaseAll 3116
TConditionVariableCS fields 3114
TConditionVariableCS Members 3113
TConditionVariableCS methods 3115
WaitFor 3116
WaitFor method 3116
TConditionVariableHelper class helper 3117
about TConditionVariableHelper class helper 3117
Create 3118
Free 3119
SleepCS 3119
TConditionVariableHelper Members 3118
TConditionVariableHelper methods 3118
Wake 3120
WakeAll 3120
TConditionVariableMutex class 3120
about TConditionVariableMutex class 3120
Acquire 3123
Create 3122
Destroy 3123
Release 3124
ReleaseAll 3124
TConditionVariableMutex Members 3121
TConditionVariableMutex methods 3123
WaitFor 3124
TConnect 4545
TConnect namespace 4545
Classes 4546
TConnect.TLocalConnection 4546
TConnect.TLocalConnection.AppServer 4553
TConnect.TLocalConnection.AS_ApplyUpdates 4549
TConnect.TLocalConnection.AS_DataRequest 4549
TConnect.TLocalConnection.AS_Execute 4549
TConnect.TLocalConnection.AS_GetParams 4549
TConnect.TLocalConnection.AS_GetProviderNames 4550
TConnect.TLocalConnection.AS_GetRecords 4550
TConnect.TLocalConnection.AS_RowRequest 4550
TConnect.TLocalConnection.Create 4547
TConnect.TLocalConnection.Destroy 4548
TConnect.TLocalConnection.GetConnected 4550
TConnect.TLocalConnection.GetProvider 4551
TConnect.TLocalConnection.GetProviderNames 4551
TConnect.TLocalConnection.GetServer 4551

TConnect.TLocalConnection.InterfaceSupportsErrorInfo 4552
TConnect.TLocalConnection.Notification 4552
TConnect.TLocalConnection.ProviderCount 4554
TConnect.TLocalConnection.Providers 4554
TConnect.TLocalConnection.RegisterProvider 4552
TConnect.TLocalConnection.SafeCallException 4553
TConnect.TLocalConnection.UnRegisterProvider 4553
TConnectionData class 1515
about TConnectionData class 1515
AddProperties 1517
BeginUpdate 1518
ConnectionName 1519
Create 1516
Create constructor 1516
DelegateConnection 1519
Destroy 1517
DriverName 1520
EndUpdate 1518
GetFullParams 1518
IsModified 1520
Properties 1520
RefreshProperties 1518
ReloadProperties 1518
TConnectionData Members 1515
TConnectionData methods 1517
TConnectionData properties 1519
UpdateProperties 1519
TContainedObject class 3320
about TContainedObject class 3320
QueryInterface 3322
TContainedObject Members 3321
TContainedObject methods 3322
TControlRequirements class 6425
about TControlRequirements class 6425
Create 6426
DefineFunction 6427
DefineVariable 6428
Destroy 6427
IsFunctionDefined 6428
IsVariableDefined 6429
NeedMultipartForm 6429
RequiresMultipartForm 6429
TControlRequirements Members 6426
TControlRequirements methods 6427
TControlRequirements properties 6429
TCriticalSection class 3125
about TCriticalSection class 3125
Acquire 3128
Create 3127
Destroy 3127
Enter 3128
FSection 3128
Leave 3129
Release 3129
TCriticalSection fields 3127
TCriticalSection Members 3126
TCriticalSection methods 3128
TryEnter 3130
TCriticalSectionHelper class helper 3130
about TCriticalSectionHelper class helper 3130
Destroy 3131
Enter 3132
Free 3132
Initialize 3132
Leave 3133
TCriticalSectionHelper Members 3131
TCriticalSectionHelper methods 3131
TryEnter 3133
tcTimeNormal enumeration member 4598
TCurrentWebPageInfo class 6657
about TCurrentWebPageInfo class 6657
Create 6659
GetHasViewAccess 6660
GetLoginRequired 6660
GetPageDescription 6660
GetPageHREF 6661
GetPageIsPublished 6661
GetPageName 6661

GetPageTitle 6662
TCurrentWebPageInfo Members 6658
TCurrentWebPageInfo methods 6660
WebPageInfo 6662
TCustomAdapter class 5531
about TCustomAdapter class 5531
Actions 5572
AddAdapterModeHiddenField 5538
AddOrigValueHiddenFields 5539
AddRecordKeysHiddenFields 5539
CanAddActionClass 5539
CanAddFieldClass 5540
CheckModifyAccess 5540
ClearAdapterValues 5540
ClearErrors 5541
Create 5536
CreateActionRequestContext 5541
CreateErrorsList 5542
Data 5572
Destroy 5536
EchoActionFieldValues 5572
EncodeActionParams 5542
EndIterator 5542
Errors 5573
ExecuteAccess 5573
ExtractRequestParams 5543
FindAction 5543
FindField 5543
FindNamedAction 5544
FindNamedField 5544
FindNamedVariable 5544
FindVariable 5545
FindVariableContainer 5545
GetActionsClass 5545
GetActionsList 5546
GetAdapterActions 5546
GetAdapterErrors 5546
GetAdapterErrorsList 5547
GetAdapterFields 5547
GetAdapterHiddenFields 5547
GetAdapterHiddenRecordFields 5548
GetChildren 5548
GetDataClass 5548
GetEchoActionFieldValues 5549
GetFieldsList 5549
GetHiddenFieldOptions 5549
GetIteratorIndex 5550
GetVariable 5550
GetVariableCount 5550
GetVariableName 5551
GetVisibleActions 5551
GetVisibleFields 5551
GetWebSnapObjClass 5552
HasExecuteAccess 5552
HasModifyAccess 5552
HasViewAccess 5553
ImplCanAddActionClass 5553
ImplCanAddFieldClass 5553
ImplClearAdapterValues 5554
ImplEndIterator 5554
ImplFindVariable 5554
ImplFindVariableContainer 5555
ImplGetActionsList 5555
ImplGetEchoActionFieldValues 5555
ImplGetFieldsList 5556
ImplGetIteratorIndex 5556
ImplGetVariable 5556
ImplGetVariableCount 5557
ImplGetVariableName 5557
ImplGetVisibleActions 5557
ImplGetVisibleFields 5558
ImplHasExecuteAccess 5558
ImplHasModifyAccess 5558
ImplHasViewAccess 5559
ImplInIterator 5559
ImplNextIteration 5560
ImplNotifyActivate 5560
ImplNotifyDeactivate 5560

ImplSetActions 5561
ImplSetData 5561
ImplSetEchoActionFieldValues 5561
ImplStartIterator 5562
InIterator 5562
IsActionInUse 5563
IsFieldInUse 5563
ListHiddenFields 5563
ListHiddenRecordFields 5564
ModifyAccess 5574
NextIteration 5564
Notification 5565
NotifyActivate 5565
NotifyAdapterChange 5565
NotifyDeactivate 5566
NotifyList 5574
OnAfterExecuteAction 5576
OnAfterGetActionResponse 5576
OnAfterUpdateFields 5576
OnAfterValidateFields 5577
OnBeforeExecuteAction 5577
OnBeforeGetActionResponse 5578
OnBeforeUpdateFields 5578
OnBeforeValidateFields 5578
OnGetActionParams 5579
OnIterateRecords 5579
SetActions 5566
SetData 5566
SetEchoActionFieldValues 5567
StartIterator 5567
TCustomAdapter events 5575
TCustomAdapter Members 5533
TCustomAdapter methods 5536
TCustomAdapter properties 5571
UpdateFieldsExecuteValues 5567
UpdateFieldsGetAnyChanges 5568
UpdateFieldsUpdateValues 5568
UpdateFieldsValidateValues 5569
UpdateRecords 5569
UpdateRecordsGetAnyChanges 5570
UpdateRecordsUpdateValues 5570
UpdateRecordsValidateValues 5571
ViewAccess 5574
VisibleActions 5575
VisibleFields 5575
WebComponentsChange 5571
TCustomAdapterAction class 5579
about TCustomAdapterAction class 5579
ActionName 5586
ImplExecuteActionRequest 5584
ImplGetActionName 5584
ImplGetAdapterItemRequestParams 5584
ImplGetSuccessOptions 5585
ImplSetActionName 5585
ImplWebEnabled 5585
OnAfterExecute 5587
OnAfterGetResponse 5587
OnBeforeExecute 5588
OnBeforeGetResponse 5588
OnExecute 5588
OnGetEnabled 5589
SetActionName 5586
TCustomAdapterAction events 5587
TCustomAdapterAction Members 5580
TCustomAdapterAction methods 5583
TCustomAdapterAction properties 5586
TCustomAdapterActionButton class 6430
about TCustomAdapterActionButton class 6430
ActionName 6438
Align 6438
Create 6431
Destroy 6432
DisplayColumns 6438
DisplayType 6439
ErrorPageName 6439
FindAssociatedAction 6433
GetActionName 6433
GetAdapterModeOfAdapter 6433

GetAdapterVariableName 6433
GetCaption 6434
GetControlRequirements 6434
GetDesigntimeWarnings 6434
GetHTMLFormVarName 6434
GetLayoutAttributes 6435
HideOptions 6440
ImageHeight 6440
ImageProducer 6441
ImageWidth 6441
ImplContent 6435
ImplGetControlRequirements 6435
ImplRestoreDefaults 6436
Notification 6436
PageName 6441
RestoreDefaults 6436
SetActionName 6437
SetCaption 6437
TCustomAdapterActionButton Members 6431
TCustomAdapterActionButton methods 6432
TCustomAdapterActionButton properties 6437
VAlign 6442
TCustomAdapterBooleanField class 5589
about TCustomAdapterBooleanField class 5589
ConvertValue 5594
GetDisplayStyleType 5594
GetInputStyleType 5595
ImplGetValue 5595
ImplIsEqual 5596
ImplUpdateValue 5596
IsEqual 5596
OnGetValue 5597
OnUpdateValue 5597
TCustomAdapterBooleanField events 5597
TCustomAdapterBooleanField Members 5591
TCustomAdapterBooleanField methods 5594
TCustomAdapterCommandColumn class 6442
about TCustomAdapterCommandColumn class 6442
AdapterActionsListHelper 6449
AddDefaultCommands 6449
Align 6450
Caption 6450
CaptionAttributes 6451
Create 6444
Custom 6451
Destroy 6445
DisplayColumns 6451
FormatColumnData 6445
FormatColumnHeading 6446
GetAdapterDisplayCharacteristics 6446
GetLayoutAttributes 6446
ImplCanAddClass 6447
ImplContent 6447
NotifyAdapterChange 6448
Style 6452
StyleRule 6452
TCustomAdapterCommandColumn Members 6443
TCustomAdapterCommandColumn methods 6445
TCustomAdapterCommandColumn properties 6449
VAlign 6453
VisibleCommands 6453
TCustomAdapterCommandGroup class 6454
about TCustomAdapterCommandGroup class 6454
Adapter 6459
AdapterActionsListHelper 6459
AddDefaultCommands 6459
Create 6456
Custom 6459
Destroy 6456
GetAdapter 6457
GetAdapterFormLayout 6457
GetAdapterModeOfAdapter 6457
GetAdapterModeOfDisplay 6457
ImplCanAddClass 6458
ImplContent 6458
Style 6460
StyleRule 6460
TCustomAdapterCommandGroup Members 6454

TCustomAdapterCommandGroup methods 6456
TCustomAdapterCommandGroup properties 6458
VisibleCommands 6460
TCustomAdapterDispatcher class 6168
about TCustomAdapterDispatcher class 6168
AddSpecialParams 6170
DoAdapterRequestNotHandled 6171
DoAfterDispatchAdapterRequest 6172
DoBeforeDispatchAdapterRequest 6172
EncodeAdapterItemRequestAsFieldValue 6173
EncodeAdapterItemRequestAsHREF 6173
GetAdapterItemRequestParams 6174
GetDispatchParams 6174
ImplEncodeAdapterItemRequestAsFieldValue 6175
ImplEncodeAdapterItemRequestAsHREF 6175
ImplGetAdapterItemRequestParams 6176
ImplHandleRequest 6176
OnActionRequestNotHandled 6177
OnAfterDispatchAction 6178
OnAfterDispatchImage 6178
OnBeforeDispatchAction 6179
OnBeforeDispatchImage 6179
OnGetAdapterRequestParams 6180
OnImageRequestNotHandled 6180
TCustomAdapterDispatcher events 6177
TCustomAdapterDispatcher Members 6169
TCustomAdapterDispatcher methods 6170
TCustomAdapterDisplayColumn class 6461
about TCustomAdapterDisplayColumn class 6461
CanAddClass 6463
ControlContent 6463
Create 6463
DisplayType 6466
FormatImage 6464
FormatListDisplay 6464
FormatTextDisplay 6464
ImageHeight 6466
ImageWidth 6467
ImplCanAddClass 6465
ImplFormatColumnHeading 6465
ImplGetControlRequirements 6465
ImplRestoreDefaults 6465
IsColumn 6466
TCustomAdapterDisplayColumn Members 6461
TCustomAdapterDisplayColumn methods 6463
TCustomAdapterDisplayColumn properties 6466
TCustomAdapterDisplayField class 6467
about TCustomAdapterDisplayField class 6467
AddAttributes 6470
ControlContent 6470
Create 6469
DisplayColumns 6472
DisplayRows 6472
DisplayType 6473
DisplayWidth 6473
GetLayoutAttributes 6470
ImageHeight 6473
ImageWidth 6474
ImplContent 6471
ImplGetControlRequirements 6471
ImplRestoreDefaults 6471
InputType 6474
MaxLength 6474
SelectRows 6475
TCustomAdapterDisplayField Members 6468
TCustomAdapterDisplayField methods 6469
TCustomAdapterDisplayField properties 6472
TextAreaWrap 6475
ViewMode 6475
TCustomAdapterEditColumn class 6475
about TCustomAdapterEditColumn class 6475
ImplContent 6478
TCustomAdapterEditColumn Members 6476
TCustomAdapterEditColumn methods 6477
TCustomAdapterErrorList class 6478
about TCustomAdapterErrorList class 6478
Adapter 6482
Content 6480

Create 6480
Destroy 6480
GetDesigntimeWarnings 6481
ImplContent 6481
Notification 6482
TCustomAdapterErrorList Members 6479
TCustomAdapterErrorList methods 6480
TCustomAdapterErrorList properties 6482
TCustomAdapterField class 5597
about TCustomAdapterField class 5597
DoOnValidateValue 5602
ImplGetValue 5602
ImplUpdateValue 5603
OnGetValue 5603
OnUpdateValue 5604
TCustomAdapterField events 5603
TCustomAdapterField Members 5598
TCustomAdapterField methods 5602
TCustomAdapterFieldGroup class 6483
about TCustomAdapterFieldGroup class 6483
Create 6485
Custom 6489
Destroy 6485
GetAdapterDisplayCharacteristics 6486
GetAdapterFormLayout 6486
GetAddAllFieldsItem 6486
GetAddFieldsDlgCaption 6487
GetAddFieldsItem 6487
GetLayoutAttributes 6487
GetNewFieldPrefix 6487
ImplCanAddClass 6488
ImplContent 6488
Style 6489
StyleRule 6489
TCustomAdapterFieldGroup Members 6484
TCustomAdapterFieldGroup methods 6486
TCustomAdapterFieldGroup properties 6488
TCustomAdapterFileField class 5604
about TCustomAdapterFileField class 5604
DoOnValidateValue 5609
FileCount 5612
FileNames 5612
GetDisplayStyleType 5609
GetFileNames 5610
GetInputStyleType 5610
GetValueCount 5610
GetValues 5611
ImplCheckValueChange 5611
ImplUpdateValue 5611
OnGetFileCount 5613
OnGetFileNames 5613
OnUploadFiles 5613
OnValidateFiles 5614
TCustomAdapterFileField events 5613
TCustomAdapterFileField Members 5605
TCustomAdapterFileField methods 5608
TCustomAdapterFileField properties 5612
TCustomAdapterForm class 6490
about TCustomAdapterForm class 6490
Create 6491
Custom 6495
Destroy 6491
FormMethod 6495
GetAdapterFormLayout 6492
GetFormRequirements 6492
GetHTMLFormName 6492
GetHTMLFormVarName 6493
GetLayoutAttributes 6493
ImplCanAddClass 6493
ImplContent 6494
Style 6495
StyleRule 6496
TCustomAdapterForm Members 6491
TCustomAdapterForm methods 6492
TCustomAdapterForm properties 6494
TCustomAdapterGotoPageAction class 5614
about TCustomAdapterGotoPageAction class 5614
GetDefaultActionName 5619

GetDisplayStyleType 5619
GetItemCount 5620
ImplExecuteActionRequest 5620
ImplGetAdapterItemRequestParams 5620
ImplGetDisplayLabel 5621
ImplGetIsAdapterActionList 5621
ImplIsDefaultAction 5622
ImplWebEnabled 5622
OnGetPageName 5622
TCustomAdapterGotoPageAction events 5622
TCustomAdapterGotoPageAction Members 5615
TCustomAdapterGotoPageAction methods 5619
TCustomAdapterGrid class 6496
about TCustomAdapterGrid class 6496
Create 6499
Destroy 6499
GetAdapterDisplayCharacteristics 6500
GetAdapterFormLayout 6500
GetAddAllFieldsItem 6500
GetAddFieldsDlgCaption 6501
GetAddFieldsItem 6501
GetNewFieldPrefix 6501
HeadingAttributes 6502
ImplCanAddClass 6501
ImplContent 6502
RowAttributes 6503
TableAttributes 6503
TCustomAdapterGrid Members 6498
TCustomAdapterGrid methods 6500
TCustomAdapterGrid properties 6502
TCustomAdapterImageField class 5623
about TCustomAdapterImageField class 5623
CreateRequestContext 5628
DoOnValidateValue 5628
GetAdapterImage 5629
GetAdapterItemRequestParams 5629
GetDisplayStyleType 5630
GetInputStyleType 5630
HandleRequest 5630
HREF 5634
ImplCheckValueChange 5631
ImplCreateRequestContext 5631
ImplGetValue 5632
ImplHandleRequest 5632
ImplUpdateValue 5632
ImplWebImageHREF 5633
OnGetHREF 5635
OnGetImage 5636
OnGetImageName 5636
OnGetParams 5636
OnUploadFile 5637
OnValidateFile 5637
ReadOnly 5635
RenderAdapterImage 5633
TCustomAdapterImageField events 5635
TCustomAdapterImageField Members 5624
TCustomAdapterImageField methods 5627
TCustomAdapterImageField properties 5634
WebImageHREF 5634
TCustomAdapterMemoField class 5638
about TCustomAdapterMemoField class 5638
DoOnValidateValue 5642
GetDisplayStyleType 5642
GetInputStyleType 5643
ImplCheckValueChange 5643
ImplGetValue 5644
ImplUpdateValue 5644
OnGetValue 5645
OnUpdateValue 5645
OnUploadFile 5645
TCustomAdapterMemoField events 5644
TCustomAdapterMemoField Members 5639
TCustomAdapterMemoField methods 5642
TCustomAdapterMultiValueField class 5646
about TCustomAdapterMultiValueField class 5646
DoOnValidateValue 5651
GetDisplayStyleType 5651
GetInputStyleType 5651

GetValueCount 5652
GetValues 5652
ImplHasValue 5653
ImplUpdateValue 5653
OnGetValueCount 5654
OnGetValues 5655
OnHasValue 5655
OnUpdateValues 5656
OnValidateValues 5656
TCustomAdapterMultiValueField events 5654
TCustomAdapterMultiValueField Members 5647
TCustomAdapterMultiValueField methods 5650
TCustomAdapterMultiValueField properties 5653
ValueCount 5653
Values 5654
TCustomAdapterPageAction class 5656
about TCustomAdapterPageAction class 5656
GetDisplayStyleType 5661
GetNewPage 5661
ImplExecuteActionRequest 5662
ImplGetAdapterItemRequestParams 5662
ImplIsDefaultAction 5663
ImplWebEnabled 5663
TCustomAdapterPageAction Members 5657
TCustomAdapterPageAction methods 5661
TCustomAdapterPageProducer class 1159
about TCustomAdapterPageProducer class 1159
Create 1161
Destroy 1162
GetTemplateFileName 1162
GetTemplateStream 1163
HTMLDoc 1165
HTMLFile 1166
SetHTMLDoc 1163
SetHTMLFile 1164
TCustomAdapterPageProducer Members 1160
TCustomAdapterPageProducer methods 1162
TCustomAdapterPageProducer properties 1165
WebContent 1164
TCustomApplicationAdapter class 5663
about TCustomApplicationAdapter class 5663
ApplicationTitle 5671
GetApplicationAdapter 5669
GetApplicationTitle 5669
GetWebSnapObjClass 5669
ImplCanAddActionClass 5669
ImplCanAddFieldClass 5670
ImplGetActionsList 5670
ImplGetFieldsList 5670
TCustomApplicationAdapter Members 5664
TCustomApplicationAdapter methods 5668
TCustomApplicationAdapter properties 5671
TCustomApplicationAdapterAction class 5671
about TCustomApplicationAdapterAction class 5671
Adapter 5676
TCustomApplicationAdapterAction Members 5672
TCustomApplicationAdapterAction properties 5676
TCustomButton class 2259
about TCustomButton class 2259
ActionChange 2262
Cancel 2267
Click 2263
CommandLinkHint 2267
Create 2260
CreateParams 2263
CreateWnd 2264
Default 2268
Destroy 2261
DisabledImageIndex 2269
DropDownMenu 2269
ElevationRequired 2270
GetActionLinkClass 2264
HotImageIndex 2270
ImageAlignment 2271
ImageIndex 2271
ImageMargins 2272
Images 2272
Loaded 2264

ModalResult 2273
Notification 2265
OnDropDownClick 2276
PressedImageIndex 2274
SelectedImageIndex 2274
SetButtonStyle 2265
Style 2275
TabStop 2275
TCustomButton enumerations 2261
TCustomButton events 2275
TCustomButton Members 2259
TCustomButton methods 2262
TCustomButton properties 2267
UpdateImageList 2266
UpdateImages 2266
UseRightToLeftAlignment 2266
TCustomCheckBox class 2276
about TCustomCheckBox class 2276
Alignment 2281
AllowGrayed 2281
Click 2278
Create 2277
CreateParams 2278
CreateWnd 2279
GetChecked 2279
GetControlsAlignment 2279
SetChecked 2280
State 2282
TabStop 2282
TCustomCheckBox Members 2276
TCustomCheckBox methods 2277
TCustomCheckBox properties 2280
Toggle 2280
TCustomCombo class 2283
about TCustomCombo class 2283
AddItem 2291
AdjustDropDown 2292
Canvas 2305
Change 2292
Clear 2293
ClearSelection 2293
CloseUp 2293
ComboWndProc 2294
CopySelection 2294
Create 2285
CreateWnd 2295
DeleteSelected 2295
Destroy 2286
DestroyWindowHandle 2296
DropDown 2296
DropDownCount 2306
DroppedDown 2306
EditHandle 2307
EditWndProc 2296
FDefEditProc 2286
FDefListProc 2287
FDropHandle 2287
FDroppingDown 2287
FEditHandle 2288
FEditInstance 2288
FFocusChanged 2288
FIsFocused 2288
FListHandle 2289
FListInstance 2289
Focused 2297
FSaveIndex 2289
GetCount 2297
GetDroppedDown 2297
GetItemCount 2298
GetItemHt 2298
GetItemIndex 2298
GetItemsClass 2299
GetSelLength 2299
GetSelStart 2299
ItemCount 2307
ItemHeight 2307
Items 2308
ListHandle 2310

ListWndProc 2300
Loaded 2300
MaxLength 2310
OnChange 2318
OnCloseUp 2320
OnDropDown 2320
OnSelect 2322
ParentColor 2311
Select 2301
SelectAll 2301
SelLength 2311
SelStart 2314
SetDropDownCount 2301
SetDroppedDown 2302
SetItemHeight 2302
SetItemIndex 2302
SetItems 2303
SetMaxLength 2303
SetSelLength 2303
SetSelStart 2304
TabStop 2317
TCustomCombo events 2318
TCustomCombo fields 2286
TCustomCombo Members 2284
TCustomCombo methods 2290
TCustomCombo properties 2304
WndProc 2304
TCustomComboBox class 2322
about TCustomComboBox class 2322
AutoCloseUp 2332
AutoComplete 2332
AutoCompleteDelay 2333
AutoDropDown 2334
CharCase 2334
Create 2325
CreateParams 2326
CreateWnd 2327
Destroy 2326
DestroyWnd 2327
DoSetTextHint 2328
DrawItem 2328
GetItemCount 2328
GetItemHt 2329
GetItemsClass 2329
GetSelText 2329
KeyPress 2329
MeasureItem 2330
OnDrawItem 2338
OnMeasureItem 2338
SelectItem 2330
SelText 2335
SetStyle 2331
Sorted 2335
Style 2337
TCustomComboBox events 2337
TCustomComboBox Members 2323
TCustomComboBox methods 2326
TCustomComboBox properties 2331
TextHint 2337
WndProc 2331
TCustomComboBoxStrings class 2339
about TCustomComboBoxStrings class 2339
Clear 2341
ComboBox 2344
Create 2340
Delete 2341
Destroy 2340
Get 2342
GetCount 2342
GetObject 2342
IndexOf 2343
PutObject 2343
SetUpdateState 2344
TCustomComboBoxStrings Members 2339
TCustomComboBoxStrings methods 2340
TCustomComboBoxStrings properties 2344
TCustomDataBlockInterpreter class 192
about TCustomDataBlockInterpreter class 192

AddDispatch 194
CallCreateObject 194
CallFreeObject 194
CallGetIDsOfNames 194
CallGetServerList 195
CallInvoke 195
CanCreateObject 195
CreateObject 195
InternalCreateObject 195
InterpretData 196
LockObject 196
ReleaseObject 196
RemoveDispatch 196
StoreObject 197
TCustomDataBlockInterpreter Members 193
TCustomDataBlockInterpreter methods 193
UnlockObject 197
TCustomEdit class 2345
about TCustomEdit class 2345
Alignment 2365
AutoSelect 2365
AutoSize 2366
BorderStyle 2367
CanUndo 2367
Change 2348
CharCase 2367
Clear 2348
ClearSelection 2349
ClearUndo 2349
CopyToClipboard 2349
Create 2347
CreateParams 2351
CreateWindowHandle 2351
CreateWnd 2352
CutToClipboard 2352
DefaultHandler 2354
DestroyWnd 2354
DoSetMaxLength 2354
DoSetTextHint 2355
GetControlsAlignment 2355
GetSelLength 2356
GetSelStart 2356
GetSelText 2356
GetSelTextBuf 2357
HideSelection 2368
MaxLength 2368
Modified 2369
NumbersOnly 2369
OEMConvert 2369
OnChange 2381
ParentColor 2370
PasswordChar 2370
PasteFromClipboard 2358
ReadOnly 2371
SelectAll 2359
SelLength 2371
SelStart 2374
SelText 2377
SetAlignment 2361
SetAutoSize 2361
SetSelLength 2361
SetSelStart 2362
SetSelText 2362
SetSelTextBuf 2363
TabStop 2379
TCustomEdit events 2380
TCustomEdit Members 2345
TCustomEdit methods 2347
TCustomEdit properties 2364
Text 2379
TextHint 2380
Undo 2364
TCustomEndUserAdapter class 5676
about TCustomEndUserAdapter class 5676
ExecuteLogin 5683
ExecuteLogout 5683
GetDisplayNameValue 5683
GetLoggedInValue 5684

GetUserID 5684
OnGetDisplayName 5685
OnGetUserID 5685
OnIsLoggedIn 5686
OnLogin 5686
OnLogout 5687
TCustomEndUserAdapter events 5685
TCustomEndUserAdapter Members 5678
TCustomEndUserAdapter methods 5682
TCustomEndUserAdapterAction class 5687
about TCustomEndUserAdapterAction class 5687
Adapter 5692
TCustomEndUserAdapterAction Members 5688
TCustomEndUserAdapterAction properties 5691
TCustomEndUserSessionAdapter class 5692
about TCustomEndUserSessionAdapter class 5692
ExecuteLogin 5699
ExecuteLogout 5699
GetDisplayNameValue 5699
GetLoggedInValue 5700
GetUserID 5700
TCustomEndUserSessionAdapter Members 5694
TCustomEndUserSessionAdapter methods 5698
TCustomEndUserSessionAdapter properties 5700
UserID 5701
TCustomGroupBox class 2381
about TCustomGroupBox class 2381
AdjustClientRect 2382
Create 2382
CreateParams 2383
Paint 2383
TCustomGroupBox Members 2382
TCustomGroupBox methods 2382
TCustomIpClient class 1400
about TCustomIpClient class 1400
Close 1402
Connect 1403
Connected 1405
Create 1402
Disconnect 1403
DoConnect 1403
DoDisconnect 1404
GetThreadObject 1404
OnConnect 1405
OnDisconnect 1406
Open 1404
TCustomIpClient events 1405
TCustomIpClient Members 1400
TCustomIpClient methods 1402
TCustomIpClient properties 1405
TCustomLabel class 2384
about TCustomLabel class 2384
AdjustBounds 2385
Alignment 2388
AutoSize 2389
Canvas 2390
Caption 2392
Create 2385
DoDrawText 2386
EllipsisPosition 2395
FocusControl 2395
GetLabelText 2386
GlowSize 2395
Layout 2396
Loaded 2387
Notification 2387
Paint 2387
SetAutoSize 2388
ShowAccelChar 2396
TCustomLabel Members 2384
TCustomLabel methods 2385
TCustomLabel properties 2388
Transparent 2397
WordWrap 2397
TCustomListBox class 2398
about TCustomListBox class 2398
AddItem 2402
AutoComplete 2423

AutoCompleteDelay 2423
BorderStyle 2423
Canvas 2424
Clear 2403
ClearSelection 2403
Columns 2426
CopySelection 2404
Count 2427
Create 2400
CreateParams 2404
CreateWnd 2405
DeleteSelected 2406
DeleteString 2406
Destroy 2401
DestroyWnd 2406
DoFindData 2407
DoGetData 2407
DoGetDataObject 2408
DragCanceled 2408
DrawItem 2408
ExtendedSelect 2427
FMoving 2401
GetCount 2409
GetItemData 2409
GetItemIndex 2410
GetSelCount 2410
GetSelected 2410
IntegralHeight 2428
InternalGetItemData 2411
InternalSetItemData 2411
ItemAtPos 2411
ItemHeight 2428
ItemRect 2413
Items 2429
KeyPress 2415
LoadRecreateItems 2415
MeasureItem 2415
OnData 2438
OnDataFind 2439
OnDataObject 2439
OnDrawItem 2439
OnMeasureItem 2442
ParentColor 2431
ResetContent 2416
SaveRecreateItems 2416
ScrollWidth 2432
SelectAll 2416
Selected 2432
SetItemData 2417
SetItemIndex 2417
SetMultiSelect 2417
Sorted 2434
Style 2436
TabStop 2436
TabWidth 2437
TCustomListBox events 2438
TCustomListBox fields 2401
TCustomListBox Members 2399
TCustomListBox methods 2402
TCustomListBox properties 2422
TopIndex 2437
WndProc 2418
TCustomLocateFileService class 6181
about TCustomLocateFileService class 6181
FindIncludeFile 6182
FindStream 6183
FindTemplateFile 6183
GetFileStream 6184
GetTemplateStream 6184
ImplFindIncludeFile 6185
ImplGetTemplateStream 6185
OnFindIncludeFile 6186
OnFindStream 6187
OnFindTemplateFile 6187
TCustomLocateFileService events 6186
TCustomLocateFileService Members 6181
TCustomLocateFileService methods 6182
TCustomLoginFormAdapter class 5701

about TCustomLoginFormAdapter class 5701
EndUser 5707
ExecuteLogin 5707
GetNextPage 5708
ImplCanAddActionClass 5708
ImplCanAddFieldClass 5709
ImplGetActionsList 5709
ImplGetFieldsList 5710
ImplNotifyDeactivate 5710
NextPage 5713
OnLogin 5715
OnValidatePassword 5716
OnValidateUserName 5716
Password 5714
PasswordRequired 5714
RaiseBlankPassword 5711
RaiseBlankUserName 5711
TCustomLoginFormAdapter events 5715
TCustomLoginFormAdapter Members 5702
TCustomLoginFormAdapter methods 5706
TCustomLoginFormAdapter properties 5713
UserList 5711
UserName 5714
ValidatePassword 5712
ValidateUserName 5712
TCustomLoginFormAdapterAction class 5717
about TCustomLoginFormAdapterAction class 5717
Adapter 5721
TCustomLoginFormAdapterAction Members 5718
TCustomLoginFormAdapterAction properties 5721
TCustomMemo class 2443
about TCustomMemo class 2443
CaretPos 2450
Create 2446
CreateParams 2447
CreateWindowHandle 2447
Destroy 2446
GetCaretPos 2448
KeyPress 2448
Lines 2451
Loaded 2448
ScrollBars 2452
SetCaretPos 2449
SetLines 2449
SetScrollBars 2449
SetWordWrap 2450
TCustomMemo Members 2444
TCustomMemo methods 2446
TCustomMemo properties 2450
WantReturns 2454
WantTabs 2454
WordWrap 2455
TCustomPagedAdapter class 5721
about TCustomPagedAdapter class 5721
TCustomPagedAdapterAction class 5722
about TCustomPagedAdapterAction class 5722
Adapter 5727
GetAdapter 5727
TCustomPagedAdapterAction Members 5723
TCustomPagedAdapterAction methods 5727
TCustomPagedAdapterAction properties 5727
TCustomPageDispatcher class 6188
about TCustomPageDispatcher class 6188
DefaultPage 6195
DispatchPageName 6190
EndUser 6190
GetCanView 6191
GetDefaultPageName 6191
GetLoginRequired 6191
GetRequestedPage 6192
ImplDispatchPageName 6192
ImplGetCanView 6192
ImplGetDefaultPageName 6192
ImplGetLoginRequired 6193
ImplHandleRequest 6193
ImplPageNameOfRequest 6193
ImplRedirectToPageName 6193
IsLoggedIn 6194

OnAfterDispatchPage 6196
OnAfterRedirectToPage 6197
OnBeforeDispatchPage 6197
OnBeforeRedirectToPage 6198
OnCanViewPage 6198
OnIsLoginRequired 6199
OnPageAccessDenied 6200
OnPageNotFound 6200
PageNameOfRequest 6194
ProduceLoginPage 6194
RedirectToPageName 6194
TCustomPageDispatcher events 6196
TCustomPageDispatcher Members 6188
TCustomPageDispatcher methods 6189
TCustomPageDispatcher properties 6195
ValidatePage 6195
TCustomScreenTipsManager class 305
about TCustomScreenTipsManager class 305
AddPopup 308
BorderStyle 313
ClearScreenTips 308
Create 306
CreateScreenTip 308
Destroy 307
DisplayOffset 313
DoCreateScreenTip 309
DoDuplicateCreateError 309
DoGetScreenTipClass 309
DoScreenTipsLoaded 309
DoShow 309
DoShowScreenTip 310
Font 313
Footer 313
FooterImage 314
GenerateScreenTips 310
GetScreenTip 310
GradientEndColor 314
GradientStartColor 314
LinkedActionLists 314
Loaded 310
Notification 311
OnCreateScreenTip 315
OnDuplicateCreateError 315
OnGetScreenTipWinClass 316
OnScreenTipsLoaded 316
OnShowScreenTip 316
Options 314
RemovePopup 311
RemoveScreenTip 312
ScreenTips 315
TCustomScreenTipsManager events 315
TCustomScreenTipsManager Members 305
TCustomScreenTipsManager methods 307
TCustomScreenTipsManager properties 312
UpdateActionInfo 312
UpdateOptions 312
TCustomScreenTipsPopup class 316
about TCustomScreenTipsPopup class 316
Associate 321
Create 317
Destroy 318
GetScreenTipItem 319
Glyph 321
LinkPopupToManager 319
Notification 319
Paint 320
PopupType 321
RemoveLinkToManager 320
ScreenTip 322
ScreenTipManager 322
ShowHint 322
TCustomScreenTipsPopup Members 317
TCustomScreenTipsPopup methods 318
TCustomScreenTipsPopup properties 321
Transparent 324
UpdateAssociateToManager 320
TCustomServerSocket class 39
about TCustomServerSocket class 39

Destroy 41
DoActivate 42
FServerSocket 42
GetGetSocketEvent 43
GetGetThreadEvent 43
GetOnClientError 44
GetOnClientEvent 44
GetOnThreadEnd 44
GetOnThreadStart 45
GetServerType 45
GetThreadCacheSize 45
OnClientConnect 51
OnClientDisconnect 52
OnClientError 52
OnClientRead 53
OnClientWrite 53
OnGetSocket 54
OnGetThread 54
OnThreadEnd 55
OnThreadStart 55
ServerType 49
SetGetSocketEvent 46
SetGetThreadEvent 46
SetOnClientError 47
SetOnClientEvent 47
SetOnThreadEnd 47
SetOnThreadStart 48
SetServerType 48
SetThreadCacheSize 49
TCustomServerSocket events 50
TCustomServerSocket fields 42
TCustomServerSocket Members 40
TCustomServerSocket methods 42
TCustomServerSocket properties 49
ThreadCacheSize 50
TCustomSessionsService class 6746
about TCustomSessionsService class 6746
CheckStatusChange 6748
Clear 6749
Connect 6749
ConnectActive 6750
Create 6747
DefaultTimeout 6754
DoEndSession 6750
DoStartSession 6750
GetSessionIDs 6751
LoadFromFile 6751
LoadFromStream 6751
MaxSessions 6754
NewSession 6752
NotifyStatusChange 6752
OnEndSession 6755
OnStartSession 6755
SaveToFile 6753
SaveToStream 6753
TCustomSessionsService events 6755
TCustomSessionsService Members 6747
TCustomSessionsService methods 6748
TCustomSessionsService properties 6754
TCustomSocket class 56
about TCustomSocket class 56
Error 58
Event 58
OnAccept 59
OnConnect 60
OnConnecting 60
OnDisconnect 61
OnError 61
OnListen 61
OnLookup 62
OnRead 63
OnWrite 63
TCustomSocket events 59
TCustomSocket Members 56
TCustomSocket methods 57
TCustomSQLDataSet class 1520
about TCustomSQLDataSet class 1520
Active 1552

AddFieldDesc 1527
AddIndexDefs 1527
AfterClose 1568
AfterOpen 1568
AfterRefresh 1568
AfterScroll 1568
BeforeClose 1569
BeforeOpen 1569
BeforeRefresh 1569
BeforeScroll 1569
BlobBuffer 1554
CheckPrepareError 1527
ClearIndexDefs 1528
CloseCursor 1528
CloseStatement 1528
CommandText 1554
CommandType 1555
Create 1524
Create constructor 1524
CreateBlobStream 1528
CurrentBlobSize 1556
DataLink 1556
DataSource 1557
DbxCommandType 1557
DefineProperties 1529
DesignerData 1557
Destroy 1525
ExecSQL 1529
ExecuteStatement 1529
FreeBuffers 1530
FreeCommand 1530
FreeReader 1530
GetBlobFieldData 1530
GetCanModify 1531
GetCommandNames 1531
GetDataSource 1531
GetDetailLinkFields 1531
GetFieldData 1532
GetFieldData method 1532
GetKeyFieldNames 1533
GetKeyFieldNames method 1533
GetMetadata 1558
GetObjectTypeNames 1533
GetOutputParams 1534
GetQuoteChar 1534
GetRecord 1534
GetRecordCount 1534
GetSortFieldNames 1535
IndexDefs 1558
InitRecord 1535
InternalClose 1535
InternalConnection 1559
InternalFreeCommand 1535
InternalHandleException 1535
InternalInitFieldDefs 1536
InternalOpen 1536
InternalRefresh 1536
IsCursorOpen 1536
IsSequenced 1536
Loaded 1537
Locate 1538
LocateRecord 1538
Lookup 1538
MaxBlobSize 1559
NativeCommand 1560
NumericMapping 1560
ObjectView 1560
OnCalcFields 1569
OpenCursor 1539
OpenSchema 1539
ParamByName 1539
ParamCheck 1561
ParamCount 1562
Params 1562
ParseDeleteSql 1569
ParseInsertSql 1570
ParseSelectSql 1570
ParseUpdateSql 1570

Prepared 1563
PrepareStatement 1540
ProcParams 1564
PropertyChanged 1540
PSEndTransaction 1540
PSExecute 1541
PSExecuteStatement 1541
PSGetAttributes 1541
PSGetCommandText 1542
PSGetCommandType 1542
PSGetDefaultOrder 1542
PSGetIndexDefs 1543
PSGetKeyFieldsW 1543
PSGetParams 1544
PSGetQuoteCharW 1544
PSGetTableNameW 1544
PSGetUpdateException 1544
PSInTransaction 1545
PSIsSQLBased 1545
PSIsSQLSupported 1546
PSReset 1546
PSSetCommandText 1546
PSSetParams 1547
PSStartTransaction 1547
PSUpdateRecord 1548
RecordCount 1564
RowsAffected 1564
SchemaInfo 1565
SchemaName 1565
SetBufListSize 1548
SetCommandText 1549
SetFCommandText 1549
SetFieldData 1549
SetMaxBlobSize 1549
SetParamsFromCursor 1549
SetSchemaInfo 1550
SetSortFieldNames 1551
SortFieldNames 1566
SQLConnection 1566
TCustomSQLDataSet events 1567
TCustomSQLDataSet Members 1521
TCustomSQLDataSet methods 1525
TCustomSQLDataSet properties 1551
TransactionLevel 1567
UpdateIndexDefs 1551
TCustomStaticText class 2457
about TCustomStaticText class 2457
Alignment 2461
AutoSize 2461
BorderStyle 2462
Create 2458
CreateParams 2459
FocusControl 2462
Loaded 2459
Notification 2460
SetAutoSize 2460
ShowAccelChar 2462
TCustomStaticText Members 2458
TCustomStaticText methods 2459
TCustomStaticText properties 2461
Transparent 2463
TCustomStringsValuesList class 5728
about TCustomStringsValuesList class 5728
Create 5733
Destroy 5733
GetNameFieldClass 5734
GetValueFieldClass 5734
ImplEndIterator 5734
ImplGetListName 5735
ImplGetListNameOfValue 5735
ImplGetListValue 5735
ImplNextIteration 5736
ImplNotifyDeactivate 5736
ImplStartIterator 5736
OnPrepareStrings 5738
PrepareStrings 5737
Strings 5737
StringsPrepared 5737

TCustomStringsValuesList events 5738
TCustomStringsValuesList Members 5729
TCustomStringsValuesList methods 5733
TCustomStringsValuesList properties 5737
TCustomTcpServer class 1406
about TCustomTcpServer class 1406
Accept 1409
Accept method 1409
BlockMode 1411
Close 1410
Create 1408
Destroy 1408
DoAccept 1410
Listen 1410
Listening 1412
OnAccept 1413
OnGetThread 1415
OnListening 1416
Open 1411
ServerSocketThread 1412
TCustomTcpServer events 1412
TCustomTcpServer Members 1407
TCustomTcpServer methods 1409
TCustomTcpServer properties 1411
WaitForConnection 1411
TCustomVariantType class 4987
_AddRef 4993
_Release 4993
about TCustomVariantType class 4987
BinaryOp 4994
Cast 4995
CastTo 4995
CastToOle 4996
Clear 4996
Compare 4997
CompareOp 4998
Copy 4999
Create 4991
Create constructor 4990
Destroy 4991
DispInvoke 4999
IsClear 5000
LeftPromotion 5000
OlePromotion 5001
QueryInterface 5002
RaiseCastError 5002
RaiseDispError 5003
RaiseInvalidOp 5003
RightPromotion 5004
SimplisticClear 5005
SimplisticCopy 5005
TCustomVariantType Members 4988
TCustomVariantType methods 4992
TCustomVariantType properties 5013
UnaryOp 5006
VarDataCast 5006
VarDataCastTo 5007
VarDataCastTo method 5007
VarDataCastToOleStr 5007
VarDataClear 5008
VarDataCopy 5008
VarDataCopyNoInd 5009
VarDataFromOleStr 5009
VarDataFromStr 5010
VarDataInit 5010
VarDataIsArray 5010
VarDataIsByRef 5011
VarDataIsEmptyParam 5011
VarDataIsFloat 5011
VarDataIsNumeric 5012
VarDataIsOrdinal 5012
VarDataIsStr 5013
VarDataToStr 5013
VarType 5014
TCustomWebAppComponents class 6201
about TCustomWebAppComponents class 6201
AdapterDispatcher 6211
ApplicationAdapter 6212

CreateContext 6204
DispatchActions 6212
EndUserAdapter 6213
GetAdapterDispatcher 6205
GetLocateFileService 6205
GetPageDispatcher 6205
GetWebApplicationInfo 6206
GetWebEndUser 6206
GetWebUserList 6206
ImplFinishContext 6207
ImplHandleException 6207
ImplHandleRequest 6208
ImplInitContext 6208
InitModule 6209
InitModules 6210
ListRequestHandlers 6210
LocateFileService 6213
Notification 6211
OnException 6215
PageDispatcher 6214
Sessions 6214
TCustomWebAppComponents events 6215
TCustomWebAppComponents Members 6202
TCustomWebAppComponents methods 6204
TCustomWebAppComponents properties 6211
UserListService 6215
TCustomWebAppDataModule class 6539
about TCustomWebAppDataModule class 6539
AppServices 6544
Create 6541
CreateNew 6541
Destroy 6542
GetWebAppServices 6542
Notification 6543
QueryInterface 6543
TCustomWebAppDataModule Members 6540
TCustomWebAppDataModule methods 6542
TCustomWebAppDataModule properties 6543
TCustomWebAppPageModule class 6544
about TCustomWebAppPageModule class 6544
AppServices 6549
Create 6546
CreateNew 6547
Destroy 6547
GetWebAppServices 6548
Notification 6548
QueryInterface 6548
SetAppServices 6549
TCustomWebAppPageModule Members 6545
TCustomWebAppPageModule methods 6547
TCustomWebAppPageModule properties 6549
TCustomWebDataModule class 6550
about TCustomWebDataModule class 6550
Create 6551
CreateNew 6552
Destroy 6552
GetSiteModuleHelper 6553
GetWebSnapObjClass 6553
Notification 6553
OnActivate 6556
OnDeactivate 6556
Request 6554
Response 6554
Session 6555
SiteModuleHelper 6555
TCustomWebDataModule events 6555
TCustomWebDataModule Members 6551
TCustomWebDataModule methods 6553
TCustomWebDataModule properties 6554
TCustomWebPageModule class 6556
about TCustomWebPageModule class 6556
Create 6559
CreateNew 6559
DefaultAction 6561
Destroy 6560
GetSiteModuleHelper 6560
GetSitePageModuleHelper 6561
Notification 6561

OnAfterDispatchPage 6563
OnAfterRedirectToPage 6563
OnBeforeDispatchPage 6564
OnBeforeRedirectToPage 6564
PageProducer 6562
SitePageModuleHelper 6562
TCustomWebPageModule events 6563
TCustomWebPageModule Members 6557
TCustomWebPageModule methods 6560
TCustomWebPageModule properties 6561
TCustomWebUserList class 6781
about TCustomWebUserList class 6781
CheckAccessRights 6783
Create 6782
Destroy 6782
LoadFromFile 6783
OnAfterCheckAccessRights 6785
OnAfterValidateUser 6786
OnBeforeCheckAccessRights 6786
OnBeforeValidateUser 6786
OnUserIDNotFound 6787
OnValidateUserError 6787
SaveToFile 6784
TCustomWebUserList events 6785
TCustomWebUserList Members 6781
TCustomWebUserList methods 6782
TCustomWebUserList properties 6784
UserItems 6784
Users 6785
ValidateUser 6784
TCustomWinSocket class 64
about TCustomWinSocket class 64
Accept 68
Addr 82
AsyncInitSocket 68
ASyncStyles 83
Close 69
Connect 69
Connected 83
Create 66
Data 84
DefaultHandler 70
Destroy 67
Disconnect 70
DoListen 70
DoOpen 71
Error 71
Event 72
Handle 84
InitSocket 72
Listen 73
LocalAddress 85
LocalHost 85
LocalPort 86
Lock 74
LookupName 74
LookupService 75
LookupState 86
OnErrorEvent 89
OnSocketEvent 90
Open 75
Read 76
ReceiveBuf 77
ReceiveLength 77
ReceiveText 78
RemoteAddr 87
RemoteAddress 87
RemoteHost 88
RemotePort 88
SendBuf 78
SendStream 79
SendStreamThenDrop 79
SendText 80
SetAsyncStyles 81
SocketHandle 89
TCustomWinSocket events 89
TCustomWinSocket Members 64
TCustomWinSocket methods 67

TCustomWinSocket properties 82
Unlock 81
Write 81
TCustomXSLPageProducer class 7878
about TCustomXSLPageProducer class 7878
Content 7885
ContentFromStream 7885
ContentFromString 7886
ContentFromWideString 7886
Dispatcher 7891
DispatcherComponent 7892
GetAppDispatcher 7887
GetProducerTemplateFileName 7887
GetProducerTemplateStream 7887
GetTemplateFileType 7887
GetTemplateStream 7888
GetXMLData 7888
GetXMLDocument 7888
GetXSLData 7888
HasHTMLBrowserView 7888
HasScriptView 7889
HasXMLBrowserView 7889
HasXSLBrowserView 7889
Notification 7889
ProduceContent 7890
ProduceContentFromStream 7890
ProduceContentFromString 7890
SetAppDispatcher 7891
TCustomXSLPageProducer Members 7879
TCustomXSLPageProducer methods 7884
TCustomXSLPageProducer properties 7891
XMLData 7892
TCustomZStream class 7905
about TCustomZStream class 7905
Create 7906
DoProgress 7906
OnProgress 7907
TCustomZStream events 7907
TCustomZStream Members 7906
TCustomZStream methods 7906
TDataBlock class 197
about TDataBlock class 197
BytesReserved 205
Clear 200
Create 199
Destroy 199
GetBytesReserved 201
GetMemory 201
GetSignature 201
GetSize 202
GetStream 202
IgnoreStream 202
InitData 202
Memory 205
Read 203
SetSignature 203
SetSize 204
Signature 206
Size 206
Stream 206
TDataBlock Members 197
TDataBlock methods 200
TDataBlock properties 204
Write 204
TDataBlockInterpreter class 207
about TDataBlockInterpreter class 207
AddDispatch 209
CallCreateObject 210
CallFreeObject 210
CallGetIDsOfNames 210
CallGetServerList 210
CallInvoke 211
CanCreateObject 211
Create 208
CreateObject 211
Destroy 208
DoCreateObject 211
DoCustomAction 211

DoFreeObject 212
DoGetAppServerList 212
DoGetIDsOfNames 212
DoGetServerList 212
DoInvoke 213
InternalCreateObject 213
InterpretData 213
LockObject 213
ReleaseObject 214
RemoveDispatch 214
StoreObject 214
TDataBlockInterpreter Members 207
TDataBlockInterpreter methods 209
UnlockObject 214
TDataDispatch class 215
about TDataDispatch class 215
Create 216
Destroy 217
DispatchIndex 219
GetIDsOfNames 217
GetTypeInfo 218
GetTypeInfoCount 218
Invoke 219
TDataDispatch Members 215
TDataDispatch methods 217
TDataDispatch properties 219
TDateTime
TDateTime class 3325
Variant class 3463
TDateTime class 3323
- 3334
-- 3334
- operator 3334
-- operator 3334
!= 3335
+ 3335
+ operator 3335
++ 3336
++ operator 3336
+= 3337
+= operator 3336
< 3337
<= 3337
= 3338
-= 3339
= operator 3338
-= operator 3338
== 3339
> 3339
>= 3340
about TDateTime class 3323
CurrentDate 3328
CurrentDateTime 3328
CurrentTime 3328
DateString 3329
DateTimeString 3329
DayOfWeek 3329
DecodeDate 3330
DecodeTime 3330
double 3340
FileDate 3330
FileDateToDateTime 3331
FormatString 3331
int 3340
System::String 3341
TDateTime 3325
TDateTime constructor 3325
TDateTime enumerations 3327
TDateTime Members 3324
TDateTime methods 3327
TDateTime operators 3333
TimeString 3333
TDateTime*
Variant class 3464
TDateTimeBase class 3341
about TDateTimeBase class 3341
TDateTimeBase data members 3342
TDateTimeBase Members 3341

Val 3342
TDefaultFieldsAdapter class 5738
about TDefaultFieldsAdapter class 5738
Actions 5746
AddDefaultActions 5743
AddDefaultFields 5744
Data 5747
DefaultWebComponentsChange 5744
GetActionsClass 5744
GetDataClass 5745
ImplGetVisibleActions 5745
ImplGetVisibleFields 5745
ImplSetActions 5746
ImplSetData 5746
TDefaultFieldsAdapter Members 5739
TDefaultFieldsAdapter methods 5743
TDefaultFieldsAdapter properties 5746
TDefaultFieldsPagedAdapter class 5747
about TDefaultFieldsPagedAdapter class 5747
AddDefaultActions 5753
CurrentPage 5760
EncodeActionParams 5754
EndOfPage 5754
ExtractRequestParams 5754
GetEOF 5755
GetFirstRecord 5755
GetNextRecord 5755
GetPageCount 5756
GetRecordCount 5756
GetRecordIndex 5756
ImplCanAddActionClass 5757
ImplEndIterator 5757
ImplGetActionsList 5757
ImplNextIteration 5758
ImplNotifyDeactivate 5758
ImplStartIterator 5759
OnGetEOF 5762
OnGetFirstRecord 5763
OnGetNextRecord 5763
OnGetRecordCount 5763
OnGetRecordIndex 5763
PageCount 5761
PageSize 5761
RecordCount 5761
RecordIndex 5762
SetPageSize 5759
SkipToPage 5760
TDefaultFieldsPagedAdapter events 5762
TDefaultFieldsPagedAdapter Members 5749
TDefaultFieldsPagedAdapter methods 5753
TDefaultFieldsPagedAdapter properties 5760
TDefinition class 6930
about TDefinition class 6930
AfterConstruction 6935
GetBindings 6936
GetImports 6936
GetMessages 6936
GetName 6936
GetPortTypes 6937
GetServices 6937
GetTargetNamespace 6937
GetTypes 6937
SetName 6938
SetTargetNamespace 6938
TDefinition Members 6930
TDefinition methods 6935
TDependencies class 3063
about TDependencies class 3063
Create 3064
GetOwner 3064
Items 3065
TDependencies Members 3063
TDependencies methods 3064
TDependencies properties 3065
TDependency class 3066
about TDependency class 3066
GetDisplayName 3067
IsGroup 3067

Name 3068
TDependency Members 3066
TDependency methods 3067
TDependency properties 3067
TDesigntimeWarnings class 6504
about TDesigntimeWarnings class 6504
AddObject 6505
AddString 6505
Create 6504
Destroy 6504
Strings 6506
TDesigntimeWarnings Members 6504
TDesigntimeWarnings methods 6505
TDesigntimeWarnings properties 6506
TDisplayOffset class 324
about TDisplayOffset class 324
Control 325
Create 324
TDisplayOffset Members 324
TDisplayOffset properties 325
X 325
Y 325
TDocumentation class 6938
about TDocumentation class 6938
TDocumentationCollection class 6939
about TDocumentationCollection class 6939
Add 6944
GetDocumentation 6945
Insert 6945
TDocumentationCollection Members 6939
TDocumentationCollection methods 6944
TDocumented class 6945
about TDocumented class 6945
AfterConstruction 6950
GetDocumentationCollection 6951
TDocumented Members 6946
TDocumented methods 6950
TDOMStrings class 7181
about TDOMStrings class 7181
Add 7182
Clear 7183
Count 7183
Create 7182
Destroy 7182
IndexOf 7183
IndexOfIgnoreCase 7183
Insert 7183
Strings 7184
TDOMStrings Members 7181
TDOMStrings methods 7182
TDOMStrings properties 7184
TDOMVendor class 7514
about TDOMVendor class 7514
Description 7515
DOMImplementation 7515
TDOMVendor Members 7514
TDOMVendor methods 7514
TDOMVendorList class 7516
about TDOMVendorList class 7516
Add 7517
Count 7517
Find 7518
GetVendors 7518
Remove 7519
TDOMVendorList Members 7516
TDOMVendorList methods 7517
TDOMVendorList properties 7519
Vendors 7519
tebExplorerBarDontCare enumeration member 4603
tebExplorerBarRoot enumeration member 4603
tebHeaderBackgroundHot enumeration member 4603
tebHeaderBackgroundNormal enumeration member 4603
tebHeaderBackgroundPressed enumeration member 4603
tebHeaderCloseHot enumeration member 4603
tebHeaderCloseNormal enumeration member 4603
tebHeaderClosePressed enumeration member 4603
tebHeaderPinHot enumeration member 4603
tebHeaderPinNormal enumeration member 4603

tebHeaderPinPressed enumeration member 4603
tebHeaderPinSelectedHot enumeration member 4603
tebHeaderPinSelectedNormal enumeration member 4603
tebHeaderPinSelectedPressed enumeration member 4603
tebIEBarMenuHot enumeration member 4603
tebIEBarMenuNormal enumeration member 4603
tebIEBarMenuPressed enumeration member 4603
tebNormalGroupBackground enumeration member 4603
tebNormalGroupCollapseHot enumeration member 4603
tebNormalGroupCollapseNormal enumeration member 4603
tebNormalGroupCollapsePressed enumeration member 4603
tebNormalGroupExpandHot enumeration member 4603
tebNormalGroupExpandNormal enumeration member 4603
tebNormalGroupExpandPressed enumeration member 4603
tebNormalGroupHead enumeration member 4603
tebSpecialGroupBackground enumeration member 4603
tebSpecialGroupCollapseHot enumeration member 4603
tebSpecialGroupCollapsePressed enumeration member 4603
tebSpecialGroupCollapseSpecial enumeration member 4603
tebSpecialGroupExpandHot enumeration member 4603
tebSpecialGroupExpandPressed enumeration member 4603
tebSpecialGroupExpandSpecial enumeration member 4603
tebSpecialGroupHead enumeration member 4603
teButton enumeration member 4600
teClock enumeration member 4600
teComboBox enumeration member 4600
TEdit class 2463
about TEdit class 2463
Align 2467
Alignment 2471
Anchors 2471
AutoSelect 2472
AutoSize 2472
BevelEdges 2473
BevelInner 2474
BevelKind 2474
BevelOuter 2474
BevelWidth 2475
BiDiMode 2475
BorderStyle 2476
CharCase 2476
Color 2476
Constraints 2477
Ctl3D 2478
DoubleBuffered 2478
DragCursor 2480
DragKind 2481
DragMode 2481
Enabled 2482
Font 2483
HideSelection 2484
ImeMode 2484
ImeName 2485
MaxLength 2486
NumbersOnly 2486
OEMConvert 2486
OnChange 2501
OnClick 2502
OnContextPopup 2504
OnDblClick 2504
OnDragDrop 2505
OnDragOver 2508
OnEndDock 2511
OnEndDrag 2511
OnEnter 2512
OnExit 2515
OnKeyDown 2519
OnKeyPress 2522
OnKeyUp 2523
OnMouseActivate 2525
OnMouseDown 2526
OnMouseEnter 2528
OnMouseLeave 2529
OnMouseMove 2529
OnMouseUp 2531
OnStartDock 2533
OnStartDrag 2534
ParentBiDiMode 2487

ParentColor 2487
ParentCtl3D 2488
ParentDoubleBuffered 2488
ParentFont 2488
ParentShowHint 2489
PasswordChar 2489
PopupMenu 2490
ReadOnly 2495
ShowHint 2495
TabOrder 2497
TabStop 2497
TEdit events 2501
TEdit Members 2464
TEdit properties 2466
Text 2498
TextHint 2499
Visible 2499
TEditAction class 1791
about TEditAction class 1791
Control 1795
Destroy 1792
GetControl 1793
HandlesTarget 1793
Notification 1794
TEditAction Members 1792
TEditAction methods 1793
TEditAction properties 1794
UpdateTarget 1794
TEditCopy class 1795
about TEditCopy class 1795
ExecuteTarget 1796
TEditCopy Members 1796
TEditCopy methods 1796
TEditCut class 1797
about TEditCut class 1797
ExecuteTarget 1798
TEditCut Members 1798
TEditCut methods 1798
TEditDelete class 1799
about TEditDelete class 1799
ExecuteTarget 1800
TEditDelete Members 1799
TEditDelete methods 1800
UpdateTarget 1800
TEditPaste class 1801
about TEditPaste class 1801
ExecuteTarget 1802
TEditPaste Members 1802
TEditPaste methods 1802
UpdateTarget 1803
TEditSelectAll class 1803
about TEditSelectAll class 1803
ExecuteTarget 1804
TEditSelectAll Members 1804
TEditSelectAll methods 1804
UpdateTarget 1805
TEditUndo class 1805
about TEditUndo class 1805
ExecuteTarget 1807
TEditUndo Members 1806
TEditUndo methods 1806
UpdateTarget 1807
teEdit enumeration member 4600
teEditCaret enumeration member 4599
teEditDontCare enumeration member 4599
teEditRoot enumeration member 4599
teEditTextAssist enumeration member 4599
teEditTextDisabled enumeration member 4599
teEditTextFocused enumeration member 4599
teEditTextHot enumeration member 4599
teEditTextNormal enumeration member 4599
teEditTextReadOnly enumeration member 4599
teEditTextSelected enumeration member 4599
teExplorerBar enumeration member 4600
teHeader enumeration member 4600
teListView enumeration member 4600
teMenu enumeration member 4600
TEncoding class 3969

about TEncoding class 3969
ASCII 3985
BigEndianUnicode 3985
Convert 3975
Convert method 3974
Default 3985
FIsSingleByte 3973
FMaxCharSize 3974
FreeEncodings 3975
GetBufferEncoding 3976
GetByteCount 3976
GetByteCount method 3976
GetBytes 3977
GetBytes method 3977
GetCharCount 3978
GetCharCount method 3978
GetChars 3979
GetChars method 3979
GetEncoding 3980
GetMaxByteCount 3980
GetMaxCharCount 3980
GetPreamble 3981
GetString 3981
GetString method 3981
IsSingleByte 3986
IsStandardEncoding 3984
TEncoding fields 3973
TEncoding Members 3972
TEncoding methods 3974
TEncoding properties 3984
Unicode 3986
UTF7 3986
UTF8 3987
TEndUserAdapter class 5764
about TEndUserAdapter class 5764
Actions 5770
Data 5770
LoginPage 5771
OnAfterExecuteAction 5771
OnAfterGetActionResponse 5772
OnBeforeExecuteAction 5772
OnBeforeGetActionResponse 5772
OnGetActionParams 5773
OnGetDisplayName 5773
OnHasRights 5774
OnIsLoggedIn 5774
OnLogin 5775
OnLogout 5775
OnProduceLoginPage 5776
TEndUserAdapter events 5771
TEndUserAdapter Members 5765
TEndUserAdapter properties 5770
TEndUserAdapterLoginFormAction class 5776
about TEndUserAdapterLoginFormAction class 5776
DisplayLabel 5782
GetDefaultActionName 5780
ImplExecuteActionRequest 5781
ImplWebEnabled 5781
OnAfterExecute 5782
OnAfterGetResponse 5783
OnBeforeExecute 5783
OnBeforeGetResponse 5783
OnGetParams 5784
TEndUserAdapterLoginFormAction events 5782
TEndUserAdapterLoginFormAction Members 5777
TEndUserAdapterLoginFormAction methods 5780
TEndUserAdapterLoginFormAction properties 5781
TEndUserAdapterLogoutAction class 5784
about TEndUserAdapterLogoutAction class 5784
DisplayLabel 5790
GetDefaultActionName 5789
ImplExecuteActionRequest 5789
ImplWebEnabled 5789
OnAfterExecute 5791
OnAfterGetResponse 5791
OnBeforeExecute 5791
OnBeforeGetResponse 5792
OnGetParams 5792

TEndUserAdapterLogoutAction events 5790
TEndUserAdapterLogoutAction Members 5785
TEndUserAdapterLogoutAction methods 5788
TEndUserAdapterLogoutAction properties 5790
TEndUserSessionAdapter class 5792
about TEndUserSessionAdapter class 5792
Actions 5799
Data 5800
LoginPage 5800
OnAfterExecuteAction 5800
OnAfterGetActionResponse 5801
OnBeforeExecuteAction 5801
OnBeforeGetActionResponse 5801
OnGetActionParams 5802
OnHasRights 5802
OnProduceLoginPage 5803
TEndUserSessionAdapter events 5800
TEndUserSessionAdapter Members 5794
TEndUserSessionAdapter properties 5799
tePage enumeration member 4600
teProgress enumeration member 4600
teRebar enumeration member 4600
Terminate
IWebSession interface 1103
TInvalidWebSession class 6219
TWebSession class 6255
TWebSessionImpl class 6769
Terminated
IWebSessionAttributes interface 1107
TService class 3084
TSessionItem class 383
TWebSessionImpl class 6771
TerminateSession
TSessions class 394
teScrollBar enumeration member 4600
teSpin enumeration member 4600
teStartPanel enumeration member 4600
teStatus enumeration member 4600
teTab enumeration member 4600
teTaskBand enumeration member 4600
teTaskBar enumeration member 4600
teToolBar enumeration member 4600
teToolTip enumeration member 4600
teTrackBar enumeration member 4600
teTrayNotify enumeration member 4600
teTreeview enumeration member 4600
TEvent class 3133
about TEvent class 3133
Create 3135
Create constructor 3135
ResetEvent 3136
SetEvent 3137
TEvent Members 3134
TEvent methods 3136
TEventLogger class 3068
about TEventLogger class 3068
Create 3069
Destroy 3070
LogMessage 3070
TEventLogger Members 3069
TEventLogger methods 3070
teWindow enumeration member 4600
text
IDOMNodeEx interface 7580
Text
IXMLNode interface 7697
TComboBox class 2224
TCustomEdit class 2379
TEdit class 2498
TSQLQuery class 1665
TWideStrings class 6854
TextAreaWrap
TAdapterDisplayField class 6380
TAdapterEditColumn class 6396
TCustomAdapterDisplayField class 6475
TextHint
TComboBox class 2226
TCustomComboBox class 2337

TCustomEdit class 2380
TEdit class 2499
TFault class 6951
about TFault class 6951
TFaults class 6952
about TFaults class 6952
Add 6957
GetFault 6958
Insert 6958
TFaults Members 6952
TFaults methods 6957
TFieldValuesAdapter class 5803
about TFieldValuesAdapter class 5803
Create 5809
Field 5812
GetValueFieldClass 5809
ImplEndIterator 5810
ImplGetField 5810
ImplGetValueIndex 5811
ImplNextIteration 5811
ImplStartIterator 5812
TFieldValuesAdapter Members 5805
TFieldValuesAdapter methods 5809
TFieldValuesAdapter properties 5812
TFieldValuesValueField class 5813
about TFieldValuesValueField class 5813
Field 5817
GetDefaultFieldName 5816
ImplAsFormatted 5816
ImplGetValue 5817
TFieldValuesValueField Members 5814
TFieldValuesValueField methods 5816
TFieldValuesValueField properties 5817
TFileAction class 1807
about TFileAction class 1807
FileName 1809
GetDialog 1809
TFileAction Members 1808
TFileAction methods 1809
TFileAction properties 1809
TFileExit class 1810
about TFileExit class 1810
Caption 1812
Enabled 1812
ExecuteTarget 1811
HandlesTarget 1811
HelpContext 1813
HelpKeyword 1813
HelpType 1814
Hint 1814
ImageIndex 1814
OnHint 1816
OnUpdate 1816
SecondaryShortCuts 1815
ShortCut 1815
TFileExit events 1816
TFileExit Members 1810
TFileExit methods 1811
TFileExit properties 1812
Visible 1815
TFileOpen class 1817
about TFileOpen class 1817
BeforeExecute 1826
Caption 1821
Create 1819
Dialog 1821
Enabled 1822
ExecuteTarget 1820
GetDialogClass 1820
HelpContext 1822
HelpKeyword 1823
HelpType 1823
Hint 1824
ImageIndex 1824
OnAccept 1826
OnCancel 1827
OnHint 1827
OnUpdate 1828

SecondaryShortCuts 1824
ShortCut 1825
TFileOpen events 1826
TFileOpen Members 1818
TFileOpen methods 1820
TFileOpen properties 1821
UseDefaultApp 1825
Visible 1825
TFileOpenWith class 1829
about TFileOpenWith class 1829
AfterOpen 1834
BeforeExecute 1834
ExecuteTarget 1831
FileName 1832
HelpContext 1832
HelpKeyword 1833
HelpType 1833
OnUpdate 1835
TFileOpenWith events 1834
TFileOpenWith Members 1830
TFileOpenWith methods 1831
TFileOpenWith properties 1832
TFilePageSetup class 1836
about TFilePageSetup class 1836
BeforeExecute 1842
Caption 1838
Dialog 1838
Enabled 1839
GetDialogClass 1837
HelpContext 1839
HelpKeyword 1840
HelpType 1840
Hint 1840
ImageIndex 1841
OnAccept 1843
OnCancel 1843
OnHint 1844
OnUpdate 1844
SecondaryShortCuts 1841
ShortCut 1842
TFilePageSetup events 1842
TFilePageSetup Members 1836
TFilePageSetup methods 1837
TFilePageSetup properties 1838
Visible 1842
TFilePrintSetup class 1845
about TFilePrintSetup class 1845
BeforeExecute 1852
Caption 1848
Dialog 1848
Enabled 1848
GetDialogClass 1847
HelpContext 1849
HelpKeyword 1849
HelpType 1850
Hint 1850
ImageIndex 1850
OnAccept 1852
OnCancel 1853
OnHint 1853
OnUpdate 1854
SecondaryShortCuts 1851
ShortCut 1851
TFilePrintSetup events 1852
TFilePrintSetup Members 1846
TFilePrintSetup methods 1847
TFilePrintSetup properties 1847
Visible 1851
TFileSaveAs class 1855
about TFileSaveAs class 1855
BeforeExecute 1861
Caption 1858
Create 1856
Dialog 1858
Enabled 1858
GetDialogClass 1857
HelpContext 1859
Hint 1859

ImageIndex 1860
OnAccept 1862
OnCancel 1862
OnHint 1863
OnUpdate 1863
SecondaryShortCuts 1860
ShortCut 1860
TFileSaveAs events 1861
TFileSaveAs Members 1855
TFileSaveAs methods 1857
TFileSaveAs properties 1857
Visible 1861
TFLDDesc class 1570
about TFLDDesc class 1570
bCalcField 1575
efldrRights 1575
efldvVchk 1575
FCalcField 1572
FFldNum 1572
FFLDRights 1572
FFldType 1572
FFLDVchk 1573
FLen 1573
FName 1573
FNullOffset 1573
FOffset 1573
FSubType 1574
FUnits1 1574
FUnits2 1574
iFldNum 1575
iFldType 1576
iLen 1576
iNullOffset 1576
iOffset 1576
iSubType 1576
iUnits1 1577
iUnits2 1577
szName 1577
TFLDDesc fields 1571
TFLDDesc Members 1571
TFLDDesc properties 1574
TFontEdit class 1864
about TFontEdit class 1864
BeforeExecute 1871
Caption 1866
Dialog 1867
Enabled 1867
GetDialogClass 1866
HelpContext 1867
HelpKeyword 1868
HelpType 1868
Hint 1869
ImageIndex 1869
OnAccept 1871
OnCancel 1872
OnHint 1872
OnUpdate 1872
SecondaryShortCuts 1870
ShortCut 1870
TFontEdit events 1871
TFontEdit Members 1865
TFontEdit methods 1866
TFontEdit properties 1866
Visible 1870
TFTCompT class 1311
about TFTCompT class 1311
CStringEvent 1312
fireEvent 1312
fireEventEx 1312
StringEvent 1312
TFTCompT events 1312
TFTCompT Members 1311
TFTCompT methods 1311
VStringEvent 1313
TFTCompTI class 1313
about TFTCompTI class 1313
Add 1315
Append 1315

Doubler 1315
GetAsObject 1316
TFTCompTI Members 1313
TFTCompTI methods 1315
UpCase 1316
TGeneratedFunctions class 6506
about TGeneratedFunctions class 6506
Add 6507
Create 6507
Destroy 6507
Exists 6508
ItemCount 6508
Names 6508
Script 6508
TGeneratedFunctions Members 6506
TGeneratedFunctions methods 6507
TGeneratedFunctions properties 6508
TGroupBox class 2535
about TGroupBox class 2535
Align 2537
Anchors 2541
BiDiMode 2541
Caption 2542
Color 2544
Constraints 2545
Ctl3D 2546
DockSite 2546
DoubleBuffered 2547
DragCursor 2549
DragKind 2550
DragMode 2550
Enabled 2551
Font 2552
OnAlignInsertBefore 2566
OnAlignPosition 2566
OnClick 2567
OnContextPopup 2569
OnDblClick 2570
OnDockDrop 2570
OnDockOver 2572
OnDragDrop 2574
OnDragOver 2577
OnEndDock 2580
OnEndDrag 2580
OnEnter 2581
OnExit 2584
OnGetSiteInfo 2588
OnMouseActivate 2590
OnMouseDown 2591
OnMouseEnter 2593
OnMouseLeave 2594
OnMouseMove 2594
OnMouseUp 2596
OnStartDock 2598
OnStartDrag 2599
OnUnDock 2600
Padding 2553
ParentBackground 2553
ParentBiDiMode 2554
ParentColor 2554
ParentCtl3D 2555
ParentDoubleBuffered 2555
ParentFont 2555
ParentShowHint 2556
PopupMenu 2556
ShowHint 2561
TabOrder 2563
TabStop 2563
TGroupBox events 2565
TGroupBox Members 2535
TGroupBox properties 2537
Visible 2564
THandleObject class 3137
about THandleObject class 3137
Create 3139
Destroy 3139
FHandle 3140
FLastError 3140

FUseCOMWait 3140
Handle 3142
LastError 3142
THandleObject fields 3140
THandleObject Members 3138
THandleObject methods 3140
THandleObject properties 3142
WaitFor 3141
WaitForMultiple 3141
TheaterMode
IWebBrowser2 interface 762
IWebBrowser2Disp interface 774
TInternetExplorer class 456
TWebBrowser class 543
THelpAction class 1874
about THelpAction class 1874
Create 1875
HandlesTarget 1875
THelpAction Members 1874
THelpAction methods 1875
UpdateTarget 1875
THelpContents class 1876
about THelpContents class 1876
ExecuteTarget 1877
THelpContents Members 1876
THelpContents methods 1877
THelpContextAction class 1877
about THelpContextAction class 1877
ExecuteTarget 1878
THelpContextAction Members 1878
THelpContextAction methods 1878
UpdateTarget 1879
THelpOnHelp class 1879
about THelpOnHelp class 1879
ExecuteTarget 1880
THelpOnHelp Members 1880
THelpOnHelp methods 1880
THelpTopicSearch class 1881
about THelpTopicSearch class 1881
ExecuteTarget 1881
THelpTopicSearch Members 1881
THelpTopicSearch methods 1881
Theme
TThemeServices class 4591
ThemedActnCtrls 4555
ThemedActnCtrls namespace 4555
Classes 4556
ThemedActnCtrls.TThemedAddRemoveItem 4556
ThemedActnCtrls.TThemedAddRemoveItem.CalcBounds 4557
ThemedActnCtrls.TThemedAddRemoveItem.DrawBackground 4557
ThemedActnCtrls.TThemedAddRemoveItem.DrawGlyph 4558
ThemedActnCtrls.TThemedAddRemoveItem.DrawSeparator 4558
ThemedActnCtrls.TThemedAddRemoveItem.DrawText 4558
ThemedActnCtrls.TThemedAddRemoveItem.DrawUnusedEdges 4559
ThemedActnCtrls.TThemedAddRemoveItem.GetImageSize 4559
ThemedActnCtrls.TThemedButtonControl 4559
ThemedActnCtrls.TThemedButtonControl.DrawBackground 4560
ThemedActnCtrls.TThemedButtonControl.DrawFrame 4560
ThemedActnCtrls.TThemedButtonControl.DrawGlyph 4560
ThemedActnCtrls.TThemedButtonControl.DrawLargeGlyph 4561
ThemedActnCtrls.TThemedButtonControl.DrawText 4561
ThemedActnCtrls.TThemedCustomizePopup 4561
ThemedActnCtrls.TThemedCustomizePopup.Create 4562
ThemedActnCtrls.TThemedCustomizePopup.GetAddRemoveItemClass 4562
ThemedActnCtrls.TThemedCustomizePopup.GetDefaultColorMapClass 4563
ThemedActnCtrls.TThemedCustomizePopup.NCPaint 4563
ThemedActnCtrls.TThemedDropDownButton 4563
ThemedActnCtrls.TThemedDropDownButton.DrawBackground 4564
ThemedActnCtrls.TThemedDropDownButton.DrawFrame 4564
ThemedActnCtrls.TThemedDropDownButton.DrawGlyph 4565
ThemedActnCtrls.TThemedDropDownButton.DrawLargeGlyph

4565
ThemedActnCtrls.TThemedDropDownButton.GetDropDownButtonWidth 4565
ThemedActnCtrls.TThemedDropDownButton.GetPopupClass 4565
ThemedActnCtrls.TThemedExpandBtn 4566
ThemedActnCtrls.TThemedExpandBtn.DrawBackground 4566
ThemedActnCtrls.TThemedExpandBtn.DrawFrame 4567
ThemedActnCtrls.TThemedExpandBtn.GetImageSize 4567
ThemedActnCtrls.TThemedMenuButton 4567
ThemedActnCtrls.TThemedMenuButton.CalcBounds 4568
ThemedActnCtrls.TThemedMenuButton.DrawBackground 4568
ThemedActnCtrls.TThemedMenuButton.DrawText 4569
ThemedActnCtrls.TThemedMenuItem 4569
ThemedActnCtrls.TThemedMenuItem.CalcBounds 4570
ThemedActnCtrls.TThemedMenuItem.DrawBackground 4570
ThemedActnCtrls.TThemedMenuItem.DrawGlyph 4571
ThemedActnCtrls.TThemedMenuItem.DrawLargeGlyph 4571
ThemedActnCtrls.TThemedMenuItem.DrawSeparator 4571
ThemedActnCtrls.TThemedMenuItem.DrawShadowedText 4572
ThemedActnCtrls.TThemedMenuItem.DrawSubMenuGlyph 4572
ThemedActnCtrls.TThemedMenuItem.DrawText 4572
ThemedActnCtrls.TThemedMenuItem.DrawUnusedEdges 4572
ThemedActnCtrls.TThemedMenuItem.GetImageSize 4573
ThemedActnCtrls.TThemedPopupMenu 4573
ThemedActnCtrls.TThemedPopupMenu.Create 4573
ThemedActnCtrls.TThemedPopupMenu.GetExpandBtnClass 4574
ThemedActnCtrls.TThemedPopupMenu.NCPaint 4574
ThemedActnCtrls.TThemedPopupMenu.PositionPopup 4574
ThemedActnCtrls.TThemedPopupMenu.VisibleChanging 4575
ThemedActnCtrls.TThemedToolScrollBtn 4575
ThemedActnCtrls.TThemedToolScrollBtn.DrawBackground 4576
Themes 4577
Themes namespace 4577
Classes 4579
Functions 4593
Structs, Records, Enums 4595
Types 4629
Variables 4631
Themes.PThemedElementDetails 4629
Themes.PThemedElementDetails type 4629
Themes.ThemeControl 4593
Themes.ThemeControl function 4593
Themes.ThemeServices 4593
Themes.ThemeServices function 4593
Themes.ThemeServicesClass 4631
Themes.ThemeServicesClass variable 4631
Themes.TThemeData 4629
Themes.TThemeData type 4629
Themes.TThemedButton 4595
Themes.TThemedButton enumeration 4595
Themes.TThemedClock 4598
Themes.TThemedClock enumeration 4598
Themes.TThemedComboBox 4598
Themes.TThemedComboBox enumeration 4598
Themes.TThemedEdit 4599
Themes.TThemedEdit enumeration 4599
Themes.TThemedElement 4600
Themes.TThemedElement enumeration 4600
Themes.TThemedElementDetails 4602
Themes.TThemedElementDetails record 4602
Themes.TThemedExplorerBar 4603
Themes.TThemedExplorerBar enumeration 4603
Themes.TThemedHeader 4604
Themes.TThemedHeader enumeration 4604
Themes.TThemedListview 4605
Themes.TThemedListview enumeration 4605
Themes.TThemedMenu 4606
Themes.TThemedMenu enumeration 4606
Themes.TThemedPage 4607
Themes.TThemedPage enumeration 4607
Themes.TThemedProgress 4608
Themes.TThemedProgress enumeration 4608
Themes.TThemedRebar 4609
Themes.TThemedRebar enumeration 4609
Themes.TThemedScrollBar 4610

Themes.TThemedScrollBar enumeration 4610
Themes.TThemedSpin 4612
Themes.TThemedSpin enumeration 4612
Themes.TThemedStartPanel 4613
Themes.TThemedStartPanel enumeration 4613
Themes.TThemedStatus 4614
Themes.TThemedStatus enumeration 4614
Themes.TThemedTab 4615
Themes.TThemedTab enumeration 4615
Themes.TThemedTaskBand 4616
Themes.TThemedTaskBand enumeration 4616
Themes.TThemedTaskBar 4617
Themes.TThemedTaskBar enumeration 4617
Themes.TThemedToolBar 4618
Themes.TThemedToolBar enumeration 4618
Themes.TThemedToolTip 4620
Themes.TThemedToolTip enumeration 4620
Themes.TThemedTrackBar 4621
Themes.TThemedTrackBar enumeration 4621
Themes.TThemedTrayNotify 4622
Themes.TThemedTrayNotify enumeration 4622
Themes.TThemedTreeview 4623
Themes.TThemedTreeview enumeration 4623
Themes.TThemedWindow 4624
Themes.TThemedWindow enumeration 4624
Themes.TThemeServices 4579
Themes.TThemeServices.ApplyThemeChange 4582
Themes.TThemeServices.ColorToRGB 4582
Themes.TThemeServices.ContentRect 4583
Themes.TThemeServices.Create 4581
Themes.TThemeServices.Destroy 4581
Themes.TThemeServices.DoOnThemeChange 4583
Themes.TThemeServices.DrawEdge 4584
Themes.TThemeServices.DrawElement 4585
Themes.TThemeServices.DrawIcon 4585
Themes.TThemeServices.DrawParentBackground 4586
Themes.TThemeServices.DrawText 4586
Themes.TThemeServices.GetElementDetails 4588
Themes.TThemeServices.HasTransparentParts 4589
Themes.TThemeServices.OnThemeChange 4592
Themes.TThemeServices.PaintBorder 4590
Themes.TThemeServices.Theme 4591
Themes.TThemeServices.ThemesAvailable 4591
Themes.TThemeServices.ThemesEnabled 4592
Themes.TThemeServices.UnloadThemeData 4590
Themes.TThemeServices.UpdateThemes 4590
Themes.TThemeServicesClass 4630
Themes.TThemeServicesClass type 4630
Themes.UnthemedDesigner 4594
Themes.UnthemedDesigner function 4594
ThemesAvailable
    TThemeServices class 4591
ThemesEnabled
    TThemeServices class 4592
thHeaderDontCare enumeration member 4604
thHeaderItemHot enumeration member 4604
thHeaderItemLeftHot enumeration member 4604
thHeaderItemLeftNormal enumeration member 4604
thHeaderItemLeftPressed enumeration member 4604
thHeaderItemNormal enumeration member 4604
thHeaderItemPressed enumeration member 4604
thHeaderItemRightHot enumeration member 4604
thHeaderItemRightNormal enumeration member 4604
thHeaderItemRightPressed enumeration member 4604
thHeaderRoot enumeration member 4604
thHeaderSortArrowSortedDown enumeration member 4604
thHeaderSortArrowSortedUp enumeration member 4604
THintAction class 1882
    about THintAction class 1882
    Create 1883
    Hint 1884
    THintAction Members 1883
    THintAction properties 1884
ThreadCacheSize
    TCustomServerSocket class 50
    TServerSocket class 113
    TServerSocketThread class 1438
    TServerWinSocket class 132

ThreadDesc
TSocketForm class 170
ThreadEnd
TServerWinSocket class 129
ThreadException
TStreamedConnection class 244
ThreadGroup
TSocketForm class 171
ThreadPool
TServerSocketThread class 1438
ThreadReceivedStream
TStreamedConnection class 244
ThreadSize
TSocketForm class 171
ThreadStart
TServerWinSocket class 129
ThreadUpDown
TSocketForm class 171
ThrowExcept
ECustomException class 1310
ThrowIfOutOfRange
AnsiStringBase class 3200
UnicodeString class 3410
ThrowOuterException
Exception class 3963
THTMLItem class 6662
about THTMLItem class 6662
AssignString 6664
Create 6664
HTML 6664
Ident 6665
ItemLength 6665
TagToken 6665
THTMLItem Members 6663
THTMLItem methods 6664
THTMLItem properties 6664
THTMLItems class 6665
about THTMLItems class 6665
Create 6666
Destroy 6666
GetHTMLItem 6667
Items 6667
MakeIdentifier 6667
THTMLItems Members 6666
THTMLItems methods 6667
THTMLItems properties 6667
THTTPReqResp class 1276
_AddRef 1279
_Release 1280
about THTTPReqResp class 1276
AfterConstruction 1280
Agent 1288
BeforeExecute 1280
Check 1281
CheckContentType 1282
Connect 1282
ConnectTimeout 1288
Create 1278
Destroy 1278
Execute 1283
Execute method 1282
Get 1283
GetHTTPReqResp 1284
GetMimeBoundary 1284
HandleWinInetError 1285
InvokeOptions 1288
MaxSinglePostSize 1289
NewInstance 1285
OnBeforePost 1294
OnPostingData 1294
OnReceivingData 1295
OnWinInetError 1295
Password 1289
Proxy 1289
ProxyByPass 1290
Receive 1285
ReceiveTimeout 1290
Send 1286

SendGet 1286
SendTimeout 1291
SetMimeBoundary 1287
SoapAction 1291
THTTPReqResp events 1294
THTTPReqResp Members 1277
THTTPReqResp methods 1279
THTTPReqResp properties 1287
UDDIBindingKey 1291
UDDIOperator 1292
URL 1292
UserName 1293
UseUTF8InHeader 1293
WSDLView 1293
THTTPRIO class 1248
about THTTPRIO class 1248
Converter 1252
Create 1249
Destroy 1250
HTTPWebNode 1252
Notification 1250
Port 1253
QueryInterface 1251
Service 1253
THTTPRIO Members 1249
THTTPRIO methods 1250
THTTPRIO properties 1252
URL 1254
WSDLItems 1255
WSDLLocation 1255
THTTPSoapCppInvoker class 1266
about THTTPSoapCppInvoker class 1266
THTTPSoapDispatcher class 5946
about THTTPSoapDispatcher class 5946
Create 5947
Destroy 5948
DispatchEnabled 5948
DispatchMask 5949
DispatchMethodType 5949
DispatchRequest 5949
THTTPSoapDispatcher Members 5947
THTTPSoapDispatcher methods 5948
THTTPSoapDispatcher properties 5950
WebDispatch 5950
THTTPSoapDispatchNode class 1258
about THTTPSoapDispatchNode class 1258
Dispatcher 1261
DispatchSOAP 1259
FSoapDispatcher 1259
Notification 1260
THTTPSoapDispatchNode fields 1259
THTTPSoapDispatchNode Members 1258
THTTPSoapDispatchNode methods 1259
THTTPSoapDispatchNode properties 1260
THTTPSoapPascalInvoker class 1267
about THTTPSoapPascalInvoker class 1267
DispatchSOAP 1269
THTTPSoapPascalInvoker Members 1268
THTTPSoapPascalInvoker methods 1269
TImageMargins class 2602
about TImageMargins class 2602
Assign 2604
Bottom 2605
Change 2604
Left 2605
OnChange 2607
Right 2606
TImageMargins events 2607
TImageMargins Members 2603
TImageMargins methods 2603
TImageMargins properties 2605
Top 2606
Time enumeration member 3327
Timeout
TSocketForm class 171
TimeOut
TWinSocketStream class 143
TimeoutDesc

TSocketForm class 171
TimeoutGroup
TSocketForm class 172
TimeoutMinutes
IWebSessionAttributes interface 1107
TSessionItem class 383
TWebSessionImpl class 6771
TimeoutUpDown
TSocketForm class 172
TimeString
TDateTime class 3333
TImplementedAdapterAction class 5818
about TImplementedAdapterAction class 5818
TImport class 6958
about TImport class 6958
GetLocation 6964
GetNamespace 6964
SetLocation 6964
SetNamespace 6964
TImport Members 6959
TImport methods 6963
TImports class 6965
about TImports class 6965
Add 6970
GetImport 6971
Insert 6971
TImports Members 6965
TImports methods 6970
TInterfacedObject class 3342
_AddRef 3345
_Release 3345
about TInterfacedObject class 3342
AfterConstruction 3345
BeforeDestruction 3346
FRefCount 3344
NewInstance 3346
QueryInterface 3347
RefCount 3347
TInterfacedObject data members 3344
TInterfacedObject Members 3343
TInterfacedObject methods 3344
TInterfacedObject properties 3347
TInternalSQLDataSet class 888
about TInternalSQLDataSet class 888
CommandText 891
CommandType 892
DataSource 893
GetMetadata 894
MaxBlobSize 894
ParamCheck 895
Params 895
SortFieldNames 896
TInternalSQLDataSet Members 888
TInternalSQLDataSet properties 891
TInternetExplorer class 421
about TInternetExplorer class 421
AddressBar 449
Application 449
Busy 449
ClientToWindow 428
Connect 428
ConnectTo 428
Container 450
Create 425
DefaultInterface 450
Destroy 425
Disconnect 429
Document 450
ExecWB 429, 430
ExecWB method 429
FullName 450
FullScreen 450
Get_AddressBar 430
Get_Application 430
Get_Busy 430
Get_Container 431
Get_Document 431
Get_FullName 431

Get_FullScreen 431
Get_Height 431
Get_HWnd 432
Get_Left 432
Get_LocationName 432
Get_LocationURL 432
Get_MenuBar 433
Get_Name 433
Get_Offline 433
Get_Parent 433
Get_Path 433
Get_ReadyState 434
Get_RegisterAsBrowser 434
Get_RegisterAsDropTarget 434
Get_Resizable 434
Get_Silent 435
Get_StatusBar 435
Get_StatusText 435
Get_TheaterMode 435
Get_ToolBar 435
Get_Top 436
Get_TopLevelContainer 436
Get_type_ 436
Get_Visible 436
Get_Width 437
GetProperty 437
GoBack 437
GoForward 437
GoHome 437
GoSearch 438
Height 451
HWnd 451
InitServerData 438
InvokeEvent 438
Left 451
LocationName 451
LocationURL 452
MenuBar 452
Name 452
Navigate 439, 440
Navigate method 439
Navigate2 441, 442
Navigate2 method 440
Offline 454
OnBeforeNavigate2 459
OnClientToHostWindow 459
OnCommandStateChange 459
OnDocumentComplete 459
OnDownloadBegin 459
OnDownloadComplete 460
OnFileDownload 460
OnFullScreen 460
OnMenuBar 460
OnNavigateComplete2 461
OnNavigateError 461
OnNewWindow2 461
OnPrintTemplateInstantiation 461
OnPrintTemplateTeardown 461
OnPrivacyImpactedStateChange 462
OnProgressChange 462
OnPropertyChange 462
OnQuit 462
OnSetSecureLockIcon 463
OnStatusBar 463
OnStatusTextChange 463
OnTheaterMode 463
OnTitleChange 463
OnToolBar 464
OnUpdatePageStatus 464
OnVisible 464
OnWindowClosing 464
OnWindowSetHeight 465
OnWindowSetLeft 465
OnWindowSetResizable 465
OnWindowSetTop 465
OnWindowSetWidth 465
Parent 454
Path 455

PutProperty 442
QueryStatusWB 442
Quit 442
ReadyState 455
Refresh 442
Refresh2 443
Refresh2 method 443
RegisterAsBrowser 455
RegisterAsDropTarget 455
Resizable 455
Set_AddressBar 443
Set_FullScreen 444
Set_Height 444
Set_Left 444
Set_MenuBar 444
Set_Offline 444
Set_RegisterAsBrowser 445
Set_RegisterAsDropTarget 445
Set_Resizable 445
Set_Silent 445
Set_StatusBar 446
Set_StatusText 446
Set_TheaterMode 446
Set_ToolBar 446
Set_Top 446
Set_Visible 447
Set_Width 447
ShowBrowserBar 447, 448
ShowBrowserBar method 447
Silent 456
StatusBar 456
StatusText 456
Stop 448
TheaterMode 456
TInternetExplorer events 458
TInternetExplorer Members 422
TInternetExplorer methods 426
TInternetExplorer properties 448
ToolBar 457
Top 457
TopLevelContainer 457
type_ 457
Visible 457
Width 458
TInvalidWebSession class 6216
about TInvalidWebSession class 6216
Create 6217
GetSessionID 6217
GetTimeoutMinutes 6218
GetValue 6218
SetTimeoutMinutes 6219
SetValue 6219
Terminate 6219
TInvalidWebSession Members 6216
TInvalidWebSession methods 6217
UpdateResponse 6220
TInvokeableVariantType class 5014
about TInvokeableVariantType class 5014
DispInvoke 5018
DoFunction 5018
DoProcedure 5019
FixupIdent 5020
GetProperty 5020
SetProperty 5021
TInvokeableVariantType Members 5015
TInvokeableVariantType methods 5017
TipBorderStyle
TScreenTipsWindow class 359
TIpSocket class 1416
about TIpSocket class 1416
Bind 1419
Create 1418
Domain 1423
GetSocketAddr 1419
LocalDomainName 1419
LocalHost 1423
LocalHostAddr 1420
LocalHostName 1420

LocalPort 1424
LookupHostAddr 1420
LookupHostName 1421
LookupPort 1421
LookupProtocol 1421
ReceiveFrom 1422
RemoteHost 1426
RemotePort 1428
SendTo 1422
TIpSocket Members 1417
TIpSocket methods 1418
TIpSocket properties 1423
TItemProp class 4778
about TItemProp class 4778
AssignTo 4780
Create 4779
Destroy 4779
EditMask 4781
EditStyle 4782
HasPickList 4781
KeyDesc 4783
MaxLength 4783
PickList 4783
ReadOnly 4784
TItemProp Members 4778
TItemProp methods 4780
TItemProp properties 4781
UpdateEdit 4781
Title
ISearch interface 646
ISearchDisp interface 686
TServiceApplication class 3095
TWebApplication class 5940
TitleCaptions
TValueListEditor class 4835
TitleCaptionsStored
TValueListEditor class 4800
TitleChange
DWebBrowserEvents interface 625
DWebBrowserEvents2 interface 633
tkArray enumeration member 4760
tkChar enumeration member 4760
tkClass enumeration member 4760
tkDynArray enumeration member 4760
tkEnumeration enumeration member 4760
tkFloat enumeration member 4760
tkInt64 enumeration member 4760
tkInteger enumeration member 4760
tkInterface enumeration member 4760
tkLString enumeration member 4760
tkMethod enumeration member 4760
tkRecord enumeration member 4760
tkSet enumeration member 4760
tkString enumeration member 4760
tkUnknown enumeration member 4760
tkUString enumeration member 4760
tkVariant enumeration member 4760
tkWChar enumeration member 4760
tkWString enumeration member 4760
TLabel class 2607
about TLabel class 2607
Align 2610
Alignment 2613
Anchors 2614
AutoSize 2614
BiDiMode 2615
Caption 2616
Color 2618
Constraints 2619
DragCursor 2620
DragKind 2620
DragMode 2621
EllipsisPosition 2622
Enabled 2623
FocusControl 2623
Font 2623
GlowSize 2624
Layout 2625

OnClick 2637
OnContextPopup 2639
OnDblClick 2640
OnDragDrop 2640
OnDragOver 2643
OnEndDock 2646
OnEndDrag 2647
OnMouseActivate 2648
OnMouseDown 2649
OnMouseEnter 2651
OnMouseLeave 2651
OnMouseMove 2652
OnMouseUp 2654
OnStartDock 2656
OnStartDrag 2657
ParentBiDiMode 2625
ParentColor 2626
ParentFont 2626
ParentShowHint 2627
PopupMenu 2627
ShowAccelChar 2632
ShowHint 2633
TLabel events 2637
TLabel Members 2608
TLabel properties 2609
Transparent 2635
Visible 2635
WordWrap 2636
TLanguages class 3987
about TLanguages class 3987
Count 3989
Create 3988
Ext 3989
ID 3990
IndexOf 3988
LocaleID 3990
Name 3991
NameFromLCID 3991
NameFromLocaleID 3991
TLanguages Members 3987
TLanguages methods 3988
TLanguages properties 3989
TLayoutAttributes class 6011
about TLayoutAttributes class 6011
ControlAttributes 6013
Create 6013
LabelAttributes 6013
LabelPosition 6014
TLayoutAttributes fields 6013
TLayoutAttributes Members 6012
TLayoutWebContent class 6014
_AddRef 6017
_Release 6017
about TLayoutWebContent class 6014
Create 6016
ImplLayoutButton 6017
ImplLayoutField 6018
ImplLayoutLabelAndField 6018
ImplLayoutTable 6019
LayoutButton 6020
LayoutField 6020
LayoutLabelAndField 6020
LayoutTable 6021
ParentLayout 6022
QueryInterface 6021
TLayoutWebContent Members 6015
TLayoutWebContent methods 6016
TLayoutWebContent properties 6022
tlBottom enumeration member 3007
tlbsCRLF enumeration member 3737
tlbsLF enumeration member 3737
tlCenter enumeration member 3007
tlEmptyText enumeration member 4605
TLineNumberMap class 6668
about TLineNumberMap class 6668
Count 6670
DestLineToSourceLine 6670
Items 6671

New 6670
TLineNumberMap Members 6669
TLineNumberMap methods 6670
TLineNumberMap properties 6670
TLinkedRIO class 1316
about TLinkedRIO class 1316
Create 1317
CreateFile 1318
Destroy 1318
GetResponseStream 1319
QueryInterface 1319
TLinkedRIO Members 1317
TLinkedRIO methods 1319
TLinkedRIO properties 1320
WebNode 1320
TLinkedWebNode class 1321
about TLinkedWebNode class 1321
BeforeExecute 1323
Create 1322
Destroy 1322
Execute 1324
Execute method 1323
GetMimeBoundary 1324
GetResponseStream 1325
Invoker 1326
MimeBoundary 1326
SetMimeBoundary 1325
TLinkedWebNode Members 1321
TLinkedWebNode methods 1322
TLinkedWebNode properties 1325
TListBox class 2657
about TListBox class 2657
Align 2662
Anchors 2665
AutoComplete 2666
AutoCompleteDelay 2666
BevelEdges 2666
BevelInner 2667
BevelKind 2667
BevelOuter 2668
BevelWidth 2668
BiDiMode 2668
BorderStyle 2669
Color 2669
Columns 2670
Constraints 2671
Ctl3D 2671
DoubleBuffered 2672
DragCursor 2674
DragKind 2674
DragMode 2674
Enabled 2676
ExtendedSelect 2676
Font 2677
ImeMode 2678
ImeName 2678
IntegralHeight 2679
ItemHeight 2680
Items 2680
MultiSelect 2683
OnClick 2699
OnContextPopup 2701
OnData 2701
OnDataFind 2702
OnDataObject 2702
OnDblClick 2703
OnDragDrop 2703
OnDragOver 2706
OnDrawItem 2709
OnEndDock 2711
OnEndDrag 2712
OnEnter 2713
OnExit 2715
OnKeyDown 2719
OnKeyPress 2723
OnKeyUp 2724
OnMeasureItem 2725
OnMouseActivate 2727

OnMouseDown 2728
OnMouseEnter 2730
OnMouseLeave 2731
OnMouseMove 2731
OnMouseUp 2733
OnStartDock 2735
OnStartDrag 2736
ParentBiDiMode 2684
ParentColor 2684
ParentCtl3D 2685
ParentDoubleBuffered 2685
ParentFont 2685
ParentShowHint 2686
PopupMenu 2686
ScrollWidth 2691
ShowHint 2692
Sorted 2693
Style 2695
TabOrder 2695
TabStop 2696
TabWidth 2696
TListBox events 2698
TListBox Members 2658
TListBox properties 2661
Visible 2697
tlListDetail enumeration member 4605
tlListGroup enumeration member 4605
tlListItemDisabled enumeration member 4605
tlListItemHot enumeration member 4605
tlListItemNormal enumeration member 4605
tlListItemSelected enumeration member 4605
tlListItemSelectedNotFocus enumeration member 4605
tlListSortDetail enumeration member 4605
tlListviewDontCare enumeration member 4605
tlListviewRoot enumeration member 4605
TLocalConnection class 4546
about TLocalConnection class 4546
AppServer 4553
AS_ApplyUpdates 4549
AS_DataRequest 4549
AS_Execute 4549
AS_GetParams 4549
AS_GetProviderNames 4550
AS_GetRecords 4550
AS_RowRequest 4550
Create 4547
Destroy 4548
GetConnected 4550
GetProvider 4551
GetProviderNames 4551
GetServer 4551
InterfaceSupportsErrorInfo 4552
Notification 4552
ProviderCount 4554
Providers 4554
RegisterProvider 4552
SafeCallException 4553
TLocalConnection Members 4547
TLocalConnection methods 4548
TLocalConnection properties 4553
UnRegisterProvider 4553
TLocateFileService class 6220
about TLocateFileService class 6220
OnFindIncludeFile 6221
OnFindStream 6222
OnFindTemplateFile 6222
TLocateFileService events 6221
TLocateFileService Members 6221
TLogicalPageList class 5819
about TLogicalPageList class 5819
Create 5824
TLogicalPageList Members 5819
TLoginFormAdapter class 5824
about TLoginFormAdapter class 5824
Actions 5830
Data 5831
OnAfterExecuteAction 5832
OnAfterGetActionResponse 5832

OnBeforeExecuteAction 5833
OnBeforeGetActionResponse 5833
OnGetActionParams 5833
OnLogin 5834
OnValidatePassword 5834
OnValidateUserName 5835
PasswordRequired 5831
TLoginFormAdapter events 5832
TLoginFormAdapter Members 5826
TLoginFormAdapter properties 5830
TLoginFormAdapterLoginAction class 5835
about TLoginFormAdapterLoginAction class 5835
DisplayLabel 5842
GetDefaultActionName 5840
ImplExecuteActionRequest 5840
ImplGetAdapterItemRequestParams 5841
NextPage 5841
OnAfterExecute 5842
OnAfterGetResponse 5843
OnBeforeExecute 5843
OnBeforeGetResponse 5844
OnGetParams 5844
TLoginFormAdapterLoginAction events 5842
TLoginFormAdapterLoginAction Members 5836
TLoginFormAdapterLoginAction methods 5840
TLoginFormAdapterLoginAction properties 5842
TLogLinkedWebNode class 1326
about TLogLinkedWebNode class 1326
Create 1327
Destroy 1328
Execute 1328
GetResponseStream 1328
ReqFile 1329
RespFile 1329
TLogLinkedWebNode Members 1327
TLogLinkedWebNode methods 1328
TLogLinkedWebNode properties 1329
tlTop enumeration member 3007
TMBCSEncoding class 3992
about TMBCSEncoding class 3992
Create 3993, 3994
Create constructor 3993
GetByteCount 3994
GetBytes 3995
GetCharCount 3995
GetChars 3996
GetMaxByteCount 3996
GetMaxCharCount 3996
GetPreamble 3997
TMBCSEncoding Members 3992
TMBCSEncoding methods 3994
tmChevron enumeration member 4606
TMemo class 2737
about TMemo class 2737
Align 2741
Alignment 2745
Anchors 2745
BevelEdges 2746
BevelInner 2746
BevelKind 2747
BevelOuter 2747
BiDiMode 2747
BorderStyle 2748
CharCase 2748
Color 2749
Constraints 2749
Ctl3D 2750
DoubleBuffered 2750
DragCursor 2752
DragKind 2753
DragMode 2753
Enabled 2755
Font 2755
HideSelection 2756
ImeMode 2757
ImeName 2757
Lines 2758
MaxLength 2759

OEMConvert 2759
OnChange 2777
OnClick 2778
OnContextPopup 2780
OnDblClick 2780
OnDragDrop 2781
OnDragOver 2784
OnEndDock 2787
OnEndDrag 2787
OnEnter 2788
OnExit 2791
OnKeyDown 2795
OnKeyPress 2798
OnKeyUp 2799
OnMouseActivate 2801
OnMouseDown 2802
OnMouseEnter 2804
OnMouseLeave 2805
OnMouseMove 2805
OnMouseUp 2807
OnStartDock 2809
OnStartDrag 2810
ParentBiDiMode 2760
ParentColor 2760
ParentCtl3D 2761
ParentDoubleBuffered 2761
ParentFont 2761
ParentShowHint 2762
PopupMenu 2762
ReadOnly 2767
ScrollBars 2768
ShowHint 2770
TabOrder 2771
TabStop 2772
TMemo events 2777
TMemo Members 2737
TMemo properties 2740
Visible 2772
WantReturns 2774
WantTabs 2774
WordWrap 2775
TMessage class 6971
about TMessage class 6971
AfterConstruction 6977
GetName 6977
GetParts 6977
SetName 6977
TMessage Members 6972
TMessage methods 6976
TMessages class 6978
about TMessages class 6978
Add 6984
GetMessage 6984
Insert 6984
TMessages Members 6978
TMessages methods 6983
TMetaClass class 3348
about TMetaClass class 3348
tmMenuBarDropDown enumeration member 4606
tmMenuBarItem enumeration member 4606
tmMenuDontCare enumeration member 4606
tmMenuDropDown enumeration member 4606
tmMenuItemDemoted enumeration member 4606
tmMenuItemNormal enumeration member 4606
tmMenuItemSelected enumeration member 4606
tmMenuRoot enumeration member 4606
TMonitor record 3349
about TMonitor record 3349
Enter 3350, 3351
Enter method 3350
Exit 3351
Pulse 3352
PulseAll 3352
SetSpinCount 3353
TMonitor Members 3350
TMonitor methods 3350
TryEnter 3353
Wait 3353

tmSeparator enumeration member 4606
TMultiModuleWebAppServices class 6223
about TMultiModuleWebAppServices class 6223
Access_Request 6225
Access_Response 6225
CreateContext 6225
DoAfterDispatch 6226
DoBeforeDispatch 6226
DoneModule 6227
FinishContext 6227
GetExceptionHandler 6228
HandleException 6228
HandleRequest 6229
ImplFinishContext 6229
ImplHandleException 6230
ImplHandleRequest 6230
ImplInitContext 6231
InitContext 6231
InitModule 6232
InitModules 6233
ModuleAdded 6233
OnAfterDispatch 6235
OnBeforeDispatch 6235
Request 6234
Response 6234
TMultiModuleWebAppServices events 6235
TMultiModuleWebAppServices Members 6223
TMultiModuleWebAppServices methods 6224
TMultiModuleWebAppServices properties 6233
WebModules 6234
TMultiReadExclusiveWriteSynchronizer class 3997
about TMultiReadExclusiveWriteSynchronizer class 3997
BeginRead 4000
BeginWrite 4000
Create 3999
Destroy 3999
EndRead 4001
EndWrite 4001
RevisionLevel 4002
TMultiReadExclusiveWriteSynchronizer Members 3998
TMultiReadExclusiveWriteSynchronizer methods 4000
TMultiReadExclusiveWriteSynchronizer properties 4002
TMutex class 3143
about TMutex class 3143
Acquire 3146
Create 3145
Create constructor 3145
Release 3146
TMutex Members 3143
TMutex methods 3146
TNamedVariantsList class 6134
about TNamedVariantsList class 6134
Clear 6136
Delete 6136
Destroy 6135
Get 6137
GetCapacity 6137
GetCount 6138
Insert 6138
Put 6139
SetCapacity 6139
TNamedVariantsList Members 6134
TNamedVariantsList methods 6136
TNotifyList class 968
about TNotifyList class 968
AddNotify 970
Count 972
Create 969
Destroy 970
GetCount 971
GetItem 971
Items 972
RemoveNotify 971
TNotifyList Members 968
TNotifyList methods 970
TNotifyList properties 972
TObject class 3354
about TObject class 3354

AfterConstruction 3357
BeforeDestruction 3358
ClassInfo 3358
ClassName 3359
ClassNameIs 3361
ClassParent 3361
ClassType 3364
CleanupInstance 3367
Create 3355
DefaultHandler 3367
Destroy 3356
Dispatch 3369
Equals 3371
FieldAddress 3372
FieldAddress method 3371
Free 3372
FreeInstance 3374
GetHashCode 3375
GetInterface 3375
GetInterfaceEntry 3376
GetInterfaceTable 3376
InheritsFrom 3376
InitInstance 3377
InstanceSize 3377
MethodAddress 3378
MethodAddress method 3378
MethodName 3379
NewInstance 3379
SafeCallException 3380
TObject Members 3355
TObject methods 3356
ToString 3380
UnitName 3380
ToDouble
AnsiStringBase class 3200
UnicodeString class 3410
WideString class 3485
Toggle
TCustomCheckBox class 2280
ToInt
AnsiStringBase class 3200
UnicodeString class 3410
WideString class 3485
ToIntDef
AnsiStringBase class 3201
UnicodeString class 3411
WideString class 3485
ToolBar
IWebBrowser2Disp interface 774
IWebBrowserApp interface 785
IWebBrowserAppDisp interface 794
TInternetExplorer class 457
TWebBrowser class 543
ToolWin 4633
ToolWin namespace 4633
Classes 4634
Structs, Records, Enums 4644
Types 4647
ToolWin.TEdgeBorder 4644
ToolWin.TEdgeBorder enumeration 4644
ToolWin.TEdgeBorders 4647
ToolWin.TEdgeBorders type 4647
ToolWin.TEdgeStyle 4645
ToolWin.TEdgeStyle enumeration 4645
ToolWin.TSizingOrientation 4645
ToolWin.TSizingOrientation enumeration 4645
ToolWin.TToolDockForm 4634
ToolWin.TToolDockForm.CanResize 4635
ToolWin.TToolDockForm.Create 4635
ToolWin.TToolDockForm.CreateParams 4636
ToolWin.TToolDockForm.DoAddDockClient 4636
ToolWin.TToolDockForm.WMNCCreate 4636
ToolWin.TToolDockForm.WMNCHitTest 4636
ToolWin.TToolDockForm.WMNCLButtonDown 4637
ToolWin.TToolDockForm.WMSize 4637
ToolWin.TToolDockForm.WMSysCommand 4637
ToolWin.TToolDockObject 4637
ToolWin.TToolDockObject.AdjustDockRect 4638

ToolWin.TToolDockObject.Create 4638
ToolWin.TToolDockObject.DrawDragDockImage 4639
ToolWin.TToolDockObject.EraseDragDockImage 4639
ToolWin.TToolWindow 4639
ToolWin.TToolWindow.CMBorderChanged 4641
ToolWin.TToolWindow.CMCtl3DChanged 4641
ToolWin.TToolWindow.Create 4640
ToolWin.TToolWindow.EdgeBorders 4642
ToolWin.TToolWindow.EdgeInner 4642
ToolWin.TToolWindow.EdgeOuter 4643
ToolWin.TToolWindow.NCPaint 4641
ToolWin.TToolWindow.WMNCCalcSize 4641
ToolWin.TToolWindow.WMNCPaint 4642
Top
IWebBrowser interface 748
IWebBrowser2Disp interface 775
IWebBrowserAppDisp interface 794
IWebBrowserDisp interface 800
TImageMargins class 2606
TInternetExplorer class 457
TOperation class 6984
about TOperation class 6984
AfterConstruction 6990
GetFaults 6990
GetInput 6990
GetName 6991
GetOutput 6991
SetName 6991
TOperation Members 6985
TOperation methods 6989
TOperations class 6991
about TOperations class 6991
Add 6997, 6998
Add method 6997
GetOperation 6998
Insert 6998
TOperations Members 6992
TOperations methods 6997
TopFont
TTabbedNotebook class 4433
TopIndex
TCustomListBox class 2437
TopLevelContainer
IWebBrowser interface 748
IWebBrowser2Disp interface 775
IWebBrowserAppDisp interface 794
IWebBrowserDisp interface 801
TInternetExplorer class 457
TWebBrowser class 543
TWebBrowser_V1 class 595
ToString
Exception class 3963
TObject class 3380
TStringBuilder class 4021
TouchSession
TSessions class 394
TouchTime
IWebSessionAttributes interface 1108
TSessionItem class 384
TWebSessionImpl class 6771
TPacketInterceptFactory class 219
about TPacketInterceptFactory class 219
TPacketInterceptFactory Members 220
TPacketInterceptFactory methods 220
UpdateRegistry 221
TPagedAdapter class 5844
about TPagedAdapter class 5844
Actions 5850
Data 5850
ModifyAccess 5851
OnAfterExecuteAction 5852
OnAfterGetActionResponse 5853
OnAfterUpdateFields 5853
OnAfterValidateFields 5853
OnBeforeExecuteAction 5854
OnBeforeGetActionResponse 5854
OnBeforeUpdateFields 5854
OnBeforeValidateFields 5855

OnGetActionParams 5855
OnGetEOF 5856
OnGetFirstRecord 5856
OnGetNextRecord 5856
OnGetRecordCount 5856
OnGetRecordIndex 5857
PageSize 5851
TPagedAdapter events 5852
TPagedAdapter Members 5845
TPagedAdapter properties 5850
ViewAccess 5851
TPageDispatcher class 6236
about TPageDispatcher class 6236
DefaultPage 6238
OnAfterDispatchPage 6239
OnBeforeDispatchPage 6239
OnCanViewPage 6240
OnIsLoginRequired 6241
OnPageAccessDenied 6241
OnPageNotFound 6242
TPageDispatcher events 6238
TPageDispatcher Members 6236
TPageDispatcher properties 6238
TParam class 6998
about TParam class 6998
GetMessage 7004
GetName 7004
SetMessage 7004
SetName 7004
TParam Members 6999
TParam methods 7003
TPart class 7005
about TPart class 7005
GetElement 7010
GetName 7010
GetType_ 7011
SetElement 7011
SetName 7011
SetType_ 7011
TPart Members 7005
TPart methods 7010
TParts class 7012
about TParts class 7012
Add 7017
GetPart 7018
Insert 7018
TParts Members 7012
TParts methods 7017
tpBar enumeration member 4608
tpBarVert enumeration member 4608
tpChunk enumeration member 4608
tpChunkVert enumeration member 4608
tpDownDisabled enumeration member 4607
tpDownHorzDisabled enumeration member 4607
tpDownHorzHot enumeration member 4607
tpDownHorzNormal enumeration member 4607
tpDownHorzPressed enumeration member 4607
tpDownHot enumeration member 4607
tpDownNormal enumeration member 4607
tpDownPressed enumeration member 4607
TPort class 7018
about TPort class 7018
GetBinding 7024
GetName 7024
SetBinding 7024
SetName 7024
TPort Members 7019
TPort methods 7023
TPorts class 7025
about TPorts class 7025
Add 7030
GetPort 7031
Insert 7031
TPorts Members 7025
TPorts methods 7030
TPortType class 7031
about TPortType class 7031
AfterConstruction 7037

GetName 7037
GetOperations 7037
SetName 7037
TPortType Members 7032
TPortType methods 7036
TPortTypes class 7038
about TPortTypes class 7038
Add 7044
GetPortType 7044
Insert 7044
TPortTypes Members 7038
TPortTypes methods 7043
tpPageDontCare enumeration member 4607
tpPageRoot enumeration member 4607
tpProgressDontCare enumeration member 4608
tpProgressRoot enumeration member 4608
TPrintDlg class 1884
about TPrintDlg class 1884
BeforeExecute 1891
Caption 1886
Dialog 1887
Enabled 1887
GetDialogClass 1886
HelpContext 1888
HelpKeyword 1888
HelpType 1889
Hint 1889
ImageIndex 1889
OnAccept 1891
OnCancel 1892
OnHint 1892
OnUpdate 1893
SecondaryShortCuts 1890
ShortCut 1890
TPrintDlg events 1891
TPrintDlg Members 1885
TPrintDlg methods 1886
TPrintDlg properties 1886
Visible 1890
TPublishableVariantType class 4694
about TPublishableVariantType class 4694
GetInstance 4698
GetProperty 4698
SetProperty 4699
TPublishableVariantType Members 4695
TPublishableVariantType methods 4698
tpUpDisabled enumeration member 4607
tpUpHorzDisabled enumeration member 4607
tpUpHorzHot enumeration member 4607
tpUpHorzNormal enumeration member 4607
tpUpHorzPressed enumeration member 4607
tpUpHot enumeration member 4607
tpUpNormal enumeration member 4607
tpUpPressed enumeration member 4607
TPushButtonActionLink class 2811
about TPushButtonActionLink class 2811
IsImageIndexLinked 2812
SetImageIndex 2812
TPushButtonActionLink Members 2811
TPushButtonActionLink methods 2811
Trace
TSQLMonitor class 1648
TraceCallbackEvent
TSQLConnection class 1628
TraceCount
TSQLMonitor class 1651
TraceList
TSQLMonitor class 1652
TRadioButton class 2813
about TRadioButton class 2813
Action 2818
Align 2820
Alignment 2823
Anchors 2823
BiDiMode 2824
Caption 2824
Checked 2827
Color 2827

Constraints 2828
Create 2815
CreateParams 2816
CreateWnd 2816
Ctl3D 2829
DoubleBuffered 2829
DragCursor 2831
DragKind 2831
DragMode 2832
Enabled 2833
Font 2834
GetChecked 2817
GetControlsAlignment 2817
OnClick 2848
OnContextPopup 2849
OnDblClick 2850
OnDragDrop 2850
OnDragOver 2853
OnEndDock 2857
OnEndDrag 2857
OnEnter 2858
OnExit 2861
OnKeyDown 2865
OnKeyPress 2868
OnKeyUp 2869
OnMouseActivate 2871
OnMouseDown 2872
OnMouseEnter 2874
OnMouseLeave 2874
OnMouseMove 2875
OnMouseUp 2877
OnStartDock 2879
OnStartDrag 2880
ParentBiDiMode 2835
ParentColor 2835
ParentCtl3D 2836
ParentDoubleBuffered 2836
ParentFont 2837
ParentShowHint 2837
PopupMenu 2838
SetChecked 2817
ShowHint 2843
TabOrder 2844
TabStop 2845
TRadioButton events 2847
TRadioButton Members 2813
TRadioButton methods 2815
TRadioButton properties 2818
Visible 2845
WordWrap 2847
TransactionLevel
TCustomSQLDataSet class 1567
TransactionsSupported
TSQLConnection class 1624
transform
IDOMXSLProcessor interface 7610
Transform
TXMLTransform class 7729
TransformApplyUpdates
TXMLTransformClient class 7746
TransformationDocument
TXMLTransform class 7735
TransformationFile
TXMLTransform class 7736
TransformGetData
TXMLTransformClient class 7746
transformNode
IDOMNodeEx interface 7579, 7580
TransformNode
IXMLNode interface 7687
TXMLNode class 7437
TransformRead
TXMLTransformProvider class 7752
TransformSetParams
TXMLTransformClient class 7747
TransformWrite
TXMLTransformProvider class 7753
TransformXML

TXMLTransform class 7730
Translate
TXMLTransform class 7730
TranslateException class 7726
about TranslateException class 7726
Transparent
TCustomLabel class 2397
TCustomScreenTipsPopup class 324
TCustomStaticText class 2463
TLabel class 2635
TScreenTipsPopup class 349
TStaticText class 2975
Transport_GetConnected
TWebConnection class 261
Transport_SetConnected
TWebConnection class 261
TransportNode
TWSDLClientNode class 7270
TRawSocket class 1431
about TRawSocket class 1431
Create 1432
SockType 1433
TRawSocket Members 1431
TRawSocket properties 1433
trBandChecked enumeration member 4609
trBandDisabled enumeration member 4609
trBandHot enumeration member 4609
trBandHotChecked enumeration member 4609
trBandNormal enumeration member 4609
trBandPressed enumeration member 4609
trChevronDisabled enumeration member 4609
trChevronHot enumeration member 4609
trChevronNormal enumeration member 4609
trChevronPressed enumeration member 4609
trChevronVertDisabled enumeration member 4609
trChevronVertHot enumeration member 4609
trChevronVertNormal enumeration member 4609
trChevronVertPressed enumeration member 4609
trGripper enumeration member 4609
trGripperVert enumeration member 4609
Trim
AnsiStringBase class 3201
AnsiStringT template 3229
UnicodeString class 3411
WideString class 3486
TrimLeft
AnsiStringBase class 3201
AnsiStringT template 3229
UnicodeString class 3411
WideString class 3486
TrimRight
AnsiStringBase class 3202
AnsiStringT template 3229
UnicodeString class 3412
WideString class 3486
trRebarDontCare enumeration member 4609
trRebarRoot enumeration member 4609
TryEnter
TCriticalSection class 3130
TCriticalSectionHelper class helper 3133
TMonitor record 3353
tsArrowBtnDownDisabled enumeration member 4610
tsArrowBtnDownHot enumeration member 4610
tsArrowBtnDownNormal enumeration member 4610
tsArrowBtnDownPressed enumeration member 4610
tsArrowBtnLeftDisabled enumeration member 4610
tsArrowBtnLeftHot enumeration member 4610
tsArrowBtnLeftNormal enumeration member 4610
tsArrowBtnLeftPressed enumeration member 4610
tsArrowBtnRightDisabled enumeration member 4610
tsArrowBtnRightHot enumeration member 4610
tsArrowBtnRightNormal enumeration member 4610
tsArrowBtnRightPressed enumeration member 4610
tsArrowBtnUpDisabled enumeration member 4610
tsArrowBtnUpHot enumeration member 4610
tsArrowBtnUpNormal enumeration member 4610
tsArrowBtnUpPressed enumeration member 4610
TScreenTipBorderStyle class 326

about TScreenTipBorderStyle class 326
Color 327
CornerSize 327
Create 326
Style 327
TScreenTipBorderStyle Members 326
TScreenTipBorderStyle properties 327
TScreenTipItem class 328
about TScreenTipItem class 328
Action 332
Assign 330
Create 329
Description 332
Destroy 329
DisabledDescription 332
DisabledHeader 332
GetDisplayName 330
Header 332
Height 333
Image 333
IsEnabled 331
Manager 333
SetManager 331
ShowFooter 333
ShowImage 334
TScreenTipItem Members 328
TScreenTipItem methods 329
TScreenTipItem properties 331
Width 334
TScreenTipsCollection class 334
about TScreenTipsCollection class 334
Add 335
Create 335
GetOwner 336
Insert 336
Items 337
TScreenTipsCollection Members 334
TScreenTipsCollection methods 335
TScreenTipsCollection properties 337
TScreenTipsManager class 337
about TScreenTipsManager class 337
BorderStyle 339
DisplayOffset 339
Font 339
Footer 340
FooterImage 340
GradientEndColor 340
GradientStartColor 340
LinkedActionLists 340
OnCreateScreenTip 341
OnDuplicateCreateError 342
OnGetScreenTipWinClass 342
OnScreenTipsLoaded 342
OnShowScreenTip 342
Options 341
ScreenTips 341
TScreenTipsManager events 341
TScreenTipsManager Members 338
TScreenTipsManager properties 339
TScreenTipsPopup class 343
about TScreenTipsPopup class 343
Align 344
Anchors 347
Associate 348
Glyph 348
PopupType 348
ScreenTip 349
ScreenTipManager 349
Transparent 349
TScreenTipsPopup Members 343
TScreenTipsPopup properties 344
Visible 349
TScreenTipsWindow class 351
about TScreenTipsWindow class 351
ActivateToolTip 353
ActiveItem 357
Body 358
CMMouseenter 353

CMMouseLeave 354
CornerSize 358
Create 352
CreateParams 354
CreateRoundedStyle 354
CreateTipBorderStyle 355
CreateWnd 355
Destroy 352
DrawBackground 355
DrawBorder 356
DrawFooter 356
DrawHeader 356
Footer 358
FooterImage 358
GradientEndColor 359
GradientStartColor 359
HideHint 356
HintControl 359
Image 359
Paint 356
TipBorderStyle 359
TScreenTipsWindow Members 351
TScreenTipsWindow methods 353
TScreenTipsWindow properties 357
WMNCHitTest 357
TScriptCodeItem class 6671
about TScriptCodeItem class 6671
AssignString 6672
GetScript 6673
TScriptCodeItem Members 6671
TScriptCodeItem methods 6672
TScriptComponent class 973
about TScriptComponent class 973
Create 975
GetComponent 975
Obj 975
TScriptComponent Members 973
TScriptComponent methods 975
TScriptComponent properties 975
TScriptEnginesList class 6673
about TScriptEnginesList class 6673
AddEngine 6675
AddObjectFactory 6675
Count 6676
Create 6674
Destroy 6674
Find 6675
FindScriptProducerClass 6676
Items 6677
ListLanguageNames 6676
RemoveEngine 6676
TScriptEnginesList Members 6673
TScriptEnginesList methods 6675
TScriptEnginesList properties 6676
TScriptError class 6677
about TScriptError class 6677
Create 6678
FCharPos 6678
FDescription 6678
FFileName 6679
FLine 6679
FSourceLine 6679
GetCharPos 6679
GetDescription 6680
GetFileName 6680
GetLine 6680
GetSourceLine 6680
SetDescription 6681
TScriptError fields 6678
TScriptError Members 6677
TScriptError methods 6679
TScriptErrors class 6681
about TScriptErrors class 6681
Add 6683
Create 6681
Destroy 6682
GetError 6683
GetErrorCount 6683

TScriptErrors Members 6681
TScriptErrors methods 6682
TScriptFile class 6683
about TScriptFile class 6683
ApplicationName 6686
Create 6684
Destroy 6684
FLineNumberMap 6685
LongName 6686
ParseStream 6685
ParseString 6685
Script 6686
ShortName 6687
Source 6687
TScriptFile fields 6685
TScriptFile Members 6684
TScriptFile methods 6685
TScriptFile properties 6686
TScriptGlobalObjects class 6687
about TScriptGlobalObjects class 6687
AddNamedIntf 6690
Create 6688
Destroy 6689
FindNamedItemIntf 6690
GetNamedItemCount 6690
GetNamedItemName 6690
GetScriptEngine 6690
NamedItemCount 6691
NamedItemName 6691
Producer 6691
ScriptEngine 6692
TScriptGlobalObjects Members 6687
TScriptGlobalObjects methods 6689
TScriptGlobalObjects properties 6691
TScriptIncludeFile class 6692
about TScriptIncludeFile class 6692
Create 6693
LoadFile 6693
TScriptIncludeFile Members 6692
TScriptIncludeFile methods 6693
TScriptIncludeFiles class 6693
about TScriptIncludeFiles class 6693
Create 6694
GetScriptIncludeFile 6694
Items 6695
TScriptIncludeFiles Members 6694
TScriptIncludeFiles methods 6694
TScriptIncludeFiles properties 6695
TScriptIncludeItem class 6696
about TScriptIncludeItem class 6696
Create 6697
GetScript 6697
TScriptIncludeItem Members 6696
TScriptIncludeItem methods 6697
TScriptItem class 6698
about TScriptItem class 6698
Create 6699
GetScript 6699
Script 6700
TScriptItem Members 6698
TScriptItem methods 6699
TScriptItem properties 6699
TScriptItems class 6700
about TScriptItems class 6700
Create 6701
GetScriptItem 6701
Items 6701
TScriptItems Members 6700
TScriptItems methods 6701
TScriptItems properties 6701
TScriptObject class 976
about TScriptObject class 976
TScriptObjectFactories class 6702
about TScriptObjectFactories class 6702
Count 6706
Create 6704
CreateGlobalObject 6705
Destroy 6704

Factories 6706
GetCount 6705
GetFactory 6705
GetObjectList 6706
TScriptObjectFactories Members 6703
TScriptObjectFactories methods 6705
TScriptObjectFactories properties 6706
TScriptProducer class 6707
about TScriptProducer class 6707
Component 6714
Content 6715
Create 6708
Designing 6715
Destroy 6708
Errors 6715
Evaluate 6710
EvaluateExpression 6710
FErrors 6709
FScriptEngine 6709
FScriptFile 6709
GetContent 6710
GetErrors 6711
GetHTMLBlock 6711
GetHTMLBlockCount 6711
GetWebModuleContext 6711
HandleParserError 6712
HandleScriptError 6712
ImplHandleScriptError 6712
ParseStream 6712
ParseString 6712
ReplaceTags 6713
Script 6715
SetContent 6713
TScriptProducer fields 6709
TScriptProducer Members 6707
TScriptProducer methods 6710
TScriptProducer properties 6714
Write 6713, 6714
Write method 6713
WriteItem 6714
TScrollBar class 2881
about TScrollBar class 2881
Align 2886
Anchors 2890
BiDiMode 2890
Change 2883
Constraints 2891
Create 2883
CreateParams 2884
CreateWnd 2884
Ctl3D 2892
DoubleBuffered 2892
DragCursor 2894
DragKind 2894
DragMode 2895
Enabled 2896
Kind 2897
LargeChange 2898
Max 2900
Min 2901
OnChange 2920
OnContextPopup 2920
OnDragDrop 2921
OnDragOver 2924
OnEndDock 2927
OnEndDrag 2928
OnEnter 2929
OnExit 2931
OnKeyDown 2935
OnKeyPress 2939
OnKeyUp 2940
OnMouseEnter 2942
OnMouseLeave 2942
OnScroll 2942
OnStartDock 2944
OnStartDrag 2944
PageSize 2903
ParentBiDiMode 2905

ParentCtl3D 2905
ParentDoubleBuffered 2906
ParentShowHint 2906
PopupMenu 2907
Position 2912
Scroll 2885
SetParams 2885
ShowHint 2914
SmallChange 2915
TabOrder 2917
TabStop 2918
TScrollBar events 2920
TScrollBar Members 2881
TScrollBar methods 2883
TScrollBar properties 2886
Visible 2918
TScroller class 4458
about TScroller class 4458
Change 4461
Create 4459
Destroy 4460
Max 4461
Min 4462
OnClick 4463
Paint 4460
Position 4462
ScrollOrientation 4462
TScroller events 4463
TScroller Members 4458
TScroller methods 4460
TScroller properties 4460
tsDownDisabled enumeration member 4612
tsDownHorzDisabled enumeration member 4612
tsDownHorzHot enumeration member 4612
tsDownHorzNormal enumeration member 4612
tsDownHorzPressed enumeration member 4612
tsDownHot enumeration member 4612
tsDownNormal enumeration member 4612
tsDownPressed enumeration member 4612
TSearchAction class 1894
about TSearchAction class 1894
Create 1895
Destroy 1896
ExecuteTarget 1897
FControl 1896
FFindFirst 1897
HandlesTarget 1898
Notification 1898
Search 1898
TSearchAction fields 1896
TSearchAction Members 1895
TSearchAction methods 1897
UpdateTarget 1899
TSearchFind class 1899
about TSearchFind class 1899
BeforeExecute 1906
Caption 1902
Dialog 1902
Enabled 1903
GetDialogClass 1901
HelpContext 1903
HelpKeyword 1903
HelpType 1904
Hint 1904
ImageIndex 1905
OnAccept 1907
OnCancel 1907
OnHint 1908
OnUpdate 1908
SecondaryShortCuts 1905
ShortCut 1905
TSearchFind events 1906
TSearchFind Members 1900
TSearchFind methods 1901
TSearchFind properties 1902
Visible 1906
TSearchFindFirst class 1909
about TSearchFindFirst class 1909

Create 1911
TSearchFindFirst Members 1910
TSearchFindNext class 1912
about TSearchFindNext class 1912
Caption 1915
Create 1913
Enabled 1916
ExecuteTarget 1914
HandlesTarget 1914
HelpContext 1916
HelpKeyword 1917
HelpType 1917
Hint 1917
ImageIndex 1918
OnHint 1920
OnUpdate 1920
SearchFind 1918
SecondaryShortCuts 1919
ShortCut 1919
TSearchFindNext events 1920
TSearchFindNext Members 1912
TSearchFindNext methods 1913
TSearchFindNext properties 1915
UpdateTarget 1915
Visible 1919
TSearchReplace class 1921
about TSearchReplace class 1921
BeforeExecute 1929
Caption 1925
Dialog 1925
Enabled 1925
ExecuteTarget 1923
GetDialogClass 1924
HelpContext 1926
HelpKeyword 1926
HelpType 1927
Hint 1927
ImageIndex 1927
OnAccept 1930
OnCancel 1930
OnHint 1930
OnUpdate 1931
SecondaryShortCuts 1928
ShortCut 1928
TSearchReplace events 1929
TSearchReplace Members 1922
TSearchReplace methods 1923
TSearchReplace properties 1924
Visible 1929
TSemaphore class 3147
about TSemaphore class 3147
Acquire 3151
Create 3149
Create constructor 3149
Release 3151
Release method 3151
TSemaphore Members 3148
TSemaphore methods 3150
TServerAcceptThread class 90
about TServerAcceptThread class 90
Create 91
Execute 91
ServerSocket 95
TServerAcceptThread Members 91
TServerAcceptThread methods 91
TServerAcceptThread properties 95
TServerClientThread class 95
about TServerClientThread class 95
ClientExecute 98
ClientSocket 102
Create 97
Data 103
Destroy 98
DoTerminate 99
EndConnect 99
Error 100
Event 100
Execute 101

HandleException 101
KeepInCache 103
ReActivate 101
ServerSocket 104
StartConnect 102
TServerClientThread Members 96
TServerClientThread methods 98
TServerClientThread properties 102
TServerClientWinSocket class 104
about TServerClientWinSocket class 104
Create 106
Destroy 107
ServerWinSocket 107
TServerClientWinSocket Members 105
TServerClientWinSocket properties 107
TServerSocket class 108
about TServerSocket class 108
Active 111
Create 110
OnAccept 114
OnClientConnect 115
OnClientDisconnect 116
OnClientError 116
OnClientRead 117
OnClientWrite 117
OnGetSocket 118
OnGetThread 118
OnListen 119
OnThreadEnd 119
OnThreadStart 120
Port 111
ServerType 112
Service 112
Socket 113
ThreadCacheSize 113
TServerSocket events 114
TServerSocket Members 108
TServerSocket properties 110
TServerSocketThread class 1433
about TServerSocketThread class 1433
AddClientSocketThread 1436
ClearThreadPool 1436
Create 1434
CreateThread 1436
Destroy 1435
Execute 1436
FetchClientSocketThread 1437
OnGetThread 1439
RemoveClientSocketThread 1437
ServerSocket 1438
ThreadCacheSize 1438
ThreadPool 1438
TServerSocketThread events 1439
TServerSocketThread Members 1434
TServerSocketThread methods 1435
TServerSocketThread properties 1437
TServerWinSocket class 120
about TServerWinSocket class 120
Accept 124
ActiveConnections 129
ActiveThreads 130
ClientConnect 125
ClientDisconnect 125
ClientErrorEvent 125
ClientRead 125
ClientWrite 126
Connections 130
Create 123
Destroy 123
Disconnect 126
DoCreateThread 126
GetClientSocket 127
GetClientThread 127
GetServerThread 127
IdleThreads 131
Listen 127
OnClientConnect 133
OnClientDisconnect 134

OnClientError 135
OnClientRead 135
OnClientWrite 136
OnGetSocket 137
OnGetThread 137
OnThreadEnd 138
OnThreadStart 138
ServerType 131
SetServerType 128
SetThreadCacheSize 128
ThreadCacheSize 132
ThreadEnd 129
ThreadStart 129
TServerWinSocket events 132
TServerWinSocket Members 121
TServerWinSocket methods 124
TServerWinSocket properties 129
TService class 3071, 7044
about TService class 3071, 7044
AfterConstruction 7050
AfterInstall 3085
AfterUninstall 3086
AllowPause 3078
AllowStop 3078
BeforeInstall 3086
BeforeUninstall 3087
Controller 3074
CreateNew 3073
Dependencies 3078
Destroy 3073
DisplayName 3079
DoContinue 3074
DoCustomControl 3074
DoInterrogate 3074
DoPause 3075
DoShutdown 3075
DoStart 3075
DoStop 3075
ErrCode 3079
ErrorSeverity 3080
GetName 7050
GetPorts 7050
GetServiceController 3076
Interactive 3080
LoadGroup 3080
LogMessage 3076
Main 3077
OnContinue 3087
OnExecute 3088
OnPause 3088
OnShutdown 3088
OnStart 3089
OnStop 3089
Param 3081
ParamCount 3081
Password 3081
ReportStatus 3077
ServiceStartName 3082
ServiceThread 3082
ServiceType 3083
SetName 7051
StartType 3083
Status 3083
TagID 3084
Terminated 3084
TService events 3085
TService Members 3071, 7045
TService methods 3073, 7049
TService properties 3077
WaitHint 3084
Win32ErrCode 3085
TServiceApplication class 3089
about TServiceApplication class 3089
Create 3091
CreateForm 3092
DelayInitialize 3094
Destroy 3091
DispatchServiceMain 3092

DoHandleException 3092
Hook 3093
Initialize 3093
Installing 3093
RegisterServices 3093
Run 3094
ServiceCount 3094
Title 3095
TServiceApplication Members 3090
TServiceApplication methods 3091
TServiceApplication properties 3094
TServices class 7051
about TServices class 7051
Add 7057
GetService 7057
Insert 7057
TServices Members 7051
TServices methods 7056
TServiceThread class 3095
about TServiceThread class 3095
Create 3096
Execute 3097
ProcessRequests 3100
TServiceThread Members 3096
TServiceThread methods 3096
TSessionIDGen class 377
about TSessionIDGen class 377
Create 377
NextID 378
TSessionIDGen Members 377
TSessionIDGen methods 378
TSessionItem class 378
about TSessionItem class 378
Assign 380
Create 379
Destroy 379
ExpirationTime 381
Expired 381
ID 382
IsActive 380
Items 382
StartTime 382
StatusFlags 382
Terminated 383
TimeoutMinutes 383
TouchTime 384
TSessionItem Members 378
TSessionItem methods 380
TSessionItem properties 380
TSessions class 384
about TSessions class 384
ActiveCount 394
CheckStatusChange 386
ConnectActiveSession 387
ConnectSession 387
Create 385
CreateSessionIDGen 387
Destroy 385
DisconnectSession 387
FileName 395
GetExpirationTime 388
GetExpired 388
GetIDCount 388
GetIDs 388
GetIsActive 389
GetItemCount 389
GetItems 389
GetSession 389
GetSessionIDGen 390
GetStartTime 390
GetTerminated 390
GetTimeoutMinutes 390
GetTouchTime 391
ItemValue 391
LoadFromFile 391
LoadFromStream 391
MaxSessions 395
NotifyStatusChange 392

SaveToFile 392
SaveToStream 392
SessionExists 393
SetItemValue 393
SetTimeoutMinutes 393
StartSession 393
TerminateSession 394
TouchSession 394
TSessions Members 384
TSessions methods 386
TSessions properties 394
TSessionsService class 6756
about TSessionsService class 6756
DefaultTimeout 6758
MaxSessions 6758
OnEndSession 6759
OnStartSession 6759
TSessionsService events 6759
TSessionsService Members 6757
TSessionsService properties 6757
tsGripper enumeration member 4614
tsGripperHorzDisabled enumeration member 4610
tsGripperHorzHot enumeration member 4610
tsGripperHorzNormal enumeration member 4610
tsGripperHorzPressed enumeration member 4610
tsGripperPane enumeration member 4614
tsGripperVertDisabled enumeration member 4610
tsGripperVertHot enumeration member 4610
tsGripperVertNormal enumeration member 4610
tsGripperVertPressed enumeration member 4610
TShadowWindow class 400
about TShadowWindow class 400
Control 405
Create 401
CreateParams 402
CreateShadow 401
Destroy 401
Notification 403
Paint 403
SetBounds 404
Side 405
TShadowWindow Members 400
TShadowWindow methods 402
TShadowWindow properties 405
WMEraseBkgnd 404
TShellNameSpace class 466
about TShellNameSpace class 466
Columns 480
Connect 470
ConnectTo 470
CountViewTypes 480
Create 467
CreateSubscriptionForSelection 471
DefaultInterface 480
DeleteSubscriptionForSelection 471
Depth 480
Destroy 468
Disconnect 471
EnumOptions 481
Expand 472
Export 472
Flags 481
Get_Columns 472
Get_CountViewTypes 472
Get_Depth 472
Get_EnumOptions 473
Get_Flags 473
Get_Mode 473
Get_Root 473
Get_SelectedItem 474
Get_SubscriptionsEnabled 474
Get_TVFlags 474
Import 474
InitServerData 474
InvokeContextMenuCommand 475
InvokeEvent 475
Mode 481
MoveSelectionDown 475

MoveSelectionTo 476
MoveSelectionUp 476
NewFolder 476
OnDoubleClick 482
OnFavoritesSelectionChange 483
OnInitialized 483
OnSelectionChange 483
ResetSort 476
Root 481
SelectedItem 482
SelectedItems 477
Set_Columns 477
Set_Depth 477
Set_EnumOptions 477
Set_Flags 477
Set_Mode 478
Set_Root 478
Set_SelectedItem 478
Set_TVFlags 478
SetRoot 479
SetViewType 479
SubscriptionsEnabled 482
Synchronize 479
TShellNameSpace events 482
TShellNameSpace Members 466
TShellNameSpace methods 469
TShellNameSpace properties 479
TVFlags 482
UnselectAll 479
TShellResources class 886
about TShellResources class 886
TShellUIHelper class 483
about TShellUIHelper class 483
AddChannel 486
AddDesktopComponent 487, 488
AddDesktopComponent method 487
AddFavorite 488, 489
AddFavorite method 488
AutoCompleteAttach 489
AutoCompleteAttach method 489
AutoCompleteSaveForm 490
AutoCompleteSaveForm method 490
AutoScan 490, 491
AutoScan method 490
Connect 491
ConnectTo 491
Create 484
DefaultInterface 494
Destroy 485
Disconnect 492
ImportExportFavorites 492
InitServerData 492
IsSubscribed 493
NavigateAndFind 493
RefreshOfflineDesktop 493
ResetFirstBootMode 493
ResetSafeMode 494
ShowBrowserUI 494
TShellUIHelper Members 484
TShellUIHelper methods 486
TShellUIHelper properties 494
TShellWindows class 494
_NewEnum 498
about TShellWindows class 494
Connect 498
ConnectTo 498
Count 503
Create 495
DefaultInterface 503
Destroy 496
Disconnect 499
FindWindowSW 499
Get_Count 499
InitServerData 499
InvokeEvent 500
Item 500, 501
Item method 500
OnActivated 501

OnCreated 501
OnNavigate 501
OnWindowRegistered 503
OnWindowRevoked 504
ProcessAttachDetach 502
Register 502
RegisterPending 502
Revoke 502
TShellWindows events 503
TShellWindows Members 495
TShellWindows methods 497
TShellWindows properties 502
TSimpleDataSet class 896
about TSimpleDataSet class 896
Active 904
AfterApplyUpdates 931
AfterCancel 931
AfterClose 933
AfterDelete 935
AfterEdit 937
AfterExecute 938
AfterGetParams 939
AfterGetRecords 940
AfterInsert 940
AfterOpen 941
AfterPost 942
AfterRefresh 943
AfterRowRequest 943
AfterScroll 944
Aggregates 906
AggregatesActive 907
AllocConnection 900
AllocDataSet 900
AllocProvider 901
AutoCalcFields 907
BeforeApplyUpdates 944
BeforeCancel 945
BeforeClose 946
BeforeDelete 946
BeforeEdit 947
BeforeExecute 947
BeforeGetParams 948
BeforeGetRecords 949
BeforeInsert 950
BeforeOpen 952
BeforePost 952
BeforeRefresh 953
BeforeRowRequest 953
BeforeScroll 954
Connection 908
Constraints 908
Create 899
DataSet 909
Destroy 900
DisableStringTrim 909
FetchOnDemand 910
FieldDefs 911
FileName 914
Filter 915
Filtered 917
FilterOptions 917
IndexDefs 918
IndexFieldNames 919
IndexName 920
Loaded 901
MasterFields 921
MasterSource 921
Notification 901
ObjectView 922
OnCalcFields 955
OnDeleteError 955
OnEditError 956
OnFilterRecord 956
OnNewRecord 957
OnPostError 957
OnReconcileError 958
OpenCursor 902
PacketRecords 923

Params 924
PSGetCommandText 902
ReadOnly 929
SetConnection 903
StoreDefs 929
TSimpleDataSet events 930
TSimpleDataSet Members 897
TSimpleDataSet methods 900
TSimpleDataSet properties 903
TSimpleEvent class 3152
about TSimpleEvent class 3152
TSimpleRWSync class 4002
about TSimpleRWSync class 4002
BeginRead 4005
BeginWrite 4005
Create 4004
Destroy 4004
EndRead 4006
EndWrite 4006
TSimpleRWSync Members 4003
TSimpleRWSync methods 4005
TSiteModuleHelper class 6565
about TSiteModuleHelper class 6565
Create 6567
Destroy 6567
EndIterator 6568
FindNamedVariable 6568
FindVariable 6569
FindVariableContainer 6569
GetCurrentObject 6569
GetVariable 6569
GetVariableCount 6569
Module 6571
NextIteration 6570
Notification 6570
NotifyActivate 6570
NotifyDeactivate 6570
OnActivate 6573
OnDeactivate 6573
Request 6571
Response 6572
Session 6572
StartIterator 6571
TSiteModuleHelper events 6573
TSiteModuleHelper Members 6566
TSiteModuleHelper methods 6568
TSiteModuleHelper properties 6571
WebModuleContext 6572
TSitePageElements class 1167
about TSitePageElements class 1167
ServerFunctionsContent 1168
ServerScriptContent 1169
ServerVariablesContent 1169
StylesContent 1169
TSitePageElements fields 1168
TSitePageElements Members 1167
WarningsContent 1169
TSitePageModuleHelper class 6574
about TSitePageModuleHelper class 6574
Create 6576
DefaultAction 6583
Destroy 6577
DispatchPage 6577
GetDefaultAction 6578
GetPageProducer 6578
GetProducerComponent 6578
GetRedirectParams 6578
GetTemplateFileType 6579
GetTemplateStream 6579
GetXMLData 6579
GetXSLData 6579
HasHTMLBrowserView 6579
HasScriptView 6580
HasXMLBrowserView 6580
HasXSLBrowserView 6580
HTMLFromTemplate 6580
ImplDispatchPage 6581
ImplIncludePage 6581

ImplRedirectToPage 6581
ImplSetWebContentOptions 6581
IncludePage 6582
OnAfterDispatchPage 6583
OnAfterRedirectToPage 6584
OnBeforeDispatchPage 6584
OnBeforeRedirectToPage 6584
RedirectToPage 6582
SetDefaultPageFileName 6582
SetWebContentOptions 6582
TSitePageModuleHelper events 6583
TSitePageModuleHelper Members 6574
TSitePageModuleHelper methods 6577
TSitePageModuleHelper properties 6582
tsLowerTrackHorzDisabled enumeration member 4610
tsLowerTrackHorzHot enumeration member 4610
tsLowerTrackHorzNormal enumeration member 4610
tsLowerTrackHorzPressed enumeration member 4610
tsLowerTrackVertDisabled enumeration member 4610
tsLowerTrackVertHot enumeration member 4610
tsLowerTrackVertNormal enumeration member 4610
tsLowerTrackVertPressed enumeration member 4610
tsModernPopout enumeration member 4540
tsModernTabs enumeration member 4540
TSoapConnection class 1188
about TSoapConnection class 1188
AfterConnect 1201
AfterDisconnect 1201
Agent 1196
BeforeConnect 1202
BeforeDisconnect 1202
Connected 1196
Create 1190
Destroy 1191
DoConnect 1191
DoDisconnect 1192
GetConnected 1192
GetProviderNames 1192
GetServer 1193
GetServerList 1193
GetSOAPServer 1194
GetSOAPServerIID 1194
Notification 1194
OnAfterExecute 1203
OnBeforeExecute 1203
Password 1197
Proxy 1197
ProxyByPass 1197
RIO 1198
SetSOAPServerIID 1195
SetUseSoapAdapter 1195
SOAPServerIID 1198
TSoapConnection events 1201
TSoapConnection Members 1189
TSoapConnection methods 1191
TSoapConnection properties 1195
URL 1199
UserName 1200
UseSOAPAdapter 1200
TSoapDataList class 1172
about TSoapDataList class 1172
Add 1173
Assign 1173
Clear 1174
Insert 1174
Remove 1174
TSoapDataList Members 1172
TSoapDataList methods 1173
TSoapDataModule class 1206
_AddRef 1210
_Release 1210
about TSoapDataModule class 1206
AfterConstruction 1210
ApplyUpdates 1211
AS_ApplyUpdates 1211
AS_DataRequest 1212
AS_Execute 1213
AS_GetParams 1214

AS_GetProviderNames 1215
AS_GetRecords 1215
AS_RowRequest 1216
BeforeDestruction 1217
Create 1208
DataRequest 1217
Destroy 1208
Execute 1218
GetParams 1219
GetProvider 1219
GetProviderNames 1220
GetRecords 1220
Lock 1221
NewInstance 1221
ProviderCount 1230
Providers 1230
RefCount 1231
RegisterProvider 1222
RowRequest 1222
SafeCallException 1223
SAS_ApplyUpdates 1223
SAS_DataRequest 1224
SAS_Execute 1225
SAS_GetParams 1226
SAS_GetProviderNames 1226
SAS_GetRecords 1227
SAS_RowRequest 1228
TSoapDataModule Members 1207
TSoapDataModule methods 1209
TSoapDataModule properties 1230
Unlock 1229
UnRegisterProvider 1229
TSOAPDOMProcessor class 1234
_AddRef 1236
_Release 1236
about TSOAPDOMProcessor class 1234
AddHeaderProcessor 1237
AfterConstruction 1237
DefaultProcessHeader 1238
Destroy 1235
FHeaderProcessors 1235
FindHeaderProcessor 1238
MakeHeaderNodes 1239
NewInstance 1239
TSOAPDOMProcessor fields 1235
TSOAPDOMProcessor Members 1234
TSOAPDOMProcessor methods 1236
TSoapPascalInvoker class 1350
about TSoapPascalInvoker class 1350
AfterDispatchEvent 1356
BeforeDispatchEvent 1357
BeforeDispatchEvent2 1357
Converter 1355
Create 1351
Destroy 1352
FConverter 1352
FDomConverter 1352
FHeadersIn 1353
FHeadersOut 1353
FSOAPHeaders 1353
GetDomConverter 1354
GetInvoker 1354
Invoke 1354
Notification 1355
OnExceptionEvent 1357
OnGetAttachment 1358
OnSendAttachment 1358
TSoapPascalInvoker events 1356
TSoapPascalInvoker fields 1352
TSoapPascalInvoker Members 1350
TSoapPascalInvoker methods 1353
TSoapPascalInvoker properties 1355
TSocketConnection class 221
about TSocketConnection class 221
Address 225
Create 223
CreateTransport 224
DoConnect 224

Host 226
InterceptGUID 226
InterceptName 227
ObjectBroker 228
Port 228
SupportCallbacks 229
TSocketConnection Members 222
TSocketConnection methods 223
TSocketConnection properties 225
TSocketForm class 158
about TSocketForm class 158
ActionList1 161
AddClient 174
AddIcon 174
AllowXML 161
AllowXMLExecute 174
ApplyAction 162
ApplyActionExecute 174
ApplyActionUpdate 174
ApplyButton 162
ClearModifications 175
ConnectionList 162
ConnectionListColumnClick 175
ConnectionListCompare 175
Connections1 162
DisconnectAction 163
DisconnectActionUpdate 175
ExportedObjectOnly1 163
FormCloseQuery 176
FormCreate 176
FormDestroy 176
FormShow 176
GUIDDesc 163
HeaderControl1 163
Initialize 176
IntegerExit 177
InterceptGroup 163
InterceptGUID 164
ItemIndex 181
Label1 164
Label4 164
Label5 164
Label7 165
MainMenu1 165
miAdd 165
miAddClick 177
miClose 165
miCloseClick 177
miDisconnect 165
miDisconnectClick 177
miExit 166
miExitClick 178
miPorts 166
miProperties 166
miPropertiesClick 178
miRemove 166
miShowHostName 167
N1 167
N2 167
N3 167
Pages 167
Panel1 168
PopupMenu 168
PortDesc 168
PortGroup 168
PortList 169
PortListClick 178
PortNo 169
PortUpDown 169
PropPage 169
ReadSettings 178
RegisteredAction 169
RegisteredActionExecute 178
RemoveClient 179
RemovePortAction 170
RemovePortActionExecute 179
RemovePortActionUpdate 179
SelectedSocket 181

ShowHostAction 170
ShowHostActionExecute 179
StatPage 170
ThreadDesc 170
ThreadGroup 171
ThreadSize 171
ThreadUpDown 171
Timeout 171
TimeoutDesc 171
TimeoutGroup 172
TimeoutUpDown 172
TSocketForm fields 160
TSocketForm Members 158
TSocketForm methods 173
TSocketForm properties 181
UIInitialize 180
UpdateTimer 172
UpdateTimerTimer 180
UpDownClick 180
UserStatus 172
WMMIDASIcon 180
WndProc 180
WriteSettings 181
XMLPacket1 173
TSocketService class 182
about TSocketService class 182
CreateNew 183
GetServiceController 184
Start 184
Stop 185
TSocketService Members 182
TSocketService methods 184
TSocketTransport class 229
about TSocketTransport class 229
Address 234
Create 231
Destroy 231
GetConnected 232
GetWaitEvent 233
Host 235
InterceptGUID 235
Port 235
Receive 233
Send 234
SetConnected 234
Socket 235
TSocketTransport Members 230
TSocketTransport methods 232
TSocketTransport properties 234
tsOwnerDraw enumeration member 4540
tsPane enumeration member 4614
tspLogOff enumeration member 4613
tspLogOffButtonsHot enumeration member 4613
tspLogOffButtonsNormal enumeration member 4613
tspLogOffButtonsPressed enumeration member 4613
tspMorePrograms enumeration member 4613
tspMoreProgramsArrowHot enumeration member 4613
tspMoreProgramsArrowNormal enumeration member 4613
tspMoreProgramsArrowPressed enumeration member 4613
tspPlacesList enumeration member 4613
tspPlacesListSeparator enumeration member 4613
tspPreview enumeration member 4613
tspProgList enumeration member 4613
tspProgListSeparator enumeration member 4613
tspStartPanelDontCare enumeration member 4613
tspStartPanelRoot enumeration member 4613
tspUserPane enumeration member 4613
tspUserPicture enumeration member 4613
TSQLBlobStream class 1577
about TSQLBlobStream class 1577
Create 1578
Destroy 1579
ReadBlobData 1580
TSQLBlobStream Members 1578
TSQLBlobStream methods 1579
TSQLConnection class 1580
about TSQLConnection class 1580
ActiveStatements 1607

AddConnectNotification 1585
AfterConnect 1626
AfterDisconnect 1626
AutoClone 1608
BeforeConnect 1626
BeforeDisconnect 1627
BeginTransaction 1586
BeginTransaction method 1585
CheckConnection 1586
CheckDisconnect 1587
CloneConnection 1587
CloseDataSets 1587
Commit 1588
CommitFreeAndNil 1588
Connected 1609
ConnectionData 1609
ConnectionName 1610
ConnectionOptions 1589
ConnectionRegistryFile 1610
ConnectionState 1611
Create 1583
DataSets 1611
DBXConnection 1611
Destroy 1584
DoConnect 1589
DoDisconnect 1589
DriverName 1612
Execute 1590
ExecuteDirect 1590
GetCommandTypes 1591
GetConnected 1591
GetDataSet 1591
GetDefaultSchemaName 1592
GetDriverFunc 1613
GetFieldNames 1592
GetFieldNames method 1592
GetIndexNames 1593
GetIndexNames method 1593
GetLoginUsername 1594
GetPackageNames 1594
GetPackageNames method 1594
GetProcedureNames 1595
GetProcedureNames method 1595
GetProcedureParams 1597
GetProcedureParams method 1596
GetSchemaNames 1598
GetSchemaNames method 1598
GetServerMethodNames 1598
GetTableNames 1599
GetTableNames method 1598
HasTransaction 1599
InTransaction 1613
KeepConnection 1614
LibraryName 1614
Loaded 1600
LoadParamsFromIniFile 1600
LoadParamsOnConnect 1615
LocaleCode 1615
LoginPrompt 1616
MaxStmtsPerConn 1620
MetaData 1620
MultipleTransactionsSupported 1621
Notification 1601
OnLogin 1627
OpenSchema 1601, 1602
OpenSchema method 1601
Params 1621
ParamsLoaded 1622
QuoteChar 1623
RegisterClient 1602
RemoveConnectNotification 1602
Rollback 1603
RollbackFreeAndNil 1603
RollbackIncompleteFreeAndNil 1604
SetTraceEvent 1605
SQLDllHandle 1623
SQLError 1605
SQLHourGlass 1623

StartTransaction 1605
TableScope 1623
TraceCallbackEvent 1628
TransactionsSupported 1624
TSQLConnection events 1625
TSQLConnection Members 1581
TSQLConnection methods 1584
TSQLConnection properties 1607
UnRegisterClient 1606
VendorLib 1625
TSQLDataLink class 1628
about TSQLDataLink class 1628
ActiveChanged 1630
CheckBrowseMode 1630
Create 1629
GetDetailDataSet 1631
RecordChanged 1631
TSQLDataLink Members 1629
TSQLDataLink methods 1630
TSQLDataSet class 1631
about TSQLDataSet class 1631
CommandText 1637
CommandType 1638
Create 1635
DataSource 1639
DbxCommandType 1639
ExecSQL 1636
MaxBlobSize 1640
ParamCheck 1641
Params 1641
SortFieldNames 1642
SQLConnection 1642
TSQLDataSet Members 1633
TSQLDataSet methods 1636
TSQLDataSet properties 1637
TSQLMonitor class 1643
about TSQLMonitor class 1643
Active 1649
AutoSave 1650
Create 1645
Destroy 1645
FileName 1650
InvokeCallBack 1646
LoadFromFile 1646
MaxTraceCount 1651
OnLogTrace 1653
OnTrace 1653
SaveToFile 1646
SetActive 1647
SetFileName 1647
SetSQLConnection 1647
SetStreamedActive 1648
SetTraceList 1648
SQLConnection 1651
SwitchConnection 1648
Trace 1648
TraceCount 1651
TraceList 1652
TSQLMonitor events 1652
TSQLMonitor Members 1644
TSQLMonitor methods 1645
TSQLMonitor properties 1649
UpdateTraceCallBack 1648
TSQLQuery class 1654
about TSQLQuery class 1654
Create 1657
DataSource 1660
Destroy 1658
ExecSQL 1658
GetMetadata 1660
MaxBlobSize 1661
ParamCheck 1661
Params 1662
PrepareStatement 1659
RowsAffected 1663
SQL 1663
SQLConnection 1664
Text 1665

TSQLQuery Members 1654
TSQLQuery methods 1658
TSQLQuery properties 1659
TSqlServerMethod class 1665
about TSqlServerMethod class 1665
Create 1668
ExecuteMethod 1669
Params 1669
ServerMethodName 1670
SQLConnection 1670
TSqlServerMethod Members 1666
TSqlServerMethod methods 1669
TSqlServerMethod properties 1669
TSQLStoredProc class 1671
about TSQLStoredProc class 1671
Create 1674
ExecProc 1675
MaxBlobSize 1677
NextRecordSet 1676
PackageName 1677
ParamCheck 1678
Params 1678
PrepareStatement 1676
SQLConnection 1679
StoredProcName 1680
TSQLStoredProc Members 1671
TSQLStoredProc methods 1675
TSQLStoredProc properties 1676
TSQLTable class 1680
about TSQLTable class 1680
Active 1687
Create 1684
DeleteRecords 1685
Destroy 1685
GetIndexNames 1686
IndexFieldCount 1689
IndexFieldNames 1689
IndexFields 1690
IndexName 1690
MasterFields 1691
MasterLink 1692
MasterSource 1692
MaxBlobSize 1692
OpenCursor 1686
PrepareStatement 1686
SetIndexField 1687
SQLConnection 1693
TableName 1693
TSQLTable Members 1681
TSQLTable methods 1685
TSQLTable properties 1687
tsScrollBarDontCare enumeration member 4610
tsScrollBarRoot enumeration member 4610
tsSizeBoxLeftAlign enumeration member 4610
tsSizeBoxRightAlign enumeration member 4610
tsSoftTabs enumeration member 4540
tsSpinDontCare enumeration member 4612
tsSpinRoot enumeration member 4612
tsStandard enumeration member 4540
tsStatusDontCare enumeration member 4614
tsStatusRoot enumeration member 4614
tsSynonym enumeration member 1703
tsSysTable enumeration member 1703
tsTable enumeration member 1703
TStandardAddRemoveItem class 1742
about TStandardAddRemoveItem class 1742
DrawBackground 1743
DrawGlyph 1743
TStandardAddRemoveItem Members 1742
TStandardAddRemoveItem methods 1743
TStandardButtonControl class 1744
about TStandardButtonControl class 1744
DrawFrame 1744
DrawGlyph 1745
DrawText 1745
TStandardButtonControl Members 1744
TStandardButtonControl methods 1744
TStandardCustomizePopup class 1745

about TStandardCustomizePopup class 1745
CanAutoSize 1746
NCPaint 1746
PositionPopup 1747
TStandardCustomizePopup Members 1745
TStandardCustomizePopup methods 1746
TStandardDropDownButton class 1747
about TStandardDropDownButton class 1747
DrawFrame 1748
GetPopupClass 1748
TStandardDropDownButton Members 1747
TStandardDropDownButton methods 1747
TStandardMenuButton class 1748
about TStandardMenuButton class 1748
DrawBackground 1749
DrawDesignFocus 1749
DrawFrame 1750
DrawText 1750
TStandardMenuButton Members 1749
TStandardMenuButton methods 1749
TStandardMenuExpandBtn class 1750
about TStandardMenuExpandBtn class 1750
DrawBackground 1751
DrawFrame 1751
TStandardMenuExpandBtn Members 1750
TStandardMenuExpandBtn methods 1751
TStandardMenuItem class 1751
about TStandardMenuItem class 1751
CMTextchanged 1753
DrawBackground 1753
DrawDesignFocus 1753
DrawEdge 1754
DrawGlyph 1754
DrawGlyphFrame 1754
DrawLargeGlyph 1755
DrawSeparator 1755
DrawText 1755
TStandardMenuItem Members 1752
TStandardMenuItem methods 1752
TStandardMenuPopup class 1756
about TStandardMenuPopup class 1756
CanAutoSize 1757
GetExpandBtnClass 1757
NCPaint 1757
PositionPopup 1758
TStandardMenuPopup Members 1756
TStandardMenuPopup methods 1756
TStandardStyleActionBars class 3016
about TStandardStyleActionBars class 3016
GetAddRemoveItemClass 3017
GetColorMapClass 3017
GetControlClass 3017
GetPopupClass 3018
GetScrollBtnClass 3018
GetStyleName 3018
TStandardStyleActionBars Members 3016
TStandardStyleActionBars methods 3016
TStandardToolScrollBtn class 1758
about TStandardToolScrollBtn class 1758
DrawFrame 1759
TStandardToolScrollBtn Members 1758
TStandardToolScrollBtn methods 1758
TStaticText class 2945
about TStaticText class 2945
Align 2948
Alignment 2951
Anchors 2952
AutoSize 2952
BevelEdges 2953
BevelInner 2953
BevelKind 2954
BevelOuter 2954
BiDiMode 2954
BorderStyle 2955
Caption 2955
Color 2958
Constraints 2959
DoubleBuffered 2959

DragCursor 2961
DragKind 2961
DragMode 2962
Enabled 2963
FocusControl 2964
Font 2964
OnClick 2978
OnContextPopup 2980
OnDblClick 2980
OnDragDrop 2981
OnDragOver 2984
OnEndDock 2987
OnEndDrag 2987
OnMouseActivate 2988
OnMouseDown 2989
OnMouseEnter 2991
OnMouseLeave 2992
OnMouseMove 2992
OnMouseUp 2994
OnStartDock 2996
OnStartDrag 2997
ParentBiDiMode 2965
ParentColor 2966
ParentDoubleBuffered 2966
ParentFont 2966
ParentShowHint 2967
PopupMenu 2968
ShowAccelChar 2973
ShowHint 2973
TabOrder 2974
TabStop 2975
Transparent 2975
TStaticText events 2977
TStaticText Members 2946
TStaticText properties 2947
Visible 2976
tsThumbBtnHorzDisabled enumeration member 4610
tsThumbBtnHorzHot enumeration member 4610
tsThumbBtnHorzNormal enumeration member 4610
tsThumbBtnHorzPressed enumeration member 4610
tsThumbBtnVertDisabled enumeration member 4610
tsThumbBtnVertHot enumeration member 4610
tsThumbBtnVertNormal enumeration member 4610
tsThumbBtnVertPressed enumeration member 4610
TStreamedConnection class 236
_AddRef 239
_Release 239
about TStreamedConnection class 236
Create 237
CreateTransport 239
Destroy 238
DoConnect 240
DoDisconnect 240
DoError 241
GetInterceptorList 241
GetInterpreter 242
GetServerList 242
Handle 246
InterceptGUID 246
InterceptName 247
InternalClose 242
InternalOpen 243
Interpreter 247
QueryInterface 243
Send 244
SupportCallbacks 248
ThreadException 244
ThreadReceivedStream 244
TStreamedConnection Members 236
TStreamedConnection methods 238
TStreamedConnection properties 245
WndProc 245
TStringBuilder class 4006
about TStringBuilder class 4006
Append 4014
Append method 4013
AppendFormat 4015
AppendLine 4016

AppendLine method 4015
Capacity 4021
Chars 4022
CopyTo 4016
Create 4011
Create constructor 4010
EnsureCapacity 4016
Equals 4017
FData 4012
FLength 4012
FMaxCapacity 4012
Insert 4018
Insert method 4017
Length 4022
MaxCapacity 4022
Remove 4019
Replace 4020
Replace method 4019
ToString 4021
ToString method 4020
TStringBuilder fields 4011
TStringBuilder Members 4010
TStringBuilder methods 4012
TStringBuilder properties 4021
TStringsValuesList class 5857
about TStringsValuesList class 5857
Actions 5862
Data 5863
OnPrepareStrings 5864
Strings 5863
TStringsValuesList events 5863
TStringsValuesList Members 5858
TStringsValuesList properties 5862
tsUpDisabled enumeration member 4612
tsUpHorzDisabled enumeration member 4612
tsUpHorzHot enumeration member 4612
tsUpHorzNormal enumeration member 4612
tsUpHorzPressed enumeration member 4612
tsUpHot enumeration member 4612
tsUpNormal enumeration member 4612
tsUpperTrackHorzDisabled enumeration member 4610
tsUpperTrackHorzHot enumeration member 4610
tsUpperTrackHorzNormal enumeration member 4610
tsUpperTrackHorzPressed enumeration member 4610
tsUpperTrackVertDisabled enumeration member 4610
tsUpperTrackVertHot enumeration member 4610
tsUpperTrackVertNormal enumeration member 4610
tsUpperTrackVertPressed enumeration member 4610
tsUpPressed enumeration member 4612
tsView enumeration member 1703
TSynchroObject class 3152
about TSynchroObject class 3152
Acquire 3154
Release 3154
TSynchroObject Members 3153
TSynchroObject methods 3154
TTabbedNotebook class 4404
about TTabbedNotebook class 4404
ActivePage 4413
Align 4414
AlignControls 4407
Anchors 4417
Change 4407
Click 4407
CMTabFontChanged 4408
Constraints 4418
Create 4405
CreateHandle 4408
CreateParams 4409
Destroy 4406
DoubleBuffered 4418
Enabled 4420
Font 4420
GetChildOwner 4409
GetChildren 4409
GetIndexForPage 4410
KeyDown 4410
Loaded 4411

OnChange 4435
OnClick 4436
OnContextPopup 4436
OnEnter 4437
OnExit 4439
PageIndex 4421
Pages 4422
ParentDoubleBuffered 4422
ParentShowHint 4423
PopupMenu 4423
ReadState 4411
SetPageIndex 4412
ShowControl 4412
ShowHint 4428
TabFont 4430
TabFontChanged 4412
TabOrder 4430
Tabs 4431
TabsPerRow 4432
TabStop 4433
TopFont 4433
TTabbedNotebook events 4435
TTabbedNotebook Members 4404
TTabbedNotebook methods 4406
TTabbedNotebook properties 4413
Visible 4433
TTabList class 4463
about TTabList class 4463
Add 4464
AddStrings 4467
Clear 4468
Create 4464
Delete 4469
Insert 4469
Put 4470
TTabList Members 4464
TTabList methods 4464
TTabPage class 4443
about TTabPage class 4443
Caption 4445
Create 4444
Enabled 4448
Height 4449
ReadState 4445
TabOrder 4450
TTabPage Members 4444
TTabPage methods 4445
TTabPage properties 4445
Visible 4450
Width 4452
TTabSet class 4470
about TTabSet class 4470
Align 4481
Anchors 4485
AutoScroll 4485
BackgroundColor 4486
CanChange 4474
Canvas 4486
CMHintShow 4474
Constraints 4488
Create 4473
CreateParams 4475
Destroy 4473
DitherBackground 4489
DoubleBuffered 4489
DragCursor 4491
DragKind 4491
DragMode 4492
DrawTab 4475
Enabled 4493
EndMargin 4494
FirstIndex 4494
Font 4494
GetChildren 4475
GetImageIndex 4476
Images 4496
ItemAtPos 4476
ItemRect 4476

ItemWidth 4477
MeasureTab 4477
MinClientRect 4477
MinClientRect method 4477
MouseDown 4478
Notification 4479
OnAlignInsertBefore 4509
OnAlignPosition 4510
OnChange 4511
OnClick 4511
OnDragDrop 4513
OnDragOver 4516
OnDrawTab 4519
OnEndDock 4519
OnEndDrag 4520
OnEnter 4521
OnExit 4523
OnGetImageIndex 4527
OnMeasureTab 4527
OnMouseActivate 4528
OnMouseDown 4529
OnMouseEnter 4531
OnMouseLeave 4531
OnMouseMove 4532
OnMouseUp 4534
OnStartDock 4536
OnStartDrag 4537
Paint 4479
ParentBackground 4496
ParentDoubleBuffered 4496
ParentShowHint 4496
PopupMenu 4497
Scroller 4502
SelectedColor 4502
SelectNext 4480
SetTabIndex 4480
ShowHint 4502
ShrinkToFit 4504
SoftTop 4504
StartMargin 4504
Style 4505
TabHeight 4505
TabIndex 4506
TabPosition 4506
Tabs 4506
TTabSet events 4509
TTabSet Members 4471
TTabSet methods 4474
TTabSet properties 4480
UnselectedColor 4507
Visible 4507
VisibleTabs 4508
ttbButtonChecked enumeration member 4618
ttbButtonCheckedHot enumeration member 4618
ttbButtonDisabled enumeration member 4618
ttbButtonHot enumeration member 4618
ttbButtonNormal enumeration member 4618
ttbButtonPressed enumeration member 4618
ttbDropDownButtonChecked enumeration member 4618
ttbDropDownButtonCheckedHot enumeration member 4618
ttbDropDownButtonDisabled enumeration member 4618
ttbDropDownButtonHot enumeration member 4618
ttbDropDownButtonNormal enumeration member 4618
ttbDropDownButtonPressed enumeration member 4618
ttbFlashButton enumeration member 4616
ttbGroupCount enumeration member 4616
ttBody enumeration member 4615
ttBranch enumeration member 4623
ttbSeparatorChecked enumeration member 4618
ttbSeparatorCheckedHot enumeration member 4618
ttbSeparatorDisabled enumeration member 4618
ttbSeparatorHot enumeration member 4618
ttbSeparatorNormal enumeration member 4618
ttbSeparatorPressed enumeration member 4618
ttbSeparatorVertChecked enumeration member 4618
ttbSeparatorVertCheckedHot enumeration member 4618
ttbSeparatorVertDisabled enumeration member 4618
ttbSeparatorVertHot enumeration member 4618

ttbSeparatorVertNormal enumeration member 4618
ttbSeparatorVertPressed enumeration member 4618
ttbSplitButtonChecked enumeration member 4618
ttbSplitButtonCheckedHot enumeration member 4618
ttbSplitButtonDisabled enumeration member 4618
ttbSplitButtonDropDownChecked enumeration member 4618
ttbSplitButtonDropDownCheckedHot enumeration member 4618
ttbSplitButtonDropDownDisabled enumeration member 4618
ttbSplitButtonDropDownHot enumeration member 4618
ttbSplitButtonDropDownNormal enumeration member 4618
ttbSplitButtonDropDownPressed enumeration member 4618
ttbSplitButtonHot enumeration member 4618
ttbSplitButtonNormal enumeration member 4618
ttbSplitButtonPressed enumeration member 4618
ttbTaskBandDontCare enumeration member 4616
ttbTaskBandRoot enumeration member 4616
ttbThumbBottomDisabled enumeration member 4621
ttbThumbBottomFocused enumeration member 4621
ttbThumbBottomHot enumeration member 4621
ttbThumbBottomNormal enumeration member 4621
ttbThumbBottomPressed enumeration member 4621
ttbThumbDisabled enumeration member 4621
ttbThumbFocused enumeration member 4621
ttbThumbHot enumeration member 4621
ttbThumbLeftDisabled enumeration member 4621
ttbThumbLeftFocused enumeration member 4621
ttbThumbLeftHot enumeration member 4621
ttbThumbLeftNormal enumeration member 4621
ttbThumbLeftPressed enumeration member 4621
ttbThumbNormal enumeration member 4621
ttbThumbPressed enumeration member 4621
ttbThumbRightDisabled enumeration member 4621
ttbThumbRightFocused enumeration member 4621
ttbThumbRightHot enumeration member 4621
ttbThumbRightNormal enumeration member 4621
ttbThumbRightPressed enumeration member 4621
ttbThumbTics enumeration member 4621
ttbThumbTicsVert enumeration member 4621
ttbThumbTopDisabled enumeration member 4621
ttbThumbTopFocused enumeration member 4621
ttbThumbTopHot enumeration member 4621
ttbThumbTopNormal enumeration member 4621
ttbThumbTopPressed enumeration member 4621
ttbThumbVertDisabled enumeration member 4621
ttbThumbVertFocused enumeration member 4621
ttbThumbVertHot enumeration member 4621
ttbThumbVertNormal enumeration member 4621
ttbThumbVertPressed enumeration member 4621
ttbTimeNormal enumeration member 4617
ttbToolBarDontCare enumeration member 4618
ttbToolBarRoot enumeration member 4618
ttbTrack enumeration member 4621
ttbTrackBarDontCare enumeration member 4621
ttbTrackBarRoot enumeration member 4621
ttbTrackVert enumeration member 4621
TTcpClient class 1439
- about TTcpClient class 1439
- Active 1443
- BlockMode 1445
- Connected 1446
- OnConnect 1450
- OnCreateHandle 1451
- OnDestroyHandle 1453
- OnDisconnect 1456
- OnError 1456
- OnReceive 1456
- OnSend 1457
- RemoteHost 1446
- RemotePort 1448
- TTcpClient events 1450
- TTcpClient Members 1441
- TTcpClient properties 1443

TTcpServer class 1459
- about TTcpServer class 1459
- Active 1463
- BlockMode 1465
- LocalHost 1466
- LocalPort 1466

OnAccept 1469
OnCreateHandle 1472
OnDestroyHandle 1474
OnGetThread 1477
OnListening 1477
TTcpServer events 1469
TTcpServer Members 1461
TTcpServer properties 1463
ttGlyphClosed enumeration member 4623
ttGlyphOpened enumeration member 4623
TThemedAddRemoveItem class 4556
about TThemedAddRemoveItem class 4556
CalcBounds 4557
DrawBackground 4557
DrawGlyph 4558
DrawSeparator 4558
DrawText 4558
DrawUnusedEdges 4559
GetImageSize 4559
TThemedAddRemoveItem Members 4556
TThemedAddRemoveItem methods 4557
TThemedButtonControl class 4559
about TThemedButtonControl class 4559
DrawBackground 4560
DrawFrame 4560
DrawGlyph 4560
DrawLargeGlyph 4561
DrawText 4561
TThemedButtonControl Members 4559
TThemedButtonControl methods 4560
TThemedCustomizePopup class 4561
about TThemedCustomizePopup class 4561
Create 4562
GetAddRemoveItemClass 4562
GetDefaultColorMapClass 4563
NCPaint 4563
TThemedCustomizePopup Members 4562
TThemedCustomizePopup methods 4562
TThemedDropDownButton class 4563
about TThemedDropDownButton class 4563
DrawBackground 4564
DrawFrame 4564
DrawGlyph 4565
DrawLargeGlyph 4565
GetDropDownButtonWidth 4565
GetPopupClass 4565
TThemedDropDownButton Members 4563
TThemedDropDownButton methods 4564
TThemedExpandBtn class 4566
about TThemedExpandBtn class 4566
DrawBackground 4566
DrawFrame 4567
GetImageSize 4567
TThemedExpandBtn Members 4566
TThemedExpandBtn methods 4566
TThemedMenuButton class 4567
about TThemedMenuButton class 4567
CalcBounds 4568
DrawBackground 4568
DrawText 4569
TThemedMenuButton Members 4567
TThemedMenuButton methods 4568
TThemedMenuItem class 4569
about TThemedMenuItem class 4569
CalcBounds 4570
DrawBackground 4570
DrawGlyph 4571
DrawLargeGlyph 4571
DrawSeparator 4571
DrawShadowedText 4572
DrawSubMenuGlyph 4572
DrawText 4572
DrawUnusedEdges 4572
GetImageSize 4573
TThemedMenuItem Members 4569
TThemedMenuItem methods 4570
TThemedPopupMenu class 4573
about TThemedPopupMenu class 4573

Create 4573
GetExpandBtnClass 4574
NCPaint 4574
PositionPopup 4574
TThemedPopupMenu Members 4573
TThemedPopupMenu methods 4574
VisibleChanging 4575
TThemedToolScrollBtn class 4575
about TThemedToolScrollBtn class 4575
DrawBackground 4576
TThemedToolScrollBtn Members 4575
TThemedToolScrollBtn methods 4576
TThemeServices class 4579
about TThemeServices class 4579
ApplyThemeChange 4582
ColorToRGB 4582
ColorToRGB method 4582
ContentRect 4583
Create 4581
Destroy 4581
DoOnThemeChange 4583
DrawEdge 4584
DrawEdge method 4584
DrawElement 4585
DrawElement method 4584
DrawIcon 4585
DrawParentBackground 4586
DrawParentBackground method 4586
DrawText 4586
GetElementDetails 4588
GetElementDetails method 4587
HasTransparentParts 4589
OnThemeChange 4592
PaintBorder 4590
Theme 4591
ThemesAvailable 4591
ThemesEnabled 4592
TThemeServices events 4592
TThemeServices Members 4579
TThemeServices methods 4581
TThemeServices properties 4591
UnloadThemeData 4590
UpdateThemes 4590
TThreadLocalCounter class 4023
about TThreadLocalCounter class 4023
Close 4024
Delete 4024
Destroy 4023
Open 4024
TThreadLocalCounter Members 4023
TThreadLocalCounter methods 4024
TTimeZone record 1727
about TTimeZone record 1727
FInfo 1728
FUsesDayligtSavings 1728
GetTimeZone 1728
TTimeZone fields 1727
TTimeZone Members 1727
TTimeZone methods 1728
ttItemDisabled enumeration member 4623
ttItemHot enumeration member 4623
ttItemNormal enumeration member 4623
ttItemSelected enumeration member 4623
ttItemSelectedNotFocus enumeration member 4623
ttnAnimBackground enumeration member 4622
ttnBackground enumeration member 4622
ttnTrayNotifyDontCare enumeration member 4622
ttnTrayNotifyRoot enumeration member 4622
TToolDockForm class 4634
about TToolDockForm class 4634
CanResize 4635
Create 4635
CreateParams 4636
DoAddDockClient 4636
TToolDockForm Members 4634
TToolDockForm methods 4635
WMNCCreate 4636
WMNCHitTest 4636

WMNCLButtonDown 4637
WMSize 4637
WMSysCommand 4637
TToolDockObject class 4637
about TToolDockObject class 4637
AdjustDockRect 4638
Create 4638
DrawDragDockImage 4639
EraseDragDockImage 4639
TToolDockObject Members 4638
TToolDockObject methods 4638
TToolWindow class 4639
about TToolWindow class 4639
CMBorderChanged 4641
CMCtl3DChanged 4641
Create 4640
EdgeBorders 4642
EdgeInner 4642
EdgeOuter 4643
NCPaint 4641
TToolWindow Members 4640
TToolWindow methods 4641
TToolWindow properties 4642
WMNCCalcSize 4641
WMNCPaint 4642
ttPane enumeration member 4615
ttpFlashButtonGroupMenu enumeration member 4616
TTransactionItem class 1694
about TTransactionItem class 1694
FNext 1695
FTransaction 1695
FTransactionDesc 1695
TTransactionItem fields 1694
TTransactionItem Members 1694
TTransportThread class 248
about TTransportThread class 248
Create 249
Destroy 249
Execute 250
Semaphore 254
TTransportThread Members 249
TTransportThread methods 250
TTransportThread properties 254
ttTabDontCare enumeration member 4615
ttTabItemBothEdgeDisabled enumeration member 4615
ttTabItemBothEdgeFocused enumeration member 4615
ttTabItemBothEdgeHot enumeration member 4615
ttTabItemBothEdgeNormal enumeration member 4615
ttTabItemBothEdgeSelected enumeration member 4615
ttTabItemDisabled enumeration member 4615
ttTabItemFocused enumeration member 4615
ttTabItemHot enumeration member 4615
ttTabItemLeftEdgeDisabled enumeration member 4615
ttTabItemLeftEdgeFocused enumeration member 4615
ttTabItemLeftEdgeHot enumeration member 4615
ttTabItemLeftEdgeNormal enumeration member 4615
ttTabItemLeftEdgeSelected enumeration member 4615
ttTabItemNormal enumeration member 4615
ttTabItemRightEdgeDisabled enumeration member 4615
ttTabItemRightEdgeFocused enumeration member 4615
ttTabItemRightEdgeHot enumeration member 4615
ttTabItemRightEdgeNormal enumeration member 4615
ttTabItemRightEdgeSelected enumeration member 4615
ttTabItemSelected enumeration member 4615
ttTabRoot enumeration member 4615
ttTaskBarDontCare enumeration member 4617
ttTaskBarRoot enumeration member 4617
tttBaloonLink enumeration member 4620
tttBaloonNormal enumeration member 4620
tttBaloonStemDownLeft enumeration member 4620
tttBaloonStemUpLeft enumeration member 4620
tttBaloonTitleLink enumeration member 4620
tttBaloonTitleNormal enumeration member 4620
tttCloseHot enumeration member 4620
tttCloseNormal enumeration member 4620
tttClosePressed enumeration member 4620
ttTopTabItemBothEdgeDisabled enumeration member 4615
ttTopTabItemBothEdgeFocused enumeration member 4615

ttTopTabItemBothEdgeHot enumeration member 4615
ttTopTabItemBothEdgeNormal enumeration member 4615
ttTopTabItemBothEdgeSelected enumeration member 4615
ttTopTabItemDisabled enumeration member 4615
ttTopTabItemFocused enumeration member 4615
ttTopTabItemHot enumeration member 4615
ttTopTabItemLeftEdgeDisabled enumeration member 4615
ttTopTabItemLeftEdgeFocused enumeration member 4615
ttTopTabItemLeftEdgeHot enumeration member 4615
ttTopTabItemLeftEdgeNormal enumeration member 4615
ttTopTabItemLeftEdgeSelected enumeration member 4615
ttTopTabItemNormal enumeration member 4615
ttTopTabItemRightEdgeDisabled enumeration member 4615
ttTopTabItemRightEdgeFocused enumeration member 4615
ttTopTabItemRightEdgeHot enumeration member 4615
ttTopTabItemRightEdgeNormal enumeration member 4615
ttTopTabItemRightEdgeSelected enumeration member 4615
ttTopTabItemSelected enumeration member 4615
ttTreeviewDontCare enumeration member 4623
ttTreeviewRoot enumeration member 4623
tttStandardLink enumeration member 4620
tttStandardNormal enumeration member 4620
tttStandardTitleLink enumeration member 4620
tttStandardTitleNormal enumeration member 4620
tttToolTipDontCare enumeration member 4620
tttToolTipRoot enumeration member 4620
TTypes class 7057
- about TTypes class 7057
- AfterConstruction 7063
- GetSchemaDefs 7063
- TTypes Members 7058
- TTypes methods 7062

TTypeTranslator class 4680
- about TTypeTranslator class 4680
- Base64ToVar 4682
- Base64ToVar method 4682
- CastNativeToSoap 4682
- CastSoapToNative 4683
- CastSoapToVariant 4683
- CastSoapToVariant method 4683
- Create 4681
- Destroy 4681
- TTypeTranslator Members 4681
- TTypeTranslator methods 4681

TUdpSocket class 1477
- about TUdpSocket class 1477
- Active 1480
- BlockMode 1482
- Create 1479
- LocalHost 1482
- LocalPort 1483
- OnConnect 1490
- OnCreateHandle 1490
- OnDestroyHandle 1493
- OnDisconnect 1495
- OnError 1496
- OnReceive 1496
- OnSend 1496
- RemoteHost 1486
- RemotePort 1488
- TUdpSocket events 1490
- TUdpSocket Members 1478
- TUdpSocket properties 1480

TUnicodeEncoding class 4024
- about TUnicodeEncoding class 4024
- Create 4026
- GetByteCount 4026
- GetBytes 4027
- GetCharCount 4027
- GetChars 4028
- GetMaxByteCount 4028
- GetMaxCharCount 4028
- GetPreamble 4029
- TUnicodeEncoding Members 4025
- TUnicodeEncoding methods 4026

TUpdateFileList class 5864
- about TUpdateFileList class 5864
- Count 5865

Create 5865
Files 5865
TUpdateFileList Members 5864
TUpdateFileList properties 5865
TUTF7Encoding class 4029
about TUTF7Encoding class 4029
Create 4030
GetByteCount 4031
GetBytes 4031
GetCharCount 4032
GetChars 4032
GetMaxByteCount 4033
GetMaxCharCount 4033
TUTF7Encoding Members 4029
TUTF7Encoding methods 4031
TUTF8Encoding class 4034
about TUTF8Encoding class 4034
Create 4035
GetMaxByteCount 4035
GetMaxCharCount 4036
GetPreamble 4036
TUTF8Encoding Members 4034
TUTF8Encoding methods 4035
TValueListEditor class 4784
about TValueListEditor class 4784
AdjustColWidths 4789
AdjustRowCount 4789
Align 4802
Anchors 4805
BiDiMode 4806
BorderStyle 4806
CanEditModify 4790
Cells 4806
ColCount 4807
Color 4807
ColWidthsChanged 4790
Constraints 4808
Create 4788
CreateEditor 4790
CreateWnd 4791
Ctl3D 4809
DefaultColWidth 4809
DefaultDrawing 4811
DefaultRowHeight 4811
DeleteRow 4791
Destroy 4788
DisplayOptions 4812
DoExit 4792
DoOnValidate 4792
DoubleBuffered 4812
DragCursor 4814
DragKind 4814
DragMode 4814
DrawCell 4792
DropDownRows 4816
EditList 4817
EditListGetItems 4793
Enabled 4817
FindRow 4793
FixedColor 4817
FixedCols 4818
Font 4818
GetCell 4794
GetColCount 4794
GetEditLimit 4794
GetEditMask 4794
GetEditStyle 4795
GetEditText 4795
GetPickList 4795
GetRowCount 4795
GridLineWidth 4819
InsertRow 4796
IsEmptyRow 4796
ItemProps 4821
KeyDown 4797
KeyOptions 4821
Keys 4822
MouseDown 4797

OnClick 4839
OnContextPopup 4841
OnDblClick 4841
OnDragDrop 4842
OnDragOver 4845
OnDrawCell 4848
OnEditButtonClick 4849
OnEndDock 4850
OnEndDrag 4850
OnEnter 4851
OnExit 4854
OnGetEditMask 4858
OnGetEditText 4859
OnGetPickList 4860
OnKeyDown 4861
OnKeyPress 4864
OnKeyUp 4865
OnMouseActivate 4867
OnMouseDown 4868
OnMouseEnter 4870
OnMouseLeave 4871
OnMouseMove 4871
OnMouseUp 4873
OnMouseWheelDown 4875
OnMouseWheelUp 4876
OnRowMoved 4876
OnSelectCell 4877
OnSetEditText 4879
OnStartDock 4879
OnStartDrag 4880
OnStringsChange 4881
OnStringsChanging 4881
OnTopLeftChanged 4882
OnValidate 4882
Options 4822
ParentBiDiMode 4824
ParentColor 4824
ParentCtl3D 4825
ParentDoubleBuffered 4825
ParentFont 4825
ParentShowHint 4826
PopupMenu 4826
Refresh 4798
Resize 4798
RestoreCurrentRow 4798
RowCount 4831
RowMoved 4799
ScrollBars 4832
SelectCell 4799
SetCell 4799
SetEditText 4800
SetStrings 4800
ShowHint 4832
Strings 4834
StringsChanging 4800
TabOrder 4834
TitleCaptions 4835
TitleCaptionsStored 4800
TValueListEditor events 4838
TValueListEditor Members 4785
TValueListEditor methods 4788
TValueListEditor properties 4800
Values 4835
Visible 4836
VisibleColCount 4837
VisibleRowCount 4838
TValueListStrings class 4882
about TValueListStrings class 4882
Assign 4884
Changed 4887
Changing 4887
Clear 4887
Create 4883
CustomSort 4888
Delete 4888
Destroy 4884
Exchange 4889
FindItemProp 4889

InsertItem 4889
ItemProps 4890
KeyIsValid 4890
Put 4890
TValueListStrings Members 4883
TValueListStrings methods 4884
TValueListStrings properties 4890
UpdateCount 4890
TValuesListNameField class 5866
about TValuesListNameField class 5866
GetDefaultFieldName 5869
ImplGetValue 5869
TValuesListNameField Members 5867
TValuesListNameField methods 5869
TValuesListValueField class 5869
about TValuesListValueField class 5869
GetDefaultFieldName 5873
ImplGetValue 5873
TValuesListValueField Members 5870
TValuesListValueField methods 5872
TVarData::Reserved1
OleVariant class 3284
TVarData::Reserved2
OleVariant class 3284
TVarData::Reserved3
OleVariant class 3284
TVarData::VArray
OleVariant class 3284
TVarData::VBoolean
OleVariant class 3285
TVarData::VByte
OleVariant class 3285
TVarData::VCurrency
OleVariant class 3285
TVarData::VDate
OleVariant class 3285
TVarData::VDispatch
OleVariant class 3285
TVarData::VDouble
OleVariant class 3286
TVarData::VError
OleVariant class 3286
TVarData::VInteger
OleVariant class 3286
TVarData::VOleStr
OleVariant class 3286
TVarData::VPointer
OleVariant class 3287
TVarData::VSingle
OleVariant class 3287
TVarData::VSmallint
OleVariant class 3287
TVarData::VType
OleVariant class 3287
TVarData::VUnknown
OleVariant class 3287
TVariableLookup class 6716
about TVariableLookup class 6716
Create 6717
FindGlobalVariable 6717
FindGlobalVariableContainer 6718
Ident 6718
LookupObject 6718
Obj 6718
TVariableLookup Members 6716
TVariableLookup methods 6717
TVariableLookup properties 6718
TVFlags
IShellNameSpace interface 709
IShellNameSpaceDisp interface 716
TShellNameSpace class 482
twCaptionActive enumeration member 4624
twCaptionDisabled enumeration member 4624
twCaptionInactive enumeration member 4624
twCaptionSizingTemplate enumeration member 4624
twCloseButtonDisabled enumeration member 4624
twCloseButtonHot enumeration member 4624
twCloseButtonNormal enumeration member 4624

twCloseButtonPushed enumeration member 4624
twDialog enumeration member 4624
TWebAppComponents class 6243
about TWebAppComponents class 6243
AdapterDispatcher 6245
ApplicationAdapter 6246
DispatchActions 6246
EndUserAdapter 6247
LocateFileService 6247
OnAfterDispatch 6249
OnBeforeDispatch 6250
OnException 6250
PageDispatcher 6248
Sessions 6248
TWebAppComponents events 6249
TWebAppComponents Members 6244
TWebAppComponents properties 6245
UserListService 6249
TWebAppDataModule class 6585
about TWebAppDataModule class 6585
AppServices 6587
OnActivate 6587
OnDeactivate 6588
TWebAppDataModule events 6587
TWebAppDataModule Members 6585
TWebAppDataModule properties 6586
TWebAppDataModuleFactory class 6295
about TWebAppDataModuleFactory class 6295
Create 6296
GetIsAppModule 6296
TWebAppDataModuleFactory Members 6295
TWebAppDataModuleFactory methods 6296
TWebApplication class 5936
about TWebApplication class 5936
Create 5937
CreateForm 5938
Destroy 5938
Initialize 5939
Run 5939
Title 5940
TWebApplication Members 5937
TWebApplication methods 5938
TWebApplication properties 5939
TWebAppPageModule class 6588
about TWebAppPageModule class 6588
AppServices 6591
DefaultAction 6591
OnActivate 6592
OnAfterDispatchPage 6593
OnBeforeDispatchPage 6593
OnDeactivate 6594
PageProducer 6592
TWebAppPageModule events 6592
TWebAppPageModule Members 6589
TWebAppPageModule properties 6591
TWebAppPageModuleFactory class 6297
about TWebAppPageModuleFactory class 6297
Create 6298
GetIsAppModule 6299
TWebAppPageModuleFactory Members 6298
TWebAppPageModuleFactory methods 6298
TWebAppPageModuleHelper class 6594
about TWebAppPageModuleHelper class 6594
GetPageProducer 6597
TWebAppPageModuleHelper Members 6594
TWebAppPageModuleHelper methods 6596
TWebBrowser class 504
about TWebBrowser class 504
AddressBar 520
Align 520
Anchors 523
Application 524
Busy 524
ClientToWindow 507
Container 525
ControlInterface 525
CreateControl 508
DefaultInterface 526

Document 526
DragCursor 526
DragMode 527
ExecWB 508
ExecWB method 508
FullName 528
FullScreen 529
Get_Application 509
Get_Container 509
Get_Document 509
Get_Parent 509
GetProperty 510
GoBack 510
GoForward 510
GoHome 511
GoSearch 511
HWnd 529
InitControlData 512
LocationName 529
LocationURL 530
MenuBar 530
Name 531
Navigate 512
Navigate method 512
Navigate2 514
Navigate2 method 514
Offline 531
OnBeforeNavigate2 545
OnClientToHostWindow 546
OnCommandStateChange 546
OnDocumentComplete 546
OnDownloadBegin 547
OnDownloadComplete 547
OnDragDrop 548
OnDragOver 551
OnEndDrag 554
OnEnter 555
OnExit 558
OnFileDownload 562
OnFullScreen 562
OnMenuBar 562
OnNavigateComplete2 562
OnNavigateError 563
OnNewWindow2 563
OnPrintTemplateInstantiation 563
OnPrintTemplateTeardown 564
OnPrivacyImpactedStateChange 564
OnProgressChange 564
OnPropertyChange 565
OnQuit 565
OnSetSecureLockIcon 565
OnStartDrag 565
OnStatusBar 566
OnStatusTextChange 566
OnTheaterMode 567
OnTitleChange 567
OnToolBar 567
OnUpdatePageStatus 567
OnVisible 568
OnWindowClosing 568
OnWindowSetHeight 568
OnWindowSetLeft 568
OnWindowSetResizable 569
OnWindowSetTop 569
OnWindowSetWidth 569
Parent 531
ParentShowHint 532
Path 532
PopupMenu 533
PutProperty 515
QueryStatusWB 516
Quit 516
ReadyState 538
Refresh 516
Refresh2 517
Refresh2 method 517
RegisterAsBrowser 538
RegisterAsDropTarget 539

Resizable 539
ShowBrowserBar 518
ShowBrowserBar method 518
ShowHint 539
Silent 541
StatusBar 541
StatusText 541
Stop 518
TabOrder 542
TabStop 542
TheaterMode 543
ToolBar 543
TopLevelContainer 543
TWebBrowser events 544
TWebBrowser Members 504
TWebBrowser methods 507
TWebBrowser properties 519
type_ 544
Visible 544
TWebBrowser_V1 class 569
about TWebBrowser_V1 class 569
Align 577
Anchors 581
Application 581
Busy 581
Container 582
ControlInterface 582
CreateControl 572
DefaultInterface 582
Document 582
DragCursor 582
DragMode 583
Get_Application 572
Get_Container 572
Get_Document 572
Get_Parent 572
GoBack 573
GoForward 573
GoHome 573
GoSearch 573
InitControlData 574
LocationName 584
LocationURL 585
Navigate 574, 575
Navigate method 574
OnBeforeNavigate 597
OnCommandStateChange 597
OnDownloadBegin 598
OnDownloadComplete 598
OnDragDrop 598
OnDragOver 601
OnEndDrag 604
OnEnter 605
OnExit 608
OnFrameBeforeNavigate 612
OnFrameNavigateComplete 612
OnFrameNewWindow 612
OnNavigateComplete 612
OnNewWindow 613
OnProgressChange 613
OnPropertyChange 613
OnQuit 613
OnStartDrag 613
OnStatusTextChange 614
OnTitleChange 614
OnWindowActivate 615
OnWindowMove 615
OnWindowResize 615
Parent 585
ParentShowHint 586
PopupMenu 587
Refresh 575
Refresh2 576
Refresh2 method 576
ShowHint 592
Stop 576
TabOrder 593
TabStop 594

TopLevelContainer 595
TWebBrowser_V1 events 597
TWebBrowser_V1 Members 570
TWebBrowser_V1 methods 571
TWebBrowser_V1 properties 577
type_ 595
Visible 595
TWebComponentClassDescription class 6023
about TWebComponentClassDescription class 6023
ComponentClass 6024
ComponentClassName 6024
Editor 6025
FrameClass 6025
TWebComponentClassDescription fields 6024
TWebComponentClassDescription Members 6023
TWebComponentList class 6025
_AddRef 6028
_Release 6028
about TWebComponentList class 6025
Add 6029
Changed 6029
Clear 6029
Count 6033
Create 6027
Destroy 6027
GetComponent 6030
GetCount 6030
GetOwner 6030
IndexOf 6030
LockNotify 6031
NewComponent 6031
NotifyLocked 6031
OnChange 6035
ParentComponent 6034
QueryInterface 6032
Remove 6032
SetComponentIndex 6032
TWebComponentList events 6035
TWebComponentList Members 6026
TWebComponentList methods 6028
TWebComponentList properties 6033
UnlockNotify 6033
WebComponentContainer_GetComponent 6033
WebComponentContainer_GetComponentCount 6033
WebComponents 6034
TWebComponentsEditorHelper class 6035
_AddRef 6037
_Release 6037
about TWebComponentsEditorHelper class 6035
CanAddClassHelper 6037
ImplCanAddClassHelper 6037
QueryInterface 6038
TWebComponentsEditorHelper Members 6036
TWebComponentsEditorHelper methods 6037
TWebConnection class 254
about TWebConnection class 254
Agent 262
Create 256
CreateTransport 257
Destroy 257
DoConnect 258
GetInterpreter 258
GetWaitEvent 259
ObjectBroker 262
Password 263
Proxy 263
ProxyByPass 263
Receive 259
Send 260
SetConnected 260
SupportCallbacks 264
Transport_GetConnected 261
Transport_SetConnected 261
TWebConnection Members 255
TWebConnection methods 257
TWebConnection properties 261
URL 264
UserName 265

TWebContainedComponent class 6038
about TWebContainedComponent class 6038
Create 6039
Destroy 6040
GetContainerList 6041
GetParentComponent 6041
HasParent 6041
IsDefaultField 6042
ParentChanged 6042
ReadState 6043
SetParentComponent 6043
SetWebParent 6043
TWebContainedComponent Members 6039
TWebContainedComponent methods 6040
TWebContainedComponent properties 6044
WebContainedComponentHelper 6044
WebParent 6044
TWebContainedComponentHelper class 6045
about TWebContainedComponentHelper class 6045
ContainerList 6051
Create 6047
Destroy 6047
GetContainer 6048
GetIndex 6048
IsDefaultField 6048
OnParentChanged 6052
ParentChanged 6049
SetComponentList 6049
SetContainer 6050
SetIndex 6050
SetWebParent 6050
TWebContainedComponentHelper events 6052
TWebContainedComponentHelper Members 6045
TWebContainedComponentHelper methods 6047
TWebContainedComponentHelper properties 6051
WebParent 6051
TWebContainedContainerComponent class 6052
about TWebContainedContainerComponent class 6052
Create 6054
Destroy 6054
GetChildren 6055
GetContainerList 6056
GetDefaultWebComponents 6056
GetParentComponent 6056
GetWebComponents 6057
HasParent 6057
ImplGetDefaultComponentList 6058
IsDefaultField 6058
ParentChanged 6058
ReadState 6059
SetChildOrder 6059
SetDefaultWebComponents 6060
SetParentComponent 6060
SetWebParent 6061
TWebContainedContainerComponent Members 6053
TWebContainedContainerComponent methods 6055
TWebContainedContainerComponent properties 6061
WebContainedComponentHelper 6061
WebContainerComponentHelper 6062
WebParent 6062
TWebContainerComponent class 6063
about TWebContainerComponent class 6063
Create 6064
Destroy 6064
GetChildren 6065
GetDefaultWebComponents 6065
GetWebComponents 6065
ImplGetDefaultComponentList 6066
SetChildOrder 6066
SetDefaultWebComponents 6067
TWebContainerComponent Members 6063
TWebContainerComponent methods 6064
TWebContainerComponent properties 6067
WebContainerComponentHelper 6067
TWebContainerComponentHelper class 6068
about TWebContainerComponentHelper class 6068
Create 6069
DefaultWebComponents 6074

Destroy 6070
GetChildren 6071
GetComponentList 6071
GetDefaultComponentList 6072
GetDefaultWebComponents 6072
OnGetDefaultComponentList 6075
OnSetDefaultComponentList 6075
SetChildOrder 6072
SetDefaultWebComponents 6073
TWebContainerComponentHelper events 6074
TWebContainerComponentHelper Members 6068
TWebContainerComponentHelper methods 6070
TWebContainerComponentHelper properties 6073
Webcomponents 6074
TWebContentOptions class 6076
about TWebContentOptions class 6076
Create 6077
Flags 6077
TWebContentOptions Members 6076
TWebContentOptions properties 6077
TWebContext class 5976
about TWebContext class 5976
Create 5978
Destroy 5979
FindApplicationModule 5980
FindDefaultPageInfo 5980
FindModuleClass 5981
FindModuleName 5981
FindPage 5982
FindPageInfo 5982
FindVariableContainer 5983
GetAdapterRequest 5983
GetAdapterResponse 5983
GetDispatchedPageName 5984
GetEndUser 5984
GetPageCount 5984
GetPageInfo 5984
GetRequestedPageName 5985
GetSession 5985
GetWebModules 5985
GetWebRequest 5985
GetWebResponse 5986
SetAdapterRequest 5986
SetAdapterResponse 5986
SetDispatchedPageName 5986
SetRequestedPageName 5986
TWebContext Members 5976
TWebContext methods 5979
TWebDataModule class 6597
about TWebDataModule class 6597
OnActivate 6598
OnDeactivate 6599
TWebDataModule events 6598
TWebDataModule Members 6597
TWebDataModuleFactory class 6299
about TWebDataModuleFactory class 6299
Create 6300
TWebDataModuleFactory Members 6300
TWebModuleFactoryList class 6622
about TWebModuleFactoryList class 6622
AddFactory 6624
AppModuleFactory 6625
Create 6623
Destroy 6624
GetItem 6625
ItemCount 6625
Items 6626
TWebModuleFactoryList Members 6622
TWebModuleFactoryList methods 6624
TWebModuleFactoryList properties 6625
TWebModuleList class 6626
about TWebModuleList class 6626
AddModule 6628
AddModuleClass 6629
AddModuleName 6629
AutoCreateModules 6630
AutoDestroyModules 6630
Create 6627

Destroy 6628
Factories 6633
FindModuleClass 6631
FindModuleName 6631
GetFactory 6632
GetFactoryCount 6632
GetItem 6632
GetItemCount 6632
GetOnModuleAdded 6633
SetOnModuleAdded 6633
TWebModuleList Members 6627
TWebModuleList methods 6628
TWebModuleList properties 6633
TWebNotifyContainer class 6599
about TWebNotifyContainer class 6599
Add 6602
Create 6601
Destroy 6601
FList 6602
GetCount 6602
NotifyWebActivate 6603
NotifyWebDeactivate 6603
TWebNotifyContainer fields 6602
TWebNotifyContainer Members 6600
TWebNotifyContainer methods 6602
TWebPageInfo class 6301
about TWebPageInfo class 6301
Create 6302
TWebPageInfo Members 6301
TWebPageModule class 6603
about TWebPageModule class 6603
DefaultAction 6606
OnActivate 6607
OnAfterDispatchPage 6607
OnBeforeDispatchPage 6608
OnDeactivate 6608
PageProducer 6606
TWebPageModule events 6607
TWebPageModule Members 6604
TWebPageModule properties 6606
TWebPageModuleFactory class 6303
about TWebPageModuleFactory class 6303
Create 6304
TWebPageModuleFactory Members 6303
TWebPageModuleHelper class 6609
about TWebPageModuleHelper class 6609
GetPageProducer 6611
TWebPageModuleHelper Members 6609
TWebPageModuleHelper methods 6611
TWebRequestHandler class 6633
about TWebRequestHandler class 6633
ActivateWebModules 6636
ActiveCount 6638
AddWebModuleFactory 6636
CacheConnections 6638
Create 6635
DeactivateWebModules 6636
Destroy 6635
HandleException 6637
HandleRequest 6637
InactiveCount 6639
MaxConnections 6640
TWebRequestHandler Members 6634
TWebRequestHandler methods 6635
TWebRequestHandler properties 6638
WebModuleClass 6640
WebModuleFactory 6640
WebModuleFactoryCount 6641
TWebServExp class 6727
about TWebServExp class 6727
ArrayAsComplexContent 6731
BindingType 6732
Create 6729
Destroy 6729
GenerateXMLSchema 6730
GetWebServExp 6731
GetWSDLForInterface 6731
ImportLocations 6732

ImportNames 6732
OnAfterPublishingWSDL 6733
OnBeforePublishingTypes 6733
OnPublishingType 6733
ServiceAddress 6732
TargetNameSpace 6732
TWebServExp events 6733
TWebServExp Members 6727
TWebServExp methods 6730
TWebServExp properties 6731
WSDLElements 6733
TWebSession class 6251
about TWebSession class 6251
Create 6252
Create constructor 6252
GetSessionID 6253
GetTimeoutMinutes 6253
GetValue 6254
SetTimeoutMinutes 6254
SetValue 6254
Terminate 6255
TWebSession Members 6251
TWebSession methods 6252
UpdateResponse 6255
TWebSessionIDsImpl class 6759
about TWebSessionIDsImpl class 6759
Create 6761
Destroy 6761
GetID 6762
GetIDCount 6763
TWebSessionIDsImpl Members 6760
TWebSessionIDsImpl methods 6762
TWebSessionImpl class 6763
about TWebSessionImpl class 6763
Create 6765
Destroy 6765
ExpirationTime 6770
Expired 6770
GetExpirationTime 6766
GetExpired 6767
GetID 6767
GetIsActive 6767
GetItemCount 6767
GetItems 6767
GetStartTime 6768
GetTerminated 6768
GetTimeoutMinutes 6768
GetTouchTime 6768
GetValue 6769
ID 6770
SetTimeoutMinutes 6769
SetValue 6769
StartTime 6771
Terminate 6769
Terminated 6771
TimeoutMinutes 6771
TouchTime 6771
TWebSessionImpl Members 6763
TWebSessionImpl methods 6766
TWebSessionImpl properties 6770
Values 6771
TWebUserItem class 6787
about TWebUserItem class 6787
AccessRights 6790
AccessRightsStrings 6791
Assign 6789
CheckRights 6790
Create 6788
Destroy 6789
Password 6791
TWebUserItem Members 6788
TWebUserItem methods 6789
TWebUserItem properties 6790
UserID 6791
UserName 6791
TWebUserItems class 6792
about TWebUserItems class 6792
Create 6793

FindUserID 6793
FindUserName 6793
GetAttr 6794
GetAttrCount 6794
GetItemAttr 6795
GetOwner 6795
Items 6796
TWebUserItems Members 6792
TWebUserItems methods 6793
TWebUserItems properties 6796
TWebUserList class 6796
about TWebUserList class 6796
OnAfterCheckAccessRights 6798
OnAfterValidateUser 6798
OnBeforeCheckAccessRights 6799
OnBeforeValidateUser 6799
OnUserIDNotFound 6799
OnValidateUserError 6800
TWebUserList events 6798
TWebUserList Members 6797
TWebUserList properties 6797
UserItems 6797
twFrameBottomActive enumeration member 4624
twFrameBottomInactive enumeration member 4624
twFrameBottomSizingTemplate enumeration member 4624
twFrameLeftActive enumeration member 4624
twFrameLeftInactive enumeration member 4624
twFrameLeftSizingTemplate enumeration member 4624
twFrameRightActive enumeration member 4624
twFrameRightInactive enumeration member 4624
twFrameRightSizingTemplate enumeration member 4624
twHelpButtonDisabled enumeration member 4624
twHelpButtonHot enumeration member 4624
twHelpButtonNormal enumeration member 4624
twHelpButtonPushed enumeration member 4624
twHorzScrollDisabled enumeration member 4624
twHorzScrollHot enumeration member 4624
twHorzScrollNormal enumeration member 4624
twHorzScrollPushed enumeration member 4624
twHorzThumbDisabled enumeration member 4624
twHorzThumbHot enumeration member 4624
twHorzThumbNormal enumeration member 4624
twHorzThumbPushed enumeration member 4624
TWideStringList class 6808
about TWideStringList class 6808
Add 6812
AddObject 6813
CaseSensitive 6822
Changed 6813
Changing 6814
Clear 6814
CompareStrings 6814
Create 6811
Create constructor 6810
CustomSort 6815
Delete 6815
Destroy 6811
Duplicates 6822
Exchange 6816
Find 6816
Get 6817
GetCapacity 6817
GetCount 6817
GetObject 6818
IndexOf 6818
Insert 6819
InsertItem 6819
InsertObject 6819
OnChange 6824
OnChanging 6825
OwnsObjects 6823
Put 6820
PutObject 6820
SetCapacity 6820
SetUpdateState 6821
Sort 6821
Sorted 6823
TWideStringList events 6824

TWideStringList Members 6809
TWideStringList methods 6812
TWideStringList properties 6822
TWideStrings class 6825
about TWideStrings class 6825
Add 6829
AddObject 6830
AddStrings 6830
AddStrings method 6830
Append 6831
Assign 6831
AssignTo 6832
BeginUpdate 6832
Capacity 6847
Clear 6833
CommaText 6848
CompareStrings 6833
Count 6849
DefineProperties 6834
Delete 6834
DelimitedText 6849
Delimiter 6850
Destroy 6828
EndUpdate 6835
Equals 6835
Error 6836
Error method 6836
Exchange 6836
ExtractName 6837
Get 6837
GetCapacity 6837
GetCount 6838
GetEnumerator 6838
GetObject 6838
GetText 6839
GetTextStr 6839
IndexOf 6840
IndexOfName 6840
IndexOfObject 6841
Insert 6841
InsertObject 6842
LineBreak 6850
LoadFromFile 6842
LoadFromStream 6842
Move 6843
Names 6850
NameValueSeparator 6851
Objects 6851
Put 6843
PutObject 6844
QuoteChar 6852
SaveToFile 6844
SaveToStream 6845
SetCapacity 6845
SetText 6845
SetTextStr 6846
SetUpdateState 6846
StrictDelimiter 6852
Strings 6853
StringsAdapter 6854
Text 6854
TWideStrings Members 6826
TWideStrings methods 6828
TWideStrings properties 6847
UpdateCount 6855
ValueFromIndex 6855
Values 6856
TWideStringsEnumerator class 6856
about TWideStringsEnumerator class 6856
Create 6857
Current 6858
GetCurrent 6857
MoveNext 6857
TWideStringsEnumerator Members 6856
TWideStringsEnumerator methods 6857
TWideStringsEnumerator properties 6858
TWindowAction class 1932
about TWindowAction class 1932

Create 1933
Form 1935
GetForm 1934
HandlesTarget 1934
Notification 1934
TWindowAction Members 1933
TWindowAction methods 1933
TWindowAction properties 1935
UpdateTarget 1935
TWindowArrange class 1936
about TWindowArrange class 1936
ExecuteTarget 1937
TWindowArrange Members 1936
TWindowArrange methods 1937
TWindowCascade class 1937
about TWindowCascade class 1937
ExecuteTarget 1938
TWindowCascade Members 1938
TWindowCascade methods 1938
TWindowClose class 1939
about TWindowClose class 1939
ExecuteTarget 1940
TWindowClose Members 1940
TWindowClose methods 1940
UpdateTarget 1941
TWindowMinimizeAll class 1941
about TWindowMinimizeAll class 1941
ExecuteTarget 1942
TWindowMinimizeAll Members 1942
TWindowMinimizeAll methods 1942
TWindowTileHorizontal class 1943
about TWindowTileHorizontal class 1943
ExecuteTarget 1944
TWindowTileHorizontal Members 1943
TWindowTileHorizontal methods 1944
TWindowTileVertical class 1944
about TWindowTileVertical class 1944
ExecuteTarget 1946
TWindowTileVertical Members 1945
TWindowTileVertical methods 1945
TWinHandleUser class 1329
about TWinHandleUser class 1329
TWinHandleUser Members 1330
TWinHandleUser methods 1331
UseHandle1 1331
UseHandle2 1331
TWinSocketStream class 138
about TWinSocketStream class 138
Create 140
Destroy 140
Read 141
Seek 141
TimeOut 143
TWinSocketStream Members 139
TWinSocketStream methods 141
TWinSocketStream properties 143
WaitForData 142
Write 142
twMaxButtonDisabled enumeration member 4624
twMaxButtonHot enumeration member 4624
twMaxButtonNormal enumeration member 4624
twMaxButtonPushed enumeration member 4624
twMaxCaptionActive enumeration member 4624
twMaxCaptionDisabled enumeration member 4624
twMaxCaptionInactive enumeration member 4624
twMDICloseButtonDisabled enumeration member 4624
twMDICloseButtonHot enumeration member 4624
twMDICloseButtonNormal enumeration member 4624
twMDICloseButtonPushed enumeration member 4624
twMDIHelpButtonDisabled enumeration member 4624
twMDIHelpButtonHot enumeration member 4624
twMDIHelpButtonNormal enumeration member 4624
twMDIHelpButtonPushed enumeration member 4624
twMDIMinButtonDisabled enumeration member 4624
twMDIMinButtonHot enumeration member 4624
twMDIMinButtonNormal enumeration member 4624
twMDIMinButtonPushed enumeration member 4624
twMDIRestoreButtonDisabled enumeration member 4624

twMDIRestoreButtonHot enumeration member 4624
twMDIRestoreButtonNormal enumeration member 4624
twMDIRestoreButtonPushed enumeration member 4624
twMDISysButtonDisabled enumeration member 4624
twMDISysButtonHot enumeration member 4624
twMDISysButtonNormal enumeration member 4624
twMDISysButtonPushed enumeration member 4624
twMinButtonDisabled enumeration member 4624
twMinButtonHot enumeration member 4624
twMinButtonNormal enumeration member 4624
twMinButtonPushed enumeration member 4624
twMinCaptionActive enumeration member 4624
twMinCaptionDisabled enumeration member 4624
twMinCaptionInactive enumeration member 4624
twRestoreButtonDisabled enumeration member 4624
twRestoreButtonHot enumeration member 4624
twRestoreButtonNormal enumeration member 4624
twRestoreButtonPushed enumeration member 4624
TWSDLClientNode class 7266
- about TWSDLClientNode class 7266
- BeforeExecute 7267
- Execute 7268, 7269
- Execute method 7268
- GetMimeBoundary 7269
- Notification 7269
- SetMimeBoundary 7270
- TransportNode 7270
- TWSDLClientNode Members 7267
- TWSDLClientNode methods 7267
- TWSDLClientNode properties 7270
- WSDLView 7271

TWSDLDocument class 7063
- about TWSDLDocument class 7063
- AfterConstruction 7069
- Definition 7070
- GetDefinition 7069
- LoadData 7069
- SetDefinition 7070
- TWSDLDocument Members 7064
- TWSDLDocument methods 7068
- TWSDLDocument properties 7070

TWSDLHTMLPublish class 7280
- about TWSDLHTMLPublish class 7280
- AddInterfaceList 7283
- AddPortList 7283
- AdminEnabled 7287
- Create 7281
- Destroy 7282
- DispatchEnabled 7283
- DispatchMask 7283
- DispatchMethodType 7284
- DispatchRequest 7284
- GetHostScriptBaseURL 7285
- GetTargetNamespace 7285
- HandleRequest 7285
- OnAfterPublishingWSDL 7289
- OnBeforePublishingTypes 7290
- OnBeforePublishingWSDL 7290
- OnPublishingType 7290
- PublishOptions 7288
- ServiceInfo 7286
- TargetNamespace 7288
- TWSDLHTMLPublish events 7289
- TWSDLHTMLPublish Members 7281
- TWSDLHTMLPublish methods 7282
- TWSDLHTMLPublish properties 7287
- UpdatePortList 7286
- WebDispatch 7288
- WSILInfo 7286

TWSDLItems class 7211
- about TWSDLItems class 7211
- CompareName 7219
- Create 7217
- Create constructor 7217
- GetBindingForServicePort 7220
- GetBindingOfType 7221
- GetBindingOperationNode 7221
- GetBindings 7222

GetBindingType 7222
GetHttpBindingAttribute 7223
GetHttpBindingAttribute method 7223
GetImports 7223
GetLocationForImport 7224
GetMessageNode 7224
GetMessages 7224
GetName 7225
GetOperationNode 7225
GetOperations 7226
GetOperationsForBinding 7226
GetPartNode 7227
GetParts 7227
GetPartsForOperation 7228
GetPortsForService 7229
GetPortTypeNode 7229
GetPortTypes 7229
GetSchemaNode 7230
GetSchemas 7230
GetServiceAndPortOfBinding 7231
GetServiceNode 7231
GetServices 7232
GetSoapAction 7232
GetSoapActionList 7233
GetSoapAddressForServicePort 7234
GetSoapBindingAttribute 7234
GetSoapBindingInputNode 7235
GetSoapBindingIONode 7235
GetSoapBindingOutputNode 7236
GetSoapBodyAttribute 7236
GetSoapBodyInputAttribute 7237
GetSoapBodyNamespace 7238
GetSoapBodyOutputAttribute 7238
GetSoapHeaders 7239
GetSoapInputHeaders 7240
GetSoapOperationAttribute 7240
GetSoapOperationStyle 7241
GetSoapOutputHeaders 7241
GetTargetNamespace 7242
GetTargetNamespacePrefix 7243
GetWSDLDoc 7243
GetWSDLItems 7243
HasTypesNode 7244
IsPortTypeHTTPBound 7244
IterateImports 7244
Load 7245
StreamLoader 7245
TargetNamespace 7246
TWSDLItems Members 7212
TWSDLItems methods 7218
TWSDLItems properties 7245
TWSDLPublish class 7291
about TWSDLPublish class 7291
GetPortTypeEntries 7292
GetPortTypeList 7292
GetTypeSystemsList 7292
GetWSDLForPortType 7293
GetXSDForTypeSystem 7293
OnAfterPublishingWSDL 7294
OnBeforePublishingTypes 7294
OnBeforePublishingWSDL 7294
OnPublishingType 7294
TargetNamespace 7293
TWSDLPublish events 7293
TWSDLPublish Members 7291
TWSDLPublish methods 7292
TWSDLPublish properties 7293
TWSDLView class 7271
about TWSDLView class 7271
Activate 7273
IntfInfo 7272
Operation 7273
Password 7274
Port 7274
PortType 7274
Proxy 7275
Service 7275
SetDesignState 7273

TWSDLView fields 7272
TWSDLView Members 7271
TWSDLView methods 7272
TWSDLView properties 7273
UserName 7275
WSDL 7276
twSmallCaptionActive enumeration member 4624
twSmallCaptionDisabled enumeration member 4624
twSmallCaptionInactive enumeration member 4624
twSmallCaptionSizingTemplate enumeration member 4624
twSmallCloseButtonDisabled enumeration member 4624
twSmallCloseButtonHot enumeration member 4624
twSmallCloseButtonNormal enumeration member 4624
twSmallCloseButtonPushed enumeration member 4624
twSmallFrameBottomActive enumeration member 4624
twSmallFrameBottomInactive enumeration member 4624
twSmallFrameBottomSizingTemplate enumeration member 4624
twSmallFrameLeftActive enumeration member 4624
twSmallFrameLeftInactive enumeration member 4624
twSmallFrameLeftSizingTemplate enumeration member 4624
twSmallFrameRightActive enumeration member 4624
twSmallFrameRightInactive enumeration member 4624
twSmallFrameRightSizingTemplate enumeration member 4624
twSmallMaxCaptionActive enumeration member 4624
twSmallMaxCaptionDisabled enumeration member 4624
twSmallMaxCaptionInactive enumeration member 4624
twSmallMinCaptionActive enumeration member 4624
twSmallMinCaptionDisabled enumeration member 4624
twSmallMinCaptionInactive enumeration member 4624
twSysButtonDisabled enumeration member 4624
twSysButtonHot enumeration member 4624
twSysButtonNormal enumeration member 4624
twSysButtonPushed enumeration member 4624
twVertScrollDisabled enumeration member 4624
twVertScrollHot enumeration member 4624
twVertScrollNormal enumeration member 4624
twVertScrollPushed enumeration member 4624
twVertThumbDisabled enumeration member 4624
twVertThumbHot enumeration member 4624
twVertThumbNormal enumeration member 4624
twVertThumbPushed enumeration member 4624
twWindowDontCare enumeration member 4624
twWindowRoot enumeration member 4624
TXMLBroker class 7303
about TXMLBroker class 7303
AddNotify 7307
AfterDispatch 7329
ApplyXMLUpdates 7308
AppServer 7322
AS_ApplyUpdates 7308
AS_FetchParams 7309
AS_GetRecords 7309
BeforeDispatch 7330
CheckInactive 7309
Connected 7322
Create 7305
Destroy 7306
DispatchEnabled 7309
DispatchMask 7309
DispatchMethodType 7310
DispatchRequest 7310
FetchParams 7310
GetAppServer 7310
GetConnected 7311
GetDelta 7311
GetErrorCount 7311
GetErrorResponse 7312
GetErrors 7312
GetHasAppServer 7312
GetNotify 7312
GetNotifyCount 7313
GetProducerName 7313
GetRedirect 7313
GetResponse 7313
GetXMLRecords 7314
GetXMLStream 7314
HasAppServer 7323
HTMLSubmitFormName 7315

MasterRowSetVarName 7315
MaxErrors 7323
MaxRecords 7324
Notification 7316
Notify 7325
NotifyCount 7325
OnGetErrorResponse 7330
OnGetResponse 7331
OnRequestRecords 7332
OnRequestUpdate 7332
Params 7325
ProviderName 7326
ReconcileProducer 7327
RemoteServer 7328
RemoveNotify 7316
RequestRecords 7316
RequestUpdate 7317
RowSetVarName 7318
SendConnectEvent 7319
SetAppServer 7319
SetConnected 7319
SetParams 7319
SetProvider 7320
SetProviderName 7320
SetReconcileProducer 7320
SetRemoteServer 7321
SetWebDispatch 7321
SubmitFormVarName 7321
TXMLBroker events 7329
TXMLBroker Members 7304
TXMLBroker methods 7306
TXMLBroker properties 7321
WebDispatch 7328
TXMLData class 7797
about TXMLData class 7797
Create 7798
Destroy 7798
LoadFromXML 7799
LoadFromXML method 7799
ObjectToSOAP 7799
SOAPToObject 7800
TXMLData Members 7797
TXMLData methods 7798
TXMLData properties 7801
XMLNode 7801
TXMLDocument class 7347
_AddRef 7354
_Release 7354
about TXMLDocument class 7347
Active 7381
AddChild 7355
AddChild method 7354
AfterClose 7393
AfterConstruction 7355
AfterNodeChange 7393
AfterOpen 7394
AssignParseOptions 7356
AsyncLoadState 7382
BeforeClose 7394
BeforeNodeChange 7394
BeforeOpen 7395
CheckActive 7356
CheckAutoSave 7356
CheckDOM 7356
ChildNodes 7383
Create 7351
Create constructor 7351
CreateElement 7357
CreateNode 7357
DefineProperties 7358
Destroy 7352
DoAfterClose 7359
DoAfterOpen 7359
DoBeforeClose 7359
DoBeforeOpen 7360
DocBindingInfo 7383
DocSource 7383
DocumentElement 7383

DOMDocument 7384
DOMImplementation 7385
DOMParseOptions 7385
DOMPersist 7386
DOMVendor 7386
DoNodeChange 7360
Encoding 7386
FileName 7387
GeneratePrefix 7360
GetActive 7360
GetAsyncLoadState 7361
GetChildNodeClass 7361
GetChildNodes 7361
GetDocBinding 7362
GetDocumentElement 7362
GetDocumentNode 7362
GetDocumentObject 7363
GetDOMDocument 7363
GetDOMPersist 7363
GetEncoding 7363
GetFileName 7364
GetModified 7364
GetNodeIndentStr 7364
GetOptions 7365
GetParseOptions 7365
GetPrologNode 7365
GetPrologValue 7366
GetSchemaRef 7366
GetStandAlone 7366
GetVersion 7366
GetXML 7367
InternalSetPrologValue 7367
IsEmptyDoc 7367
LoadData 7368
Loaded 7368
LoadFromFile 7368
LoadFromStream 7369
LoadFromXML 7370
LoadFromXML method 7370
Modified 7387
NewInstance 7370
Node 7388
NodeIndentStr 7388
NSPrefixBase 7389
OnAsyncLoad 7395
Options 7389
ParseOptions 7390
PrefixID 7391
ReadDOMVendor 7371
Refresh 7371
RegisterDocBinding 7372
ReleaseDoc 7372
Resync 7372
SaveToFile 7373
SaveToStream 7373
SaveToUTF8String 7374
SaveToXML 7374
SaveToXML method 7374
SaveToXMLStrings 7375
SchemaRef 7391
SetActive 7375
SetDocumentElement 7375
SetDOMDocument 7376
SetEncoding 7376
SetFileName 7376
SetModified 7377
SetNodeIndentStr 7377
SetOnAsyncLoad 7377
SetOptions 7378
SetParseOptions 7378
SetPrologValue 7378
SetStandAlone 7378
SetVersion 7379
SetXML 7379
SetXMLStrings 7379
StandAlone 7391
TXMLDocument events 7392
TXMLDocument Members 7348

TXMLDocument methods 7352
TXMLDocument properties 7380
Version 7391
WriteDOMVendor 7380
XML 7392
XMLStringsChanging 7380
TXMLNode class 7396
_AddRef 7403
_Release 7404
about TXMLNode class 7396
AddChild 7404
AddChild method 7404
AddHostedNode 7405
AttributeListNotify 7406
AttributeNodes 7438
CheckNotHosted 7406
CheckReadOnly 7407
CheckTextNode 7407
ChildListNotify 7408
ChildNodeClasses 7438
ChildNodes 7439
ClearDocumentRef 7408
CloneNode 7409
Collection 7439
Create 7400
CreateAttributeList 7409
CreateAttributeNode 7410
CreateChildList 7410
CreateChildNode 7410
CreateCollection 7411
CreateHosted 7401
DeclareNamespace 7412
Destroy 7401
DOMElement 7412
DOMNode 7440
DoNodeChange 7413
FindHostedNode 7413
FindNamespaceDecl 7414
FindNamespaceURI 7414
GetAttribute 7415
GetAttributeNodes 7415
GetAttributeNS 7416
GetChildNodeClasses 7416
GetChildNodes 7417
GetChildValue 7417
GetCollection 7418
GetDOMNode 7418
GetHasChildNodes 7418
GetHostedNodes 7419
GetHostNode 7419
GetIsTextElement 7419
GetLocalName 7420
GetNamespaceURI 7421
GetNodeName 7421
GetNodeObject 7422
GetNodeType 7422
GetNodeValue 7423
GetOwnerDocument 7423
GetParentNode 7424
GetPrefix 7424
GetPrefixedName 7425
GetReadOnly 7425
GetText 7426
GetXML 7426
HasAttribute 7427
HasAttribute method 7426
HasChildNode 7427
HasChildNode method 7427
HostedNodes 7441
HostNode 7441
InternalAddChild 7428
NestingLevel 7428
NextSibling 7429
Normalize 7429
OnHostAttrNotify 7443
OnHostChildNotify 7444
OwnerDocument 7442
ParentNode 7443

PreviousSibling 7430
RegisterChildNode 7430
RegisterChildNodes 7431
RemoveHostedNode 7431
Resync 7432
SetAttribute 7432
SetAttributeNodes 7432
SetAttributeNS 7433
SetChildNodes 7433
SetChildValue 7434
SetCollection 7434
SetNodeValue 7435
SetParentNode 7435
SetReadOnly 7436
SetText 7436
TransformNode 7437
TransformNode method 7437
TXMLNode events 7443
TXMLNode Members 7397
TXMLNode methods 7402
TXMLNode properties 7437
TXMLNodeCollection class 7444
about TXMLNodeCollection class 7444
AddItem 7451
AfterConstruction 7451
ChildListNotify 7452
Clear 7452
Count 7457
CreateItemList 7453
Delete 7453
GetCount 7454
GetList 7454
GetNode 7454
InsertInCollection 7455
IsCollectionItem 7455
ItemInterface 7458
ItemNS 7458
ItemTag 7459
List 7459
Remove 7456
SetChildNodes 7456
TXMLNodeCollection Members 7445
TXMLNodeCollection methods 7450
TXMLNodeCollection properties 7457
UpdateCollectionList 7457
TXMLNodeList class 7460
about TXMLNodeList class 7460
Add 7464
BeginUpdate 7465
Clear 7465
Count 7474
Create 7462
DefaultNamespaceURI 7475
Delete 7466
Delete method 7465
Destroy 7463
DoNotify 7466
EndUpdate 7467
FindNode 7468
FindNode method 7467
FindSibling 7468
First 7469
Get 7469
GetCount 7470
GetNode 7470
GetUpdateCount 7471
IndexOf 7471
IndexOf method 7471
Insert 7472
InternalInsert 7472
Last 7473
List 7475
NotificationProc 7476
Owner 7475
Remove 7473
ReplaceNode 7473
TXMLNodeList events 7476
TXMLNodeList Members 7461

TXMLNodeList methods 7464
TXMLNodeList properties 7474
UpdateCount 7476
TXMLParams class 7333
about TXMLParams class 7333
AssignStrings 7335
AssignTo 7335
TXMLParams Members 7334
TXMLParams methods 7334
TXMLSchemaDefs class 7071
about TXMLSchemaDefs class 7071
Add 7076
GetXMLSchemaDef 7077
TXMLSchemaDefs Members 7071
TXMLSchemaDefs methods 7076
TXMLTransform class 7727
about TXMLTransform class 7727
AfterEachRow 7737
AfterEachRowSet 7737
BeforeEachRow 7738
BeforeEachRowSet 7738
Data 7731
DoTransform 7729
EmptyDestinationDocument 7732
GetData 7729
GetResultString 7729
OnTranslate 7739
ResultDocument 7733
ResultString 7733
SourceXml 7734
SourceXmlDocument 7734
SourceXmlFile 7735
Transform 7729
TransformationDocument 7735
TransformationFile 7736
TransformXML 7730
Translate 7730
TXMLTransform events 7737
TXMLTransform Members 7727
TXMLTransform methods 7728
TXMLTransform properties 7731
TXMLTransformClient class 7739
about TXMLTransformClient class 7739
ApplyUpdates 7742
Create 7741
Destroy 7741
GetDataAsXml 7742
ProviderName 7745
RemoteServer 7745
SetParams 7743
SetupAppServer 7744
TransformApplyUpdates 7746
TransformGetData 7746
TransformSetParams 7747
TXMLTransformClient Members 7740
TXMLTransformClient methods 7741
TXMLTransformClient properties 7744
TXMLTransformProvider class 7747
about TXMLTransformProvider class 7747
AfterApplyUpdates 7754
AfterGetRecords 7755
AfterRowRequest 7756
BeforeApplyUpdates 7756
BeforeGetRecords 7757
BeforeRowRequest 7758
CacheData 7751
Create 7749
Destroy 7749
InternalApplyUpdates 7750
InternalGetRecords 7750
OnDataRequest 7759
TransformRead 7752
TransformWrite 7753
TXMLTransformProvider events 7754
TXMLTransformProvider Members 7748
TXMLTransformProvider methods 7750
TXMLTransformProvider properties 7751
XMLDataFile 7753

TXPManifest class 7786
about TXPManifest class 7786
TXPStyleActionBars class 7788
about TXPStyleActionBars class 7788
GetAddRemoveItemClass 7789
GetColorMapClass 7789
GetControlClass 7789
GetPopupClass 7790
GetScrollBtnClass 7790
GetStyleName 7790
TXPStyleActionBars Members 7788
TXPStyleActionBars methods 7788
TXPStyleAddRemoveItem class 7764
about TXPStyleAddRemoveItem class 7764
CalcBounds 7765
DrawBackground 7765
DrawGlyph 7766
DrawSeparator 7766
DrawText 7766
DrawUnusedEdges 7766
TXPStyleAddRemoveItem Members 7764
TXPStyleAddRemoveItem methods 7765
TXPStyleButton class 7767
about TXPStyleButton class 7767
DrawBackground 7767
DrawFrame 7768
DrawGlyph 7768
DrawText 7768
TXPStyleButton Members 7767
TXPStyleButton methods 7767
TXPStyleCustomizePopup class 7769
about TXPStyleCustomizePopup class 7769
Create 7769
GetAddRemoveItemClass 7770
GetDefaultColorMapClass 7770
NCPaint 7770
TXPStyleCustomizePopup Members 7769
TXPStyleCustomizePopup methods 7770
TXPStyleDropDownBtn class 7770
about TXPStyleDropDownBtn class 7770
DrawBackground 7771
DrawFrame 7772
DrawGlyph 7772
GetPopupClass 7772
TXPStyleDropDownBtn Members 7771
TXPStyleDropDownBtn methods 7771
TXPStyleExpandBtn class 7772
about TXPStyleExpandBtn class 7772
DrawBackground 7773
DrawFrame 7773
TXPStyleExpandBtn Members 7773
TXPStyleExpandBtn methods 7773
TXPStyleMenuButton class 7774
about TXPStyleMenuButton class 7774
DrawBackground 7774
DrawText 7775
TXPStyleMenuButton Members 7774
TXPStyleMenuButton methods 7774
TXPStyleMenuItem class 7775
about TXPStyleMenuItem class 7775
CalcBounds 7776
CalcLayout 7776
DrawBackground 7777
DrawGlyph 7777
DrawSeparator 7777
DrawShadowedText 7777
DrawSubMenuGlyph 7778
DrawText 7778
DrawUnusedEdges 7778
TXPStyleMenuItem Members 7775
TXPStyleMenuItem methods 7776
TXPStylePopupMenu class 7778
about TXPStylePopupMenu class 7778
Create 7779
Destroy 7779
DisplayShadow 7780
GetExpandBtnClass 7780
HideShadow 7781

NCPaint 7781
PositionPopup 7781
TXPStylePopupMenu Members 7779
TXPStylePopupMenu methods 7780
VisibleChanging 7781
TXPStyleToolScrollBtn class 7782
about TXPStyleToolScrollBtn class 7782
DrawBackground 7782
TXPStyleToolScrollBtn Members 7782
TXPStyleToolScrollBtn methods 7782
TXSBoolean class 7801
about TXSBoolean class 7801
AsBoolean 7803
NativeToXS 7802
TXSBoolean Members 7801
TXSBoolean methods 7802
TXSBoolean properties 7803
XSToNative 7802
TXSCustomDateTime class 7803
about TXSCustomDateTime class 7803
AsDateTime 7815
AsUTCDateTime 7816
Create 7805
Day 7816
Destroy 7805
FDateTime 7806
FractionalSeconds 7817
GetAsDateTime 7807
GetAsUTCDateTime 7807
GetDay 7807
GetFractionalSeconds 7808
GetHour 7808
GetHourOffset 7808
GetMillisecond 7808
GetMinute 7809
GetMinuteOffset 7809
GetMonth 7809
GetSecond 7810
GetUseZeroMilliseconds 7810
GetYear 7810
Hour 7817
HourOffset 7818
Millisecond 7818
Minute 7819
MinuteOffset 7819
Month 7820
Second 7820
SetAsDateTime 7811
SetAsUTCDateTime 7811
SetDay 7811
SetFractionalSeconds 7811
SetHour 7812
SetHourOffset 7812
SetMillisecond 7812
SetMinute 7813
SetMinuteOffset 7813
SetMonth 7813
SetSecond 7814
SetUseZeroMilliseconds 7814
SetYear 7814
TXSCustomDateTime fields 7806
TXSCustomDateTime Members 7804
TXSCustomDateTime methods 7806
TXSCustomDateTime properties 7815
UseZeroMilliseconds 7821
Year 7821
TXSDate class 7822
about TXSDate class 7822
AsDate 7827
Clone 7824
Create 7823
Day 7827
GetAsDate 7824
GetDay 7824
GetMonth 7824
GetYear 7825
Month 7827
NativeToXS 7825

SetAsDate 7825
SetDay 7825
SetMonth 7826
SetYear 7826
TXSDate Members 7822
TXSDate methods 7823
TXSDate properties 7826
XSToNative 7826
Year 7828
TXSDateTime class 7828
about TXSDateTime class 7828
Clone 7830
CompareDateTimeParam 7831
Millisecond 7832
NativeToXS 7831
TXSDateTime Members 7829
TXSDateTime methods 7830
TXSDateTime properties 7832
XSToNative 7831
TXSDecimal class 7832
about TXSDecimal class 7832
AsBcd 7835
DecimalString 7835
NativeToXS 7834
TXSDecimal Members 7833
TXSDecimal methods 7833
TXSDecimal properties 7834
XSToNative 7834
TXSDuration class 7836
about TXSDuration class 7836
Create 7838
DecimalSecond 7839
GetDecimalSecond 7838
NativeToXS 7838
SetDecimalSecond 7839
TXSDuration Members 7836
TXSDuration methods 7838
TXSDuration properties 7839
XSToNative 7839
TXSHexBinary class 7840
about TXSHexBinary class 7840
AsByteArray 7842
HexBinaryString 7843
NativeToXS 7841
TXSHexBinary Members 7841
TXSHexBinary methods 7841
TXSHexBinary properties 7842
XSToNative 7842
TXSInteger class 7843
about TXSInteger class 7843
AsInteger 7845
NativeToXS 7844
TXSInteger Members 7843
TXSInteger methods 7844
TXSInteger properties 7845
XSToNative 7844
TXSLong class 7845
about TXSLong class 7845
AsLong 7847
NativeToXS 7846
TXSLong Members 7845
TXSLong methods 7846
TXSLong properties 7847
XSToNative 7846
TXSLPageProducer class 7893
about TXSLPageProducer class 7893
TXSLPageProducer Members 7894
TXSLPageProducer properties 7899
XMLData 7899
TXSString class 7847
about TXSString class 7847
NativeToXS 7848
TXSString Members 7847
TXSString methods 7848
XSToNative 7848
TXSTime class 7849
about TXSTime class 7849
AsTime 7856

Clone 7851
FractionalSeconds 7857
FractionalSecondString 7857
GetAsTime 7851
GetFractionalSeconds 7851
GetFractionalSecondString 7852
GetHour 7852
GetHourOffset 7852
GetMillisecond 7852
GetMinute 7853
GetMinuteOffset 7853
GetSecond 7853
Hour 7857
HourOffset 7858
Millisecond 7858
Minute 7859
MinuteOffset 7859
NativeToXS 7853
Second 7859
SetAsTime 7854
SetFractionalSeconds 7854
SetHour 7854
SetHourOffset 7854
SetMillisecond 7854
SetMinute 7855
SetMinuteOffset 7855
SetSecond 7855
TXSTime Members 7849
TXSTime methods 7850
TXSTime properties 7856
UseZeroMilliseconds 7860
XSToNative 7855
Type
Variant class 3436
type_
IWebBrowser interface 749
IWebBrowser2Disp interface 775
IWebBrowserAppDisp interface 794
IWebBrowserDisp interface 801
TInternetExplorer class 457
TWebBrowser class 544
TWebBrowser_V1 class 595
Type_
IBinding interface 7082
IPart interface 7138
TypeID
TAbstractInetFileType class 968
Types 4649
IDefinition interface 7100
Types namespace 4649
Constants 4675
Functions 4651
Structs, Records, Enums 4659
Types 4663
Types.Bounds 4651
Types.Bounds function 4651
Types.CenteredRect 4652
Types.CenteredRect function 4652
Types.CenterPoint 4652
Types.CenterPoint function 4652
Types.DWORD 4663
Types.DWORD type 4663
Types.EqualRect 4652
Types.EqualRect function 4652
Types.EqualsValue 4675
Types.EqualsValue constant 4675
Types.GreaterThanValue 4675
Types.GreaterThanValue constant 4675
Types.IntersectRect 4653
Types.IntersectRect function 4653
Types.IsRectEmpty 4653
Types.IsRectEmpty function 4653
Types.LessThanValue 4676
Types.LessThanValue constant 4676
Types.OffsetRect 4654
Types.OffsetRect function 4654
Types.PByte 4664
Types.PByte type 4664

Types.PDouble 4664
Types.PDouble type 4664
Types.PInteger 4664
Types.PInteger type 4664
Types.PLongint 4665
Types.PLongint type 4665
Types.Point 4654
Types.Point function 4654
Types.PPoint 4665
Types.PPoint type 4665
Types.PRect 4666
Types.PRect type 4666
Types.PSize 4666
Types.PSize type 4666
Types.PSmallInt 4666
Types.PSmallInt type 4666
Types.PSmallPoint 4667
Types.PSmallPoint type 4667
Types.PtInRect 4655
Types.PtInRect function 4655
Types.Rect 4655
Types.Rect function 4655
Types.RectHeight 4656
Types.RectHeight function 4656
Types.RectWidth 4656
Types.RectWidth function 4656
Types.RT_RCDATA 4676
Types.RT_RCDATA constant 4676
Types.SIZE 4667
Types.SIZE type 4667
Types.SmallPoint 4656
Types.SmallPoint function 4656
Types.SplitRect 4657
Types.SplitRect function 4657
Types.tagPOINT 4667
Types.tagPOINT type 4667
Types.tagSIZE 4659
Types.tagSIZE record 4659
Types.TBooleanDynArray 4668
Types.TBooleanDynArray type 4668
Types.TByteDynArray 4668
Types.TByteDynArray type 4668
Types.TCardinalDynArray 4668
Types.TCardinalDynArray type 4668
Types.TDoubleDynArray 4669
Types.TDoubleDynArray type 4669
Types.TInt64DynArray 4669
Types.TInt64DynArray type 4669
Types.TIntegerDynArray 4670
Types.TIntegerDynArray type 4670
Types.TLongWordDynArray 4670
Types.TLongWordDynArray type 4670
Types.TPoint 4660
Types.TPoint record 4660
Types.TRect 4660
Types.TRect record 4660
Types.TShortIntDynArray 4670
Types.TShortIntDynArray type 4670
Types.TSingleDynArray 4671
Types.TSingleDynArray type 4671
Types.TSize 4671
Types.TSize type 4671
Types.TSmallIntDynArray 4672
Types.TSmallIntDynArray type 4672
Types.TSmallPoint 4661
Types.TSmallPoint record 4661
Types.TSplitRectType 4662
Types.TSplitRectType enumeration 4662
Types.TStringDynArray 4672
Types.TStringDynArray type 4672
Types.TValueRelationship 4673
Types.TValueRelationship type 4673
Types.TWideStringDynArray 4673
Types.TWideStringDynArray type 4673
Types.TWordDynArray 4674
Types.TWordDynArray type 4674
Types.UnionRect 4658
Types.UnionRect function 4658

TypeTrans 4679
TypeTrans namespace 4679
Classes 4680
Functions 4684
Variables 4687
TypeTrans.ETypeTransException 4680
TypeTrans.FloatToStrEx 4684
TypeTrans.FloatToStrEx function 4684
TypeTrans.GetEnumValueEx 4684
TypeTrans.GetEnumValueEx function 4684
TypeTrans.GetEnumValueExW 4685
TypeTrans.GetEnumValueExW function 4685
TypeTrans.SetEnumPropEx 4685
TypeTrans.SetEnumPropEx function 4685
TypeTrans.SetEnumPropExW 4685
TypeTrans.SetEnumPropExW function 4685
TypeTrans.StrToFloatEx 4686
TypeTrans.StrToFloatEx function 4686
TypeTrans.TTypeTranslator 4680
TypeTrans.TTypeTranslator.Base64ToVar 4682
TypeTrans.TTypeTranslator.CastNativeToSoap 4682
TypeTrans.TTypeTranslator.CastSoapToNative 4683
TypeTrans.TTypeTranslator.CastSoapToVariant 4683
TypeTrans.TTypeTranslator.Create 4681
TypeTrans.TTypeTranslator.Destroy 4681
TypeTrans.TypeTranslator 4687
TypeTrans.TypeTranslator variable 4687
TypInfo 4689
TypInfo namespace 4689
Classes 4693
Constants 4775
Functions 4701
Structs, Records, Enums 4747
Types 4762
Variables 4774
TypInfo.BooleanIdents 4774
TypInfo.BooleanIdents variable 4774
TypInfo.DotSep 4774
TypInfo.DotSep variable 4774
TypInfo.EPropertyConvertError 4693
TypInfo.EPropertyError 4693
TypInfo.FreeAndNilProperties 4703
TypInfo.FreeAndNilProperties function 4703
TypInfo.GetAnsiStrProp 4703
TypInfo.GetAnsiStrProp function 4703
TypInfo.GetDynArrayProp 4704
TypInfo.GetDynArrayProp function 4704
TypInfo.GetEnumName 4705
TypInfo.GetEnumName function 4705
TypInfo.GetEnumProp 4708
TypInfo.GetEnumProp function 4708
TypInfo.GetEnumValue 4710
TypInfo.GetEnumValue function 4710
TypInfo.GetFloatProp 4711
TypInfo.GetFloatProp function 4711
TypInfo.GetInt64Prop 4712
TypInfo.GetInt64Prop function 4712
TypInfo.GetInterfaceProp 4712
TypInfo.GetInterfaceProp function 4712
TypInfo.GetMethodProp 4713
TypInfo.GetMethodProp function 4713
TypInfo.GetObjectProp 4714
TypInfo.GetObjectProp function 4714
TypInfo.GetObjectPropClass 4715
TypInfo.GetObjectPropClass function 4715
TypInfo.GetOrdProp 4716
TypInfo.GetOrdProp function 4716
TypInfo.GetPropInfo 4717
TypInfo.GetPropInfo function 4717
TypInfo.GetPropInfos 4717
TypInfo.GetPropInfos function 4717
TypInfo.GetPropList 4718
TypInfo.GetPropList function 4718
TypInfo.GetPropName 4719
TypInfo.GetPropName function 4719
TypInfo.GetPropValue 4719
TypInfo.GetPropValue function 4719
TypInfo.GetSetElementName 4720

TypInfo.GetSetElementName function 4720
TypInfo.GetSetElementValue 4721
TypInfo.GetSetElementValue function 4721
TypInfo.GetSetProp 4722
TypInfo.GetSetProp function 4722
TypInfo.GetStrProp 4722
TypInfo.GetStrProp function 4722
TypInfo.GetTypeData 4723
TypInfo.GetTypeData function 4723
TypInfo.GetTypeName 4724
TypInfo.GetTypeName function 4724
TypInfo.GetUnicodeStrProp 4724
TypInfo.GetUnicodeStrProp function 4724
TypInfo.GetVariantProp 4725
TypInfo.GetVariantProp function 4725
TypInfo.GetWideStrProp 4726
TypInfo.GetWideStrProp function 4726
TypInfo.IsPublishedProp 4727
TypInfo.IsPublishedProp function 4727
TypInfo.IsStoredProp 4727
TypInfo.IsStoredProp function 4727
TypInfo.PPropInfo 4762
TypInfo.PPropInfo type 4762
TypInfo.PPropList 4763
TypInfo.PPropList type 4763
TypInfo.PPTypeInfo 4763
TypInfo.PPTypeInfo type 4763
TypInfo.PropIsType 4728
TypInfo.PropIsType function 4728
TypInfo.PropType 4729
TypInfo.PropType function 4729
TypInfo.PTypeData 4764
TypInfo.PTypeData type 4764
TypInfo.PTypeInfo 4767
TypInfo.PTypeInfo type 4767
TypInfo.SamePropTypeName 4729
TypInfo.SamePropTypeName function 4729
TypInfo.SetAnsiStrProp 4730
TypInfo.SetAnsiStrProp function 4730
TypInfo.SetDynArrayProp 4731
TypInfo.SetDynArrayProp function 4731
TypInfo.SetEnumProp 4732
TypInfo.SetEnumProp function 4732
TypInfo.SetFloatProp 4734
TypInfo.SetFloatProp function 4734
TypInfo.SetInt64Prop 4735
TypInfo.SetInt64Prop function 4735
TypInfo.SetInterfaceProp 4736
TypInfo.SetInterfaceProp function 4736
TypInfo.SetMethodProp 4737
TypInfo.SetMethodProp function 4737
TypInfo.SetObjectProp 4737
TypInfo.SetObjectProp function 4737
TypInfo.SetOrdProp 4738
TypInfo.SetOrdProp function 4738
TypInfo.SetPropValue 4739
TypInfo.SetPropValue function 4739
TypInfo.SetSetProp 4740
TypInfo.SetSetProp function 4740
TypInfo.SetStrProp 4741
TypInfo.SetStrProp function 4741
TypInfo.SetToString 4742
TypInfo.SetToString function 4742
TypInfo.SetUnicodeStrProp 4742
TypInfo.SetUnicodeStrProp function 4742
TypInfo.SetVariantProp 4743
TypInfo.SetVariantProp function 4743
TypInfo.SetWideStrProp 4744
TypInfo.SetWideStrProp function 4744
TypInfo.ShortStringBase 4770
TypInfo.ShortStringBase type 4770
TypInfo.SortPropList 4745
TypInfo.SortPropList function 4745
TypInfo.StringToSet 4746
TypInfo.StringToSet function 4746
TypInfo.TFloatType 4747
TypInfo.TFloatType enumeration 4747
TypInfo.TIntfFlag 4751

TypInfo.TIntfFlag enumeration 4751
TypInfo.TIntfFlags 4770
TypInfo.TIntfFlags type 4770
TypInfo.TIntfFlagsBase 4771
TypInfo.TIntfFlagsBase type 4771
TypInfo.tkAny 4775
TypInfo.tkAny constant 4775
TypInfo.tkMethods 4775
TypInfo.tkMethods constant 4775
TypInfo.tkProperties 4776
TypInfo.tkProperties constant 4776
TypInfo.TManagedField 4751
TypInfo.TManagedField record 4751
TypInfo.TMethodKind 4751
TypInfo.TMethodKind enumeration 4751
TypInfo.TOrdType 4753
TypInfo.TOrdType enumeration 4753
TypInfo.TParamFlag 4756
TypInfo.TParamFlag enumeration 4756
TypInfo.TParamFlags 4772
TypInfo.TParamFlags type 4772
TypInfo.TParamFlagsBase 4772
TypInfo.TParamFlagsBase type 4772
TypInfo.TPropData 4757
TypInfo.TPropData record 4757
TypInfo.TPropInfo 4757
TypInfo.TPropInfo record 4757
TypInfo.TPropInfoProc 4772
TypInfo.TPropInfoProc type 4772
TypInfo.TPropList 4773
TypInfo.TPropList type 4773
TypInfo.TPublishableVariantType 4694
TypInfo.TPublishableVariantType.GetInstance 4698
TypInfo.TPublishableVariantType.GetProperty 4698
TypInfo.TPublishableVariantType.SetProperty 4699
TypInfo.TTypeData 4758
TypInfo.TTypeData record 4758
TypInfo.TTypeInfo 4759
TypInfo.TTypeInfo record 4759
TypInfo.TTypeKind 4760
TypInfo.TTypeKind enumeration 4760
TypInfo.TTypeKinds 4773
TypInfo.TTypeKinds type 4773
TZCompressionStream class 7907
  about TZCompressionStream class 7907
  CompressionRate 7911
  Create 7909
  Create constructor 7908
  Destroy 7909
  OnProgress 7912
  Read 7910
  Seek 7910
  TZCompressionStream events 7911
  TZCompressionStream Members 7908
  TZCompressionStream methods 7909
  TZCompressionStream properties 7911
  Write 7910
TZDecompressionStream class 7912
  about TZDecompressionStream class 7912
  Create 7913
  Destroy 7913
  OnProgress 7915
  Read 7914
  Seek 7914
  TZDecompressionStream events 7915
  TZDecompressionStream Members 7913
  TZDecompressionStream methods 7914
  Write 7915

U

UDDIBindingKey
  THTTPReqResp class 1291
UDDIOperator
  THTTPReqResp class 1292
UIInitialize
  TSocketForm class 180
UnaryOp
  TCustomVariantType class 5006

Undo
TCustomEdit class 2364
Unicode
TEncoding class 3986
UnicodeString
UnicodeString class 3386
Variant class 3464
UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs)
UnicodeString class 3420
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs)
UnicodeString class 3420
UnicodeString class 3381
!= 3414
[] 3414
[] operator 3414
~UnicodeString 3384
+ 3415
+= 3416
< 3416
<= 3417
= 3418
== 3418
> 3419
>= 3419
about UnicodeString class 3381
BytesOf 3391
ByteType 3391
c_str 3392
cat_printf 3392
cat_sprintf 3393
cat_vprintf 3394
CodePage 3394
Compare 3394
CompareIC 3395
CurrToStr 3395
CurrToStrF 3396
data 3396
Delete 3396
ElementSize 3397
EnsureUnicode 3397
FloatToStrF 3397
FmtLoadStr 3398
Format 3399
FormatFloat 3399
GetRec 3401
GetRec method 3401
Insert 3401
IntToHex 3402
IsDelimiter 3402
IsEmpty 3402
IsLeadSurrogate 3403
IsPathDelimiter 3403
IsTrailSurrogate 3403
LastChar 3404
LastDelimiter 3404
Length 3404
LoadStr 3405
LoadStr method 3405
LoadString 3405
LowerCase 3406
Pos 3406
printf 3406
RefCount 3407
SetLength 3407
sprintf 3408
StringOfChar 3408
SubString 3408
swap 3409
t_str 3409
ThrowIfOutOfRange 3410
ToDouble 3410
ToInt 3410
ToIntDef 3411
Trim 3411
TrimLeft 3411
TrimRight 3412
UnicodeString 3386

UnicodeString __fastcall operator +(const char*, const UnicodeString& rhs) 3420
UnicodeString __fastcall operator +(const wchar_t*, const UnicodeString& rhs) 3420
UnicodeString constructor 3384
UnicodeString enumerations 3387
UnicodeString friends 3420
UnicodeString Members 3382
UnicodeString methods 3389
UnicodeString operators 3413
UnicodeString structures 3387
Unique 3412
UpperCase 3412
vprintf 3413
w_str 3413
Unique
AnsiStringBase class 3202
AnsiStringT template 3229
UnicodeString class 3412
UnitName
TObject class 3380
UnloadThemeData
TThemeServices class 4590
Unlock
TCustomWinSocket class 81
TSoapDataModule class 1229
UnlockNotify
TWebComponentList class 6033
UnlockObject
TCustomDataBlockInterpreter class 197
TDataBlockInterpreter class 214
UnRegisterClient
TSQLConnection class 1606
UnRegisterProvider
TLocalConnection class 4553
TSoapDataModule class 1229
UnselectAll
IShellNameSpace interface 706
IShellNameSpaceDisp interface 714
TShellNameSpace class 479
UnselectedColor
TTabSet class 4507
unsigned char
Variant class 3464
unsigned char*
Variant class 3465
unsigned int
Variant class 3465
unsigned int*
Variant class 3465
unsigned long
Variant class 3465
unsigned long*
Variant class 3466
unsigned short
Variant class 3466
unsigned short*
Variant class 3466
UpCase
IStringTest interface 1335
TFTCompTI class 1316
UpdateActionInfo
TCustomScreenTipsManager class 312
UpdateAssociateToManager
TCustomScreenTipsPopup class 320
UpdateCollectionList
TXMLNodeCollection class 7457
UpdateCount
IXMLNodeList interface 7715
TAbstractNamedVariants class 6132
TValueListStrings class 4890
TWideStrings class 6855
TXMLNodeList class 7476
UpdateEdit
TItemProp class 4781
UpdateFieldsExecuteValues
TCustomAdapter class 5567
UpdateFieldsGetAnyChanges
TCustomAdapter class 5568

UpdateFieldsUpdateValues
TCustomAdapter class 5568
UpdateFieldsValidateValues
TCustomAdapter class 5569
UpdateImageList
TCustomButton class 2266
UpdateImages
TCustomButton class 2266
UpdateIndexDefs
TCustomSQLDataSet class 1551
UpdateOptions
TCustomScreenTipsManager class 312
UpdatePageStatus
DWebBrowserEvents2 interface 633
UpdatePortList
TWSDLHTMLPublish class 7286
UpdateProperties
TConnectionData class 1519
UpdateRecords
TCustomAdapter class 5569
UpdateRecordsGetAnyChanges
TCustomAdapter class 5570
UpdateRecordsUpdateValues
TCustomAdapter class 5570
UpdateRecordsValidateValues
TCustomAdapter class 5571
UpdateRegistry
TComponentFactory class 5168
TPacketInterceptFactory class 221
UpdateResponse
TInvalidWebSession class 6220
TWebSession class 6255
UpdateTarget
TEditAction class 1794
TEditDelete class 1800
TEditPaste class 1803
TEditSelectAll class 1805
TEditUndo class 1807
THelpAction class 1875
THelpContextAction class 1879
TSearchAction class 1899
TSearchFindNext class 1915
TWindowAction class 1935
TWindowClose class 1941
UpdateThemes
TThemeServices class 4590
UpdateTimer
TSocketForm class 172
UpdateTimerTimer
TSocketForm class 180
UpdateTraceCallBack
TSQLMonitor class 1648
UpdateValue
IUpdateValue interface 5892
TAdapterNextPageField class 5386
TAdapterUpdateField class 5431
UpDownClick
TSocketForm class 180
UpperCase
AnsiStringBase class 3202
AnsiStringT template 3230
UnicodeString class 3412
WideString class 3486
url
IDOMParseError interface 7591
URL
EDOMParseError class 7513
ISearch interface 646
ISearchDisp interface 686
THTTPReqResp class 1292
THTTPRIO class 1254
TSoapConnection class 1199
TWebConnection class 264
Use
IHeaderInfo interface 7252
UseDefaultApp
TFileOpen class 1825
UseHandle1

TWinHandleUser class 1331
UseHandle2
TWinHandleUser class 1331
UserID
TCustomEndUserSessionAdapter class 5701
TWebUserItem class 6791
UseRightToLeftAlignment
TCustomButton class 2266
UserItems
TCustomWebUserList class 6784
TWebUserList class 6797
UserList
TBaseEndUserAdapter class 5515
TCustomLoginFormAdapter class 5711
UserListService
TCustomWebAppComponents class 6215
TWebAppComponents class 6249
UserName
TCustomLoginFormAdapter class 5714
THTTPReqResp class 1293
TSoapConnection class 1200
TWebConnection class 265
TWebUserItem class 6791
TWSDLView class 7275
Users
TCustomWebUserList class 6785
UserStatus
TSocketForm class 172
UseSearchCompanion
ISearchAssistantOC3 interface 669
ISearchAssistantOC3Disp interface 677
UseSOAPAdapter
TSoapConnection class 1200
UseUTF8InHeader
THTTPReqResp class 1293
UseZeroMilliseconds
TXSCustomDateTime class 7821
TXSTime class 7860
UTF7
TEncoding class 3986
UTF8
TEncoding class 3987

V

Val
CurrencyBase class 3256
TDateTimeBase class 3342
ValEdit 4777
ValEdit namespace 4777
Classes 4778
Structs, Records, Enums 4892
Types 4894
ValEdit.TDisplayOption 4892
ValEdit.TDisplayOption enumeration 4892
ValEdit.TDisplayOptions 4894
ValEdit.TDisplayOptions type 4894
ValEdit.TGetPickListEvent 4895
ValEdit.TGetPickListEvent type 4895
ValEdit.TItemProp 4778
ValEdit.TItemProp.AssignTo 4780
ValEdit.TItemProp.Create 4779
ValEdit.TItemProp.Destroy 4779
ValEdit.TItemProp.EditMask 4781
ValEdit.TItemProp.EditStyle 4782
ValEdit.TItemProp.HasPickList 4781
ValEdit.TItemProp.KeyDesc 4783
ValEdit.TItemProp.MaxLength 4783
ValEdit.TItemProp.PickList 4783
ValEdit.TItemProp.ReadOnly 4784
ValEdit.TItemProp.UpdateEdit 4781
ValEdit.TItemProps 4895
ValEdit.TItemProps type 4895
ValEdit.TKeyOption 4893
ValEdit.TKeyOption enumeration 4893
ValEdit.TKeyOptions 4895
ValEdit.TKeyOptions type 4895
ValEdit.TOnValidateEvent 4896
ValEdit.TOnValidateEvent type 4896

ValEdit.TValueListEditor 4784
ValEdit.TValueListEditor.AdjustColWidths 4789
ValEdit.TValueListEditor.AdjustRowCount 4789
ValEdit.TValueListEditor.Align 4802
ValEdit.TValueListEditor.Anchors 4805
ValEdit.TValueListEditor.BiDiMode 4806
ValEdit.TValueListEditor.BorderStyle 4806
ValEdit.TValueListEditor.CanEditModify 4790
ValEdit.TValueListEditor.Cells 4806
ValEdit.TValueListEditor.ColCount 4807
ValEdit.TValueListEditor.Color 4807
ValEdit.TValueListEditor.ColWidthsChanged 4790
ValEdit.TValueListEditor.Constraints 4808
ValEdit.TValueListEditor.Create 4788
ValEdit.TValueListEditor.CreateEditor 4790
ValEdit.TValueListEditor.CreateWnd 4791
ValEdit.TValueListEditor.Ctl3D 4809
ValEdit.TValueListEditor.DefaultColWidth 4809
ValEdit.TValueListEditor.DefaultDrawing 4811
ValEdit.TValueListEditor.DefaultRowHeight 4811
ValEdit.TValueListEditor.DeleteRow 4791
ValEdit.TValueListEditor.Destroy 4788
ValEdit.TValueListEditor.DisplayOptions 4812
ValEdit.TValueListEditor.DoExit 4792
ValEdit.TValueListEditor.DoOnValidate 4792
ValEdit.TValueListEditor.DoubleBuffered 4812
ValEdit.TValueListEditor.DragCursor 4814
ValEdit.TValueListEditor.DragKind 4814
ValEdit.TValueListEditor.DragMode 4814
ValEdit.TValueListEditor.DrawCell 4792
ValEdit.TValueListEditor.DropDownRows 4816
ValEdit.TValueListEditor.EditList 4817
ValEdit.TValueListEditor.EditListGetItems 4793
ValEdit.TValueListEditor.Enabled 4817
ValEdit.TValueListEditor.FindRow 4793
ValEdit.TValueListEditor.FixedColor 4817
ValEdit.TValueListEditor.FixedCols 4818
ValEdit.TValueListEditor.Font 4818
ValEdit.TValueListEditor.GetCell 4794
ValEdit.TValueListEditor.GetColCount 4794
ValEdit.TValueListEditor.GetEditLimit 4794
ValEdit.TValueListEditor.GetEditMask 4794
ValEdit.TValueListEditor.GetEditStyle 4795
ValEdit.TValueListEditor.GetEditText 4795
ValEdit.TValueListEditor.GetPickList 4795
ValEdit.TValueListEditor.GetRowCount 4795
ValEdit.TValueListEditor.GridLineWidth 4819
ValEdit.TValueListEditor.InsertRow 4796
ValEdit.TValueListEditor.IsEmptyRow 4796
ValEdit.TValueListEditor.ItemProps 4821
ValEdit.TValueListEditor.KeyDown 4797
ValEdit.TValueListEditor.KeyOptions 4821
ValEdit.TValueListEditor.Keys 4822
ValEdit.TValueListEditor.MouseDown 4797
ValEdit.TValueListEditor.OnClick 4839
ValEdit.TValueListEditor.OnContextPopup 4841
ValEdit.TValueListEditor.OnDblClick 4841
ValEdit.TValueListEditor.OnDragDrop 4842
ValEdit.TValueListEditor.OnDragOver 4845
ValEdit.TValueListEditor.OnDrawCell 4848
ValEdit.TValueListEditor.OnEditButtonClick 4849
ValEdit.TValueListEditor.OnEndDock 4850
ValEdit.TValueListEditor.OnEndDrag 4850
ValEdit.TValueListEditor.OnEnter 4851
ValEdit.TValueListEditor.OnExit 4854
ValEdit.TValueListEditor.OnGetEditMask 4858
ValEdit.TValueListEditor.OnGetEditText 4859
ValEdit.TValueListEditor.OnGetPickList 4860
ValEdit.TValueListEditor.OnKeyDown 4861
ValEdit.TValueListEditor.OnKeyPress 4864
ValEdit.TValueListEditor.OnKeyUp 4865
ValEdit.TValueListEditor.OnMouseActivate 4867
ValEdit.TValueListEditor.OnMouseDown 4868
ValEdit.TValueListEditor.OnMouseEnter 4870
ValEdit.TValueListEditor.OnMouseLeave 4871
ValEdit.TValueListEditor.OnMouseMove 4871
ValEdit.TValueListEditor.OnMouseUp 4873
ValEdit.TValueListEditor.OnMouseWheelDown 4875

ValEdit.TValueListEditor.OnMouseWheelUp 4876
ValEdit.TValueListEditor.OnRowMoved 4876
ValEdit.TValueListEditor.OnSelectCell 4877
ValEdit.TValueListEditor.OnSetEditText 4879
ValEdit.TValueListEditor.OnStartDock 4879
ValEdit.TValueListEditor.OnStartDrag 4880
ValEdit.TValueListEditor.OnStringsChange 4881
ValEdit.TValueListEditor.OnStringsChanging 4881
ValEdit.TValueListEditor.OnTopLeftChanged 4882
ValEdit.TValueListEditor.OnValidate 4882
ValEdit.TValueListEditor.Options 4822
ValEdit.TValueListEditor.ParentBiDiMode 4824
ValEdit.TValueListEditor.ParentColor 4824
ValEdit.TValueListEditor.ParentCtl3D 4825
ValEdit.TValueListEditor.ParentDoubleBuffered 4825
ValEdit.TValueListEditor.ParentFont 4825
ValEdit.TValueListEditor.ParentShowHint 4826
ValEdit.TValueListEditor.PopupMenu 4826
ValEdit.TValueListEditor.Refresh 4798
ValEdit.TValueListEditor.Resize 4798
ValEdit.TValueListEditor.RestoreCurrentRow 4798
ValEdit.TValueListEditor.RowCount 4831
ValEdit.TValueListEditor.RowMoved 4799
ValEdit.TValueListEditor.ScrollBars 4832
ValEdit.TValueListEditor.SelectCell 4799
ValEdit.TValueListEditor.SetCell 4799
ValEdit.TValueListEditor.SetEditText 4800
ValEdit.TValueListEditor.SetStrings 4800
ValEdit.TValueListEditor.ShowHint 4832
ValEdit.TValueListEditor.Strings 4834
ValEdit.TValueListEditor.StringsChanging 4800
ValEdit.TValueListEditor.TabOrder 4834
ValEdit.TValueListEditor.TitleCaptions 4835
ValEdit.TValueListEditor.TitleCaptionsStored 4800
ValEdit.TValueListEditor.Values 4835
ValEdit.TValueListEditor.Visible 4836
ValEdit.TValueListEditor.VisibleColCount 4837
ValEdit.TValueListEditor.VisibleRowCount 4838
ValEdit.TValueListStrings 4882
ValEdit.TValueListStrings.Assign 4884
ValEdit.TValueListStrings.Changed 4887
ValEdit.TValueListStrings.Changing 4887
ValEdit.TValueListStrings.Clear 4887
ValEdit.TValueListStrings.Create 4883
ValEdit.TValueListStrings.CustomSort 4888
ValEdit.TValueListStrings.Delete 4888
ValEdit.TValueListStrings.Destroy 4884
ValEdit.TValueListStrings.Exchange 4889
ValEdit.TValueListStrings.FindItemProp 4889
ValEdit.TValueListStrings.InsertItem 4889
ValEdit.TValueListStrings.ItemProps 4890
ValEdit.TValueListStrings.KeyIsValid 4890
ValEdit.TValueListStrings.Put 4890
ValEdit.TValueListStrings.UpdateCount 4890
validate
IDOMParseOptions interface 7595
ValidatePage
TCustomPageDispatcher class 6195
ValidatePassword
TCustomLoginFormAdapter class 5712
ValidateUser
IWebUserList interface 1117
TCustomWebUserList class 6784
ValidateUserName
TCustomLoginFormAdapter class 5712
ValidateValue
IValidateValue interface 5893
TAdapterUpdateField class 5431
VAlign
TAdapterActionButton class 6324
TAdapterCommandColumn class 6337
TAdapterDataDisplay class 6363
TAdapterDisplayColumn class 6370
TAdapterDisplayField class 6380
TAdapterEditColumn class 6396
TCustomAdapterActionButton class 6442
TCustomAdapterCommandColumn class 6453
value

IDOMAttr interface 7525
Value
IAdapterHiddenField interface 1003
IWebGetFieldValue interface 1092
TAdapterNamedField class 5369
TAdapterUpdateField class 5432
ValueCount
TCustomAdapterMultiValueField class 5653
ValueFromIndex
TWideStrings class 6855
Values
IWebSessionValues interface 1114
TAbstractNamedVariants class 6133
TCustomAdapterMultiValueField class 5654
TValueListEditor class 4835
TWebSessionImpl class 6771
TWideStrings class 6856
ValuesList
TAdapterBooleanField class 5229
TAdapterField class 5274
TAdapterImageField class 5316
TAdapterMultiValueField class 5340
TAdapterUpdateField class 5432
VarCmplx 4899
VarCmplx namespace 4899
Functions 4901
Variables 4944
VarCmplx.ComplexNumberDefuzzAtZero 4944
VarCmplx.ComplexNumberDefuzzAtZero variable 4944
VarCmplx.ComplexNumberSymbol 4944
VarCmplx.ComplexNumberSymbol variable 4944
VarCmplx.ComplexNumberSymbolBeforeImaginary 4945
VarCmplx.ComplexNumberSymbolBeforeImaginary variable 4945
VarCmplx.VarAsComplex 4902
VarCmplx.VarAsComplex function 4902
VarCmplx.VarComplex 4903
VarCmplx.VarComplex function 4903
VarCmplx.VarComplexAbs 4903
VarCmplx.VarComplexAbs function 4903
VarCmplx.VarComplexAbsSqr 4904
VarCmplx.VarComplexAbsSqr function 4904
VarCmplx.VarComplexAngle 4905
VarCmplx.VarComplexAngle function 4905
VarCmplx.VarComplexArcCos 4905
VarCmplx.VarComplexArcCos function 4905
VarCmplx.VarComplexArcCosH 4906
VarCmplx.VarComplexArcCosH function 4906
VarCmplx.VarComplexArcCot 4907
VarCmplx.VarComplexArcCot function 4907
VarCmplx.VarComplexArcCotH 4908
VarCmplx.VarComplexArcCotH function 4908
VarCmplx.VarComplexArcCsc 4909
VarCmplx.VarComplexArcCsc function 4909
VarCmplx.VarComplexArcCscH 4911
VarCmplx.VarComplexArcCscH function 4911
VarCmplx.VarComplexArcSec 4912
VarCmplx.VarComplexArcSec function 4912
VarCmplx.VarComplexArcSecH 4913
VarCmplx.VarComplexArcSecH function 4913
VarCmplx.VarComplexArcSin 4914
VarCmplx.VarComplexArcSin function 4914
VarCmplx.VarComplexArcSinH 4915
VarCmplx.VarComplexArcSinH function 4915
VarCmplx.VarComplexArcTan 4916
VarCmplx.VarComplexArcTan function 4916
VarCmplx.VarComplexArcTanH 4917
VarCmplx.VarComplexArcTanH function 4917
VarCmplx.VarComplexConjugate 4918
VarCmplx.VarComplexConjugate function 4918
VarCmplx.VarComplexCos 4919
VarCmplx.VarComplexCos function 4919
VarCmplx.VarComplexCosH 4920
VarCmplx.VarComplexCosH function 4920
VarCmplx.VarComplexCot 4921
VarCmplx.VarComplexCot function 4921
VarCmplx.VarComplexCotH 4922
VarCmplx.VarComplexCotH function 4922
VarCmplx.VarComplexCreate 4923

VarCmplx.VarComplexCreate function 4923
VarCmplx.VarComplexCsc 4924
VarCmplx.VarComplexCsc function 4924
VarCmplx.VarComplexCscH 4925
VarCmplx.VarComplexCscH function 4925
VarCmplx.VarComplexExp 4926
VarCmplx.VarComplexExp function 4926
VarCmplx.VarComplexFromPolar 4927
VarCmplx.VarComplexFromPolar function 4927
VarCmplx.VarComplexInverse 4927
VarCmplx.VarComplexInverse function 4927
VarCmplx.VarComplexLn 4928
VarCmplx.VarComplexLn function 4928
VarCmplx.VarComplexLog10 4929
VarCmplx.VarComplexLog10 function 4929
VarCmplx.VarComplexLog2 4929
VarCmplx.VarComplexLog2 function 4929
VarCmplx.VarComplexLogN 4930
VarCmplx.VarComplexLogN function 4930
VarCmplx.VarComplexPower 4930
VarCmplx.VarComplexPower function 4930
VarCmplx.VarComplexSec 4931
VarCmplx.VarComplexSec function 4931
VarCmplx.VarComplexSecH 4932
VarCmplx.VarComplexSecH function 4932
VarCmplx.VarComplexSign 4933
VarCmplx.VarComplexSign function 4933
VarCmplx.VarComplexSimplify 4934
VarCmplx.VarComplexSimplify function 4934
VarCmplx.VarComplexSin 4934
VarCmplx.VarComplexSin function 4934
VarCmplx.VarComplexSinH 4935
VarCmplx.VarComplexSinH function 4935
VarCmplx.VarComplexSqr 4936
VarCmplx.VarComplexSqr function 4936
VarCmplx.VarComplexSqrt 4937
VarCmplx.VarComplexSqrt function 4937
VarCmplx.VarComplexTan 4938
VarCmplx.VarComplexTan function 4938
VarCmplx.VarComplexTanH 4939
VarCmplx.VarComplexTanH function 4939
VarCmplx.VarComplexTimesImaginary 4940
VarCmplx.VarComplexTimesImaginary function 4940
VarCmplx.VarComplexTimesNegI 4940
VarCmplx.VarComplexTimesNegI function 4940
VarCmplx.VarComplexTimesPosI 4941
VarCmplx.VarComplexTimesPosI function 4941
VarCmplx.VarComplexTimesReal 4941
VarCmplx.VarComplexTimesReal function 4941
VarCmplx.VarComplexToPolar 4942
VarCmplx.VarComplexToPolar function 4942
VarCmplx.VarIsComplex 4942
VarCmplx.VarIsComplex function 4942
VarConv 4947
VarConv namespace 4947
    Functions 4948
VarConv.VarAsConvert 4948
VarConv.VarAsConvert function 4948
VarConv.VarConvert 4949
VarConv.VarConvert function 4949
VarConv.VarConvertCreate 4950
VarConv.VarConvertCreate function 4950
VarConv.VarIsConvert 4951
VarConv.VarIsConvert function 4951
VarDataCast
    TCustomVariantType class 5006
VarDataCastTo
    TCustomVariantType class 5007
VarDataCastToOleStr
    TCustomVariantType class 5007
VarDataClear
    TCustomVariantType class 5008
VarDataCopy
    TCustomVariantType class 5008
VarDataCopyNoInd
    TCustomVariantType class 5009
VarDataFromOleStr
    TCustomVariantType class 5009

VarDataFromStr
TCustomVariantType class 5010
VarDataInit
TCustomVariantType class 5010
VarDataIsArray
TCustomVariantType class 5010
VarDataIsByRef
TCustomVariantType class 5011
VarDataIsEmptyParam
TCustomVariantType class 5011
VarDataIsFloat
TCustomVariantType class 5011
VarDataIsNumeric
TCustomVariantType class 5012
VarDataIsOrdinal
TCustomVariantType class 5012
VarDataIsStr
TCustomVariantType class 5013
VarDataToStr
TCustomVariantType class 5013
VarHlpr 4953
VarHlpr namespace 4953
Functions 4955
VarHlpr.VariantAdd 4955
VarHlpr.VariantAdd function 4955
VarHlpr.VariantAdd2 4956
VarHlpr.VariantAdd2 function 4956
VarHlpr.VariantAnd 4956
VarHlpr.VariantAnd function 4956
VarHlpr.VariantAnd2 4957
VarHlpr.VariantAnd2 function 4957
VarHlpr.VariantArrayRedim 4957
VarHlpr.VariantArrayRedim function 4957
VarHlpr.VariantCast 4958
VarHlpr.VariantCast function 4958
VarHlpr.VariantClear 4958
VarHlpr.VariantClear function 4958
VarHlpr.VariantCpy 4959
VarHlpr.VariantCpy function 4959
VarHlpr.VariantDiv 4959
VarHlpr.VariantDiv function 4959
VarHlpr.VariantDiv2 4960
VarHlpr.VariantDiv2 function 4960
VarHlpr.VariantFromUnicodeString 4960
VarHlpr.VariantFromUnicodeString function 4960
VarHlpr.VariantGetElement 4961
VarHlpr.VariantGetElement function 4961
VarHlpr.VariantMod 4962
VarHlpr.VariantMod function 4962
VarHlpr.VariantMod2 4962
VarHlpr.VariantMod2 function 4962
VarHlpr.VariantMul 4963
VarHlpr.VariantMul function 4963
VarHlpr.VariantMul2 4963
VarHlpr.VariantMul2 function 4963
VarHlpr.VariantNeg 4964
VarHlpr.VariantNeg function 4964
VarHlpr.VariantNot 4964
VarHlpr.VariantNot function 4964
VarHlpr.VariantOr 4964
VarHlpr.VariantOr function 4964
VarHlpr.VariantOr2 4965
VarHlpr.VariantOr2 function 4965
VarHlpr.VariantPutElement 4966
VarHlpr.VariantPutElement function 4966
VarHlpr.VariantShl 4966
VarHlpr.VariantShl function 4966
VarHlpr.VariantShl2 4967
VarHlpr.VariantShl2 function 4967
VarHlpr.VariantShr 4967
VarHlpr.VariantShr function 4967
VarHlpr.VariantShr2 4968
VarHlpr.VariantShr2 function 4968
VarHlpr.VariantSub 4968
VarHlpr.VariantSub function 4968
VarHlpr.VariantSub2 4969
VarHlpr.VariantSub2 function 4969
VarHlpr.VariantToUnicodeString 4969

VarHlpr.VariantToUnicodeString function 4969
VarHlpr.VariantXor 4970
VarHlpr.VariantXor function 4970
VarHlpr.VariantXor2 4970
VarHlpr.VariantXor2 function 4970
VariablesInUse
TAdapterSubComponent class 5419
Variant
Variant class 3426
Variant class 3421
- 3439
- operator 3439
! 3439
!= 3440
!= operator 3440
% 3441
% operator 3440
%= 3441
& 3442
& operator 3442
&= 3442
* 3443
* operator 3443
*= 3443
/ 3444
/ operator 3444
/= 3444
^ 3445
^ operator 3445
^= 3445
__int64 3446
| 3446
| operator 3446
|= 3446
~Variant 3424
+ 3447
+ operator 3447
+= 3447
< 3448
< operator 3448
<< 3449
<< operator 3448
<<= 3449
<= 3450
<= operator 3449
= 3453
-= 3454
= operator 3450
== 3455
== operator 3455
> 3456
> operator 3456
>= 3457
>= operator 3456
>> 3457
>> operator 3457
>>= 3458
about Variant class 3421
AnsiString 3458
ArrayDimCount 3429
ArrayHighBound 3429
ArrayLock 3429
ArrayLowBound 3430
ArrayRedim 3430
ArrayUnlock 3430
AsType 3431
bool 3458
ChangeType 3431
char 3459
Clear 3431
Currency 3459
Currency* 3459
double 3460
double* 3460
Empty 3431
float 3460
float* 3460
GetBaseVariant 3432

GetElement 3432
GetElement method 3432
GetError 3433
IDispatch* 3461
IInterface* 3461
int 3461
int* 3461
IsArray 3433
IsEmpty 3433
IsNull 3434
IUnknown* 3462
long 3462
long* 3462
NoParam 3434
PutElement 3435
PutElement method 3434
Sanitized 3435
SetError 3435
short 3462
short* 3463
signed char 3463
signed char* 3463
TDateTime 3463
TDateTime* 3464
Type 3436
UnicodeString 3464
unsigned char 3464
unsigned char* 3465
unsigned int 3465
unsigned int* 3465
unsigned long 3465
unsigned long* 3466
unsigned short 3466
unsigned short* 3466
Variant 3426
Variant constructor 3424
Variant Members 3422
Variant methods 3428
Variant operators 3437
WideString 3466
Variant::-
OleVariant class 3288
Variant::!
OleVariant class 3288
Variant::%=
OleVariant class 3288
Variant::&=
OleVariant class 3288
Variant::*=
OleVariant class 3289
Variant::/=
OleVariant class 3289
Variant::^=
OleVariant class 3289
Variant::__int64
OleVariant class 3289
Variant::|=
OleVariant class 3289
Variant::+=
OleVariant class 3290
Variant::<<=
OleVariant class 3290
Variant::-=
OleVariant class 3290
Variant::>>=
OleVariant class 3290
Variant::AnsiString
OleVariant class 3291
Variant::ArrayDimCount
OleVariant class 3291
Variant::ArrayHighBound
OleVariant class 3291
Variant::ArrayLock
OleVariant class 3291
Variant::ArrayLowBound
OleVariant class 3291
Variant::ArrayRedim
OleVariant class 3292

Variant::ArrayUnlock
OleVariant class 3292
Variant::AsType
OleVariant class 3292
Variant::bool
OleVariant class 3292
Variant::Byte*
OleVariant class 3293
Variant::ChangeType
OleVariant class 3293
Variant::char
OleVariant class 3293
Variant::Clear
OleVariant class 3293
Variant::Currency
OleVariant class 3293
Variant::Currency*
OleVariant class 3294
Variant::double
OleVariant class 3294
Variant::double*
OleVariant class 3294
Variant::float
OleVariant class 3294
Variant::float*
OleVariant class 3295
Variant::GetElement
OleVariant class 3295
Variant::GetError
OleVariant class 3295
Variant::IDispatch*
OleVariant class 3295
Variant::IInterface*
OleVariant class 3295
Variant::int
OleVariant class 3296
Variant::int*
OleVariant class 3296
Variant::IsArray
OleVariant class 3296
Variant::IsEmpty
OleVariant class 3296
Variant::IsNull
OleVariant class 3297
Variant::IUnknown*
OleVariant class 3297
Variant::long
OleVariant class 3297
Variant::long*
OleVariant class 3297
Variant::PutElement
OleVariant class 3297
Variant::SetError
OleVariant class 3298
Variant::short
OleVariant class 3298
Variant::short*
OleVariant class 3298
Variant::signed char
OleVariant class 3298
Variant::signed char*
OleVariant class 3299
Variant::TDateTime
OleVariant class 3299
Variant::TDateTime*
OleVariant class 3299
Variant::Type
OleVariant class 3299
Variant::UnicodeString
OleVariant class 3299
Variant::unsigned char
OleVariant class 3300
Variant::unsigned int
OleVariant class 3300
Variant::unsigned int*
OleVariant class 3300
Variant::unsigned long
OleVariant class 3300

Variant::unsigned short
OleVariant class 3301
Variant::unsigned short*
OleVariant class 3301
Variant::WideString
OleVariant class 3301
Variants 4973
TAbstractNamedVariants class 6133
Variants namespace 4973
Classes 4978
Functions 5027
Interfaces 5022
Structs, Records, Enums 5094
Types 5097
Variables 5100
Variants._DispInvoke 5029
Variants._DispInvoke function 5029
Variants._OleVarFromInt 5030
Variants._OleVarFromInt function 5030
Variants._OleVarFromInt64 5030
Variants._OleVarFromInt64 function 5030
Variants._OleVarFromLStr 5031
Variants._OleVarFromLStr function 5031
Variants._OleVarFromPStr 5031
Variants._OleVarFromPStr function 5031
Variants._OleVarFromUInt64 5031
Variants._OleVarFromUInt64 function 5031
Variants._OleVarFromUStr 5032
Variants._OleVarFromUStr function 5032
Variants._OleVarFromVar 5032
Variants._OleVarFromVar function 5032
Variants._VarAdd 5032
Variants._VarAdd function 5032
Variants._VarAddRef 5033
Variants._VarAddRef function 5033
Variants._VarAnd 5033
Variants._VarAnd function 5033
Variants._VarArrayGet 5034
Variants._VarArrayGet function 5034
Variants._VarArrayPut 5034
Variants._VarArrayPut function 5034
Variants._VarArrayRedim 5034
Variants._VarArrayRedim function 5034
Variants._VarCast 5035
Variants._VarCast function 5035
Variants._VarCastOle 5035
Variants._VarCastOle function 5035
Variants._VarClear 5035
Variants._VarClear function 5035
Variants._VarClr 5036
Variants._VarClr function 5036
Variants._VarCmpEQ 5036
Variants._VarCmpEQ function 5036
Variants._VarCmpGE 5037
Variants._VarCmpGE function 5037
Variants._VarCmpGT 5037
Variants._VarCmpGT function 5037
Variants._VarCmpLE 5037
Variants._VarCmpLE function 5037
Variants._VarCmpLT 5038
Variants._VarCmpLT function 5038
Variants._VarCmpNE 5038
Variants._VarCmpNE function 5038
Variants._VarCopy 5038
Variants._VarCopy function 5038
Variants._VarCopyNoInd 5039
Variants._VarCopyNoInd function 5039
Variants._VarDiv 5039
Variants._VarDiv function 5039
Variants._VarFromBool 5040
Variants._VarFromBool function 5040
Variants._VarFromCurr 5040
Variants._VarFromCurr function 5040
Variants._VarFromDisp 5040
Variants._VarFromDisp function 5040
Variants._VarFromDynArray 5041
Variants._VarFromDynArray function 5041
Variants._VarFromInt 5041

Variants._VarFromInt function 5041
Variants._VarFromInt64 5041
Variants._VarFromInt64 function 5041
Variants._VarFromIntf 5042
Variants._VarFromIntf function 5042
Variants._VarFromLStr 5042
Variants._VarFromLStr function 5042
Variants._VarFromPStr 5043
Variants._VarFromPStr function 5043
Variants._VarFromReal 5043
Variants._VarFromReal function 5043
Variants._VarFromTDateTime 5043
Variants._VarFromTDateTime function 5043
Variants._VarFromUInt64 5044
Variants._VarFromUInt64 function 5044
Variants._VarFromUStr 5044
Variants._VarFromUStr function 5044
Variants._VarFromWChar 5044
Variants._VarFromWChar function 5044
Variants._VarFromWStr 5045
Variants._VarFromWStr function 5045
Variants._VarMod 5045
Variants._VarMod function 5045
Variants._VarMul 5046
Variants._VarMul function 5046
Variants._VarNeg 5046
Variants._VarNeg function 5046
Variants._VarNot 5046
Variants._VarNot function 5046
Variants._VarOr 5047
Variants._VarOr function 5047
Variants._VarRDiv 5047
Variants._VarRDiv function 5047
Variants._VarShl 5047
Variants._VarShl function 5047
Variants._VarShr 5048
Variants._VarShr function 5048
Variants._VarSub 5048
Variants._VarSub function 5048
Variants._VarToBool 5049
Variants._VarToBool function 5049
Variants._VarToCurrency 5049
Variants._VarToCurrency function 5049
Variants._VarToDisp 5049
Variants._VarToDisp function 5049
Variants._VarToDynArray 5050
Variants._VarToDynArray function 5050
Variants._VarToInt64 5050
Variants._VarToInt64 function 5050
Variants._VarToInteger 5050
Variants._VarToInteger function 5050
Variants._VarToIntf 5051
Variants._VarToIntf function 5051
Variants._VarToLStr 5051
Variants._VarToLStr function 5051
Variants._VarToPStr 5052
Variants._VarToPStr function 5052
Variants._VarToReal 5052
Variants._VarToReal function 5052
Variants._VarToUInt64 5052
Variants._VarToUInt64 function 5052
Variants._VarToUStr 5053
Variants._VarToUStr function 5053
Variants._VarToWStr 5053
Variants._VarToWStr function 5053
Variants._VarXor 5053
Variants._VarXor function 5053
Variants.BooleanToStringRule 5100
Variants.BooleanToStringRule variable 5100
Variants.ChangeAnyProc 5101
Variants.ChangeAnyProc variable 5101
Variants.ClearAnyProc 5101
Variants.ClearAnyProc variable 5101
Variants.DynArrayFromVariant 5054
Variants.DynArrayFromVariant function 5054
Variants.DynArrayToVariant 5054
Variants.DynArrayToVariant function 5054
Variants.EmptyParam 5102

Variants.EmptyParam variable 5102
Variants.EVariantArrayCreateError 4978
Variants.EVariantArrayLockedError 4979
Variants.EVariantBadIndexError 4980
Variants.EVariantBadVarTypeError 4980
Variants.EVariantDispatchError 4981
Variants.EVariantInvalidArgError 4981
Variants.EVariantInvalidNullOpError 4982
Variants.EVariantInvalidOpError 4983
Variants.EVariantNotAnArrayError 4983
Variants.EVariantNotImplError 4984
Variants.EVariantOutOfMemoryError 4984
Variants.EVariantOverflowError 4985
Variants.EVariantRangeCheckError 4986
Variants.EVariantTypeCastError 4986
Variants.EVariantUnexpectedError 4987
Variants.FindCustomVariantType 5055
Variants.FindCustomVariantType function 5055
Variants.FindVarData 5055
Variants.FindVarData function 5055
Variants.HandleConversionException 5056
Variants.HandleConversionException function 5056
Variants.IVarInstanceReference 5022
Variants.IVarInstanceReference.GetInstance 5023
Variants.IVarInvokeable 5023
Variants.IVarInvokeable.DoFunction 5024
Variants.IVarInvokeable.DoProcedure 5025
Variants.IVarInvokeable.GetProperty 5025
Variants.IVarInvokeable.SetProperty 5026
Variants.Null 5056
Variants.Null function 5056
Variants.NullAsStringValue 5102
Variants.NullAsStringValue variable 5102
Variants.NullEqualityRule 5103
Variants.NullEqualityRule variable 5103
Variants.NullMagnitudeRule 5103
Variants.NullMagnitudeRule variable 5103
Variants.NullStrictConvert 5104
Variants.NullStrictConvert variable 5104
Variants.OleVariantInt64AsDouble 5105
Variants.OleVariantInt64AsDouble variable 5105
Variants.PackVarCreation 5105
Variants.PackVarCreation variable 5105
Variants.RefAnyProc 5106
Variants.RefAnyProc variable 5106
Variants.TAnyProc 5097
Variants.TAnyProc type 5097
Variants.TBooleanToStringRule 5094
Variants.TBooleanToStringRule enumeration 5094
Variants.TCustomVariantType 4987
Variants.TCustomVariantType._AddRef 4993
Variants.TCustomVariantType._Release 4993
Variants.TCustomVariantType.BinaryOp 4994
Variants.TCustomVariantType.Cast 4995
Variants.TCustomVariantType.CastTo 4995
Variants.TCustomVariantType.CastToOle 4996
Variants.TCustomVariantType.Clear 4996
Variants.TCustomVariantType.Compare 4997
Variants.TCustomVariantType.CompareOp 4998
Variants.TCustomVariantType.Copy 4999
Variants.TCustomVariantType.Create 4991
Variants.TCustomVariantType.Destroy 4991
Variants.TCustomVariantType.DispInvoke 4999
Variants.TCustomVariantType.IsClear 5000
Variants.TCustomVariantType.LeftPromotion 5000
Variants.TCustomVariantType.OlePromotion 5001
Variants.TCustomVariantType.QueryInterface 5002
Variants.TCustomVariantType.RaiseCastError 5002
Variants.TCustomVariantType.RaiseDispError 5003
Variants.TCustomVariantType.RaiseInvalidOp 5003
Variants.TCustomVariantType.RightPromotion 5004
Variants.TCustomVariantType.SimplisticClear 5005
Variants.TCustomVariantType.SimplisticCopy 5005
Variants.TCustomVariantType.UnaryOp 5006
Variants.TCustomVariantType.VarDataCast 5006
Variants.TCustomVariantType.VarDataCastTo 5007
Variants.TCustomVariantType.VarDataCastToOleStr 5007
Variants.TCustomVariantType.VarDataClear 5008

Variants.TCustomVariantType.VarDataCopy 5008
Variants.TCustomVariantType.VarDataCopyNoInd 5009
Variants.TCustomVariantType.VarDataFromOleStr 5009
Variants.TCustomVariantType.VarDataFromStr 5010
Variants.TCustomVariantType.VarDataInit 5010
Variants.TCustomVariantType.VarDataIsArray 5010
Variants.TCustomVariantType.VarDataIsByRef 5011
Variants.TCustomVariantType.VarDataIsEmptyParam 5011
Variants.TCustomVariantType.VarDataIsFloat 5011
Variants.TCustomVariantType.VarDataIsNumeric 5012
Variants.TCustomVariantType.VarDataIsOrdinal 5012
Variants.TCustomVariantType.VarDataIsStr 5013
Variants.TCustomVariantType.VarDataToStr 5013
Variants.TCustomVariantType.VarType 5014
Variants.TCustomVariantTypeClass 5097
Variants.TCustomVariantTypeClass type 5097
Variants.TInvokeableVariantType 5014
Variants.TInvokeableVariantType.DispInvoke 5018
Variants.TInvokeableVariantType.DoFunction 5018
Variants.TInvokeableVariantType.DoProcedure 5019
Variants.TInvokeableVariantType.FixupIdent 5020
Variants.TInvokeableVariantType.GetProperty 5020
Variants.TInvokeableVariantType.SetProperty 5021
Variants.TNullCompareRule 5094
Variants.TNullCompareRule enumeration 5094
Variants.TVarCompareResult 5095
Variants.TVarCompareResult enumeration 5095
Variants.TVarDataArray 5098
Variants.TVarDataArray type 5098
Variants.TVarDispProc 5098
Variants.TVarDispProc type 5098
Variants.TVariantRelationship 5096
Variants.TVariantRelationship enumeration 5096
Variants.Unassigned 5058
Variants.Unassigned function 5058
Variants.VarArrayAsPSafeArray 5059
Variants.VarArrayAsPSafeArray function 5059
Variants.VarArrayCreate 5059
Variants.VarArrayCreate function 5059
Variants.VarArrayCreateError 5061
Variants.VarArrayCreateError function 5061
Variants.VarArrayDimCount 5061
Variants.VarArrayDimCount function 5061
Variants.VarArrayGet 5063
Variants.VarArrayGet function 5063
Variants.VarArrayHighBound 5065
Variants.VarArrayHighBound function 5065
Variants.VarArrayLock 5067
Variants.VarArrayLock function 5067
Variants.VarArrayLowBound 5067
Variants.VarArrayLowBound function 5067
Variants.VarArrayOf 5069
Variants.VarArrayOf function 5069
Variants.VarArrayPut 5069
Variants.VarArrayPut function 5069
Variants.VarArrayRef 5071
Variants.VarArrayRef function 5071
Variants.VarArrayUnlock 5072
Variants.VarArrayUnlock function 5072
Variants.VarAsError 5072
Variants.VarAsError function 5072
Variants.VarAsType 5073
Variants.VarAsType function 5073
Variants.VarCastError 5073
Variants.VarCastError function 5073
Variants.VarCheckEmpty 5074
Variants.VarCheckEmpty function 5074
Variants.VarCompareValue 5074
Variants.VarCompareValue function 5074
Variants.VarCopyNoInd 5075
Variants.VarCopyNoInd function 5075
Variants.VarDispProc 5106
Variants.VarDispProc variable 5106
Variants.VarEnsureRange 5076
Variants.VarEnsureRange function 5076
Variants.VarFromDateTime 5076
Variants.VarFromDateTime function 5076
Variants.VarInRange 5077

Variants.VarInRange function 5077
Variants.VarInvalidNullOp 5077
Variants.VarInvalidNullOp function 5077
Variants.VarInvalidOp 5078
Variants.VarInvalidOp function 5078
Variants.VarIsArray 5078
Variants.VarIsArray function 5078
Variants.VarIsByRef 5079
Variants.VarIsByRef function 5079
Variants.VarIsClear 5079
Variants.VarIsClear function 5079
Variants.VarIsCustom 5080
Variants.VarIsCustom function 5080
Variants.VarIsEmpty 5080
Variants.VarIsEmpty function 5080
Variants.VarIsEmptyParam 5081
Variants.VarIsEmptyParam function 5081
Variants.VarIsError 5082
Variants.VarIsError function 5082
Variants.VarIsFloat 5082
Variants.VarIsFloat function 5082
Variants.VarIsNull 5083
Variants.VarIsNull function 5083
Variants.VarIsNumeric 5083
Variants.VarIsNumeric function 5083
Variants.VarIsOrdinal 5084
Variants.VarIsOrdinal function 5084
Variants.VarIsStr 5084
Variants.VarIsStr function 5084
Variants.VarIsType 5085
Variants.VarIsType function 5085
Variants.VarOverflowError 5085
Variants.VarOverflowError function 5085
Variants.VarRangeCheckError 5086
Variants.VarRangeCheckError function 5086
Variants.VarResultCheck 5086
Variants.VarResultCheck function 5086
Variants.VarSameValue 5087
Variants.VarSameValue function 5087
Variants.VarSupports 5087
Variants.VarSupports function 5087
Variants.VarToDateTime 5088
Variants.VarToDateTime function 5088
Variants.VarToStr 5088
Variants.VarToStr function 5088
Variants.VarToStrDef 5089
Variants.VarToStrDef function 5089
Variants.VarToWideStr 5089
Variants.VarToWideStr function 5089
Variants.VarToWideStrDef 5090
Variants.VarToWideStrDef function 5090
Variants.VarType 5090
Variants.VarType function 5090
Variants.VarTypeAsText 5092
Variants.VarTypeAsText function 5092
Variants.VarTypeIsValidArrayType 5092
Variants.VarTypeIsValidArrayType function 5092
Variants.VarTypeIsValidElementType 5093
Variants.VarTypeIsValidElementType function 5093
VarType
    TCustomVariantType class 5014
VarUtils 5107
VarUtils namespace 5107
    Classes 5110
    Constants 5149
    Functions 5115
    Structs, Records, Enums 5132
    Variables 5133
VarUtils.ARR_DISPATCH 5150
VarUtils.ARR_DISPATCH constant 5150
VarUtils.ARR_FIXEDSIZE 5150
VarUtils.ARR_FIXEDSIZE constant 5150
VarUtils.ARR_NONE 5151
VarUtils.ARR_NONE constant 5151
VarUtils.ARR_OLESTR 5151
VarUtils.ARR_OLESTR constant 5151
VarUtils.ARR_UNKNOWN 5152
VarUtils.ARR_UNKNOWN constant 5152

VarUtils.ARR_VARIANT 5152
VarUtils.ARR_VARIANT constant 5152
VarUtils.CMaxArrayVarType 5153
VarUtils.CMaxArrayVarType constant 5153
VarUtils.CMinArrayVarType 5153
VarUtils.CMinArrayVarType constant 5153
VarUtils.CVarTypeToElementInfo 5154
VarUtils.CVarTypeToElementInfo constant 5154
VarUtils.ESafeArrayBoundsError 5110
VarUtils.ESafeArrayCreateError 5110
VarUtils.ESafeArrayError 5111
VarUtils.ESafeArrayError.CreateHResult 5113
VarUtils.ESafeArrayError.ErrorCode 5113
VarUtils.ESafeArrayLockedError 5114
VarUtils.GenericOperations 5155
VarUtils.GenericOperations constant 5155
VarUtils.GenericSafeArrays 5155
VarUtils.GenericSafeArrays constant 5155
VarUtils.GenericSafeArrayUsesLibC 5156
VarUtils.GenericSafeArrayUsesLibC constant 5156
VarUtils.GenericVariants 5156
VarUtils.GenericVariants constant 5156
VarUtils.GenericVarUtils 5157
VarUtils.GenericVarUtils constant 5157
VarUtils.SafeArrayAccessData 5115
VarUtils.SafeArrayAccessData function 5115
VarUtils.SafeArrayAllocData 5116
VarUtils.SafeArrayAllocData function 5116
VarUtils.SafeArrayAllocDescriptor 5116
VarUtils.SafeArrayAllocDescriptor function 5116
VarUtils.SafeArrayCheck 5117
VarUtils.SafeArrayCheck function 5117
VarUtils.SafeArrayCopy 5117
VarUtils.SafeArrayCopy function 5117
VarUtils.SafeArrayCopyData 5118
VarUtils.SafeArrayCopyData function 5118
VarUtils.SafeArrayCreate 5118
VarUtils.SafeArrayCreate function 5118
VarUtils.SafeArrayDestroy 5119
VarUtils.SafeArrayDestroy function 5119
VarUtils.SafeArrayDestroyData 5120
VarUtils.SafeArrayDestroyData function 5120
VarUtils.SafeArrayDestroyDescriptor 5120
VarUtils.SafeArrayDestroyDescriptor function 5120
VarUtils.SafeArrayError 5120
VarUtils.SafeArrayError function 5120
VarUtils.SafeArrayGetDim 5121
VarUtils.SafeArrayGetDim function 5121
VarUtils.SafeArrayGetElement 5121
VarUtils.SafeArrayGetElement function 5121
VarUtils.SafeArrayGetElemSize 5122
VarUtils.SafeArrayGetElemSize function 5122
VarUtils.SafeArrayGetLBound 5122
VarUtils.SafeArrayGetLBound function 5122
VarUtils.SafeArrayGetUBound 5123
VarUtils.SafeArrayGetUBound function 5123
VarUtils.SafeArrayLock 5123
VarUtils.SafeArrayLock function 5123
VarUtils.SafeArrayPtrOfIndex 5124
VarUtils.SafeArrayPtrOfIndex function 5124
VarUtils.SafeArrayPutElement 5125
VarUtils.SafeArrayPutElement function 5125
VarUtils.SafeArrayRedim 5125
VarUtils.SafeArrayRedim function 5125
VarUtils.SafeArrayUnaccessData 5126
VarUtils.SafeArrayUnaccessData function 5126
VarUtils.SafeArrayUnlock 5127
VarUtils.SafeArrayUnlock function 5127
VarUtils.TVarTypeToElementInfo 5132
VarUtils.TVarTypeToElementInfo record 5132
VarUtils.VAR_ARRAYISLOCKED 5157
VarUtils.VAR_ARRAYISLOCKED constant 5157
VarUtils.VAR_BADINDEX 5158
VarUtils.VAR_BADINDEX constant 5158
VarUtils.VAR_BADVARTYPE 5158
VarUtils.VAR_BADVARTYPE constant 5158
VarUtils.VAR_CMP_EQ 5158
VarUtils.VAR_CMP_EQ constant 5158

VarUtils.VAR_CMP_GT 5159
VarUtils.VAR_CMP_GT constant 5159
VarUtils.VAR_CMP_LT 5159
VarUtils.VAR_CMP_LT constant 5159
VarUtils.VAR_CMP_NULL 5160
VarUtils.VAR_CMP_NULL constant 5160
VarUtils.VAR_EXCEPTION 5160
VarUtils.VAR_EXCEPTION constant 5160
VarUtils.VAR_INVALIDARG 5161
VarUtils.VAR_INVALIDARG constant 5161
VarUtils.VAR_LOCALE_USER_DEFAULT 5161
VarUtils.VAR_LOCALE_USER_DEFAULT constant 5161
VarUtils.VAR_NOTIMPL 5161
VarUtils.VAR_NOTIMPL constant 5161
VarUtils.VAR_OK 5162
VarUtils.VAR_OK constant 5162
VarUtils.VAR_OUTOFMEMORY 5162
VarUtils.VAR_OUTOFMEMORY constant 5162
VarUtils.VAR_OVERFLOW 5163
VarUtils.VAR_OVERFLOW constant 5163
VarUtils.VAR_PARAMNOTFOUND 5163
VarUtils.VAR_PARAMNOTFOUND constant 5163
VarUtils.VAR_TYPEMISMATCH 5163
VarUtils.VAR_TYPEMISMATCH constant 5163
VarUtils.VAR_UNEXPECTED 5164
VarUtils.VAR_UNEXPECTED constant 5164
VarUtils.VarAdd 5133
VarUtils.VarAdd variable 5133
VarUtils.VarAnd 5134
VarUtils.VarAnd variable 5134
VarUtils.VarBoolFromStr 5134
VarUtils.VarBoolFromStr variable 5134
VarUtils.VarBstrFromBool 5135
VarUtils.VarBstrFromBool variable 5135
VarUtils.VarBstrFromCy 5136
VarUtils.VarBstrFromCy variable 5136
VarUtils.VarBstrFromDate 5137
VarUtils.VarBstrFromDate variable 5137
VarUtils.VarCmp 5138
VarUtils.VarCmp variable 5138
VarUtils.VarCyFromStr 5139
VarUtils.VarCyFromStr variable 5139
VarUtils.VarDateFromStr 5139
VarUtils.VarDateFromStr variable 5139
VarUtils.VarDiv 5140
VarUtils.VarDiv variable 5140
VarUtils.VarExceptionToResult 5127
VarUtils.VarExceptionToResult function 5127
VarUtils.VarI4FromStr 5141
VarUtils.VarI4FromStr variable 5141
VarUtils.VariantChangeType 5128
VarUtils.VariantChangeType function 5128
VarUtils.VariantChangeTypeEx 5142
VarUtils.VariantChangeTypeEx variable 5142
VarUtils.VariantClear 5128
VarUtils.VariantClear function 5128
VarUtils.VariantCopy 5129
VarUtils.VariantCopy function 5129
VarUtils.VariantCopyInd 5130
VarUtils.VariantCopyInd function 5130
VarUtils.VariantInit 5130
VarUtils.VariantInit function 5130
VarUtils.VarIDiv 5142
VarUtils.VarIDiv variable 5142
VarUtils.VarMod 5143
VarUtils.VarMod variable 5143
VarUtils.VarMul 5143
VarUtils.VarMul variable 5143
VarUtils.VarNeg 5144
VarUtils.VarNeg variable 5144
VarUtils.VarNot 5144
VarUtils.VarNot variable 5144
VarUtils.VarOr 5145
VarUtils.VarOr variable 5145
VarUtils.VarR4FromStr 5145
VarUtils.VarR4FromStr variable 5145
VarUtils.VarR8FromStr 5146
VarUtils.VarR8FromStr variable 5146

VarUtils.VarSub 5147
VarUtils.VarSub variable 5147
VarUtils.VarXor 5147
VarUtils.VarXor variable 5147
VCLCom 5165
VCLCom namespace 5165
Classes 5166
VCLCom.TComponentFactory 5166
VCLCom.TComponentFactory.Create 5167
VCLCom.TComponentFactory.CreateComObject 5167
VCLCom.TComponentFactory.CreateInstance 5168
VCLCom.TComponentFactory.UpdateRegistry 5168
VendorLib
TSQLConnection class 1625
Vendors
TDOMVendorList class 7519
Version
IDOMXMLProlog interface 7607
ISearchAssistantOC interface 656
ISearchAssistantOC2Disp interface 666
ISearchAssistantOC3Disp interface 677
ISearchAssistantOCDisp interface 684
IXMLDocument interface 7668
TXMLDocument class 7391
vfByRef enumeration member 283
vfVariant enumeration member 283
ViewAccess
TAdapter class 5196
TAdapterBooleanField class 5230
TAdapterField class 5274
TAdapterFileField class 5289
TAdapterImageField class 5317
TAdapterMemoField class 5330
TAdapterMultiValueField class 5340
TAdapterNamedField class 5370
TCustomAdapter class 5574
TPagedAdapter class 5851
ViewMode
TAdapterDisplayField class 6381
TAdapterEditColumn class 6396
TCustomAdapterDisplayField class 6475
Visible
IAdapterActionAttributes interface 984
IAdapterFieldAttributes interface 1001
IWebBrowser2Disp interface 775
IWebBrowserApp interface 785
IWebBrowserAppDisp interface 794
TBrowseForFolder class 1771
TButton class 2070
TCheckBox class 2149
TColorSelect class 1780
TComboBox class 2226
TEdit class 2499
TFileExit class 1815
TFileOpen class 1825
TFilePageSetup class 1842
TFilePrintSetup class 1851
TFileSaveAs class 1861
TFontEdit class 1870
TGroupBox class 2564
TInternetExplorer class 457
TLabel class 2635
TListBox class 2697
TMemo class 2772
TPrintDlg class 1890
TRadioButton class 2845
TScreenTipsPopup class 349
TScrollBar class 2918
TSearchFind class 1906
TSearchFindNext class 1919
TSearchReplace class 1929
TStaticText class 2976
TTabbedNotebook class 4433
TTabPage class 4450
TTabSet class 4507
TValueListEditor class 4836
TWebBrowser class 544
TWebBrowser_V1 class 595

VisibleActions
IWebActionsList interface 1078
TCustomAdapter class 5575
VisibleChanging
TThemedPopupMenu class 4575
TXPStylePopupMenu class 7781
VisibleColCount
TValueListEditor class 4837
VisibleCommands
TCustomAdapterCommandColumn class 6453
TCustomAdapterCommandGroup class 6460
VisibleFields
IWebDataFields interface 1083
TAdapterControlGroup class 6354
TCustomAdapter class 5575
VisibleRowCount
TValueListEditor class 4838
VisibleTabs
TTabSet class 4508
vmDefault enumeration member 6525
vmDisplay enumeration member 6525
vmInput enumeration member 6525
vmToggleOnAccess enumeration member 6525
vprintf
AnsiStringBase class 3202
AnsiStringT template 3230
UnicodeString class 3413
WideString class 3487
vrEqual enumeration member 5096
vrGreaterThan enumeration member 5096
vrLessThan enumeration member 5096
vrNotEqual enumeration member 5096
VStringEvent
TFTCompT class 1313
vuBlankPassword enumeration member 6801
vuBlankUserName enumeration member 6801
vuUnknownPassword enumeration member 6801
vuUnknownUserName enumeration member 6801

W

w_str
UnicodeString class 3413
Wait
TMonitor record 3353
WaitFor
TConditionVariableCS class 3116
TConditionVariableMutex class 3124
THandleObject class 3141
WaitForConnection
TCustomTcpServer class 1411
WaitForData
TBaseSocket class 1383
TWinSocketStream class 142
WaitForMultiple
THandleObject class 3141
WaitHint
TService class 3084
Wake
TConditionVariableHelper class helper 3120
WakeAll
TConditionVariableHelper class helper 3120
WantReturns
TCustomMemo class 2454
TMemo class 2774
WantTabs
TCustomMemo class 2454
TMemo class 2774
WarningsContent
TSitePageElements class 1169
WebActions
TAdapterActions class 5214
WebAdapt 5169
WebAdapt namespace 5169
Classes 5175
Constants 5931
Functions 5897
Interfaces 5874

Structs, Records, Enums 5901
Types 5903
Variables 5930
WebAdapt.AddDefaultActions 5897
WebAdapt.AddDefaultActions function 5897
WebAdapt.AddDefaultFields 5897
WebAdapt.AddDefaultFields function 5897
WebAdapt.EAdapterException 5177
WebAdapt.EAdapterException.CreateID 5178
WebAdapt.EAdapterException.ID 5178
WebAdapt.EAdapterFieldException 5178
WebAdapt.EAdapterFieldException.Create 5179
WebAdapt.EAdapterFieldException.FieldName 5180
WebAdapt.FindAdapterAction 5898
WebAdapt.FindAdapterAction function 5898
WebAdapt.FindAdapterField 5898
WebAdapt.FindAdapterField function 5898
WebAdapt.FindAdapterInParent 5898
WebAdapt.FindAdapterInParent function 5898
WebAdapt.IActionFieldValue 5904
WebAdapt.IActionFieldValue type 5904
WebAdapt.IActionRequest 5905
WebAdapt.IActionRequest type 5905
WebAdapt.IAdapterEditor 5874
WebAdapt.IAdapterEditor.CanAddActionClass 5875
WebAdapt.IAdapterEditor.CanAddFieldClass 5875
WebAdapt.IAdapterNotifyAdapterChange 5875
WebAdapt.IAdapterNotifyAdapterChange.NotifyAdapterChange 5876
WebAdapt.ICheckOrigValue 5876
WebAdapt.ICheckOrigValue.CheckOrigValue 5877
WebAdapt.ICheckValueChange 5877
WebAdapt.ICheckValueChange.CheckValueChange 5878
WebAdapt.IClearAdapterValues 5878
WebAdapt.IClearAdapterValues.ClearAdapterValues 5879
WebAdapt.ICreateActionRequestContext 5879
WebAdapt.ICreateActionRequestContext.CreateActionRequestContext 5880
WebAdapt.IEchoActionFieldValue 5880
WebAdapt.IEchoActionFieldValue.EchoActionFieldValue 5882
WebAdapt.IEchoActionFieldValue.GetEchoActionFieldValue 5881
WebAdapt.IEchoActionFieldValue.SetEchoActionFieldValue 5881
WebAdapt.IEchoAdapterFieldValues 5882
WebAdapt.IEchoAdapterFieldValues.EchoActionFieldValues 5883
WebAdapt.IEchoAdapterFieldValues.GetEchoActionFieldValues 5883
WebAdapt.IEchoAdapterFieldValues.SetEchoActionFieldValues 5883
WebAdapt.IExecuteValue 5883
WebAdapt.IExecuteValue.ExecuteValue 5884
WebAdapt.IFullyQualifyInputNames 5885
WebAdapt.IFullyQualifyInputNames.FullyQualifyInputNames 5885
WebAdapt.IGetAdapterErrorsList 5886
WebAdapt.IGetAdapterErrorsList.GetAdapterErrorsList 5886
WebAdapt.IGetAdapterModeNames 5887
WebAdapt.IGetAdapterModeNames.GetAdapterModeNames 5887
WebAdapt.IGetRecordKeys 5888
WebAdapt.IGetRecordKeys.GetRecordKeys 5888
WebAdapt.IIdentifyAdapter 5889
WebAdapt.ISetSubComponents 5889
WebAdapt.ISetSubComponents.SetActions 5890
WebAdapt.ISetSubComponents.SetData 5890
WebAdapt.ISetWebFieldValue 5890
WebAdapt.ISetWebFieldValue.SetValue 5891
WebAdapt.IUpdateValue 5891
WebAdapt.IUpdateValue.UpdateValue 5892
WebAdapt.IValidateValue 5892
WebAdapt.IValidateValue.ValidateValue 5893
WebAdapt.IWebActionsSubcomponent 5893
WebAdapt.IWebFieldsSubcomponent 5893
WebAdapt.IWebGetFieldPrevValue 5894
WebAdapt.IWebGetFieldPrevValue.GetNeedPrevValue 5895
WebAdapt.IWebGetFieldPrevValue.GetPrevValue 5895
WebAdapt.IWebGetFieldPrevValue.NeedPrevValue 5895
WebAdapt.IWebGetFieldPrevValue.PrevValue 5895

WebAdapt.MakeAdapterRequestIdentifier 5899
WebAdapt.MakeAdapterRequestIdentifier function 5899
WebAdapt.MakeValidAdapterFieldName 5899
WebAdapt.MakeValidAdapterFieldName function 5899
WebAdapt.NotifyPossibleHTMLChange 5930
WebAdapt.NotifyPossibleHTMLChange variable 5930
WebAdapt.RaiseFileUploadNotSupported 5899
WebAdapt.RaiseFileUploadNotSupported function 5899
WebAdapt.RaiseMultipleFilesException 5900
WebAdapt.RaiseMultipleFilesException function 5900
WebAdapt.RaiseMultipleValuesException 5900
WebAdapt.RaiseMultipleValuesException function 5900
WebAdapt.sActionRequestLong 5931
WebAdapt.sActionRequestLong constant 5931
WebAdapt.sCurrentPage 5931
WebAdapt.sCurrentPage constant 5931
WebAdapt.sGotoPageAction 5932
WebAdapt.sGotoPageAction constant 5932
WebAdapt.sImageRequest 5932
WebAdapt.sImageRequest constant 5932
WebAdapt.sLoginPassword 5932
WebAdapt.sLoginPassword constant 5932
WebAdapt.sLoginUserName 5933
WebAdapt.sLoginUserName constant 5933
WebAdapt.sNextPageAction 5933
WebAdapt.sNextPageAction constant 5933
WebAdapt.sPrevPageAction 5933
WebAdapt.sPrevPageAction constant 5933
WebAdapt.TAbstractFieldValuesAdapter 5180
WebAdapt.TAbstractFieldValuesAdapter.GetField 5186
WebAdapt.TAbstractFieldValuesAdapter.GetValueFieldClass 5186
WebAdapt.TAbstractFieldValuesAdapter.GetValueIndex 5187
WebAdapt.TAbstractFieldValuesAdapter.ImplCanAddFieldClass 5187
WebAdapt.TAbstractFieldValuesAdapter.ImplEndIterator 5188
WebAdapt.TAbstractFieldValuesAdapter.ImplGetField 5188
WebAdapt.TAbstractFieldValuesAdapter.ImplGetFieldsList 5188
WebAdapt.TAbstractFieldValuesAdapter.ImplGetValueIndex 5189
WebAdapt.TAbstractFieldValuesAdapter.ImplNextIteration 5189
WebAdapt.TAbstractFieldValuesAdapter.ImplStartIterator 5190
WebAdapt.TAbstractWebRequestFile 5905
WebAdapt.TAbstractWebRequestFile type 5905
WebAdapt.TActionExecuteEvent 5905
WebAdapt.TActionExecuteEvent type 5905
WebAdapt.TActionExecuteHandledEvent 5906
WebAdapt.TActionExecuteHandledEvent type 5906
WebAdapt.TActionGetEnabledEvent 5906
WebAdapt.TActionGetEnabledEvent type 5906
WebAdapt.TActionGetParamsEvent 5907
WebAdapt.TActionGetParamsEvent type 5907
WebAdapt.TActionRequestFieldsEvent 5907
WebAdapt.TActionRequestFieldsEvent type 5907
WebAdapt.TActionRequestFieldsHandledEvent 5908
WebAdapt.TActionRequestFieldsHandledEvent type 5908
WebAdapt.TAdapter 5190
WebAdapt.TAdapter.Actions 5195
WebAdapt.TAdapter.Data 5196
WebAdapt.TAdapter.ModifyAccess 5196
WebAdapt.TAdapter.OnAfterExecuteAction 5197
WebAdapt.TAdapter.OnAfterGetActionResponse 5197
WebAdapt.TAdapter.OnAfterUpdateFields 5198
WebAdapt.TAdapter.OnAfterValidateFields 5198
WebAdapt.TAdapter.OnBeforeExecuteAction 5198
WebAdapt.TAdapter.OnBeforeGetActionResponse 5199
WebAdapt.TAdapter.OnBeforeUpdateFields 5199
WebAdapt.TAdapter.OnBeforeValidateFields 5200
WebAdapt.TAdapter.OnGetActionParams 5200
WebAdapt.TAdapter.OnIterateRecords 5200
WebAdapt.TAdapter.ViewAccess 5196
WebAdapt.TAdapterAction 5201
WebAdapt.TAdapterAction.ActionName 5205
WebAdapt.TAdapterAction.DisplayLabel 5205
WebAdapt.TAdapterAction.ExecuteAccess 5206
WebAdapt.TAdapterAction.OnAfterExecute 5207
WebAdapt.TAdapterAction.OnAfterGetResponse 5207
WebAdapt.TAdapterAction.OnBeforeExecute 5208
WebAdapt.TAdapterAction.OnBeforeGetResponse 5208

WebAdapt.TAdapterAction.OnExecute 5208
WebAdapt.TAdapterAction.OnGetEnabled 5209
WebAdapt.TAdapterAction.OnGetParams 5209
WebAdapt.TAdapterAction.RedirectOptions 5206
WebAdapt.TAdapterActionExecuteEvent 5909
WebAdapt.TAdapterActionExecuteEvent type 5909
WebAdapt.TAdapterActionExecuteHandledEvent 5909
WebAdapt.TAdapterActionExecuteHandledEvent type 5909
WebAdapt.TAdapterActionGetParamsEvent 5910
WebAdapt.TAdapterActionGetParamsEvent type 5910
WebAdapt.TAdapterActions 5209
WebAdapt.TAdapterActions.AdapterCanAddClass 5212
WebAdapt.TAdapterActions.ImplCanAddClass 5212
WebAdapt.TAdapterActions.ImplGetWebComponentNodeName 5213
WebAdapt.TAdapterActions.SetWebParent 5213
WebAdapt.TAdapterActions.WebActions 5214
WebAdapt.TAdapterActionsClass 5911
WebAdapt.TAdapterActionsClass type 5911
WebAdapt.TAdapterApplicationField 5214
WebAdapt.TAdapterApplicationField.Adapter 5218
WebAdapt.TAdapterApplicationTitleField 5218
WebAdapt.TAdapterApplicationTitleField.GetDefaultFieldName 5222
WebAdapt.TAdapterApplicationTitleField.ImplGetValue 5222
WebAdapt.TAdapterBooleanField 5223
WebAdapt.TAdapterBooleanField.DisplayLabel 5228
WebAdapt.TAdapterBooleanField.DisplayWidth 5228
WebAdapt.TAdapterBooleanField.FieldName 5228
WebAdapt.TAdapterBooleanField.ModifyAccess 5229
WebAdapt.TAdapterBooleanField.OnGetValue 5230
WebAdapt.TAdapterBooleanField.OnUpdateValue 5230
WebAdapt.TAdapterBooleanField.OnValidateValue 5231
WebAdapt.TAdapterBooleanField.ReadOnly 5229
WebAdapt.TAdapterBooleanField.ValuesList 5229
WebAdapt.TAdapterBooleanField.ViewAccess 5230
WebAdapt.TAdapterDefaultActions 5231
WebAdapt.TAdapterDefaultActions.ClearDefaultComponents 5234
WebAdapt.TAdapterDefaultActions.DefaultWebComponents 5237
WebAdapt.TAdapterDefaultActions.Destroy 5233
WebAdapt.TAdapterDefaultActions.GetAddAllFieldsItem 5235
WebAdapt.TAdapterDefaultActions.GetAddFieldsDlgCaption 5235
WebAdapt.TAdapterDefaultActions.GetAddFieldsItem 5235
WebAdapt.TAdapterDefaultActions.GetDefaultWebComponents 5235
WebAdapt.TAdapterDefaultActions.GetNewFieldPrefix 5236
WebAdapt.TAdapterDefaultActions.GetVariablesInUse 5236
WebAdapt.TAdapterDefaultActions.SetDefaultWebComponents 5236
WebAdapt.TAdapterDefaultFields 5237
WebAdapt.TAdapterDefaultFields.ClearDefaultComponents 5240
WebAdapt.TAdapterDefaultFields.DefaultWebComponents 5243
WebAdapt.TAdapterDefaultFields.Destroy 5240
WebAdapt.TAdapterDefaultFields.GetAddAllFieldsItem 5241
WebAdapt.TAdapterDefaultFields.GetAddFieldsDlgCaption 5241
WebAdapt.TAdapterDefaultFields.GetAddFieldsItem 5241
WebAdapt.TAdapterDefaultFields.GetDefaultWebComponents 5241
WebAdapt.TAdapterDefaultFields.GetNewFieldPrefix 5242
WebAdapt.TAdapterDefaultFields.GetVariablesInUse 5242
WebAdapt.TAdapterDefaultFields.SetDefaultWebComponents 5242
WebAdapt.TAdapterEndUserDisplayNameField 5244
WebAdapt.TAdapterEndUserDisplayNameField.GetDefaultFieldName 5247
WebAdapt.TAdapterEndUserDisplayNameField.ImplGetValue 5247
WebAdapt.TAdapterEndUserField 5248
WebAdapt.TAdapterEndUserField.Adapter 5251
WebAdapt.TAdapterEndUserLoggedInField 5252
WebAdapt.TAdapterEndUserLoggedInField.GetDefaultFieldName 5255
WebAdapt.TAdapterEndUserLoggedInField.ImplGetValue 5255
WebAdapt.TAdapterError 5256
WebAdapt.TAdapterError.Create 5258
WebAdapt.TAdapterError.GetErrorText 5258
WebAdapt.TAdapterError.GetField 5258

WebAdapt.TAdapterError.GetID 5259
WebAdapt.TAdapterErrorsList 5259
WebAdapt.TAdapterErrorsList.AddError 5263
WebAdapt.TAdapterErrorsList.Create 5262
WebAdapt.TAdapterErrorsList.Destroy 5262
WebAdapt.TAdapterErrorsList.FieldNames 5266
WebAdapt.TAdapterErrorsList.IDs 5266
WebAdapt.TAdapterErrorsList.ImplClearErrors 5264
WebAdapt.TAdapterErrorsList.ImplDefineLabel 5264
WebAdapt.TAdapterErrorsList.ImplGetError 5264
WebAdapt.TAdapterErrorsList.ImplGetErrorCount 5265
WebAdapt.TAdapterErrorsList.ImplGetObjectErrors 5265
WebAdapt.TAdapterErrorsList.ImplHasObjectErrors 5265
WebAdapt.TAdapterErrorsList.Messages 5266
WebAdapt.TAdapterField 5267
WebAdapt.TAdapterField.DisplayLabel 5272
WebAdapt.TAdapterField.DisplayWidth 5272
WebAdapt.TAdapterField.FieldName 5272
WebAdapt.TAdapterField.ModifyAccess 5273
WebAdapt.TAdapterField.OnGetDisplayText 5275
WebAdapt.TAdapterField.OnGetValue 5275
WebAdapt.TAdapterField.OnUpdateValue 5275
WebAdapt.TAdapterField.OnValidateValue 5276
WebAdapt.TAdapterField.ReadOnly 5273
WebAdapt.TAdapterField.Required 5273
WebAdapt.TAdapterField.ValuesList 5274
WebAdapt.TAdapterField.ViewAccess 5274
WebAdapt.TAdapterFieldBooleanValueEvent 5911
WebAdapt.TAdapterFieldBooleanValueEvent type 5911
WebAdapt.TAdapterFieldFileEvent 5912
WebAdapt.TAdapterFieldFileEvent type 5912
WebAdapt.TAdapterFieldFileListEvent 5912
WebAdapt.TAdapterFieldFileListEvent type 5912
WebAdapt.TAdapterFieldGetBooleanEvent 5913
WebAdapt.TAdapterFieldGetBooleanEvent type 5913
WebAdapt.TAdapterFieldGetMultiValueEvent 5913
WebAdapt.TAdapterFieldGetMultiValueEvent type 5913
WebAdapt.TAdapterFieldGetMultiValueStringEvent 5914
WebAdapt.TAdapterFieldGetMultiValueStringEvent type 5914
WebAdapt.TAdapterFieldGetStringEvent 5914
WebAdapt.TAdapterFieldGetStringEvent type 5914
WebAdapt.TAdapterFieldGetValueCountEvent 5915
WebAdapt.TAdapterFieldGetValueCountEvent type 5915
WebAdapt.TAdapterFieldGetValueEvent 5915
WebAdapt.TAdapterFieldGetValueEvent type 5915
WebAdapt.TAdapterFieldHasValueEvent 5916
WebAdapt.TAdapterFieldHasValueEvent type 5916
WebAdapt.TAdapterFields 5276
WebAdapt.TAdapterFields.AdapterCanAddClass 5278
WebAdapt.TAdapterFields.ImplCanAddClass 5279
WebAdapt.TAdapterFields.ImplGetWebComponentNodeName 5280
WebAdapt.TAdapterFields.SetWebParent 5280
WebAdapt.TAdapterFields.WebFields 5281
WebAdapt.TAdapterFieldsClass 5916
WebAdapt.TAdapterFieldsClass type 5916
WebAdapt.TAdapterFieldStringEvent 5917
WebAdapt.TAdapterFieldStringEvent type 5917
WebAdapt.TAdapterFieldStringsEvent 5917
WebAdapt.TAdapterFieldStringsEvent type 5917
WebAdapt.TAdapterFieldValidateValueEvent 5918
WebAdapt.TAdapterFieldValidateValueEvent type 5918
WebAdapt.TAdapterFieldValueEvent 5918
WebAdapt.TAdapterFieldValueEvent type 5918
WebAdapt.TAdapterFileField 5281
WebAdapt.TAdapterFileField.DisplayLabel 5287
WebAdapt.TAdapterFileField.DisplayWidth 5287
WebAdapt.TAdapterFileField.FieldName 5288
WebAdapt.TAdapterFileField.ModifyAccess 5288
WebAdapt.TAdapterFileField.OnGetFileCount 5289
WebAdapt.TAdapterFileField.OnGetFileNames 5290
WebAdapt.TAdapterFileField.OnUploadFiles 5290
WebAdapt.TAdapterFileField.OnValidateFiles 5290
WebAdapt.TAdapterFileField.ReadOnly 5288
WebAdapt.TAdapterFileField.Required 5289
WebAdapt.TAdapterFileField.ViewAccess 5289
WebAdapt.TAdapterGotoPageAction 5291
WebAdapt.TAdapterGotoPageAction.ExecuteAccess 5296
WebAdapt.TAdapterGotoPageAction.OnAfterExecute 5297

WebAdapt.TAdapterGotoPageAction.OnAfterGetResponse 5297
WebAdapt.TAdapterGotoPageAction.OnBeforeExecute 5297
WebAdapt.TAdapterGotoPageAction.OnBeforeGetResponse 5298
WebAdapt.TAdapterGotoPageAction.OnGetPageName 5298
WebAdapt.TAdapterGotoPageAction.OnGetParams 5298
WebAdapt.TAdapterGotoPageAction.RedirectOptions 5296
WebAdapt.TAdapterHiddenField 5299
WebAdapt.TAdapterHiddenField.Create 5301
WebAdapt.TAdapterHiddenField.GetName 5301
WebAdapt.TAdapterHiddenField.GetValue 5301
WebAdapt.TAdapterHiddenFieldOption 5901
WebAdapt.TAdapterHiddenFieldOption enumeration 5901
WebAdapt.TAdapterHiddenFieldOptions 5919
WebAdapt.TAdapterHiddenFieldOptions type 5919
WebAdapt.TAdapterHiddenFieldsList 5301
WebAdapt.TAdapterHiddenFieldsList.Add 5305
WebAdapt.TAdapterHiddenFieldsList.Clear 5305
WebAdapt.TAdapterHiddenFieldsList.Create 5304
WebAdapt.TAdapterHiddenFieldsList.Destroy 5304
WebAdapt.TAdapterHiddenFieldsList.EndIterator 5305
WebAdapt.TAdapterHiddenFieldsList.FieldCount 5308
WebAdapt.TAdapterHiddenFieldsList.Fields 5308
WebAdapt.TAdapterHiddenFieldsList.GetCurrentIntf 5306
WebAdapt.TAdapterHiddenFieldsList.GetField 5306
WebAdapt.TAdapterHiddenFieldsList.GetFieldCount 5306
WebAdapt.TAdapterHiddenFieldsList.NextIteration 5307
WebAdapt.TAdapterHiddenFieldsList.StartIterator 5307
WebAdapt.TAdapterImageField 5308
WebAdapt.TAdapterImageField.DisplayLabel 5314
WebAdapt.TAdapterImageField.DisplayWidth 5314
WebAdapt.TAdapterImageField.FieldName 5315
WebAdapt.TAdapterImageField.HREF 5315
WebAdapt.TAdapterImageField.ModifyAccess 5315
WebAdapt.TAdapterImageField.OnGetHREF 5317
WebAdapt.TAdapterImageField.OnGetImage 5318
WebAdapt.TAdapterImageField.OnGetImageName 5318
WebAdapt.TAdapterImageField.OnGetParams 5318
WebAdapt.TAdapterImageField.OnUploadFile 5319
WebAdapt.TAdapterImageField.OnValidateFile 5319
WebAdapt.TAdapterImageField.ReadOnly 5316
WebAdapt.TAdapterImageField.Required 5316
WebAdapt.TAdapterImageField.ValuesList 5316
WebAdapt.TAdapterImageField.ViewAccess 5317
WebAdapt.TAdapterLoginField 5320
WebAdapt.TAdapterLoginField.Adapter 5323
WebAdapt.TAdapterMemoField 5323
WebAdapt.TAdapterMemoField.DisplayLabel 5328
WebAdapt.TAdapterMemoField.DisplayWidth 5329
WebAdapt.TAdapterMemoField.FieldName 5329
WebAdapt.TAdapterMemoField.ModifyAccess 5329
WebAdapt.TAdapterMemoField.OnGetValue 5331
WebAdapt.TAdapterMemoField.OnUpdateValue 5331
WebAdapt.TAdapterMemoField.OnUploadFile 5332
WebAdapt.TAdapterMemoField.ReadOnly 5330
WebAdapt.TAdapterMemoField.Required 5330
WebAdapt.TAdapterMemoField.ViewAccess 5330
WebAdapt.TAdapterMultiValueField 5332
WebAdapt.TAdapterMultiValueField.DisplayLabel 5338
WebAdapt.TAdapterMultiValueField.DisplayWidth 5338
WebAdapt.TAdapterMultiValueField.FieldName 5339
WebAdapt.TAdapterMultiValueField.ModifyAccess 5339
WebAdapt.TAdapterMultiValueField.OnGetValueCount 5341
WebAdapt.TAdapterMultiValueField.OnGetValues 5341
WebAdapt.TAdapterMultiValueField.OnHasValue 5342
WebAdapt.TAdapterMultiValueField.OnUpdateValues 5342
WebAdapt.TAdapterMultiValueField.OnValidateValues 5342
WebAdapt.TAdapterMultiValueField.ReadOnly 5339
WebAdapt.TAdapterMultiValueField.Required 5340
WebAdapt.TAdapterMultiValueField.ValuesList 5340
WebAdapt.TAdapterMultiValueField.ViewAccess 5340
WebAdapt.TAdapterNamedDisplayField 5343
WebAdapt.TAdapterNamedDisplayField.Create 5346
WebAdapt.TAdapterNamedDisplayField.DisplayLabel 5350
WebAdapt.TAdapterNamedDisplayField.DisplayWidth 5350
WebAdapt.TAdapterNamedDisplayField.GetDisplayStyle 5346
WebAdapt.TAdapterNamedDisplayField.GetDisplayStyleType 5347
WebAdapt.TAdapterNamedDisplayField.GetInputStyle 5347

WebAdapt.TAdapterNamedDisplayField.GetInputStyleType 5347
WebAdapt.TAdapterNamedDisplayField.GetViewMode 5348
WebAdapt.TAdapterNamedDisplayField.GetViewModeType 5348
WebAdapt.TAdapterNamedDisplayField.ImplGetDisplayLabel 5348
WebAdapt.TAdapterNamedDisplayField.ImplGetDisplayWidth 5349
WebAdapt.TAdapterNamedDisplayField.ImplGetRequired 5349
WebAdapt.TAdapterNamedDisplayField.ReadOnly 5351
WebAdapt.TAdapterNamedDisplayField.Required 5351
WebAdapt.TAdapterNamedDisplayField.SetDisplayLabel 5349
WebAdapt.TAdapterNamedField 5351
WebAdapt.TAdapterNamedField.ActionOrigValue 5367
WebAdapt.TAdapterNamedField.ActionValue 5368
WebAdapt.TAdapterNamedField.Adapter 5368
WebAdapt.TAdapterNamedField.AsFormatted 5355
WebAdapt.TAdapterNamedField.CheckModifyAccess 5355
WebAdapt.TAdapterNamedField.CheckViewAccess 5356
WebAdapt.TAdapterNamedField.EchoActionFieldValue 5368
WebAdapt.TAdapterNamedField.FieldName 5369
WebAdapt.TAdapterNamedField.FullyQualifyInputName 5356
WebAdapt.TAdapterNamedField.GetAdapter 5356
WebAdapt.TAdapterNamedField.GetDefaultDisplayLabel 5357
WebAdapt.TAdapterNamedField.GetDefaultFieldName 5357
WebAdapt.TAdapterNamedField.GetDisplayLabel 5357
WebAdapt.TAdapterNamedField.GetDisplayWidth 5358
WebAdapt.TAdapterNamedField.GetEchoActionFieldValue 5358
WebAdapt.TAdapterNamedField.GetInputName 5358
WebAdapt.TAdapterNamedField.GetMaxLength 5359
WebAdapt.TAdapterNamedField.GetRequired 5359
WebAdapt.TAdapterNamedField.GetValue 5359
WebAdapt.TAdapterNamedField.GetVariableName 5360
WebAdapt.TAdapterNamedField.GetVisible 5360
WebAdapt.TAdapterNamedField.GetWebSnapObjClass 5360
WebAdapt.TAdapterNamedField.HasModifyAccess 5361
WebAdapt.TAdapterNamedField.HasViewAccess 5361
WebAdapt.TAdapterNamedField.ImplAsFormatted 5361
WebAdapt.TAdapterNamedField.ImplGetDisplayLabel 5361
WebAdapt.TAdapterNamedField.ImplGetDisplayWidth 5362
WebAdapt.TAdapterNamedField.ImplGetEchoActionFieldValue 5362
WebAdapt.TAdapterNamedField.ImplGetInputName 5362
WebAdapt.TAdapterNamedField.ImplGetMaxLength 5363
WebAdapt.TAdapterNamedField.ImplGetRequired 5363
WebAdapt.TAdapterNamedField.ImplGetValue 5363
WebAdapt.TAdapterNamedField.ImplGetVariableName 5364
WebAdapt.TAdapterNamedField.ImplGetVisible 5364
WebAdapt.TAdapterNamedField.ImplHasModifyAccess 5365
WebAdapt.TAdapterNamedField.ImplHasViewAccess 5365
WebAdapt.TAdapterNamedField.ImplSetEchoActionFieldValue 5365
WebAdapt.TAdapterNamedField.ImplSetFieldName 5366
WebAdapt.TAdapterNamedField.ModifyAccess 5369
WebAdapt.TAdapterNamedField.ParentChanged 5366
WebAdapt.TAdapterNamedField.SetEchoActionFieldValue 5366
WebAdapt.TAdapterNamedField.SetFieldName 5367
WebAdapt.TAdapterNamedField.Value 5369
WebAdapt.TAdapterNamedField.ViewAccess 5370
WebAdapt.TAdapterNextPageAction 5370
WebAdapt.TAdapterNextPageAction.DisplayLabel 5376
WebAdapt.TAdapterNextPageAction.ExecuteAccess 5376
WebAdapt.TAdapterNextPageAction.GetDefaultActionName 5374
WebAdapt.TAdapterNextPageAction.GetDefaultDisplayLabel 5375
WebAdapt.TAdapterNextPageAction.GetNewPage 5375
WebAdapt.TAdapterNextPageAction.OnAfterExecute 5377
WebAdapt.TAdapterNextPageAction.OnAfterGetResponse 5378
WebAdapt.TAdapterNextPageAction.OnBeforeExecute 5378
WebAdapt.TAdapterNextPageAction.OnBeforeGetResponse 5378
WebAdapt.TAdapterNextPageAction.OnGetParams 5379
WebAdapt.TAdapterNextPageAction.RedirectOptions 5377
WebAdapt.TAdapterNextPageField 5379
WebAdapt.TAdapterNextPageField.Destroy 5383
WebAdapt.TAdapterNextPageField.DisplayLabel 5386
WebAdapt.TAdapterNextPageField.DisplayWidth 5386
WebAdapt.TAdapterNextPageField.FieldName 5387
WebAdapt.TAdapterNextPageField.GetAdapterValuesList 5384
WebAdapt.TAdapterNextPageField.GetInputStyleType 5384

WebAdapt.TAdapterNextPageField.ImplGetEchoActionFieldValue 5384
WebAdapt.TAdapterNextPageField.ImplGetValue 5385
WebAdapt.TAdapterNextPageField.ImplSetEchoActionFieldValue 5385
WebAdapt.TAdapterNextPageField.UpdateValue 5386
WebAdapt.TAdapterObjectErrorsList 5387
WebAdapt.TAdapterObjectErrorsList.Create 5389
WebAdapt.TAdapterObjectErrorsList.ImplDefineLabel 5390
WebAdapt.TAdapterObjectErrorsList.ImplGetError 5390
WebAdapt.TAdapterObjectErrorsList.ImplGetErrorCount 5390
WebAdapt.TAdapterPasswordField 5391
WebAdapt.TAdapterPasswordField.DisplayLabel 5395
WebAdapt.TAdapterPasswordField.DisplayWidth 5395
WebAdapt.TAdapterPasswordField.FieldName 5396
WebAdapt.TAdapterPasswordField.GetInputStyleType 5394
WebAdapt.TAdapterPasswordField.ImplGetValue 5395
WebAdapt.TAdapterPrevPageAction 5396
WebAdapt.TAdapterPrevPageAction.DisplayLabel 5402
WebAdapt.TAdapterPrevPageAction.ExecuteAccess 5403
WebAdapt.TAdapterPrevPageAction.GetDefaultActionName 5401
WebAdapt.TAdapterPrevPageAction.GetDefaultDisplayLabel 5401
WebAdapt.TAdapterPrevPageAction.GetNewPage 5402
WebAdapt.TAdapterPrevPageAction.OnAfterExecute 5404
WebAdapt.TAdapterPrevPageAction.OnAfterGetResponse 5404
WebAdapt.TAdapterPrevPageAction.OnBeforeExecute 5404
WebAdapt.TAdapterPrevPageAction.OnBeforeGetResponse 5405
WebAdapt.TAdapterPrevPageAction.OnGetParams 5405
WebAdapt.TAdapterPrevPageAction.RedirectOptions 5403
WebAdapt.TAdapterSubComponent 5406
WebAdapt.TAdapterSubComponent.Adapter 5418
WebAdapt.TAdapterSubComponent.CanAddClass 5408
WebAdapt.TAdapterSubComponent.EndIterator 5409
WebAdapt.TAdapterSubComponent.FindNamedVariable 5409
WebAdapt.TAdapterSubComponent.FindVariable 5410
WebAdapt.TAdapterSubComponent.FindVariableContainer 5410
WebAdapt.TAdapterSubComponent.GetCurrentObject 5411
WebAdapt.TAdapterSubComponent.GetDefaultWebComponents 5411
WebAdapt.TAdapterSubComponent.GetParentComponent 5412
WebAdapt.TAdapterSubComponent.GetVariable 5412
WebAdapt.TAdapterSubComponent.GetVariableCount 5413
WebAdapt.TAdapterSubComponent.GetVariablesInUse 5413
WebAdapt.TAdapterSubComponent.GetWebComponentNodeName 5413
WebAdapt.TAdapterSubComponent.HasParent 5414
WebAdapt.TAdapterSubComponent.ImplCanAddClass 5414
WebAdapt.TAdapterSubComponent.ImplGetWebComponentNodeName 5415
WebAdapt.TAdapterSubComponent.NextIteration 5415
WebAdapt.TAdapterSubComponent.NotifyAdapterChange 5416
WebAdapt.TAdapterSubComponent.ReadState 5416
WebAdapt.TAdapterSubComponent.SetDefaultWebComponents 5416
WebAdapt.TAdapterSubComponent.SetParentComponent 5417
WebAdapt.TAdapterSubComponent.SetWebParent 5417
WebAdapt.TAdapterSubComponent.StartIterator 5418
WebAdapt.TAdapterSubComponent.VariablesInUse 5419
WebAdapt.TAdapterUpdateField 5419
WebAdapt.TAdapterUpdateField.CheckValueChange 5424
WebAdapt.TAdapterUpdateField.ConvertValue 5425
WebAdapt.TAdapterUpdateField.ConvertValueCount 5425
WebAdapt.TAdapterUpdateField.ConvertValues 5425
WebAdapt.TAdapterUpdateField.Create 5423
WebAdapt.TAdapterUpdateField.DoOnValidateValue 5426
WebAdapt.TAdapterUpdateField.GetAdapterValuesList 5426
WebAdapt.TAdapterUpdateField.GetInputStyleType 5426
WebAdapt.TAdapterUpdateField.GetValueCount 5427
WebAdapt.TAdapterUpdateField.GetValues 5427
WebAdapt.TAdapterUpdateField.ImplAsFormatted 5427
WebAdapt.TAdapterUpdateField.ImplCheckValueChange 5428
WebAdapt.TAdapterUpdateField.ImplGetDisplayLabel 5428
WebAdapt.TAdapterUpdateField.ImplGetDisplayWidth 5428
WebAdapt.TAdapterUpdateField.ImplGetEchoActionFieldValue

5429
WebAdapt.TAdapterUpdateField.ImplGetVariableName 5429
WebAdapt.TAdapterUpdateField.ImplSetEchoActionFieldValue 5429
WebAdapt.TAdapterUpdateField.ImplSetFieldName 5429
WebAdapt.TAdapterUpdateField.ImplUpdateValue 5430
WebAdapt.TAdapterUpdateField.ImplValidateValue 5430
WebAdapt.TAdapterUpdateField.IsMultiValueField 5430
WebAdapt.TAdapterUpdateField.Notification 5431
WebAdapt.TAdapterUpdateField.OnGetDisplayText 5433
WebAdapt.TAdapterUpdateField.OnValidateValue 5433
WebAdapt.TAdapterUpdateField.UpdateValue 5431
WebAdapt.TAdapterUpdateField.ValidateValue 5431
WebAdapt.TAdapterUpdateField.Value 5432
WebAdapt.TAdapterUpdateField.ValuesList 5432
WebAdapt.TAdapterUserNameField 5433
WebAdapt.TAdapterUserNameField.DisplayLabel 5438
WebAdapt.TAdapterUserNameField.DisplayWidth 5438
WebAdapt.TAdapterUserNameField.FieldName 5438
WebAdapt.TAdapterUserNameField.ImplGetValue 5437
WebAdapt.TApplicationAdapter 5439
WebAdapt.TApplicationAdapter.Actions 5444
WebAdapt.TApplicationAdapter.ApplicationTitle 5444
WebAdapt.TApplicationAdapter.Data 5444
WebAdapt.TApplicationAdapter.OnAfterExecuteAction 5445
WebAdapt.TApplicationAdapter.OnAfterGetActionResponse 5445
WebAdapt.TApplicationAdapter.OnBeforeExecuteAction 5446
WebAdapt.TApplicationAdapter.OnBeforeGetActionResponse 5446
WebAdapt.TApplicationAdapter.OnGetActionParams 5446
WebAdapt.TBaseAdapterAction 5447
WebAdapt.TBaseAdapterAction.ActionName 5474
WebAdapt.TBaseAdapterAction.Adapter 5474
WebAdapt.TBaseAdapterAction.CheckExecuteAccess 5452
WebAdapt.TBaseAdapterAction.CreateRequestContext 5452
WebAdapt.TBaseAdapterAction.DisplayLabel 5474
WebAdapt.TBaseAdapterAction.DoAfterExecuteActionRequest 5453
WebAdapt.TBaseAdapterAction.DoAfterGetExecuteActionResponse 5453
WebAdapt.TBaseAdapterAction.DoBeforeExecuteActionRequest 5453
WebAdapt.TBaseAdapterAction.DoBeforeGetExecuteActionResponse 5454
WebAdapt.TBaseAdapterAction.EndIterator 5454
WebAdapt.TBaseAdapterAction.Errors 5475
WebAdapt.TBaseAdapterAction.ExecuteAccess 5475
WebAdapt.TBaseAdapterAction.ExecuteActionRequest 5455
WebAdapt.TBaseAdapterAction.GetActionName 5455
WebAdapt.TBaseAdapterAction.GetAdapter 5456
WebAdapt.TBaseAdapterAction.GetAdapterItemRequestParams 5456
WebAdapt.TBaseAdapterAction.GetDefaultActionName 5456
WebAdapt.TBaseAdapterAction.GetDefaultDisplayLabel 5457
WebAdapt.TBaseAdapterAction.GetDisplayLabel 5457
WebAdapt.TBaseAdapterAction.GetDisplayStyle 5457
WebAdapt.TBaseAdapterAction.GetDisplayStyleType 5458
WebAdapt.TBaseAdapterAction.GetErrors 5458
WebAdapt.TBaseAdapterAction.GetExecuteActionResponse 5458
WebAdapt.TBaseAdapterAction.GetFailureOptions 5459
WebAdapt.TBaseAdapterAction.GetInputStyle 5459
WebAdapt.TBaseAdapterAction.GetIsAdapterActionList 5460
WebAdapt.TBaseAdapterAction.GetItemCount 5460
WebAdapt.TBaseAdapterAction.GetIteratorIndex 5460
WebAdapt.TBaseAdapterAction.GetSuccessOptions 5461
WebAdapt.TBaseAdapterAction.GetVariableName 5461
WebAdapt.TBaseAdapterAction.GetViewMode 5462
WebAdapt.TBaseAdapterAction.GetViewModeType 5462
WebAdapt.TBaseAdapterAction.GetVisible 5462
WebAdapt.TBaseAdapterAction.GetWebSnapObjClass 5463
WebAdapt.TBaseAdapterAction.HandleRequest 5463
WebAdapt.TBaseAdapterAction.HasExecuteAccess 5463
WebAdapt.TBaseAdapterAction.ImplCreateRequestContext 5464
WebAdapt.TBaseAdapterAction.ImplEndIterator 5464
WebAdapt.TBaseAdapterAction.ImplExecuteActionRequest 5464

WebAdapt.TBaseAdapterAction.ImplGetActionName 5465
WebAdapt.TBaseAdapterAction.ImplGetAdapterItemRequestParams 5465
WebAdapt.TBaseAdapterAction.ImplGetDisplayLabel 5465
WebAdapt.TBaseAdapterAction.ImplGetExecuteActionResponse 5466
WebAdapt.TBaseAdapterAction.ImplGetFailureOptions 5466
WebAdapt.TBaseAdapterAction.ImplGetIsAdapterActionList 5466
WebAdapt.TBaseAdapterAction.ImplGetIteratorIndex 5467
WebAdapt.TBaseAdapterAction.ImplGetSuccessOptions 5467
WebAdapt.TBaseAdapterAction.ImplGetVariableName 5467
WebAdapt.TBaseAdapterAction.ImplGetVisible 5468
WebAdapt.TBaseAdapterAction.ImplHandleRequest 5468
WebAdapt.TBaseAdapterAction.ImplHasExecuteAccess 5468
WebAdapt.TBaseAdapterAction.ImplInIterator 5469
WebAdapt.TBaseAdapterAction.ImplIsDefaultAction 5469
WebAdapt.TBaseAdapterAction.ImplNextIteration 5470
WebAdapt.TBaseAdapterAction.ImplStartIterator 5470
WebAdapt.TBaseAdapterAction.ImplWebEnabled 5470
WebAdapt.TBaseAdapterAction.InIterator 5471
WebAdapt.TBaseAdapterAction.IsDefaultAction 5471
WebAdapt.TBaseAdapterAction.ItemCount 5475
WebAdapt.TBaseAdapterAction.ItemIndex 5476
WebAdapt.TBaseAdapterAction.NextIteration 5472
WebAdapt.TBaseAdapterAction.OnAfterExecute 5477
WebAdapt.TBaseAdapterAction.OnAfterGetResponse 5477
WebAdapt.TBaseAdapterAction.OnBeforeExecute 5478
WebAdapt.TBaseAdapterAction.OnBeforeGetResponse 5478
WebAdapt.TBaseAdapterAction.OnGetParams 5478
WebAdapt.TBaseAdapterAction.ParentChanged 5472
WebAdapt.TBaseAdapterAction.RedirectOptions 5476
WebAdapt.TBaseAdapterAction.SetDisplayLabel 5472
WebAdapt.TBaseAdapterAction.StartIterator 5473
WebAdapt.TBaseAdapterAction.WebEnabled 5473
WebAdapt.TBaseAdapterErrorsList 5479
WebAdapt.TBaseAdapterErrorsList.AddError 5482
WebAdapt.TBaseAdapterErrorsList.ClearErrors 5482
WebAdapt.TBaseAdapterErrorsList.DefineLabel 5483
WebAdapt.TBaseAdapterErrorsList.EndIterator 5483
WebAdapt.TBaseAdapterErrorsList.ErrorCount 5487
WebAdapt.TBaseAdapterErrorsList.Errors 5487
WebAdapt.TBaseAdapterErrorsList.GetCurrentIntf 5483
WebAdapt.TBaseAdapterErrorsList.GetError 5484
WebAdapt.TBaseAdapterErrorsList.GetErrorCount 5484
WebAdapt.TBaseAdapterErrorsList.GetObjectErrors 5484
WebAdapt.TBaseAdapterErrorsList.HasObjectErrors 5484
WebAdapt.TBaseAdapterErrorsList.ImplClearErrors 5485
WebAdapt.TBaseAdapterErrorsList.ImplDefineLabel 5485
WebAdapt.TBaseAdapterErrorsList.ImplGetError 5485
WebAdapt.TBaseAdapterErrorsList.ImplGetErrorCount 5485
WebAdapt.TBaseAdapterErrorsList.ImplGetObjectErrors 5486
WebAdapt.TBaseAdapterErrorsList.ImplHasObjectErrors 5486
WebAdapt.TBaseAdapterErrorsList.NextIteration 5486
WebAdapt.TBaseAdapterErrorsList.StartIterator 5486
WebAdapt.TBaseAdapterMultiValueField 5488
WebAdapt.TBaseAdapterMultiValueField.Create 5492
WebAdapt.TBaseAdapterMultiValueField.Destroy 5492
WebAdapt.TBaseAdapterMultiValueField.EndValueIterator 5493
WebAdapt.TBaseAdapterMultiValueField.GetCurrentValue 5494
WebAdapt.TBaseAdapterMultiValueField.GetFieldValuesAdapter 5494
WebAdapt.TBaseAdapterMultiValueField.GetValueCount 5494
WebAdapt.TBaseAdapterMultiValueField.GetValues 5495
WebAdapt.TBaseAdapterMultiValueField.HasValue 5495
WebAdapt.TBaseAdapterMultiValueField.ImplGetCurrentValue 5495
WebAdapt.TBaseAdapterMultiValueField.ImplGetFieldValuesAdapter 5496
WebAdapt.TBaseAdapterMultiValueField.ImplHasValue 5496
WebAdapt.TBaseAdapterMultiValueField.NextValueIteration 5496
WebAdapt.TBaseAdapterMultiValueField.StartValueIterator 5497
WebAdapt.TBaseEndUserAdapter 5497
WebAdapt.TBaseEndUserAdapter.CanExecuteLoginForm 5504
WebAdapt.TBaseEndUserAdapter.CanExecuteLogout 5504
WebAdapt.TBaseEndUserAdapter.CanProduceLoginPage 5505
WebAdapt.TBaseEndUserAdapter.DoOnHasRights 5505

WebAdapt.TBaseEndUserAdapter.ExecuteLogin 5505
WebAdapt.TBaseEndUserAdapter.ExecuteLoginForm 5506
WebAdapt.TBaseEndUserAdapter.ExecuteLogout 5506
WebAdapt.TBaseEndUserAdapter.GetDisplayName 5506
WebAdapt.TBaseEndUserAdapter.GetDisplayNameValue 5507
WebAdapt.TBaseEndUserAdapter.GetEndUserAdapter 5507
WebAdapt.TBaseEndUserAdapter.GetLoggedIn 5507
WebAdapt.TBaseEndUserAdapter.GetLoggedInValue 5508
WebAdapt.TBaseEndUserAdapter.GetLoginFormAction 5508
WebAdapt.TBaseEndUserAdapter.GetLogoutAction 5508
WebAdapt.TBaseEndUserAdapter.GetUserID 5509
WebAdapt.TBaseEndUserAdapter.HasExecuteRights 5509
WebAdapt.TBaseEndUserAdapter.HasModifyRights 5509
WebAdapt.TBaseEndUserAdapter.HasViewRights 5510
WebAdapt.TBaseEndUserAdapter.ImplCanAddActionClass 5510
WebAdapt.TBaseEndUserAdapter.ImplCanAddFieldClass 5510
WebAdapt.TBaseEndUserAdapter.ImplGetActionsList 5511
WebAdapt.TBaseEndUserAdapter.ImplGetDisplayName 5511
WebAdapt.TBaseEndUserAdapter.ImplGetEndUserAdapter 5511
WebAdapt.TBaseEndUserAdapter.ImplGetFieldsList 5512
WebAdapt.TBaseEndUserAdapter.ImplGetLoggedIn 5512
WebAdapt.TBaseEndUserAdapter.ImplGetLoginFormAction 5512
WebAdapt.TBaseEndUserAdapter.ImplGetLogoutAction 5513
WebAdapt.TBaseEndUserAdapter.ImplHasExecuteRights 5513
WebAdapt.TBaseEndUserAdapter.ImplHasModifyRights 5513
WebAdapt.TBaseEndUserAdapter.ImplHasViewRights 5514
WebAdapt.TBaseEndUserAdapter.ImplLogIn 5514
WebAdapt.TBaseEndUserAdapter.ImplProduceLoginPage 5514
WebAdapt.TBaseEndUserAdapter.LogIn 5515
WebAdapt.TBaseEndUserAdapter.LoginPage 5516
WebAdapt.TBaseEndUserAdapter.OnHasRights 5516
WebAdapt.TBaseEndUserAdapter.OnProduceLoginPage 5516
WebAdapt.TBaseEndUserAdapter.ProduceLoginPage 5515
WebAdapt.TBaseEndUserAdapter.UserList 5515
WebAdapt.TBaseValuesListAdapter 5517
WebAdapt.TBaseValuesListAdapter.GetListImage 5522
WebAdapt.TBaseValuesListAdapter.GetListImageOfValue 5523
WebAdapt.TBaseValuesListAdapter.GetListName 5523
WebAdapt.TBaseValuesListAdapter.GetListNameOfValue 5523
WebAdapt.TBaseValuesListAdapter.GetListValue 5524
WebAdapt.TBaseValuesListAdapter.GetNameFieldClass 5524
WebAdapt.TBaseValuesListAdapter.GetValueFieldClass 5524
WebAdapt.TBaseValuesListAdapter.GetWebSnapObjClass 5524
WebAdapt.TBaseValuesListAdapter.ImplCanAddFieldClass 5525
WebAdapt.TBaseValuesListAdapter.ImplGetFieldsList 5525
WebAdapt.TBaseValuesListAdapter.ImplGetListImage 5525
WebAdapt.TBaseValuesListAdapter.ImplGetListImageOfValue 5526
WebAdapt.TBaseValuesListAdapter.ImplGetListName 5526
WebAdapt.TBaseValuesListAdapter.ImplGetListNameOfValue 5526
WebAdapt.TBaseValuesListAdapter.ImplGetListValue 5527
WebAdapt.TBaseValuesListField 5527
WebAdapt.TBaseValuesListField.Adapter 5531
WebAdapt.TBaseValuesListField.GetAdapter 5530
WebAdapt.TCustomAdapter 5531
WebAdapt.TCustomAdapter.Actions 5572
WebAdapt.TCustomAdapter.AddAdapterModeHiddenField 5538
WebAdapt.TCustomAdapter.AddOrigValueHiddenFields 5539
WebAdapt.TCustomAdapter.AddRecordKeysHiddenFields 5539
WebAdapt.TCustomAdapter.CanAddActionClass 5539
WebAdapt.TCustomAdapter.CanAddFieldClass 5540
WebAdapt.TCustomAdapter.CheckModifyAccess 5540
WebAdapt.TCustomAdapter.ClearAdapterValues 5540
WebAdapt.TCustomAdapter.ClearErrors 5541
WebAdapt.TCustomAdapter.Create 5536
WebAdapt.TCustomAdapter.CreateActionRequestContext 5541
WebAdapt.TCustomAdapter.CreateErrorsList 5542
WebAdapt.TCustomAdapter.Data 5572
WebAdapt.TCustomAdapter.Destroy 5536
WebAdapt.TCustomAdapter.EchoActionFieldValues 5572
WebAdapt.TCustomAdapter.EncodeActionParams 5542
WebAdapt.TCustomAdapter.EndIterator 5542
WebAdapt.TCustomAdapter.Errors 5573
WebAdapt.TCustomAdapter.ExecuteAccess 5573
WebAdapt.TCustomAdapter.ExtractRequestParams 5543

WebAdapt.TCustomAdapter.FindAction 5543
WebAdapt.TCustomAdapter.FindField 5543
WebAdapt.TCustomAdapter.FindNamedAction 5544
WebAdapt.TCustomAdapter.FindNamedField 5544
WebAdapt.TCustomAdapter.FindNamedVariable 5544
WebAdapt.TCustomAdapter.FindVariable 5545
WebAdapt.TCustomAdapter.FindVariableContainer 5545
WebAdapt.TCustomAdapter.GetActionsClass 5545
WebAdapt.TCustomAdapter.GetActionsList 5546
WebAdapt.TCustomAdapter.GetAdapterActions 5546
WebAdapt.TCustomAdapter.GetAdapterErrors 5546
WebAdapt.TCustomAdapter.GetAdapterErrorsList 5547
WebAdapt.TCustomAdapter.GetAdapterFields 5547
WebAdapt.TCustomAdapter.GetAdapterHiddenFields 5547
WebAdapt.TCustomAdapter.GetAdapterHiddenRecordFields 5548
WebAdapt.TCustomAdapter.GetChildren 5548
WebAdapt.TCustomAdapter.GetDataClass 5548
WebAdapt.TCustomAdapter.GetEchoActionFieldValues 5549
WebAdapt.TCustomAdapter.GetFieldsList 5549
WebAdapt.TCustomAdapter.GetHiddenFieldOptions 5549
WebAdapt.TCustomAdapter.GetIteratorIndex 5550
WebAdapt.TCustomAdapter.GetVariable 5550
WebAdapt.TCustomAdapter.GetVariableCount 5550
WebAdapt.TCustomAdapter.GetVariableName 5551
WebAdapt.TCustomAdapter.GetVisibleActions 5551
WebAdapt.TCustomAdapter.GetVisibleFields 5551
WebAdapt.TCustomAdapter.GetWebSnapObjClass 5552
WebAdapt.TCustomAdapter.HasExecuteAccess 5552
WebAdapt.TCustomAdapter.HasModifyAccess 5552
WebAdapt.TCustomAdapter.HasViewAccess 5553
WebAdapt.TCustomAdapter.ImplCanAddActionClass 5553
WebAdapt.TCustomAdapter.ImplCanAddFieldClass 5553
WebAdapt.TCustomAdapter.ImplClearAdapterValues 5554
WebAdapt.TCustomAdapter.ImplEndIterator 5554
WebAdapt.TCustomAdapter.ImplFindVariable 5554
WebAdapt.TCustomAdapter.ImplFindVariableContainer 5555
WebAdapt.TCustomAdapter.ImplGetActionsList 5555
WebAdapt.TCustomAdapter.ImplGetEchoActionFieldValues 5555
WebAdapt.TCustomAdapter.ImplGetFieldsList 5556
WebAdapt.TCustomAdapter.ImplGetIteratorIndex 5556
WebAdapt.TCustomAdapter.ImplGetVariable 5556
WebAdapt.TCustomAdapter.ImplGetVariableCount 5557
WebAdapt.TCustomAdapter.ImplGetVariableName 5557
WebAdapt.TCustomAdapter.ImplGetVisibleActions 5557
WebAdapt.TCustomAdapter.ImplGetVisibleFields 5558
WebAdapt.TCustomAdapter.ImplHasExecuteAccess 5558
WebAdapt.TCustomAdapter.ImplHasModifyAccess 5558
WebAdapt.TCustomAdapter.ImplHasViewAccess 5559
WebAdapt.TCustomAdapter.ImplInIterator 5559
WebAdapt.TCustomAdapter.ImplNextIteration 5560
WebAdapt.TCustomAdapter.ImplNotifyActivate 5560
WebAdapt.TCustomAdapter.ImplNotifyDeactivate 5560
WebAdapt.TCustomAdapter.ImplSetActions 5561
WebAdapt.TCustomAdapter.ImplSetData 5561
WebAdapt.TCustomAdapter.ImplSetEchoActionFieldValues 5561
WebAdapt.TCustomAdapter.ImplStartIterator 5562
WebAdapt.TCustomAdapter.InIterator 5562
WebAdapt.TCustomAdapter.IsActionInUse 5563
WebAdapt.TCustomAdapter.IsFieldInUse 5563
WebAdapt.TCustomAdapter.ListHiddenFields 5563
WebAdapt.TCustomAdapter.ListHiddenRecordFields 5564
WebAdapt.TCustomAdapter.ModifyAccess 5574
WebAdapt.TCustomAdapter.NextIteration 5564
WebAdapt.TCustomAdapter.Notification 5565
WebAdapt.TCustomAdapter.NotifyActivate 5565
WebAdapt.TCustomAdapter.NotifyAdapterChange 5565
WebAdapt.TCustomAdapter.NotifyDeactivate 5566
WebAdapt.TCustomAdapter.NotifyList 5574
WebAdapt.TCustomAdapter.OnAfterExecuteAction 5576
WebAdapt.TCustomAdapter.OnAfterGetActionResponse 5576
WebAdapt.TCustomAdapter.OnAfterUpdateFields 5576
WebAdapt.TCustomAdapter.OnAfterValidateFields 5577
WebAdapt.TCustomAdapter.OnBeforeExecuteAction 5577
WebAdapt.TCustomAdapter.OnBeforeGetActionResponse 5578
WebAdapt.TCustomAdapter.OnBeforeUpdateFields 5578
WebAdapt.TCustomAdapter.OnBeforeValidateFields 5578
WebAdapt.TCustomAdapter.OnGetActionParams 5579

WebAdapt.TCustomAdapter.OnIterateRecords 5579
WebAdapt.TCustomAdapter.SetActions 5566
WebAdapt.TCustomAdapter.SetData 5566
WebAdapt.TCustomAdapter.SetEchoActionFieldValues 5567
WebAdapt.TCustomAdapter.StartIterator 5567
WebAdapt.TCustomAdapter.UpdateFieldsExecuteValues 5567
WebAdapt.TCustomAdapter.UpdateFieldsGetAnyChanges 5568
WebAdapt.TCustomAdapter.UpdateFieldsUpdateValues 5568
WebAdapt.TCustomAdapter.UpdateFieldsValidateValues 5569
WebAdapt.TCustomAdapter.UpdateRecords 5569
WebAdapt.TCustomAdapter.UpdateRecordsGetAnyChanges 5570
WebAdapt.TCustomAdapter.UpdateRecordsUpdateValues 5570
WebAdapt.TCustomAdapter.UpdateRecordsValidateValues 5571
WebAdapt.TCustomAdapter.ViewAccess 5574
WebAdapt.TCustomAdapter.VisibleActions 5575
WebAdapt.TCustomAdapter.VisibleFields 5575
WebAdapt.TCustomAdapter.WebComponentsChange 5571
WebAdapt.TCustomAdapterAction 5579
WebAdapt.TCustomAdapterAction.ActionName 5586
WebAdapt.TCustomAdapterAction.ImplExecuteActionRequest 5584
WebAdapt.TCustomAdapterAction.ImplGetActionName 5584
WebAdapt.TCustomAdapterAction.ImplGetAdapterItemRequestParams 5584
WebAdapt.TCustomAdapterAction.ImplGetSuccessOptions 5585
WebAdapt.TCustomAdapterAction.ImplSetActionName 5585
WebAdapt.TCustomAdapterAction.ImplWebEnabled 5585
WebAdapt.TCustomAdapterAction.OnAfterExecute 5587
WebAdapt.TCustomAdapterAction.OnAfterGetResponse 5587
WebAdapt.TCustomAdapterAction.OnBeforeExecute 5588
WebAdapt.TCustomAdapterAction.OnBeforeGetResponse 5588
WebAdapt.TCustomAdapterAction.OnExecute 5588
WebAdapt.TCustomAdapterAction.OnGetEnabled 5589
WebAdapt.TCustomAdapterAction.SetActionName 5586
WebAdapt.TCustomAdapterBooleanField 5589
WebAdapt.TCustomAdapterBooleanField.ConvertValue 5594
WebAdapt.TCustomAdapterBooleanField.GetDisplayStyleType 5594
WebAdapt.TCustomAdapterBooleanField.GetInputStyleType 5595
WebAdapt.TCustomAdapterBooleanField.ImplGetValue 5595
WebAdapt.TCustomAdapterBooleanField.ImplIsEqual 5596
WebAdapt.TCustomAdapterBooleanField.ImplUpdateValue 5596
WebAdapt.TCustomAdapterBooleanField.IsEqual 5596
WebAdapt.TCustomAdapterBooleanField.OnGetValue 5597
WebAdapt.TCustomAdapterBooleanField.OnUpdateValue 5597
WebAdapt.TCustomAdapterField 5597
WebAdapt.TCustomAdapterField.DoOnValidateValue 5602
WebAdapt.TCustomAdapterField.ImplGetValue 5602
WebAdapt.TCustomAdapterField.ImplUpdateValue 5603
WebAdapt.TCustomAdapterField.OnGetValue 5603
WebAdapt.TCustomAdapterField.OnUpdateValue 5604
WebAdapt.TCustomAdapterFileField 5604
WebAdapt.TCustomAdapterFileField.DoOnValidateValue 5609
WebAdapt.TCustomAdapterFileField.FileCount 5612
WebAdapt.TCustomAdapterFileField.FileNames 5612
WebAdapt.TCustomAdapterFileField.GetDisplayStyleType 5609
WebAdapt.TCustomAdapterFileField.GetFileNames 5610
WebAdapt.TCustomAdapterFileField.GetInputStyleType 5610
WebAdapt.TCustomAdapterFileField.GetValueCount 5610
WebAdapt.TCustomAdapterFileField.GetValues 5611
WebAdapt.TCustomAdapterFileField.ImplCheckValueChange 5611
WebAdapt.TCustomAdapterFileField.ImplUpdateValue 5611
WebAdapt.TCustomAdapterFileField.OnGetFileCount 5613
WebAdapt.TCustomAdapterFileField.OnGetFileNames 5613
WebAdapt.TCustomAdapterFileField.OnUploadFiles 5613
WebAdapt.TCustomAdapterFileField.OnValidateFiles 5614
WebAdapt.TCustomAdapterGotoPageAction 5614
WebAdapt.TCustomAdapterGotoPageAction.GetDefaultActionName 5619
WebAdapt.TCustomAdapterGotoPageAction.GetDisplayStyleType 5619
WebAdapt.TCustomAdapterGotoPageAction.GetItemCount 5620
WebAdapt.TCustomAdapterGotoPageAction.ImplExecuteActionRequest 5620

WebAdapt.TCustomAdapterGotoPageAction.ImplGetAdapterItemRequestParams 5620
WebAdapt.TCustomAdapterGotoPageAction.ImplGetDisplayLabel 5621
WebAdapt.TCustomAdapterGotoPageAction.ImplGetIsAdapterActionList 5621
WebAdapt.TCustomAdapterGotoPageAction.ImplIsDefaultAction 5622
WebAdapt.TCustomAdapterGotoPageAction.ImplWebEnabled 5622
WebAdapt.TCustomAdapterGotoPageAction.OnGetPageName 5622
WebAdapt.TCustomAdapterImageField 5623
WebAdapt.TCustomAdapterImageField.CreateRequestContext 5628
WebAdapt.TCustomAdapterImageField.DoOnValidateValue 5628
WebAdapt.TCustomAdapterImageField.GetAdapterImage 5629
WebAdapt.TCustomAdapterImageField.GetAdapterItemRequestParams 5629
WebAdapt.TCustomAdapterImageField.GetDisplayStyleType 5630
WebAdapt.TCustomAdapterImageField.GetInputStyleType 5630
WebAdapt.TCustomAdapterImageField.HandleRequest 5630
WebAdapt.TCustomAdapterImageField.HREF 5634
WebAdapt.TCustomAdapterImageField.ImplCheckValueChange 5631
WebAdapt.TCustomAdapterImageField.ImplCreateRequestContext 5631
WebAdapt.TCustomAdapterImageField.ImplGetValue 5632
WebAdapt.TCustomAdapterImageField.ImplHandleRequest 5632
WebAdapt.TCustomAdapterImageField.ImplUpdateValue 5632
WebAdapt.TCustomAdapterImageField.ImplWebImageHREF 5633
WebAdapt.TCustomAdapterImageField.OnGetHREF 5635
WebAdapt.TCustomAdapterImageField.OnGetImage 5636
WebAdapt.TCustomAdapterImageField.OnGetImageName 5636
WebAdapt.TCustomAdapterImageField.OnGetParams 5636
WebAdapt.TCustomAdapterImageField.OnUploadFile 5637
WebAdapt.TCustomAdapterImageField.OnValidateFile 5637
WebAdapt.TCustomAdapterImageField.ReadOnly 5635
WebAdapt.TCustomAdapterImageField.RenderAdapterImage 5633
WebAdapt.TCustomAdapterImageField.WebImageHREF 5634
WebAdapt.TCustomAdapterMemoField 5638
WebAdapt.TCustomAdapterMemoField.DoOnValidateValue 5642
WebAdapt.TCustomAdapterMemoField.GetDisplayStyleType 5642
WebAdapt.TCustomAdapterMemoField.GetInputStyleType 5643
WebAdapt.TCustomAdapterMemoField.ImplCheckValueChange 5643
WebAdapt.TCustomAdapterMemoField.ImplGetValue 5644
WebAdapt.TCustomAdapterMemoField.ImplUpdateValue 5644
WebAdapt.TCustomAdapterMemoField.OnGetValue 5645
WebAdapt.TCustomAdapterMemoField.OnUpdateValue 5645
WebAdapt.TCustomAdapterMemoField.OnUploadFile 5645
WebAdapt.TCustomAdapterMultiValueField 5646
WebAdapt.TCustomAdapterMultiValueField.DoOnValidateValue 5651
WebAdapt.TCustomAdapterMultiValueField.GetDisplayStyleType 5651
WebAdapt.TCustomAdapterMultiValueField.GetInputStyleType 5651
WebAdapt.TCustomAdapterMultiValueField.GetValueCount 5652
WebAdapt.TCustomAdapterMultiValueField.GetValues 5652
WebAdapt.TCustomAdapterMultiValueField.ImplHasValue 5653
WebAdapt.TCustomAdapterMultiValueField.ImplUpdateValue 5653
WebAdapt.TCustomAdapterMultiValueField.OnGetValueCount 5654
WebAdapt.TCustomAdapterMultiValueField.OnGetValues 5655
WebAdapt.TCustomAdapterMultiValueField.OnHasValue 5655
WebAdapt.TCustomAdapterMultiValueField.OnUpdateValues 5656
WebAdapt.TCustomAdapterMultiValueField.OnValidateValues 5656
WebAdapt.TCustomAdapterMultiValueField.ValueCount 5653
WebAdapt.TCustomAdapterMultiValueField.Values 5654
WebAdapt.TCustomAdapterPageAction 5656

WebAdapt.TCustomAdapterPageAction.GetDisplayStyleType 5661
WebAdapt.TCustomAdapterPageAction.GetNewPage 5661
WebAdapt.TCustomAdapterPageAction.ImplExecuteActionRequest 5662
WebAdapt.TCustomAdapterPageAction.ImplGetAdapterItemRequestParams 5662
WebAdapt.TCustomAdapterPageAction.ImplIsDefaultAction 5663
WebAdapt.TCustomAdapterPageAction.ImplWebEnabled 5663
WebAdapt.TCustomApplicationAdapter 5663
WebAdapt.TCustomApplicationAdapter.ApplicationTitle 5671
WebAdapt.TCustomApplicationAdapter.GetApplicationAdapter 5669
WebAdapt.TCustomApplicationAdapter.GetApplicationTitle 5669
WebAdapt.TCustomApplicationAdapter.GetWebSnapObjClass 5669
WebAdapt.TCustomApplicationAdapter.ImplCanAddActionClass 5669
WebAdapt.TCustomApplicationAdapter.ImplCanAddFieldClass 5670
WebAdapt.TCustomApplicationAdapter.ImplGetActionsList 5670
WebAdapt.TCustomApplicationAdapter.ImplGetFieldsList 5670
WebAdapt.TCustomApplicationAdapterAction 5671
WebAdapt.TCustomApplicationAdapterAction.Adapter 5676
WebAdapt.TCustomEndUserAdapter 5676
WebAdapt.TCustomEndUserAdapter.ExecuteLogin 5683
WebAdapt.TCustomEndUserAdapter.ExecuteLogout 5683
WebAdapt.TCustomEndUserAdapter.GetDisplayNameValue 5683
WebAdapt.TCustomEndUserAdapter.GetLoggedInValue 5684
WebAdapt.TCustomEndUserAdapter.GetUserID 5684
WebAdapt.TCustomEndUserAdapter.OnGetDisplayName 5685
WebAdapt.TCustomEndUserAdapter.OnGetUserID 5685
WebAdapt.TCustomEndUserAdapter.OnIsLoggedIn 5686
WebAdapt.TCustomEndUserAdapter.OnLogin 5686
WebAdapt.TCustomEndUserAdapter.OnLogout 5687
WebAdapt.TCustomEndUserAdapterAction 5687
WebAdapt.TCustomEndUserAdapterAction.Adapter 5692
WebAdapt.TCustomEndUserSessionAdapter 5692
WebAdapt.TCustomEndUserSessionAdapter.ExecuteLogin 5699
WebAdapt.TCustomEndUserSessionAdapter.ExecuteLogout 5699
WebAdapt.TCustomEndUserSessionAdapter.GetDisplayNameValue 5699
WebAdapt.TCustomEndUserSessionAdapter.GetLoggedInValue 5700
WebAdapt.TCustomEndUserSessionAdapter.GetUserID 5700
WebAdapt.TCustomEndUserSessionAdapter.UserID 5701
WebAdapt.TCustomLoginFormAdapter 5701
WebAdapt.TCustomLoginFormAdapter.EndUser 5707
WebAdapt.TCustomLoginFormAdapter.ExecuteLogin 5707
WebAdapt.TCustomLoginFormAdapter.GetNextPage 5708
WebAdapt.TCustomLoginFormAdapter.ImplCanAddActionClass 5708
WebAdapt.TCustomLoginFormAdapter.ImplCanAddFieldClass 5709
WebAdapt.TCustomLoginFormAdapter.ImplGetActionsList 5709
WebAdapt.TCustomLoginFormAdapter.ImplGetFieldsList 5710
WebAdapt.TCustomLoginFormAdapter.ImplNotifyDeactivate 5710
WebAdapt.TCustomLoginFormAdapter.NextPage 5713
WebAdapt.TCustomLoginFormAdapter.OnLogin 5715
WebAdapt.TCustomLoginFormAdapter.OnValidatePassword 5716
WebAdapt.TCustomLoginFormAdapter.OnValidateUserName 5716
WebAdapt.TCustomLoginFormAdapter.Password 5714
WebAdapt.TCustomLoginFormAdapter.PasswordRequired 5714
WebAdapt.TCustomLoginFormAdapter.RaiseBlankPassword 5711
WebAdapt.TCustomLoginFormAdapter.RaiseBlankUserName 5711
WebAdapt.TCustomLoginFormAdapter.UserList 5711
WebAdapt.TCustomLoginFormAdapter.UserName 5714
WebAdapt.TCustomLoginFormAdapter.ValidatePassword 5712
WebAdapt.TCustomLoginFormAdapter.ValidateUserName 5712
WebAdapt.TCustomLoginFormAdapterAction 5717
WebAdapt.TCustomLoginFormAdapterAction.Adapter 5721
WebAdapt.TCustomPagedAdapter 5721
WebAdapt.TCustomPagedAdapterAction 5722
WebAdapt.TCustomPagedAdapterAction.Adapter 5727

WebAdapt.TCustomPagedAdapterAction.GetAdapter 5727
WebAdapt.TCustomStringsValuesList 5728
WebAdapt.TCustomStringsValuesList.Create 5733
WebAdapt.TCustomStringsValuesList.Destroy 5733
WebAdapt.TCustomStringsValuesList.GetNameFieldClass 5734
WebAdapt.TCustomStringsValuesList.GetValueFieldClass 5734
WebAdapt.TCustomStringsValuesList.ImplEndIterator 5734
WebAdapt.TCustomStringsValuesList.ImplGetListName 5735
WebAdapt.TCustomStringsValuesList.ImplGetListNameOfValue 5735
WebAdapt.TCustomStringsValuesList.ImplGetListValue 5735
WebAdapt.TCustomStringsValuesList.ImplNextIteration 5736
WebAdapt.TCustomStringsValuesList.ImplNotifyDeactivate 5736
WebAdapt.TCustomStringsValuesList.ImplStartIterator 5736
WebAdapt.TCustomStringsValuesList.OnPrepareStrings 5738
WebAdapt.TCustomStringsValuesList.PrepareStrings 5737
WebAdapt.TCustomStringsValuesList.Strings 5737
WebAdapt.TCustomStringsValuesList.StringsPrepared 5737
WebAdapt.TDefaultFieldsAdapter 5738
WebAdapt.TDefaultFieldsAdapter.Actions 5746
WebAdapt.TDefaultFieldsAdapter.AddDefaultActions 5743
WebAdapt.TDefaultFieldsAdapter.AddDefaultFields 5744
WebAdapt.TDefaultFieldsAdapter.Data 5747
WebAdapt.TDefaultFieldsAdapter.DefaultWebComponentsChange 5744
WebAdapt.TDefaultFieldsAdapter.GetActionsClass 5744
WebAdapt.TDefaultFieldsAdapter.GetDataClass 5745
WebAdapt.TDefaultFieldsAdapter.ImplGetVisibleActions 5745
WebAdapt.TDefaultFieldsAdapter.ImplGetVisibleFields 5745
WebAdapt.TDefaultFieldsAdapter.ImplSetActions 5746
WebAdapt.TDefaultFieldsAdapter.ImplSetData 5746
WebAdapt.TDefaultFieldsPagedAdapter 5747
WebAdapt.TDefaultFieldsPagedAdapter.AddDefaultActions 5753
WebAdapt.TDefaultFieldsPagedAdapter.CurrentPage 5760
WebAdapt.TDefaultFieldsPagedAdapter.EncodeActionParams 5754
WebAdapt.TDefaultFieldsPagedAdapter.EndOfPage 5754
WebAdapt.TDefaultFieldsPagedAdapter.ExtractRequestParams 5754
WebAdapt.TDefaultFieldsPagedAdapter.GetEOF 5755
WebAdapt.TDefaultFieldsPagedAdapter.GetFirstRecord 5755
WebAdapt.TDefaultFieldsPagedAdapter.GetNextRecord 5755
WebAdapt.TDefaultFieldsPagedAdapter.GetPageCount 5756
WebAdapt.TDefaultFieldsPagedAdapter.GetRecordCount 5756
WebAdapt.TDefaultFieldsPagedAdapter.GetRecordIndex 5756
WebAdapt.TDefaultFieldsPagedAdapter.ImplCanAddActionClass 5757
WebAdapt.TDefaultFieldsPagedAdapter.ImplEndIterator 5757
WebAdapt.TDefaultFieldsPagedAdapter.ImplGetActionsList 5757
WebAdapt.TDefaultFieldsPagedAdapter.ImplNextIteration 5758
WebAdapt.TDefaultFieldsPagedAdapter.ImplNotifyDeactivate 5758
WebAdapt.TDefaultFieldsPagedAdapter.ImplStartIterator 5759
WebAdapt.TDefaultFieldsPagedAdapter.OnGetEOF 5762
WebAdapt.TDefaultFieldsPagedAdapter.OnGetFirstRecord 5763
WebAdapt.TDefaultFieldsPagedAdapter.OnGetNextRecord 5763
WebAdapt.TDefaultFieldsPagedAdapter.OnGetRecordCount 5763
WebAdapt.TDefaultFieldsPagedAdapter.OnGetRecordIndex 5763
WebAdapt.TDefaultFieldsPagedAdapter.PageCount 5761
WebAdapt.TDefaultFieldsPagedAdapter.PageSize 5761
WebAdapt.TDefaultFieldsPagedAdapter.RecordCount 5761
WebAdapt.TDefaultFieldsPagedAdapter.RecordIndex 5762
WebAdapt.TDefaultFieldsPagedAdapter.SetPageSize 5759
WebAdapt.TDefaultFieldsPagedAdapter.SkipToPage 5760
WebAdapt.TEndUserAdapter 5764
WebAdapt.TEndUserAdapter.Actions 5770
WebAdapt.TEndUserAdapter.Data 5770
WebAdapt.TEndUserAdapter.LoginPage 5771
WebAdapt.TEndUserAdapter.OnAfterExecuteAction 5771
WebAdapt.TEndUserAdapter.OnAfterGetActionResponse 5772
WebAdapt.TEndUserAdapter.OnBeforeExecuteAction 5772
WebAdapt.TEndUserAdapter.OnBeforeGetActionResponse 5772
WebAdapt.TEndUserAdapter.OnGetActionParams 5773
WebAdapt.TEndUserAdapter.OnGetDisplayName 5773

WebAdapt.TEndUserAdapter.OnHasRights 5774
WebAdapt.TEndUserAdapter.OnIsLoggedIn 5774
WebAdapt.TEndUserAdapter.OnLogin 5775
WebAdapt.TEndUserAdapter.OnLogout 5775
WebAdapt.TEndUserAdapter.OnProduceLoginPage 5776
WebAdapt.TEndUserAdapterLoginFormAction 5776
WebAdapt.TEndUserAdapterLoginFormAction.DisplayLabel 5782
WebAdapt.TEndUserAdapterLoginFormAction.GetDefaultActionName 5780
WebAdapt.TEndUserAdapterLoginFormAction.ImplExecuteActionRequest 5781
WebAdapt.TEndUserAdapterLoginFormAction.ImplWebEnabled 5781
WebAdapt.TEndUserAdapterLoginFormAction.OnAfterExecute 5782
WebAdapt.TEndUserAdapterLoginFormAction.OnAfterGetResponse 5783
WebAdapt.TEndUserAdapterLoginFormAction.OnBeforeExecute 5783
WebAdapt.TEndUserAdapterLoginFormAction.OnBeforeGetResponse 5783
WebAdapt.TEndUserAdapterLoginFormAction.OnGetParams 5784
WebAdapt.TEndUserAdapterLogoutAction 5784
WebAdapt.TEndUserAdapterLogoutAction.DisplayLabel 5790
WebAdapt.TEndUserAdapterLogoutAction.GetDefaultActionName 5789
WebAdapt.TEndUserAdapterLogoutAction.ImplExecuteActionRequest 5789
WebAdapt.TEndUserAdapterLogoutAction.ImplWebEnabled 5789
WebAdapt.TEndUserAdapterLogoutAction.OnAfterExecute 5791
WebAdapt.TEndUserAdapterLogoutAction.OnAfterGetResponse 5791
WebAdapt.TEndUserAdapterLogoutAction.OnBeforeExecute 5791
WebAdapt.TEndUserAdapterLogoutAction.OnBeforeGetResponse 5792
WebAdapt.TEndUserAdapterLogoutAction.OnGetParams 5792
WebAdapt.TEndUserGetNameEvent 5919
WebAdapt.TEndUserGetNameEvent type 5919
WebAdapt.TEndUserGetUserIDEvent 5920
WebAdapt.TEndUserGetUserIDEvent type 5920
WebAdapt.TEndUserHandledEvent 5920
WebAdapt.TEndUserHandledEvent type 5920
WebAdapt.TEndUserIsLoggedInEvent 5921
WebAdapt.TEndUserIsLoggedInEvent type 5921
WebAdapt.TEndUserRightsEvent 5921
WebAdapt.TEndUserRightsEvent type 5921
WebAdapt.TEndUserSessionAdapter 5792
WebAdapt.TEndUserSessionAdapter.Actions 5799
WebAdapt.TEndUserSessionAdapter.Data 5800
WebAdapt.TEndUserSessionAdapter.LoginPage 5800
WebAdapt.TEndUserSessionAdapter.OnAfterExecuteAction 5800
WebAdapt.TEndUserSessionAdapter.OnAfterGetActionResponse 5801
WebAdapt.TEndUserSessionAdapter.OnBeforeExecuteAction 5801
WebAdapt.TEndUserSessionAdapter.OnBeforeGetActionResponse 5801
WebAdapt.TEndUserSessionAdapter.OnGetActionParams 5802
WebAdapt.TEndUserSessionAdapter.OnHasRights 5802
WebAdapt.TEndUserSessionAdapter.OnProduceLoginPage 5803
WebAdapt.TEndUserUserIDEvent 5922
WebAdapt.TEndUserUserIDEvent type 5922
WebAdapt.TFieldValuesAdapter 5803
WebAdapt.TFieldValuesAdapter.Create 5809
WebAdapt.TFieldValuesAdapter.Field 5812
WebAdapt.TFieldValuesAdapter.GetValueFieldClass 5809
WebAdapt.TFieldValuesAdapter.ImplEndIterator 5810
WebAdapt.TFieldValuesAdapter.ImplGetField 5810
WebAdapt.TFieldValuesAdapter.ImplGetValueIndex 5811
WebAdapt.TFieldValuesAdapter.ImplNextIteration 5811
WebAdapt.TFieldValuesAdapter.ImplStartIterator 5812
WebAdapt.TFieldValuesValueField 5813

WebAdapt.TFieldValuesValueField.Field 5817
WebAdapt.TFieldValuesValueField.GetDefaultFieldName 5816
WebAdapt.TFieldValuesValueField.ImplAsFormatted 5816
WebAdapt.TFieldValuesValueField.ImplGetValue 5817
WebAdapt.TGetEOFEvent 5922
WebAdapt.TGetEOFEvent type 5922
WebAdapt.TGetRecordCountEvent 5923
WebAdapt.TGetRecordCountEvent type 5923
WebAdapt.TGetRecordEvent 5923
WebAdapt.TGetRecordEvent type 5923
WebAdapt.TGetRecordIndexEvent 5924
WebAdapt.TGetRecordIndexEvent type 5924
WebAdapt.THandledEvent 5924
WebAdapt.THandledEvent type 5924
WebAdapt.TImageFieldGetHREFEvent 5925
WebAdapt.TImageFieldGetHREFEvent type 5925
WebAdapt.TImageFieldGetImageEvent 5925
WebAdapt.TImageFieldGetImageEvent type 5925
WebAdapt.TImageFieldGetParamsEvent 5926
WebAdapt.TImageFieldGetParamsEvent type 5926
WebAdapt.TImplementedAdapterAction 5818
WebAdapt.TIterateRecordsEvent 5926
WebAdapt.TIterateRecordsEvent type 5926
WebAdapt.TIteratorMethod 5902
WebAdapt.TIteratorMethod enumeration 5902
WebAdapt.TLogicalPageList 5819
WebAdapt.TLogicalPageList.Create 5824
WebAdapt.TLoginFormAdapter 5824
WebAdapt.TLoginFormAdapter.Actions 5830
WebAdapt.TLoginFormAdapter.Data 5831
WebAdapt.TLoginFormAdapter.OnAfterExecuteAction 5832
WebAdapt.TLoginFormAdapter.OnAfterGetActionResponse 5832
WebAdapt.TLoginFormAdapter.OnBeforeExecuteAction 5833
WebAdapt.TLoginFormAdapter.OnBeforeGetActionResponse 5833
WebAdapt.TLoginFormAdapter.OnGetActionParams 5833
WebAdapt.TLoginFormAdapter.OnLogin 5834
WebAdapt.TLoginFormAdapter.OnValidatePassword 5834
WebAdapt.TLoginFormAdapter.OnValidateUserName 5835
WebAdapt.TLoginFormAdapter.PasswordRequired 5831
WebAdapt.TLoginFormAdapterLoginAction 5835
WebAdapt.TLoginFormAdapterLoginAction.DisplayLabel 5842
WebAdapt.TLoginFormAdapterLoginAction.GetDefaultActionName 5840
WebAdapt.TLoginFormAdapterLoginAction.ImplExecuteActionRequest 5840
WebAdapt.TLoginFormAdapterLoginAction.ImplGetAdapterItemRequestParams 5841
WebAdapt.TLoginFormAdapterLoginAction.NextPage 5841
WebAdapt.TLoginFormAdapterLoginAction.OnAfterExecute 5842
WebAdapt.TLoginFormAdapterLoginAction.OnAfterGetResponse 5843
WebAdapt.TLoginFormAdapterLoginAction.OnBeforeExecute 5843
WebAdapt.TLoginFormAdapterLoginAction.OnBeforeGetResponse 5844
WebAdapt.TLoginFormAdapterLoginAction.OnGetParams 5844
WebAdapt.TLoginUserIDEvent 5927
WebAdapt.TLoginUserIDEvent type 5927
WebAdapt.TObjectList 5928
WebAdapt.TObjectList type 5928
WebAdapt.TPagedAdapter 5844
WebAdapt.TPagedAdapter.Actions 5850
WebAdapt.TPagedAdapter.Data 5850
WebAdapt.TPagedAdapter.ModifyAccess 5851
WebAdapt.TPagedAdapter.OnAfterExecuteAction 5852
WebAdapt.TPagedAdapter.OnAfterGetActionResponse 5853
WebAdapt.TPagedAdapter.OnAfterUpdateFields 5853
WebAdapt.TPagedAdapter.OnAfterValidateFields 5853
WebAdapt.TPagedAdapter.OnBeforeExecuteAction 5854
WebAdapt.TPagedAdapter.OnBeforeGetActionResponse 5854
WebAdapt.TPagedAdapter.OnBeforeUpdateFields 5854
WebAdapt.TPagedAdapter.OnBeforeValidateFields 5855
WebAdapt.TPagedAdapter.OnGetActionParams 5855
WebAdapt.TPagedAdapter.OnGetEOF 5856
WebAdapt.TPagedAdapter.OnGetFirstRecord 5856
WebAdapt.TPagedAdapter.OnGetNextRecord 5856

WebAdapt.TPagedAdapter.OnGetRecordCount 5856
WebAdapt.TPagedAdapter.OnGetRecordIndex 5857
WebAdapt.TPagedAdapter.PageSize 5851
WebAdapt.TPagedAdapter.ViewAccess 5851
WebAdapt.TStringsValuesList 5857
WebAdapt.TStringsValuesList.Actions 5862
WebAdapt.TStringsValuesList.Data 5863
WebAdapt.TStringsValuesList.OnPrepareStrings 5864
WebAdapt.TStringsValuesList.Strings 5863
WebAdapt.TUpdateFileList 5864
WebAdapt.TUpdateFileList.Count 5865
WebAdapt.TUpdateFileList.Create 5865
WebAdapt.TUpdateFileList.Files 5865
WebAdapt.TValidateAdapterFieldEvent 5928
WebAdapt.TValidateAdapterFieldEvent type 5928
WebAdapt.TValuesListNameField 5866
WebAdapt.TValuesListNameField.GetDefaultFieldName 5869
WebAdapt.TValuesListNameField.ImplGetValue 5869
WebAdapt.TValuesListValueField 5869
WebAdapt.TValuesListValueField.GetDefaultFieldName 5873
WebAdapt.TValuesListValueField.ImplGetValue 5873
WebAdapt.TWebFormGetPageNameEvent 5928
WebAdapt.TWebFormGetPageNameEvent type 5928
WebApplicationInfo
IGetWebAppComponents interface 1039
WebBroker 5935
WebBroker namespace 5935
Classes 5936
Types 5941
Variables 5942
WebBroker.Application 5942
WebBroker.Application variable 5942
WebBroker.HandleShutdownException 5942
WebBroker.HandleShutdownException variable 5942
WebBroker.THandleShutdownException 5941
WebBroker.THandleShutdownException type 5941
WebBroker.TServerExceptionEvent 5941
WebBroker.TServerExceptionEvent type 5941
WebBroker.TWebApplication 5936
WebBroker.TWebApplication.Create 5937
WebBroker.TWebApplication.CreateForm 5938
WebBroker.TWebApplication.Destroy 5938
WebBroker.TWebApplication.Initialize 5939
WebBroker.TWebApplication.Run 5939
WebBroker.TWebApplication.Title 5940
WebBrokerSOAP 5945
WebBrokerSOAP namespace 5945
Classes 5946
Functions 5951
WebBrokerSOAP.GetSOAPWebModule 5951
WebBrokerSOAP.GetSOAPWebModule function 5951
WebBrokerSOAP.THTTPSoapDispatcher 5946
WebBrokerSOAP.THTTPSoapDispatcher.Create 5947
WebBrokerSOAP.THTTPSoapDispatcher.Destroy 5948
WebBrokerSOAP.THTTPSoapDispatcher.DispatchEnabled 5948
WebBrokerSOAP.THTTPSoapDispatcher.DispatchMask 5949
WebBrokerSOAP.THTTPSoapDispatcher.DispatchMethodType 5949
WebBrokerSOAP.THTTPSoapDispatcher.DispatchRequest 5949
WebBrokerSOAP.THTTPSoapDispatcher.WebDispatch 5950
WebCntxt 5953
WebCntxt namespace 5953
Classes 5955
Functions 6003
Interfaces 5988
Structs, Records, Enums 6004
Types 6005
Variables 6007
WebCntxt.GetModuleFileNameProc 6007
WebCntxt.GetModuleFileNameProc variable 6007
WebCntxt.GetWebContextProc 6007
WebCntxt.GetWebContextProc variable 6007
WebCntxt.GetWebEndUserProc 6008
WebCntxt.GetWebEndUserProc variable 6008
WebCntxt.IWebEndUser 5988
WebCntxt.IWebEndUser.DisplayName 5994
WebCntxt.IWebEndUser.EndUserAdapter 5994

WebCntxt.IWebEndUser.GetDisplayName 5989
WebCntxt.IWebEndUser.GetEndUserAdapter 5990
WebCntxt.IWebEndUser.GetLoggedIn 5990
WebCntxt.IWebEndUser.GetLoginFormAction 5990
WebCntxt.IWebEndUser.GetLogoutAction 5991
WebCntxt.IWebEndUser.HasExecuteRights 5991
WebCntxt.IWebEndUser.HasModifyRights 5992
WebCntxt.IWebEndUser.HasViewRights 5992
WebCntxt.IWebEndUser.LoggedIn 5994
WebCntxt.IWebEndUser.LogIn 5993
WebCntxt.IWebEndUser.LoginFormAction 5995
WebCntxt.IWebEndUser.LogoutAction 5996
WebCntxt.IWebEndUser.ProduceLoginPage 5993
WebCntxt.IWebPageInfo 5996
WebCntxt.IWebPageInfo.GetHasViewAccess 5997
WebCntxt.IWebPageInfo.GetLoginRequired 5998
WebCntxt.IWebPageInfo.GetPageDescription 5998
WebCntxt.IWebPageInfo.GetPageHREF 5998
WebCntxt.IWebPageInfo.GetPageIsPublished 5999
WebCntxt.IWebPageInfo.GetPageName 5999
WebCntxt.IWebPageInfo.GetPageTitle 5999
WebCntxt.IWebPageInfo.HasViewAccess 6000
WebCntxt.IWebPageInfo.LoginRequired 6000
WebCntxt.IWebPageInfo.PageDescription 6000
WebCntxt.IWebPageInfo.PageHREF 6001
WebCntxt.IWebPageInfo.PageIsPublished 6001
WebCntxt.IWebPageInfo.PageName 6002
WebCntxt.IWebPageInfo.PageTitle 6002
WebCntxt.SetWebContextProc 6008
WebCntxt.SetWebContextProc variable 6008
WebCntxt.TAbstractAdapterRequest 5955
WebCntxt.TAbstractAdapterRequest.Create 5957
WebCntxt.TAbstractAdapterRequest.Request 5957
WebCntxt.TAbstractAdapterResponse 5958
WebCntxt.TAbstractAdapterResponse.Create 5959
WebCntxt.TAbstractAdapterResponse.Response 5960
WebCntxt.TAbstractWebContext 5960
WebCntxt.TAbstractWebContext.AdapterRequest 5971
WebCntxt.TAbstractWebContext.AdapterResponse 5972
WebCntxt.TAbstractWebContext.Create 5962
WebCntxt.TAbstractWebContext.Destroy 5963
WebCntxt.TAbstractWebContext.DispatchedPageName 5972
WebCntxt.TAbstractWebContext.EndUser 5973
WebCntxt.TAbstractWebContext.FindApplicationModule 5964
WebCntxt.TAbstractWebContext.FindDefaultPageInfo 5964
WebCntxt.TAbstractWebContext.FindModuleClass 5965
WebCntxt.TAbstractWebContext.FindModuleName 5965
WebCntxt.TAbstractWebContext.FindPage 5966
WebCntxt.TAbstractWebContext.FindPageInfo 5966
WebCntxt.TAbstractWebContext.FindVariableContainer 5967
WebCntxt.TAbstractWebContext.GetAdapterRequest 5967
WebCntxt.TAbstractWebContext.GetAdapterResponse 5967
WebCntxt.TAbstractWebContext.GetDispatchedPageName 5968
WebCntxt.TAbstractWebContext.GetEndUser 5968
WebCntxt.TAbstractWebContext.GetPageCount 5968
WebCntxt.TAbstractWebContext.GetPageInfo 5968
WebCntxt.TAbstractWebContext.GetRequestedPageName 5969
WebCntxt.TAbstractWebContext.GetSession 5969
WebCntxt.TAbstractWebContext.GetWebModules 5969
WebCntxt.TAbstractWebContext.GetWebRequest 5969
WebCntxt.TAbstractWebContext.GetWebResponse 5970
WebCntxt.TAbstractWebContext.PageCount 5973
WebCntxt.TAbstractWebContext.Pages 5973
WebCntxt.TAbstractWebContext.Request 5974
WebCntxt.TAbstractWebContext.RequestedPageName 5974
WebCntxt.TAbstractWebContext.Response 5975
WebCntxt.TAbstractWebContext.Session 5975
WebCntxt.TAbstractWebContext.SetAdapterRequest 5970
WebCntxt.TAbstractWebContext.SetAdapterResponse 5970
WebCntxt.TAbstractWebContext.SetDispatchedPageName 5970
WebCntxt.TAbstractWebContext.SetRequestedPageName 5970
WebCntxt.TAbstractWebContext.WebModules 5975
WebCntxt.TFindPageOption 6004
WebCntxt.TFindPageOption enumeration 6004
WebCntxt.TFindPageOptions 6005
WebCntxt.TFindPageOptions type 6005

WebCntxt.TGetWebContextProc 6005
WebCntxt.TGetWebContextProc type 6005
WebCntxt.TGetWebEndUserProc 6006
WebCntxt.TGetWebEndUserProc type 6006
WebCntxt.TSetWebContextProc 6006
WebCntxt.TSetWebContextProc type 6006
WebCntxt.TWebContext 5976
WebCntxt.TWebContext.Create 5978
WebCntxt.TWebContext.Destroy 5979
WebCntxt.TWebContext.FindApplicationModule 5980
WebCntxt.TWebContext.FindDefaultPageInfo 5980
WebCntxt.TWebContext.FindModuleClass 5981
WebCntxt.TWebContext.FindModuleName 5981
WebCntxt.TWebContext.FindPage 5982
WebCntxt.TWebContext.FindPageInfo 5982
WebCntxt.TWebContext.FindVariableContainer 5983
WebCntxt.TWebContext.GetAdapterRequest 5983
WebCntxt.TWebContext.GetAdapterResponse 5983
WebCntxt.TWebContext.GetDispatchedPageName 5984
WebCntxt.TWebContext.GetEndUser 5984
WebCntxt.TWebContext.GetPageCount 5984
WebCntxt.TWebContext.GetPageInfo 5984
WebCntxt.TWebContext.GetRequestedPageName 5985
WebCntxt.TWebContext.GetSession 5985
WebCntxt.TWebContext.GetWebModules 5985
WebCntxt.TWebContext.GetWebRequest 5985
WebCntxt.TWebContext.GetWebResponse 5986
WebCntxt.TWebContext.SetAdapterRequest 5986
WebCntxt.TWebContext.SetAdapterResponse 5986
WebCntxt.TWebContext.SetDispatchedPageName 5986
WebCntxt.TWebContext.SetRequestedPageName 5986
WebCntxt.WebContext 6003
WebCntxt.WebContext function 6003
WebComp 6009
WebComp namespace 6009
Classes 6011
Constants 6114
Functions 6106
Interfaces 6079
Structs, Records, Enums 6109
Types 6111
WebComp.DesigningComponent 6106
WebComp.DesigningComponent function 6106
WebComp.EnumRegisteredWebComponents 6106
WebComp.EnumRegisteredWebComponents function 6106
WebComp.EnumRegisteredWebComponentsProc 6114
WebComp.EnumRegisteredWebComponentsProc constant 6114
WebComp.FindWebComponentEditorHelper 6107
WebComp.FindWebComponentEditorHelper function 6107
WebComp.FindWebComponentEditorHelperProc 6114
WebComp.FindWebComponentEditorHelperProc constant 6114
WebComp.IAddFieldsEditor 6079
WebComp.IAddFieldsEditor.GetAddAllFieldsItem 6080
WebComp.IAddFieldsEditor.GetAddFieldsDlgCaption 6080
WebComp.IAddFieldsEditor.GetAddFieldsItem 6080
WebComp.IAddFieldsEditor.GetNewFieldPrefix 6081
WebComp.IGetWebComponentList 6081
WebComp.IGetWebComponentList.GetComponentList 6082
WebComp.IGetWebComponentList.GetDefaultComponentList 6082
WebComp.IGetWebComponentNodeName 6082
WebComp.IGetWebComponentNodeName.GetWebComponent NodeName
6083
WebComp.IGetXMLStream 6083
WebComp.IGetXMLStream.GetXMLStream 6084
WebComp.ILayoutWebContent 6084
WebComp.ILayoutWebContent.LayoutButton 6085
WebComp.ILayoutWebContent.LayoutField 6085
WebComp.ILayoutWebContent.LayoutLabelAndField 6086
WebComp.ILayoutWebContent.LayoutTable 6086
WebComp.INotifyActivateModule 6087
WebComp.INotifyActivateModule.NotifyActivateModule 6087
WebComp.INotifyActivateModule.NotifyDeactivateModule 6088
WebComp.INotifyWebActivate 6088
WebComp.INotifyWebActivate.NotifyActivate 6089
WebComp.INotifyWebActivate.NotifyDeactivate 6089
WebComp.IProducerEditorViewSupport 6089
WebComp.IProducerEditorViewSupport.GetTemplateFileType

6090
WebComp.IProducerEditorViewSupport.GetXMLData 6090
WebComp.IProducerEditorViewSupport.GetXSLData 6090
WebComp.IProducerEditorViewSupport.HasHTMLBrowserView 6091
WebComp.IProducerEditorViewSupport.HasScriptView 6091
WebComp.IProducerEditorViewSupport.HasXMLBrowserView 6091
WebComp.IProducerEditorViewSupport.HasXSLBrowserView 6091
WebComp.ISetWebContentOptions 6091
WebComp.ISetWebContentOptions.SetWebContentOptions 6092
WebComp.ITopLevelWebComponent 6092
WebComp.IWebComponent 6093
WebComp.IWebComponent.ComponentList 6096
WebComp.IWebComponent.Container 6096
WebComp.IWebComponent.GetContainer 6094
WebComp.IWebComponent.GetIndex 6094
WebComp.IWebComponent.Index 6097
WebComp.IWebComponent.SetComponentList 6095
WebComp.IWebComponent.SetContainer 6095
WebComp.IWebComponent.SetIndex 6095
WebComp.IWebComponentCommands 6097
WebComp.IWebComponentCommands.ExecuteVerb 6098
WebComp.IWebComponentCommands.GetVerb 6098
WebComp.IWebComponentCommands.GetVerbCount 6098
WebComp.IWebComponentContainer 6098
WebComp.IWebComponentContainer.ComponentCount 6100
WebComp.IWebComponentContainer.Components 6100
WebComp.IWebComponentContainer.GetComponent 6099
WebComp.IWebComponentContainer.GetComponentCount 6099
WebComp.IWebComponentEditor 6100
WebComp.IWebComponentEditor.CanAddClass 6101
WebComp.IWebComponentEditorHelper 6102
WebComp.IWebComponentEditorHelper.CanAddClassHelper 6102
WebComp.IWebContent 6103
WebComp.IWebContent.Content 6104
WebComp.IWebVariablesContainerName 6104
WebComp.IWebVariablesContainerName.ContainerName 6105
WebComp.IWebVariablesContainerName.GetContainerName 6105
WebComp.QualifyFileName 6107
WebComp.QualifyFileName function 6107
WebComp.RegisterWebComponents 6107
WebComp.RegisterWebComponents function 6107
WebComp.RegisterWebComponentsProc 6115
WebComp.RegisterWebComponentsProc constant 6115
WebComp.TEnumWebComponentsProc 6111
WebComp.TEnumWebComponentsProc type 6111
WebComp.TGetComponentListEvent 6111
WebComp.TGetComponentListEvent type 6111
WebComp.TLabelPosition 6109
WebComp.TLabelPosition enumeration 6109
WebComp.TLayout 6112
WebComp.TLayout type 6112
WebComp.TLayoutAttributes 6011
WebComp.TLayoutAttributes.ControlAttributes 6013
WebComp.TLayoutAttributes.Create 6013
WebComp.TLayoutAttributes.LabelAttributes 6013
WebComp.TLayoutAttributes.LabelPosition 6014
WebComp.TLayoutWebContent 6014
WebComp.TLayoutWebContent._AddRef 6017
WebComp.TLayoutWebContent._Release 6017
WebComp.TLayoutWebContent.Create 6016
WebComp.TLayoutWebContent.ImplLayoutButton 6017
WebComp.TLayoutWebContent.ImplLayoutField 6018
WebComp.TLayoutWebContent.ImplLayoutLabelAndField 6018
WebComp.TLayoutWebContent.ImplLayoutTable 6019
WebComp.TLayoutWebContent.LayoutButton 6020
WebComp.TLayoutWebContent.LayoutField 6020
WebComp.TLayoutWebContent.LayoutLabelAndField 6020
WebComp.TLayoutWebContent.LayoutTable 6021
WebComp.TLayoutWebContent.ParentLayout 6022
WebComp.TLayoutWebContent.QueryInterface 6021
WebComp.TSetComponentListEvent 6112
WebComp.TSetComponentListEvent type 6112
WebComp.TWebComponentClassDescription 6023
WebComp.TWebComponentClassDescription.ComponentClass

6024
WebComp.TWebComponentClassDescription.ComponentClassName 6024
WebComp.TWebComponentClassDescription.Editor 6025
WebComp.TWebComponentClassDescription.FrameClass 6025
WebComp.TWebComponentContainer 6113
WebComp.TWebComponentContainer type 6113
WebComp.TWebComponentList 6025
WebComp.TWebComponentList._AddRef 6028
WebComp.TWebComponentList._Release 6028
WebComp.TWebComponentList.Add 6029
WebComp.TWebComponentList.Changed 6029
WebComp.TWebComponentList.Clear 6029
WebComp.TWebComponentList.Count 6033
WebComp.TWebComponentList.Create 6027
WebComp.TWebComponentList.Destroy 6027
WebComp.TWebComponentList.GetComponent 6030
WebComp.TWebComponentList.GetCount 6030
WebComp.TWebComponentList.GetOwner 6030
WebComp.TWebComponentList.IndexOf 6030
WebComp.TWebComponentList.LockNotify 6031
WebComp.TWebComponentList.NewComponent 6031
WebComp.TWebComponentList.NotifyLocked 6031
WebComp.TWebComponentList.OnChange 6035
WebComp.TWebComponentList.ParentComponent 6034
WebComp.TWebComponentList.QueryInterface 6032
WebComp.TWebComponentList.Remove 6032
WebComp.TWebComponentList.SetComponentIndex 6032
WebComp.TWebComponentList.UnlockNotify 6033
WebComp.TWebComponentList.WebComponentContainer_GetComponent 6033
WebComp.TWebComponentList.WebComponentContainer_GetComponentCount 6033
WebComp.TWebComponentList.WebComponents 6034
WebComp.TWebComponentsEditorHelper 6035
WebComp.TWebComponentsEditorHelper._AddRef 6037
WebComp.TWebComponentsEditorHelper._Release 6037
WebComp.TWebComponentsEditorHelper.CanAddClassHelper 6037
WebComp.TWebComponentsEditorHelper.ImplCanAddClassHelper 6037
WebComp.TWebComponentsEditorHelper.QueryInterface 6038
WebComp.TWebContainedComponent 6038
WebComp.TWebContainedComponent.Create 6039
WebComp.TWebContainedComponent.Destroy 6040
WebComp.TWebContainedComponent.GetContainerList 6041
WebComp.TWebContainedComponent.GetParentComponent 6041
WebComp.TWebContainedComponent.HasParent 6041
WebComp.TWebContainedComponent.IsDefaultField 6042
WebComp.TWebContainedComponent.ParentChanged 6042
WebComp.TWebContainedComponent.ReadState 6043
WebComp.TWebContainedComponent.SetParentComponent 6043
WebComp.TWebContainedComponent.SetWebParent 6043
WebComp.TWebContainedComponent.WebContainedComponentHelper 6044
WebComp.TWebContainedComponent.WebParent 6044
WebComp.TWebContainedComponentHelper 6045
WebComp.TWebContainedComponentHelper.ContainerList 6051
WebComp.TWebContainedComponentHelper.Create 6047
WebComp.TWebContainedComponentHelper.Destroy 6047
WebComp.TWebContainedComponentHelper.GetContainer 6048
WebComp.TWebContainedComponentHelper.GetIndex 6048
WebComp.TWebContainedComponentHelper.IsDefaultField 6048
WebComp.TWebContainedComponentHelper.OnParentChanged 6052
WebComp.TWebContainedComponentHelper.ParentChanged 6049
WebComp.TWebContainedComponentHelper.SetComponentList 6049
WebComp.TWebContainedComponentHelper.SetContainer 6050
WebComp.TWebContainedComponentHelper.SetIndex 6050
WebComp.TWebContainedComponentHelper.SetWebParent 6050
WebComp.TWebContainedComponentHelper.WebParent 6051

WebComp.TWebContainedContainerComponent 6052
WebComp.TWebContainedContainerComponent.Create 6054
WebComp.TWebContainedContainerComponent.Destroy 6054
WebComp.TWebContainedContainerComponent.GetChildren 6055
WebComp.TWebContainedContainerComponent.GetContainerList 6056
WebComp.TWebContainedContainerComponent.GetDefaultWebComponents 6056
WebComp.TWebContainedContainerComponent.GetParentComponent 6056
WebComp.TWebContainedContainerComponent.GetWebComponents 6057
WebComp.TWebContainedContainerComponent.HasParent 6057
WebComp.TWebContainedContainerComponent.ImplGetDefaultComponentList 6058
WebComp.TWebContainedContainerComponent.IsDefaultField 6058
WebComp.TWebContainedContainerComponent.ParentChanged 6058
WebComp.TWebContainedContainerComponent.ReadState 6059
WebComp.TWebContainedContainerComponent.SetChildOrder 6059
WebComp.TWebContainedContainerComponent.SetDefaultWebComponents 6060
WebComp.TWebContainedContainerComponent.SetParentComponent 6060
WebComp.TWebContainedContainerComponent.SetWebParent 6061
WebComp.TWebContainedContainerComponent.WebContainedComponentHelper 6061
WebComp.TWebContainedContainerComponent.WebContainerComponentHelper 6062
WebComp.TWebContainedContainerComponent.WebParent 6062
WebComp.TWebContainerComponent 6063
WebComp.TWebContainerComponent.Create 6064
WebComp.TWebContainerComponent.Destroy 6064
WebComp.TWebContainerComponent.GetChildren 6065
WebComp.TWebContainerComponent.GetDefaultWebComponents 6065
WebComp.TWebContainerComponent.GetWebComponents 6065
WebComp.TWebContainerComponent.ImplGetDefaultComponentList 6066
WebComp.TWebContainerComponent.SetChildOrder 6066
WebComp.TWebContainerComponent.SetDefaultWebComponents 6067
WebComp.TWebContainerComponent.WebContainerComponentHelper 6067
WebComp.TWebContainerComponentHelper 6068
WebComp.TWebContainerComponentHelper.Create 6069
WebComp.TWebContainerComponentHelper.DefaultWebComponents 6074
WebComp.TWebContainerComponentHelper.Destroy 6070
WebComp.TWebContainerComponentHelper.GetChildren 6071
WebComp.TWebContainerComponentHelper.GetComponentList 6071
WebComp.TWebContainerComponentHelper.GetDefaultComponentList 6072
WebComp.TWebContainerComponentHelper.GetDefaultWebComponents 6072
WebComp.TWebContainerComponentHelper.OnGetDefaultComponentList 6075
WebComp.TWebContainerComponentHelper.OnSetDefaultComponentList 6075
WebComp.TWebContainerComponentHelper.SetChildOrder 6072
WebComp.TWebContainerComponentHelper.SetDefaultWebComponents 6073
WebComp.TWebContainerComponentHelper.Webcomponents 6074
WebComp.TWebContentFlag 6109

WebComp.TWebContentFlag enumeration 6109
WebComp.TWebContentFlags 6113
WebComp.TWebContentFlags type 6113
WebComp.TWebContentOptions 6076
WebComp.TWebContentOptions.Create 6077
WebComp.TWebContentOptions.Flags 6077
WebComp.UnRegisterWebComponents 6108
WebComp.UnRegisterWebComponents function 6108
WebComp.UnRegisterWebComponentsProc 6115
WebComp.UnRegisterWebComponentsProc constant 6115
WebComp.ValidateWebParent 6108
WebComp.ValidateWebParent function 6108
WebComponentContainer_GetComponent
TWebComponentList class 6033
WebComponentContainer_GetComponentCount
TWebComponentList class 6033
Webcomponents
TWebContainerComponentHelper class 6074
WebComponents
TWebComponentList class 6034
WebComponentsChange
TCustomAdapter class 5571
WebContainedComponentHelper
TWebContainedComponent class 6044
TWebContainedContainerComponent class 6061
WebContainerComponentHelper
TWebContainedContainerComponent class 6062
TWebContainerComponent class 6067
WebContent
TCustomAdapterPageProducer class 1164
WebContnrs 6117
WebContnrs namespace 6117
Classes 6118
Structs, Records, Enums 6140
Types 6141
WebContnrs.ENamedVariantsError 6118
WebContnrs.PNamedVariantItem 6141
WebContnrs.PNamedVariantItem type 6141
WebContnrs.PNamedVariantList 6141
WebContnrs.PNamedVariantList type 6141
WebContnrs.TAbstractNamedVariants 6118
WebContnrs.TAbstractNamedVariants.Add 6121
WebContnrs.TAbstractNamedVariants.Append 6122
WebContnrs.TAbstractNamedVariants.Assign 6122
WebContnrs.TAbstractNamedVariants.BeginUpdate 6122
WebContnrs.TAbstractNamedVariants.Capacity 6131
WebContnrs.TAbstractNamedVariants.Clear 6123
WebContnrs.TAbstractNamedVariants.CompareStrings 6123
WebContnrs.TAbstractNamedVariants.Count 6131
WebContnrs.TAbstractNamedVariants.DefineProperties 6124
WebContnrs.TAbstractNamedVariants.Delete 6124
WebContnrs.TAbstractNamedVariants.Destroy 6120
WebContnrs.TAbstractNamedVariants.EndUpdate 6125
WebContnrs.TAbstractNamedVariants.Equals 6125
WebContnrs.TAbstractNamedVariants.Error 6126
WebContnrs.TAbstractNamedVariants.Exchange 6126
WebContnrs.TAbstractNamedVariants.Get 6126
WebContnrs.TAbstractNamedVariants.GetCapacity 6127
WebContnrs.TAbstractNamedVariants.GetCount 6127
WebContnrs.TAbstractNamedVariants.IndexOfName 6128
WebContnrs.TAbstractNamedVariants.Insert 6128
WebContnrs.TAbstractNamedVariants.Move 6129
WebContnrs.TAbstractNamedVariants.Names 6132
WebContnrs.TAbstractNamedVariants.Put 6129
WebContnrs.TAbstractNamedVariants.SetCapacity 6130
WebContnrs.TAbstractNamedVariants.SetUpdateState 6130
WebContnrs.TAbstractNamedVariants.UpdateCount 6132
WebContnrs.TAbstractNamedVariants.Values 6133
WebContnrs.TAbstractNamedVariants.Variants 6133
WebContnrs.TNamedVariantItem 6140
WebContnrs.TNamedVariantItem record 6140
WebContnrs.TNamedVariantList 6141
WebContnrs.TNamedVariantList type 6141
WebContnrs.TNamedVariantsList 6134
WebContnrs.TNamedVariantsList.Clear 6136
WebContnrs.TNamedVariantsList.Delete 6136
WebContnrs.TNamedVariantsList.Destroy 6135
WebContnrs.TNamedVariantsList.Get 6137

WebContnrs.TNamedVariantsList.GetCapacity 6137
WebContnrs.TNamedVariantsList.GetCount 6138
WebContnrs.TNamedVariantsList.Insert 6138
WebContnrs.TNamedVariantsList.Put 6139
WebContnrs.TNamedVariantsList.SetCapacity 6139
WebDisp 6143
WebDisp namespace 6143
Classes 6146
Constants 6277
Functions 6259
Interfaces 6257
Structs, Records, Enums 6265
Types 6266
WebDisp.AdapterRequestParamsAsStrings 6259
WebDisp.AdapterRequestParamsAsStrings function 6259
WebDisp.DispatchPageName 6259
WebDisp.DispatchPageName function 6259
WebDisp.EAdapterDispatcherException 6146
WebDisp.EncodeParamNameValues 6260
WebDisp.EncodeParamNameValues function 6260
WebDisp.EncodeParamsAsHTTPQuery 6260
WebDisp.EncodeParamsAsHTTPQuery function 6260
WebDisp.EPageDispatcherException 6147
WebDisp.ESessionException 6147
WebDisp.ExpandScriptException 6260
WebDisp.ExpandScriptException function 6260
WebDisp.ExtractParamNameValues 6261
WebDisp.ExtractParamNameValues function 6261
WebDisp.FindDefaultAppPageName 6261
WebDisp.FindDefaultAppPageName function 6261
WebDisp.IWebSessionAccess 6257
WebDisp.IWebSessionAccess.GetSessionsService 6258
WebDisp.IWebSessionAccess.GetWebSession 6258
WebDisp.PageNameOfRequest 6261
WebDisp.PageNameOfRequest function 6261
WebDisp.RaiseDispatchBlankPageNameException 6262
WebDisp.RaiseDispatchBlankPageNameException function 6262
WebDisp.RaisePageDoesNotProvideContent 6262
WebDisp.RaisePageDoesNotProvideContent function 6262
WebDisp.RaisePageNotFound 6262
WebDisp.RaisePageNotFound function 6262
WebDisp.RedirectToPageName 6263
WebDisp.RedirectToPageName function 6263
WebDisp.RegisterAdapterRequestIdentifier 6263
WebDisp.RegisterAdapterRequestIdentifier function 6263
WebDisp.SafeRedirect 6263
WebDisp.SafeRedirect function 6263
WebDisp.sFailurePageParamName 6277
WebDisp.sFailurePageParamName constant 6277
WebDisp.sPageParamName 6277
WebDisp.sPageParamName constant 6277
WebDisp.sRequestIDParamName 6278
WebDisp.sRequestIDParamName constant 6278
WebDisp.sSessionCookieName 6278
WebDisp.sSessionCookieName constant 6278
WebDisp.sSuccessPageParamName 6278
WebDisp.sSuccessPageParamName constant 6278
WebDisp.TActivateWebModuleEvent 6266
WebDisp.TActivateWebModuleEvent type 6266
WebDisp.TAdapterDispatcher 6148
WebDisp.TAdapterDispatcher.OnActionRequestNotHandled 6150
WebDisp.TAdapterDispatcher.OnAfterDispatchAction 6151
WebDisp.TAdapterDispatcher.OnAfterDispatchImage 6151
WebDisp.TAdapterDispatcher.OnBeforeDispatchAction 6151
WebDisp.TAdapterDispatcher.OnBeforeDispatchImage 6152
WebDisp.TAdapterDispatcher.OnGetAdapterRequestParams 6152
WebDisp.TAdapterDispatcher.OnImageRequestNotHandled 6153
WebDisp.TAdapterDispatchParams 6153
WebDisp.TAdapterDispatchParams.Create 6156
WebDisp.TAdapterDispatchParams.Destroy 6156
WebDisp.TAdapterDispatchParams.GetFailurePage 6157
WebDisp.TAdapterDispatchParams.GetHandler 6157
WebDisp.TAdapterDispatchParams.GetObjectIdentifier 6157
WebDisp.TAdapterDispatchParams.GetParams 6158
WebDisp.TAdapterDispatchParams.GetRequestIdentifier 6158
WebDisp.TAdapterDispatchParams.GetSuccessPage 6158

WebDisp.TAdapterDispatchParams.Handler 6159
WebDisp.TAdapterDispatchParams.ObjectIdentifier 6159
WebDisp.TAdapterDispatchParams.RequestIdentifier 6160
WebDisp.TAdapterItemRequestParams 6160
WebDisp.TAdapterItemRequestParams.Create 6162
WebDisp.TAdapterItemRequestParams.Destroy 6162
WebDisp.TAdapterItemRequestParams.FailurePage 6165
WebDisp.TAdapterItemRequestParams.GetFailurePage 6163
WebDisp.TAdapterItemRequestParams.GetParamValues 6163
WebDisp.TAdapterItemRequestParams.GetSuccessPage 6164
WebDisp.TAdapterItemRequestParams.ParamValues 6165
WebDisp.TAdapterItemRequestParams.SetFailurePage 6164
WebDisp.TAdapterItemRequestParams.SetSuccessPage 6164
WebDisp.TAdapterItemRequestParams.SuccessPage 6165
WebDisp.TBaseRequestHandler 6166
WebDisp.TBaseRequestHandler.AppDispatcher 6168
WebDisp.TBaseRequestHandler.HandleRequest 6167
WebDisp.TBaseRequestHandler.ImplHandleRequest 6167
WebDisp.TBaseRequestHandler.SetAppDispatcher 6167
WebDisp.TCanViewPageEvent 6267
WebDisp.TCanViewPageEvent type 6267
WebDisp.TCustomAdapterDispatcher 6168
WebDisp.TCustomAdapterDispatcher.AddSpecialParams 6170
WebDisp.TCustomAdapterDispatcher.DoAdapterRequestNotHandled 6171
WebDisp.TCustomAdapterDispatcher.DoAfterDispatchAdapterRequest 6172
WebDisp.TCustomAdapterDispatcher.DoBeforeDispatchAdapterRequest 6172
WebDisp.TCustomAdapterDispatcher.EncodeAdapterItemRequestAsFieldValue 6173
WebDisp.TCustomAdapterDispatcher.EncodeAdapterItemRequestAsHREF 6173
WebDisp.TCustomAdapterDispatcher.GetAdapterItemRequestParams 6174
WebDisp.TCustomAdapterDispatcher.GetDispatchParams 6174
WebDisp.TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsFieldValue 6175
WebDisp.TCustomAdapterDispatcher.ImplEncodeAdapterItemRequestAsHREF 6175
WebDisp.TCustomAdapterDispatcher.ImplGetAdapterItemRequestParams 6176
WebDisp.TCustomAdapterDispatcher.ImplHandleRequest 6176
WebDisp.TCustomAdapterDispatcher.OnActionRequestNotHandled 6177
WebDisp.TCustomAdapterDispatcher.OnAfterDispatchAction 6178
WebDisp.TCustomAdapterDispatcher.OnAfterDispatchImage 6178
WebDisp.TCustomAdapterDispatcher.OnBeforeDispatchAction 6179
WebDisp.TCustomAdapterDispatcher.OnBeforeDispatchImage 6179
WebDisp.TCustomAdapterDispatcher.OnGetAdapterRequestParams 6180
WebDisp.TCustomAdapterDispatcher.OnImageRequestNotHandled 6180
WebDisp.TCustomLocateFileService 6181
WebDisp.TCustomLocateFileService.FindIncludeFile 6182
WebDisp.TCustomLocateFileService.FindStream 6183
WebDisp.TCustomLocateFileService.FindTemplateFile 6183
WebDisp.TCustomLocateFileService.GetFileStream 6184
WebDisp.TCustomLocateFileService.GetTemplateStream 6184
WebDisp.TCustomLocateFileService.ImplFindIncludeFile 6185
WebDisp.TCustomLocateFileService.ImplGetTemplateStream 6185
WebDisp.TCustomLocateFileService.OnFindIncludeFile 6186
WebDisp.TCustomLocateFileService.OnFindStream 6187
WebDisp.TCustomLocateFileService.OnFindTemplateFile 6187
WebDisp.TCustomPageDispatcher 6188
WebDisp.TCustomPageDispatcher.DefaultPage 6195
WebDisp.TCustomPageDispatcher.DispatchPageName 6190
WebDisp.TCustomPageDispatcher.EndUser 6190
WebDisp.TCustomPageDispatcher.GetCanView 6191
WebDisp.TCustomPageDispatcher.GetDefaultPageName 6191
WebDisp.TCustomPageDispatcher.GetLoginRequired 6191
WebDisp.TCustomPageDispatcher.GetRequestedPage 6192

WebDisp.TCustomPageDispatcher.ImplDispatchPageName 6192
WebDisp.TCustomPageDispatcher.ImplGetCanView 6192
WebDisp.TCustomPageDispatcher.ImplGetDefaultPageName 6192
WebDisp.TCustomPageDispatcher.ImplGetLoginRequired 6193
WebDisp.TCustomPageDispatcher.ImplHandleRequest 6193
WebDisp.TCustomPageDispatcher.ImplPageNameOfRequest 6193
WebDisp.TCustomPageDispatcher.ImplRedirectToPageName 6193
WebDisp.TCustomPageDispatcher.IsLoggedIn 6194
WebDisp.TCustomPageDispatcher.OnAfterDispatchPage 6196
WebDisp.TCustomPageDispatcher.OnAfterRedirectToPage 6197
WebDisp.TCustomPageDispatcher.OnBeforeDispatchPage 6197
WebDisp.TCustomPageDispatcher.OnBeforeRedirectToPage 6198
WebDisp.TCustomPageDispatcher.OnCanViewPage 6198
WebDisp.TCustomPageDispatcher.OnIsLoginRequired 6199
WebDisp.TCustomPageDispatcher.OnPageAccessDenied 6200
WebDisp.TCustomPageDispatcher.OnPageNotFound 6200
WebDisp.TCustomPageDispatcher.PageNameOfRequest 6194
WebDisp.TCustomPageDispatcher.ProduceLoginPage 6194
WebDisp.TCustomPageDispatcher.RedirectToPageName 6194
WebDisp.TCustomPageDispatcher.ValidatePage 6195
WebDisp.TCustomWebAppComponents 6201
WebDisp.TCustomWebAppComponents.AdapterDispatcher 6211
WebDisp.TCustomWebAppComponents.ApplicationAdapter 6212
WebDisp.TCustomWebAppComponents.CreateContext 6204
WebDisp.TCustomWebAppComponents.DispatchActions 6212
WebDisp.TCustomWebAppComponents.EndUserAdapter 6213
WebDisp.TCustomWebAppComponents.GetAdapterDispatcher 6205
WebDisp.TCustomWebAppComponents.GetLocateFileService 6205
WebDisp.TCustomWebAppComponents.GetPageDispatcher 6205
WebDisp.TCustomWebAppComponents.GetWebApplicationInfo 6206
WebDisp.TCustomWebAppComponents.GetWebEndUser 6206
WebDisp.TCustomWebAppComponents.GetWebUserList 6206
WebDisp.TCustomWebAppComponents.ImplFinishContext 6207
WebDisp.TCustomWebAppComponents.ImplHandleException 6207
WebDisp.TCustomWebAppComponents.ImplHandleRequest 6208
WebDisp.TCustomWebAppComponents.ImplInitContext 6208
WebDisp.TCustomWebAppComponents.InitModule 6209
WebDisp.TCustomWebAppComponents.InitModules 6210
WebDisp.TCustomWebAppComponents.ListRequestHandlers 6210
WebDisp.TCustomWebAppComponents.LocateFileService 6213
WebDisp.TCustomWebAppComponents.Notification 6211
WebDisp.TCustomWebAppComponents.OnException 6215
WebDisp.TCustomWebAppComponents.PageDispatcher 6214
WebDisp.TCustomWebAppComponents.Sessions 6214
WebDisp.TCustomWebAppComponents.UserListService 6215
WebDisp.TDispatchActionEvent 6267
WebDisp.TDispatchActionEvent type 6267
WebDisp.TDispatchActionHandledEvent 6268
WebDisp.TDispatchActionHandledEvent type 6268
WebDisp.TDispatchImageEvent 6269
WebDisp.TDispatchImageEvent type 6269
WebDisp.TDispatchImageHandledEvent 6269
WebDisp.TDispatchImageHandledEvent type 6269
WebDisp.TDispatchPageEvent 6270
WebDisp.TDispatchPageEvent type 6270
WebDisp.TDispatchPageHandledEvent 6270
WebDisp.TDispatchPageHandledEvent type 6270
WebDisp.TDispatchPageParamsHandledEvent 6271
WebDisp.TDispatchPageParamsHandledEvent type 6271
WebDisp.TGetAdapterRequestParamsEvent 6272
WebDisp.TGetAdapterRequestParamsEvent type 6272
WebDisp.TInvalidWebSession 6216
WebDisp.TInvalidWebSession.Create 6217
WebDisp.TInvalidWebSession.GetSessionID 6217
WebDisp.TInvalidWebSession.GetTimeoutMinutes 6218
WebDisp.TInvalidWebSession.GetValue 6218
WebDisp.TInvalidWebSession.SetTimeoutMinutes 6219

WebDisp.TInvalidWebSession.SetValue 6219
WebDisp.TInvalidWebSession.Terminate 6219
WebDisp.TInvalidWebSession.UpdateResponse 6220
WebDisp.TLocateFileService 6220
WebDisp.TLocateFileService.OnFindIncludeFile 6221
WebDisp.TLocateFileService.OnFindStream 6222
WebDisp.TLocateFileService.OnFindTemplateFile 6222
WebDisp.TLocateFileServiceFindFileEvent 6273
WebDisp.TLocateFileServiceFindFileEvent type 6273
WebDisp.TLocateFileServiceFindStreamEvent 6273
WebDisp.TLocateFileServiceFindStreamEvent type 6273
WebDisp.TLoginPagedHandledEvent 6274
WebDisp.TLoginPagedHandledEvent type 6274
WebDisp.TLoginRequiredEvent 6274
WebDisp.TLoginRequiredEvent type 6274
WebDisp.TMultiModuleWebAppServices 6223
WebDisp.TMultiModuleWebAppServices.Access_Request 6225
WebDisp.TMultiModuleWebAppServices.Access_Response 6225
WebDisp.TMultiModuleWebAppServices.CreateContext 6225
WebDisp.TMultiModuleWebAppServices.DoAfterDispatch 6226
WebDisp.TMultiModuleWebAppServices.DoBeforeDispatch 6226
WebDisp.TMultiModuleWebAppServices.DoneModule 6227
WebDisp.TMultiModuleWebAppServices.FinishContext 6227
WebDisp.TMultiModuleWebAppServices.GetExceptionHandler 6228
WebDisp.TMultiModuleWebAppServices.HandleException 6228
WebDisp.TMultiModuleWebAppServices.HandleRequest 6229
WebDisp.TMultiModuleWebAppServices.ImplFinishContext 6229
WebDisp.TMultiModuleWebAppServices.ImplHandleException 6230
WebDisp.TMultiModuleWebAppServices.ImplHandleRequest 6230
WebDisp.TMultiModuleWebAppServices.ImplInitContext 6231
WebDisp.TMultiModuleWebAppServices.InitContext 6231
WebDisp.TMultiModuleWebAppServices.InitModule 6232
WebDisp.TMultiModuleWebAppServices.InitModules 6233
WebDisp.TMultiModuleWebAppServices.ModuleAdded 6233
WebDisp.TMultiModuleWebAppServices.OnAfterDispatch 6235
WebDisp.TMultiModuleWebAppServices.OnBeforeDispatch 6235
WebDisp.TMultiModuleWebAppServices.Request 6234
WebDisp.TMultiModuleWebAppServices.Response 6234
WebDisp.TMultiModuleWebAppServices.WebModules 6234
WebDisp.TPageAccessDenied 6265
WebDisp.TPageAccessDenied enumeration 6265
WebDisp.TPageAccessDeniedEvent 6275
WebDisp.TPageAccessDeniedEvent type 6275
WebDisp.TPageDispatcher 6236
WebDisp.TPageDispatcher.DefaultPage 6238
WebDisp.TPageDispatcher.OnAfterDispatchPage 6239
WebDisp.TPageDispatcher.OnBeforeDispatchPage 6239
WebDisp.TPageDispatcher.OnCanViewPage 6240
WebDisp.TPageDispatcher.OnIsLoginRequired 6241
WebDisp.TPageDispatcher.OnPageAccessDenied 6241
WebDisp.TPageDispatcher.OnPageNotFound 6242
WebDisp.TWebAppComponents 6243
WebDisp.TWebAppComponents.AdapterDispatcher 6245
WebDisp.TWebAppComponents.ApplicationAdapter 6246
WebDisp.TWebAppComponents.DispatchActions 6246
WebDisp.TWebAppComponents.EndUserAdapter 6247
WebDisp.TWebAppComponents.LocateFileService 6247
WebDisp.TWebAppComponents.OnAfterDispatch 6249
WebDisp.TWebAppComponents.OnBeforeDispatch 6250
WebDisp.TWebAppComponents.OnException 6250
WebDisp.TWebAppComponents.PageDispatcher 6248
WebDisp.TWebAppComponents.Sessions 6248
WebDisp.TWebAppComponents.UserListService 6249
WebDisp.TWebRequest 6276
WebDisp.TWebRequest type 6276
WebDisp.TWebResponse 6276
WebDisp.TWebResponse type 6276
WebDisp.TWebSession 6251
WebDisp.TWebSession.Create 6252
WebDisp.TWebSession.GetSessionID 6253
WebDisp.TWebSession.GetTimeoutMinutes 6253
WebDisp.TWebSession.GetValue 6254
WebDisp.TWebSession.SetTimeoutMinutes 6254
WebDisp.TWebSession.SetValue 6254

WebDisp.TWebSession.Terminate 6255
WebDisp.TWebSession.UpdateResponse 6255
WebDispatch
THTTPSoapDispatcher class 5950
TWSDLHTMLPublish class 7288
TXMLBroker class 7328
WebEnabled
IWebEnabled interface 1084
TBaseAdapterAction class 5473
WebEndUser
IGetWebAppComponents interface 1040
WebFact 6279
WebFact namespace 6279
Classes 6280
Structs, Records, Enums 6305
Types 6306
WebFact.TBaseWebModuleFactory 6280
WebFact.TBaseWebModuleFactory.Create 6281
WebFact.TBaseWebModuleFactory.GetCacheMode 6282
WebFact.TBaseWebModuleFactory.GetComponentClass 6282
WebFact.TBaseWebModuleFactory.GetCreateMode 6283
WebFact.TBaseWebModuleFactory.GetIsAppModule 6283
WebFact.TBaseWebModuleFactory.GetModule 6283
WebFact.TBaseWebModuleFactory.GetModuleName 6284
WebFact.TBaseWebModuleFactory.PreventDestruction 6284
WebFact.TBaseWebPageInfo 6285
WebFact.TBaseWebPageInfo.Create 6286
WebFact.TBaseWebPageInfo.GetIsLoginRequired 6287
WebFact.TBaseWebPageInfo.GetIsPublished 6287
WebFact.TBaseWebPageInfo.GetPageDescription 6287
WebFact.TBaseWebPageInfo.GetPageFile 6288
WebFact.TBaseWebPageInfo.GetPageHREF 6288
WebFact.TBaseWebPageInfo.GetPageName 6288
WebFact.TBaseWebPageInfo.GetPageTitle 6289
WebFact.TBaseWebPageInfo.GetViewAccess 6289
WebFact.TBaseWebPageInfo.SetFactory 6290
WebFact.TBaseWebPageModuleFactory 6290
WebFact.TBaseWebPageModuleFactory.Create 6291
WebFact.TBaseWebPageModuleFactory.GetCacheMode 6292
WebFact.TBaseWebPageModuleFactory.GetComponentClass 6292
WebFact.TBaseWebPageModuleFactory.GetCreateMode 6293
WebFact.TBaseWebPageModuleFactory.GetIsAppModule 6293
WebFact.TBaseWebPageModuleFactory.GetModule 6293
WebFact.TBaseWebPageModuleFactory.GetModuleName 6294
WebFact.TBaseWebPageModuleFactory.PreventDestruction 6294
WebFact.TWebAppDataModuleFactory 6295
WebFact.TWebAppDataModuleFactory.Create 6296
WebFact.TWebAppDataModuleFactory.GetIsAppModule 6296
WebFact.TWebAppPageModuleFactory 6297
WebFact.TWebAppPageModuleFactory.Create 6298
WebFact.TWebAppPageModuleFactory.GetIsAppModule 6299
WebFact.TWebDataModuleFactory 6299
WebFact.TWebDataModuleFactory.Create 6300
WebFact.TWebPageAccess 6306
WebFact.TWebPageAccess type 6306
WebFact.TWebPageAccessFlags 6305
WebFact.TWebPageAccessFlags enumeration 6305
WebFact.TWebPageInfo 6301
WebFact.TWebPageInfo.Create 6302
WebFact.TWebPageModuleFactory 6303
WebFact.TWebPageModuleFactory.Create 6304
WebFields
TAdapterFields class 5281
WebForm 6307
WebForm namespace 6307
Classes 6310
Constants 6530
Functions 6519
Interfaces 6510
Structs, Records, Enums 6522
Types 6528
WebForm.CalcDisplayFieldHTMLElementType 6519
WebForm.CalcDisplayFieldHTMLElementType function 6519
WebForm.CalcDisplayFieldViewMode 6519
WebForm.CalcDisplayFieldViewMode function 6519
WebForm.FindVariablesContainer 6520
WebForm.FindVariablesContainer function 6520

WebForm.FindVariablesContainerInParent 6520
WebForm.FindVariablesContainerInParent function 6520
WebForm.GetAdapterModeOfAdapterInParent 6520
WebForm.GetAdapterModeOfAdapterInParent function 6520
WebForm.IAdapterDisplay 6510
WebForm.IGetAdapterFormLayout 6510
WebForm.IGetAdapterFormLayout.GetAdapterFormLayout 6511
WebForm.IGetAdapterModeOfDisplay 6511
WebForm.IGetAdapterModeOfDisplay.GetAdapterModeOfAdapter 6512
WebForm.IGetAdapterModeOfDisplay.GetAdapterModeOfDisplay 6512
WebForm.IGetAdapterOfDisplay 6512
WebForm.IGetAdapterOfDisplay.GetAdapter 6513
WebForm.IGetAdapterVariableName 6513
WebForm.IGetAdapterVariableName.AdapterVariableName 6515
WebForm.IGetAdapterVariableName.GetAdapterVariableName 6514
WebForm.IGetControlRequirements 6515
WebForm.IGetControlRequirements.GetControlRequirements 6516
WebForm.IGetFormRequirements 6516
WebForm.IGetFormRequirements.GetFormRequirements 6517
WebForm.IGetVariablesContainerOfDisplay 6517
WebForm.IGetVariablesContainerOfDisplay.GetVariablesContainer 6518
WebForm.MakeAdapterHTMLName 6521
WebForm.MakeAdapterHTMLName function 6521
WebForm.MakeAdapterVariableName 6521
WebForm.MakeAdapterVariableName function 6521
WebForm.sDisplayText 6530
WebForm.sDisplayText constant 6530
WebForm.sFieldText 6530
WebForm.sFieldText constant 6530
WebForm.sFormGet 6531
WebForm.sFormGet constant 6531
WebForm.sFormPost 6531
WebForm.sFormPost constant 6531
WebForm.sMultipartForm 6531
WebForm.sMultipartForm constant 6531
WebForm.TAbstractControlRequirements 6311
WebForm.TAbstractControlRequirements.DefineFunction 6312
WebForm.TAbstractControlRequirements.DefineVariable 6312
WebForm.TAbstractControlRequirements.IsFunctionDefined 6313
WebForm.TAbstractControlRequirements.IsVariableDefined 6313
WebForm.TAbstractControlRequirements.RequiresMultipartForm 6314
WebForm.TAbstractFormRequirements 6314
WebForm.TAbstractFormRequirements.DefineFunction 6315
WebForm.TAbstractFormRequirements.DefineVariable 6315
WebForm.TAbstractFormRequirements.IsFunctionDefined 6315
WebForm.TAbstractFormRequirements.IsVariableDefined 6316
WebForm.TActionButtonHideOption 6522
WebForm.TActionButtonHideOption enumeration 6522
WebForm.TActionButtonHideOptions 6528
WebForm.TActionButtonHideOptions type 6528
WebForm.TAdapterActionButton 6316
WebForm.TAdapterActionButton.ActionName 6318
WebForm.TAdapterActionButton.Align 6318
WebForm.TAdapterActionButton.Caption 6319
WebForm.TAdapterActionButton.Custom 6319
WebForm.TAdapterActionButton.DisplayColumns 6320
WebForm.TAdapterActionButton.DisplayType 6320
WebForm.TAdapterActionButton.ErrorPageName 6321
WebForm.TAdapterActionButton.HideOptions 6321
WebForm.TAdapterActionButton.ImageHeight 6321
WebForm.TAdapterActionButton.ImageProducer 6322
WebForm.TAdapterActionButton.ImageWidth 6322
WebForm.TAdapterActionButton.PageName 6323
WebForm.TAdapterActionButton.Style 6323
WebForm.TAdapterActionButton.StyleRule 6323
WebForm.TAdapterActionButton.VAlign 6324
WebForm.TAdapterActionsListHelper 6324
WebForm.TAdapterActionsListHelper.AddDefaultActions 6331
WebForm.TAdapterActionsListHelper.Create 6326
WebForm.TAdapterActionsListHelper.FindAction 6327

WebForm.TAdapterActionsListHelper.GetActionsList 6327
WebForm.TAdapterActionsListHelper.GetAddAllFieldsItem 6328
WebForm.TAdapterActionsListHelper.GetAddFieldsDlgCaption 6328
WebForm.TAdapterActionsListHelper.GetAddFieldsItem 6329
WebForm.TAdapterActionsListHelper.GetNewFieldPrefix 6329
WebForm.TAdapterActionsListHelper.GetVisibleActions 6329
WebForm.TAdapterActionsListHelper.IsActionInUse 6330
WebForm.TAdapterCommandColumn 6331
WebForm.TAdapterCommandColumn.AddDefaultCommands 6334
WebForm.TAdapterCommandColumn.Align 6334
WebForm.TAdapterCommandColumn.Caption 6334
WebForm.TAdapterCommandColumn.CaptionAttributes 6335
WebForm.TAdapterCommandColumn.Custom 6335
WebForm.TAdapterCommandColumn.DisplayColumns 6335
WebForm.TAdapterCommandColumn.Style 6336
WebForm.TAdapterCommandColumn.StyleRule 6336
WebForm.TAdapterCommandColumn.VAlign 6337
WebForm.TAdapterCommandGroup 6337
WebForm.TAdapterCommandGroup.AddDefaultCommands 6340
WebForm.TAdapterCommandGroup.Custom 6340
WebForm.TAdapterCommandGroup.DisplayComponent 6340
WebForm.TAdapterCommandGroup.Style 6341
WebForm.TAdapterCommandGroup.StyleRule 6341
WebForm.TAdapterControlGroup 6341
WebForm.TAdapterControlGroup.Adapter 6352
WebForm.TAdapterControlGroup.AdapterChange 6345
WebForm.TAdapterControlGroup.AdapterMode 6353
WebForm.TAdapterControlGroup.AddDefaultFields 6353
WebForm.TAdapterControlGroup.AddNotify 6345
WebForm.TAdapterControlGroup.Create 6343
WebForm.TAdapterControlGroup.CreateDefaultFields 6346
WebForm.TAdapterControlGroup.Destroy 6344
WebForm.TAdapterControlGroup.FindField 6346
WebForm.TAdapterControlGroup.GetAdapter 6346
WebForm.TAdapterControlGroup.GetAdapterModeOfAdapter 6347
WebForm.TAdapterControlGroup.GetAdapterModeOfDisplay 6347
WebForm.TAdapterControlGroup.GetAdapterVariableName 6348
WebForm.TAdapterControlGroup.GetControlRequirements 6348
WebForm.TAdapterControlGroup.GetDesigntimeWarnings 6348
WebForm.TAdapterControlGroup.GetFieldsList 6349
WebForm.TAdapterControlGroup.GetVisibleFields 6349
WebForm.TAdapterControlGroup.ImplGetControlRequirements 6350
WebForm.TAdapterControlGroup.IsFieldInUse 6350
WebForm.TAdapterControlGroup.Notification 6350
WebForm.TAdapterControlGroup.NotifyAdapterChange 6351
WebForm.TAdapterControlGroup.RemoveNotify 6351
WebForm.TAdapterControlGroup.SetAdapter 6352
WebForm.TAdapterControlGroup.VisibleFields 6354
WebForm.TAdapterDataDisplay 6354
WebForm.TAdapterDataDisplay.Align 6362
WebForm.TAdapterDataDisplay.Create 6356
WebForm.TAdapterDataDisplay.FieldConditions 6357
WebForm.TAdapterDataDisplay.FieldName 6363
WebForm.TAdapterDataDisplay.FindAssociatedField 6357
WebForm.TAdapterDataDisplay.FormatCaption 6357
WebForm.TAdapterDataDisplay.FormatColumnData 6358
WebForm.TAdapterDataDisplay.FormatColumnHeading 6358
WebForm.TAdapterDataDisplay.GetAdapterModeOfAdapter 6358
WebForm.TAdapterDataDisplay.GetAdapterVariableName 6358
WebForm.TAdapterDataDisplay.GetCaption 6358
WebForm.TAdapterDataDisplay.GetControlRequirements 6359
WebForm.TAdapterDataDisplay.GetDesigntimeWarnings 6359
WebForm.TAdapterDataDisplay.GetFieldName 6359
WebForm.TAdapterDataDisplay.GetVariablesContainer 6360
WebForm.TAdapterDataDisplay.HideOptions 6363
WebForm.TAdapterDataDisplay.ImplContent 6360
WebForm.TAdapterDataDisplay.ImplFormatColumnData 6360
WebForm.TAdapterDataDisplay.ImplFormatColumnHeading 6361
WebForm.TAdapterDataDisplay.ImplGetControlRequirements 6361
WebForm.TAdapterDataDisplay.ImplRestoreDefaults 6361
WebForm.TAdapterDataDisplay.RestoreDefaults 6361
WebForm.TAdapterDataDisplay.SetCaption 6362

WebForm.TAdapterDataDisplay.SetFieldName 6362
WebForm.TAdapterDataDisplay.VAlign 6363
WebForm.TAdapterDataDisplayClass 6528
WebForm.TAdapterDataDisplayClass type 6528
WebForm.TAdapterDisplayColumn 6364
WebForm.TAdapterDisplayColumn.Align 6366
WebForm.TAdapterDisplayColumn.Caption 6367
WebForm.TAdapterDisplayColumn.CaptionAttributes 6367
WebForm.TAdapterDisplayColumn.Custom 6367
WebForm.TAdapterDisplayColumn.DisplayType 6368
WebForm.TAdapterDisplayColumn.FieldName 6368
WebForm.TAdapterDisplayColumn.HideOptions 6368
WebForm.TAdapterDisplayColumn.ImageHeight 6369
WebForm.TAdapterDisplayColumn.ImageWidth 6369
WebForm.TAdapterDisplayColumn.Style 6369
WebForm.TAdapterDisplayColumn.StyleRule 6370
WebForm.TAdapterDisplayColumn.VAlign 6370
WebForm.TAdapterDisplayField 6371
WebForm.TAdapterDisplayField.Align 6374
WebForm.TAdapterDisplayField.Caption 6374
WebForm.TAdapterDisplayField.CaptionAttributes 6374
WebForm.TAdapterDisplayField.CaptionPosition 6375
WebForm.TAdapterDisplayField.Custom 6375
WebForm.TAdapterDisplayField.DisplayColumns 6376
WebForm.TAdapterDisplayField.DisplayRows 6376
WebForm.TAdapterDisplayField.DisplayType 6376
WebForm.TAdapterDisplayField.DisplayWidth 6377
WebForm.TAdapterDisplayField.FieldName 6377
WebForm.TAdapterDisplayField.HideOptions 6377
WebForm.TAdapterDisplayField.ImageHeight 6377
WebForm.TAdapterDisplayField.ImageWidth 6378
WebForm.TAdapterDisplayField.InputType 6378
WebForm.TAdapterDisplayField.MaxLength 6378
WebForm.TAdapterDisplayField.SelectRows 6379
WebForm.TAdapterDisplayField.Style 6379
WebForm.TAdapterDisplayField.StyleRule 6379
WebForm.TAdapterDisplayField.TextAreaWrap 6380
WebForm.TAdapterDisplayField.VAlign 6380
WebForm.TAdapterDisplayField.ViewMode 6381
WebForm.TAdapterDisplayReferenceGroup 6381
WebForm.TAdapterDisplayReferenceGroup.AdapterChange 6383
WebForm.TAdapterDisplayReferenceGroup.Create 6382
WebForm.TAdapterDisplayReferenceGroup.Destroy 6382
WebForm.TAdapterDisplayReferenceGroup.DisplayComponent 6386
WebForm.TAdapterDisplayReferenceGroup.DisplayComponentChange 6384
WebForm.TAdapterDisplayReferenceGroup.GetAdapterDisplayCharacteristics 6384
WebForm.TAdapterDisplayReferenceGroup.GetDesigntimeWarnings 6384
WebForm.TAdapterDisplayReferenceGroup.NotifyAdapterChange 6385
WebForm.TAdapterDisplayReferenceGroup.OnDisplayComponentChange 6385
WebForm.TAdapterDisplayReferenceGroup.SetWebDisplayComponent 6386
WebForm.TAdapterEditColumn 6387
WebForm.TAdapterEditColumn.Align 6389
WebForm.TAdapterEditColumn.Caption 6390
WebForm.TAdapterEditColumn.CaptionAttributes 6390
WebForm.TAdapterEditColumn.CaptionPosition 6390
WebForm.TAdapterEditColumn.Custom 6391
WebForm.TAdapterEditColumn.DisplayColumns 6391
WebForm.TAdapterEditColumn.DisplayRows 6392
WebForm.TAdapterEditColumn.DisplayType 6392
WebForm.TAdapterEditColumn.DisplayWidth 6392
WebForm.TAdapterEditColumn.FieldName 6393
WebForm.TAdapterEditColumn.HideOptions 6393
WebForm.TAdapterEditColumn.ImageHeight 6393
WebForm.TAdapterEditColumn.ImageWidth 6393
WebForm.TAdapterEditColumn.InputType 6394
WebForm.TAdapterEditColumn.MaxLength 6394
WebForm.TAdapterEditColumn.SelectRows 6394
WebForm.TAdapterEditColumn.Style 6395
WebForm.TAdapterEditColumn.StyleRule 6395

WebForm.TAdapterEditColumn.TextAreaWrap 6396
WebForm.TAdapterEditColumn.VAlign 6396
WebForm.TAdapterEditColumn.ViewMode 6396
WebForm.TAdapterErrorList 6397
WebForm.TAdapterErrorList.Adapter 6398
WebForm.TAdapterFieldGroup 6399
WebForm.TAdapterFieldGroup.Adapter 6402
WebForm.TAdapterFieldGroup.AdapterMode 6402
WebForm.TAdapterFieldGroup.AddDefaultFields 6403
WebForm.TAdapterFieldGroup.Custom 6403
WebForm.TAdapterFieldGroup.Style 6404
WebForm.TAdapterFieldGroup.StyleRule 6404
WebForm.TAdapterForm 6404
WebForm.TAdapterForm.Custom 6406
WebForm.TAdapterForm.FormMethod 6406
WebForm.TAdapterForm.Style 6407
WebForm.TAdapterForm.StyleRule 6407
WebForm.TAdapterFormLayout 6408
WebForm.TAdapterFormLayout.BreakButtons 6417
WebForm.TAdapterFormLayout.ChildTableIndent 6417
WebForm.TAdapterFormLayout.ColumnCount 6417
WebForm.TAdapterFormLayout.Create 6410
WebForm.TAdapterFormLayout.EndButtons 6411
WebForm.TAdapterFormLayout.EndField 6411
WebForm.TAdapterFormLayout.EndFields 6411
WebForm.TAdapterFormLayout.EndLayout 6412
WebForm.TAdapterFormLayout.EndRow 6412
WebForm.TAdapterFormLayout.EndTable 6412
WebForm.TAdapterFormLayout.FieldIndent 6418
WebForm.TAdapterFormLayout.GetAdapterFormLayout 6412
WebForm.TAdapterFormLayout.GetChildTableIndent 6413
WebForm.TAdapterFormLayout.GetFieldIndent 6413
WebForm.TAdapterFormLayout.GetTableIndent 6413
WebForm.TAdapterFormLayout.ImplLayoutButton 6413
WebForm.TAdapterFormLayout.ImplLayoutField 6414
WebForm.TAdapterFormLayout.ImplLayoutLabelAndField 6414
WebForm.TAdapterFormLayout.ImplLayoutTable 6415
WebForm.TAdapterFormLayout.NextColumn 6415
WebForm.TAdapterFormLayout.StartButtons 6415
WebForm.TAdapterFormLayout.StartFields 6416
WebForm.TAdapterFormLayout.StartRow 6416
WebForm.TAdapterFormLayout.StartTable 6416
WebForm.TAdapterFormLayout.TableHeader 6418
WebForm.TAdapterFormLayout.TableIndent 6419
WebForm.TAdapterGrid 6419
WebForm.TAdapterGrid.Adapter 6422
WebForm.TAdapterGrid.AdapterMode 6423
WebForm.TAdapterGrid.AddDefaultFields 6423
WebForm.TAdapterGrid.HeadingAttributes 6424
WebForm.TAdapterGrid.RowAttributes 6424
WebForm.TAdapterGrid.TableAttributes 6425
WebForm.TColumnHTMLElementType 6523
WebForm.TColumnHTMLElementType enumeration 6523
WebForm.TCommandHTMLElementType 6523
WebForm.TCommandHTMLElementType enumeration 6523
WebForm.TControlRequirements 6425
WebForm.TControlRequirements.Create 6426
WebForm.TControlRequirements.DefineFunction 6427
WebForm.TControlRequirements.DefineVariable 6428
WebForm.TControlRequirements.Destroy 6427
WebForm.TControlRequirements.IsFunctionDefined 6428
WebForm.TControlRequirements.IsVariableDefined 6429
WebForm.TControlRequirements.NeedMultipartForm 6429
WebForm.TControlRequirements.RequiresMultipartForm 6429
WebForm.TCustomAdapterActionButton 6430
WebForm.TCustomAdapterActionButton.ActionName 6438
WebForm.TCustomAdapterActionButton.Align 6438
WebForm.TCustomAdapterActionButton.Create 6431
WebForm.TCustomAdapterActionButton.Destroy 6432
WebForm.TCustomAdapterActionButton.DisplayColumns 6438
WebForm.TCustomAdapterActionButton.DisplayType 6439
WebForm.TCustomAdapterActionButton.ErrorPageName 6439
WebForm.TCustomAdapterActionButton.FindAssociatedAction 6433
WebForm.TCustomAdapterActionButton.GetActionName 6433
WebForm.TCustomAdapterActionButton.GetAdapterModeOfAdapter 6433
WebForm.TCustomAdapterActionButton.GetAdapterVariableName

6433

WebForm.TCustomAdapterActionButton.GetCaption 6434

WebForm.TCustomAdapterActionButton.GetControlRequirements
6434

WebForm.TCustomAdapterActionButton.GetDesigntimeWarnings
6434

WebForm.TCustomAdapterActionButton.GetHTMLFormVarName
6434

WebForm.TCustomAdapterActionButton.GetLayoutAttributes
6435

WebForm.TCustomAdapterActionButton.HideOptions 6440

WebForm.TCustomAdapterActionButton.ImageHeight 6440

WebForm.TCustomAdapterActionButton.ImageProducer 6441

WebForm.TCustomAdapterActionButton.ImageWidth 6441

WebForm.TCustomAdapterActionButton.ImplContent 6435

WebForm.TCustomAdapterActionButton.ImplGetControlRequirements
6435

WebForm.TCustomAdapterActionButton.ImplRestoreDefaults
6436

WebForm.TCustomAdapterActionButton.Notification 6436

WebForm.TCustomAdapterActionButton.PageName 6441

WebForm.TCustomAdapterActionButton.RestoreDefaults 6436

WebForm.TCustomAdapterActionButton.SetActionName 6437

WebForm.TCustomAdapterActionButton.SetCaption 6437

WebForm.TCustomAdapterActionButton.VAlign 6442

WebForm.TCustomAdapterCommandColumn 6442

WebForm.TCustomAdapterCommandColumn.AdapterActionsListHelper
6449

WebForm.TCustomAdapterCommandColumn.AddDefaultCommands
6449

WebForm.TCustomAdapterCommandColumn.Align 6450

WebForm.TCustomAdapterCommandColumn.Caption 6450

WebForm.TCustomAdapterCommandColumn.CaptionAttributes
6451

WebForm.TCustomAdapterCommandColumn.Create 6444

WebForm.TCustomAdapterCommandColumn.Custom 6451

WebForm.TCustomAdapterCommandColumn.Destroy 6445

WebForm.TCustomAdapterCommandColumn.DisplayColumns
6451

WebForm.TCustomAdapterCommandColumn.FormatColumnData
6445

WebForm.TCustomAdapterCommandColumn.FormatColumnHeading
6446

WebForm.TCustomAdapterCommandColumn.GetAdapterDisplayCharacteristics
6446

WebForm.TCustomAdapterCommandColumn.GetLayoutAttributes
6446

WebForm.TCustomAdapterCommandColumn.ImplCanAddClass
6447

WebForm.TCustomAdapterCommandColumn.ImplContent 6447

WebForm.TCustomAdapterCommandColumn.NotifyAdapterChange
6448

WebForm.TCustomAdapterCommandColumn.Style 6452

WebForm.TCustomAdapterCommandColumn.StyleRule 6452

WebForm.TCustomAdapterCommandColumn.VAlign 6453

WebForm.TCustomAdapterCommandColumn.VisibleCommands
6453

WebForm.TCustomAdapterCommandGroup 6454

WebForm.TCustomAdapterCommandGroup.Adapter 6459

WebForm.TCustomAdapterCommandGroup.AdapterActionsListHelper
6459

WebForm.TCustomAdapterCommandGroup.AddDefaultCommands
6459

WebForm.TCustomAdapterCommandGroup.Create 6456

WebForm.TCustomAdapterCommandGroup.Custom 6459

WebForm.TCustomAdapterCommandGroup.Destroy 6456

WebForm.TCustomAdapterCommandGroup.GetAdapter 6457

WebForm.TCustomAdapterCommandGroup.GetAdapterFormLayout
6457

WebForm.TCustomAdapterCommandGroup.GetAdapterModeOfAdapter
6457

WebForm.TCustomAdapterCommandGroup.GetAdapterModeOfDisplay
6457

WebForm.TCustomAdapterCommandGroup.ImplCanAddClass
6458

WebForm.TCustomAdapterCommandGroup.ImplContent 6458
WebForm.TCustomAdapterCommandGroup.Style 6460
WebForm.TCustomAdapterCommandGroup.StyleRule 6460
WebForm.TCustomAdapterCommandGroup.VisibleCommands 6460
WebForm.TCustomAdapterDisplayColumn 6461
WebForm.TCustomAdapterDisplayColumn.CanAddClass 6463
WebForm.TCustomAdapterDisplayColumn.ControlContent 6463
WebForm.TCustomAdapterDisplayColumn.Create 6463
WebForm.TCustomAdapterDisplayColumn.DisplayType 6466
WebForm.TCustomAdapterDisplayColumn.FormatImage 6464
WebForm.TCustomAdapterDisplayColumn.FormatListDisplay 6464
WebForm.TCustomAdapterDisplayColumn.FormatTextDisplay 6464
WebForm.TCustomAdapterDisplayColumn.ImageHeight 6466
WebForm.TCustomAdapterDisplayColumn.ImageWidth 6467
WebForm.TCustomAdapterDisplayColumn.ImplCanAddClass 6465
WebForm.TCustomAdapterDisplayColumn.ImplFormatColumnHeading 6465
WebForm.TCustomAdapterDisplayColumn.ImplGetControlRequirements 6465
WebForm.TCustomAdapterDisplayColumn.ImplRestoreDefaults 6465
WebForm.TCustomAdapterDisplayColumn.IsColumn 6466
WebForm.TCustomAdapterDisplayField 6467
WebForm.TCustomAdapterDisplayField.AddAttributes 6470
WebForm.TCustomAdapterDisplayField.ControlContent 6470
WebForm.TCustomAdapterDisplayField.Create 6469
WebForm.TCustomAdapterDisplayField.DisplayColumns 6472
WebForm.TCustomAdapterDisplayField.DisplayRows 6472
WebForm.TCustomAdapterDisplayField.DisplayType 6473
WebForm.TCustomAdapterDisplayField.DisplayWidth 6473
WebForm.TCustomAdapterDisplayField.GetLayoutAttributes 6470
WebForm.TCustomAdapterDisplayField.ImageHeight 6473
WebForm.TCustomAdapterDisplayField.ImageWidth 6474
WebForm.TCustomAdapterDisplayField.ImplContent 6471
WebForm.TCustomAdapterDisplayField.ImplGetControlRequirements 6471
WebForm.TCustomAdapterDisplayField.ImplRestoreDefaults 6471
WebForm.TCustomAdapterDisplayField.InputType 6474
WebForm.TCustomAdapterDisplayField.MaxLength 6474
WebForm.TCustomAdapterDisplayField.SelectRows 6475
WebForm.TCustomAdapterDisplayField.TextAreaWrap 6475
WebForm.TCustomAdapterDisplayField.ViewMode 6475
WebForm.TCustomAdapterEditColumn 6475
WebForm.TCustomAdapterEditColumn.ImplContent 6478
WebForm.TCustomAdapterErrorList 6478
WebForm.TCustomAdapterErrorList.Adapter 6482
WebForm.TCustomAdapterErrorList.Content 6480
WebForm.TCustomAdapterErrorList.Create 6480
WebForm.TCustomAdapterErrorList.Destroy 6480
WebForm.TCustomAdapterErrorList.GetDesigntimeWarnings 6481
WebForm.TCustomAdapterErrorList.ImplContent 6481
WebForm.TCustomAdapterErrorList.Notification 6482
WebForm.TCustomAdapterFieldGroup 6483
WebForm.TCustomAdapterFieldGroup.Create 6485
WebForm.TCustomAdapterFieldGroup.Custom 6489
WebForm.TCustomAdapterFieldGroup.Destroy 6485
WebForm.TCustomAdapterFieldGroup.GetAdapterDisplayCharacteristics 6486
WebForm.TCustomAdapterFieldGroup.GetAdapterFormLayout 6486
WebForm.TCustomAdapterFieldGroup.GetAddAllFieldsItem 6486
WebForm.TCustomAdapterFieldGroup.GetAddFieldsDlgCaption 6487
WebForm.TCustomAdapterFieldGroup.GetAddFieldsItem 6487
WebForm.TCustomAdapterFieldGroup.GetLayoutAttributes 6487
WebForm.TCustomAdapterFieldGroup.GetNewFieldPrefix 6487
WebForm.TCustomAdapterFieldGroup.ImplCanAddClass 6488
WebForm.TCustomAdapterFieldGroup.ImplContent 6488
WebForm.TCustomAdapterFieldGroup.Style 6489
WebForm.TCustomAdapterFieldGroup.StyleRule 6489
WebForm.TCustomAdapterForm 6490
WebForm.TCustomAdapterForm.Create 6491

WebForm.TCustomAdapterForm.Custom 6495
WebForm.TCustomAdapterForm.Destroy 6491
WebForm.TCustomAdapterForm.FormMethod 6495
WebForm.TCustomAdapterForm.GetAdapterFormLayout 6492
WebForm.TCustomAdapterForm.GetFormRequirements 6492
WebForm.TCustomAdapterForm.GetHTMLFormName 6492
WebForm.TCustomAdapterForm.GetHTMLFormVarName 6493
WebForm.TCustomAdapterForm.GetLayoutAttributes 6493
WebForm.TCustomAdapterForm.ImplCanAddClass 6493
WebForm.TCustomAdapterForm.ImplContent 6494
WebForm.TCustomAdapterForm.Style 6495
WebForm.TCustomAdapterForm.StyleRule 6496
WebForm.TCustomAdapterGrid 6496
WebForm.TCustomAdapterGrid.Create 6499
WebForm.TCustomAdapterGrid.Destroy 6499
WebForm.TCustomAdapterGrid.GetAdapterDisplayCharacteristics 6500
WebForm.TCustomAdapterGrid.GetAdapterFormLayout 6500
WebForm.TCustomAdapterGrid.GetAddAllFieldsItem 6500
WebForm.TCustomAdapterGrid.GetAddFieldsDlgCaption 6501
WebForm.TCustomAdapterGrid.GetAddFieldsItem 6501
WebForm.TCustomAdapterGrid.GetNewFieldPrefix 6501
WebForm.TCustomAdapterGrid.HeadingAttributes 6502
WebForm.TCustomAdapterGrid.ImplCanAddClass 6501
WebForm.TCustomAdapterGrid.ImplContent 6502
WebForm.TCustomAdapterGrid.RowAttributes 6503
WebForm.TCustomAdapterGrid.TableAttributes 6503
WebForm.TDesigntimeWarnings 6504
WebForm.TDesigntimeWarnings.AddObject 6505
WebForm.TDesigntimeWarnings.AddString 6505
WebForm.TDesigntimeWarnings.Create 6504
WebForm.TDesigntimeWarnings.Destroy 6504
WebForm.TDesigntimeWarnings.Strings 6506
WebForm.TDisplayFieldHideOption 6524
WebForm.TDisplayFieldHideOption enumeration 6524
WebForm.TDisplayFieldHideOptions 6528
WebForm.TDisplayFieldHideOptions type 6528
WebForm.TDisplayFieldHTMLElementType 6525
WebForm.TDisplayFieldHTMLElementType enumeration 6525
WebForm.TDisplayFieldViewMode 6525
WebForm.TDisplayFieldViewMode enumeration 6525
WebForm.TGeneratedFunctions 6506
WebForm.TGeneratedFunctions.Add 6507
WebForm.TGeneratedFunctions.Create 6507
WebForm.TGeneratedFunctions.Destroy 6507
WebForm.TGeneratedFunctions.Exists 6508
WebForm.TGeneratedFunctions.ItemCount 6508
WebForm.TGeneratedFunctions.Names 6508
WebForm.TGeneratedFunctions.Script 6508
WebForm.TInputFieldHTMLElementType 6526
WebForm.TInputFieldHTMLElementType enumeration 6526
WebForm.TLayoutState 6527
WebForm.TLayoutState enumeration 6527
WebImageHREF
IWebImageHREF interface 1098
TCustomAdapterImageField class 5634
weBinding enumeration member 7186
WebModu 6533
WebModu namespace 6533
Classes 6535
Functions 6612
Types 6614
WebModu.EWebModuleException 6535
WebModu.FindNamedModuleVariable 6612
WebModu.FindNamedModuleVariable function 6612
WebModu.GetModuleVariable 6612
WebModu.GetModuleVariable function 6612
WebModu.GetModuleVariableCount 6612
WebModu.GetModuleVariableCount function 6612
WebModu.TAdapterActionName 6536
WebModu.TAdapterActionName.Action 6538
WebModu.TAdapterActionName.ActionName 6539
WebModu.TAdapterActionName.Adapter 6539
WebModu.TAdapterActionName.AssignTo 6537
WebModu.TAdapterActionName.Create 6537
WebModu.TCustomWebAppDataModule 6539
WebModu.TCustomWebAppDataModule.AppServices 6544
WebModu.TCustomWebAppDataModule.Create 6541

WebModu.TCustomWebAppDataModule.CreateNew 6541
WebModu.TCustomWebAppDataModule.Destroy 6542
WebModu.TCustomWebAppDataModule.GetWebAppServices 6542
WebModu.TCustomWebAppDataModule.Notification 6543
WebModu.TCustomWebAppDataModule.QueryInterface 6543
WebModu.TCustomWebAppPageModule 6544
WebModu.TCustomWebAppPageModule.AppServices 6549
WebModu.TCustomWebAppPageModule.Create 6546
WebModu.TCustomWebAppPageModule.CreateNew 6547
WebModu.TCustomWebAppPageModule.Destroy 6547
WebModu.TCustomWebAppPageModule.GetWebAppServices 6548
WebModu.TCustomWebAppPageModule.Notification 6548
WebModu.TCustomWebAppPageModule.QueryInterface 6548
WebModu.TCustomWebAppPageModule.SetAppServices 6549
WebModu.TCustomWebDataModule 6550
WebModu.TCustomWebDataModule.Create 6551
WebModu.TCustomWebDataModule.CreateNew 6552
WebModu.TCustomWebDataModule.Destroy 6552
WebModu.TCustomWebDataModule.GetSiteModuleHelper 6553
WebModu.TCustomWebDataModule.GetWebSnapObjClass 6553
WebModu.TCustomWebDataModule.Notification 6553
WebModu.TCustomWebDataModule.OnActivate 6556
WebModu.TCustomWebDataModule.OnDeactivate 6556
WebModu.TCustomWebDataModule.Request 6554
WebModu.TCustomWebDataModule.Response 6554
WebModu.TCustomWebDataModule.Session 6555
WebModu.TCustomWebDataModule.SiteModuleHelper 6555
WebModu.TCustomWebPageModule 6556
WebModu.TCustomWebPageModule.Create 6559
WebModu.TCustomWebPageModule.CreateNew 6559
WebModu.TCustomWebPageModule.DefaultAction 6561
WebModu.TCustomWebPageModule.Destroy 6560
WebModu.TCustomWebPageModule.GetSiteModuleHelper 6560
WebModu.TCustomWebPageModule.GetSitePageModuleHelper
6561
WebModu.TCustomWebPageModule.Notification 6561
WebModu.TCustomWebPageModule.OnAfterDispatchPage 6563
WebModu.TCustomWebPageModule.OnAfterRedirectToPage 6563
WebModu.TCustomWebPageModule.OnBeforeDispatchPage 6564
WebModu.TCustomWebPageModule.OnBeforeRedirectToPage 6564
WebModu.TCustomWebPageModule.PageProducer 6562
WebModu.TCustomWebPageModule.SitePageModuleHelper 6562
WebModu.TGetParamsEvent 6614
WebModu.TGetParamsEvent type 6614
WebModu.TSiteModuleHelper 6565
WebModu.TSiteModuleHelper.Create 6567
WebModu.TSiteModuleHelper.Destroy 6567
WebModu.TSiteModuleHelper.EndIterator 6568
WebModu.TSiteModuleHelper.FindNamedVariable 6568
WebModu.TSiteModuleHelper.FindVariable 6569
WebModu.TSiteModuleHelper.FindVariableContainer 6569
WebModu.TSiteModuleHelper.GetCurrentObject 6569
WebModu.TSiteModuleHelper.GetVariable 6569
WebModu.TSiteModuleHelper.GetVariableCount 6569
WebModu.TSiteModuleHelper.Module 6571
WebModu.TSiteModuleHelper.NextIteration 6570
WebModu.TSiteModuleHelper.Notification 6570
WebModu.TSiteModuleHelper.NotifyActivate 6570
WebModu.TSiteModuleHelper.NotifyDeactivate 6570
WebModu.TSiteModuleHelper.OnActivate 6573
WebModu.TSiteModuleHelper.OnDeactivate 6573
WebModu.TSiteModuleHelper.Request 6571
WebModu.TSiteModuleHelper.Response 6572
WebModu.TSiteModuleHelper.Session 6572
WebModu.TSiteModuleHelper.StartIterator 6571
WebModu.TSiteModuleHelper.WebModuleContext 6572
WebModu.TSitePageModuleHelper 6574
WebModu.TSitePageModuleHelper.Create 6576
WebModu.TSitePageModuleHelper.DefaultAction 6583
WebModu.TSitePageModuleHelper.Destroy 6577
WebModu.TSitePageModuleHelper.DispatchPage 6577
WebModu.TSitePageModuleHelper.GetDefaultAction 6578

WebModu.TSitePageModuleHelper.GetPageProducer 6578
WebModu.TSitePageModuleHelper.GetProducerComponent 6578
WebModu.TSitePageModuleHelper.GetRedirectParams 6578
WebModu.TSitePageModuleHelper.GetTemplateFileType 6579
WebModu.TSitePageModuleHelper.GetTemplateStream 6579
WebModu.TSitePageModuleHelper.GetXMLData 6579
WebModu.TSitePageModuleHelper.GetXSLData 6579
WebModu.TSitePageModuleHelper.HasHTMLBrowserView 6579
WebModu.TSitePageModuleHelper.HasScriptView 6580
WebModu.TSitePageModuleHelper.HasXMLBrowserView 6580
WebModu.TSitePageModuleHelper.HasXSLBrowserView 6580
WebModu.TSitePageModuleHelper.HTMLFromTemplate 6580
WebModu.TSitePageModuleHelper.ImplDispatchPage 6581
WebModu.TSitePageModuleHelper.ImplIncludePage 6581
WebModu.TSitePageModuleHelper.ImplRedirectToPage 6581
WebModu.TSitePageModuleHelper.ImplSetWebContentOptions 6581
WebModu.TSitePageModuleHelper.IncludePage 6582
WebModu.TSitePageModuleHelper.OnAfterDispatchPage 6583
WebModu.TSitePageModuleHelper.OnAfterRedirectToPage 6584
WebModu.TSitePageModuleHelper.OnBeforeDispatchPage 6584
WebModu.TSitePageModuleHelper.OnBeforeRedirectToPage 6584
WebModu.TSitePageModuleHelper.RedirectToPage 6582
WebModu.TSitePageModuleHelper.SetDefaultPageFileName 6582
WebModu.TSitePageModuleHelper.SetWebContentOptions 6582
WebModu.TWebAppDataModule 6585
WebModu.TWebAppDataModule.AppServices 6587
WebModu.TWebAppDataModule.OnActivate 6587
WebModu.TWebAppDataModule.OnDeactivate 6588
WebModu.TWebAppPageModule 6588
WebModu.TWebAppPageModule.AppServices 6591
WebModu.TWebAppPageModule.DefaultAction 6591
WebModu.TWebAppPageModule.OnActivate 6592
WebModu.TWebAppPageModule.OnAfterDispatchPage 6593
WebModu.TWebAppPageModule.OnBeforeDispatchPage 6593
WebModu.TWebAppPageModule.OnDeactivate 6594
WebModu.TWebAppPageModule.PageProducer 6592
WebModu.TWebAppPageModuleHelper 6594
WebModu.TWebAppPageModuleHelper.GetPageProducer 6597
WebModu.TWebDataModule 6597
WebModu.TWebDataModule.OnActivate 6598
WebModu.TWebDataModule.OnDeactivate 6599
WebModu.TWebNotifyContainer 6599
WebModu.TWebNotifyContainer.Add 6602
WebModu.TWebNotifyContainer.Create 6601
WebModu.TWebNotifyContainer.Destroy 6601
WebModu.TWebNotifyContainer.FList 6602
WebModu.TWebNotifyContainer.GetCount 6602
WebModu.TWebNotifyContainer.NotifyWebActivate 6603
WebModu.TWebNotifyContainer.NotifyWebDeactivate 6603
WebModu.TWebPageModule 6603
WebModu.TWebPageModule.DefaultAction 6606
WebModu.TWebPageModule.OnActivate 6607
WebModu.TWebPageModule.OnAfterDispatchPage 6607
WebModu.TWebPageModule.OnBeforeDispatchPage 6608
WebModu.TWebPageModule.OnDeactivate 6608
WebModu.TWebPageModule.PageProducer 6606
WebModu.TWebPageModuleHelper 6609
WebModu.TWebPageModuleHelper.GetPageProducer 6611
WebModuleClass
TWebRequestHandler class 6640
WebModuleContext
TSiteModuleHelper class 6572
WebModuleFactory
TWebRequestHandler class 6640
WebModuleFactoryCount
TWebRequestHandler class 6641
WebModules
TAbstractWebContext class 5975
TMultiModuleWebAppServices class 6234
WebNode 6615
TLinkedRIO class 1320
WebNode namespace 6615

Interfaces 6616
WebNode.IWebNode 6616
WebNode.IWebNode.BeforeExecute 6617
WebNode.IWebNode.Execute 6618
WebNode.IWebNode.GetMimeBoundary 6619
WebNode.IWebNode.MimeBoundary 6619
WebNode.IWebNode.SetMimeBoundary 6619
WebPageInfo
TCurrentWebPageInfo class 6662
WebPageItems
TAdapterPageProducer class 1141
WebParent
TWebContainedComponent class 6044
TWebContainedComponentHelper class 6051
TWebContainedContainerComponent class 6062
WebReq 6621
WebReq namespace 6621
Classes 6622
Functions 6642
Structs, Records, Enums 6643
Variables 6644
WebReq.TRequestNotification 6643
WebReq.TRequestNotification enumeration 6643
WebReq.TWebModuleFactoryList 6622
WebReq.TWebModuleFactoryList.AddFactory 6624
WebReq.TWebModuleFactoryList.AppModuleFactory 6625
WebReq.TWebModuleFactoryList.Create 6623
WebReq.TWebModuleFactoryList.Destroy 6624
WebReq.TWebModuleFactoryList.GetItem 6625
WebReq.TWebModuleFactoryList.ItemCount 6625
WebReq.TWebModuleFactoryList.Items 6626
WebReq.TWebModuleList 6626
WebReq.TWebModuleList.AddModule 6628
WebReq.TWebModuleList.AddModuleClass 6629
WebReq.TWebModuleList.AddModuleName 6629
WebReq.TWebModuleList.AutoCreateModules 6630
WebReq.TWebModuleList.AutoDestroyModules 6630
WebReq.TWebModuleList.Create 6627
WebReq.TWebModuleList.Destroy 6628
WebReq.TWebModuleList.Factories 6633
WebReq.TWebModuleList.FindModuleClass 6631
WebReq.TWebModuleList.FindModuleName 6631
WebReq.TWebModuleList.GetFactory 6632
WebReq.TWebModuleList.GetFactoryCount 6632
WebReq.TWebModuleList.GetItem 6632
WebReq.TWebModuleList.GetItemCount 6632
WebReq.TWebModuleList.GetOnModuleAdded 6633
WebReq.TWebModuleList.SetOnModuleAdded 6633
WebReq.TWebRequestHandler 6633
WebReq.TWebRequestHandler.ActivateWebModules 6636
WebReq.TWebRequestHandler.ActiveCount 6638
WebReq.TWebRequestHandler.AddWebModuleFactory 6636
WebReq.TWebRequestHandler.CacheConnections 6638
WebReq.TWebRequestHandler.Create 6635
WebReq.TWebRequestHandler.DeactivateWebModules 6636
WebReq.TWebRequestHandler.Destroy 6635
WebReq.TWebRequestHandler.HandleException 6637
WebReq.TWebRequestHandler.HandleRequest 6637
WebReq.TWebRequestHandler.InactiveCount 6639
WebReq.TWebRequestHandler.MaxConnections 6640
WebReq.TWebRequestHandler.WebModuleClass 6640
WebReq.TWebRequestHandler.WebModuleFactory 6640
WebReq.TWebRequestHandler.WebModuleFactoryCount 6641
WebReq.WebRequestHandler 6642
WebReq.WebRequestHandler function 6642
WebReq.WebRequestHandlerProc 6644
WebReq.WebRequestHandlerProc variable 6644
WebScript 6645
WebScript namespace 6645
Classes 6647
Constants 6723
Structs, Records, Enums 6720
Types 6722
WebScript.CreateComponentStubFactory 6722
WebScript.CreateComponentStubFactory type 6722
WebScript.ERecursiveIncludeFile 6647
WebScript.EScriptEvalError 6648
WebScript.GlobalScriptVariableNames 6723

WebScript.GlobalScriptVariableNames constant 6723
WebScript.PLineNumbers 6722
WebScript.PLineNumbers type 6722
WebScript.TAbstractScriptEngine 6648
WebScript.TAbstractScriptEngine.AddObjectFactory 6650
WebScript.TAbstractScriptEngine.Create 6649
WebScript.TAbstractScriptEngine.CreateComponentStub 6651
WebScript.TAbstractScriptEngine.Destroy 6649
WebScript.TAbstractScriptEngine.ListLanguageNames 6651
WebScript.TAbstractScriptEngine.Name 6651
WebScript.TAbstractScriptEngine.ScriptObjectFactories 6652
WebScript.TAbstractScriptEngine.ScriptProducerClass 6652
WebScript.TAbstractScriptEngine.SupportsLanguage 6651
WebScript.TAbstractScriptObjectFactories 6652
WebScript.TAbstractScriptObjectFactories.Count 6655
WebScript.TAbstractScriptObjectFactories.Factories 6655
WebScript.TAbstractScriptObjectFactories.GetCount 6654
WebScript.TAbstractScriptObjectFactories.GetFactory 6654
WebScript.TAbstractScriptObjectFactories.GetObjectList 6654
WebScript.TAbstractScriptObjectFactories.ObjectList 6655
WebScript.TAbstractScriptObjectFactory 6655
WebScript.TAbstractScriptObjectFactory.AddGlobalObjects 6656
WebScript.TAbstractScriptObjectFactory.CreateComponentObject 6656
WebScript.TAbstractScriptObjectFactory.CreateGlobalObject 6657
WebScript.TAbstractScriptObjectFactory.CreateProducerObject 6657
WebScript.TAbstractScriptObjectFactory.GetScriptEngine 6657
WebScript.TCurrentWebPageInfo 6657
WebScript.TCurrentWebPageInfo.Create 6659
WebScript.TCurrentWebPageInfo.GetHasViewAccess 6660
WebScript.TCurrentWebPageInfo.GetLoginRequired 6660
WebScript.TCurrentWebPageInfo.GetPageDescription 6660
WebScript.TCurrentWebPageInfo.GetPageHREF 6661
WebScript.TCurrentWebPageInfo.GetPageIsPublished 6661
WebScript.TCurrentWebPageInfo.GetPageName 6661
WebScript.TCurrentWebPageInfo.GetPageTitle 6662
WebScript.TCurrentWebPageInfo.WebPageInfo 6662
WebScript.TGlobalScriptVariables 6720
WebScript.TGlobalScriptVariables enumeration 6720
WebScript.THTMLItem 6662
WebScript.THTMLItem.AssignString 6664
WebScript.THTMLItem.Create 6664
WebScript.THTMLItem.HTML 6664
WebScript.THTMLItem.Ident 6665
WebScript.THTMLItem.ItemLength 6665
WebScript.THTMLItem.TagToken 6665
WebScript.THTMLItems 6665
WebScript.THTMLItems.Create 6666
WebScript.THTMLItems.Destroy 6666
WebScript.THTMLItems.GetHTMLItem 6667
WebScript.THTMLItems.Items 6667
WebScript.THTMLItems.MakeIdentifier 6667
WebScript.TIncludeType 6721
WebScript.TIncludeType enumeration 6721
WebScript.TLineNumberMap 6668
WebScript.TLineNumberMap.Count 6670
WebScript.TLineNumberMap.DestLineToSourceLine 6670
WebScript.TLineNumberMap.Items 6671
WebScript.TLineNumberMap.New 6670
WebScript.TLineNumbers 6721
WebScript.TLineNumbers record 6721
WebScript.TScriptCodeItem 6671
WebScript.TScriptCodeItem.AssignString 6672
WebScript.TScriptCodeItem.GetScript 6673
WebScript.TScriptEnginesList 6673
WebScript.TScriptEnginesList.AddEngine 6675
WebScript.TScriptEnginesList.AddObjectFactory 6675
WebScript.TScriptEnginesList.Count 6676
WebScript.TScriptEnginesList.Create 6674
WebScript.TScriptEnginesList.Destroy 6674
WebScript.TScriptEnginesList.Find 6675
WebScript.TScriptEnginesList.FindScriptProducerClass 6676
WebScript.TScriptEnginesList.Items 6677
WebScript.TScriptEnginesList.ListLanguageNames 6676
WebScript.TScriptEnginesList.RemoveEngine 6676
WebScript.TScriptError 6677

WebScript.TScriptError.Create 6678
WebScript.TScriptError.FCharPos 6678
WebScript.TScriptError.FDescription 6678
WebScript.TScriptError.FFileName 6679
WebScript.TScriptError.FLine 6679
WebScript.TScriptError.FSourceLine 6679
WebScript.TScriptError.GetCharPos 6679
WebScript.TScriptError.GetDescription 6680
WebScript.TScriptError.GetFileName 6680
WebScript.TScriptError.GetLine 6680
WebScript.TScriptError.GetSourceLine 6680
WebScript.TScriptError.SetDescription 6681
WebScript.TScriptErrors 6681
WebScript.TScriptErrors.Add 6683
WebScript.TScriptErrors.Create 6681
WebScript.TScriptErrors.Destroy 6682
WebScript.TScriptErrors.GetError 6683
WebScript.TScriptErrors.GetErrorCount 6683
WebScript.TScriptFile 6683
WebScript.TScriptFile.ApplicationName 6686
WebScript.TScriptFile.Create 6684
WebScript.TScriptFile.Destroy 6684
WebScript.TScriptFile.FLineNumberMap 6685
WebScript.TScriptFile.LongName 6686
WebScript.TScriptFile.ParseStream 6685
WebScript.TScriptFile.ParseString 6685
WebScript.TScriptFile.Script 6686
WebScript.TScriptFile.ShortName 6687
WebScript.TScriptFile.Source 6687
WebScript.TScriptGlobalObjects 6687
WebScript.TScriptGlobalObjects.AddNamedIntf 6690
WebScript.TScriptGlobalObjects.Create 6688
WebScript.TScriptGlobalObjects.Destroy 6689
WebScript.TScriptGlobalObjects.FindNamedItemIntf 6690
WebScript.TScriptGlobalObjects.GetNamedItemCount 6690
WebScript.TScriptGlobalObjects.GetNamedItemName 6690
WebScript.TScriptGlobalObjects.GetScriptEngine 6690
WebScript.TScriptGlobalObjects.NamedItemCount 6691
WebScript.TScriptGlobalObjects.NamedItemName 6691
WebScript.TScriptGlobalObjects.Producer 6691
WebScript.TScriptGlobalObjects.ScriptEngine 6692
WebScript.TScriptIncludeFile 6692
WebScript.TScriptIncludeFile.Create 6693
WebScript.TScriptIncludeFile.LoadFile 6693
WebScript.TScriptIncludeFiles 6693
WebScript.TScriptIncludeFiles.Create 6694
WebScript.TScriptIncludeFiles.GetScriptIncludeFile 6694
WebScript.TScriptIncludeFiles.Items 6695
WebScript.TScriptIncludeItem 6696
WebScript.TScriptIncludeItem.Create 6697
WebScript.TScriptIncludeItem.GetScript 6697
WebScript.TScriptItem 6698
WebScript.TScriptItem.Create 6699
WebScript.TScriptItem.GetScript 6699
WebScript.TScriptItem.Script 6700
WebScript.TScriptItems 6700
WebScript.TScriptItems.Create 6701
WebScript.TScriptItems.GetScriptItem 6701
WebScript.TScriptItems.Items 6701
WebScript.TScriptObjectFactories 6702
WebScript.TScriptObjectFactories.Count 6706
WebScript.TScriptObjectFactories.Create 6704
WebScript.TScriptObjectFactories.CreateGlobalObject 6705
WebScript.TScriptObjectFactories.Destroy 6704
WebScript.TScriptObjectFactories.Factories 6706
WebScript.TScriptObjectFactories.GetCount 6705
WebScript.TScriptObjectFactories.GetFactory 6705
WebScript.TScriptObjectFactories.GetObjectList 6706
WebScript.TScriptProducer 6707
WebScript.TScriptProducer.Component 6714
WebScript.TScriptProducer.Content 6715
WebScript.TScriptProducer.Create 6708
WebScript.TScriptProducer.Designing 6715
WebScript.TScriptProducer.Destroy 6708
WebScript.TScriptProducer.Errors 6715
WebScript.TScriptProducer.Evaluate 6710
WebScript.TScriptProducer.EvaluateExpression 6710
WebScript.TScriptProducer.FErrors 6709

WebScript.TScriptProducer.FScriptEngine 6709
WebScript.TScriptProducer.FScriptFile 6709
WebScript.TScriptProducer.GetContent 6710
WebScript.TScriptProducer.GetErrors 6711
WebScript.TScriptProducer.GetHTMLBlock 6711
WebScript.TScriptProducer.GetHTMLBlockCount 6711
WebScript.TScriptProducer.GetWebModuleContext 6711
WebScript.TScriptProducer.HandleParserError 6712
WebScript.TScriptProducer.HandleScriptError 6712
WebScript.TScriptProducer.ImplHandleScriptError 6712
WebScript.TScriptProducer.ParseStream 6712
WebScript.TScriptProducer.ParseString 6712
WebScript.TScriptProducer.ReplaceTags 6713
WebScript.TScriptProducer.Script 6715
WebScript.TScriptProducer.SetContent 6713
WebScript.TScriptProducer.Write 6713, 6714
WebScript.TScriptProducer.WriteItem 6714
WebScript.TVariableLookup 6716
WebScript.TVariableLookup.Create 6717
WebScript.TVariableLookup.FindGlobalVariable 6717
WebScript.TVariableLookup.FindGlobalVariableContainer 6718
WebScript.TVariableLookup.Ident 6718
WebScript.TVariableLookup.LookupObject 6718
WebScript.TVariableLookup.Obj 6718
WebServExp 6725
WebServExp namespace 6725
Classes 6727
Functions 6738
Interfaces 6735
Structs, Records, Enums 6740
Types 6742
WebServExp.ArgumentType 6740
WebServExp.ArgumentType enumeration 6740
WebServExp.GetAliasBaseTypeInfo 6738
WebServExp.GetAliasBaseTypeInfo function 6738
WebServExp.GetBindingType 6738
WebServExp.GetBindingType function 6738
WebServExp.IsBaseClassTypeInfo 6739
WebServExp.IsBaseClassTypeInfo function 6739
WebServExp.IWebServExp 6735
WebServExp.IWebServExp.FindOrAddSchema 6736
WebServExp.IWebServExp.GenerateXMLSchema 6736
WebServExp.IWebServExpAccess 6737
WebServExp.IWebServExpAccess.GetWebServExp 6737
WebServExp.MessageType 6740
WebServExp.MessageType enumeration 6740
WebServExp.TAfterPublishingWSDLEvent 6742
WebServExp.TAfterPublishingWSDLEvent type 6742
WebServExp.TBeforePublishingTypesEvent 6742
WebServExp.TBeforePublishingTypesEvent type 6742
WebServExp.TPublishingTypeEvent 6743
WebServExp.TPublishingTypeEvent type 6743
WebServExp.TSchemaType 6741
WebServExp.TSchemaType record 6741
WebServExp.TSchemaTypeArray 6744
WebServExp.TSchemaTypeArray type 6744
WebServExp.TWebServExp 6727
WebServExp.TWebServExp.ArrayAsComplexContent 6731
WebServExp.TWebServExp.BindingType 6732
WebServExp.TWebServExp.Create 6729
WebServExp.TWebServExp.Destroy 6729
WebServExp.TWebServExp.GenerateXMLSchema 6730
WebServExp.TWebServExp.GetWebServExp 6731
WebServExp.TWebServExp.GetWSDLForInterface 6731
WebServExp.TWebServExp.ImportLocations 6732
WebServExp.TWebServExp.ImportNames 6732
WebServExp.TWebServExp.OnAfterPublishingWSDL 6733
WebServExp.TWebServExp.OnBeforePublishingTypes 6733
WebServExp.TWebServExp.OnPublishingType 6733
WebServExp.TWebServExp.ServiceAddress 6732
WebServExp.TWebServExp.TargetNameSpace 6732
WebServExp.TWebServExp.WSDLElements 6733
WebSess 6745
WebSess namespace 6745
Classes 6746
Structs, Records, Enums 6773
Types 6774
WebSess.TCustomSessionsService 6746

WebSess.TCustomSessionsService.CheckStatusChange 6748
WebSess.TCustomSessionsService.Clear 6749
WebSess.TCustomSessionsService.Connect 6749
WebSess.TCustomSessionsService.ConnectActive 6750
WebSess.TCustomSessionsService.Create 6747
WebSess.TCustomSessionsService.DefaultTimeout 6754
WebSess.TCustomSessionsService.DoEndSession 6750
WebSess.TCustomSessionsService.DoStartSession 6750
WebSess.TCustomSessionsService.GetSessionIDs 6751
WebSess.TCustomSessionsService.LoadFromFile 6751
WebSess.TCustomSessionsService.LoadFromStream 6751
WebSess.TCustomSessionsService.MaxSessions 6754
WebSess.TCustomSessionsService.NewSession 6752
WebSess.TCustomSessionsService.NotifyStatusChange 6752
WebSess.TCustomSessionsService.OnEndSession 6755
WebSess.TCustomSessionsService.OnStartSession 6755
WebSess.TCustomSessionsService.SaveToFile 6753
WebSess.TCustomSessionsService.SaveToStream 6753
WebSess.TEndSessionEvent 6774
WebSess.TEndSessionEvent type 6774
WebSess.TEndSessionReason 6773
WebSess.TEndSessionReason enumeration 6773
WebSess.TSessionsService 6756
WebSess.TSessionsService.DefaultTimeout 6758
WebSess.TSessionsService.MaxSessions 6758
WebSess.TSessionsService.OnEndSession 6759
WebSess.TSessionsService.OnStartSession 6759
WebSess.TStartSessionEvent 6774
WebSess.TStartSessionEvent type 6774
WebSess.TWebSessionIDsImpl 6759
WebSess.TWebSessionIDsImpl.Create 6761
WebSess.TWebSessionIDsImpl.Destroy 6761
WebSess.TWebSessionIDsImpl.GetID 6762
WebSess.TWebSessionIDsImpl.GetIDCount 6763
WebSess.TWebSessionImpl 6763
WebSess.TWebSessionImpl.Create 6765
WebSess.TWebSessionImpl.Destroy 6765
WebSess.TWebSessionImpl.ExpirationTime 6770
WebSess.TWebSessionImpl.Expired 6770
WebSess.TWebSessionImpl.GetExpirationTime 6766
WebSess.TWebSessionImpl.GetExpired 6767
WebSess.TWebSessionImpl.GetID 6767
WebSess.TWebSessionImpl.GetIsActive 6767
WebSess.TWebSessionImpl.GetItemCount 6767
WebSess.TWebSessionImpl.GetItems 6767
WebSess.TWebSessionImpl.GetStartTime 6768
WebSess.TWebSessionImpl.GetTerminated 6768
WebSess.TWebSessionImpl.GetTimeoutMinutes 6768
WebSess.TWebSessionImpl.GetTouchTime 6768
WebSess.TWebSessionImpl.GetValue 6769
WebSess.TWebSessionImpl.ID 6770
WebSess.TWebSessionImpl.SetTimeoutMinutes 6769
WebSess.TWebSessionImpl.SetValue 6769
WebSess.TWebSessionImpl.StartTime 6771
WebSess.TWebSessionImpl.Terminate 6769
WebSess.TWebSessionImpl.Terminated 6771
WebSess.TWebSessionImpl.TimeoutMinutes 6771
WebSess.TWebSessionImpl.TouchTime 6771
WebSess.TWebSessionImpl.Values 6771
WebUsers 6777
WebUsers namespace 6777
Classes 6779
Structs, Records, Enums 6801
Types 6802
WebUsers.EUserIDNotFoundException 6779
WebUsers.EValidateUserException 6779
WebUsers.EWebUsersListException 6780
WebUsers.TCheckAccessRightsEvent 6802
WebUsers.TCheckAccessRightsEvent type 6802
WebUsers.TCheckAccessRightsHandledEvent 6802
WebUsers.TCheckAccessRightsHandledEvent type 6802
WebUsers.TCustomWebUserList 6781
WebUsers.TCustomWebUserList.CheckAccessRights 6783
WebUsers.TCustomWebUserList.Create 6782
WebUsers.TCustomWebUserList.Destroy 6782
WebUsers.TCustomWebUserList.LoadFromFile 6783
WebUsers.TCustomWebUserList.OnAfterCheckAccessRights 6785
WebUsers.TCustomWebUserList.OnAfterValidateUser 6786

WebUsers.TCustomWebUserList.OnBeforeCheckAccessRights 6786
WebUsers.TCustomWebUserList.OnBeforeValidateUser 6786
WebUsers.TCustomWebUserList.OnUserIDNotFound 6787
WebUsers.TCustomWebUserList.OnValidateUserError 6787
WebUsers.TCustomWebUserList.SaveToFile 6784
WebUsers.TCustomWebUserList.UserItems 6784
WebUsers.TCustomWebUserList.Users 6785
WebUsers.TCustomWebUserList.ValidateUser 6784
WebUsers.TValidateUserError 6801
WebUsers.TValidateUserError enumeration 6801
WebUsers.TValidateUserErrorEvent 6803
WebUsers.TValidateUserErrorEvent type 6803
WebUsers.TValidateUserEvent 6804
WebUsers.TValidateUserEvent type 6804
WebUsers.TValidateUserHandledEvent 6804
WebUsers.TValidateUserHandledEvent type 6804
WebUsers.TWebUserItem 6787
WebUsers.TWebUserItem.AccessRights 6790
WebUsers.TWebUserItem.AccessRightsStrings 6791
WebUsers.TWebUserItem.Assign 6789
WebUsers.TWebUserItem.CheckRights 6790
WebUsers.TWebUserItem.Create 6788
WebUsers.TWebUserItem.Destroy 6789
WebUsers.TWebUserItem.Password 6791
WebUsers.TWebUserItem.UserID 6791
WebUsers.TWebUserItem.UserName 6791
WebUsers.TWebUserItems 6792
WebUsers.TWebUserItems.Create 6793
WebUsers.TWebUserItems.FindUserID 6793
WebUsers.TWebUserItems.FindUserName 6793
WebUsers.TWebUserItems.GetAttr 6794
WebUsers.TWebUserItems.GetAttrCount 6794
WebUsers.TWebUserItems.GetItemAttr 6795
WebUsers.TWebUserItems.GetOwner 6795
WebUsers.TWebUserItems.Items 6796
WebUsers.TWebUserList 6796
WebUsers.TWebUserList.OnAfterCheckAccessRights 6798
WebUsers.TWebUserList.OnAfterValidateUser 6798
WebUsers.TWebUserList.OnBeforeCheckAccessRights 6799
WebUsers.TWebUserList.OnBeforeValidateUser 6799
WebUsers.TWebUserList.OnUserIDNotFound 6799
WebUsers.TWebUserList.OnValidateUserError 6800
WebUsers.TWebUserList.UserItems 6797
WeImport enumeration member 7186
weMessage enumeration member 7186
weOperation enumeration member 7186
wePart enumeration member 7186
wePortType enumeration member 7186
weService enumeration member 7186
weServiceImpl enumeration member 7186
weServiceIntf enumeration member 7186
weTypes enumeration member 7186
WideChar
    AnsiStringBase class 3203
    AnsiStringT template 3230
WideCharBufSize
    AnsiStringBase class 3203
    AnsiStringT template 3231
WideString
    Variant class 3466
    WideString class 3470
WideString class 3467
    != 3488
    != operator 3487
    [] 3488
    [] operator 3488
    ~WideString 3469
    + 3489
    += 3489
    < 3489
    < operator 3489
    <= 3490
    <= operator 3490
    = 3490
    == 3491
    == operator 3491
    > 3491
    > operator 3491

>= 3492
>= operator 3492
about WideString class 3467
c_bstr 3473
cat_printf 3473
cat_sprintf 3474
cat_vprintf 3474
Copy 3475
CurrToStr 3475
CurrToStrF 3475
data 3476
Delete 3476
FloatToStrF 3477
Format 3477
FormatFloat 3478
Insert 3479
IntToHex 3480
IsDelimiter 3480
IsEmpty 3480
IsPathDelimiter 3481
LastChar 3481
LastDelimiter 3481
Length 3481
LowerCase 3482
Pos 3482
printf 3482
SetLength 3483
sprintf 3483
StringOfChar 3484
SubString 3484
swap 3484
ToDouble 3485
ToInt 3485
ToIntDef 3485
Trim 3486
TrimLeft 3486
TrimRight 3486
UpperCase 3486
vprintf 3487
WideString 3470
WideString constructor 3469
WideString enumerations 3471
WideString Members 3467
WideString methods 3472
WideString operators 3487
WideStrings 6807
WideStrings namespace 6807
Classes 6808
Interfaces 6859
Structs, Records, Enums 6861
Types 6862
WideStrings.IWideStringsAdapter 6859
WideStrings.IWideStringsAdapter.ReferenceStrings 6860
WideStrings.IWideStringsAdapter.ReleaseStrings 6860
WideStrings.PWideStringItem 6862
WideStrings.PWideStringItem type 6862
WideStrings.PWideStringItemList 6862
WideStrings.PWideStringItemList type 6862
WideStrings.TWideStringItem 6861
WideStrings.TWideStringItem record 6861
WideStrings.TWideStringItemList 6863
WideStrings.TWideStringItemList type 6863
WideStrings.TWideStringList 6808
WideStrings.TWideStringList.Add 6812
WideStrings.TWideStringList.AddObject 6813
WideStrings.TWideStringList.CaseSensitive 6822
WideStrings.TWideStringList.Changed 6813
WideStrings.TWideStringList.Changing 6814
WideStrings.TWideStringList.Clear 6814
WideStrings.TWideStringList.CompareStrings 6814
WideStrings.TWideStringList.Create 6811
WideStrings.TWideStringList.CustomSort 6815
WideStrings.TWideStringList.Delete 6815
WideStrings.TWideStringList.Destroy 6811
WideStrings.TWideStringList.Duplicates 6822
WideStrings.TWideStringList.Exchange 6816
WideStrings.TWideStringList.Find 6816
WideStrings.TWideStringList.Get 6817

WideStrings.TWideStringList.GetCapacity 6817
WideStrings.TWideStringList.GetCount 6817
WideStrings.TWideStringList.GetObject 6818
WideStrings.TWideStringList.IndexOf 6818
WideStrings.TWideStringList.Insert 6819
WideStrings.TWideStringList.InsertItem 6819
WideStrings.TWideStringList.InsertObject 6819
WideStrings.TWideStringList.OnChange 6824
WideStrings.TWideStringList.OnChanging 6825
WideStrings.TWideStringList.OwnsObjects 6823
WideStrings.TWideStringList.Put 6820
WideStrings.TWideStringList.PutObject 6820
WideStrings.TWideStringList.SetCapacity 6820
WideStrings.TWideStringList.SetUpdateState 6821
WideStrings.TWideStringList.Sort 6821
WideStrings.TWideStringList.Sorted 6823
WideStrings.TWideStringListSortCompare 6863
WideStrings.TWideStringListSortCompare type 6863
WideStrings.TWideStrings 6825
WideStrings.TWideStrings.Add 6829
WideStrings.TWideStrings.AddObject 6830
WideStrings.TWideStrings.AddStrings 6830
WideStrings.TWideStrings.Append 6831
WideStrings.TWideStrings.Assign 6831
WideStrings.TWideStrings.AssignTo 6832
WideStrings.TWideStrings.BeginUpdate 6832
WideStrings.TWideStrings.Capacity 6847
WideStrings.TWideStrings.Clear 6833
WideStrings.TWideStrings.CommaText 6848
WideStrings.TWideStrings.CompareStrings 6833
WideStrings.TWideStrings.Count 6849
WideStrings.TWideStrings.DefineProperties 6834
WideStrings.TWideStrings.Delete 6834
WideStrings.TWideStrings.DelimitedText 6849
WideStrings.TWideStrings.Delimiter 6850
WideStrings.TWideStrings.Destroy 6828
WideStrings.TWideStrings.EndUpdate 6835
WideStrings.TWideStrings.Equals 6835
WideStrings.TWideStrings.Error 6836
WideStrings.TWideStrings.Exchange 6836
WideStrings.TWideStrings.ExtractName 6837
WideStrings.TWideStrings.Get 6837
WideStrings.TWideStrings.GetCapacity 6837
WideStrings.TWideStrings.GetCount 6838
WideStrings.TWideStrings.GetEnumerator 6838
WideStrings.TWideStrings.GetObject 6838
WideStrings.TWideStrings.GetText 6839
WideStrings.TWideStrings.GetTextStr 6839
WideStrings.TWideStrings.IndexOf 6840
WideStrings.TWideStrings.IndexOfName 6840
WideStrings.TWideStrings.IndexOfObject 6841
WideStrings.TWideStrings.Insert 6841
WideStrings.TWideStrings.InsertObject 6842
WideStrings.TWideStrings.LineBreak 6850
WideStrings.TWideStrings.LoadFromFile 6842
WideStrings.TWideStrings.LoadFromStream 6842
WideStrings.TWideStrings.Move 6843
WideStrings.TWideStrings.Names 6850
WideStrings.TWideStrings.NameValueSeparator 6851
WideStrings.TWideStrings.Objects 6851
WideStrings.TWideStrings.Put 6843
WideStrings.TWideStrings.PutObject 6844
WideStrings.TWideStrings.QuoteChar 6852
WideStrings.TWideStrings.SaveToFile 6844
WideStrings.TWideStrings.SaveToStream 6845
WideStrings.TWideStrings.SetCapacity 6845
WideStrings.TWideStrings.SetText 6845
WideStrings.TWideStrings.SetTextStr 6846
WideStrings.TWideStrings.SetUpdateState 6846
WideStrings.TWideStrings.StrictDelimiter 6852
WideStrings.TWideStrings.Strings 6853
WideStrings.TWideStrings.StringsAdapter 6854
WideStrings.TWideStrings.Text 6854
WideStrings.TWideStrings.UpdateCount 6855
WideStrings.TWideStrings.ValueFromIndex 6855
WideStrings.TWideStrings.Values 6856
WideStrings.TWideStringsEnumerator 6856
WideStrings.TWideStringsEnumerator.Create 6857

WideStrings.TWideStringsEnumerator.Current 6858
WideStrings.TWideStringsEnumerator.GetCurrent 6857
WideStrings.TWideStringsEnumerator.MoveNext 6857
WideStrUtils 6865
WideStrUtils namespace 6865
Constants 6891
Functions 6867
Structs, Records, Enums 6889
Types 6890
WideStrUtils.AnsiToUtf8Ex 6868
WideStrUtils.AnsiToUtf8Ex function 6868
WideStrUtils.CharSet 6890
WideStrUtils.CharSet type 6890
WideStrUtils.ConvertStreamFromAnsiToUTF8 6868
WideStrUtils.ConvertStreamFromAnsiToUTF8 function 6868
WideStrUtils.ConvertStreamFromUTF8ToAnsi 6869
WideStrUtils.ConvertStreamFromUTF8ToAnsi function 6869
WideStrUtils.DetectUTF8Encoding 6869
WideStrUtils.DetectUTF8Encoding function 6869
WideStrUtils.HasExtendCharacter 6870
WideStrUtils.HasExtendCharacter function 6870
WideStrUtils.HasUTF8BOM 6870
WideStrUtils.HasUTF8BOM function 6870
WideStrUtils.inOpArray 6871
WideStrUtils.inOpArray function 6871
WideStrUtils.inOpSet 6871
WideStrUtils.inOpSet function 6871
WideStrUtils.IsUTF8LeadByte 6871
WideStrUtils.IsUTF8LeadByte function 6871
WideStrUtils.IsUTF8String 6872
WideStrUtils.IsUTF8String function 6872
WideStrUtils.IsUTF8TrailByte 6872
WideStrUtils.IsUTF8TrailByte function 6872
WideStrUtils.LoadResWideString 6873
WideStrUtils.LoadResWideString function 6873
WideStrUtils.LoadWideStr 6873
WideStrUtils.LoadWideStr function 6873
WideStrUtils.sUTF8BOMString 6891
WideStrUtils.sUTF8BOMString constant 6891
WideStrUtils.TEncodeType 6889
WideStrUtils.TEncodeType enumeration 6889
WideStrUtils.UTF8CharLength 6874
WideStrUtils.UTF8CharLength function 6874
WideStrUtils.UTF8CharSize 6874
WideStrUtils.UTF8CharSize function 6874
WideStrUtils.UTF8LowerCase 6875
WideStrUtils.UTF8LowerCase function 6875
WideStrUtils.Utf8ToAnsiEx 6875
WideStrUtils.Utf8ToAnsiEx function 6875
WideStrUtils.UTF8UpperCase 6876
WideStrUtils.UTF8UpperCase function 6876
WideStrUtils.WideAdjustLineBreaks 6876
WideStrUtils.WideAdjustLineBreaks function 6876
WideStrUtils.WideDequotedStr 6877
WideStrUtils.WideDequotedStr function 6877
WideStrUtils.WideExtractQuotedStr 6877
WideStrUtils.WideExtractQuotedStr function 6877
WideStrUtils.WideLastChar 6878
WideStrUtils.WideLastChar function 6878
WideStrUtils.WideQuotedStr 6878
WideStrUtils.WideQuotedStr function 6878
WideStrUtils.WideReplaceStr 6879
WideStrUtils.WideReplaceStr function 6879
WideStrUtils.WideReplaceText 6880
WideStrUtils.WideReplaceText function 6880
WideStrUtils.WideStringReplace 6880
WideStrUtils.WideStringReplace function 6880
WideStrUtils.WStrAlloc 6881
WideStrUtils.WStrAlloc function 6881
WideStrUtils.WStrBufSize 6881
WideStrUtils.WStrBufSize function 6881
WideStrUtils.WStrCat 6882
WideStrUtils.WStrCat function 6882
WideStrUtils.WStrComp 6882
WideStrUtils.WStrComp function 6882
WideStrUtils.WStrCopy 6883
WideStrUtils.WStrCopy function 6883
WideStrUtils.WStrDispose 6883

WideStrUtils.WStrDispose function 6883
WideStrUtils.WStrEnd 6884
WideStrUtils.WStrEnd function 6884
WideStrUtils.WStrLCopy 6884
WideStrUtils.WStrLCopy function 6884
WideStrUtils.WStrLen 6885
WideStrUtils.WStrLen function 6885
WideStrUtils.WStrMove 6885
WideStrUtils.WStrMove function 6885
WideStrUtils.WStrNew 6886
WideStrUtils.WStrNew function 6886
WideStrUtils.WStrPCopy 6886
WideStrUtils.WStrPCopy function 6886
WideStrUtils.WStrPLCopy 6887
WideStrUtils.WStrPLCopy function 6887
WideStrUtils.WStrPos 6887
WideStrUtils.WStrPos function 6887
WideStrUtils.WStrScan 6888
WideStrUtils.WStrScan function 6888
Width
IWebBrowser interface 749
IWebBrowser2Disp interface 775
IWebBrowserAppDisp interface 795
IWebBrowserDisp interface 801
TInternetExplorer class 458
TScreenTipItem class 334
TTabPage class 4452
Win32ErrCode
TService class 3085
WindowActivate
DWebBrowserEvents interface 625
WindowClosing
DWebBrowserEvents2 interface 633
WindowMove
DWebBrowserEvents interface 625
WindowRegistered
DShellWindowsEvents interface 620
WindowResize
DWebBrowserEvents interface 626
WindowRevoked
DShellWindowsEvents interface 620
WindowSetHeight
DWebBrowserEvents2 interface 634
WindowSetLeft
DWebBrowserEvents2 interface 634
WindowSetResizable
DWebBrowserEvents2 interface 634
WindowSetTop
DWebBrowserEvents2 interface 634
WindowSetWidth
DWebBrowserEvents2 interface 635
WinHelpViewer 6893
WinHelpViewer namespace 6893
Interfaces 6894
Variables 6897
WinHelpViewer.IWinHelpTester 6894
WinHelpViewer.IWinHelpTester.CanShowALink 6895
WinHelpViewer.IWinHelpTester.CanShowContext 6895
WinHelpViewer.IWinHelpTester.CanShowTopic 6895
WinHelpViewer.IWinHelpTester.GetDefaultHelpFile 6895
WinHelpViewer.IWinHelpTester.GetHelpPath 6896
WinHelpViewer.IWinHelpTester.GetHelpStrings 6896
WinHelpViewer.ViewerName 6897
WinHelpViewer.ViewerName variable 6897
WinHelpViewer.WinHelpTester 6897
WinHelpViewer.WinHelpTester variable 6897
WMEraseBkgnd
TShadowWindow class 404
WMMIDASIcon
TSocketForm class 180
WMNCCalcSize
TToolWindow class 4641
WMNCCreate
TToolDockForm class 4636
WMNCHitTest
TScreenTipsWindow class 357
TToolDockForm class 4636
WMNCLButtonDown

TToolDockForm class 4637
WMNCPaint
TToolWindow class 4642
WMSize
TToolDockForm class 4637
WMSysCommand
TToolDockForm class 4637
WndProc
TButtonControl class 2112
TCustomCombo class 2304
TCustomComboBox class 2331
TCustomListBox class 2418
TSocketForm class 180
TStreamedConnection class 245
WordWrap
TButton class 2072
TButtonControl class 2115
TCheckBox class 2150
TCustomLabel class 2397
TCustomMemo class 2455
TLabel class 2636
TMemo class 2775
TRadioButton class 2847
wpLoginRequired enumeration member 6305
wpPublished enumeration member 6305
wrAbandoned enumeration member 3156
wrError enumeration member 3156
wrIOCompletion enumeration member 3156
Write
IDataBlock interface 271
TCustomWinSocket class 81
TDataBlock class 204
TScriptProducer class 6713, 6714
TWinSocketStream class 142
TZCompressionStream class 7910
TZDecompressionStream class 7915
WriteDOMVendor
TXMLDocument class 7380
WriteItem
TScriptProducer class 6714
WriteSessionID
ISessionsServiceStoreID interface 1072
WriteSettings
TSocketForm class 181
wrSignaled enumeration member 3156
wrTimeout enumeration member 3156
WSDL
TWSDLView class 7276
WSDLBind 6899
WSDLBind namespace 6899
Classes 6902
Constants 7173
Functions 7171
Interfaces 7078
WSDLBind.GetDefinition 7171
WSDLBind.GetDefinition function 7171
WSDLBind.IBinding 7078
WSDLBind.IBinding.BindingOperations 7082
WSDLBind.IBinding.GetBindingOperations 7080
WSDLBind.IBinding.GetName 7081
WSDLBind.IBinding.GetType_ 7081
WSDLBind.IBinding.Name 7082
WSDLBind.IBinding.SetName 7081
WSDLBind.IBinding.SetType_ 7081
WSDLBind.IBinding.Type_ 7082
WSDLBind.IBindingInput 7082
WSDLBind.IBindingOperation 7083
WSDLBind.IBindingOperation.Faults 7086
WSDLBind.IBindingOperation.GetFaults 7085
WSDLBind.IBindingOperation.GetInput 7085
WSDLBind.IBindingOperation.GetName 7085
WSDLBind.IBindingOperation.GetOutput 7085
WSDLBind.IBindingOperation.Input 7086
WSDLBind.IBindingOperation.Name 7087
WSDLBind.IBindingOperation.Output 7087
WSDLBind.IBindingOperation.SetName 7086
WSDLBind.IBindingOperations 7087
WSDLBind.IBindingOperations.Add 7089

WSDLBind.IBindingOperations.BindingOperation 7090
WSDLBind.IBindingOperations.GetBindingOperation 7089
WSDLBind.IBindingOperations.Insert 7090
WSDLBind.IBindingOutput 7090
WSDLBind.IBindings 7091
WSDLBind.IBindings.Add 7093
WSDLBind.IBindings.Binding 7094
WSDLBind.IBindings.GetBinding 7093
WSDLBind.IBindings.Insert 7093
WSDLBind.IDefinition 7094
WSDLBind.IDefinition.Bindings 7099
WSDLBind.IDefinition.GetBindings 7096
WSDLBind.IDefinition.GetImports 7096
WSDLBind.IDefinition.GetMessages 7097
WSDLBind.IDefinition.GetName 7097
WSDLBind.IDefinition.GetPortTypes 7097
WSDLBind.IDefinition.GetServices 7097
WSDLBind.IDefinition.GetTargetNamespace 7097
WSDLBind.IDefinition.GetTypes 7098
WSDLBind.IDefinition.Imports 7099
WSDLBind.IDefinition.Messages 7099
WSDLBind.IDefinition.Name 7099
WSDLBind.IDefinition.PortTypes 7099
WSDLBind.IDefinition.Services 7100
WSDLBind.IDefinition.SetName 7098
WSDLBind.IDefinition.SetTargetNamespace 7098
WSDLBind.IDefinition.TargetNamespace 7100
WSDLBind.IDefinition.Types 7100
WSDLBind.IDocumentation 7100
WSDLBind.IDocumentationCollection 7101
WSDLBind.IDocumentationCollection.Add 7103
WSDLBind.IDocumentationCollection.Documentation 7104
WSDLBind.IDocumentationCollection.GetDocumentation 7103
WSDLBind.IDocumentationCollection.Insert 7103
WSDLBind.IDocumented 7104
WSDLBind.IDocumented.Documentation 7106
WSDLBind.IDocumented.GetDocumentationCollection 7106
WSDLBind.IFault 7106
WSDLBind.IFaults 7107
WSDLBind.IFaults.Add 7109
WSDLBind.IFaults.Fault 7110
WSDLBind.IFaults.GetFault 7109
WSDLBind.IFaults.Insert 7109
WSDLBind.IImport 7110
WSDLBind.IImport.GetLocation 7112
WSDLBind.IImport.GetNamespace 7112
WSDLBind.IImport.Location 7113
WSDLBind.IImport.Namespace 7113
WSDLBind.IImport.SetLocation 7112
WSDLBind.IImport.SetNamespace 7112
WSDLBind.IImports 7113
WSDLBind.IImports.Add 7115
WSDLBind.IImports.GetImport 7116
WSDLBind.IImports.Import 7116
WSDLBind.IImports.Insert 7116
WSDLBind.IMessage 7116
WSDLBind.IMessage.GetName 7118
WSDLBind.IMessage.GetParts 7119
WSDLBind.IMessage.Name 7119
WSDLBind.IMessage.Parts 7119
WSDLBind.IMessage.SetName 7119
WSDLBind.IMessages 7120
WSDLBind.IMessages.Add 7122
WSDLBind.IMessages.GetMessage 7122
WSDLBind.IMessages.Insert 7122
WSDLBind.IMessages.Message 7123
WSDLBind.IOperation 7123
WSDLBind.IOperation.Faults 7126
WSDLBind.IOperation.GetFaults 7125
WSDLBind.IOperation.GetInput 7125
WSDLBind.IOperation.GetName 7125
WSDLBind.IOperation.GetOutput 7126
WSDLBind.IOperation.Input 7126
WSDLBind.IOperation.Name 7127
WSDLBind.IOperation.Output 7127
WSDLBind.IOperation.SetName 7126
WSDLBind.IOperations 7127
WSDLBind.IOperations.Add 7129, 7130

WSDLBind.IOperations.GetOperation 7130
WSDLBind.IOperations.Insert 7130
WSDLBind.IOperations.Operation 7130
WSDLBind.IParam 7131
WSDLBind.IParam.GetMessage 7133
WSDLBind.IParam.GetName 7133
WSDLBind.IParam.Message 7134
WSDLBind.IParam.Name 7134
WSDLBind.IParam.SetMessage 7133
WSDLBind.IParam.SetName 7133
WSDLBind.IPart 7134
WSDLBind.IPart.Element 7138
WSDLBind.IPart.GetElement 7136
WSDLBind.IPart.GetName 7137
WSDLBind.IPart.GetType_ 7137
WSDLBind.IPart.Name 7138
WSDLBind.IPart.SetElement 7137
WSDLBind.IPart.SetName 7137
WSDLBind.IPart.SetType_ 7137
WSDLBind.IPart.Type_ 7138
WSDLBind.IParts 7139
WSDLBind.IParts.Add 7141
WSDLBind.IParts.GetPart 7141
WSDLBind.IParts.Insert 7141
WSDLBind.IParts.Part 7142
WSDLBind.IPort 7142
WSDLBind.IPort.Binding 7145
WSDLBind.IPort.GetBinding 7144
WSDLBind.IPort.GetName 7144
WSDLBind.IPort.Name 7145
WSDLBind.IPort.SetBinding 7144
WSDLBind.IPort.SetName 7144
WSDLBind.IPorts 7145
WSDLBind.IPorts.Add 7147
WSDLBind.IPorts.GetPort 7148
WSDLBind.IPorts.Insert 7148
WSDLBind.IPorts.Port 7148
WSDLBind.IPortType 7148
WSDLBind.IPortType.GetName 7150
WSDLBind.IPortType.GetOperations 7151
WSDLBind.IPortType.Name 7151
WSDLBind.IPortType.Operations 7151
WSDLBind.IPortType.SetName 7151
WSDLBind.IPortTypes 7152
WSDLBind.IPortTypes.Add 7154
WSDLBind.IPortTypes.GetPortType 7154
WSDLBind.IPortTypes.Insert 7154
WSDLBind.IPortTypes.PortType 7155
WSDLBind.IService 7155
WSDLBind.IService.GetName 7157
WSDLBind.IService.GetPorts 7157
WSDLBind.IService.Name 7158
WSDLBind.IService.Ports 7158
WSDLBind.IService.SetName 7157
WSDLBind.IServices 7158
WSDLBind.IServices.Add 7160
WSDLBind.IServices.GetService 7160
WSDLBind.IServices.Insert 7161
WSDLBind.IServices.Service 7161
WSDLBind.ITypes 7161
WSDLBind.ITypes.GetSchemaDefs 7163
WSDLBind.ITypes.SchemaDefs 7163
WSDLBind.IWSDLDocument 7164
WSDLBind.IWSDLDocument.Definition 7167
WSDLBind.IWSDLDocument.GetDefinition 7166
WSDLBind.IWSDLDocument.SetDefinition 7166
WSDLBind.IXMLSchemaDefs 7168
WSDLBind.IXMLSchemaDefs.Add 7170
WSDLBind.IXMLSchemaDefs.GetXMLSchemaDef 7170
WSDLBind.IXMLSchemaDefs.SchemaDef 7170
WSDLBind.LoadWSDLDoc 7171
WSDLBind.LoadWSDLDoc function 7171
WSDLBind.NewWSDLDoc 7171
WSDLBind.NewWSDLDoc function 7171
WSDLBind.SBinding 7173
WSDLBind.SBinding constant 7173
WSDLBind.SDefinitions 7173
WSDLBind.SDefinitions constant 7173

WSDLBind.SFault 7174
WSDLBind.SFault constant 7174
WSDLBind.SImport 7174
WSDLBind.SImport constant 7174
WSDLBind.SInput 7174
WSDLBind.SInput constant 7174
WSDLBind.SLocation 7175
WSDLBind.SLocation constant 7175
WSDLBind.SMessage 7175
WSDLBind.SMessage constant 7175
WSDLBind.SNamespace 7175
WSDLBind.SNamespace constant 7175
WSDLBind.SOperation 7176
WSDLBind.SOperation constant 7176
WSDLBind.SOutput 7176
WSDLBind.SOutput constant 7176
WSDLBind.SPart 7176
WSDLBind.SPart constant 7176
WSDLBind.SPort 7177
WSDLBind.SPort constant 7177
WSDLBind.SPortType 7177
WSDLBind.SPortType constant 7177
WSDLBind.SService 7177
WSDLBind.SService constant 7177
WSDLBind.SType 7178
WSDLBind.SType constant 7178
WSDLBind.STypes 7178
WSDLBind.STypes constant 7178
WSDLBind.TBinding 6902
WSDLBind.TBinding.AfterConstruction 6908
WSDLBind.TBinding.GetBindingOperations 6908
WSDLBind.TBinding.GetName 6908
WSDLBind.TBinding.GetType_ 6909
WSDLBind.TBinding.SetName 6909
WSDLBind.TBinding.SetType_ 6909
WSDLBind.TBindingOperation 6909
WSDLBind.TBindingOperation.AfterConstruction 6915
WSDLBind.TBindingOperation.GetFaults 6915
WSDLBind.TBindingOperation.GetInput 6916
WSDLBind.TBindingOperation.GetName 6916
WSDLBind.TBindingOperation.GetOutput 6916
WSDLBind.TBindingOperation.SetName 6916
WSDLBind.TBindingOperations 6917
WSDLBind.TBindingOperations.Add 6922
WSDLBind.TBindingOperations.GetBindingOperation 6923
WSDLBind.TBindingOperations.Insert 6923
WSDLBind.TBindings 6923
WSDLBind.TBindings.Add 6929
WSDLBind.TBindings.GetBinding 6929
WSDLBind.TBindings.Insert 6929
WSDLBind.TDefinition 6930
WSDLBind.TDefinition.AfterConstruction 6935
WSDLBind.TDefinition.GetBindings 6936
WSDLBind.TDefinition.GetImports 6936
WSDLBind.TDefinition.GetMessages 6936
WSDLBind.TDefinition.GetName 6936
WSDLBind.TDefinition.GetPortTypes 6937
WSDLBind.TDefinition.GetServices 6937
WSDLBind.TDefinition.GetTargetNamespace 6937
WSDLBind.TDefinition.GetTypes 6937
WSDLBind.TDefinition.SetName 6938
WSDLBind.TDefinition.SetTargetNamespace 6938
WSDLBind.TDocumentation 6938
WSDLBind.TDocumentationCollection 6939
WSDLBind.TDocumentationCollection.Add 6944
WSDLBind.TDocumentationCollection.GetDocumentation 6945
WSDLBind.TDocumentationCollection.Insert 6945
WSDLBind.TDocumented 6945
WSDLBind.TDocumented.AfterConstruction 6950
WSDLBind.TDocumented.GetDocumentationCollection 6951
WSDLBind.TFault 6951
WSDLBind.TFaults 6952
WSDLBind.TFaults.Add 6957
WSDLBind.TFaults.GetFault 6958
WSDLBind.TFaults.Insert 6958
WSDLBind.TImport 6958
WSDLBind.TImport.GetLocation 6964
WSDLBind.TImport.GetNamespace 6964

WSDLBind.TImport.SetLocation 6964
WSDLBind.TImport.SetNamespace 6964
WSDLBind.TImports 6965
WSDLBind.TImports.Add 6970
WSDLBind.TImports.GetImport 6971
WSDLBind.TImports.Insert 6971
WSDLBind.TMessage 6971
WSDLBind.TMessage.AfterConstruction 6977
WSDLBind.TMessage.GetName 6977
WSDLBind.TMessage.GetParts 6977
WSDLBind.TMessage.SetName 6977
WSDLBind.TMessages 6978
WSDLBind.TMessages.Add 6984
WSDLBind.TMessages.GetMessage 6984
WSDLBind.TMessages.Insert 6984
WSDLBind.TOperation 6984
WSDLBind.TOperation.AfterConstruction 6990
WSDLBind.TOperation.GetFaults 6990
WSDLBind.TOperation.GetInput 6990
WSDLBind.TOperation.GetName 6991
WSDLBind.TOperation.GetOutput 6991
WSDLBind.TOperation.SetName 6991
WSDLBind.TOperations 6991
WSDLBind.TOperations.Add 6997, 6998
WSDLBind.TOperations.GetOperation 6998
WSDLBind.TOperations.Insert 6998
WSDLBind.TParam 6998
WSDLBind.TParam.GetMessage 7004
WSDLBind.TParam.GetName 7004
WSDLBind.TParam.SetMessage 7004
WSDLBind.TParam.SetName 7004
WSDLBind.TPart 7005
WSDLBind.TPart.GetElement 7010
WSDLBind.TPart.GetName 7010
WSDLBind.TPart.GetType_ 7011
WSDLBind.TPart.SetElement 7011
WSDLBind.TPart.SetName 7011
WSDLBind.TPart.SetType_ 7011
WSDLBind.TParts 7012
WSDLBind.TParts.Add 7017
WSDLBind.TParts.GetPart 7018
WSDLBind.TParts.Insert 7018
WSDLBind.TPort 7018
WSDLBind.TPort.GetBinding 7024
WSDLBind.TPort.GetName 7024
WSDLBind.TPort.SetBinding 7024
WSDLBind.TPort.SetName 7024
WSDLBind.TPorts 7025
WSDLBind.TPorts.Add 7030
WSDLBind.TPorts.GetPort 7031
WSDLBind.TPorts.Insert 7031
WSDLBind.TPortType 7031
WSDLBind.TPortType.AfterConstruction 7037
WSDLBind.TPortType.GetName 7037
WSDLBind.TPortType.GetOperations 7037
WSDLBind.TPortType.SetName 7037
WSDLBind.TPortTypes 7038
WSDLBind.TPortTypes.Add 7044
WSDLBind.TPortTypes.GetPortType 7044
WSDLBind.TPortTypes.Insert 7044
WSDLBind.TService 7044
WSDLBind.TService.AfterConstruction 7050
WSDLBind.TService.GetName 7050
WSDLBind.TService.GetPorts 7050
WSDLBind.TService.SetName 7051
WSDLBind.TServices 7051
WSDLBind.TServices.Add 7057
WSDLBind.TServices.GetService 7057
WSDLBind.TServices.Insert 7057
WSDLBind.TTypes 7057
WSDLBind.TTypes.AfterConstruction 7063
WSDLBind.TTypes.GetSchemaDefs 7063
WSDLBind.TWSDLDocument 7063
WSDLBind.TWSDLDocument.AfterConstruction 7069
WSDLBind.TWSDLDocument.Definition 7070
WSDLBind.TWSDLDocument.GetDefinition 7069
WSDLBind.TWSDLDocument.LoadData 7069
WSDLBind.TWSDLDocument.SetDefinition 7070

WSDLBind.TXMLSchemaDefs 7071
WSDLBind.TXMLSchemaDefs.Add 7076
WSDLBind.TXMLSchemaDefs.GetXMLSchemaDef 7077
WSDLElements
TWebServExp class 6733
WSDLIntf 7179
WSDLIntf namespace 7179
Classes 7181
Constants 7188
Structs, Records, Enums 7185
Types 7187
WSDLIntf.Directives 7189
WSDLIntf.Directives constant 7189
WSDLIntf.DynArrayTypes 7189
WSDLIntf.DynArrayTypes constant 7189
WSDLIntf.Httpns 7190
WSDLIntf.Httpns constant 7190
WSDLIntf.Operators 7190
WSDLIntf.Operators constant 7190
WSDLIntf.OtherReservedWords 7190
WSDLIntf.OtherReservedWords constant 7190
WSDLIntf.ReservedWords 7191
WSDLIntf.ReservedWords constant 7191
WSDLIntf.SAddress 7191
WSDLIntf.SAddress constant 7191
WSDLIntf.SAnySimpleType 7192
WSDLIntf.SAnySimpleType constant 7192
WSDLIntf.SAnyType 7192
WSDLIntf.SAnyType constant 7192
WSDLIntf.SArray 7192
WSDLIntf.SArray constant 7192
WSDLIntf.SArrayOf 7193
WSDLIntf.SArrayOf constant 7193
WSDLIntf.SArrayType 7193
WSDLIntf.SArrayType constant 7193
WSDLIntf.SBinding 7193
WSDLIntf.SBinding constant 7193
WSDLIntf.SBody 7194
WSDLIntf.SBody constant 7194
WSDLIntf.ScalarArrayTypes 7194
WSDLIntf.ScalarArrayTypes constant 7194
WSDLIntf.SComplexType 7194
WSDLIntf.SComplexType constant 7194
WSDLIntf.SDefinitions 7195
WSDLIntf.SDefinitions constant 7195
WSDLIntf.SDynArray 7195
WSDLIntf.SDynArray constant 7195
WSDLIntf.SElement 7195
WSDLIntf.SElement constant 7195
WSDLIntf.SEncodingStyle 7196
WSDLIntf.SEncodingStyle constant 7196
WSDLIntf.SFault 7196
WSDLIntf.SFault constant 7196
WSDLIntf.SHeader 7196
WSDLIntf.SHeader constant 7196
WSDLIntf.SImport 7197
WSDLIntf.SImport constant 7197
WSDLIntf.SInOut 7197
WSDLIntf.SInOut constant 7197
WSDLIntf.SInput 7197
WSDLIntf.SInput constant 7197
WSDLIntf.SLocation 7198
WSDLIntf.SLocation constant 7198
WSDLIntf.SMessage 7198
WSDLIntf.SMessage constant 7198
WSDLIntf.SName 7198
WSDLIntf.SName constant 7198
WSDLIntf.SNameSpace 7199
WSDLIntf.SNameSpace constant 7199
WSDLIntf.SNsPrefix 7199
WSDLIntf.SNsPrefix constant 7199
WSDLIntf.Soapns 7199
WSDLIntf.Soapns constant 7199
WSDLIntf.SOperation 7200
WSDLIntf.SOperation constant 7200
WSDLIntf.SOutput 7200
WSDLIntf.SOutput constant 7200
WSDLIntf.SPart 7200

WSDLIntf.SPart constant 7200
WSDLIntf.SParts 7201
WSDLIntf.SParts constant 7201
WSDLIntf.SPort 7201
WSDLIntf.SPort constant 7201
WSDLIntf.SPortType 7201
WSDLIntf.SPortType constant 7201
WSDLIntf.SRequest 7202
WSDLIntf.SRequest constant 7202
WSDLIntf.SResponse 7202
WSDLIntf.SResponse constant 7202
WSDLIntf.SReturn 7202
WSDLIntf.SReturn constant 7202
WSDLIntf.SSchema 7203
WSDLIntf.SSchema constant 7203
WSDLIntf.SService 7203
WSDLIntf.SService constant 7203
WSDLIntf.SSoapAction 7203
WSDLIntf.SSoapAction constant 7203
WSDLIntf.SStyle 7204
WSDLIntf.SStyle constant 7204
WSDLIntf.STns 7204
WSDLIntf.STns constant 7204
WSDLIntf.STransport 7204
WSDLIntf.STransport constant 7204
WSDLIntf.SType 7205
WSDLIntf.SType constant 7205
WSDLIntf.STypes 7205
WSDLIntf.STypes constant 7205
WSDLIntf.SUnknown 7205
WSDLIntf.SUnknown constant 7205
WSDLIntf.SUse 7206
WSDLIntf.SUse constant 7206
WSDLIntf.SVerb 7206
WSDLIntf.SVerb constant 7206
WSDLIntf.TDOMStrings 7181
WSDLIntf.TDOMStrings.Add 7182
WSDLIntf.TDOMStrings.Clear 7183
WSDLIntf.TDOMStrings.Count 7183
WSDLIntf.TDOMStrings.Create 7182
WSDLIntf.TDOMStrings.Destroy 7182
WSDLIntf.TDOMStrings.IndexOf 7183
WSDLIntf.TDOMStrings.IndexOfIgnoreCase 7183
WSDLIntf.TDOMStrings.Insert 7183
WSDLIntf.TDOMStrings.Strings 7184
WSDLIntf.tns 7206
WSDLIntf.tns constant 7206
WSDLIntf.TWebServiceBindingType 7185
WSDLIntf.TWebServiceBindingType enumeration 7185
WSDLIntf.TWSDLElement 7186
WSDLIntf.TWSDLElement enumeration 7186
WSDLIntf.TWSDLElements 7187
WSDLIntf.TWSDLElements type 7187
WSDLIntf.TWString 7186
WSDLIntf.TWString record 7186
WSDLIntf.Wsdlns 7207
WSDLIntf.Wsdlns constant 7207
WSDLIntf.xsdns 7207
WSDLIntf.xsdns constant 7207
WSDLItems 7209
  THTTPRIO class 1255
WSDLItems namespace 7209
  Classes 7211
  Functions 7259
  Interfaces 7247
  Structs, Records, Enums 7261
  Types 7262
WSDLItems.HasDefinition 7259
WSDLItems.HasDefinition function 7259
WSDLItems.IHeaderInfo 7247
WSDLItems.IHeaderInfo.EncodingStyle 7251
WSDLItems.IHeaderInfo.GetEncodingStyle 7248
WSDLItems.IHeaderInfo.GetName 7248
WSDLItems.IHeaderInfo.GetNamespace 7249
WSDLItems.IHeaderInfo.GetPart 7249
WSDLItems.IHeaderInfo.GetRequired 7249
WSDLItems.IHeaderInfo.GetUse 7249
WSDLItems.IHeaderInfo.Name 7251

WSDLItems.IHeaderInfo.Namespace 7251
WSDLItems.IHeaderInfo.Part 7252
WSDLItems.IHeaderInfo.Required 7252
WSDLItems.IHeaderInfo.SetEncodingStyle 7249
WSDLItems.IHeaderInfo.SetName 7250
WSDLItems.IHeaderInfo.SetNamespace 7250
WSDLItems.IHeaderInfo.SetPart 7250
WSDLItems.IHeaderInfo.SetRequired 7250
WSDLItems.IHeaderInfo.SetUse 7251
WSDLItems.IHeaderInfo.Use 7252
WSDLItems.IHeaderInfoArray 7262
WSDLItems.IHeaderInfoArray type 7262
WSDLItems.IQualifiedName 7252
WSDLItems.IQualifiedName.GetName 7253
WSDLItems.IQualifiedName.GetNamespace 7254
WSDLItems.IQualifiedName.Name 7255
WSDLItems.IQualifiedName.Namespace 7255
WSDLItems.IQualifiedName.SetName 7254
WSDLItems.IQualifiedName.SetNamespace 7254
WSDLItems.IQualifiedNameArray 7262
WSDLItems.IQualifiedNameArray type 7262
WSDLItems.IterateImportOptions 7261
WSDLItems.IterateImportOptions enumeration 7261
WSDLItems.IWSDLItems 7256
WSDLItems.IWSDLItems.GetWSDLDoc 7257
WSDLItems.IWSDLItems.GetWSDLItems 7257
WSDLItems.IWSDLItems.Load 7257
WSDLItems.NewQualifiedName 7259
WSDLItems.NewQualifiedName function 7259
WSDLItems.TWSDLItems 7211
WSDLItems.TWSDLItems.CompareName 7219
WSDLItems.TWSDLItems.Create 7217
WSDLItems.TWSDLItems.GetBindingForServicePort 7220
WSDLItems.TWSDLItems.GetBindingOfType 7221
WSDLItems.TWSDLItems.GetBindingOperationNode 7221
WSDLItems.TWSDLItems.GetBindings 7222
WSDLItems.TWSDLItems.GetBindingType 7222
WSDLItems.TWSDLItems.GetHttpBindingAttribute 7223
WSDLItems.TWSDLItems.GetImports 7223
WSDLItems.TWSDLItems.GetLocationForImport 7224
WSDLItems.TWSDLItems.GetMessageNode 7224
WSDLItems.TWSDLItems.GetMessages 7224
WSDLItems.TWSDLItems.GetName 7225
WSDLItems.TWSDLItems.GetOperationNode 7225
WSDLItems.TWSDLItems.GetOperations 7226
WSDLItems.TWSDLItems.GetOperationsForBinding 7226
WSDLItems.TWSDLItems.GetPartNode 7227
WSDLItems.TWSDLItems.GetParts 7227
WSDLItems.TWSDLItems.GetPartsForOperation 7228
WSDLItems.TWSDLItems.GetPortsForService 7229
WSDLItems.TWSDLItems.GetPortTypeNode 7229
WSDLItems.TWSDLItems.GetPortTypes 7229
WSDLItems.TWSDLItems.GetSchemaNode 7230
WSDLItems.TWSDLItems.GetSchemas 7230
WSDLItems.TWSDLItems.GetServiceAndPortOfBinding 7231
WSDLItems.TWSDLItems.GetServiceNode 7231
WSDLItems.TWSDLItems.GetServices 7232
WSDLItems.TWSDLItems.GetSoapAction 7232
WSDLItems.TWSDLItems.GetSoapActionList 7233
WSDLItems.TWSDLItems.GetSoapAddressForServicePort 7234
WSDLItems.TWSDLItems.GetSoapBindingAttribute 7234
WSDLItems.TWSDLItems.GetSoapBindingInputNode 7235
WSDLItems.TWSDLItems.GetSoapBindingIONode 7235
WSDLItems.TWSDLItems.GetSoapBindingOutputNode 7236
WSDLItems.TWSDLItems.GetSoapBodyAttribute 7236
WSDLItems.TWSDLItems.GetSoapBodyInputAttribute 7237
WSDLItems.TWSDLItems.GetSoapBodyNamespace 7238
WSDLItems.TWSDLItems.GetSoapBodyOutputAttribute 7238
WSDLItems.TWSDLItems.GetSoapHeaders 7239
WSDLItems.TWSDLItems.GetSoapInputHeaders 7240
WSDLItems.TWSDLItems.GetSoapOperationAttribute 7240
WSDLItems.TWSDLItems.GetSoapOperationStyle 7241
WSDLItems.TWSDLItems.GetSoapOutputHeaders 7241
WSDLItems.TWSDLItems.GetTargetNamespace 7242
WSDLItems.TWSDLItems.GetTargetNamespacePrefix 7243
WSDLItems.TWSDLItems.GetWSDLDoc 7243
WSDLItems.TWSDLItems.GetWSDLItems 7243
WSDLItems.TWSDLItems.HasTypesNode 7244

WSDLItems.TWSDLItems.IsPortTypeHTTPBound 7244
WSDLItems.TWSDLItems.IterateImports 7244
WSDLItems.TWSDLItems.Load 7245
WSDLItems.TWSDLItems.StreamLoader 7245
WSDLItems.TWSDLItems.TargetNamespace 7246
WSDLItems.TWSDLIterateProc 7263
WSDLItems.TWSDLIterateProc type 7263
WSDLLocation
THTTPRIO class 1255
WSDLNode 7265
WSDLNode namespace 7265
Classes 7266
Functions 7277
WSDLNode.ActivateWSDL 7277
WSDLNode.ActivateWSDL function 7277
WSDLNode.EWSDLLoadException 7266
WSDLNode.TWSDLClientNode 7266
WSDLNode.TWSDLClientNode.BeforeExecute 7267
WSDLNode.TWSDLClientNode.Execute 7268, 7269
WSDLNode.TWSDLClientNode.GetMimeBoundary 7269
WSDLNode.TWSDLClientNode.Notification 7269
WSDLNode.TWSDLClientNode.SetMimeBoundary 7270
WSDLNode.TWSDLClientNode.TransportNode 7270
WSDLNode.TWSDLClientNode.WSDLView 7271
WSDLNode.TWSDLView 7271
WSDLNode.TWSDLView.Activate 7273
WSDLNode.TWSDLView.IntfInfo 7272
WSDLNode.TWSDLView.Operation 7273
WSDLNode.TWSDLView.Password 7274
WSDLNode.TWSDLView.Port 7274
WSDLNode.TWSDLView.PortType 7274
WSDLNode.TWSDLView.Proxy 7275
WSDLNode.TWSDLView.Service 7275
WSDLNode.TWSDLView.SetDesignState 7273
WSDLNode.TWSDLView.UserName 7275
WSDLNode.TWSDLView.WSDL 7276
WSDLPub 7279
WSDLPub namespace 7279
Classes 7280
Functions 7297
Interfaces 7295
Structs, Records, Enums 7298
Types 7299
WSDLPub.IWSDLPublish 7295
WSDLPub.IWSDLPublish.GetPortTypeList 7296
WSDLPub.IWSDLPublish.GetTypeSystemsList 7296
WSDLPub.IWSDLPublish.GetWSDLForPortType 7296
WSDLPub.IWSDLPublish.GetXSDForTypeSystem 7296
WSDLPub.SOAPPublishOptions 7298
WSDLPub.SOAPPublishOptions enumeration 7298
WSDLPub.TBeforePublishingWSDLEvent 7299
WSDLPub.TBeforePublishingWSDLEvent type 7299
WSDLPub.TSOAPPublishOptions 7299
WSDLPub.TSOAPPublishOptions type 7299
WSDLPub.TWSDLHTMLPublish 7280
WSDLPub.TWSDLHTMLPublish.AddInterfaceList 7283
WSDLPub.TWSDLHTMLPublish.AddPortList 7283
WSDLPub.TWSDLHTMLPublish.AdminEnabled 7287
WSDLPub.TWSDLHTMLPublish.Create 7281
WSDLPub.TWSDLHTMLPublish.Destroy 7282
WSDLPub.TWSDLHTMLPublish.DispatchEnabled 7283
WSDLPub.TWSDLHTMLPublish.DispatchMask 7283
WSDLPub.TWSDLHTMLPublish.DispatchMethodType 7284
WSDLPub.TWSDLHTMLPublish.DispatchRequest 7284
WSDLPub.TWSDLHTMLPublish.GetHostScriptBaseURL 7285
WSDLPub.TWSDLHTMLPublish.GetTargetNamespace 7285
WSDLPub.TWSDLHTMLPublish.HandleRequest 7285
WSDLPub.TWSDLHTMLPublish.OnAfterPublishingWSDL 7289
WSDLPub.TWSDLHTMLPublish.OnBeforePublishingTypes 7290
WSDLPub.TWSDLHTMLPublish.OnBeforePublishingWSDL 7290
WSDLPub.TWSDLHTMLPublish.OnPublishingType 7290
WSDLPub.TWSDLHTMLPublish.PublishOptions 7288
WSDLPub.TWSDLHTMLPublish.ServiceInfo 7286
WSDLPub.TWSDLHTMLPublish.TargetNamespace 7288
WSDLPub.TWSDLHTMLPublish.UpdatePortList 7286
WSDLPub.TWSDLHTMLPublish.WebDispatch 7288
WSDLPub.TWSDLHTMLPublish.WSILInfo 7286

WSDLPub.TWSDLPublish 7291
WSDLPub.TWSDLPublish.GetPortTypeEntries 7292
WSDLPub.TWSDLPublish.GetPortTypeList 7292
WSDLPub.TWSDLPublish.GetTypeSystemsList 7292
WSDLPub.TWSDLPublish.GetWSDLForPortType 7293
WSDLPub.TWSDLPublish.GetXSDForTypeSystem 7293
WSDLPub.TWSDLPublish.OnAfterPublishingWSDL 7294
WSDLPub.TWSDLPublish.OnBeforePublishingTypes 7294
WSDLPub.TWSDLPublish.OnBeforePublishingWSDL 7294
WSDLPub.TWSDLPublish.OnPublishingType 7294
WSDLPub.TWSDLPublish.TargetNamespace 7293
WSDLPub.WSDLPubFactory 7297
WSDLPub.WSDLPubFactory function 7297
WSDLView
THTTPReqResp class 1293
TWSDLClientNode class 7271
WSILInfo
TWSDLHTMLPublish class 7286

X

X
TDisplayOffset class 325
xdsFile enumeration member 7496
xdsNone enumeration member 7496
xdsStream enumeration member 7496
xdsXMLData enumeration member 7496
xdsXMLProperty enumeration member 7496
xetEBCDICLike enumeration member 7719
xetUCS_4BE enumeration member 7719
xetUCS_4LE enumeration member 7719
xetUCS_4Like enumeration member 7719
xetUCS_4Order2134 enumeration member 7719
xetUCS_4Order3412 enumeration member 7719
xetUnknown enumeration member 7719
xetUTF_16BE enumeration member 7719
xetUTF_16BELike enumeration member 7719
xetUTF_16LE enumeration member 7719
xetUTF_16LELike enumeration member 7719
xetUTF_8 enumeration member 7719
xetUTF_8Like enumeration member 7719
xilCUSTOM enumeration member 1704
xilDIRTYREAD enumeration member 1704
xilREADCOMMITTED enumeration member 1704
xilREPEATABLEREAD enumeration member 1704
xml
IDOMNodeEx interface 7580
IDOMPersist interface 7599
XML
IXMLDocument interface 7668
IXMLNode interface 7697
TXMLDocument class 7392
XMLBrokr 7301
XMLBrokr namespace 7301
Classes 7303
Constants 7343
Functions 7339
Interfaces 7337
Structs, Records, Enums 7340
Types 7341
XMLBrokr.FormatXML 7339
XMLBrokr.FormatXML function 7339
XMLBrokr.INotifyConnectionChange 7337
XMLBrokr.INotifyConnectionChange.ConnectionChange 7338
XMLBrokr.sPostDelta 7343
XMLBrokr.sPostDelta constant 7343
XMLBrokr.sProducer 7343
XMLBrokr.sProducer constant 7343
XMLBrokr.sRedirect 7343
XMLBrokr.sRedirect constant 7343
XMLBrokr.TGetErrorResponseEvent 7341
XMLBrokr.TGetErrorResponseEvent type 7341
XMLBrokr.TGetResponseEvent 7341
XMLBrokr.TGetResponseEvent type 7341
XMLBrokr.TRequestRecordsEvent 7342
XMLBrokr.TRequestRecordsEvent type 7342
XMLBrokr.TRequestUpdateEvent 7342
XMLBrokr.TRequestUpdateEvent type 7342
XMLBrokr.TXMLBroker 7303

XMLBrokr.TXMLBroker.AddNotify 7307
XMLBrokr.TXMLBroker.AfterDispatch 7329
XMLBrokr.TXMLBroker.ApplyXMLUpdates 7308
XMLBrokr.TXMLBroker.AppServer 7322
XMLBrokr.TXMLBroker.AS_ApplyUpdates 7308
XMLBrokr.TXMLBroker.AS_FetchParams 7309
XMLBrokr.TXMLBroker.AS_GetRecords 7309
XMLBrokr.TXMLBroker.BeforeDispatch 7330
XMLBrokr.TXMLBroker.CheckInactive 7309
XMLBrokr.TXMLBroker.Connected 7322
XMLBrokr.TXMLBroker.Create 7305
XMLBrokr.TXMLBroker.Destroy 7306
XMLBrokr.TXMLBroker.DispatchEnabled 7309
XMLBrokr.TXMLBroker.DispatchMask 7309
XMLBrokr.TXMLBroker.DispatchMethodType 7310
XMLBrokr.TXMLBroker.DispatchRequest 7310
XMLBrokr.TXMLBroker.FetchParams 7310
XMLBrokr.TXMLBroker.GetAppServer 7310
XMLBrokr.TXMLBroker.GetConnected 7311
XMLBrokr.TXMLBroker.GetDelta 7311
XMLBrokr.TXMLBroker.GetErrorCount 7311
XMLBrokr.TXMLBroker.GetErrorResponse 7312
XMLBrokr.TXMLBroker.GetErrors 7312
XMLBrokr.TXMLBroker.GetHasAppServer 7312
XMLBrokr.TXMLBroker.GetNotify 7312
XMLBrokr.TXMLBroker.GetNotifyCount 7313
XMLBrokr.TXMLBroker.GetProducerName 7313
XMLBrokr.TXMLBroker.GetRedirect 7313
XMLBrokr.TXMLBroker.GetResponse 7313
XMLBrokr.TXMLBroker.GetXMLRecords 7314
XMLBrokr.TXMLBroker.GetXMLStream 7314
XMLBrokr.TXMLBroker.HasAppServer 7323
XMLBrokr.TXMLBroker.HTMLSubmitFormName 7315
XMLBrokr.TXMLBroker.MasterRowSetVarName 7315
XMLBrokr.TXMLBroker.MaxErrors 7323
XMLBrokr.TXMLBroker.MaxRecords 7324
XMLBrokr.TXMLBroker.Notification 7316
XMLBrokr.TXMLBroker.Notify 7325
XMLBrokr.TXMLBroker.NotifyCount 7325
XMLBrokr.TXMLBroker.OnGetErrorResponse 7330
XMLBrokr.TXMLBroker.OnGetResponse 7331
XMLBrokr.TXMLBroker.OnRequestRecords 7332
XMLBrokr.TXMLBroker.OnRequestUpdate 7332
XMLBrokr.TXMLBroker.Params 7325
XMLBrokr.TXMLBroker.ProviderName 7326
XMLBrokr.TXMLBroker.ReconcileProducer 7327
XMLBrokr.TXMLBroker.RemoteServer 7328
XMLBrokr.TXMLBroker.RemoveNotify 7316
XMLBrokr.TXMLBroker.RequestRecords 7316
XMLBrokr.TXMLBroker.RequestUpdate 7317
XMLBrokr.TXMLBroker.RowSetVarName 7318
XMLBrokr.TXMLBroker.SendConnectEvent 7319
XMLBrokr.TXMLBroker.SetAppServer 7319
XMLBrokr.TXMLBroker.SetConnected 7319
XMLBrokr.TXMLBroker.SetParams 7319
XMLBrokr.TXMLBroker.SetProvider 7320
XMLBrokr.TXMLBroker.SetProviderName 7320
XMLBrokr.TXMLBroker.SetReconcileProducer 7320
XMLBrokr.TXMLBroker.SetRemoteServer 7321
XMLBrokr.TXMLBroker.SetWebDispatch 7321
XMLBrokr.TXMLBroker.SubmitFormVarName 7321
XMLBrokr.TXMLBroker.WebDispatch 7328
XMLBrokr.TXMLOption 7340
XMLBrokr.TXMLOption enumeration 7340
XMLBrokr.TXMLOptions 7342
XMLBrokr.TXMLOptions type 7342
XMLBrokr.TXMLParams 7333
XMLBrokr.TXMLParams.AssignStrings 7335
XMLBrokr.TXMLParams.AssignTo 7335
XMLData
    TCustomXSLPageProducer class 7892
    TXSLPageProducer class 7899
XMLDataFile
    TXMLTransformProvider class 7753
XMLDoc 7345
XMLDoc namespace 7345
    Classes 7347
    Functions 7488

Interfaces 7478
Structs, Records, Enums 7494
Types 7498
Variables 7502
XMLDoc.CheckEncoding 7488
XMLDoc.CheckEncoding function 7488
XMLDoc.CreateDOMNode 7488
XMLDoc.CreateDOMNode function 7488
XMLDoc.DetectCharEncoding 7489
XMLDoc.DetectCharEncoding function 7489
XMLDoc.FormatXMLData 7490
XMLDoc.FormatXMLData function 7490
XMLDoc.IXMLDocumentAccess 7478
XMLDoc.IXMLDocumentAccess.DocumentObject 7479
XMLDoc.IXMLDocumentAccess.DOMPersist 7480
XMLDoc.IXMLDocumentAccess.GetDocumentObject 7479
XMLDoc.IXMLDocumentAccess.GetDOMPersist 7479
XMLDoc.IXMLNodeAccess 7480
XMLDoc.IXMLNodeAccess.AddChild 7482
XMLDoc.IXMLNodeAccess.CheckTextNode 7483
XMLDoc.IXMLNodeAccess.ChildNodeClasses 7487
XMLDoc.IXMLNodeAccess.ClearDocumentRef 7483
XMLDoc.IXMLNodeAccess.CreateAttributeNode 7483
XMLDoc.IXMLNodeAccess.CreateChildNode 7483
XMLDoc.IXMLNodeAccess.CreateCollection 7483
XMLDoc.IXMLNodeAccess.DOMElement 7484
XMLDoc.IXMLNodeAccess.FindHostedNode 7484
XMLDoc.IXMLNodeAccess.GetChildNodeClasses 7484
XMLDoc.IXMLNodeAccess.GetHostNode 7484
XMLDoc.IXMLNodeAccess.GetNodeObject 7485
XMLDoc.IXMLNodeAccess.HasChildNode 7485
XMLDoc.IXMLNodeAccess.HostNode 7487
XMLDoc.IXMLNodeAccess.InternalAddChild 7485
XMLDoc.IXMLNodeAccess.NestingLevel 7486
XMLDoc.IXMLNodeAccess.RegisterChildNode 7486
XMLDoc.IXMLNodeAccess.RegisterChildNodes 7486
XMLDoc.IXMLNodeAccess.SetCollection 7486
XMLDoc.IXMLNodeAccess.SetParentNode 7487
XMLDoc.LoadXMLData 7490
XMLDoc.LoadXMLData function 7490
XMLDoc.LoadXMLDocument 7491
XMLDoc.LoadXMLDocument function 7491
XMLDoc.NewXMLDocument 7491
XMLDoc.NewXMLDocument function 7491
XMLDoc.TNodeChange 7494
XMLDoc.TNodeChange enumeration 7494
XMLDoc.TNodeChangeEvent 7498
XMLDoc.TNodeChangeEvent type 7498
XMLDoc.TNodeClassArray 7499
XMLDoc.TNodeClassArray type 7499
XMLDoc.TNodeClassInfo 7495
XMLDoc.TNodeClassInfo record 7495
XMLDoc.TNodeListNotification 7499
XMLDoc.TNodeListNotification type 7499
XMLDoc.TNodeListOperation 7495
XMLDoc.TNodeListOperation enumeration 7495
XMLDoc.TXMLDocument 7347
XMLDoc.TXMLDocument._AddRef 7354
XMLDoc.TXMLDocument._Release 7354
XMLDoc.TXMLDocument.Active 7381
XMLDoc.TXMLDocument.AddChild 7355
XMLDoc.TXMLDocument.AfterClose 7393
XMLDoc.TXMLDocument.AfterConstruction 7355
XMLDoc.TXMLDocument.AfterNodeChange 7393
XMLDoc.TXMLDocument.AfterOpen 7394
XMLDoc.TXMLDocument.AssignParseOptions 7356
XMLDoc.TXMLDocument.AsyncLoadState 7382
XMLDoc.TXMLDocument.BeforeClose 7394
XMLDoc.TXMLDocument.BeforeNodeChange 7394
XMLDoc.TXMLDocument.BeforeOpen 7395
XMLDoc.TXMLDocument.CheckActive 7356
XMLDoc.TXMLDocument.CheckAutoSave 7356
XMLDoc.TXMLDocument.CheckDOM 7356
XMLDoc.TXMLDocument.ChildNodes 7383
XMLDoc.TXMLDocument.Create 7351
XMLDoc.TXMLDocument.CreateElement 7357
XMLDoc.TXMLDocument.CreateNode 7357
XMLDoc.TXMLDocument.DefineProperties 7358

XMLDoc.TXMLDocument.Destroy 7352
XMLDoc.TXMLDocument.DoAfterClose 7359
XMLDoc.TXMLDocument.DoAfterOpen 7359
XMLDoc.TXMLDocument.DoBeforeClose 7359
XMLDoc.TXMLDocument.DoBeforeOpen 7360
XMLDoc.TXMLDocument.DocBindingInfo 7383
XMLDoc.TXMLDocument.DocSource 7383
XMLDoc.TXMLDocument.DocumentElement 7383
XMLDoc.TXMLDocument.DOMDocument 7384
XMLDoc.TXMLDocument.DOMImplementation 7385
XMLDoc.TXMLDocument.DOMParseOptions 7385
XMLDoc.TXMLDocument.DOMPersist 7386
XMLDoc.TXMLDocument.DOMVendor 7386
XMLDoc.TXMLDocument.DoNodeChange 7360
XMLDoc.TXMLDocument.Encoding 7386
XMLDoc.TXMLDocument.FileName 7387
XMLDoc.TXMLDocument.GeneratePrefix 7360
XMLDoc.TXMLDocument.GetActive 7360
XMLDoc.TXMLDocument.GetAsyncLoadState 7361
XMLDoc.TXMLDocument.GetChildNodeClass 7361
XMLDoc.TXMLDocument.GetChildNodes 7361
XMLDoc.TXMLDocument.GetDocBinding 7362
XMLDoc.TXMLDocument.GetDocumentElement 7362
XMLDoc.TXMLDocument.GetDocumentNode 7362
XMLDoc.TXMLDocument.GetDocumentObject 7363
XMLDoc.TXMLDocument.GetDOMDocument 7363
XMLDoc.TXMLDocument.GetDOMPersist 7363
XMLDoc.TXMLDocument.GetEncoding 7363
XMLDoc.TXMLDocument.GetFileName 7364
XMLDoc.TXMLDocument.GetModified 7364
XMLDoc.TXMLDocument.GetNodeIndentStr 7364
XMLDoc.TXMLDocument.GetOptions 7365
XMLDoc.TXMLDocument.GetParseOptions 7365
XMLDoc.TXMLDocument.GetPrologNode 7365
XMLDoc.TXMLDocument.GetPrologValue 7366
XMLDoc.TXMLDocument.GetSchemaRef 7366
XMLDoc.TXMLDocument.GetStandAlone 7366
XMLDoc.TXMLDocument.GetVersion 7366
XMLDoc.TXMLDocument.GetXML 7367
XMLDoc.TXMLDocument.InternalSetPrologValue 7367
XMLDoc.TXMLDocument.IsEmptyDoc 7367
XMLDoc.TXMLDocument.LoadData 7368
XMLDoc.TXMLDocument.Loaded 7368
XMLDoc.TXMLDocument.LoadFromFile 7368
XMLDoc.TXMLDocument.LoadFromStream 7369
XMLDoc.TXMLDocument.LoadFromXML 7370
XMLDoc.TXMLDocument.Modified 7387
XMLDoc.TXMLDocument.NewInstance 7370
XMLDoc.TXMLDocument.Node 7388
XMLDoc.TXMLDocument.NodeIndentStr 7388
XMLDoc.TXMLDocument.NSPrefixBase 7389
XMLDoc.TXMLDocument.OnAsyncLoad 7395
XMLDoc.TXMLDocument.Options 7389
XMLDoc.TXMLDocument.ParseOptions 7390
XMLDoc.TXMLDocument.PrefixID 7391
XMLDoc.TXMLDocument.ReadDOMVendor 7371
XMLDoc.TXMLDocument.Refresh 7371
XMLDoc.TXMLDocument.RegisterDocBinding 7372
XMLDoc.TXMLDocument.ReleaseDoc 7372
XMLDoc.TXMLDocument.Resync 7372
XMLDoc.TXMLDocument.SaveToFile 7373
XMLDoc.TXMLDocument.SaveToStream 7373
XMLDoc.TXMLDocument.SaveToUTF8String 7374
XMLDoc.TXMLDocument.SaveToXML 7374
XMLDoc.TXMLDocument.SaveToXMLStrings 7375
XMLDoc.TXMLDocument.SchemaRef 7391
XMLDoc.TXMLDocument.SetActive 7375
XMLDoc.TXMLDocument.SetDocumentElement 7375
XMLDoc.TXMLDocument.SetDOMDocument 7376
XMLDoc.TXMLDocument.SetEncoding 7376
XMLDoc.TXMLDocument.SetFileName 7376
XMLDoc.TXMLDocument.SetModified 7377
XMLDoc.TXMLDocument.SetNodeIndentStr 7377
XMLDoc.TXMLDocument.SetOnAsyncLoad 7377
XMLDoc.TXMLDocument.SetOptions 7378
XMLDoc.TXMLDocument.SetParseOptions 7378
XMLDoc.TXMLDocument.SetPrologValue 7378
XMLDoc.TXMLDocument.SetStandAlone 7378

XMLDoc.TXMLDocument.SetVersion 7379
XMLDoc.TXMLDocument.SetXML 7379
XMLDoc.TXMLDocument.SetXMLStrings 7379
XMLDoc.TXMLDocument.StandAlone 7391
XMLDoc.TXMLDocument.Version 7391
XMLDoc.TXMLDocument.WriteDOMVendor 7380
XMLDoc.TXMLDocument.XML 7392
XMLDoc.TXMLDocument.XMLStringsChanging 7380
XMLDoc.TXMLDocumentSource 7496
XMLDoc.TXMLDocumentSource enumeration 7496
XMLDoc.TXMLNode 7396
XMLDoc.TXMLNode._AddRef 7403
XMLDoc.TXMLNode._Release 7404
XMLDoc.TXMLNode.AddChild 7404
XMLDoc.TXMLNode.AddHostedNode 7405
XMLDoc.TXMLNode.AttributeListNotify 7406
XMLDoc.TXMLNode.AttributeNodes 7438
XMLDoc.TXMLNode.CheckNotHosted 7406
XMLDoc.TXMLNode.CheckReadOnly 7407
XMLDoc.TXMLNode.CheckTextNode 7407
XMLDoc.TXMLNode.ChildListNotify 7408
XMLDoc.TXMLNode.ChildNodeClasses 7438
XMLDoc.TXMLNode.ChildNodes 7439
XMLDoc.TXMLNode.ClearDocumentRef 7408
XMLDoc.TXMLNode.CloneNode 7409
XMLDoc.TXMLNode.Collection 7439
XMLDoc.TXMLNode.Create 7400
XMLDoc.TXMLNode.CreateAttributeList 7409
XMLDoc.TXMLNode.CreateAttributeNode 7410
XMLDoc.TXMLNode.CreateChildList 7410
XMLDoc.TXMLNode.CreateChildNode 7410
XMLDoc.TXMLNode.CreateCollection 7411
XMLDoc.TXMLNode.CreateHosted 7401
XMLDoc.TXMLNode.DeclareNamespace 7412
XMLDoc.TXMLNode.Destroy 7401
XMLDoc.TXMLNode.DOMElement 7412
XMLDoc.TXMLNode.DOMNode 7440
XMLDoc.TXMLNode.DoNodeChange 7413
XMLDoc.TXMLNode.FindHostedNode 7413
XMLDoc.TXMLNode.FindNamespaceDecl 7414
XMLDoc.TXMLNode.FindNamespaceURI 7414
XMLDoc.TXMLNode.GetAttribute 7415
XMLDoc.TXMLNode.GetAttributeNodes 7415
XMLDoc.TXMLNode.GetAttributeNS 7416
XMLDoc.TXMLNode.GetChildNodeClasses 7416
XMLDoc.TXMLNode.GetChildNodes 7417
XMLDoc.TXMLNode.GetChildValue 7417
XMLDoc.TXMLNode.GetCollection 7418
XMLDoc.TXMLNode.GetDOMNode 7418
XMLDoc.TXMLNode.GetHasChildNodes 7418
XMLDoc.TXMLNode.GetHostedNodes 7419
XMLDoc.TXMLNode.GetHostNode 7419
XMLDoc.TXMLNode.GetIsTextElement 7419
XMLDoc.TXMLNode.GetLocalName 7420
XMLDoc.TXMLNode.GetNamespaceURI 7421
XMLDoc.TXMLNode.GetNodeName 7421
XMLDoc.TXMLNode.GetNodeObject 7422
XMLDoc.TXMLNode.GetNodeType 7422
XMLDoc.TXMLNode.GetNodeValue 7423
XMLDoc.TXMLNode.GetOwnerDocument 7423
XMLDoc.TXMLNode.GetParentNode 7424
XMLDoc.TXMLNode.GetPrefix 7424
XMLDoc.TXMLNode.GetPrefixedName 7425
XMLDoc.TXMLNode.GetReadOnly 7425
XMLDoc.TXMLNode.GetText 7426
XMLDoc.TXMLNode.GetXML 7426
XMLDoc.TXMLNode.HasAttribute 7427
XMLDoc.TXMLNode.HasChildNode 7427
XMLDoc.TXMLNode.HostedNodes 7441
XMLDoc.TXMLNode.HostNode 7441
XMLDoc.TXMLNode.InternalAddChild 7428
XMLDoc.TXMLNode.NestingLevel 7428
XMLDoc.TXMLNode.NextSibling 7429
XMLDoc.TXMLNode.Normalize 7429
XMLDoc.TXMLNode.OnHostAttrNotify 7443
XMLDoc.TXMLNode.OnHostChildNotify 7444
XMLDoc.TXMLNode.OwnerDocument 7442
XMLDoc.TXMLNode.ParentNode 7443

XMLDoc.TXMLNode.PreviousSibling 7430
XMLDoc.TXMLNode.RegisterChildNode 7430
XMLDoc.TXMLNode.RegisterChildNodes 7431
XMLDoc.TXMLNode.RemoveHostedNode 7431
XMLDoc.TXMLNode.Resync 7432
XMLDoc.TXMLNode.SetAttribute 7432
XMLDoc.TXMLNode.SetAttributeNodes 7432
XMLDoc.TXMLNode.SetAttributeNS 7433
XMLDoc.TXMLNode.SetChildNodes 7433
XMLDoc.TXMLNode.SetChildValue 7434
XMLDoc.TXMLNode.SetCollection 7434
XMLDoc.TXMLNode.SetNodeValue 7435
XMLDoc.TXMLNode.SetParentNode 7435
XMLDoc.TXMLNode.SetReadOnly 7436
XMLDoc.TXMLNode.SetText 7436
XMLDoc.TXMLNode.TransformNode 7437
XMLDoc.TXMLNodeArray 7500
XMLDoc.TXMLNodeArray type 7500
XMLDoc.TXMLNodeClass 7501
XMLDoc.TXMLNodeClass type 7501
XMLDoc.TXMLNodeCollection 7444
XMLDoc.TXMLNodeCollection.AddItem 7451
XMLDoc.TXMLNodeCollection.AfterConstruction 7451
XMLDoc.TXMLNodeCollection.ChildListNotify 7452
XMLDoc.TXMLNodeCollection.Clear 7452
XMLDoc.TXMLNodeCollection.Count 7457
XMLDoc.TXMLNodeCollection.CreateItemList 7453
XMLDoc.TXMLNodeCollection.Delete 7453
XMLDoc.TXMLNodeCollection.GetCount 7454
XMLDoc.TXMLNodeCollection.GetList 7454
XMLDoc.TXMLNodeCollection.GetNode 7454
XMLDoc.TXMLNodeCollection.InsertInCollection 7455
XMLDoc.TXMLNodeCollection.IsCollectionItem 7455
XMLDoc.TXMLNodeCollection.ItemInterface 7458
XMLDoc.TXMLNodeCollection.ItemNS 7458
XMLDoc.TXMLNodeCollection.ItemTag 7459
XMLDoc.TXMLNodeCollection.List 7459
XMLDoc.TXMLNodeCollection.Remove 7456
XMLDoc.TXMLNodeCollection.SetChildNodes 7456
XMLDoc.TXMLNodeCollection.UpdateCollectionList 7457
XMLDoc.TXMLNodeCollectionClass 7501
XMLDoc.TXMLNodeCollectionClass type 7501
XMLDoc.TXMLNodeList 7460
XMLDoc.TXMLNodeList.Add 7464
XMLDoc.TXMLNodeList.BeginUpdate 7465
XMLDoc.TXMLNodeList.Clear 7465
XMLDoc.TXMLNodeList.Count 7474
XMLDoc.TXMLNodeList.Create 7462
XMLDoc.TXMLNodeList.DefaultNamespaceURI 7475
XMLDoc.TXMLNodeList.Delete 7466
XMLDoc.TXMLNodeList.Destroy 7463
XMLDoc.TXMLNodeList.DoNotify 7466
XMLDoc.TXMLNodeList.EndUpdate 7467
XMLDoc.TXMLNodeList.FindNode 7468
XMLDoc.TXMLNodeList.FindSibling 7468
XMLDoc.TXMLNodeList.First 7469
XMLDoc.TXMLNodeList.Get 7469
XMLDoc.TXMLNodeList.GetCount 7470
XMLDoc.TXMLNodeList.GetNode 7470
XMLDoc.TXMLNodeList.GetUpdateCount 7471
XMLDoc.TXMLNodeList.IndexOf 7471
XMLDoc.TXMLNodeList.Insert 7472
XMLDoc.TXMLNodeList.InternalInsert 7472
XMLDoc.TXMLNodeList.Last 7473
XMLDoc.TXMLNodeList.List 7475
XMLDoc.TXMLNodeList.NotificationProc 7476
XMLDoc.TXMLNodeList.Owner 7475
XMLDoc.TXMLNodeList.Remove 7473
XMLDoc.TXMLNodeList.ReplaceNode 7473
XMLDoc.TXMLNodeList.UpdateCount 7476
XMLDoc.TXMLPrologItem 7497
XMLDoc.TXMLPrologItem enumeration 7497
XMLDoc.UseXSDBooleanStrings 7502
XMLDoc.UseXSDBooleanStrings variable 7502
XMLDoc.XMLDocError 7492
XMLDoc.XMLDocError function 7492
XMLDoc.XMLStringToUnicodeString 7492
XMLDoc.XMLStringToUnicodeString function 7492

XMLDoc.XMLStringToWideString 7492
XMLDoc.XMLStringToWideString function 7492
xmldom 7503
xmldom namespace 7503
Classes 7506
Constants 7624
Functions 7612
Interfaces 7521
Types 7619
Variables 7622
xmldom.ATTRIBUTE_NODE 7624
xmldom.ATTRIBUTE_NODE constant 7624
xmldom.CDATA_SECTION_NODE 7625
xmldom.CDATA_SECTION_NODE constant 7625
xmldom.COMMENT_NODE 7625
xmldom.COMMENT_NODE constant 7625
xmldom.DefaultDOMVendor 7622
xmldom.DefaultDOMVendor variable 7622
xmldom.DOCUMENT_FRAGMENT_NODE 7625
xmldom.DOCUMENT_FRAGMENT_NODE constant 7625
xmldom.DOCUMENT_NODE 7626
xmldom.DOCUMENT_NODE constant 7626
xmldom.DOCUMENT_TYPE_NODE 7626
xmldom.DOCUMENT_TYPE_NODE constant 7626
xmldom.DOMException 7506
xmldom.DOMException.code 7508
xmldom.DOMNodeType 7619
xmldom.DOMNodeType type 7619
xmldom.DOMString 7619
xmldom.DOMString type 7619
xmldom.DOMSTRING_SIZE_ERR 7626
xmldom.DOMSTRING_SIZE_ERR constant 7626
xmldom.DOMTimeStamp 7620
xmldom.DOMTimeStamp type 7620
xmldom.DOMVendorNotSupported 7612
xmldom.DOMVendorNotSupported function 7612
xmldom.DOMVendors 7622
xmldom.DOMVendors variable 7622
xmldom.DOMWrapperVersion 7627
xmldom.DOMWrapperVersion constant 7627
xmldom.EDOMParseError 7508
xmldom.EDOMParseError.Create 7510
xmldom.EDOMParseError.ErrorCode 7511
xmldom.EDOMParseError.FilePos 7511
xmldom.EDOMParseError.Line 7512
xmldom.EDOMParseError.LinePos 7512
xmldom.EDOMParseError.ParseError 7512
xmldom.EDOMParseError.Reason 7513
xmldom.EDOMParseError.SrcText 7513
xmldom.EDOMParseError.URL 7513
xmldom.ELEMENT_NODE 7627
xmldom.ELEMENT_NODE constant 7627
xmldom.ENTITY_NODE 7627
xmldom.ENTITY_NODE constant 7627
xmldom.ENTITY_REFERENCE_NODE 7628
xmldom.ENTITY_REFERENCE_NODE constant 7628
xmldom.ExtractLocalName 7612
xmldom.ExtractLocalName function 7612
xmldom.ExtractPrefix 7613
xmldom.ExtractPrefix function 7613
xmldom.GetDOM 7614
xmldom.GetDOM function 7614
xmldom.GetDOMNodeEx 7614
xmldom.GetDOMNodeEx function 7614
xmldom.GetDOMVendor 7615
xmldom.GetDOMVendor function 7615
xmldom.HIERARCHY_REQUEST_ERR 7628
xmldom.HIERARCHY_REQUEST_ERR constant 7628
xmldom.IDOMAttr 7521
xmldom.IDOMAttr.get_name 7523
xmldom.IDOMAttr.get_ownerElement 7523
xmldom.IDOMAttr.get_specified 7523
xmldom.IDOMAttr.get_value 7524
xmldom.IDOMAttr.name 7524
xmldom.IDOMAttr.ownerElement 7524
xmldom.IDOMAttr.set_value 7524
xmldom.IDOMAttr.specified 7525
xmldom.IDOMAttr.value 7525

xmldom.IDOMCDATASection 7525
xmldom.IDOMCharacterData 7526
xmldom.IDOMCharacterData.appendData 7527
xmldom.IDOMCharacterData.data 7529
xmldom.IDOMCharacterData.deleteData 7528
xmldom.IDOMCharacterData.get_data 7528
xmldom.IDOMCharacterData.get_length 7528
xmldom.IDOMCharacterData.insertData 7528
xmldom.IDOMCharacterData.length 7529
xmldom.IDOMCharacterData.replaceData 7528
xmldom.IDOMCharacterData.set_data 7529
xmldom.IDOMCharacterData.substringData 7529
xmldom.IDOMComment 7530
xmldom.IDOMDocument 7530
xmldom.IDOMDocument.createAttribute 7532
xmldom.IDOMDocument.createAttributeNS 7533
xmldom.IDOMDocument.createCDATASection 7533
xmldom.IDOMDocument.createComment 7533
xmldom.IDOMDocument.createDocumentFragment 7533
xmldom.IDOMDocument.createElement 7533
xmldom.IDOMDocument.createElementNS 7534
xmldom.IDOMDocument.createEntityReference 7534
xmldom.IDOMDocument.createProcessingInstruction 7534
xmldom.IDOMDocument.createTextNode 7534
xmldom.IDOMDocument.doctype 7537
xmldom.IDOMDocument.documentElement 7537
xmldom.IDOMDocument.domImplementation 7537
xmldom.IDOMDocument.get_doctype 7535
xmldom.IDOMDocument.get_documentElement 7535
xmldom.IDOMDocument.get_domImplementation 7535
xmldom.IDOMDocument.getElementById 7535
xmldom.IDOMDocument.getElementsByTagName 7536
xmldom.IDOMDocument.getElementsByTagNameNS 7536
xmldom.IDOMDocument.importNode 7536
xmldom.IDOMDocument.set_documentElement 7536
xmldom.IDOMDocumentFragment 7537
xmldom.IDOMDocumentType 7538
xmldom.IDOMDocumentType.entities 7541
xmldom.IDOMDocumentType.get_entities 7540
xmldom.IDOMDocumentType.get_internalSubset 7540
xmldom.IDOMDocumentType.get_name 7540
xmldom.IDOMDocumentType.get_notations 7540
xmldom.IDOMDocumentType.get_publicId 7540
xmldom.IDOMDocumentType.get_systemId 7541
xmldom.IDOMDocumentType.internalSubset 7541
xmldom.IDOMDocumentType.name 7542
xmldom.IDOMDocumentType.notations 7542
xmldom.IDOMDocumentType.publicId 7542
xmldom.IDOMDocumentType.systemId 7542
xmldom.IDOMElement 7542
xmldom.IDOMElement.get_tagName 7545
xmldom.IDOMElement.getAttribute 7545
xmldom.IDOMElement.getAttributeNode 7545
xmldom.IDOMElement.getAttributeNodeNS 7545
xmldom.IDOMElement.getAttributeNS 7545
xmldom.IDOMElement.getElementsByTagName 7546
xmldom.IDOMElement.getElementsByTagNameNS 7546
xmldom.IDOMElement.hasAttribute 7546
xmldom.IDOMElement.hasAttributeNS 7546
xmldom.IDOMElement.removeAttribute 7547
xmldom.IDOMElement.removeAttributeNode 7547
xmldom.IDOMElement.removeAttributeNS 7547
xmldom.IDOMElement.setAttribute 7547
xmldom.IDOMElement.setAttributeNode 7547
xmldom.IDOMElement.setAttributeNodeNS 7548
xmldom.IDOMElement.setAttributeNS 7548
xmldom.IDOMElement.tagName 7548
xmldom.IDOMEntity 7549
xmldom.IDOMEntity.get_notationName 7550
xmldom.IDOMEntity.get_publicId 7550
xmldom.IDOMEntity.get_systemId 7551
xmldom.IDOMEntity.notationName 7551
xmldom.IDOMEntity.publicId 7551
xmldom.IDOMEntity.systemId 7551
xmldom.IDOMEntityReference 7552
xmldom.IDOMImplementation 7552
xmldom.IDOMImplementation.createDocument 7553
xmldom.IDOMImplementation.createDocumentType 7553

xmldom.IDOMImplementation.hasFeature 7553
xmldom.IDOMNamedNodeMap 7554
xmldom.IDOMNamedNodeMap.get_item 7555
xmldom.IDOMNamedNodeMap.get_length 7555
xmldom.IDOMNamedNodeMap.getNamedItem 7555
xmldom.IDOMNamedNodeMap.getNamedItemNS 7555
xmldom.IDOMNamedNodeMap.item 7557
xmldom.IDOMNamedNodeMap.length 7557
xmldom.IDOMNamedNodeMap.removeNamedItem 7556
xmldom.IDOMNamedNodeMap.removeNamedItemNS 7556
xmldom.IDOMNamedNodeMap.setNamedItem 7556
xmldom.IDOMNamedNodeMap.setNamedItemNS 7556
xmldom.IDOMNode 7557
xmldom.IDOMNode.appendChild 7559
xmldom.IDOMNode.attributes 7568
xmldom.IDOMNode.childNodes 7568
xmldom.IDOMNode.cloneNode 7560
xmldom.IDOMNode.firstChild 7569
xmldom.IDOMNode.get_attributes 7561
xmldom.IDOMNode.get_childNodes 7561
xmldom.IDOMNode.get_firstChild 7561
xmldom.IDOMNode.get_lastChild 7561
xmldom.IDOMNode.get_localName 7562
xmldom.IDOMNode.get_namespaceURI 7562
xmldom.IDOMNode.get_nextSibling 7562
xmldom.IDOMNode.get_nodeName 7562
xmldom.IDOMNode.get_nodeType 7563
xmldom.IDOMNode.get_nodeValue 7563
xmldom.IDOMNode.get_ownerDocument 7563
xmldom.IDOMNode.get_parentNode 7563
xmldom.IDOMNode.get_prefix 7564
xmldom.IDOMNode.get_previousSibling 7564
xmldom.IDOMNode.hasChildNodes 7564
xmldom.IDOMNode.insertBefore 7565
xmldom.IDOMNode.lastChild 7569
xmldom.IDOMNode.localName 7570
xmldom.IDOMNode.namespaceURI 7571
xmldom.IDOMNode.nextSibling 7571
xmldom.IDOMNode.nodeName 7572
xmldom.IDOMNode.nodeType 7572
xmldom.IDOMNode.nodeValue 7574
xmldom.IDOMNode.normalize 7565
xmldom.IDOMNode.ownerDocument 7575
xmldom.IDOMNode.parentNode 7575
xmldom.IDOMNode.prefix 7576
xmldom.IDOMNode.previousSibling 7576
xmldom.IDOMNode.removeChild 7566
xmldom.IDOMNode.replaceChild 7566
xmldom.IDOMNode.set_nodeValue 7567
xmldom.IDOMNode.supports 7567
xmldom.IDOMNodeEx 7577
xmldom.IDOMNodeEx.get_text 7579
xmldom.IDOMNodeEx.get_xml 7579
xmldom.IDOMNodeEx.set_text 7579
xmldom.IDOMNodeEx.text 7580
xmldom.IDOMNodeEx.transformNode 7579, 7580
xmldom.IDOMNodeEx.xml 7580
xmldom.IDOMNodeList 7581
xmldom.IDOMNodeList.get_item 7581
xmldom.IDOMNodeList.get_length 7582
xmldom.IDOMNodeList.item 7582
xmldom.IDOMNodeList.length 7582
xmldom.IDOMNodeSelect 7583
xmldom.IDOMNodeSelect.selectNode 7583
xmldom.IDOMNodeSelect.selectNodes 7584
xmldom.IDOMNotation 7584
xmldom.IDOMNotation.get_publicId 7585
xmldom.IDOMNotation.get_systemId 7586
xmldom.IDOMNotation.publicId 7586
xmldom.IDOMNotation.systemId 7586
xmldom.IDOMParseError 7586
xmldom.IDOMParseError.errorCode 7589
xmldom.IDOMParseError.filePos 7590
xmldom.IDOMParseError.get_errorCode 7588
xmldom.IDOMParseError.get_filepos 7588
xmldom.IDOMParseError.get_line 7588
xmldom.IDOMParseError.get_linepos 7588
xmldom.IDOMParseError.get_reason 7588

xmldom.IDOMParseError.get_srcText 7589
xmldom.IDOMParseError.get_url 7589
xmldom.IDOMParseError.line 7590
xmldom.IDOMParseError.linePos 7590
xmldom.IDOMParseError.reason 7590
xmldom.IDOMParseError.srcText 7590
xmldom.IDOMParseError.url 7591
xmldom.IDOMParseOptions 7591
xmldom.IDOMParseOptions.async 7594
xmldom.IDOMParseOptions.get_async 7592
xmldom.IDOMParseOptions.get_preserveWhiteSpace 7592
xmldom.IDOMParseOptions.get_resolveExternals 7592
xmldom.IDOMParseOptions.get_validate 7593
xmldom.IDOMParseOptions.preserveWhiteSpace 7594
xmldom.IDOMParseOptions.resolveExternals 7594
xmldom.IDOMParseOptions.set_async 7593
xmldom.IDOMParseOptions.set_preserveWhiteSpace 7593
xmldom.IDOMParseOptions.set_resolveExternals 7593
xmldom.IDOMParseOptions.set_validate 7593
xmldom.IDOMParseOptions.validate 7595
xmldom.IDOMPersist 7595
xmldom.IDOMPersist.asyncLoadState 7596
xmldom.IDOMPersist.get_xml 7596
xmldom.IDOMPersist.load 7596
xmldom.IDOMPersist.loadFromStream 7597
xmldom.IDOMPersist.loadxml 7597
xmldom.IDOMPersist.save 7597
xmldom.IDOMPersist.saveToStream 7598
xmldom.IDOMPersist.set_OnAsyncLoad 7598
xmldom.IDOMPersist.xml 7599
xmldom.IDOMProcessingInstruction 7599
xmldom.IDOMProcessingInstruction.data 7601
xmldom.IDOMProcessingInstruction.get_data 7601
xmldom.IDOMProcessingInstruction.get_target 7601
xmldom.IDOMProcessingInstruction.set_data 7601
xmldom.IDOMProcessingInstruction.target 7602
xmldom.IDOMText 7602
xmldom.IDOMText.splitText 7604
xmldom.IDOMXMLProlog 7604
xmldom.IDOMXMLProlog.Encoding 7606
xmldom.IDOMXMLProlog.get_Encoding 7605
xmldom.IDOMXMLProlog.get_Standalone 7605
xmldom.IDOMXMLProlog.get_Version 7605
xmldom.IDOMXMLProlog.set_Encoding 7605
xmldom.IDOMXMLProlog.set_Standalone 7606
xmldom.IDOMXMLProlog.set_Version 7606
xmldom.IDOMXMLProlog.Standalone 7607
xmldom.IDOMXMLProlog.Version 7607
xmldom.IDOMXSLProcessor 7607
xmldom.IDOMXSLProcessor.Get_input 7608
xmldom.IDOMXSLProcessor.Get_output 7608
xmldom.IDOMXSLProcessor.Get_stylesheet 7609
xmldom.IDOMXSLProcessor.input 7610
xmldom.IDOMXSLProcessor.output 7610
xmldom.IDOMXSLProcessor.reset 7609
xmldom.IDOMXSLProcessor.Set_input 7609
xmldom.IDOMXSLProcessor.Set_output 7609
xmldom.IDOMXSLProcessor.setParameter 7609
xmldom.IDOMXSLProcessor.stylesheet 7611
xmldom.IDOMXSLProcessor.transform 7610
xmldom.INDEX_SIZE_ERR 7628
xmldom.INDEX_SIZE_ERR constant 7628
xmldom.INUSE_ATTRIBUTE_ERR 7629
xmldom.INUSE_ATTRIBUTE_ERR constant 7629
xmldom.INVALID_ACCESS_ERR 7629
xmldom.INVALID_ACCESS_ERR constant 7629
xmldom.INVALID_CHARACTER_ERR 7629
xmldom.INVALID_CHARACTER_ERR constant 7629
xmldom.INVALID_MODIFICATION_ERR 7630
xmldom.INVALID_MODIFICATION_ERR constant 7630
xmldom.INVALID_STATE_ERR 7630
xmldom.INVALID_STATE_ERR constant 7630
xmldom.IsPrefixed 7615
xmldom.IsPrefixed function 7615
xmldom.MakeNodeName 7616
xmldom.MakeNodeName function 7616
xmldom.NAMESPACE_ERR 7630
xmldom.NAMESPACE_ERR constant 7630

xmldom.NO_DATA_ALLOWED_ERR 7631
xmldom.NO_DATA_ALLOWED_ERR constant 7631
xmldom.NO_MODIFICATION_ALLOWED_ERR 7631
xmldom.NO_MODIFICATION_ALLOWED_ERR constant 7631
xmldom.NodeMatches 7616
xmldom.NodeMatches function 7616
xmldom.NOT_FOUND_ERR 7631
xmldom.NOT_FOUND_ERR constant 7631
xmldom.NOT_SUPPORTED_ERR 7632
xmldom.NOT_SUPPORTED_ERR constant 7632
xmldom.NOTATION_NODE 7632
xmldom.NOTATION_NODE constant 7632
xmldom.NSDelim 7632
xmldom.NSDelim constant 7632
xmldom.PROCESSING_INSTRUCTION_NODE 7633
xmldom.PROCESSING_INSTRUCTION_NODE constant 7633
xmldom.RegisterDOMVendor 7617
xmldom.RegisterDOMVendor function 7617
xmldom.SameNamespace 7617
xmldom.SameNamespace function 7617
xmldom.SEncoding 7633
xmldom.SEncoding constant 7633
xmldom.SHttp 7633
xmldom.SHttp constant 7633
xmldom.SStandalone 7634
xmldom.SStandalone constant 7634
xmldom.SVersion 7634
xmldom.SVersion constant 7634
xmldom.SXML 7634
xmldom.SXML constant 7634
xmldom.SXMLNamespaceURI 7635
xmldom.SXMLNamespaceURI constant 7635
xmldom.SXMLNS 7635
xmldom.SXMLNS constant 7635
xmldom.SXMLPrefixNamespaceURI 7635
xmldom.SXMLPrefixNamespaceURI constant 7635
xmldom.SYNTAX_ERR 7636
xmldom.SYNTAX_ERR constant 7636
xmldom.TAsyncEventHandler 7620
xmldom.TAsyncEventHandler type 7620
xmldom.TDOMVendor 7514
xmldom.TDOMVendor.Description 7515
xmldom.TDOMVendor.DOMImplementation 7515
xmldom.TDOMVendorArray 7621
xmldom.TDOMVendorArray type 7621
xmldom.TDOMVendorList 7516
xmldom.TDOMVendorList.Add 7517
xmldom.TDOMVendorList.Count 7517
xmldom.TDOMVendorList.Find 7518
xmldom.TDOMVendorList.GetVendors 7518
xmldom.TDOMVendorList.Remove 7519
xmldom.TDOMVendorList.Vendors 7519
xmldom.TEXT_NODE 7636
xmldom.TEXT_NODE constant 7636
xmldom.UnRegisterDOMVendor 7618
xmldom.UnRegisterDOMVendor function 7618
xmldom.WRONG_DOCUMENT_ERR 7636
xmldom.WRONG_DOCUMENT_ERR constant 7636
XMLIntf 7637
XMLIntf namespace 7637
    Classes 7639
    Constants 7723
    Interfaces 7640
    Structs, Records, Enums 7716
    Types 7721
XMLIntf.EXMLDocError 7639
XMLIntf.IXMLDocument 7640
XMLIntf.IXMLDocument.Active 7661
XMLIntf.IXMLDocument.AddChild 7643
XMLIntf.IXMLDocument.AsyncLoadState 7662
XMLIntf.IXMLDocument.ChildNodes 7663
XMLIntf.IXMLDocument.CreateElement 7644
XMLIntf.IXMLDocument.CreateNode 7645
XMLIntf.IXMLDocument.DocumentElement 7663
XMLIntf.IXMLDocument.DOMDocument 7663
XMLIntf.IXMLDocument.Encoding 7664
XMLIntf.IXMLDocument.FileName 7664
XMLIntf.IXMLDocument.GeneratePrefix 7646

XMLIntf.IXMLDocument.GetActive 7646
XMLIntf.IXMLDocument.GetAsyncLoadState 7647
XMLIntf.IXMLDocument.GetChildNodes 7647
XMLIntf.IXMLDocument.GetDocBinding 7647
XMLIntf.IXMLDocument.GetDocumentElement 7648
XMLIntf.IXMLDocument.GetDocumentNode 7648
XMLIntf.IXMLDocument.GetDOMDocument 7648
XMLIntf.IXMLDocument.GetEncoding 7649
XMLIntf.IXMLDocument.GetFileName 7649
XMLIntf.IXMLDocument.GetModified 7649
XMLIntf.IXMLDocument.GetNodeIndentStr 7650
XMLIntf.IXMLDocument.GetOptions 7650
XMLIntf.IXMLDocument.GetParseOptions 7650
XMLIntf.IXMLDocument.GetSchemaRef 7650
XMLIntf.IXMLDocument.GetStandAlone 7651
XMLIntf.IXMLDocument.GetVersion 7651
XMLIntf.IXMLDocument.GetXML 7651
XMLIntf.IXMLDocument.IsEmptyDoc 7652
XMLIntf.IXMLDocument.LoadFromFile 7652
XMLIntf.IXMLDocument.LoadFromStream 7653
XMLIntf.IXMLDocument.LoadFromXML 7653
XMLIntf.IXMLDocument.Modified 7665
XMLIntf.IXMLDocument.Node 7665
XMLIntf.IXMLDocument.NodeIndentStr 7666
XMLIntf.IXMLDocument.Options 7666
XMLIntf.IXMLDocument.ParseOptions 7667
XMLIntf.IXMLDocument.Refresh 7654
XMLIntf.IXMLDocument.RegisterDocBinding 7654
XMLIntf.IXMLDocument.Resync 7655
XMLIntf.IXMLDocument.SaveToFile 7655
XMLIntf.IXMLDocument.SaveToStream 7656
XMLIntf.IXMLDocument.SaveToXML 7656
XMLIntf.IXMLDocument.SchemaRef 7667
XMLIntf.IXMLDocument.SetActive 7657
XMLIntf.IXMLDocument.SetDocumentElement 7657
XMLIntf.IXMLDocument.SetDOMDocument 7658
XMLIntf.IXMLDocument.SetEncoding 7658
XMLIntf.IXMLDocument.SetFileName 7658
XMLIntf.IXMLDocument.SetNodeIndentStr 7659
XMLIntf.IXMLDocument.SetOnAsyncLoad 7659
XMLIntf.IXMLDocument.SetOptions 7659
XMLIntf.IXMLDocument.SetParseOptions 7660
XMLIntf.IXMLDocument.SetStandAlone 7660
XMLIntf.IXMLDocument.SetVersion 7660
XMLIntf.IXMLDocument.SetXML 7661
XMLIntf.IXMLDocument.StandAlone 7668
XMLIntf.IXMLDocument.Version 7668
XMLIntf.IXMLDocument.XML 7668
XMLIntf.IXMLNode 7669
XMLIntf.IXMLNode.AddChild 7672
XMLIntf.IXMLNode.AttributeNodes 7688
XMLIntf.IXMLNode.Attributes 7688
XMLIntf.IXMLNode.ChildNodes 7689
XMLIntf.IXMLNode.ChildValues 7690
XMLIntf.IXMLNode.CloneNode 7673
XMLIntf.IXMLNode.Collection 7690
XMLIntf.IXMLNode.DeclareNamespace 7674
XMLIntf.IXMLNode.DOMNode 7691
XMLIntf.IXMLNode.FindNamespaceDecl 7674
XMLIntf.IXMLNode.FindNamespaceURI 7674
XMLIntf.IXMLNode.GetAttribute 7675
XMLIntf.IXMLNode.GetAttributeNodes 7675
XMLIntf.IXMLNode.GetAttributeNS 7676
XMLIntf.IXMLNode.GetChildNodes 7676
XMLIntf.IXMLNode.GetChildValue 7676
XMLIntf.IXMLNode.GetCollection 7677
XMLIntf.IXMLNode.GetDOMNode 7677
XMLIntf.IXMLNode.GetHasChildNodes 7678
XMLIntf.IXMLNode.GetIsTextElement 7678
XMLIntf.IXMLNode.GetLocalName 7678
XMLIntf.IXMLNode.GetNamespaceURI 7679
XMLIntf.IXMLNode.GetNodeName 7679
XMLIntf.IXMLNode.GetNodeType 7679
XMLIntf.IXMLNode.GetNodeValue 7680
XMLIntf.IXMLNode.GetOwnerDocument 7680
XMLIntf.IXMLNode.GetParentNode 7680
XMLIntf.IXMLNode.GetPrefix 7680
XMLIntf.IXMLNode.GetReadOnly 7681

XMLIntf.IXMLNode.GetText 7681
XMLIntf.IXMLNode.GetXML 7681
XMLIntf.IXMLNode.HasAttribute 7682
XMLIntf.IXMLNode.HasChildNodes 7691
XMLIntf.IXMLNode.IsTextElement 7692
XMLIntf.IXMLNode.LocalName 7692
XMLIntf.IXMLNode.NamespaceURI 7693
XMLIntf.IXMLNode.NextSibling 7682
XMLIntf.IXMLNode.NodeName 7693
XMLIntf.IXMLNode.NodeType 7694
XMLIntf.IXMLNode.NodeValue 7695
XMLIntf.IXMLNode.Normalize 7683
XMLIntf.IXMLNode.OwnerDocument 7695
XMLIntf.IXMLNode.ParentNode 7695
XMLIntf.IXMLNode.Prefix 7696
XMLIntf.IXMLNode.PreviousSibling 7683
XMLIntf.IXMLNode.ReadOnly 7696
XMLIntf.IXMLNode.Resync 7684
XMLIntf.IXMLNode.SetAttribute 7684
XMLIntf.IXMLNode.SetAttributeNS 7684
XMLIntf.IXMLNode.SetChildValue 7685
XMLIntf.IXMLNode.SetNodeValue 7685
XMLIntf.IXMLNode.SetReadOnly 7686
XMLIntf.IXMLNode.SetText 7686
XMLIntf.IXMLNode.Text 7697
XMLIntf.IXMLNode.TransformNode 7687
XMLIntf.IXMLNode.XML 7697
XMLIntf.IXMLNodeCollection 7698
XMLIntf.IXMLNodeCollection.Clear 7700
XMLIntf.IXMLNodeCollection.Count 7702
XMLIntf.IXMLNodeCollection.Delete 7700
XMLIntf.IXMLNodeCollection.GetCount 7701
XMLIntf.IXMLNodeCollection.GetNode 7701
XMLIntf.IXMLNodeCollection.Nodes 7703
XMLIntf.IXMLNodeCollection.Remove 7702
XMLIntf.IXMLNodeList 7703
XMLIntf.IXMLNodeList.Add 7705
XMLIntf.IXMLNodeList.BeginUpdate 7705
XMLIntf.IXMLNodeList.Clear 7706
XMLIntf.IXMLNodeList.Count 7714
XMLIntf.IXMLNodeList.Delete 7707
XMLIntf.IXMLNodeList.EndUpdate 7707
XMLIntf.IXMLNodeList.FindNode 7708
XMLIntf.IXMLNodeList.FindSibling 7709
XMLIntf.IXMLNodeList.First 7709
XMLIntf.IXMLNodeList.Get 7710
XMLIntf.IXMLNodeList.GetCount 7710
XMLIntf.IXMLNodeList.GetNode 7710
XMLIntf.IXMLNodeList.GetUpdateCount 7711
XMLIntf.IXMLNodeList.IndexOf 7711
XMLIntf.IXMLNodeList.Insert 7712
XMLIntf.IXMLNodeList.Last 7712
XMLIntf.IXMLNodeList.Nodes 7714
XMLIntf.IXMLNodeList.Remove 7713
XMLIntf.IXMLNodeList.ReplaceNode 7713
XMLIntf.IXMLNodeList.UpdateCount 7715
XMLIntf.NodeTypeNames 7723
XMLIntf.NodeTypeNames constant 7723
XMLIntf.TNodeType 7716
XMLIntf.TNodeType enumeration 7716
XMLIntf.TParseOption 7718
XMLIntf.TParseOption enumeration 7718
XMLIntf.TParseOptions 7721
XMLIntf.TParseOptions type 7721
XMLIntf.TXMLDocOption 7718
XMLIntf.TXMLDocOption enumeration 7718
XMLIntf.TXMLDocOptions 7722
XMLIntf.TXMLDocOptions type 7722
XMLIntf.TXMLEncodingType 7719
XMLIntf.TXMLEncodingType enumeration 7719
XMLNode
    TXMLData class 7801
XMLPacket1
    TSocketForm class 173
XMLStringsChanging
    TXMLDocument class 7380
Xmlxform 7725
Xmlxform namespace 7725

Classes 7726
Types 7761
Xmlxform.TranslateException 7726
Xmlxform.TRowEvent 7761
Xmlxform.TRowEvent type 7761
Xmlxform.TTranslateEvent 7762
Xmlxform.TTranslateEvent type 7762
Xmlxform.TXMLTransform 7727
Xmlxform.TXMLTransform.AfterEachRow 7737
Xmlxform.TXMLTransform.AfterEachRowSet 7737
Xmlxform.TXMLTransform.BeforeEachRow 7738
Xmlxform.TXMLTransform.BeforeEachRowSet 7738
Xmlxform.TXMLTransform.Data 7731
Xmlxform.TXMLTransform.DoTransform 7729
Xmlxform.TXMLTransform.EmptyDestinationDocument 7732
Xmlxform.TXMLTransform.GetData 7729
Xmlxform.TXMLTransform.GetResultString 7729
Xmlxform.TXMLTransform.OnTranslate 7739
Xmlxform.TXMLTransform.ResultDocument 7733
Xmlxform.TXMLTransform.ResultString 7733
Xmlxform.TXMLTransform.SourceXml 7734
Xmlxform.TXMLTransform.SourceXmlDocument 7734
Xmlxform.TXMLTransform.SourceXmlFile 7735
Xmlxform.TXMLTransform.Transform 7729
Xmlxform.TXMLTransform.TransformationDocument 7735
Xmlxform.TXMLTransform.TransformationFile 7736
Xmlxform.TXMLTransform.TransformXML 7730
Xmlxform.TXMLTransform.Translate 7730
Xmlxform.TXMLTransformClient 7739
Xmlxform.TXMLTransformClient.ApplyUpdates 7742
Xmlxform.TXMLTransformClient.Create 7741
Xmlxform.TXMLTransformClient.Destroy 7741
Xmlxform.TXMLTransformClient.GetDataAsXml 7742
Xmlxform.TXMLTransformClient.ProviderName 7745
Xmlxform.TXMLTransformClient.RemoteServer 7745
Xmlxform.TXMLTransformClient.SetParams 7743
Xmlxform.TXMLTransformClient.SetupAppServer 7744
Xmlxform.TXMLTransformClient.TransformApplyUpdates 7746
Xmlxform.TXMLTransformClient.TransformGetData 7746
Xmlxform.TXMLTransformClient.TransformSetParams 7747
Xmlxform.TXMLTransformProvider 7747
Xmlxform.TXMLTransformProvider.AfterApplyUpdates 7754
Xmlxform.TXMLTransformProvider.AfterGetRecords 7755
Xmlxform.TXMLTransformProvider.AfterRowRequest 7756
Xmlxform.TXMLTransformProvider.BeforeApplyUpdates 7756
Xmlxform.TXMLTransformProvider.BeforeGetRecords 7757
Xmlxform.TXMLTransformProvider.BeforeRowRequest 7758
Xmlxform.TXMLTransformProvider.CacheData 7751
Xmlxform.TXMLTransformProvider.Create 7749
Xmlxform.TXMLTransformProvider.Destroy 7749
Xmlxform.TXMLTransformProvider.InternalApplyUpdates 7750
Xmlxform.TXMLTransformProvider.InternalGetRecords 7750
Xmlxform.TXMLTransformProvider.OnDataRequest 7759
Xmlxform.TXMLTransformProvider.TransformRead 7752
Xmlxform.TXMLTransformProvider.TransformWrite 7753
Xmlxform.TXMLTransformProvider.XMLDataFile 7753
xoQuote enumeration member 7340
XPActnCtrls 7763
XPActnCtrls namespace 7763
Classes 7764
XPActnCtrls.TXPStyleAddRemoveItem 7764
XPActnCtrls.TXPStyleAddRemoveItem.CalcBounds 7765
XPActnCtrls.TXPStyleAddRemoveItem.DrawBackground 7765
XPActnCtrls.TXPStyleAddRemoveItem.DrawGlyph 7766
XPActnCtrls.TXPStyleAddRemoveItem.DrawSeparator 7766
XPActnCtrls.TXPStyleAddRemoveItem.DrawText 7766
XPActnCtrls.TXPStyleAddRemoveItem.DrawUnusedEdges 7766
XPActnCtrls.TXPStyleButton 7767
XPActnCtrls.TXPStyleButton.DrawBackground 7767
XPActnCtrls.TXPStyleButton.DrawFrame 7768
XPActnCtrls.TXPStyleButton.DrawGlyph 7768
XPActnCtrls.TXPStyleButton.DrawText 7768
XPActnCtrls.TXPStyleCustomizePopup 7769
XPActnCtrls.TXPStyleCustomizePopup.Create 7769
XPActnCtrls.TXPStyleCustomizePopup.GetAddRemoveItemClass 7770
XPActnCtrls.TXPStyleCustomizePopup.GetDefaultColorMapClass

7770
XPActnCtrls.TXPStyleCustomizePopup.NCPaint 7770
XPActnCtrls.TXPStyleDropDownBtn 7770
XPActnCtrls.TXPStyleDropDownBtn.DrawBackground 7771
XPActnCtrls.TXPStyleDropDownBtn.DrawFrame 7772
XPActnCtrls.TXPStyleDropDownBtn.DrawGlyph 7772
XPActnCtrls.TXPStyleDropDownBtn.GetPopupClass 7772
XPActnCtrls.TXPStyleExpandBtn 7772
XPActnCtrls.TXPStyleExpandBtn.DrawBackground 7773
XPActnCtrls.TXPStyleExpandBtn.DrawFrame 7773
XPActnCtrls.TXPStyleMenuButton 7774
XPActnCtrls.TXPStyleMenuButton.DrawBackground 7774
XPActnCtrls.TXPStyleMenuButton.DrawText 7775
XPActnCtrls.TXPStyleMenuItem 7775
XPActnCtrls.TXPStyleMenuItem.CalcBounds 7776
XPActnCtrls.TXPStyleMenuItem.CalcLayout 7776
XPActnCtrls.TXPStyleMenuItem.DrawBackground 7777
XPActnCtrls.TXPStyleMenuItem.DrawGlyph 7777
XPActnCtrls.TXPStyleMenuItem.DrawSeparator 7777
XPActnCtrls.TXPStyleMenuItem.DrawShadowedText 7777
XPActnCtrls.TXPStyleMenuItem.DrawSubMenuGlyph 7778
XPActnCtrls.TXPStyleMenuItem.DrawText 7778
XPActnCtrls.TXPStyleMenuItem.DrawUnusedEdges 7778
XPActnCtrls.TXPStylePopupMenu 7778
XPActnCtrls.TXPStylePopupMenu.Create 7779
XPActnCtrls.TXPStylePopupMenu.Destroy 7779
XPActnCtrls.TXPStylePopupMenu.DisplayShadow 7780
XPActnCtrls.TXPStylePopupMenu.GetExpandBtnClass 7780
XPActnCtrls.TXPStylePopupMenu.HideShadow 7781
XPActnCtrls.TXPStylePopupMenu.NCPaint 7781
XPActnCtrls.TXPStylePopupMenu.PositionPopup 7781
XPActnCtrls.TXPStylePopupMenu.VisibleChanging 7781
XPActnCtrls.TXPStyleToolScrollBtn 7782
XPActnCtrls.TXPStyleToolScrollBtn.DrawBackground 7782
xpEncoding enumeration member 7497
XPMan 7785
XPMan namespace 7785
Classes 7786
XPMan.TXPManifest 7786
xpStandalone enumeration member 7497
XPStyleActnCtrls 7787
XPStyleActnCtrls namespace 7787
Classes 7788
Variables 7791
XPStyleActnCtrls.TXPStyleActionBars 7788
XPStyleActnCtrls.TXPStyleActionBars.GetAddRemoveItemClas
s
7789
XPStyleActnCtrls.TXPStyleActionBars.GetColorMapClass 7789
XPStyleActnCtrls.TXPStyleActionBars.GetControlClass 7789
XPStyleActnCtrls.TXPStyleActionBars.GetPopupClass 7790
XPStyleActnCtrls.TXPStyleActionBars.GetScrollBtnClass 7790
XPStyleActnCtrls.TXPStyleActionBars.GetStyleName 7790
XPStyleActnCtrls.XPStyle 7791
XPStyleActnCtrls.XPStyle variable 7791
xpVersion enumeration member 7497
XSBuiltIns 7793
XSBuiltIns namespace 7793
Classes 7795
Constants 7866
Functions 7861
Types 7865
XSBuiltIns.DateTimeToXMLTime 7861
XSBuiltIns.DateTimeToXMLTime function 7861
XSBuiltIns.DateTimeToXSDateTime 7861
XSBuiltIns.DateTimeToXSDateTime function 7861
XSBuiltIns.EXSDateTimeException 7795
XSBuiltIns.EXSDecimalException 7796
XSBuiltIns.EXSHexBinaryException 7796
XSBuiltIns.GetDataFromFile 7862
XSBuiltIns.GetDataFromFile function 7862
XSBuiltIns.IncrementAmount 7866
XSBuiltIns.IncrementAmount constant 7866
XSBuiltIns.InitXSTypes 7862
XSBuiltIns.InitXSTypes function 7862
XSBuiltIns.MaxDay 7867
XSBuiltIns.MaxDay constant 7867
XSBuiltIns.MaxMonth 7867
XSBuiltIns.MaxMonth constant 7867

XSBuiltIns.MinDay 7867
XSBuiltIns.MinDay constant 7867
XSBuiltIns.MinMonth 7868
XSBuiltIns.MinMonth constant 7868
XSBuiltIns.SDecimal 7868
XSBuiltIns.SDecimal constant 7868
XSBuiltIns.SHexMarker 7868
XSBuiltIns.SHexMarker constant 7868
XSBuiltIns.SLocalTimeMarker 7869
XSBuiltIns.SLocalTimeMarker constant 7869
XSBuiltIns.SNAN 7869
XSBuiltIns.SNAN constant 7869
XSBuiltIns.SNegative 7869
XSBuiltIns.SNegative constant 7869
XSBuiltIns.SoapDecimalSeparator 7870
XSBuiltIns.SoapDecimalSeparator constant 7870
XSBuiltIns.SoapFloatToStr 7863
XSBuiltIns.SoapFloatToStr function 7863
XSBuiltIns.SoapStrToFloat 7863
XSBuiltIns.SoapStrToFloat function 7863
XSBuiltIns.SoapTimePrefix 7870
XSBuiltIns.SoapTimePrefix constant 7870
XSBuiltIns.SPlus 7870
XSBuiltIns.SPlus constant 7870
XSBuiltIns.SSciNotationMarker 7871
XSBuiltIns.SSciNotationMarker constant 7871
XSBuiltIns.TXMLData 7797
XSBuiltIns.TXMLData.Create 7798
XSBuiltIns.TXMLData.Destroy 7798
XSBuiltIns.TXMLData.LoadFromXML 7799
XSBuiltIns.TXMLData.ObjectToSOAP 7799
XSBuiltIns.TXMLData.SOAPToObject 7800
XSBuiltIns.TXMLData.XMLNode 7801
XSBuiltIns.TXSBoolean 7801
XSBuiltIns.TXSBoolean.AsBoolean 7803
XSBuiltIns.TXSBoolean.NativeToXS 7802
XSBuiltIns.TXSBoolean.XSToNative 7802
XSBuiltIns.TXSCustomDateTime 7803
XSBuiltIns.TXSCustomDateTime.AsDateTime 7815
XSBuiltIns.TXSCustomDateTime.AsUTCDateTime 7816
XSBuiltIns.TXSCustomDateTime.Create 7805
XSBuiltIns.TXSCustomDateTime.Day 7816
XSBuiltIns.TXSCustomDateTime.Destroy 7805
XSBuiltIns.TXSCustomDateTime.FDateTime 7806
XSBuiltIns.TXSCustomDateTime.FractionalSeconds 7817
XSBuiltIns.TXSCustomDateTime.GetAsDateTime 7807
XSBuiltIns.TXSCustomDateTime.GetAsUTCDateTime 7807
XSBuiltIns.TXSCustomDateTime.GetDay 7807
XSBuiltIns.TXSCustomDateTime.GetFractionalSeconds 7808
XSBuiltIns.TXSCustomDateTime.GetHour 7808
XSBuiltIns.TXSCustomDateTime.GetHourOffset 7808
XSBuiltIns.TXSCustomDateTime.GetMillisecond 7808
XSBuiltIns.TXSCustomDateTime.GetMinute 7809
XSBuiltIns.TXSCustomDateTime.GetMinuteOffset 7809
XSBuiltIns.TXSCustomDateTime.GetMonth 7809
XSBuiltIns.TXSCustomDateTime.GetSecond 7810
XSBuiltIns.TXSCustomDateTime.GetUseZeroMilliseconds 7810
XSBuiltIns.TXSCustomDateTime.GetYear 7810
XSBuiltIns.TXSCustomDateTime.Hour 7817
XSBuiltIns.TXSCustomDateTime.HourOffset 7818
XSBuiltIns.TXSCustomDateTime.Millisecond 7818
XSBuiltIns.TXSCustomDateTime.Minute 7819
XSBuiltIns.TXSCustomDateTime.MinuteOffset 7819
XSBuiltIns.TXSCustomDateTime.Month 7820
XSBuiltIns.TXSCustomDateTime.Second 7820
XSBuiltIns.TXSCustomDateTime.SetAsDateTime 7811
XSBuiltIns.TXSCustomDateTime.SetAsUTCDateTime 7811
XSBuiltIns.TXSCustomDateTime.SetDay 7811
XSBuiltIns.TXSCustomDateTime.SetFractionalSeconds 7811
XSBuiltIns.TXSCustomDateTime.SetHour 7812
XSBuiltIns.TXSCustomDateTime.SetHourOffset 7812
XSBuiltIns.TXSCustomDateTime.SetMillisecond 7812
XSBuiltIns.TXSCustomDateTime.SetMinute 7813
XSBuiltIns.TXSCustomDateTime.SetMinuteOffset 7813
XSBuiltIns.TXSCustomDateTime.SetMonth 7813
XSBuiltIns.TXSCustomDateTime.SetSecond 7814
XSBuiltIns.TXSCustomDateTime.SetUseZeroMilliseconds 7814
XSBuiltIns.TXSCustomDateTime.SetYear 7814

XSBuiltIns.TXSCustomDateTime.UseZeroMilliseconds 7821
XSBuiltIns.TXSCustomDateTime.Year 7821
XSBuiltIns.TXSDate 7822
XSBuiltIns.TXSDate.AsDate 7827
XSBuiltIns.TXSDate.Clone 7824
XSBuiltIns.TXSDate.Create 7823
XSBuiltIns.TXSDate.Day 7827
XSBuiltIns.TXSDate.GetAsDate 7824
XSBuiltIns.TXSDate.GetDay 7824
XSBuiltIns.TXSDate.GetMonth 7824
XSBuiltIns.TXSDate.GetYear 7825
XSBuiltIns.TXSDate.Month 7827
XSBuiltIns.TXSDate.NativeToXS 7825
XSBuiltIns.TXSDate.SetAsDate 7825
XSBuiltIns.TXSDate.SetDay 7825
XSBuiltIns.TXSDate.SetMonth 7826
XSBuiltIns.TXSDate.SetYear 7826
XSBuiltIns.TXSDate.XSToNative 7826
XSBuiltIns.TXSDate.Year 7828
XSBuiltIns.TXSDateTime 7828
XSBuiltIns.TXSDateTime.Clone 7830
XSBuiltIns.TXSDateTime.CompareDateTimeParam 7831
XSBuiltIns.TXSDateTime.Millisecond 7832
XSBuiltIns.TXSDateTime.NativeToXS 7831
XSBuiltIns.TXSDateTime.XSToNative 7831
XSBuiltIns.TXSDecimal 7832
XSBuiltIns.TXSDecimal.AsBcd 7835
XSBuiltIns.TXSDecimal.DecimalString 7835
XSBuiltIns.TXSDecimal.NativeToXS 7834
XSBuiltIns.TXSDecimal.XSToNative 7834
XSBuiltIns.TXSDuration 7836
XSBuiltIns.TXSDuration.Create 7838
XSBuiltIns.TXSDuration.DecimalSecond 7839
XSBuiltIns.TXSDuration.GetDecimalSecond 7838
XSBuiltIns.TXSDuration.NativeToXS 7838
XSBuiltIns.TXSDuration.SetDecimalSecond 7839
XSBuiltIns.TXSDuration.XSToNative 7839
XSBuiltIns.TXSHexBinary 7840
XSBuiltIns.TXSHexBinary.AsByteArray 7842
XSBuiltIns.TXSHexBinary.HexBinaryString 7843
XSBuiltIns.TXSHexBinary.NativeToXS 7841
XSBuiltIns.TXSHexBinary.XSToNative 7842
XSBuiltIns.TXSInteger 7843
XSBuiltIns.TXSInteger.AsInteger 7845
XSBuiltIns.TXSInteger.NativeToXS 7844
XSBuiltIns.TXSInteger.XSToNative 7844
XSBuiltIns.TXSLong 7845
XSBuiltIns.TXSLong.AsLong 7847
XSBuiltIns.TXSLong.NativeToXS 7846
XSBuiltIns.TXSLong.XSToNative 7846
XSBuiltIns.TXSString 7847
XSBuiltIns.TXSString.NativeToXS 7848
XSBuiltIns.TXSString.XSToNative 7848
XSBuiltIns.TXSTime 7849
XSBuiltIns.TXSTime.AsTime 7856
XSBuiltIns.TXSTime.Clone 7851
XSBuiltIns.TXSTime.FractionalSeconds 7857
XSBuiltIns.TXSTime.FractionalSecondString 7857
XSBuiltIns.TXSTime.GetAsTime 7851
XSBuiltIns.TXSTime.GetFractionalSeconds 7851
XSBuiltIns.TXSTime.GetFractionalSecondString 7852
XSBuiltIns.TXSTime.GetHour 7852
XSBuiltIns.TXSTime.GetHourOffset 7852
XSBuiltIns.TXSTime.GetMillisecond 7852
XSBuiltIns.TXSTime.GetMinute 7853
XSBuiltIns.TXSTime.GetMinuteOffset 7853
XSBuiltIns.TXSTime.GetSecond 7853
XSBuiltIns.TXSTime.Hour 7857
XSBuiltIns.TXSTime.HourOffset 7858
XSBuiltIns.TXSTime.Millisecond 7858
XSBuiltIns.TXSTime.Minute 7859
XSBuiltIns.TXSTime.MinuteOffset 7859
XSBuiltIns.TXSTime.NativeToXS 7853
XSBuiltIns.TXSTime.Second 7859
XSBuiltIns.TXSTime.SetAsTime 7854
XSBuiltIns.TXSTime.SetFractionalSeconds 7854
XSBuiltIns.TXSTime.SetHour 7854
XSBuiltIns.TXSTime.SetHourOffset 7854

XSBuiltIns.TXSTime.SetMillisecond 7854
XSBuiltIns.TXSTime.SetMinute 7855
XSBuiltIns.TXSTime.SetMinuteOffset 7855
XSBuiltIns.TXSTime.SetSecond 7855
XSBuiltIns.TXSTime.UseZeroMilliseconds 7860
XSBuiltIns.TXSTime.XSToNative 7855
XSBuiltIns.TXSTimeInstant 7865
XSBuiltIns.TXSTimeInstant type 7865
XSBuiltIns.XMLDateSeparator 7871
XSBuiltIns.XMLDateSeparator constant 7871
XSBuiltIns.XMLDayPos 7871
XSBuiltIns.XMLDayPos constant 7871
XSBuiltIns.XMLDefaultYearDigits 7872
XSBuiltIns.XMLDefaultYearDigits constant 7872
XSBuiltIns.XMLDurationDay 7872
XSBuiltIns.XMLDurationDay constant 7872
XSBuiltIns.XMLDurationHour 7872
XSBuiltIns.XMLDurationHour constant 7872
XSBuiltIns.XMLDurationMinute 7873
XSBuiltIns.XMLDurationMinute constant 7873
XSBuiltIns.XMLDurationMonth 7873
XSBuiltIns.XMLDurationMonth constant 7873
XSBuiltIns.XMLDurationSecond 7873
XSBuiltIns.XMLDurationSecond constant 7873
XSBuiltIns.XMLDurationStart 7874
XSBuiltIns.XMLDurationStart constant 7874
XSBuiltIns.XMLDurationYear 7874
XSBuiltIns.XMLDurationYear constant 7874
XSBuiltIns.XMLHourOffsetMinusMarker 7874
XSBuiltIns.XMLHourOffsetMinusMarker constant 7874
XSBuiltIns.XMLHourOffsetPlusMarker 7875
XSBuiltIns.XMLHourOffsetPlusMarker constant 7875
XSBuiltIns.XMLMilSecPos 7875
XSBuiltIns.XMLMilSecPos constant 7875
XSBuiltIns.XMLMonthPos 7875
XSBuiltIns.XMLMonthPos constant 7875
XSBuiltIns.XMLTimeSeparator 7876
XSBuiltIns.XMLTimeSeparator constant 7876
XSBuiltIns.XMLTimeToDateTime 7863
XSBuiltIns.XMLTimeToDateTime function 7863
XSBuiltIns.XMLYearPos 7876
XSBuiltIns.XMLYearPos constant 7876
XSLProd 7877
XSLProd namespace 7877
Classes 7878
XSLProd.EXSLPageProducerError 7878
XSLProd.TCustomXSLPageProducer 7878
XSLProd.TCustomXSLPageProducer.Content 7885
XSLProd.TCustomXSLPageProducer.ContentFromStream 7885
XSLProd.TCustomXSLPageProducer.ContentFromString 7886
XSLProd.TCustomXSLPageProducer.ContentFromWideString 7886
XSLProd.TCustomXSLPageProducer.Dispatcher 7891
XSLProd.TCustomXSLPageProducer.DispatcherComponent 7892
XSLProd.TCustomXSLPageProducer.GetAppDispatcher 7887
XSLProd.TCustomXSLPageProducer.GetProducerTemplateFileName 7887
XSLProd.TCustomXSLPageProducer.GetProducerTemplateStream 7887
XSLProd.TCustomXSLPageProducer.GetTemplateFileType 7887
XSLProd.TCustomXSLPageProducer.GetTemplateStream 7888
XSLProd.TCustomXSLPageProducer.GetXMLData 7888
XSLProd.TCustomXSLPageProducer.GetXMLDocument 7888
XSLProd.TCustomXSLPageProducer.GetXSLData 7888
XSLProd.TCustomXSLPageProducer.HasHTMLBrowserView 7888
XSLProd.TCustomXSLPageProducer.HasScriptView 7889
XSLProd.TCustomXSLPageProducer.HasXMLBrowserView 7889
XSLProd.TCustomXSLPageProducer.HasXSLBrowserView 7889
XSLProd.TCustomXSLPageProducer.Notification 7889
XSLProd.TCustomXSLPageProducer.ProduceContent 7890
XSLProd.TCustomXSLPageProducer.ProduceContentFromStream 7890
XSLProd.TCustomXSLPageProducer.ProduceContentFromString 7890

XSLProd.TCustomXSLPageProducer.SetAppDispatcher 7891
XSLProd.TCustomXSLPageProducer.XMLData 7892
XSLProd.TXSLPageProducer 7893
XSLProd.TXSLPageProducer.XMLData 7899
XSToNative
TXSBoolean class 7802
TXSDate class 7826
TXSDateTime class 7831
TXSDecimal class 7834
TXSDuration class 7839
TXSHexBinary class 7842
TXSInteger class 7844
TXSLong class 7846
TXSString class 7848
TXSTime class 7855

Y

Y
TDisplayOffset class 325
Year
TXSCustomDateTime class 7821
TXSDate class 7828

Z

zcDefault enumeration member 7927
zcFastest enumeration member 7927
zcMax enumeration member 7927
zcNone enumeration member 7927
ZLib 7901
ZLib namespace 7901
Classes 7904
Constants 7932
Functions 7917
Structs, Records, Enums 7926
Types 7929
ZLib._z_errmsg 7932
ZLib._z_errmsg constant 7932
ZLib.adler32 7917
ZLib.adler32 function 7917
ZLib.deflate 7918
ZLib.deflate function 7918
ZLib.deflateEnd 7918
ZLib.deflateEnd function 7918
ZLib.deflateInit_ 7918
ZLib.deflateInit_ function 7918
ZLib.DeflateInit2_ 7919
ZLib.DeflateInit2_ function 7919
ZLib.EZCompressionError 7904
ZLib.EZDecompressionError 7904
ZLib.EZLibError 7905
ZLib.inflate 7919
ZLib.inflate function 7919
ZLib.inflateEnd 7919
ZLib.inflateEnd function 7919
ZLib.inflateInit_ 7920
ZLib.inflateInit_ function 7920
ZLib.inflateInit2_ 7920
ZLib.inflateInit2_ function 7920
ZLib.inflateReset 7920
ZLib.inflateReset function 7920
ZLib.MoveI32 7921
ZLib.MoveI32 function 7921
ZLib.SZInvalid 7933
ZLib.SZInvalid constant 7933
ZLib.TCompressionLevel 7926
ZLib.TCompressionLevel enumeration 7926
ZLib.TCompressionStream 7929
ZLib.TCompressionStream type 7929
ZLib.TCustomZLibStream 7929
ZLib.TCustomZLibStream type 7929
ZLib.TCustomZStream 7905
ZLib.TCustomZStream.Create 7906
ZLib.TCustomZStream.DoProgress 7906
ZLib.TCustomZStream.OnProgress 7907
ZLib.TDecompressionStream 7930
ZLib.TDecompressionStream type 7930
ZLib.TZAlloc 7930
ZLib.TZAlloc type 7930

ZLib.TZCompressionLevel 7927
ZLib.TZCompressionLevel enumeration 7927
ZLib.TZCompressionStream 7907
ZLib.TZCompressionStream.CompressionRate 7911
ZLib.TZCompressionStream.Create 7909
ZLib.TZCompressionStream.Destroy 7909
ZLib.TZCompressionStream.OnProgress 7912
ZLib.TZCompressionStream.Read 7910
ZLib.TZCompressionStream.Seek 7910
ZLib.TZCompressionStream.Write 7910
ZLib.TZDecompressionStream 7912
ZLib.TZDecompressionStream.Create 7913
ZLib.TZDecompressionStream.Destroy 7913
ZLib.TZDecompressionStream.OnProgress 7915
ZLib.TZDecompressionStream.Read 7914
ZLib.TZDecompressionStream.Seek 7914
ZLib.TZDecompressionStream.Write 7915
ZLib.TZFree 7930
ZLib.TZFree type 7930
ZLib.TZStreamRec 7927
ZLib.TZStreamRec record 7927
ZLib.Z_ASCII 7933
ZLib.Z_ASCII constant 7933
ZLib.Z_BEST_COMPRESSION 7933
ZLib.Z_BEST_COMPRESSION constant 7933
ZLib.Z_BEST_SPEED 7934
ZLib.Z_BEST_SPEED constant 7934
ZLib.Z_BINARY 7934
ZLib.Z_BINARY constant 7934
ZLib.Z_BUF_ERROR 7934
ZLib.Z_BUF_ERROR constant 7934
ZLib.Z_DATA_ERROR 7935
ZLib.Z_DATA_ERROR constant 7935
ZLib.Z_DEFAULT_COMPRESSION 7935
ZLib.Z_DEFAULT_COMPRESSION constant 7935
ZLib.Z_DEFAULT_STRATEGY 7935
ZLib.Z_DEFAULT_STRATEGY constant 7935
ZLib.Z_DEFLATED 7936
ZLib.Z_DEFLATED constant 7936
ZLib.Z_ERRNO 7936
ZLib.Z_ERRNO constant 7936
ZLib.Z_FILTERED 7936
ZLib.Z_FILTERED constant 7936
ZLib.Z_FINISH 7937
ZLib.Z_FINISH constant 7937
ZLib.Z_FULL_FLUSH 7937
ZLib.Z_FULL_FLUSH constant 7937
ZLib.Z_HUFFMAN_ONLY 7937
ZLib.Z_HUFFMAN_ONLY constant 7937
ZLib.Z_MEM_ERROR 7938
ZLib.Z_MEM_ERROR constant 7938
ZLib.Z_NEED_DICT 7938
ZLib.Z_NEED_DICT constant 7938
ZLib.Z_NO_COMPRESSION 7938
ZLib.Z_NO_COMPRESSION constant 7938
ZLib.Z_NO_FLUSH 7939
ZLib.Z_NO_FLUSH constant 7939
ZLib.Z_OK 7939
ZLib.Z_OK constant 7939
ZLib.Z_PARTIAL_FLUSH 7939
ZLib.Z_PARTIAL_FLUSH constant 7939
ZLib.Z_STREAM_END 7940
ZLib.Z_STREAM_END constant 7940
ZLib.Z_STREAM_ERROR 7940
ZLib.Z_STREAM_ERROR constant 7940
ZLib.Z_SYNC_FLUSH 7940
ZLib.Z_SYNC_FLUSH constant 7940
ZLib.Z_UNKNOWN 7941
ZLib.Z_UNKNOWN constant 7941
ZLib.Z_VERSION_ERROR 7941
ZLib.Z_VERSION_ERROR constant 7941
ZLib.ZCompress 7921
ZLib.ZCompress function 7921
ZLib.ZCompressStr 7922
ZLib.ZCompressStr function 7922
ZLib.ZCompressStream 7922, 7923
ZLib.ZCompressStream function 7922, 7923
ZLib.ZDecompress 7923

ZLib.ZDecompress function 7923
ZLib.ZDecompressStr 7923
ZLib.ZDecompressStr function 7923
ZLib.ZDecompressStream 7924
ZLib.ZDecompressStream function 7924
ZLib.ZLevels 7941
ZLib.ZLevels constant 7941
ZLib.ZLIB_VERSION 7942
ZLib.ZLIB_VERSION constant 7942
ZLib.zlibAllocMem 7924
ZLib.zlibAllocMem function 7924
ZLib.zlibFreeMem 7924
ZLib.zlibFreeMem function 7924
ZLib.ZSendToBrowser 7925
ZLib.ZSendToBrowser function 7925